UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUCAS LEANDRO NESI

Strategies for Distributing Task-Based Applications on Heterogeneous Platforms

Thesis prepared under a *co-tutelle* agreement and presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science at the Federal University of Rio Grande do Sul and the University Grenoble Alpes

Advisor (UFRGS): Prof. Dr. Lucas Mello Schnorr
Advisor (UGA): Dr. Arnaud Legrand

Porto Alegre
September 2023

CIP — CATALOGING-IN-PUBLICATION

Leandro Nesi, Lucas

Strategies for Distributing Task-Based Applications on Heterogeneous Platforms / Lucas Leandro Nesi. – Porto Alegre: PPGC da UFRGS, 2023.

240 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Advisor (UFRGS): Lucas Mello Schnorr; Advisor (UGA): Arnaud Legrand.

1. HPC, Heterogeneity, Task-Based, Distribution, Partitioning, Multi-Phase. I. Mello Schnorr, Lucas. II. Legrand, Arnaud. III. Título.

“All we have to decide is what to do

with the time that is given us”

— J. R. R. Tolkien, The Fellowship of the Ring

ACKNOWLEDGEMENTS

To my advisors, Lucas Mello Schnorr and Arnaud Legrand, who accepted and guided me during this journey. I am very grateful for all the opportunities, time, and motivation you gave me. All the advice, tips, and enthusiastic discussions were essential to realizing this work. I could never have wished for better advisors than you two, to whom I am infinitely grateful. I am certain that you two will continue to inspire your future students as you inspired me.

To the jury members for all the time dedicated to the review of this work.

To the research funding agencies: CNPq (Conselho de Desenvolvimento Científico e Tecnológico), scholarship grant number 141971/2020-7, and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) - Finance Code 001, for grating the scholarships, and permitting that I could conduct this work. And the projects: FAPERGS (Data Science – 19/711-6), Inria (Associated Team ReDas), and CAPES (Cofecub 04/2017).

To the three main computational environments used in this thesis. Grid5000, SDumont supercomputer, and PCAD. Some experiments were carried out using Grid'5000, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several Universities as well as other organizations (see <https://www.grid5000.fr>). Some experiments in this work used the PCAD infrastructure, <http://gppd-hpc.inf.ufrgs.br>, at INF/UFRGS. This work acknowledges the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this thesis (<http://sdumont.lncc.br>).

To the authors, contributors, and the community of the following libraries and applications: StarPU, Chameleon, Simgrid, Diodon, ExaGeoStat, R, and Tidyverse.

To all my friends in GPPD, Porto Alegre, LIG, and Grenoble. Thanks for all the in-lab time, and especially the out-lab one. Some ideas and inspiration came from such moments.

Para minha amada Gabriela Stahl, que me apoiou durante toda a realização deste trabalho. A tua motivação, paciência e companheirismo foram fundamentais para que eu pudesse alcançar este objetivo. Obrigado por todos os maravilhosos momentos que tivemos e que teremos.

E finalmente, para meus pais, Eliane e Lirio, que sempre foram modelos de pessoas para mim e que jamais mediram esforços para me ajudar nesta jornada.

Strategies for Distributing Task-Based Applications on Heterogeneous Platforms

ABSTRACT

HPC platforms are vastly heterogeneous because of intra-node resources like accelerators and inter-node heterogeneity when there are different machines. The applications that use these resources are already very complex, with many distinct operations and phases, and developers must consider all sets of diverse computational resources. The task-based programming paradigm is a modern alternative to increase the computational efficiency of intra-node heterogeneous resources while maintaining relative development simplicity. The application defines a Directed Acyclic Graph of tasks and a dynamic runtime asynchronously schedules them to the resources respecting task dependencies. However, handling different types of nodes requires new specific strategies to distribute an application in this asynchronous and heterogeneous environment. This thesis studies the problem of distributing this type of complex task-based applications over those diverse system-level resources, proposing strategies to divide their load correctly, considering computational heterogeneity, multiple-phase asynchronism, and adaptability. This work uses real applications to validate its results with experiments conducted in large testbeds and a supercomputer. The thesis' main contributions are the following. (i) Strategies for distributing a single application operation considering the trade-off of communication, critical path, and heterogeneous load balancing. (ii) A set of optimizations for improving asynchronous phase overlap in applications. (iii) A methodology for computing the relative power of each phase on each heterogeneous group of nodes considering the phase overlap. (iv) A distribution strategy for an antecedent phase reducing communication redistribution. (v) A strategy for the application dynamically adapts during execution to decide the best subset of nodes for each phase. (vi) An extended comprehensive analysis of the experiments that include a methodology to analyze the application progress per node resilient to heterogeneity and that can cluster nodes with similar behavior. Ultimately, this thesis is a step toward efficiently exploiting and combining any of these diverse resources, using them to handle applications' distinct necessities better, and improving their overall performance.

Keywords: HPC, Heterogeneity, Task-Based, Distribution, Partitioning, Multi-Phase.

Estratégias para a Distribuição de Aplicações Baseadas em Tarefas em Plataformas Heterogêneas

RESUMO

As plataformas de HPC são heterogêneas devido aos recursos intra-nó, como os aceleradores, e a heterogeneidade inter-nó, quando existem máquinas diferentes. As aplicações que utilizam estes recursos já são complexas, com muitas operações e fases distintas, e os programadores podem considerar todos os conjuntos de recursos computacionais diversos. O paradigma de programação baseado em tarefas é uma alternativa moderna para aumentar a eficiência computacional dos recursos heterogêneos intra-nó, mantendo uma relativa simplicidade de desenvolvimento. A aplicação define um grafo acíclico dirigido de tarefas e um *runtime* os escalona de forma assíncrona para os recursos, respeitando as dependências das tarefas. No entanto, o tratamento de diferentes tipos de nós requer novas estratégias específicas para distribuir uma aplicação neste ambiente assíncrono e heterogêneo. Esta tese estuda o problema da distribuição deste tipo de aplicações complexas baseadas em tarefas nesses recursos heterogêneos ao nível do sistema, propondo estratégias para dividir corretamente a sua carga, considerando a heterogeneidade computacional, o assincronismo em múltiplas fases e a adaptabilidade. Este trabalho utiliza aplicações reais para validar os resultados com experimentos realizados em grandes *testbeds* e num supercomputador. As principais contribuições da tese são as seguintes. (i) Estratégias para distribuir uma única operação considerando a razão de comunicação, caminho crítico e balanceamento de carga heterogêneo. (ii) Um conjunto de optimizações para melhorar a sobreposição de fases assíncronas em aplicações. (iii) Uma metodologia para calcular a velocidade relativa de cada fase em cada grupo heterogêneo de nós, tendo em conta a sobreposição de fases. (iv) Uma estratégia de distribuição para uma fase antecedente que reduza a redistribuição de comunicações. (v) Uma estratégia para a aplicação se adaptar dinamicamente durante a execução para decidir o melhor subconjunto de nós para cada fase. (vi) Uma análise abrangente e aprofundada dos experimentos que inclui uma metodologia para analisar o progresso da aplicação por nó considerando a heterogeneidade e que pode agrupar nós com comportamento semelhante. Finalmente, esta tese é um passo no sentido de explorar e combinar eficientemente qualquer um destes diversos recursos, utilizando-os para melhor acomodar as necessidades distintas das aplicações e melhorar o seu desempenho.

Palavras-chave: PAD, Heterogeneidade, Tarefas, Distribuição, Particionamento, Multi-Fase.

Stratégies de distribution d'applications basées sur des tâches sur des plates-formes hétérogènes

RÉSUMÉ

Les plates-formes HPC sont amplement hétérogènes en raison des ressources intra-nœuds, comme les accélérateurs, et de l'hétérogénéité inter-nœuds lorsqu'il y a différentes machines. Les applications qui utilisent ces ressources sont déjà très complexes, avec de nombreuses opérations et phases distinctes, et les développeurs doivent prendre en compte tous les diverses ressources informatiques. Le paradigme de la programmation basée sur les tâches est une alternative moderne pour augmenter l'efficacité de calcul des ressources hétérogènes intra-nœud tout en maintenant une relative simplicité de développement. L'application définit un graphe acyclique direct de tâches et un moteur d'exécution dynamique les planifie de manière asynchrone sur les ressources en respectant les dépendances des tâches. Cependant, la gestion de différents types de nœuds nécessite de nouvelles stratégies spécifiques pour distribuer une application dans cet environnement asynchrone et hétérogène. Cette thèse étudie le problème de la distribution de ce type d'applications complexes basées sur des tâches sur ces diverses ressources au niveau du système, en proposant des stratégies pour diviser leur charge correctement, en tenant compte de l'hétérogénéité de calcul, de l'asynchronisme à phases multiples et de l'adaptabilité. Ce travail utilise des applications réelles pour valider ses résultats avec des expériences menées dans de grands *testbeds* et un superordinateur. Les principales contributions de la thèse sont les suivantes. (i) Stratégies de distribution d'une opération d'application unique en tenant compte du compromis entre la communication, le chemin critique et l'équilibrage de la charge hétérogène. (ii) Un ensemble d'optimisations pour améliorer le chevauchement des phases asynchrones dans les applications. (iii) Une méthodologie pour calculer la puissance relative de chaque phase sur chaque groupe hétérogène de nœuds en tenant compte du chevauchement des phases. (iv) Une stratégie de distribution pour une phase antérieure réduisant des communications. (v) Une stratégie d'adaptation dynamique de l'application pendant l'exécution pour décider du meilleur sous-ensemble de nœuds pour chaque phase. (vi) Une analyse des expériences qui inclut une méthodologie pour analyser la progression de l'application par nœud qui résiste à l'hétérogénéité et qui peut regrouper les nœuds ayant un comportement similaire. En fin, cette thèse constitue une étape vers l'exploitation et la combinaison efficaces de ces ressources, pour mieux gérer les besoins distincts des applications et améliorer leurs performances.

Palavras-chave: HPC, Hétérogénéité, Tâches, Distribution, Partitionnement, Multiphase.

LIST OF ABBREVIATIONS AND ACRONYMS

ABE	Area-Bound Estimator
AMT	Asynchronous Many Task
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BC	Block-Cyclic
BIC	Bayesian Information Criterion
BLAS	Basic Linear Algebra Subprograms
BLR	Block Low Rank
BSP	Bulk Synchronous Parallel
CFD	Computational Fluid Dynamics
CPB	Critical Path Bound
CoA	Correspondence Analysis
DAG	Directed Acyclic Graph
DC	Divide and Conquer
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
G5K	Grid5000 infrastructure (Inria/France)
GMM	Gaussian Mixture Models
GP	Gaussian Process
GPGPU	General Purpose Graphics Processing Units
HPC	High-Performance Computing
HPL	High-Performance Linpack
LP	Linear Program
LU	LU (Lower Upper) Factorization

MDS Multidimensional Scaling

ML Machine learning

MPI Message Passing Interface

MTU Maximum Transmission Unit

MVN Multivariate Normal Distribution

MPI Message Passing Interface

MSA Modular Supercomputer Architecture

NIC Network Interface Controller

NTP Network Time Protocol

NUMA Non-Uniform Memory Access

PCA Principal Component Analysis

PCAD PCAD infrastructure (UFRGS/Brazil)

PCAM Partitioning, Communication, Agglomeration, Mapping

PIM Processing In Memory

PTG Parameterized Task Graphs

PTP Precision Time Protocol

RAPL Running Average Power Limit

RL Reinforced Learning

rSVD Randomized Singular Value Decomposition

SD Santos Dumont supercomputer (LNCC/Brazil)

SIMD Single Instruction, Multiple Data

STF Sequential Task Flow

SVD Singular Value Decomposition

TPU Tensor Processing Unit

UCB Upper Confidence Bound

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

High-Performance Computing (HPC) provides the means for complex applications to use massive computational power and conclude in a feasible time. Until the beginning of the 2000 decade, the growth of transistors per chip followed the Moore law, which stated it would double each two years (MOORE et al., 1965; MOORE et al., 1975). This growth allowed an exponential increase in computational power, as pointed out by Dennard performance scaling (DENNARD et al., 1974). However, limits in heat dissipation held the increase of chips' clock in the 2000s (HENNESSY; PATTERSON, 2017), stopping the exponential performance increase of individual cores (DONGARRA et al., 2017).

Many alternatives for continuing the computational power growth appeared. Multicore microprocessors allow single chips to possess multiple cores (HENNESSY; PATTERSON, 2017). Manycore systems enable many CPUs on the same board while sharing the same memory space using the Non-Uniform Memory Access (NUMA) (HENNESSY; PATTERSON, 2017). And other processing hardware emerged besides CPUs for specific purposes, such as accelerators like General Purpose Graphics Processing Units (GPGPU)[1] (SANDERS; KANDROT, 2010), other vectorial accelerators (KOMATSU et al., 2018), Field-Programmable Gate Array (FPGA) (UNDERWOOD; HEMMERT; ULMER, 2009), and Application-Specific Integrated Circuit (ASIC), like Tensor Processing Units (TPU) (JOUPPI et al., 2017).

The combination of a subset of these resources results in a single heterogeneous computational node. This intra-node heterogeneity has been widely employed in recent years in many applications by the HPC community (SANDERS; KANDROT, 2010; AUGONNET et al., 2011; MENG et al., 2017; PINTO, 2018). However, exploiting this heterogeneity is complex, as it requires the correct application division into those multiple and diverse resources, considering the adaptability and performance of the algorithm in each resource (SANDERS; KANDROT, 2010; AUGONNET et al., 2011). For example, computational kernels with many distinct branches (many if-else statements, for example) are more appropriate for CPUs, while algorithms that repeat the same instruction over data (like the matrix-matrix multiplication operation) are well suitable for GPUs.

From the developers' perspective, handling such heterogeneity requires extensive knowledge of the target system and the anticipation of the application behavior. In traditional programming methods and tools such as Message Passing Interface (MPI) and Bulk Synchronous Parallel (BSP) applications, using heterogeneous resources is complicated, as the programming

[1] Hereby only called GPUs, as it is the most common use in literature.

style is entirely imperative. These programming approaches specify statically where a workload executes, with barriers and synchronous communications to control the flow of the application's multiple operations or phases. With this level of synchronism, a small error in the workload division would result in non-ideal performance with idle resources. One problem is that such a mistake is almost inevitable, as systems and applications are complex. Also, ensuring the application's capability (portability) to execute on different systems requires extensive modifications to its source code, increasing the chance of mistakes.

Nevertheless, the availability of distinct processors and the diversity of applications necessities encourage heterogeneity in a system-level (HPE et al., 2022). It is thus increasingly common that HPC systems comprise many machines with diverse hardware configurations organized into homogeneous clusters or partitions to accommodate applications' necessities better. If used correctly, this heterogeneity enables the application to adjust particular internal behavior and improve performance, exploiting heterogeneity instead of suffering from it. The structure of this chapter follows. Section 1.1 details the presence of heterogeneity on HPC platforms, notably system-level heterogeneous resources in many supercomputers' sites. Section 1.2 discusses the application's challenges and perspectives when using heterogeneous resources. Finally, Section 1.3 details the problems tackled by this Thesis, opportunities, and goals.

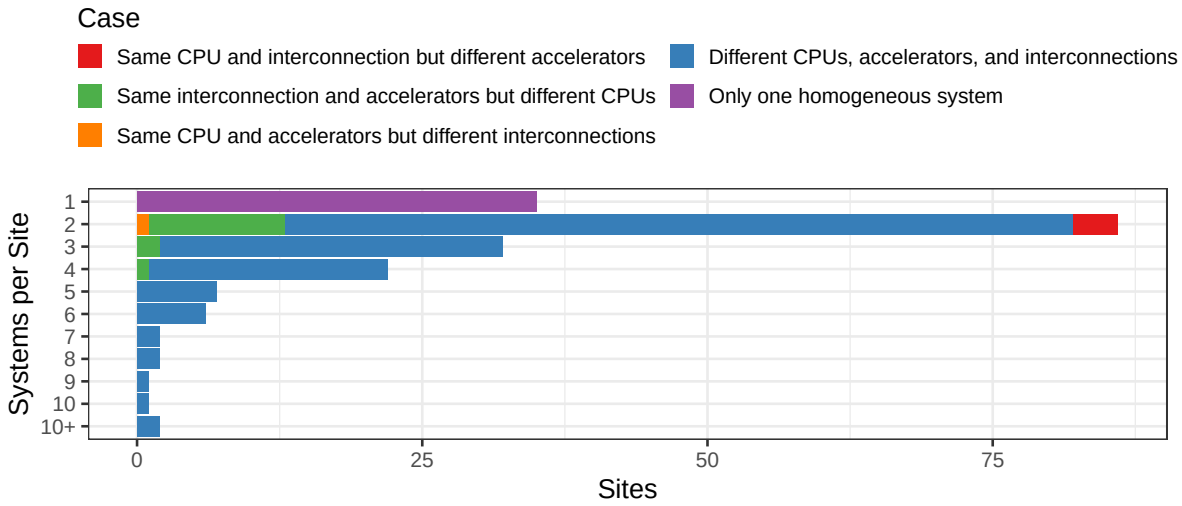
1.1 The Heterogeneous HPC Universe

The heterogeneity in HPC is vast (HPE et al., 2022). There are different processor units, CPUs, GPUs, vector Processors, and very specialized devices such as TPUs, each ideal for specific applications and algorithms. There are even CPUs with specialized and distinct cores available (Intel Corporation, 2021b). However, this heterogeneity is not limited to processing units. The expanded hierarchy with non-volatile units or the studied Processing In Memory (PIM) are examples in the memory field (HPE et al., 2022). Heterogeneity is also present in both storage devices with Solid State Drives (SSD), Hard Disk Drives (HDD), and long-term storage tapes; and in the interconnection layer with dozens of proprietary and custom hardware and links. All those choices lead to a combinatorial explosion of the possible configurations when designing system architectures. Combining different nodes enables a more robust accommodation of different applications with various needs. In this context, the notion of Modular Supercomputer Architecture (MSA) appeared (HPE et al., 2022), with separate modular homogeneous clusters targeting a kind or a particular phase of the HPC applications workflow.

Traditionally, HPC applications require supercomputers, usually located in large research

centers, both in academia and industry. The sites (the infrastructure facilities) can host different systems, as pointed out by the TOP500 (DONGARRA et al., 1997) list of supercomputers. Figure 1.1 shows the number of sites with different numbers of homogeneous systems (partitions, homogeneous clusters) that appeared in the TOP500 list between 2017/11 and 2022/11. For example, there were 86 sites with precisely two distinct homogeneous systems. The colors represent the heterogeneous type of the site. Sites in green have systems that only differ in the CPU, while red differs for GPUs and orange only for the interconnection. Sites in blue have multiple heterogeneous systems (CPU, GPU, and interconnection). For example, from the 86 sites with two different systems, the two systems in four sites (red) have the same CPU and interconnection but different accelerators. In the same case, the two systems of 69 sites (blue) have multiple sources of difference (CPUs, interconnections, and accelerators), 12 sites (green) have both systems with the same interconnection and accelerators but have different CPUs, and only one site (orange) have two systems that have the same CPU and accelerators but have different interconnections. The number of sites with multiple systems is significant, as less than 40 sites have only one system that appeared on the TOP500 list. Also, systems below the 500 positions in the ranking since 2017/11 are absent, meaning that these sites can have less powerful supercomputers that are still operational and could be used in conjunction with the more powerful systems.

Figure 1.1 – Sites of the TOP500 that have multiple systems between 2017/11 and 2022/11



Source: The Author.

The reasons behind the heterogeneity of these supercomputer's sites are three-fold: (i) by design, when the infrastructure possesses such configurations to target diverse workloads; (ii) financial limitations, when, for example, only a subset of nodes receives accelerators because of

budget constraints; and (iii) the natural infrastructure upgrades over time.

The TOP 500 list only considers and ranks a homogeneous set of nodes as a system. This restriction comes from a limitation in the adopted HPC benchmark, HPL (DONGARRA; LUSZCZEK; PETITET, 2003), as it can not run adequately in system-level heterogeneous setups. Table 1.1 presents selected examples of sites with multiple systems. An example is the Santos Dumont Supercomputer[2] installed at the National Laboratory for Scientific Computing (LNCC)[3]. The 2015 Santos Dumont site was split into three major groups, GPU, Hybrid (with Intel KNL), and CPU, describing their accelerators' capabilities. This division is both a case of design (multiple accelerators for different purposes and applications) and probably financial limitations (not all nodes received them). Recently, LNCC enhanced Santos Dumont with an extended partition containing a new processor and accelerator, a natural upgrade over time. Only this last extension partition was measured and present in the 2020/11 TOP 500 list. Santos Dumont is a clear example of multiple homogeneous systems that applications could use together, taking advantage of a more diverse, larger, and powerful system. As a matter of fact, the Santos Dumont position in the TOP 500 list (276º) would probably improve if the benchmark could use the totality of heterogeneous system-level resources, as computational power from different systems would combine.

Other remarkable examples in Table 1.1 are the Chinese PAI-ASystem/PAI-BSystem, the Saudi Arabian Dammam-7/Unizah-II, and the Swiss Piz Daint/Piz Daint Multicore. In all these cases, both systems have identical hardware specifications except for the accelerator. While one of the systems in each site has an accelerator, the other does not. Financial limitations could explain this. Examples of upgrades over time are the Australians Gadi and Raijin, and the Germans COBRA and DRACO. It is possible to observe a considerable difference in the systems' launch years. The American Selene/DGX SuperPOD may be a case of design choice, as both systems launched in 2020 with different hardware (processors and accelerators).

The full utilization of these systems over many years is desirable, as they are costly platforms and usually require significant construction time and effort. In this way, the application's ability to use and adapt to all the system resources is not just a matter of performance but also reduces these expensive systems' idleness. The old systems shutdown should be carefully studied when organizations upgrade their infrastructures. The resources could still be operational, and the inappropriate disposal or premature retirement is also an environmental challenge (WIDMER et al., 2005; ROBINSON, 2009; LYNAR et al., 2010). Simply replacing all machines often for homogeneity is an expedient decision.

[2] <https://sdumont.lncc.br/machine.php?pg=machine>
[3] <https://www.gov.br/mcti/pt-br/rede-mcti/lncc>

Table 1.1 – Supercomputers' sites examples at TOP500 containing heterogeneous systems

Site	System	Processor	Accelerator	Year	TOP 500 2020/11
LNCC (Brazil)	SDumont	Intel Xeon Gold 6252	NVIDIA Tesla V100	2019	276
	Santos Dumont GPU	Intel Xeon E5-2695v2	Nvidia K40	2015	Below 500
	Santos Dumont Hybrid	Intel Xeon E5-2695v2	Intel Xeon Phi 7120P	2015	Below 500
	Santos Dumont CPU	Intel Xeon E5-2695v2	-	2015	Below 500
CNRS/IDRIS (France)	Jean Zay (with GPU)	Intel Xeon Gold 6248	NVIDIA Tesla V100	2019	64
	Jean Zay (no GPU)	Intel Xeon Gold 6248		2019	108
NCI (Australia)	Gadi	Intel Xeon 8274/8268	NVIDIA Tesla V100	2020	27
	Raijin	Intel Xeon E5-2690v4	NVIDIA Tesla P100	2012	Below 500
CMA (China)	PAI-BSystem	Intel Xeon Gold 6142	NVIDIA Tesla P100	2018	147
	PAI-ASystem	Intel Xeon Gold 6142	-	2017	159
MPG (Germany)	COBRA	Intel Xeon Gold 6148	-	2018	51
	DRACO	Intel Xeon E5-2698v3	-	2016	Below 500
NVIDIA Corp. (US)	Selene	AMD EPYC 7742	NVIDIA A100	2020	5
	DGX SuperPOD	Intel Xeon 8174	NVIDIA Tesla V100	2020	26
Saudi Aramco (Saudi Arabia)	Dammam-7	Intel Xeon Gold 6248	NVIDIA Tesla V100	2020	10
	Unizah-II	Intel Xeon Gold 6248	-	2020	86
CSCS (Switzerland)	Piz Daint	Intel Xeon E5-2690v3	NVIDIA Tesla P100	2017	12
	Piz Daint Multicore	Intel Xeon E5-2695v4	-	2016	266

Source: Selection of TOP500 (DONGARRA et al., 1997) list of supercomputers.

The presentation of these heterogeneous supercomputers' sites shows that the HPC computational world is heterogeneous, both intra-node and inter-node. The challenge is enabling applications to use any combination of these systems efficiently. The decision about how many

nodes and what clusters to use should consider performance and efficiency. However, before this decision can be made, the application must cope and adapt to the system-level heterogeneity and exploit it to satisfy internal heterogeneous behaviors.

1.2 Applications' Perspective of such Complex Heterogeneous Systems

HPC supports many scientific applications built upon linear algebra kernels; these are the type of applications studied in this thesis. Traditional versions of these MPI applications equally divide their computational load by distributing their data into homogeneous nodes. Usually, homogeneous data distribution strategies rely on static partitioning to minimize communication (BLACKFORD et al., 1997). Because of the uniform nodes' power in the homogeneous case, the final distribution is usually cyclic with a pattern. Usually, the data-to-node assignment uses simple heuristics like equal volume division or round-robin alike techniques.

Many algorithmic challenges appear when transitioning to heterogeneous scenarios (DONGARRA; LASTOVETSKY, 2006; BEAUMONT et al., 2019). First, the ideal amount of data per node will be different. Depending on the application's data structure, this division may not be trivial. Although a simple division would be enough in one-dimensional data structures, with all data portions having an identical computational cost, specialized strategies are needed for higher dimensions or when particular data have different behavior or computational power needs (BEAUMONT et al., 2001a). Moreover, having different amounts of data per node results in various quantities of inter-node communication, as nodes that process more will need to communicate more. This communication difference means that the ideal trade-off between communication and computation per node is more critical in heterogeneous scenarios than in homogeneous ones. The same applies if the amount of computation per data element differs, resulting in a critical path that the nodes' heterogeneous capabilities will influence. Finally, there is a technical programming perspective. Simpler distributions, like cyclic ones, are easier to implement in traditional paradigms like MPI, and broadcast functions are more clearly applied. However, irregular communication patterns appear when dealing with arbitrary distributions, and other paradigms or middlewares may be necessary to reduce the developing complexity and application maintainability.

Furthermore, applications may compound different phases that have different computational needs and would admit distinct ideal data distributions. These phases could also exploit node resources differently, changing the perfect distribution for each one even more. For example, phases comprising data generation are typically more appropriate to traditional CPUs, while

some Single Instruction, Multiple Data (SIMD) compute-intensive operations, such as classical linear algebra kernels, could use accelerators to increase performance.

Classical BSP applications with strongly coupled MPI or MPI+X situations have barriers between phases, creating a synchronous execution. However, if the phases could overlap (an asynchronous scenario), each phase could use its best resource (with the better speedup ratio) instead of trying to use them all. This distinct computational demand per phase makes system-level heterogeneity a natural choice to improve load partitioning. Although overlapping phases increase possibilities in the overall node distribution, they entail more development and algorithmic challenges. These challenges of finding efficient distributions for heterogeneous nodes while considering communication and improving asynchronous executions make most applications only use homogeneous resources and miss an enormous opportunity. In this way, traditional programming paradigms, like BSP, are improper for these complex systems.

The limitations of the traditional paradigm include poor intra-node heterogeneity handling, low programming efficiency, unnecessary synchronism, and limited resource portability. Together, they suggest the resurgence of the task-based programming paradigm (BOSILCA et al., 2013; DURAN et al., 2011; AUGONNET et al., 2011; WU et al., 2015; THIBAULT, 2018). This paradigm adopts a more declarative way of programming and uses a runtime to make decisions, including dynamically scheduling work (tasks) during execution. The application describes individual tasks and data dependencies and structures them into a Directed Acyclic Graph (DAG). This approach's benefits include relieving the programming complexity of explicitly handling irregular communications and computational flow using a runtime and improving the cooperation of heterogeneous intra-node resources.

The runtime is responsible for scheduling tasks respecting the dependencies using many possible heuristics. This approach also allows easy overlap of single and multiple operations tasks. Also, some runtimes perform data transfers inter and intra-node automatically based on the DAG structure, reducing the development burden. The application developer still needs to give many hints to the runtime to assist it during execution, but once formulated, the code is highly portable to multiple systems. Examples of such runtimes are ParSEC (BOSILCA et al., 2013), OmpSS (DURAN et al., 2011), and StarPU (AUGONNET et al., 2011), the latter used in this work. These modern task-based runtimes provide a high-level programming abstraction with the required flexibility, facilitating the research and development of sophisticated static data distribution strategies across heterogeneous nodes. This approach seems more elegant and adequate to handle and combine heterogeneity from systems architecture and application needs.

Hence, the task-based programming paradigm provides many opportunities to implement

elaborated algorithmic strategies and refined distributions required to address the challenges of utilizing heterogeneous resources. Moreover, this work uses DAG-related and task-based information to guide the strategies and the performance analysis.

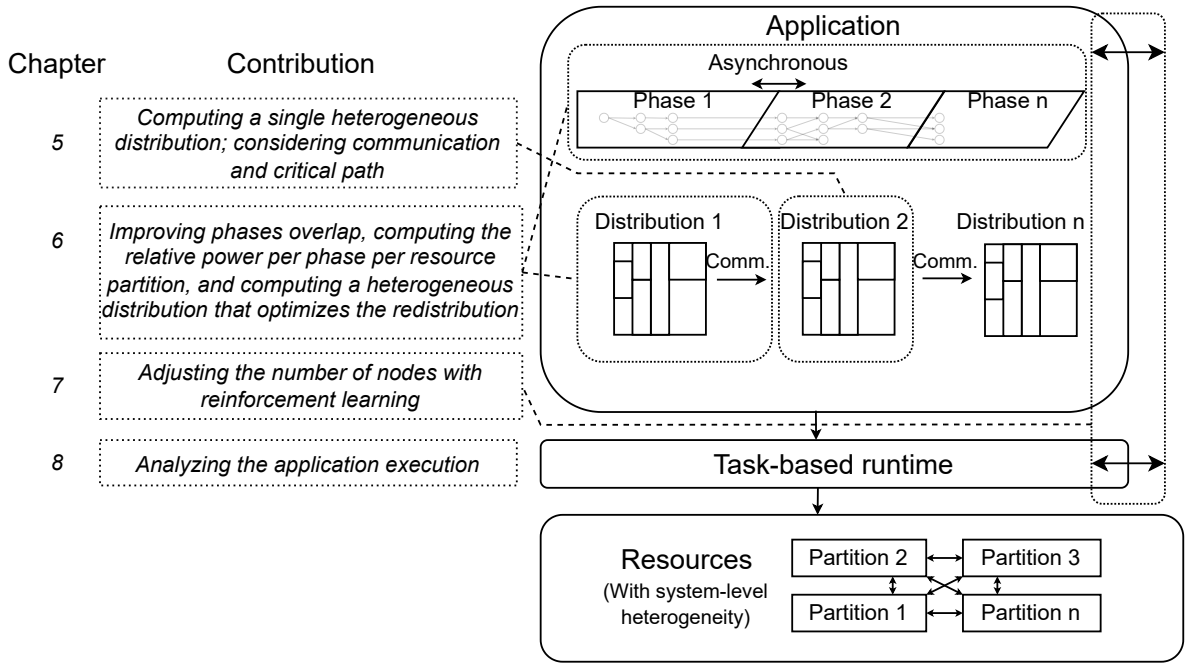
1.3 This Thesis Contributions and Structure

HPC applications require considerable computational power provided by supercomputers. These computational resources may present a system-level heterogeneity when there are two or more groups of nodes, each group with different hardware and computational power. Moreover, applications may show internal heterogeneous behavior because of distinct operations or multiple phases that can run differently at each intra-node resource. The increase of the intra-node heterogeneity in supercomputers and the difficulty of programming them encourage the use of robust parallel programming paradigms like the task-based one. While this paradigm provides enough flexibility, portability, and dynamism to handle such a complex scenario, many problems still need to be solved. All these circumstances make the problem of **distributing these task-based applications among heterogeneous nodes** challenging, although many opportunities for improving performance and resource usage arise.

The main goal of this work is to provide strategies and methods to improve distributions of task-based applications over system-level heterogeneous resources. Many challenges should be considered for an application to achieve the correct performance. The challenges tackled in this Thesis are: (1) The distributions for each application phase should consider not only the load balancing but the trade-off between communication and critical path as well; (2) Creating several distributions for multi-phase applications with different needs while considering their overlapping interaction; and (3) finding the ideal number of nodes per type for each phase.

Each challenge requires and uses the solution of the previous one. Ultimately, an optimized application would use all strategies to reach its full potential. Figure 1.2 presents the contributions highlighted on the application stack while showing the hierarchical structure of the challenges. The application executes using a task-based runtime and over a series of system-level (multiple partitions) heterogeneous resources. The application must inform the runtime about the data distribution, tailoring each one for each phase (operation). The contributions chapters organization of this thesis follows a natural order from the smallest element, a phase (distribution of just one operation), to the interaction of multiple phases (overlap and various distributions), to the application as a whole (adapting it as it goes and performance analysis). The main structure of this thesis follows.

Figure 1.2 – Main thesis contributions on the task-based application using heterogeneous resources



Source: The Author.

Preliminary Chapters. Chapter 2 presents the background for Task-Based applications, runtimes characteristics, and the StarPU ecosystem. It focuses on the applications' DAG structure from the base algorithms and how the runtime schedules tasks into many resources. The chapter also presents the applications that will be used in the remaining chapters. The first is the linear algebra library Chameleon, the second is the GeoStatistics application ExaGeoStat, and the third is the large datasets analysis library Diodon.

Chapter 3 presents many state-of-the-art related works. It starts with the general problem of distributing an application (primarily linear algebra 2D structured ones) on both homogeneous and heterogeneous scenarios. After, it introduces classical work and techniques on load balancing, followed by a presentation of related work on the management of multi-phase applications considering both multiple distributions and communication. Finally, we briefly present how some machine learning and reinforcement learning algorithms have been used in related HPC problems.

Methodology. Chapter 4 presents the methodology used in this work to perform controlled experiments and comprehensive performance analysis. The methodology builds on real executions with careful experimental control, simulations evaluating their reliability in this context, and analysis of these experiments using analytic metrics, traces, and visualization.

Contributions. Chapter 5 studies distributions for one possible operation, the LU factorization, which could be expanded to similar linear algebra ones. It provides the following contributions.

(a) A strategy that improves the performance of applications by reducing communications in the critical path. This situation occurs when the parallelism in the DAG diminishes. The approach constrains the distribution to use fewer resources toward the end of the algorithm. **(b)** A strategy to improve the computational load balancing of a given static distribution considering multiple tasks and heterogeneous resources while increasing communication. **(c)** A methodology to combine (a) and (b).

Chapter 6 considers the problem of multi-phase applications involving possibly multiple heterogeneous distributions. The case study relies on the multi-phase applications Exageostat and Diodon. The Chapter has the following contributions. **(d)** Optimizations to improve application phase asynchronism and enhance multi-distributions. **(e)** Strategy to generate efficient heterogeneous distributions for multi-phase applications with different resource performance affinities while considering phase overlap. **(f)** A technique to derive other distributions from a major one that reduces communications when performing the redistribution.

Chapter 7 presents strategies for the application to actively learn and adapt to the best heterogeneous nodes it can access. The Chapter has the following contributions. **(g)** An analysis of this problem's main characteristics and explain why generic optimization and learning techniques will likely fail. This analysis motivates the design of specific variations of a reinforcement learning technique based on the Gaussian process. **(h)** A comprehensive performance evaluation with 16 different heterogeneous machines and workloads that compare the proposed solutions with other generic optimization methods (Brent, Bandits, GP-UCB). Among these various methods, the GP-based variant is the only robust and parsimonious method to quickly reach the optimal configuration in multiple scenarios. **(i)** An actual implementation of the method to enable the application to adapt during execution, demonstrating the low overhead.

Chapter 8 discusses techniques and methods to analyze the behavior of task-based applications. Specifically, **(j)** techniques with extra focus on the platform and application heterogeneity and the actual progress of the application.

Chapter 9 concludes this thesis with the major contributions, the next directions, and the list of publications.

Finally, a companion for the thesis is publicly available[4]. It includes the data, scripts, and experiments' traces to replicate the analysis and figures.

[4] <https://gitlab.com/lnesi/thesis-companion>

2 TASK-BASED PROGRAMMING AND SELECTED HPC APPLICATIONS

The Task-based programming paradigm (THIBAULT, 2018), also known as the Data Flow Scheduling (DONGARRA et al., 2017) or the Asynchronous Many Task (AMT) paradigm and runtimes (HUMPHREY; BERZINS, 2019), uses a more descriptive approach, not imperative, to define an application. Applications express their internal algorithms with tasks and dependencies without explicitly defining where the parallelism is and where and when those tasks execute. A runtime decides the tasks' scheduling and placement during execution using internal algorithms and heuristics. Because of this flexibility and loose coupling on the platform, Dongarra et al. (2017) points out that Task-based programming will be the desired paradigm for exaflop systems. Although utilizing tasks and dynamic scheduling is an old concept (CODD, 1960), it is attracting popularity in many new and modern projects (DONGARRA et al., 2017; THIBAULT, 2018; HOUSSAM-EDDINE et al., 2020).

This Chapter has four Sections to cover the background of the Task-based paradigm. Section 2.1 describes the principles and structure of a common task-based application, giving examples of typical algorithms, parallelism, and asynchronous opportunities. Section 2.2 presents a brief discussion about task-based runtimes, examples, and traditional scheduling algorithms. Sections 2.3 introduces some application benefits when using this paradigm, including runtime management of data transfers, communication and computation overlap, and seamlessly dynamic asynchronous phases execution. Finally, Section 2.4 describes the runtime used in this thesis, StarPU, and its ecosystem, showing some internal operations and particularities that made it the desired runtime for this work. This last Section also covers applications that rely on the StarPU runtime and will be used to evaluate, in other chapters, the thesis proposed strategies.

2.1 Application Structure

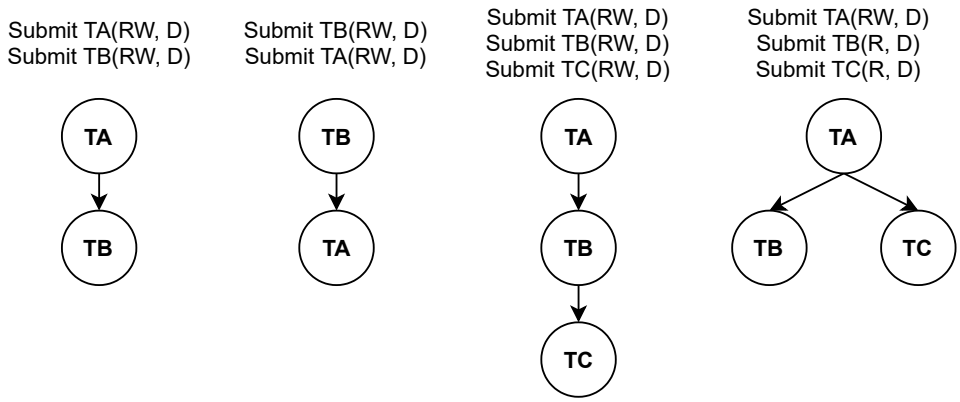
The elementary component of the application is a Task. The OpenMP specification, for multi-platform shared-memory parallel programming, defines a task as *"A specific instance of executable code and its data environment that the OpenMP implementation can schedule for execution by threads*" (OpenMP Architecture Review Board, 2020). This definition captures the essence of tasks also for other runtimes. However, this is a restricted definition in some situations, including runtimes with heterogeneous resources or ones that enable tasks to run over multiple a group of workers (like StarPU). Therefore, a possible task characterization is: "A collection of implementations of one particular algorithm and its data dependencies that a

runtime can schedule to one or a group of workers associated with computational resources."

The other main component of this paradigm is the data dependencies, which are the data inputs or outputs of a task. The successive data usage through multiple tasks will create a flow that expresses the application progress and constructs the task DAG. In the Sequential Task Flow (STF) strategy (AGULLO et al., 2016), the application only has one thread and submits the tasks sequentially, using the flow method to construct the DAG. In a simple example, with tasks `TA` and `TB`, if task `TA` generates a data `D` that `TB` will read and write, there will be a direct dependency from `TA` to `TB`. When scheduling these tasks, `TB` will only be ready to be scheduled when `TA` finishes and saves its data `D` modifications. In this way, we define that `TB` depends on `TA` (`TA` $\rightarrow$ `TB`), and `TB` is ready only when all its dependencies are satisfied (when other tasks that modify shared data dependencies conclude). Different workers (resources) can execute two tasks in parallel if they are not dependent on each other.

When describing the application, one can solely create tasks informing data input and output and submit them to the runtime. There is no need to denote task dependencies explicitly (although some runtimes allow it). The runtime can infer the DAG structure, using Bernstein conditions (BERNSTEIN, 1966), from the submitted tasks' order and the data access level (read, write, or both). Considering our earlier example and this non-explicit dependency declaration, if `TB` was submitted before `TA`, the dependency would be `TB` $\rightarrow$ `TA` and not `TA` $\rightarrow$ `TB`, as the submission order suggests that the data `D` will be modified by `TB` and `TA` will use this updated data. Figure 2.1 presents other sequences of submission and access examples (data access and data block in parenthesis), showing the resultant DAG.

Figure 2.1 – Examples of tasks submission and the respective DAG structure



Source: The Author.

Many algorithms already have task-based versions. For example, those using tile decomposition for their data are easy to express. The LU decomposition is one classical algorithm of linear algebra that decomposes the matrix A into the triangular lower matrix L and triangular

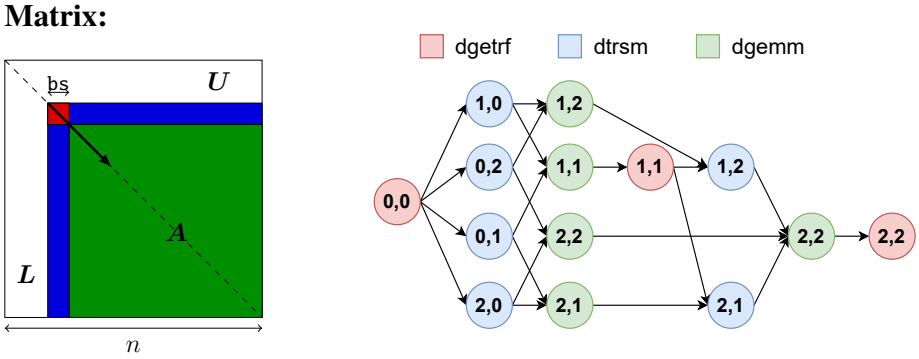
upper matrix U, $A = LU$. It is widely used to solve systems of linear equations, as one of the resultant triangular matrices can be easily used with the right-hand matrix to solve the linear system. Algorithm 1 presents the LU decomposition tile-based version. It has three main kernels, all part of Basic Linear Algebra Subprograms (BLAS), `dgetrf-nopiv`, `dtrsm`, and `dgemm`. Each kernel accesses one or more tiles with a specific permission type (read or both read and write). The task-based version can use this algorithm to naturally unroll the DAG and submit the tasks with the correct dependencies to the runtime.

Algorithm 1: Tile-based algorithm for the LU decomposition

```
1  Input: nb Number of blocks
          n Number of Matrix cells
          A[1...n][1...n] Matrix divided and accessed through nb × nb blocks
2  Output: A[1...n][1...n] Storing L and U
3  for k = 0; k < nb; k++ do
4      dgetrf-nopiv(RW, A[k][k]);
5      for i = k+1; i < nb; i++ do
6          dtrsm(RW, A[i][k], R, A[k][k]);
7          dtrsm(RW, A[k][i], R, A[k][k]);
8      end
9      for j = k+1; j < nb; j++ do
10         for i = k+1; i < nb; i++ do
11             dgemm(RW, A[i][j], R, A[i][k], R, A[k][j]);
12         end
13     end
14 end
```

Figure 2.2 presents the LU tile-based data structure on the right with nb arbitrary number of tiles, each with size bs, and on the left, the DAG when $nb = 3$. Although the data representation

Figure 2.2 – The LU algorithm data regions of matrix A updated at iteration **k** (left) and the DAG for the tiled version with $nb = 3$ (right)



Source: The Author.

shows one iteration of the data structure update and where tasks are working, the flexibility of the task-based approach enables the computation of many iterations simultaneously, provided that task dependencies are satisfied.

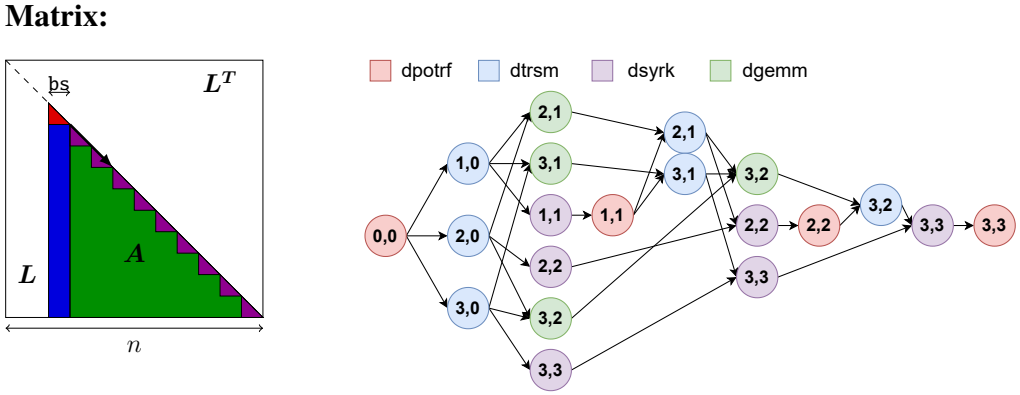
Another classic algorithm is the Cholesky factorization. It is a linear algebra algorithm that decomposes a hermitian, positive-definite matrix A into a triangular matrix L and its transpose L^T, $A = LL^T$. Algorithm 2 presents the Cholesky tiled version with its four kernels: `dpotrf`, `dtrsm`, `dsyrk`, and `dgemm`. Similar to the LU case, one task-based application can use this algorithm to unroll the DAG and submit tasks with the correct data and permissions. The runtime will infer all the parallelism.

Algorithm 2: Tile-based algorithm for the Cholesky decomposition

```
1  Input: nb Number of blocks
          n Number of Matrix cells
          A[1...n][1...n] Matrix divided and accessed through nb × nb blocks
2  Output: A[1...n][1...n]
3  for k = 0; k < nb; k++ do
4      dpotrf(RW, A[k][k]);
5      for i = k+1; i < nb; i++ do
6          dtrsm(RW, A[i][k], R, A[k][k]);
7      end
8      for i = k+1; i < nb; i++ do
9          dsyrk(RW, A[i][i], R, A[i][k]);
10         for j = i+1; j < nb; j++ do
11             dgemm(RW, A[j][i], R, A[i][k], R, A[j][k]);
12         end
13     end
14 end
```

Figure 2.3 presents the data structure at one iteration for the Cholesky, the tasks which

Figure 2.3 – The Cholesky algorithm data regions of matrix A updated at iteration k (left) and the DAG for the tiled version with $nb = 4$ (right)



Source: The Author.

write on parts of its data (left), and the resultant DAG with $nb = 4$ (right). Again, different iterations of the Cholesky algorithm can execute concurrently, having tasks of various iterations running simultaneously because of the DAG guiding an asynchronous execution. These asynchronous iterations happen if the tasks of a particular block coordinate are ready and advanced more than other positions. This multiple-iteration execution can be beneficial in accelerating specific tasks or paths in the DAG, especially the critical path, which is composed of different iterations.

2.2 Tasks Scheduling and Runtimes

The task-based runtimes enable optimizations and adaptive behaviors for the dynamic and stochastic conditions they may face (THIBAULT, 2018; BOSILCA et al., 2013). The runtime will schedule tasks following an algorithm that usually accepts parameter customization. These scheduling algorithms can generally follow a centralized or per-resource queue of tasks. In the centralized case, all the resources will pull tasks from one queue. The scheduling algorithms will operate in how to populate it, considering different heuristics, like task priorities. In the one queue per resource approach, each resource will pull a task from its queue, and the scheduling procedures will decide to which resource queue each task should go. An example of this approach is the work-stealing strategy, where one worker can steal one task from another worker's queue when its own is empty. Other techniques use information about task duration, considering historical data or other models. One popular algorithm is HEFT (TOPCUOGLU; HARIRI; WU, 2002), which considers task duration for heterogeneous resources. After prioritizing the tasks by the upward or downward ranking, HEFT will decide about scheduling one task to a given resource queue by choosing the one that minimizes the overall application time. This operation can be formalized by:

$$r_{\text{chosen}} = \arg\min_{r}(e_r + w_{t,r}) \tag{2.1}$$

where r_{chosen} is the chosen resource, e_r is the expected ending time of resource r queue, and $w_{t,r}$ is the expected time duration of task t on resource r. One possible variant of this algorithm is the `DMDAS` present in StarPU (AUGONNET et al., 2011) that this work extensively uses. This algorithm also considers data transfers. First, ready tasks will be sorted by their priorities and then scheduled into workers. The equation will have another component, the total data transfer time. Many other variants may exist, considering other attributes, including but not limited to

energy usage, data locality in the memory hierarchy, and even multiple priorities per resource type (BRAMAS, 2019).

The development of task-based runtimes is active, and there are design and execution differences in all. Some of the differences are: (i) the application's approach to express tasks and dependencies; (ii) the inner workings for scheduling, data organization, and communication; (iii) features like support for heterogeneous resources, running in distributed machines, and fault-tolerance.

Some examples of runtime are the following. Cilk (BLUMOFE et al., 1996) is one of the first ones that spread DAG of tasks concepts and inspired other runtimes. Intel Threading Building Block (Intel TBB) (ROBISON, 2011) was also another older approach for multicore usage. Intel refurbished and renamed it to Intel oneAPI Threading Building Blocks (oneTBB) (Intel Corporation, 2021a). All these examples solely consider one computational node and CPUs. The usage of multiple nodes and accelerators is only possible with other frameworks. Another interesting case is OpenMP (OpenMP Architecture Review Board, 2020), which is widely used to program shared-memory systems. It started offering task support in version 3, and it is in constant development with the addition of the `depend` clause in version 4, which allows data dependencies. Also, since version 4, it allows the usage of heterogeneous systems, with the `target` clause. Charm++ (KALE; KRISHNAN, 1993) is another popular framework that uses a declarative approach slightly different from the DAG-oriented one. The main feature of Charm++ is balancing work through many resources in many nodes during runtime in BSP-like applications, not in a full asynchronous case. It uses the over-decomposition of work into units and the migration of them. The support for GPUs is also integrated (VASUDEVAN et al., 2013).

Other projects released almost together by the high-performance linear algebra community are: ParSEC (BOSILCA et al., 2012), OmpSs (DURAN et al., 2011), and StarPU (AUGONNET et al., 2011). ParSEC uses Parameterized Task Graphs (PTG) to describe the application's tasks' structure, providing the traditional STF as well. The PTG interface enables a static definition of the algorithm that does not require the whole and memory-costly unrolling of the DAG in all machines. To control the distribution of work in different nodes, the data distribution in ParSEC follows a user-defined function. OmpSs uses code annotations similar to OpenMP tasks. The application must use MPI to execute through multiple nodes, and all communication is its responsibility. StarPU uses STF with an API to create and insert tasks. It has the flexibility to control data and task placement and has an active development ecosystem. Section 2.4 presents further information about StarPU and why it was chosen to be the runtime of this

work. Nevertheless, other similar initiatives include Uintah (MENG; HUMPHREY; BERZINS, 2012), which focuses on fluid-structured problems; HPX (KAISER et al., 2014), which focuses on providing the facilities of the C++ standard; the Minos framework (GIOIOSA et al., 2020), which focuses on the workflow of independent applications; and the Chameleon programming environment (KLINKENBERG et al., 2020) which focuses on the reactive load-balancing given a performance metric.

2.3 Application Benefits

Applications that carefully use and explore the task-based programming paradigm have many benefits. One example is the ability to program complex applications, expressing them more declaratively. It only requires a description of small recurrent tasks and dependencies that will become a complete dependency graph and can be divided into many nodes smoothly. This declarative structure can have additional information like task priorities, and the runtime can infer their performance models, improving the dynamic scheduling. Another benefit is the dynamic scheduling during execution time. While some argue that the overhead generated by the scheduler may cause a slowdown when there are too many tasks, a dynamic scheduler presents improved portability to different resources without the need to change applications' source code or explicit tune for a set of resources (THIBAULT, 2018).

Nevertheless, the runtime can take care of many operations that, in traditional paradigms, would be the programmer's responsibility. One significant and impacting one is memory management, especially the automatic communication and computation overlap. In this seamless management for the programmer, the runtime can use the DAG as a reference to prefetch data for different resources (GPUs) and nodes. This management can also provide other features, including out-of-core memory and cache behaviors. Memory management can also operate in multi-node executions. Because of the DAG and the distributions, the runtime can infer when other nodes need data and handle all inter-node communications internally.

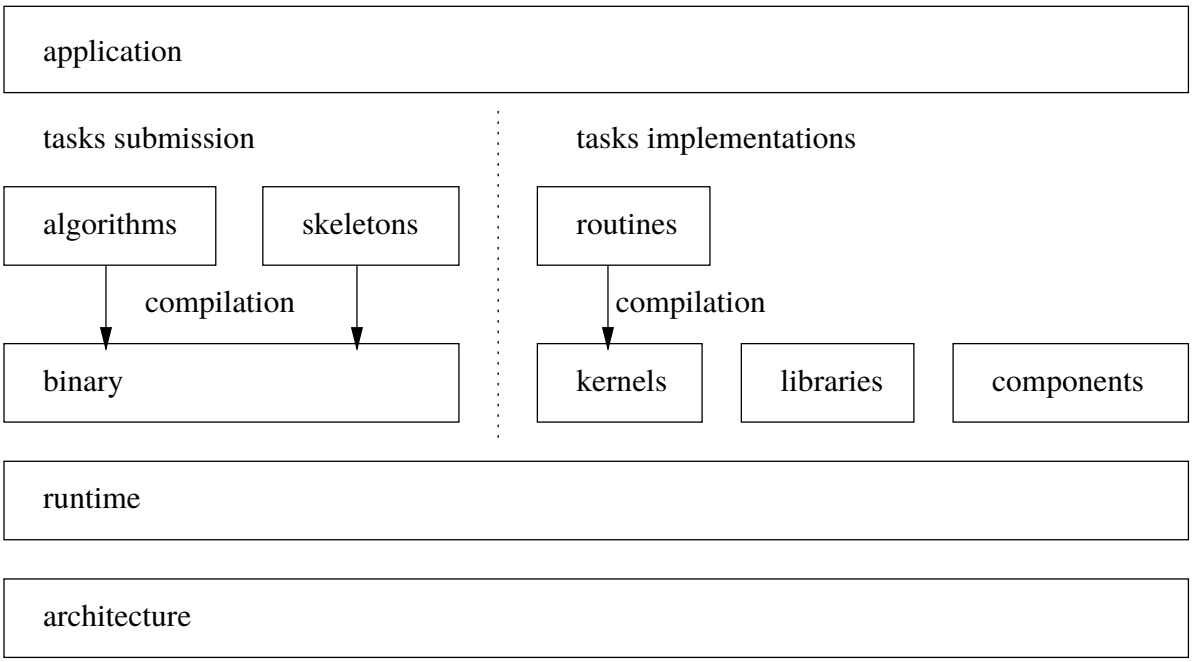
As discussed in the LU and the Cholesky case, the asynchronous execution of multiple iterations or even steps is possible because the DAG has all the necessary information to control the flow. This behavior differs significantly from traditional applications that follow BSP or similar approaches. In classical BSP Cholesky and LU applications, there will be a synchronous barrier at each iteration, reducing the possible total amount of task parallelism. When the number of available tasks reduces at each iteration's end, the application phase may not have sufficient computational work for all resources, creating idleness until a new iteration begins.

Also, applications with many operations or steps can exploit the asynchronism the task-based design provides. While traditional applications would synchronize at the end of each operation, the task-based approach is naturally asynchronous since the runtime has the application DAG to control the execution flow correctly.

2.4 The StarPU Ecosystem

The StarPU task-based runtime (AUGONNET et al., 2011), developed at Inria[1], is a library designed to work on heterogeneous resources. In StarPU, each task can provide multiple implementations, and it is the responsibility of the runtime during execution to decide which one to use based on the scheduling heuristic. The runtime assigns each resource to an entity called a worker, which is the entity that effectively executes the tasks. Figure 2.4 presents the internal structure of the StarPU runtime over a computational node. The application interacts with the runtime submitting tasks with many implementations that may access external libraries (OpenBLAS, cuBLAS, for example).

Figure 2.4 – StarPU stack over node resources



Source: Thibault (2018).

StarPU has an API for C, Fortran, and Julia programming languages. The standard usage of the API starts with a call to the `starpu_init` function to initialize the runtime. The developer can define tasks as a `starpu_codelet` struct containing the implementation's functions for the

[1] <https://www.inria.fr/>

task. The `starpu_task_insert` function inserts a task and the `starpu_task_wait_for_all` one waits for all submitted tasks to finish (a blocking call). After the end of the program, a call to the `starpu_shutdown` function is necessary to close the runtime. Algorithm 3 presents a basic example of a C application using the StarPU API, defining one task named `hello_world` with a CPU implementation using the `func_cpu` function. The application initializes the runtime, submits and waits for one task, and then shutdowns it.

Algorithm 3: Basic C application using StarPU API

```
1  void func_cpu (void *buffers[], void *args) {
2      printf("Hello World!");
3  }
4  struct starpu_codelet codelet_world = {
5      .cpu_funcs = {func_cpu},
6      .nbuffers = 0,
7      .name = "hello_world",
8  };
9  int main () {
10     starpu_init(NULL);
11     starpu_task_insert(&codelet_world, 0);
12     starpu_task_wait_for_all();
13     starpu_shutdown();
14 }
```

The StarPU follows the STF strategy (AGULLO et al., 2016). In this approach, the DAG construction is easy, and after inserting all tasks, no extra work is necessary from the application. Usually, the application will unroll the DAG and wait for all tasks with a synchronization wait call. Nevertheless, in this approach, the runtime can block one insertion and stop task submission in specific situations, like when no more memory is available, or a particular threshold of tasks is reached.

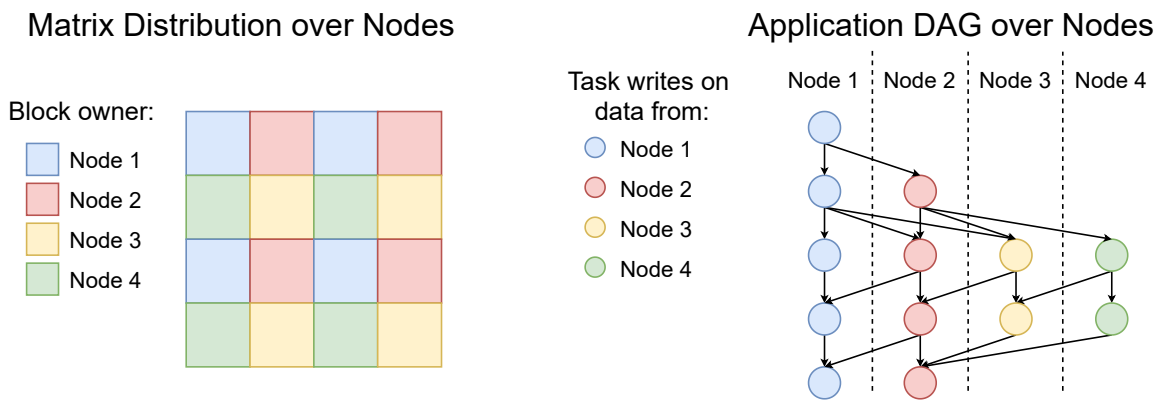
Some of the scheduling algorithms of StarPU can use a performance model of the tasks' duration. The application can define a model for each task based on parameterized equations or historical data. In the history-based performance model, StarPU will save the task's execution time in an auxiliary sampling directory each time it executes it, associating the sampling with a task type and input size. One can compute the mean and variance of the tasks' duration after many executions. This approach requires the tasks' duration calibration, running them a minimum of times to have a decent duration model. The default calibration amount in StarPU is ten times. However, this historical method may not work in complex scenarios where the task's duration is influenced by factors other than type and size. In the parameterized equations method, StarPU will use an application-defined function to get the expected task duration. This function

can receive arbitrary parameters, such as the data input structure particularities (MILETTO et al., 2022).

The StarPU MPI (AUGONNET et al., 2012) module provides the distributed execution of StarPU across multiple nodes. In this module, each node will have a unique StarPU process without a global scope. There are no control messages to synchronize or inform the complete application DAG. It actuates in an entirely decentralized method. The application has to submit in each node all the tasks that require data that this particular node should receive or send. This partial DAG unrolling enables the runtime to concretize all unilateral communications functions. When the application unrolls the DAG, and StarPU notices that another node's task requires local data, StarPU will create a send requisition, more specifically, a `MPI_isend`. When it unrolls a task that will execute locally and requires external data, it will issue a `MPI_irecv`. There are some situations where other nodes send data before the `irecv`. These earlier requests are possible because there is a handshake message per communication, so the process does know when different nodes intend to send data. All this request control moves the communication responsibility from the MPI implementation to StarPU. Moreover, StarPU offers the traditional MPI API backend plus a specialized backend for the NewMadeleine (DENIS, 2019) MPI implementation. One big difference is that the NewMadeleine implementation has message priorities informed on the `isend`.

The data in StarPU-MPI is spread across the nodes using a static distribution because of scalability concerns, and each data has a node owner. When tasks are submitted, the developer does not inform where (node) the task will execute. The runtime will verify which data the task writes and which node owns this data, selecting the appropriate node to execute this task. Figure 2.5 illustrates this approach, where blocks of the matrix are spread (owned) across four nodes, and the tasks that write on one particular data will execute on the respective node owner.

Figure 2.5 – Example of matrix ownership and DAG distribution among nodes



Source: The Author.

Although StarPU initializes data with a static node distribution, this does not limit the runtime's flexibility to execute any particular task on any node, as the application can change data ownership during the execution. However, an important technical aspect in StarPU is that, because of the STF, if the developer wants to move one task from one node to another, the application should inform this movement before this particular task insertion. In this way, to create dynamic decisions about data movement during execution, the movements should be submitted earlier than the submission of the desired task to change. Moreover, every node that contains this data should also inform the movement to make all instances of the runtime in every node know the new data ownership.

Another valuable feature of StarPU is the simulation of its applications over arbitrary infrastructures using the Simgrid (CASANOVA et al., 2014) framework, enabling the StarPU+Simgrid module (STANISIC et al., 2015b). In these simulations, StarPU will use the performance models of the tasks on arbitrary infrastructures to simulate the application behavior. Executing the actual task function is unnecessary, and the simulation can only account for its duration. The StarPU internal operations, like task scheduling, are performed over a host machine that can collect execution traces. The distributed version of StarPU is handled naturally as an MPI application running over Simgrid.

Reasons for selecting StarPU. Although this work selects the StarPU runtime because of its features and ecosystem convenience, there is no known restriction of the proposed strategies to other runtimes as long as they provide the flexibility of redistribution capabilities. However, this work uses StarPU among the various systems because its features include: its capacity to use heterogeneous intra-node resources, its task declarative C-like programming style, its ability to run over many distributed nodes using StarPU-MPI, the simulation feature with the StarPU+Simgrid module, the capability of using a developer-informed distribution and its ability to do data redistribution in a declarative manner. StarPU is generally very flexible when building applications, providing all the necessary features this work needs. The comprehensive tools for performance analysis of applications, the ecosystem containing other scientific applications, and the extensive documentation are also reasons for this choice.

2.4.1 The Chameleon Linear Algebra Library

Chameleon (AGULLO et al., 2010) is a state-of-the-art dense linear algebra library that provides most of the BLAS/LAPACK operations and others through a task-based version, targeting heterogeneous and modern infrastructures. Chameleon offers a unified API to use its

operations while preserving the ability to rely on different runtimes, including ParSEC, StarPU, and OpenMP. The library can perform its operations in heterogeneous intra-node resources, including CPUs and GPUs, and mostly let the runtime decide where to schedule such tasks. For this, each task may admit multiple implementations in Chameleon. Examples of operations that Chameleon provides are matrix-matrix multiplication, LU factorization, Cholesky decomposition, and QR factorization[2]. Each operation is a function submitting the tasks in the correct sequence to the runtimes.

The library relies on a definition of a matrix structure that helps in runtime flexibility. The matrix is divided into blocks, and the tasks use those blocks. Users of Chameleon should construct such a matrix to utilize the operations. Chameleon uses the traditional block-cyclic distribution (further explained in Section 3.1) for homogeneous nodes of ScaLAPACK (BLACKFORD et al., 1997). Also, Chameleon provides a set of control functions that will call the low-level runtime ones. For example, the ability to change the owner node of a block in StarPU is abstracted on a similar function of Chameleon. However, the Chameleon one updates its structure and implements it using the appropriate runtime function (when available). With such flexibility, Chameleon is an excellent library to be used as a starting point for other applications that may require its BLAS/LAPACK operations.

2.4.2 The ExaGeoStat Unified GeoStatistics Framework

ExaGeoStat is a machine learning application for GeoStatistical data that relies on the Gaussian process (GP) framework and can predict missing observations (ABDULAH et al., 2018). It is developed using the task-based programming paradigm and relies on the Chameleon task-based dense linear solver and StarPU. ExaGeoStat can rely on different runtimes like StarPU or ParSEC. ExaGeoStat interpolates spatial data (X, Z), where X corresponds to the measurement locations and Z corresponds to the measurements, with a Gaussian process whose smoothness and scale are controlled by a set of parameters θ that requires adjustments to the data. Therefore, the application iteratively optimizes the log-likelihood of θ through Equation (2.2):

$$l(\theta) = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_\theta| - \frac{1}{2}\mathbf{Z}^T\Sigma_\theta^{-1}\mathbf{Z}, \tag{2.2}$$

where Σ_θ is an $n \times n$ covariance matrix built from X representing the similarity between measurement locations, which is computed through a covariance function K_θ (i.e., $\Sigma_\theta[m, n] =$

[2]The complete operations list is available at: <https://gitlab.inria.fr/solverstack/chameleon>.

$K_\theta(X_m, X_n)$). Although Machine Learning commonly uses the squared exponential (Gaussian) covariance function, the Matérn covariance function is more appropriate for geostatistics data, which can be relatively rough.

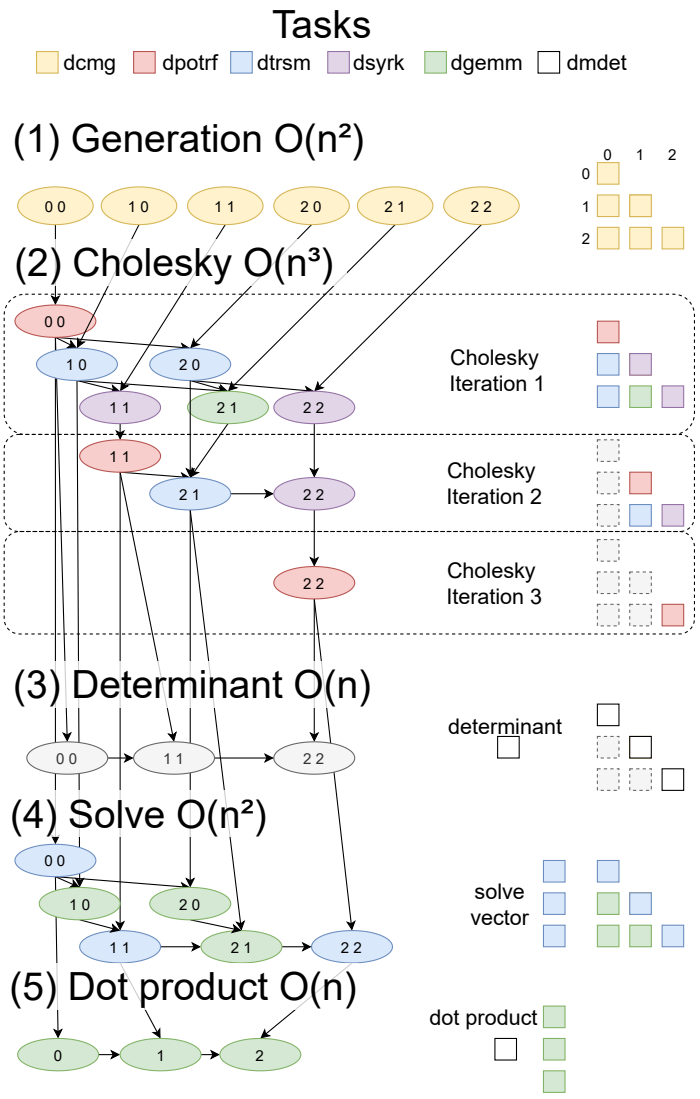
Therefore, this optimization requires the computation of the determinant and the inversion of a large, dense, symmetric, and positive definite matrix Σ_θ at each optimization iteration. To this end, this matrix is first decomposed using Cholesky factorization ($\Sigma_\theta = F^T F$) and used through a triangular solve ($F^{-1}Z$) and a dot product to compute the last term of Equation 2.2. The diagonal blocks of the factorization give the determinant of the matrix. Each optimization iteration has five phases as depicted in Figure 2.6: (1) Covariance matrix generation by Matérn function with complexity $O(n^2)$; (2) Cholesky decomposition with complexity $O(n^3)$ using the Chameleon library; (3) Matrix determinant with complexity $O(n)$; (4) Triangular solve with complexity $O(n^2)$ also using the Chameleon library; and (5) the dot product of the solve vector with complexity $O(n)$. Figure 2.6 depicts the DAG corresponding to one iteration. Also, ExaGeoStat authors reported excellent performance and scalability results with homogeneous multi-core systems (ABDULAH et al., 2018).

Usually, applications focus on their most computationally intensive phase, in this case, the Cholesky factorization, setting up distributions and scheduling priorities based solely on it. However, each phase has a different computational necessity and suitability with accelerators. ExaGeoStat distributes the data for all operations using the traditional block-cyclic distribution for homogeneous nodes of ScaLAPACK (BLACKFORD et al., 1997), the default option in the Chameleon library. Where the matrix $n \times n$ is divided into $nb \times nb$ blocks and cyclically attributed to the nodes. Although the principal operation of the Cholesky factorization, the general matrix multiplication kernel `gemm`, performs well in GPUs, the generation's main task, the Matérn covariance function, is only available through a costly CPU implementation. Therefore, the generation phase makespan can be longer than the Cholesky factorization in small and medium cases (GRAMACY, 2020), even with one complexity order of difference. ExaGeoStat official public repository[3] version presents two options for execution: (1) Synchronous, with a synchronization point between each phase, and (2) Asynchronous, where the synchronizations' barriers between factorization and determinant, and solve with the dot product disappear. Yet, this second option is not a fully asynchronous version.

As we will show in this work, when considering hybrid nodes (CPU+GPU), the Matérn covariance function raises severe load-balancing issues. Indeed, some phases better employ GPUs while others better utilize CPUs. Thus, it is natural to exploit system-level heterogeneity

[3] <https://github.com/ecrc/exageostat>

Figure 2.6 – ExaGeoStat iteration DAG for $nb = 3$



Source: The Author.

by mixing different node types to process each phase as efficiently as possible. However, a heterogeneous set of computing nodes will require different distribution strategies than the traditional block-cyclic distribution. Such heterogeneity would even imply different data distributions for each phase, with data redistribution occurring during the execution of the phases.

2.4.3 The Diodon library for large datasets

Diodon is a C++ library to compute multivariate data analysis of large datasets. It provides the methods of Principal Component Analysis (PCA), Multidimensional Scaling (MDS), and Correspondence Analysis (CoA). These methods rely on the Singular Value Decomposition (SVD), which unfortunately does not scale well with matrix dimension and is quite hard to

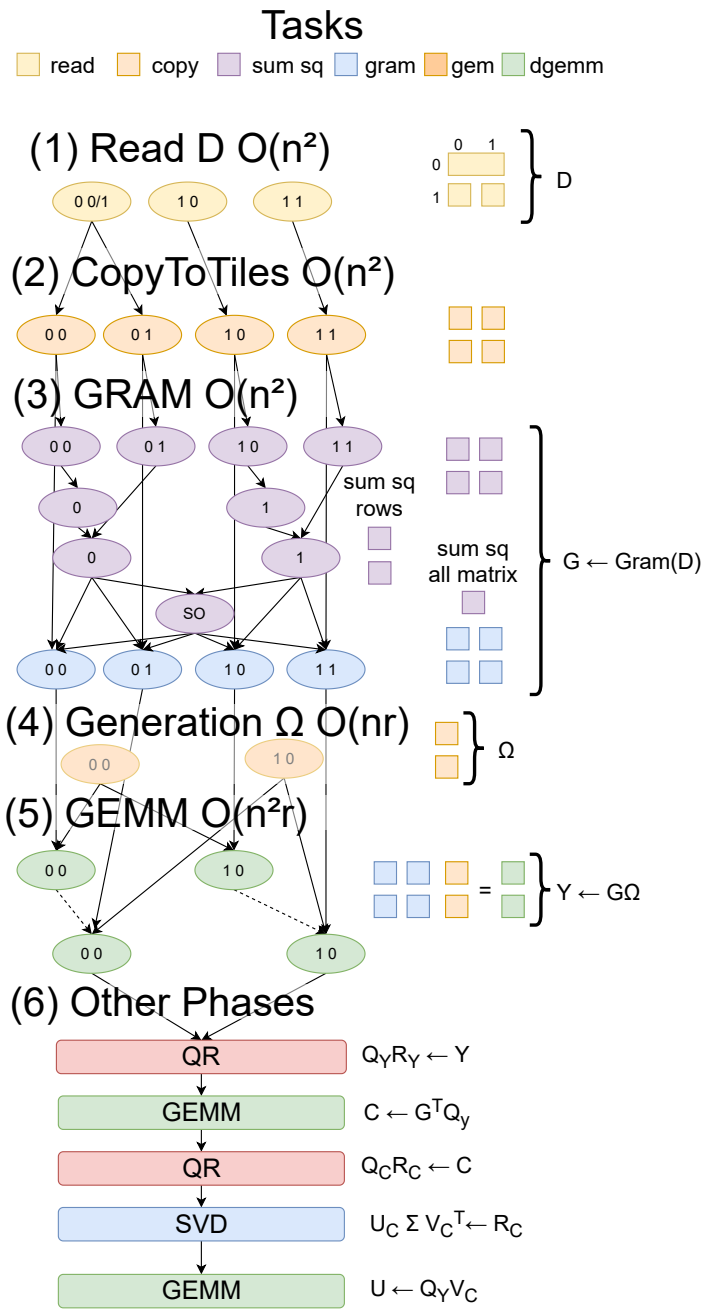
distribute and parallelize. However, it can be accelerated using the Randomized Singular Value Decomposition (rSVD) approximation. This work focuses on the MDS operation that takes as an input the matrix of dissimilarities D between n elements and finds n vectors $x_i \in \mathbb{R}^m$ such that $\forall i, j \in 1..n\ D_{i,j} \approx \|x_i - x_y\|$. Being m much smaller than the dimension of the original elements. This abstract mapping allows for simpler data visualization.

The sequence of operations for the approximation of the MDS method using the rSVD is presented in Figure 2.7, depicting the Diodon DAG. The matrix D is structured into $nb \times nb$ blocks of size $bs \times bs$. The first phase is the (1) read of D through `hdf5` files. This step may assume a different distribution because the `hdf5` files may be organized in a different grid than the one used by the linear algebra operations. If the `hdf5` file has n columns, each read task will read a complete set of rows (each set with bs rows). Otherwise, there will be a task for each set of rows on each file, totaling rb read blocks per bs rows. In the next step (2), `copypaneltotile` tasks will be responsible for mapping this grid system to the final one, generating $nb \times nb$ blocks into the matrix D. Diodon will then apply the (3) Gram operation, which requires computing the sum of squares of each block, aggregating them for each row (and columns for other operations), and finally, for the whole matrix. Then, with these values, it is possible to compute the Gramian of each block. While this happens, Diodon can (4) generate the random Gaussian Ω (of dimension $n \times r$) matrix used for the rSVD. Finally, a (5) matrix multiplication operation is performed with D and Ω, followed by a (6) QR factorization and a synchronization. Subsequently, it performs another matrix multiplication and QR factorization, and Diodon can apply the SVD in this matrix (which is much smaller than the original D). Since the SVD is not yet available in the task-based paradigm, a call to the traditional BLAS library is made. After the SVD, post-processing operations extract and adjust the eigenvalues and eigenvectors.

The read phase of Diodon uses the `hdf5` library (FOLK et al., 2011); hence, only CPU workers perform the read tasks. The original version of Diodon performs IO with only one read task at a time, as the library `hdf5` is not thread-safe. We enabled multiple read tasks simultaneously by instantiating `hdf5` with `dlopen` several times, so each worker has a unique and isolated library copy. The ideal number of parallel reads may vary per node and storage resource, as multiple simultaneous tasks create contention. In our experiments (with computational nodes equipped with SSDs), the performance improvements reach a plateau in reading data after five or six concurrent reads, which means that the contention generated when adding a new worker degrades performance slightly. In the rest of the paper, the experiments use five read workers for all SSD and distributed file-system machines.

An interesting remark when analyzing the structure of Diodon's DAG is that there is an

Figure 2.7 – Diodon MDS DAG for $nb = 2$



Source: The Author.

algorithmic synchronization point when the sum of squares is computed for the whole matrix (task SO in Figure 2.7). All the Gram operation tasks and following phases (except for the low-cost Ω generation) need to wait for the computation of the SO task. Another problem is that this task requires a complete read of matrix A. This synchronization limits resource usage because all tasks before SO do not use GPUs, and the read operation may take some time, effectively making all GPUs idle until the complete read of the matrix.

Another remark is that in the original version of Diodon, the QR operation was syn-

chronous because of the temporary memory workspace allocation for this operation. Anyway, the synchronization point in the Gram operation before already limits the overlap, so any optimization in the QR synchronization would have limited improvement. In this work, we will focus on the overlap and the performance of the first five phases plus the first QR factorization when its termination will require a synchronization barrier.

Moreover, the SVD operation is performed directly by the BLAS libraries and not handled by the Task-Based Chameleon, requiring an entire synchronization point. Finally, although MDS uses a symmetric matrix, we study operations that use the complete matrix computation to further consider the other methods, including PCA and CoA. The authors of Diodon focused on achieving performance with homogeneous nodes with large matrices in double and single precision. We aim in this thesis to further improve performance by exploiting system-level heterogeneity and application phase requirements.

2.5 Opportunities in the Heterogeneous Context

Task-based applications and runtimes offer many capabilities to exploit computational resources. However, the applications need to define the load distribution, and this problem should consider each particular case, such as application characteristics, intra-node resources, inter-node communication, and system-level heterogeneity.

While the runtimes may technically support the execution over system-level heterogeneous resources, their performance depends on how the application is structured (DAG, for example) and how the developer distributes it across the nodes. In both cases, the knowledge of how an application with asynchronous operations runs over system-level heterogeneous resources is limited and has great potential to be explored. As this thesis will show, many challenges appear that require application-based solutions. In this way, approaches or strategies that help task-based applications over heterogeneous system-level resources are desirable.

3 RELATED WORK: LOAD DISTRIBUTION

This thesis evolves across a broad spectrum of problems related to data distribution and load balancing of HPC applications over system-level heterogeneous resources. Section 3.1 introduces the problem of distributing data across homogeneous and heterogeneous resources, focusing on linear algebra problems, especially considering the 2D matrix partition and distribution case. Section 3.2 discusses dynamic strategies to balance the computational load after one initial distribution. Section 3.3 presents the case when applications have multiple steps or phases, and each has an ideal distribution, with the necessity of redistribution or multiple distributions among phases. Section 3.4 introduces the usage of reinforcement learning or other adaptive methods for optimizing HPC behavior, including parameters, configurations, and systems. Finally, Section 3.5 presents problems and opportunities for contributions in all these fields.

3.1 Data Distribution for Linear Algebra

The division of data and computation for distributed nodes is a fundamental element in parallel programming. Partitioning is an essential step in classical parallel applications design methodologies like PCAM (FOSTER, 1995). This methodology consists of four stages: Partitioning, Communication, Agglomeration, and Mapping, with the division of data and work occurring in the Partitioning and Mapping steps. The load distribution can optimize multiple objectives. The most common goals are improving the application performance (makespan) by decreasing idle times or balancing the load of the processors, reducing the total communication to avoid network contention, or increasing the available parallel computations. However, other objectives are possible, like reducing energy consumption or the cost of platform utilization.

The strategies for distributing data and computation among resources can be a static, a dynamic, or a hybrid approach (SHIRAZI; KAVI; HURSON, 1995). The static partition is performed once before all the computation and remains stationary during the execution of the application. This partitioning problem is dependent on the application domain. For example, classical algebra linear operations must partition a 2D matrix among the resources. However, the optimal solution for the static 2D strategy is an NP-complete problem (BEAUMONT et al., 2002a). Dynamic strategies rely on distributing the workload during the execution of the application. Most task-based runtimes, including StarPU (AUGONNET et al., 2011), adopt this approach for intra-node resource scheduling. Finally, a hybrid approach has both static

and dynamic techniques. A hybrid example is the StarPU-MPI module, which uses dynamic scheduling for intra-node tasks and static data distribution among different nodes mainly because of scalability concerns. The application must inform this static distribution, while one of many heuristic algorithms of StarPU performs the dynamic intra-node scheduling of tasks.

The following subsections present the distribution problem and state-of-the-art solutions for homogeneous and heterogeneous computational resources.

3.1.1 Homogeneous Distributions

Homogeneous distributions refer to mapping a set of computational tasks or data into resources of equal computational capacity. That means a task has the same duration in any possible computational resource. There are polynomial and exact algorithms to solve this problem if there is prior knowledge of tasks' time and no dependencies (KLEINBERG; TARDOS, 2006). In many applications, parallelism is achieved by concurrently processing the application domain data. Therefore, partitioning data is one way of distributing the load. For example, linear algebra problems, like matrix multiplication, LU, and Cholesky factorizations, rely on a matrix and structured data access. By distributing this matrix data, one distributes the computational tasks as well. The static partitioning of data can optimize several objectives, including improving load balancing across resources, reducing communications, and maintaining load balance across different iterations and data structures.

The *de facto* standard library for high-performance dense linear algebra routines over parallel distributed memory machines is ScaLAPACK (BLACKFORD et al., 1997), which provides efficient and scalable implementations for Cholesky, LU, and QR factorization. For simpler algorithms, like the matrix-matrix multiplication, where each output cell has the same amount of updates, the partitions of the matrix can be continuous and usually are 2D sub-matrices (blocks) to reduce communication. However, complex algorithms that may have various update iterations and a different number of accesses per block require more robust strategies. This subsection focuses on the LU factorization to explain 2D homogeneous distributions, as most other operations lead to similar parallelization challenges and need the same kind of strategies.

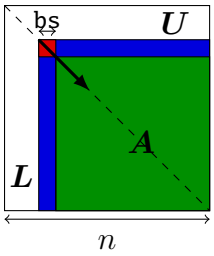
Figure 3.1 presents the blocked version of the LU factorization algorithm by relying on three LAPACK kernels: `dgetrf`, `dtrsm`, and `dgemm`. The first characteristic of this algorithm in terms of parallelization is that when nb is large, the two inner loops (i and j) surrounding the $(nb-k)^2$ `dgemm` constitute the significant part of computations and are fully parallel since every $A[i][j]$ should be updated independently from the others. In contrast, there are dependencies

between the different outer loop iterations. The second characteristic is that the portion of the matrix updated at each iteration of the outer loop gradually decreases. Therefore, one should ensure that every sub-matrix $A[k \ldots nb][k \ldots nb]$ is well distributed between the computation nodes and that all nodes can efficiently participate in each update. This necessity for cyclicality is further evident in Figure 3.2, which presents the access patterns over different iterations of a matrix with size $nb = 4$. Because the matrix area reduces during the algorithm progression, a given iteration's performance would benefit if its right-lower matrix is balanced across all the resources.

Figure 3.1 – The LU algorithm (left) without pivoting and the regions of A updated at iteration **k** (right)

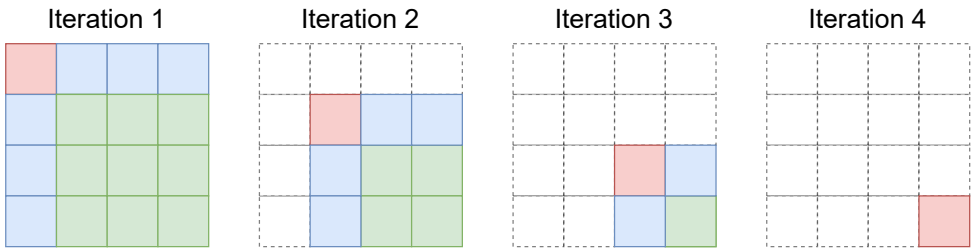
Algorithm 4: Task-based LU factorization

```
1  Input: nb Number of blocks
          n Number of Matrix cells
          A[1...n][1...n] Matrix divided and accessed
                          through nb × nb blocks
2  Output: A[1...n][1...n] Storing L and U
3  for k = 0; k < nb; k++ do
4      dgetrf-nopiv(RW, A[k][k]);
5      for i = k+1; i < nb; i++ do
6          dtrsm(RW, A[i][k], R, A[k][k]);
7          dtrsm(RW, A[k][i], R, A[k][k]);
8      end
9      for j = k+1; j < nb; j++ do
10         for i = k+1; i < nb; i++ do
11             dgemm(RW, A[i][j], R, A[i][k], R,
                 A[k][j]);
12         end
13     end
14 end
```



Source: The Author.

Figure 3.2 – Matrix access progress of the LU factorization tile-based algorithm

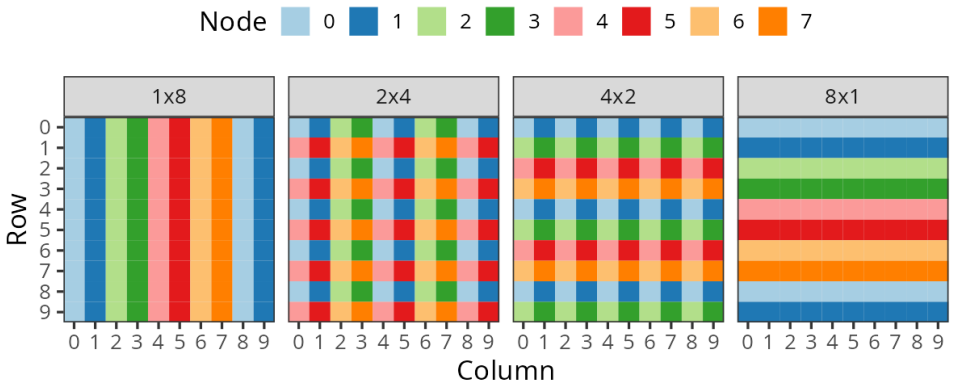


Source: The Author.

For homogeneous resources, there is a good understanding of the needs of this load balancing throughout the execution, employing a 2D block-cyclic distribution (BC for short).

Popularized by ScaLAPACK (BLACKFORD et al., 1997), BC takes advantage of the gradual decrease of sub-matrices to provide a good load balance. Two parameters, P and Q, are used to distribute the data and intercalate it into the resources. Figure 3.3 presents the matrix distribution using different values of P and Q for a complete matrix. In each case, the matrix is divided into a series of blocks (tiles), with the block's coordinate appearing on both axes. The color of the block represents the node it is assigned. The block-cyclic feature allows a regular balance (a pattern on any $P \times Q$ case) of columns and rows and hence of the sub-matrix $A[k \dots nb][k \dots nb]$ over the nodes. In contrast, the 2D feature ($P > 1$ and $Q > 1$) provides relatively low communication during the update (proportional to the square root of the total number of nodes) compared to a 1D distribution (where it would be proportional to the total number of nodes). There are also further studies of communication-avoiding algorithms relying on 3D and 2.5D data distributions (GRIGORI; DEMMEL; XIANG, 2008) that can reduce even further the communication amount by replicating data. However, they also come at the cost of higher memory usage and extra flop counts.

Figure 3.3 – Two-dimensional block-cyclic distribution with different PxQ values



Source: The Author.

There may be better distributions than the block-cyclic one, which focuses on cyclicality and blocks with many updates, for other operations present in the applications. The operation data access pattern, communication, and cost will determine the ideal distribution. As discussed, partitions (continuous divisions for each resource) for matrix-matrix multiplication are not directly applied to the LU or Cholesky factorization, as these algorithms require cyclicality that matrix-matrix multiplication does not need. Similarly, distributions can be tailored to specific operations, improving certain proprieties. For example, in the Cholesky case, some works consider the triangular shape and design a homogeneous cyclic distribution to reduce communication (BEAUMONT et al., 2022).

Another example that requires a tailored distribution is when the function applied on each block varies depending on the data. This condition is observed in sparse matrix operations (MILETTO, 2021) and algorithms using compressed matrices like Block Low Rank (BLR) (AMESTOY et al., 2015). Compression cases require specialized distributions, where some works propose a cyclical algorithm that considers the block operation's time (BEAUMONT; EYRAUD-DUBOIS; VÉRITÉ, 2020). Although all these distributions work very well on homogeneous resources, different distribution algorithms are required for heterogeneous environments.

3.1.2 Heterogeneous Distributions

The definition of the general problem of distributing an HPC parallel application using heterogeneous resources considers a set of resources (processing units) with different computer capabilities, speeds, or any other characteristic. Then, a problem is parallelized by simultaneously computing different parts of its data domain. The goal is to partition the data across the resources with a one-to-one mapping from one data entity to one computational unit. The amount of data assigned to each process should be relative to its capability (DONGARRA; LASTOVETSKY, 2006). In most cases, this partition may satisfy additional restrictions while minimizing a function, like application total makespan.

HPC Applications require different distribution strategies depending on their data structure and access patterns, considering the heterogeneity of the cluster (RACA; MEHOFER, 2020). Different applications' domains can take into account the heterogeneous case. The distribution of parallel loops into heterogeneous machines can use compiler time scheduling algorithms to handle the system heterogeneity (CIERNIAK; ZAKI; LI, 1997). Image processing can use a genetic algorithm to distribute applications (AALI; BAGHERZADEH, 2020). MapReduce operations can divide a 1D domain and map the partitions considering a dynamic calibrated measure of computer capabilities, equalizing the load between all computing nodes (FAN et al., 2012). Linear algebra kernels can use specialized distributions for heterogeneous resources (KALINOV; LASTOVETSKY, 2001; BEAUMONT et al., 2002b).

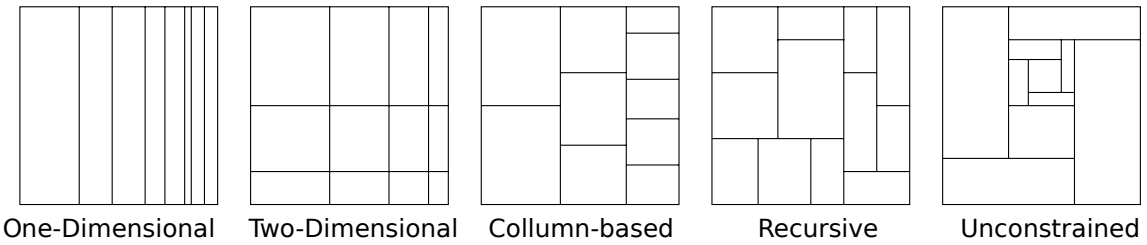
In linear algebra operations, 2D matrix partitioning is usually the main problem. The LU factorization is a great example to understand some of the work done in this problem with heterogeneous resources. Figure 3.1 (right) shows the $A[k \dots nb][k \dots nb]$ sub-matrix, which is updated with the product of the row and column `k` (computed by the `dtrsm`). When all nodes have different processing speeds, one should ensure that each node receives a fraction of the sub-

matrix, which is proportional to its speed. Nevertheless, since every node requires fragments of the row and column k to perform this update, this requirement induces communication with the nodes owning these blocks. One should thus ensure that the total amount of communications is not too large. This problem, known as the *Peri-Sum* problem, has been widely studied in the 2000s.

Peri-Sum: Given P nodes, let s_i denote the relative speed of nodes so that $\sum_i s_i = 1$. Partition the unit square into P rectangles R_i of dimension (h_i, v_i) so that $h_i \times v_i = s_i$ and $\sum_i h_i + v_i$ is minimal.

Kalinov and Lastovetsky (2001) first introduced this problem, and it was proven to be strong NP-hard in the general case (BEAUMONT et al., 2002b). Figure 3.4 illustrates particular classes of partitions for which a solution can be found. Restricting to 1D partitioning is trivial but leads to tall rectangles, producing terrible communication costs. Conversely, restricting to 2D partitioning leads to optimal communication costs but rarely allows optimal load balancing. Beaumont et al. (2001b) proposed an asymptotically optimal dynamic programming algorithm for column-based partitions. The resulting partition has the good property of grouping the faster nodes together (nodes are sorted by speed before being arranged in columns). Later, Nagamochi and Abe (2007) proposed a recursive splitting algorithm based on a Divide-and-Conquer approach.

Figure 3.4 – Taxonomy of unit-square partitions



Source: Beaumont et al. (2001a).

The problem can accept some restrictions to reduce its complexity. The cases for restricting the number of resources to three heterogeneous clusters or processors and finding the optimal shapes for the 2D matrix are present in Becker and Lastovetsky (2007) and DeFlumere and Lastovetsky (2014). Further, Patton et al. (2019) provides real experimental evaluations with the non-rectangular shapes for three resources. Additional investigation considers improving the communications functions over a homogeneous network for matrix-matrix multiplication, and the three resource cases (MALIK; LASTOVETSKY, 2020). Moreover, there are extensions for this work regarding the energy for those communications (MALIK; LASTOVETSKY, 2021). Beaumont et al. (2019) proposes two sub-optimal algorithms, Simple Non-Rectangular Recur-

sive Partitioning (SNRPP) and Non-Rectangular Recursive Partitioning (NRPP). The authors state that both algorithms work for any number of partitions. However, because of the uncertainty of the optimal shapes and ratios for four or more partitions, the authors only compare NRPP with the optimal approach for the two and three cases. The authors also state that the three-case complexity increases significantly compared to the two-case and believe that the same thing would occur for the four-case.

The problem for partitioning a matrix on high heterogeneous setups that contains multiple clusters where each cluster's node also has multiple devices is studied by Clarke et al. (2012). The authors propose a hierarchical way of first distributing inter-node and then intra-node. The work experiments applied the final distributions on matrix-matrix multiplication, where there is no need for cyclicality, with good results.

Another interesting case is when the speed capacities of the resources are not constant but presented as a continuous function of block sizes (CLARKE; LASTOVETSKY; RYCHKOV, 2012). Moreover, the makespan increase by size may not be linear, and architectural aspects, including resource contention, may influence it. The distribution problem is further expanded, and its solution involves optimizing the ideal kernel size when multiple heterogeneous resources exist. The authors in Clarke, Lastovetsky and Rychkov (2012) present an algorithm to generate a column-based distribution considering these continuous functions. Khaleghzadeh, Manumachu and Lastovetsky (2018) designed a branch-and-bound algorithm, HPOPTA, which receives functional performance models for the resources and computes a 1D distribution. It can also model some 2D cases that can be relaxed as 1D distributions. The authors study the algorithm using matrix-matrix multiplication and FFT and do not consider communication for this work.

Because of problems in the HPOPTA algorithm for a larger number of homogeneous machines, Khaleghzadeh, Manumachu and Lastovetsky (2020) expands the algorithm with a hierarchical version, HiPOPTA, that works for identical nodes comprising heterogeneous resources and a version, HiPOPTAX, designed for a wholly different set of nodes. The work is adapted for energy optimization, presenting HEOPTA (KHALEGHZADEH et al., 2020); and for a Bi-Objective of performance and energy, making HEPOPTA (KHALEGHZADEH et al., 2021) that considers the Pareto front. Again, the authors state that communication is out of the scope of these versions. The authors show that most optimal solutions are imbalanced over the resources' load in all algorithm versions; however, they provide a better makespan or objective function than the entirely balanced one.

A critical aspect of the distributions is the total communication they cause. Rico-Gallego, Lastovetsky and DÍaz-MartÍn (2017) adapted a communication cost model (τ-Lop)

from homogeneous settings to heterogeneous ones considering synchronous applications, where the communication occurs in phases and starts simultaneously. They evaluate the model in the matrix-matrix multiplications and a 2D wave equation solver. However, because of τ-Lop decouple from computation, its applicability on asynchronous and overlapping communication and computation environments is limited. Rico-Gallego et al. (2020) adapted Beaumont et al. (2001b) column-based algorithm for matrix-matrix multiplication using τ-Lop instead of the sum of half perimeters in the same conditions as above. Another work proposes a tool to construct and evaluate the τ-Lop expressions' communication cost (RICO-GALLEGO et al., 2019).

The distribution can also be done dynamically using a runtime scheduler. Beaumont et al. (2015) compares static, dynamic, and hybrid approaches. The authors show that static distributions for matrix-matrix multiplication can greatly reduce communication, and dynamic scheduling consistently provides a lower makespan while increasing communication. The hybrid approach's usage presents both worlds' benefits, indicating that the general hybrid approach offers advantages over the pure static or pure dynamic.

Although a lot of work focuses on correctly producing the partitions with continuous areas, many times using the Peri-sum problem for optimizing communications, much work still can be done to distribute these partitions in a discrete setting. Moreover, some algorithms, including the LU and Cholesky factorizations, would benefit from a distribution optimized to be balanced at all iterations, with cyclicality, not necessarily using a partition computed for matrix multiplication that would only be valid for iteration one (the only that uses the whole matrix).

Partitions that allow for processing a single update operation efficiently are thus available. However, a correct shuffling of columns and rows is required to obtain a proper load balancing throughout the whole execution of the LU algorithm. Beaumont et al. (2001a) proposed a simple shuffling procedure (1D-1D), which is asymptotically optimal, regardless of the initial rectangle partition, and which is briefly described below.

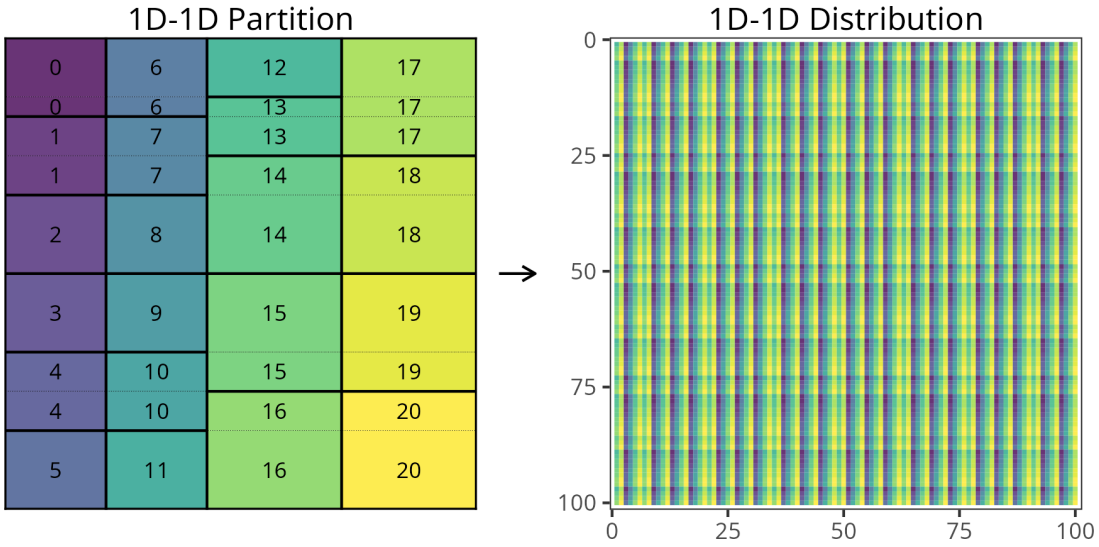
Consider a 1D partition (a division for only one dimension). The optimal allocation A of columns to nodes can be built through Algorithm 5. It consists of greedily selecting the processor that minimizes the processing time of the sub-matrix $A[k \dots nb][k \dots nb]$. This algorithm produces an optimal allocation for every $k \in \{1, \dots, nb\}$, and hence optimal overall. An almost optimal distribution can quickly be built from arbitrary partitions by extending processor boundaries (Figure 3.5 left) and applying the previous algorithm for 1D partitions independently to virtual rows and virtual columns. The resulting distribution (Figure 3.5 right) is no more optimal for every $k \in \{1, \dots, nb\}$. However, it is asymptotically optimal, i.e., its

execution time is at most $1 + O(1/nb)$ times the one achieved by an ideal distribution.

Algorithm 5: Shuffling a 1D partition into a 1D distribution

```
1 (c_1, ..., c_P) = (0, ..., 0)
2 for k = nb down to 1 do
3 |   p = arg min_{1≤j≤P} (c_j+1)/s_j
4 |   A[k] = p
5 |   c_p = c_p + 1
6 end
```

Figure 3.5 – The 1D-1D partition (left) and the reciprocal 1D-1D distribution for 14 slow and 7 fast nodes (total of 21 nodes)



Source: The Author.

So far, most of the presented heterogeneous strategies focus on a partition that could not be applied directly to the LU or Cholesky algorithms because they are not cyclical. Our work relies on state-of-the-art partitionings but focuses on distributing these partitions into discrete cyclic distributions, also considering multi-phase applications when multiple distributions should exist. In Chapter 5, we propose two algorithms to balance the linear algebra operations that require it at any iteration considering the ratio of communication and load. The first algorithm converts a continuous partition into a discrete cyclic distribution for factorizations, extending the 1D-1D algorithm. The goal of the first algorithm is to gradually restrict the end of the distribution to reduce communication on parts of the DAG that do not have enough parallelism by using only the fastest nodes. This method also helps on the critical path. The second algorithm is a general load-balancing distribution algorithm for any distribution to increase balance in favor of communication, using load computed from the application DAG.

Moreover, all examined works here focus on partitioning and distributing a matrix considering one operation. That is not the case for multi-phase applications that Chapter 6 discusses. The proposed solutions show that these applications require extra considerations when distributing data. Also, all works consider a partition using all machines. Chapter 7 discusses and demonstrates that sometimes the application can drop the utilization of some computational nodes in favor of less communication and better critical path executions.

3.2 Heterogeneous Dynamic Load Balancing Algorithms and Strategies

For this work, the definition of dynamic load balancing is a strategy that changes, during the application execution, the computation load of the processing units through moving data or processing elements (tasks, for example). Usually, these strategies recognize that the initial distribution is wrong because of incorrectly assuming resources' computational power, processing times of the application elements (including irregularity), or other performance issues like communication. These reasons also include node performance uncertainty, as if it is highly present on the target machines, online adjustments may be necessary (PEI et al., 2019). The works presented in this Section give perspective on dynamic load balancing and the differences with computing distributions for asynchronous task-based applications.

One common approach is to measure the time between two synchronization points and try to redivide the load to balance it. Resources that were slower to reach the synchronous barrier should yield some work. Some strategies rely on dynamic programming algorithms to load balancing on multi-GPU systems (ACOSTA; BLANCO; ALMEIDA, 2012). Others use a threshold of difference between the lower and highest loaded to trigger a load balancing (ACOSTA et al., 2010). In Cabrera et al. (2020), the authors develop a multi-objective (for example, energy and performance) heuristic algorithm for heterogeneous clusters. This strategy requires that the application have a gathering phase where work will be re-balanced following the established criteria, considering the chosen metric for solving a problem of a given size. BeltrÁn and GuzmÁn (2009) presents a method to balance independent processes (non-communicant) at heterogeneous nodes using the four stages of load balancing of Xu and Lau (1996). Other strategies (CLARKE; LASTOVETSKY; RYCHKOV, 2011; MANUMACHU; LASTOVETSKY, 2019) rely on the adaptive construction of the speed or energy functions concerning the problem size to balance the load.

Nevertheless, these approaches disregard communication and always try to use all the resources, endeavoring that all resources' computational time should be equal. As a result,

these solutions may present performance problems in communication-bound applications. The amount of data transfer, even intra-node, could cause contention and slow down the system, even if the load is correctly balanced across the resources. Possible solutions are changing the algorithm to reduce communication (DONGARRA et al., 2017; BALLARD et al., 2011; MOHIYUDDIN et al., 2009) or following an execution flow that enables more data-reuse (HERAULT et al., 2019).

Perfect load balancing may not be the best situation, as imbalanced loads can lead to better performance if they improve other behavior aspects. One concern is when the environment has considerable performance variability in some computational resources. In iterative applications where one resource should not lag the whole execution, one proposed approach penalizes and reduces the load on high variance resources, favoring reliability (YANG; SCHOPF; FOSTER, 2003). This approach does not necessarily divide the load equally but results in an improvement in the overall performance.

Like the distribution strategies, dynamic load balancing can be tailored to the application and requires analyzing its characteristics. For example, a case for a CFD application that the load-balancing techniques use domain-specific information (BORRELL et al., 2020). Another example is the Ondes3D application (DUPROS et al., 2010), which may present an irregular imbalance because of the algorithm particularities and case study (TESSER et al., 2017). Both require the study of a particular behavior to design load-balancing techniques.

One popular runtime for dynamic load balancing is Charm++ (KALE; KRISHNAN, 1993), and its extension for MPI applications AMPI (KALE; ZHENG, 2009). Charm++ is based on the migratable-objects programming model and the actor execution model, where the application is decomposed into computational units called chares. Its ancestor, Charm (FENTON et al., 1991), was studied in the context of heterogeneous low-cost workstations shared in a multi-user environment subjected to variable load (SALETORE; JACOB; PADALA, 1994). The authors propose a load-balancing technique based on load migration by forecasting the makespan, which considers the processing capacity. The under-loaded processing units demand extra work if they fall below a metric threshold. Other works show that topology-aware mapping in Charm++ of an application can reduce contention (BHATELÉ; BOHM; KALÉ, 2011). Also, load-balancers that model the topology of the cluster intro tree-structures topologies are proposed (JEANNOT et al., 2013). In heterogeneous settings, Gammel et al. (2017) shows some of the available algorithms provide worse results than no balancing. Because of scalability concerns, some works propose load balancers that avoid centralized information for iterative applications and homogeneous processors (MENON; KALÉ, 2013). Charm++ was extended to use GPUs

with G-Charm. It can use a static CPU and GPU chares partition or provide one simple dynamic scheduling. In the dynamic approach, a chare is assigned to a GPU if its CPU time is greater than the combined GPU time and movement (VASUDEVAN et al., 2013), similar to HEFT.One popular runtime for dynamic load balancing is Charm++ (KALE; KRISHNAN, 1993), and its extension for MPI applications AMPI (KALE; ZHENG, 2009). Charm++ is based on the migratable-objects programming model and the actor execution model, where the application is decomposed into computational units called chares. Its ancestor, Charm (FENTON et al., 1991), was studied in the context of heterogeneous low-cost workstations shared in a multi-user environment subjected to variable load (SALETORE; JACOB; PADALA, 1994). The authors propose a load-balancing technique based on load migration by forecasting the makespan, which considers the processing capacity. The under-loaded processing units demand extra work if they fall below a metric threshold. Other works show that topology-aware mapping in Charm++ of an application can reduce contention (BHATELÉ; BOHM; KALÉ, 2011). Also, load-balancers that model the topology of the cluster intro tree-structures topologies are proposed (JEANNOT et al., 2013). In heterogeneous settings, Gammel et al. (2017) shows some of the available algorithms provide worse results than no balancing. Because of scalability concerns, some works propose load balancers that avoid centralized information for iterative applications and homogeneous processors (MENON; KALÉ, 2013). Charm++ was extended to use GPUs with G-Charm. It can use a static CPU and GPU chares partition or provide one simple dynamic scheduling. In the dynamic approach, a chare is assigned to a GPU if its CPU time is greater than the combined GPU time and movement (VASUDEVAN et al., 2013), similar to HEFT.

Most of the described strategies require an iterative application and measure imbalance in one iteration to redistribute the load for the next. When considering multi-phase asynchronous applications, such strategies would not work over overlapping phases, as they require a checkpoint. We show that static distributions designed to deal with such multi-phase context work well in situations that have regular task performance (Chapter 6). When considering possible application adjustments, we also consider that yielding some resources for particular phases is beneficial. Also, instead of using thresholds of imbalance, we train a reinforcement learning method that is a surrogate for inferring the overall application behavior. The surrogate guides the decisions of which best heterogeneous resources to use (Chapter 7).

3.3 Multi-Distributions and Redistribution on Multi-Phase Applications

Applications with multiple operations may use a distinct distribution for each, leading to a data redistribution between them. These multi-distributions with redistribution are necessary because each phase's best load division is probably different (WALKER; OTTO, 1996). Prylli and Tourancheau (1996) proposed algorithms to reduce the scheduling redistribution cost between block-cyclic distributions with different block sizes used in various application phases. Many works consider the problem over homogeneous scenarios with different application characteristics. These works study, for example, situations where the granularity of the block-cyclic distribution should change across application phases (LIM; BHAT; PRASANNA, 1996; PARK; PRASANNA; RAGHAVENDRA, 1999), as each phase may admit a better granularity. Irregular applications may require better balancing over algorithms that divide data into a grid (HOFMANN; RÜNGER, 2018). Some works consider the communication redistribution phase of data moving between two distinct clusters (JEANNOT; WAGNER, 2006).

All these works consider the problem of redistribution as discovering the communications patterns that minimize the transition duration from a given origin distribution to a target distribution. These solutions assume a given origin and target distribution and focus on reducing communication time because of contention. The work Li and Yu (2019) considers irregular distributions and uses bi-planar graphs to formulate three different coloring-based problems. They propose an approximation algorithm to solve one of these problems, which translated into a reduced communication redistribution.

Moreover, with MPI, such data redistribution typically occurs synchronously between the different phases, which could be more efficient. This situation is no longer an option with the scale of supercomputers, as a synchronization point will hold too many resources. Hence the rising popularity of the data flow algorithms that minimize the number of synchronization points (DONGARRA et al., 2017). The redistribution problem focusing on the flexibility of tiles proprieties (such as size and formats) of irregular data is studied in the context of the task-based applications using ParSEC (CAO et al., 2020), assuming a given source and target distribution.

Some linear algebra solvers, like Chameleon, leverage this asynchronous capability by overlapping different tasks and iterations and exploiting more parallelism (AGULLO et al., 2010; GATES et al., 2019). A possible multi-phase strategy consists of finding the best distribution per phase while minimizing the overall communication cost when changing them, which can be challenging. Some works point out that, even in simple situations, the target distribution may present many possible permutations corresponding to similar homogeneous

nodes (HERAULT et al., 2014; HERRMANN et al., 2016). The problem is further expanded to minimize communication while generating another distribution that minimizes computation when only given the source or target distribution. In the homogeneous case, one approach is searching for a permutation of the ideal minimal computation distribution that minimizes communication. Finding the ideal distribution (of the many permutations) while minimizing the computation and the total communication cost is NP-hard (HERRMANN et al., 2016).

In this situation of minimizing computation and communication, the problem of computing distributions for different phases when considering heterogeneous resources should consider the transition from one to the other. Minimizing the communication between two heterogeneous distributions and the following computation is desirable. However, there is a gap in the literature when considering heterogeneous resources with asynchronous task-based applications. This situation is a relevant problem when studying heterogeneous distributions on multi-phase applications that will be addressed in Chapter 6.

3.4 Reinforcement Learning for optimizing HPC behavior

The number of parameters and possible configurations in HPC applications that can impact performance is enormous. Applications can have different algorithms, heuristic parameters, and problem divisions. Also, runtimes can usually employ distinct schedulers or have customized configurations while using heterogeneous infrastructures. When a developer faces so many options, it is desirable to have strategies to automatically discover the best configurations during execution and reduce all the developer's burden. However, finding the best configurations may involve evaluating time-expensive cases by executing the application repeatedly throughout the search space. More intelligent strategies would prune the search space or construct surrogates for unknown functions, reducing the cost of discovering the best configurations. Bruel (2021) conducted studies presenting many techniques and strategies for modeling surrogates for the HPC's configuration parameters tuning search spaces.

Autotuning can be used to select code variants and parameter configurations in different architectures (BALAPRAKASH et al., 2018). Moreover, Silvano et al. (2018) presents a toolbox for autotuning considering Energy Efficient. Other works include constructing surrogate models using Bayesian Optimization for finding parameters (MENON; BHATELE; GAMBLIN, 2020). Autotuning was also applied to the Chameleon linear algebra library, and StarPU (AGULLO et al., 2020). The authors analyze parameters, including tile size and the number of intra-node resources, by extensively or partially executing the search space possibilities before the

execution. Also, these authors augmented the search space results with simulation.

Some works study the problem of optimizing the best number of nodes for an HPC application. Applications such as Deep Neural Networks may present many challenges when strong scaling on homogeneous nodes (KEUPER; PREUNDT, 2016). The issues range from communication overhead to the number of parallel operations. The number of choices can also be essential in performance and energy. In some applications, it is possible to use more homogeneous nodes with slower frequencies and different voltages and improve performance while reducing energy consumption (FREEH et al., 2007).

A study for HPC checkpoint fault-tolerant environments (JIN et al., 2010) shows convex-like curves for makespan as a function of the number of homogeneous processes. They use Newton's Method to find the best number of processes to use. In another work, the authors used the Gaussian process to search for the best homogeneous cloud instance for a given application considering performance and cost (ROSARIO et al., 2021). They restrict to particular numbers (powers of 2 from 1 to 32) of homogeneous nodes. In all of these works, the focus was on using homogeneous resources.

This thesis explores the usage of reinforcement learning to optimize during the execution of an iterative application the ideal number of nodes to use on a phase (Chapter 7). This problem arises as precisely modeling communication or other unforeseen behaviors is complex, especially in the heterogeneous scenario. Using real execution behavior to construct a surrogate iteratively would aid the decision, as the application would learn how it behaves in different configurations, ideally without exploring them all.

3.5 Contributions opportunities

The problems of (i) distributing an operation on heterogeneous resources; (ii) multiple distributions and redistributions for multi-phase applications; and (iii) learning and adapting for unexpected behavior like node scalability; are mostly studied with homogeneous nodes and separately in the literature, missing an opportunity to combine them all. This work shows that these problems are challenges when distributing task-based applications on system-level heterogeneous resources. Therefore, one must solve all these problems to improve the application's performance. This thesis presents its load distribution contributions from a micro to a macro perspective. It starts by computing one operation distribution (Chapter 5) that will be later used when studying multiple distributions in multiple operations (Chapter 6). Then, such strategies will be again employed when tuning the number of nodes to use per phase (Chapter 7).

4 ANALYSIS AND EXPERIMENTAL METHODS

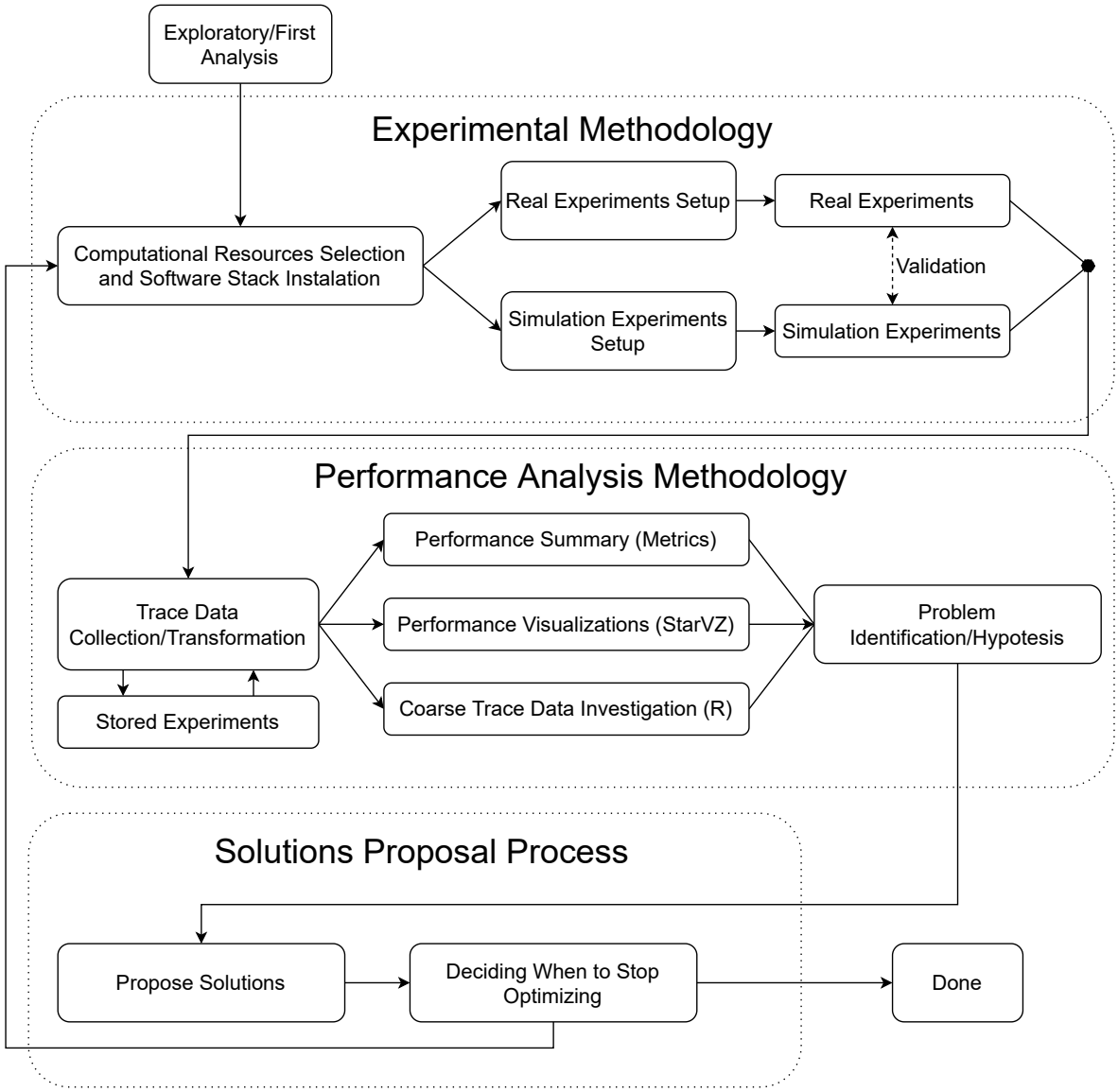
This Chapter describes the collection of this work‘s experimental and analysis methods. Figure 4.1 summarizes the general workflow employed, consisting of three stages. The first stage (Experimental Methodology) is conducting experiments in controlled environments, using real and simulation executions. The execution generates traces that describe in detail each experiment performed. The next stage (Performance Analysis Methodology) is the analysis of these traces using metrics, visualizations, and data science tools. This analysis leads to the final stage, the proposal of solutions and methods to improve behavior. A restart of the methodology occurs with new optimization candidates, the loop in the figure. The strategies offered in this thesis reflect this iterative circle of experimenting, analyzing, and proposing solutions.

The following sections will explain the experimental and analysis steps in detail, reflecting all the boxes in Figure 4.1. Section 4.1 presents the experimental methodology, listing the complete environment of physical machines with the software stack, including all custom configurations for tuning the experiments. Moreover, it explains the experimental methodology when using simulations with StarPU+Simgrid, a powerful tool to test new scenarios easily. Section 4.2 describes the analyses of task-based applications, the methodologies to collect execution information, metrics to give a fast overview of the application performance, pre-defined visualization techniques with the StarVZ package, and finally, traces data analysis. Ultimately, Section 4.3 discusses the step of proposing solutions and the methodology repetition.

4.1 Experimental Methodology

The experiments of this work are divided into two categories: real executions and simulations. In the former, a real platform directly executes the application. The program effectively computes and accomplishes its purpose, reaching an actual and correct algorithm solution. Differently, the simulation case may replace some computation with statistical models, focusing on approximating the performance behavior and not the final algorithm solution. This work uses Simgrid (CASANOVA et al., 2014) as the framework to simulate distributed HPC platforms, enabled in StarPU with its StarPU+Simgrid module (STANISIC et al., 2015b). StarPU+Simgrid generates traces that faithfully approximate the application behavior and resource utilization (STANISIC et al., 2015a). However, StarPU+Simgrid does not perform computations (tasks) and can replace data with dummy information. The remaining of this Section explains the resources, the real experiments, and the simulation ones.

Figure 4.1 – Summary of the analysis and experimental methodology



Source: The Author.

4.1.1 Computational Resources and Software Stack

The real and simulation experiments use resources from the High-Performance Computing Park (PCAD) infrastructure[1], the Grid5000 (G5K) testbed[2], and the SDumont (SD) supercomputer[3]. Table 4.1 shows the machines used in the experiments. The network of PCAD is a 1Gb/s Ethernet. The network for Chetemi and Chifflet Grid5000 resources is a 10Gb/s Ethernet, while for Chifflot and Chiclet, it is a 25Gb/s Ethernet. A 2x100Gb/s Ethernet network connects both partitions. In SDumont, the base partition has an Infiniband FDR 56Gb/s network.

[1] <http://gppd-hpc.inf.ufrgs.br/>
[2] <https://www.grid5000.fr/>
[3] <https://sdumont.lncc.br/>

Table 4.1 – Compute nodes available for experiments

Infrastructure	Machine	CPU	Memory	GPU
Grid5000	Chetemi (Che)	2× Intel Xeon E5-2630 v4	256 GiB	–
Grid5000	Chifflet (Chi)	2× Intel Xeon E5-2680 v4	768 GiB	2× GTX 1080
Grid5000	Chifflot (Cho)	2× Intel Xeon Gold 6126	192 GiB	2× Tesla P100
Grid5000	Chiclet (Cle)	2× AMD EPYC 7301	128 GiB	–
Grid5000	Troll (Tro)	2× Intel Xeon Gold 5218	384 GiB	–
PCAD	Hype	2× Intel Xeon E5-2650 v3	128 GiB	2× Tesla K80
PCAD	Tupi	Intel Xeon E5-2620 v4	80 GiB	2× GTX 1080
SDumont	B715 (SDC)	2× Xeon E5-2695v2	64 Gib	–
SDumont	B715 (SDG)	2× Xeon E5-2695v2	64 Gib	2× K40

Source: The Author.

All experiments use the StarPU version 1.3 development branch, and experimental groups use the same commit (usually per chapter). However, because of the iterative process of this work, including contributions proposed to software, the commits may be different between experimental groups. The specific software information is present explicitly in each experimental evaluation section.

4.1.2 Real Experiments Setup

The experiments use the following configurations to control the environment: (a) Intel hyper-threading off; (b) performance frequency governor; (c) Maximum frequency set per core; (d) NVIDIA GPUs set to persistence mode and maximum frequency when possible. The following configuration for G5K: (e) network cards with an MTU of 9000; (f) network interruptions only to cores of the same network card's NUMA node; (g) the `sysctl.conf` configurations present in Figure 4.2 for tuning network configurations and disabling the kernel NUMA memory balancing; (h) NTP synchronized before experiments and turned off to not cause any interference. After some study, all the experiments converged to using the NewMadeline MPI and the respective StarPU MPI backend for real experiments.

The experiments mainly use the StarPU's `DMDAS` scheduler with two reserved CPU cores: one for the MPI thread and the other for the application thread responsible for task submissions; unless stated otherwise for specific experiments. The experiments bound the MPI and GPU workers' threads to the cores belonging to the NUMA nodes of the hardware resource (NIC or GPU). StarPU configuration includes a trace buffer size of 1.5GiB to minimize the flush of traces. However, not all experiments collect traces, and when only the final makespan is of interest,

Figure 4.2 – The sysctl.conf configuration used for G5K experiments

```
net.ipv4.tcp_rmem = 4096 87380 1147483647
net.ipv4.tcp_wmem = 4096 65536 1147483647

net.core.rmem_max = 1147483647
net.core.wmem_max = 1147483647

net.core.netdev_max_backlog = 250000
net.core.rmem_default = 16777216
net.core.wmem_default = 16777216

net.core.busy_poll = 50
net.core.busy_read = 50

kernel.numa_balancing = 0
net.ipv4.tcp_mem = 16777216 16777216 16777216
net.ipv4.tcp_slow_start_after_idle=0
net.ipv4.tcp_mtu_probing=1
```

Source: The Author.

trace generation is disabled. The following parameters are set: `STARPU_MPI_COOP_SENDS=0` to disable the experimental cooperative MPI sends, `STARPU_NWORKER_PER_CUDA=1` to enable only one worker per CUDA device, `STARPU_CALIBRATE=1` to update the performance models continuously. One of our works also studies other StarPU and platform configuration options in these experimental scenarios (NESI; SCHNORR, 2020), including workers' bindings to cores and StarPU NUMA configurations.

4.1.3 Simulation Setup and Evaluation

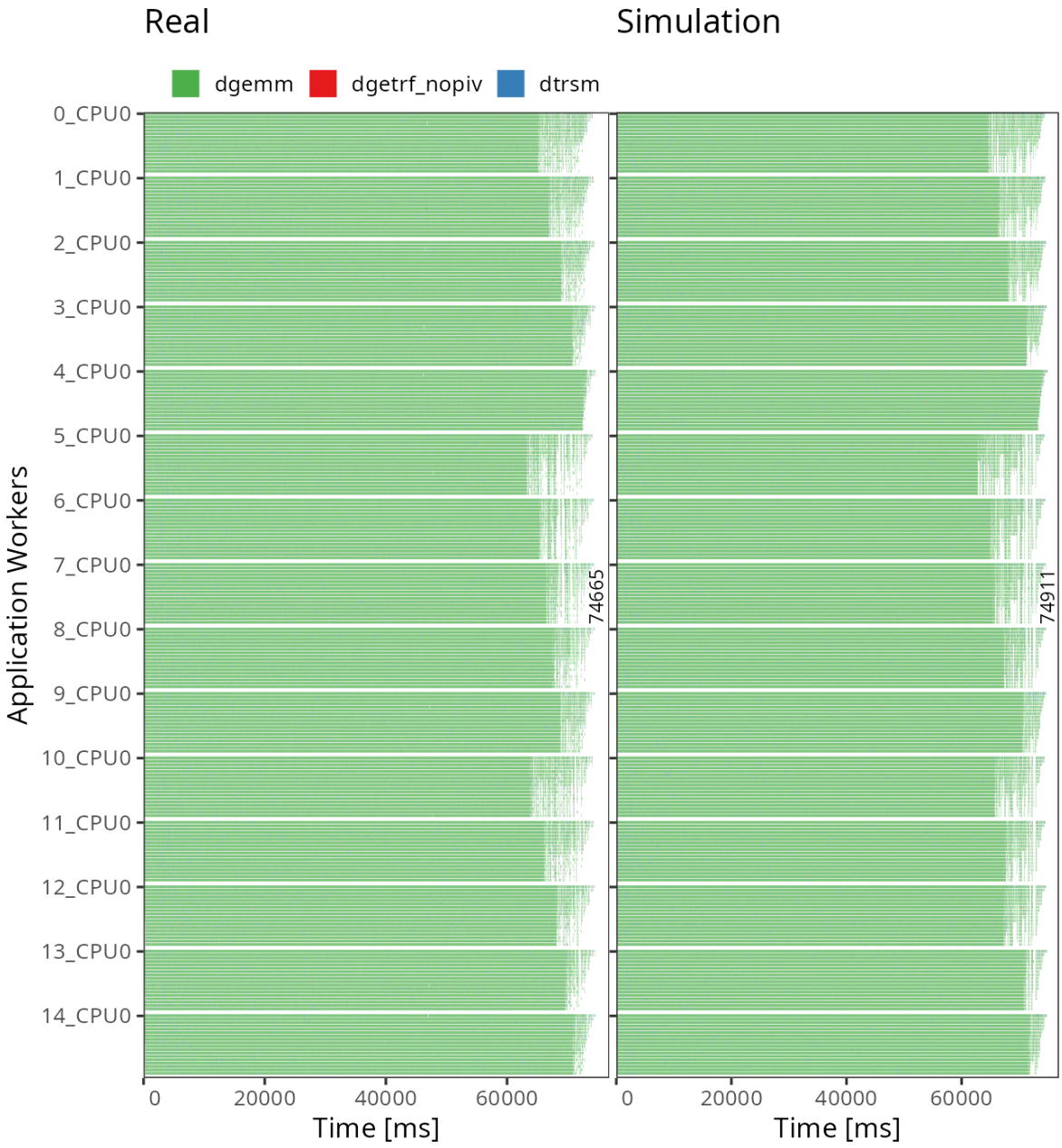
The simulation experiments use StarPU with Simgrid, where a real machine emulates the behavior of another platform using task and network models. Because the goal is to simulate the overall application, it is unnecessary to execute tasks if StarPU already has a task duration model for each platform. Although resulting elements like makespan, resource usage, and scheduling are similar to the real case, the simulation does not provide the actual application numeric solution.

The distributed version of StarPU+Simgrid uses the Simgrid MPI simulator `smpi`. Only one process exists in `smpi` that executes all MPI processes; in StarPU default behavior, one MPI process per computational node. Many works (STANISIC et al., 2015a; STANISIC et al., 2015b) show the precision of StarPU+Simgrid. However, this work double-checks the accuracy

of StarPU+Simgrid in its cases. An example is shown below.

The usage of simulations depends on their accuracy against the real counterpart. A first validation of the simulation executions of Chameleon/StarPU-MPI over homogeneous and heterogeneous clusters is performed using the state-of-the-art block-cyclic and 1D-1D distributions (described in Section 3.1). Figure 4.3 depicts the behavior of two runs considering the 15 Chetemi nodes, first in reality (left part) and then in simulation (right).

Figure 4.3 – The 3×5 BC partitioning using 15 identical CPU-only nodes to compare the behavior of a real execution (left) against the simulation (right)



Source: The Author.

Figure 4.3 presents the Application Workers panel, a Gantt Chart with individual resources (CPU workers in this case) on its Y-axis. The Y-axis labels highlight the nodes (0 are

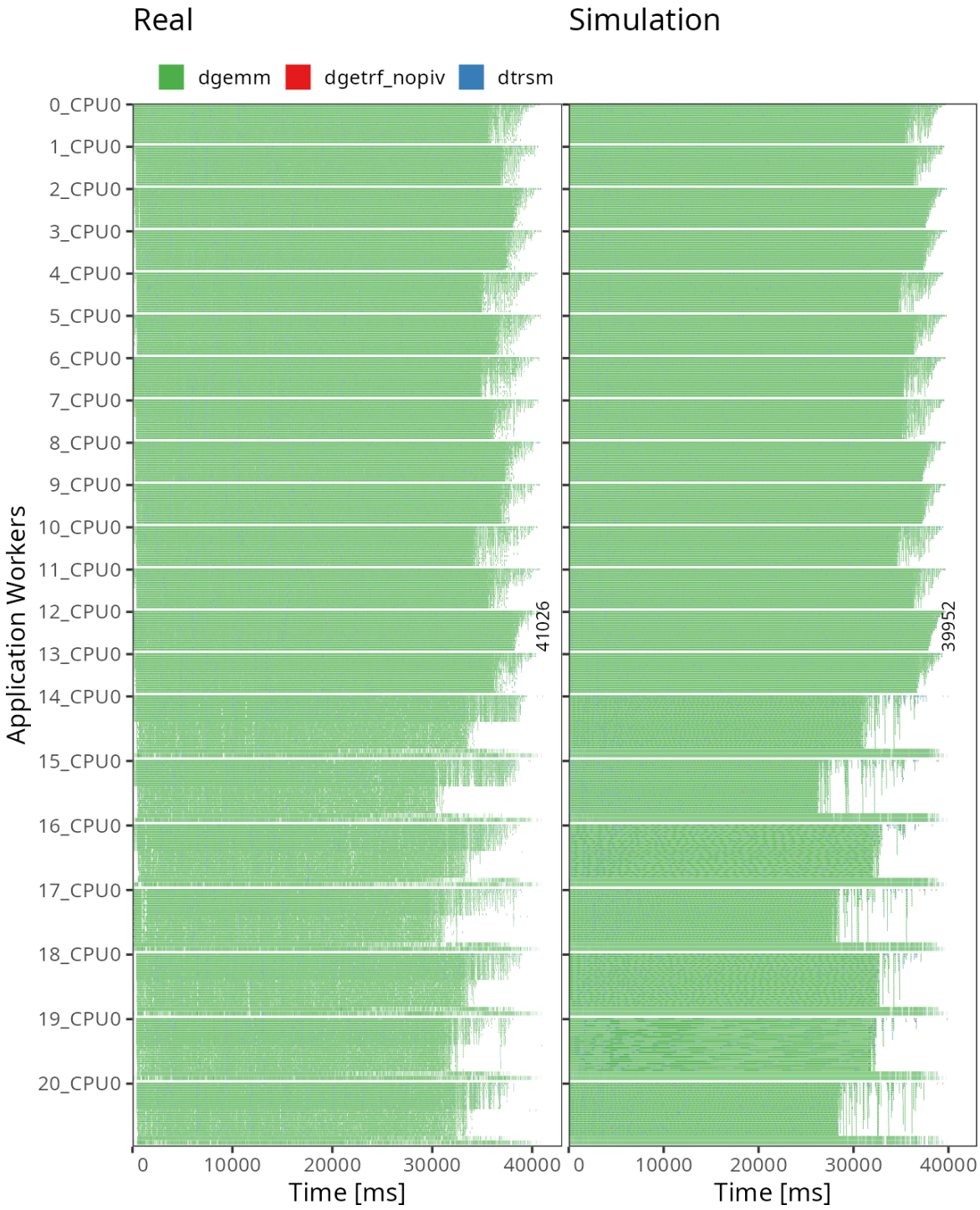
resources of node 0, and so on). Each state (rectangle) represents a task, where the color is the task type. An empirical visualization analysis of these plots indicates that both executions are very similar. First, the makespan presents a difference of 246ms out of a 74s execution (0.3%), on which the simulation is longer. This slightly longer result might come from the performance models if the `dgemms`' mean of the real execution got a smaller negative variability than the simulation's performance models.

Figure 4.4 depicts a second example where the behavior of another two runs considering a 7+14 node platform (7 fast Chifflet nodes with GPUs and 14 slower Chetemi nodes) in a real configuration (left) and in simulation (right). The Figure presents the same Gantt Chart as already described for Figure 4.3. The difference is that the last seven machines (from id 14 to 20) have 2× GPUs, each depicted as taller resources on the Y-axis at the end of the node's vertical list of resources. This case uses the 1D-1D, which considers such heterogeneity.

Similar to the other case, the simulation faithfully approximates the real execution. This case with GPUs, though, presents more differences. First, the makespan difference is 1074ms out of a 40s execution (2%), with the simulation being the optimistic version. In nodes with GPUs, there is a difference in how tasks are scheduled on the execution's end. While there is a more progressive reduction of CPUs usage in the real version with more idle times in GPUs, the simulated version preferred to schedule all those remaining last tasks in GPUs, keeping the CPUs idle. Although some difference exists, the simulations present a faithful and cheap way to verify the performance of the configurations. It is faithful because proprieties are correctly simulated, like nodes with the lower loads are the same in both executions, like node 15, which has lower usage. It is cheap because while real execution requires the reservation of very powerful nodes, simulation only requires one node without expensive and powerful accelerators like GPUs. These experiments show that simulation is sufficiently faithful to conduct studies as it captures essential details of load imbalance and approximates the makespan. During this work, we performed many experiments that continued to show the validation of the simulation with respect to real experiments.

Moreover, the usage of `smpi gdb` debug approach is much easier because it has only one process (with complete memory access to all simulated processes). Another advantage of debugging with simulation is that StarPU-Simgrid scheduling is deterministic, so stochastic scheduling behaviors of real execution are absent, and the execution will be the same for the same input parameters. Of course, some experimental situations will require real executions, but many performance-related what-if scenarios could be discarded upfront, reducing the burden and amount of cases for real executions.

Figure 4.4 – The 1D-1D partitioning using 14 CPU-only nodes plus 7 GPU-equipped nodes to compare the behavior of a real execution (left) against the simulation (right)



Source: The Author.

If simulation performs much better than real life, one can infer that an external problem from StarPU and the application is impacting the overall performance. One possible analysis approach is to use simulation as the best-case scenario, as a lower bound, to detect external problems from the runtime and application. The analyst could compute divergences of the

real execution from the simulation and understand why the simulation model does not capture it. Moreover, instead of approximating simulation to reality, one could approximate reality to simulation. Understanding simulation divergences may indicate a real problem that can be solved, not necessarily an unfaithful simulation model.

We tried to approximate real executions to simulation many times, as it emulates a real controlled scenario that, most times, is impossible to achieve in the noisy and problematic reality. The conclusions taken from this approximation will, of course, improve the real experiments. For example, we study many NUMA and communication parameters impacting StarPU (NESI; SCHNORR, 2020). The divergence between real and simulation experiments motivated this study, as the simulation was way better than real life. In the end, we discover that the MPI middleware and some NUMA hardware contention (that simulation does not model) were the reason for the bad results. Changing the MPI layer and correctly configuring the NUMA-related configurations made the real execution match the simulation performance. Without this powerful simulation tool, one could miss such optimizations because the analyst might never know the best case and behavior possible.

4.2 Performance Analysis Methodology

This Section presents the analysis process to identify problems in the task-based applications. The foundation for the analysis methodology is execution traces and complementary data that the runtime and the application collect. The following sections describe the data collection, computation of performance metrics, application performance behavior visualization, and direct data investigation.

4.2.1 Trace data collection and transformation

The StarPU trace system generates trace events by placing hardcoded functions strategically in the runtime source code. If the necessity to trace a new event emerges, it is required to edit the runtime code and add the designated trace-related macros. StarPU will save the execution trace for each compute node locally in FxT[4] files. It is possible to select only related events when aiming for tracing specific elements, reducing trace file size. After the trace generation, an internal StarPU tool (`starpu_fxt_tool`) can convert these FxT files into Paje ones. Paje is a

[4] <https://savannah.nongnu.org/projects/fkt>

structured and widely used trace file format that an analyst can directly investigate, analyze individual events, and apply specific procedures (SCHNORR; STEIN; KERGOMMEAUX, 2013). Instead of using the crude Paje trace data, this work uses a specialized auxiliary framework for task-based applications, StarVZ (GARCIA PINTO et al., 2018; NESI et al., 2019; PINTO et al., 2021), maintained on our research group[5], that process the data and generates visualizations.

StarVZ is an R CRAN available package[6] that operates on StarPU paje files to generate R tibble data.frames[7], and visualizations (GARCIA PINTO et al., 2018). The work of this thesis incorporated many elements of the methodology on the package, making many contributions to StarVZ. The analysis process using StarVZ consists of running the workflow over the FxT files to generate R-friendly data and generate visualizations. StarVZ comprises two phases: (i) data cleaning that generates intermediate trace data into R tibbles and (ii) the generation of pre-defined visualizations. After producing these intermediate data, one can compute metrics, generate custom visualizations, or work directly with them.

4.2.2 Performance metrics

One way of quickly obtaining an overview of the execution is to employ global metrics, which can estimate the makespan and be a lower bound connected to some proprieties. These metrics will summarize the whole application execution into one number that an analyst can compare against the real makespan. Depending on the algorithm, these metrics' distance from the actual execution can indicate possible problems. One of these metrics is Critical Path Bound (CPB), which is the total amount of time in the critical path (WILSON, 1979) of the DAG considering that particular execution. The critical path bound uses information of one particular execution because of the scheduling decisions regarding the task to resource placement. For example, if a given path runs entirely on CPUs, it is probably the critical path (takes longer) compared to a path with the same number of tasks that use GPUs. First, it is necessary to compute the latest parent of a task, which is the last dependency executed, given by

$$\text{Latest Parent}(t) = \underset{p \in \text{parents}(t)}{argmax}\ \text{TE}(p) \tag{4.1}$$

where $\text{TE}(p)$ is the ending time of task p. The recursive computation of the graph path comprised of the latest parents from the last task to the first one is the critical path. The critical path bound

[5] <https://github.com/schnorr/starvz>

[6] <https://cran.r-project.org/package=starvz>

[7] Tibble is a form of data table from the R package tibble (<https://tibble.tidyverse.org/>) part of tidyverse

is merely the summation of the task's duration in the critical path given by

$$\text{CPB} = \sum_{t \in \text{critical path}} \text{TD}(t) \tag{4.2}$$

where $\text{TD}(t)$ is the task duration of task t. If CPB is closer to the real makespan, but there are still idle resources, this probably indicates that there are not enough tasks to run concurrently. One possible solution is to change the data granularity of tasks, directly increasing the number of parallel tasks. However, when the application exploits the resources well with small idleness detected, but the CPB is not near the actual makespan, it is not necessarily a problem. The DAG may have many parallel paths, and such time would inevitably be necessary. However, one problem is if the runtime chooses another less import path to execute instead of the critical path.

Another metric that estimates the makespan and considers the totality of tasks (and indirectly all paths in the DAG) is the Area-Bound Estimator (ABE). ABE computes the hypothetical situation where tasks do not have dependencies, and the runtime optimally partitions them across heterogeneous resources considering the speedup per task type per resource. The following linear program can compute ABE:

$$\text{Minimize } T \; s.t. : \tag{4.3}$$

$$\forall t \in \mathcal{T} : \sum_{r \in \mathcal{R}} \alpha_{t,r} = N_t \tag{4.4}$$

$$\forall r \in \mathcal{R} : \sum_{t \in \mathcal{T}} \alpha_{t,r} w_{t,r} \leq T \tag{4.5}$$

where T is the application estimated makespan (ABE), $\mathcal{T}$ and $\mathcal{R}$ are respectively the sets of tasks types and resources, $w_{t,r}$ is the expected duration of task t on resource r, N_t is the number of tasks of type t, and $\alpha_{t,r}$ is one of the optimized variables describing how many tasks of type t executes on resource r. If the ABE is distant from the original makespan, it is a possible indication of idle resources in the execution. However, this idleness is expected in some situations because of missing parallel opportunities in the DAG. For example, the Cholesky or LU DAG lacks parallelism at the beginning and end of the execution. Also, ABE can offer an estimate per node or globally. Restricting $\mathcal{R}$ with only resources for one particular node and N_t for the tasks that the current distribution assigns to that specific node gives the per-node ABE for this chosen node. It is global if there is no such restriction.

These metrics provide a summary of the application behavior. The application makespan will always be greater than CPB and ABE, considering that the duration of the tasks is stable, making both metrics a lower bound.

4.2.3 Visualizations of performance behavior

Another approach for analyzing these task-based applications is with visualizations (KERGOMMEAUX; STEIN, 2003). A common practice is to translate the logged events into a visual representation. Performance analysis with visual representations comes in different flavors, such as histograms, call graphs, scatter plots, line charts, or treemaps (SCHNORR; LEGRAND, 2013; ISAACS et al., 2014). Regardless of this assortment, the most prevailing views are Gantt-like charts (WILSON, 2003) where one axis is the time, and the other represents resources or entities with application states being mapped on such space.

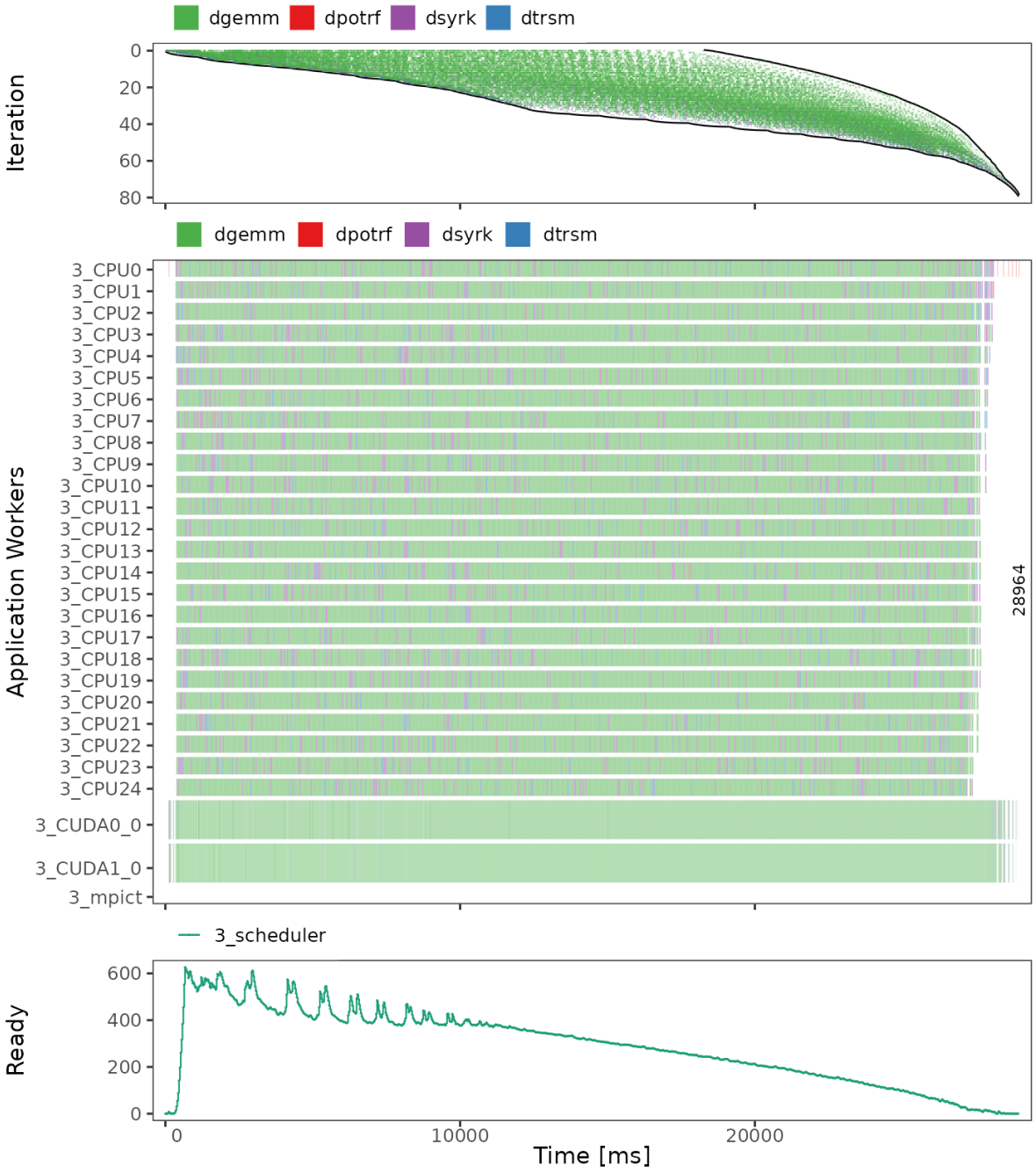
StarVZ offers many different visualizations considering the task-based paradigm and the StarPU runtime, which can contemplate memory management, application-specific visualization, and node aggregation information to reduce the size of visualizations (PINTO et al., 2021). This thesis will constantly show visualizations with three panels that are present in Figure 4.5. The X-axis in all panels is time in milliseconds (ms). First, the top one, the Iteration Panel, presents the application iteration over time, the Cholesky iterations in this particular example. This panel can represent any other applications' iterations. For instance, many linear algebra operations have outer loops. The left-most black line represents the start of the iteration, and the right-most the end. The second panel, Application Workers, describes the traditional Gantt chart of tasks, where the Y-axis is the resources, and the rectangles represent the tasks (start and end). This particular case visualization only selects the resources of node 3. The right-most number is the total makespan of the application. Some metrics can appear on this panel, including global and per node ABE, CPB, and other bounds (EYRAUD-DUBOIS, 2019). Moreover, StarVZ can compute outlier tasks using a simple detection rule when the task's duration is superior to the third quartile plus 1.5× the interquartile range (IQR). These outlier tasks appear with the same task color yet an intenser tonality, while the regular tasks have an opacity of 50% (lighter colors). The third panel is a generic variable over time visualization, in this case, ready tasks. Further details about these panels are available in Garcia Pinto et al. (2018).

In situations with many resources, aggregation is one strategy to reduce visualization required space. Figure 4.6 is the aggregated version of the second panel of the previous Figure, now considering all nodes, where instead of resources, the Y-axis is the utilization (%) per task on each node resource group. This version divides the data into timeslices and computes the utilization per task. For example, One green bar that reaches 80% on Node 1 CPUs indicates that all CPUs of Node 1 spent 80% of the time on that slice computing the `dgemm` task.

The visualizations are often responsible for many insights and the first way to look at

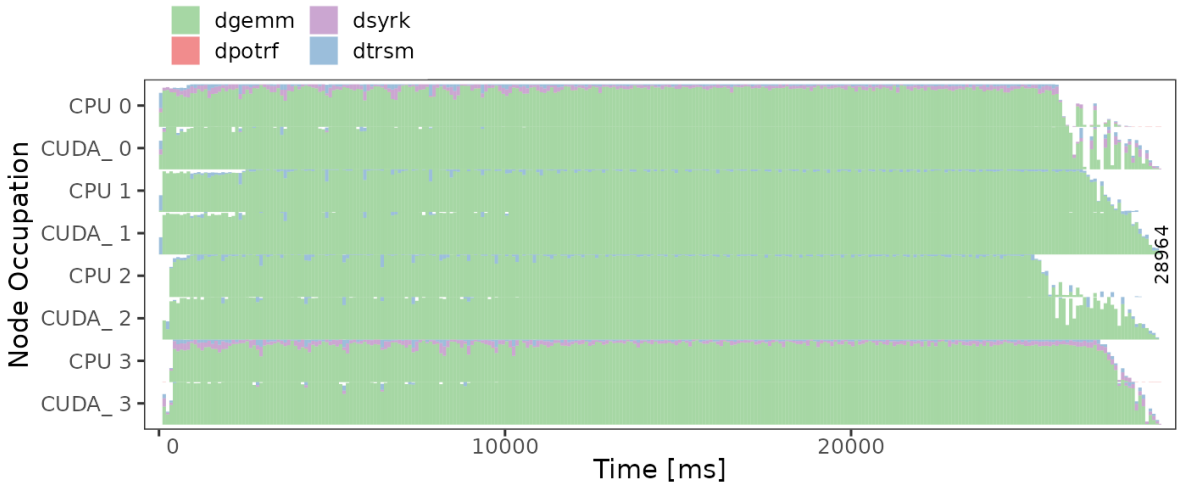
execution behavior. Since traces can have terabytes of data containing millions and millions of events, visualizations can compact all this information in a small space. However, an analyst should study behaviors requiring constant data manipulations in some complex situations. Using pre-defined graphical visualizations for all hypotheses that appear may be slower and more limiting than directly working with the data.

Figure 4.5 – StarVZ original workers Gantt Chart for application tasks



Source: The Author.

Figure 4.6 – StarVZ node aggregated Gantt Chart version for application tasks



Source: The Author.

4.2.4 Interpreting traces data

When metrics and pre-defined visualizations are insufficient to understand application behavior, one may have to analyze the direct individual events of the traces. This thesis uses the R language with its established data science ecosystem to analyze the trace data. This ecosystem comprises `tidyverse`, a meta-package that provides procedures to work and query tidy data. Some famous packages present in `tidyverse` are `readr` (for reading many data formats), `dplyr` (a grammar for data manipulation), and `ggplot` (for visualization). Finally, StarVZ prepares the raw trace data to an R-friendly structure during the first phase. With all these packages, it is possible to conduct a micro-analysis, filtering events with specific rules. For example, one example of StarVZ data is a table of the application tasks. The `tidyverse` framework enables simple queries like filtering all tasks with particular properties, for example, duration superior to duration percentile 0.95.

Considering an anomalous behavior at a specific time, it is possible to find associate events related to it, like memory transfers or allocations, scheduling decisions, or any other runtime inner work. In some situations, we discovered runtime memory-related operations, like late allocation, responsible for task lateness. It was absent from the normal critical path (as the latter is just composed of tasks), but we matched the allocation events using the R environment with custom-designed functions.

The StarVZ data structure comprises tables that contain structured trace data. The most notable ones are Application, containing application tasks data; and Links, containing memory transfer data. There are more than 20 data tables on StarVZ containing other information,

but these two are almost always present in all analysis situations. Tables 4.2 and 4.3 present small slices of the respective tables' real data with a selected representative set of columns. For example, Table 4.2 has tasks-related information, including the resource where it executed (ResourceId), the start, end, and duration of the task, the type of the task (Value), the JobId responsible for it, and the X and Y coordinates of data it writes. With this information, one can use the `tidyverse` verbs to filter for a given timeslice. For example, Algorithm 6 presents a filter for tasks that started after time 10000ms and ended before time 15000ms.

Table 4.2 – StarVZ application data table

ResourceId	Start	End	Duration	Value	JobId	X	Y
0_CPU0	0	14.49	14.49	dpotrf	0_2461	0	0
0_CPU1	14.73	50.48	35.75	dtrsm	0_2474	0	22
0_CPU0	19.11	54.85	35.75	dtrsm	0_2473	0	20
0_CUDA1_0	19.87	24.65	4.78	dtrsm	0_2467	0	8
0_CUDA0_0	20.54	25.32	4.78	dtrsm	0_2464	0	2
0_CPU1	55.07	94.76	39.7	dsyrk	0_2677	6	6
0_CPU0	59.46	117.73	58.27	dgemm	0_2597	2	20
0_CPU0	122.31	180.58	58.27	dgemm	0_2853	14	22
0_CPU0	185.15	224.85	39.7	dsyrk	0_3261	38	38
0_CPU0	229.43	287.7	58.27	dgemm	0_2894	16	26

Source: The Author.

Table 4.3 – StarVZ link data table

Type	Start	End	Duration	Key	Origin	Dest
MPI communication	14.49	21.81	7.31	mpicom_0	0_mpict	1_mpict
Intra-node TaskPreFetch	14.91	19.68	4.77	com_1	0_MEM0	0_MEM2
Intra-node TaskPreFetch	15.1	19.68	4.59	com_2	0_MEM0	0_MEM2
Intra-node TaskPreFetch	15.58	20.35	4.77	com_3	0_MEM0	0_MEM1
Intra-node Fetch	29.78	34.93	5.15	com_12	0_MEM2	0_MEM0
Intra-node Fetch	30.45	35.25	4.79	com_16	0_MEM1	0_MEM0
Intra-node Fetch	34.93	39.53	4.59	com_20	0_MEM2	0_MEM0
MPI communication	34.93	92.1	57.17	mpicom_2	0_mpict	2_mpict
MPI communication	34.93	94.03	59.09	mpicom_1	0_mpict	1_mpict

Source: The Author.

Algorithm 6: Basic R script to select events of the applications table that were executed between 10000ms and 15000ms

```
1 data$Application %>%
2     filter(Start > 10000, End < 15000)
```

Another more robust analysis is checking the critical path lateness over multi-node (with individual local clocks) executions. The lateness of a task is the time between the end of the last

dependency and when the task *de facto* started. StarVZ provides functions to compute the critical path, considering that multi-node executions can have a similar or virtual global time. NTP, PTP, or even minor corrections within traces using the last barrier as a synchronization point can adjust the precision of the time. An inter-node global time is hard to obtain. However, in our practical case, lateness is usually two, three, or more times the order of magnitude of the inter-node time inaccuracy. This precision is enough for our purposes as lateness is huge compared with the possible inter-node time maximum inaccuracy (provided by the NTP or PTP systems). Table 4.4 presents an example of a critical path displaying JobId, Node, Resource, Name, Start, End, Duration, and Lateness from a task. It is then possible to track back lateness problems to a responsible event. If MPI communications start late, more investigation in the StarPU MPI module or the machine network stack would be ideal. If there is lateness in communications between intra-node resources, there can be a problem in StarPU data management. This latter problem is the case of JobId `2_34751` lateness in Table 4.4, where it waited for a data transfer. However, there are some situations where lateness is inevitable. For example, the resources are busy when there are more high-priority tasks. This last situation could indicate that the application would benefit from expanding its resources.

Table 4.4 – Critical path data example from execution traces and R data manipulation

JobId	Node	ResourceId	Value	Start	End	Duration	Lateness
0_29507	0.0	0_CUDA1_0	dgemm	4850.54	4851.02	0.48	2.34
0_31384	0.0	0_CPU0	dtrsm	4851.65	4876.45	24.81	0.63
mpicom_5774	1.0	1_mpict	nil	4876.47	5263.13	386.66	0.02
1_31041	1.0	1_CUDA0_0	dgemm	5266.51	5266.97	0.46	3.38
1_32894	1.0	1_CPU0	dtrsm	5267.72	5292.16	24.43	0.75
mpicom_6274	2.0	2_mpict	nil	5292.18	5304.11	11.94	0.02
2_32924	2.0	2_CUDA0_0	dgemm	5305.00	5305.49	0.49	0.89
2_34751	2.0	2_CPU1	dtrsm	5329.28	5356.29	27.02	**23.78**
mpicom_6331	0.0	0_mpict	nil	5356.32	5522.86	166.54	0.03
0_35835	0.0	0_CUDA1_0	dgemm	5524.86	5525.35	0.49	2.00
0_37639	0.0	0_CPU1	dtrsm	5525.97	5551.79	25.82	0.62

Source: The Author.

4.3 Improving performance process

With the analysis methodology, it is possible to check some problems like underutilization of resources at a given point, resource imbalance, the lack of ready tasks, synchronization points in application iterations, inadequate values for any presented metrics, and memory-

related operations. All these analysis insights enable the analyst to propose possible solutions in some execution entities (application, runtime, library, system). These solutions require experimentation, analysis validation, and the repetition of the methodology. This process was applied to all problems in the following Chapters, with the results leading to the next study object, resulting in an iterative process.

5 HETEROGENEOUS DISTRIBUTIONS STRATEGIES FOR LINEAR ALGEBRA

This chapter focuses on a single operation and proposes strategies for generating static cyclic distributions for linear algebra procedures that must balance the load across many iterations. However, thanks to the flexibility of task-based runtimes, these iterations occur asynchronously in a hybrid fashion. The distribution among different computational nodes is static, and a dynamic scheduler handles intra-node imbalances. Although state-of-the-art cyclic and heterogeneous distribution algorithms like 1D-1D provide asymptotically optimal results, they still leave some opportunities for improvements and the creation of more refined distributions. Moreover, having a perfect balance load does not guarantee the best performance. Because of the critical path and nodes' heterogeneity, sometimes it may be better to have some imbalance with the extra load on the fastest nodes. The strategies discussed in this chapter actuate on a single operation, and their ideas are essential when examining multiple-phase interaction with multiple distributions in Chapters 6 and 7.

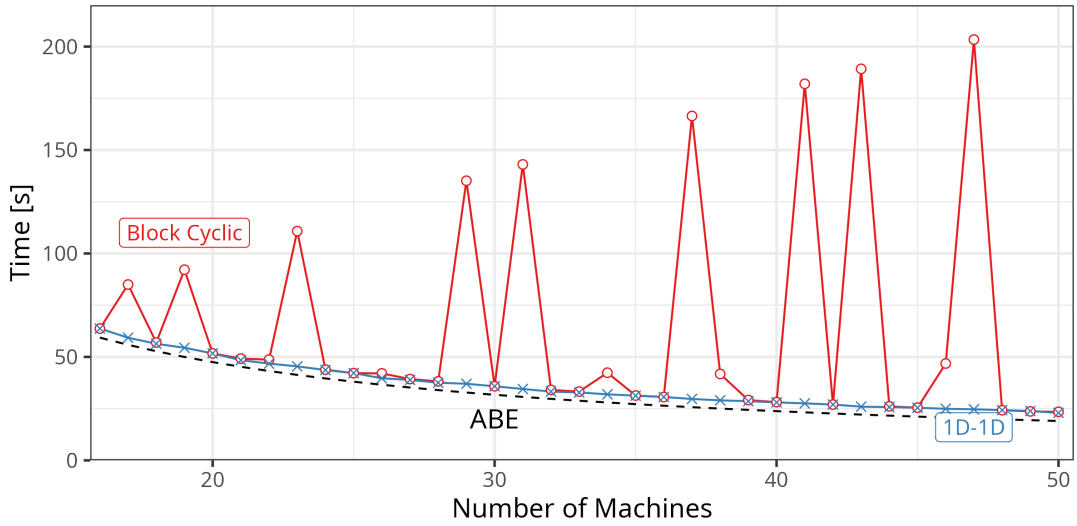
The experimental evaluation in this Chapter uses the LU operation, as most other linear algebra operations lead to similar parallelization challenges and require the same strategies. The remainder of the Chapter is organized as follows. Section 5.1 presents an analysis of strong scaling with state-of-the-art distributions over a homogeneous context. It compares the traditional homogeneous block-cyclic distribution, the standard in most libraries, and another heterogeneous distribution 1D-1D. Section 5.2 discusses the problems and available opportunities to refine this heterogeneous distribution over heterogeneous resources. Section 5.3 presents the proposed algorithms to generate distributions considering application moments with low parallelism and communication and load balance trade-offs. Section 5.4 presents the performance evaluation of a traditional heterogeneous algorithm against the block-cyclic distribution and the proposed distributions in heterogeneous environments. Section 5.5 finalizes the Chapters with a discussion and perspectives.

5.1 Strong Scaling in a Homogeneous Context

The BC strategy is common for homogeneous platforms. Unfortunately, it is known that when the number of machines does not nicely decompose in smaller prime numbers, the communication overhead may lead to significant performance degradation. Indeed, for any prime number of machines, one ends up with a pure one-dimensional distribution. Instead, as illustrated in Figure 5.1, the 1D-1D distribution handles such situations gracefully and scales

perfectly. This analysis adopts a fixed matrix size of 100×100 blocks of 960×960, gradually increasing the number of Chetemi computing nodes from 16 to 50 using simulation. Although 1D-1D distributions originated 20 years ago, to the best of our knowledge, it is the first time they appear in a real solver stack. Interestingly, the distance between makespans of the 1D-1D distribution and the (very optimistic) ABE bound of all resources is constant, corresponding to the end of the execution that cannot efficiently exploit all computing nodes.

Figure 5.1 – Strong scaling for a 100×100 matrix with 50 homogeneous Chetemis



Source: The Author.

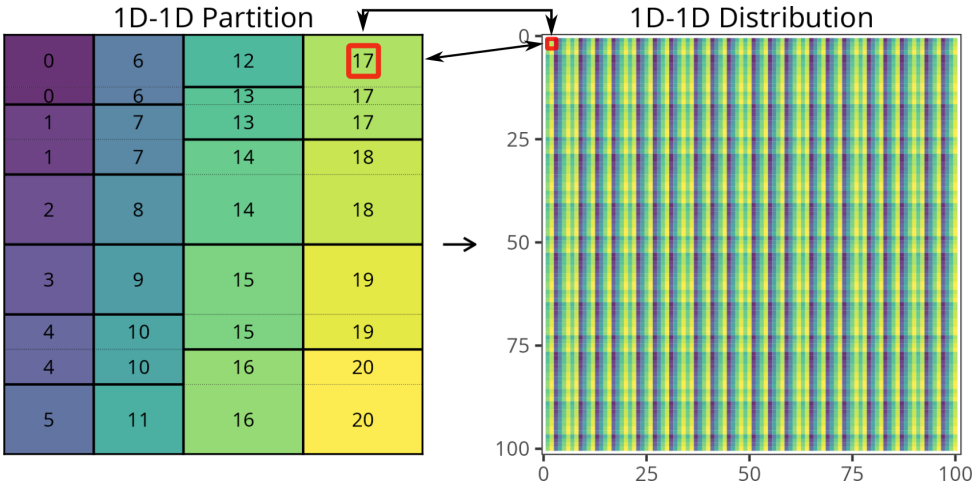
5.2 Problem: The Communications and Load-Balance Trade-off

Generally, the heterogeneous distributions (1D-1D of Chapter 3.1) want to minimize communication while maximizing load balance. However, decisions made to reduce communication may generate imbalance, while decisions to balance the load may increase communication. This situation creates a trade-off scenario between minimizing communications or improving load balance in these distributions. Considering the 1D-1D algorithm, there are two problems from a theoretical perspective that can be further enhanced: reducing communications further and improving balance.

The first problem is that the algorithm is asymptotically optimal, which means that while it does a good job balancing load, it may not be perfect at all iterations. These imbalances may cause cumulative bad behavior across iterations. For example, node A at iteration x lacked work. Still, no subsequent iterations got overloaded to compensate for the possible idleness of iteration x. The asynchronous execution may mitigate some of the imbalance by advancing through the

iterations and its critical path. However, because the other nodes still receive more tasks to do, idle time will appear at some point. This slight imbalance comes from the asymptotically optimal behavior in which the algorithm tried to shuffle and distribute a column-based partition to minimize the peri-sum problem and, subsequently, the communication. Figure 5.2 illustrates the distribution construction using the partition, where node 17 of the partition is mapped to blocks of a cyclical distribution. There is not much to do considering the column-based partition with virtual columns and lines that a node will only communicate with other nodes that share the same virtual column and line. However, if the final distribution relaxed such constraints and allowed extra communications, one could improve the balance.

Figure 5.2 – The 1D-1D partition (left) and the reciprocal 1D-1D distribution for 14 slow and 7 fast nodes (total of 21 nodes)



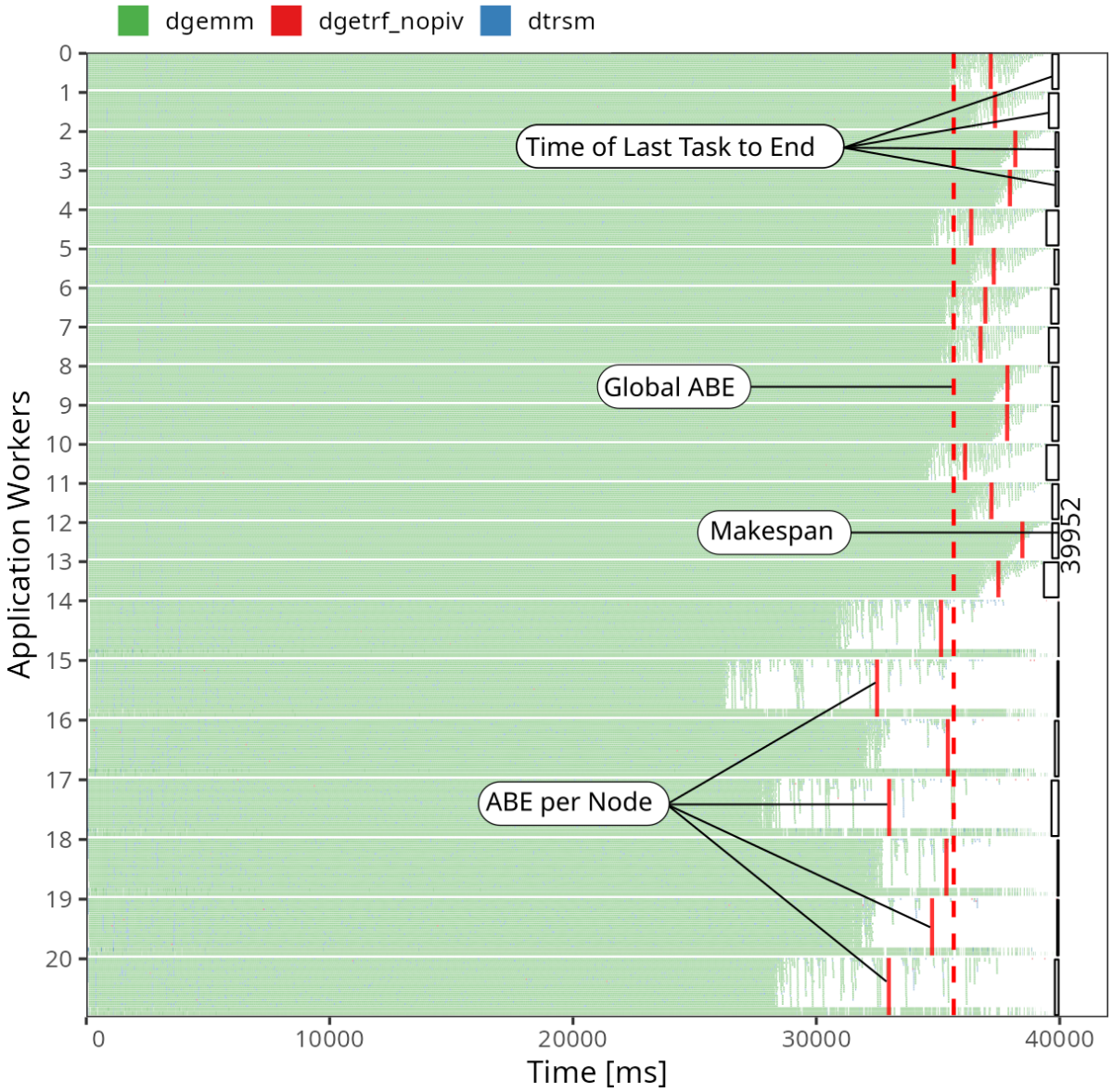
Source: The Author.

The second problem is that while the algorithm may try to balance the load across all iterations, a balanced workload may not deliver the best performance. For example, considering the end of the LU factorization, there is no reason to spread a few tasks across many nodes. This lack of parallelism at the end of the algorithm is also observed in other operations, including Cholesky. There is also the problem of who will execute these fewer tasks. When the system-level heterogeneity includes nodes with only CPUs and ones with GPUs plus GPUs, the remaining tasks should go to nodes with GPUs to advance the critical path faster.

Figure 5.3 shows the simulated execution of the LU factorization using 1D-1D with 21 nodes, where 15 nodes are slower (the Chetemis, which are CPU-only) than the other seven faster (the Chifflets). The performance ratio of these two node types is almost 2:1, considering the number of `gemm` operations per second. The solid red line is the per-node ABE, which is very different per machine and shows an imbalance. This situation reinforces that the 1D-1D distribution is only asymptotically optimal, and the power per node used when computing the

column-based partition considered only the `dgemm` computation kernels. Another problem is that all the fastest nodes (those with GPUs) are not only slightly less loaded but mostly idle toward the end of the execution. Using all 21 nodes toward the end incurs many synchronizations and communications. The computation would end sooner if only the faster nodes worked, gathering the low ending load towards faster nodes. Furthermore, using slow nodes toward the end may also negatively impact the progression along the critical path, as the critical `dgemms` would use CPUs on the slower nodes and GPUs on the fastest ones.

Figure 5.3 – The 1D-1D partitioning using 14 CPU-only nodes plus 7 GPU-equipped nodes (simulation)



Source: The Author.

The analysis of the 1D-1D algorithm showed opportunities to refine the distributions in crucial moments. First, considering a cyclic distribution, there is no need to balance work across

all iterations equally, as in some parts of the DAG (end, for example), there is limited parallelism that a node can handle, reducing communication. Figure 5.3 shows this by presenting a low usage "tail" on all the resources, where the tasks are gradually being executed in the first CPUs or only in GPUs on each node, as there are not enough tasks to distribute among all resources. Also, these algorithms are not optimal across nodes, as the per-node ABE estimations pointed out. One could expect that per-node ABEs, computed after the load partitioning, would be very similar. These results indicate that increasing communication in favor of load balancing may present benefits.

5.3 Proposal: Communication and Load-Balance Trade-off Aware Distributions

Considering the analysis of the last section, this Chapter proposes two strategies. First, using DAG information and metrics to reduce communication by concentrating load in powerful nodes in low-parallelism moments (Section 5.3.1). Effectively, not using some nodes at the end of the operation. Second, improving inter-node per iteration balance by changing the balance-communication trade-off and collaterally increasing communications (Section 5.3.2).

5.3.1 Constraining an Heterogeneous Distribution

The proposal's main idea is to constrain the final iterations of the linear algebra operation to use only the faster nodes, thereby extending and adapting the original 1D-1D algorithm. The allocation is built incrementally similarly to the 1D-1D algorithm by picking the best virtual row and column of processors for each line and column k from nb (the right-bottom part of the matrix) down to 1 (the left-top part of the matrix). However, we restrict the selection of possible columns and rows using an incremental number variable *section*. The heuristic applied to this case is to use a maximum of *section* and $section^2$ faster entities for columns and rows, respectively. This choice comes from the idea that the partition usually has more rows than columns, and the most powerful nodes are in the right-most columns. Using all rows of some columns would result in using faster nodes, but it would perform excessive communication as using a 1D pattern communicates more than a 2D one. For example, for a given k, and a partition with five columns and ten rows, the 1D-1D algorithm could select among all columns and rows if there was no such restriction. In the constrained version, the variable /section/starts with the value of 1. Then, the constrained version would only be able to select among the largest

(faster) 1 column and 1^2 row. When *section* updates to 2, the algorithm could choose among the fastest 2 columns and the fastest 4 rows. Algorithm 7 presents the pseudo-code for this 1D-1D constrained version (1D-1D C).

Algorithm 7: 1D-1D constrained

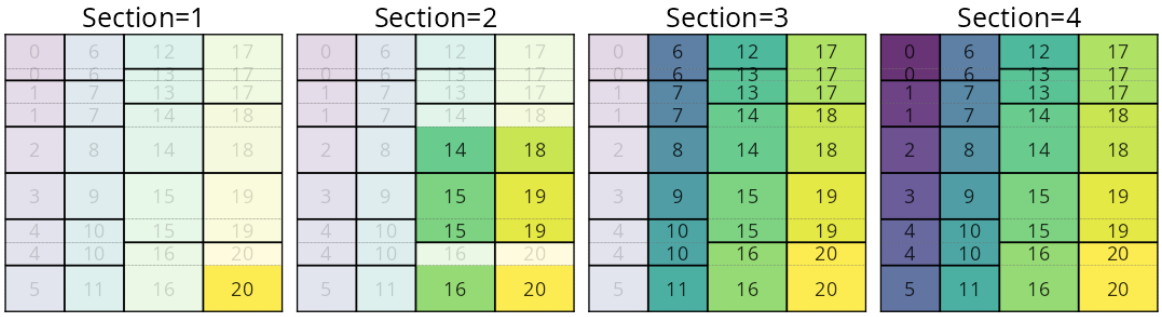
```
1  Input: partition Base distribution to apply the procedure
          nb Matrix size in blocks
   Result: Distribution
2  section ← 1
3  for Each iteration backwards from nb to 1 do
4      available_columns ← last section columns from partition
5      Find the available column that less increase the load if selected and add in
        selected_columns
6      available_rows ← section^2 largest rows from partition
7      Find the available row that less increase the load if selected and add in
        selected_rows
8      if section < number of columns in partition or section^2 < number of rows in
        partition then
9          Compute CPB_MPI of iteration
10         Compute total ABE of iteration
11         if ABE > CPB_MPI then
12             section++
13             Discard last selected row/column
14         end
15     end
16 end
17 Distribution ← Expand selected rows/columns
```

Algorithm 7 starts with a section made of a single virtual column (the right-most column on the partition, with the fastest nodes) and a single virtual row (the largest one from the fastest node). The minimum time required to process iteration k on node p is given by the inner-node ABE with the number of tasks of iteration k assigned to p, defined as ABE-IN$_p(k)$. Also, ABE-IN$^T(k) = \max_p$ ABE-IN$_p(k)$ is the minimum time required to process iteration k for a given distribution. Since we do not redistribute the matrix during the factorization, this maximum ABE-IN$^T(k)$ is always larger than the absolute ABE(k) defined as the ABE of using all nodes with all the tasks of iteration k. This way, ABE(k) expresses a fully balanced case using all nodes, while ABE-IN$^T(k)$ points to the maximum individual load per node. The algorithm also uses the critical path bound CPB(k) of this iteration, i.e., the duration of the largest sequence of tasks in the DAG (considering communications). The metric would be the optimal processing time of iteration k if an infinite number of resources were available. If CPB(k) is larger than ABE(k), that means that iteration k has a significant critical path that

inevitably will take longer than the time required if was possible to divide all the iteration work to the resources equally (relative to powers). In this way, if CPB(k) > ABE(k), there will be idle resources. Moreover, there is the opposite situation, ABE(k) > CPB(k) means that the time to process all tasks is longer than the critical path, so if more resources are added, the performance could be improved. Since initially (when $k \approx nb$, as the algorithm processes it backward), there is very little work in the sub-matrix $A[k \ldots nb][k \ldots nb]$. The distribution metrics starts with ABE-IN$^T(k)$ = ABE(k) < CPB(k). When ABE(k) becomes larger than CPB(k), some work of k will have to wait for resources, and the application would benefit from adding new nodes.

The algorithm increments the variable *section* when ABE(k) > CPB(k), adding one new column of processors from the right to the left of the partition. This selection results from the Beaumont et al. (2001b) column-based partitions that sort nodes by their processing speed. Also, the rows are sorted by decreasing height and involving the fastest possible nodes. As the partition usually has many more virtual rows than columns, it benefits from releasing them faster than columns to obtain a better load balance. Figure 5.4 presents the progress of the constrained algorithm partition with a different number of sections. First, with section = 1, it releases node 20, then with section = 2, it releases nodes 14 – 16, 18, 19; then nodes 6 – 11, 12, 13, 17; and finally all the nodes.

Figure 5.4 – The progression of the 1D-1D constrained partition from one to four sections

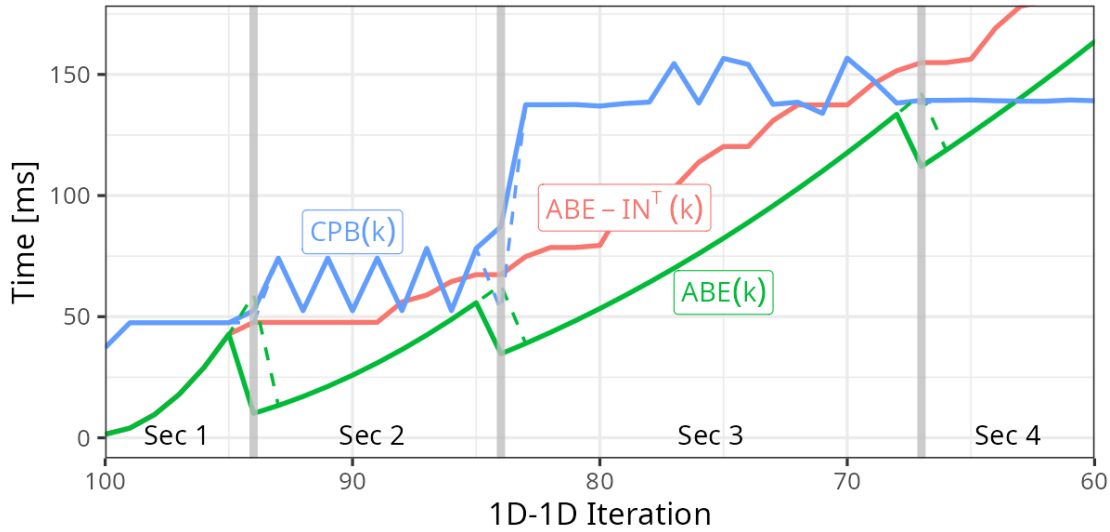


Source: The Author.

After adding the new virtual columns and lines, the algorithm keeps allocating rows and columns of the matrix using the 1D-1D procedure while updating ABE(k) and CPB(k). Figure 5.5 illustrates the progression of these metrics, where the blue line is the CPB(k) metric, the green line is the ABE(k) per iteration considering all available resources, the red line is the ABE-IN$^T(k)$, and the gray vertical lines point whenever ABE(k) would become larger than CPB(k) to create a new section. The ABE(k) increases quadratically within a section, proportionally to the sub-matrix, until it exceeds the CPB(k), as indicated by dashed lines. Therefore, we always have ABE(k) $\leq$ CPB(k) unless we run out of processors and cannot

create a new section, as it happens with four sections.

Figure 5.5 – The 1D-1D C metrics (CPB(k), ABE(k) and ABE-IN$^T(k)$) for 7+14 machines from iteration 100 to 60 with four sections

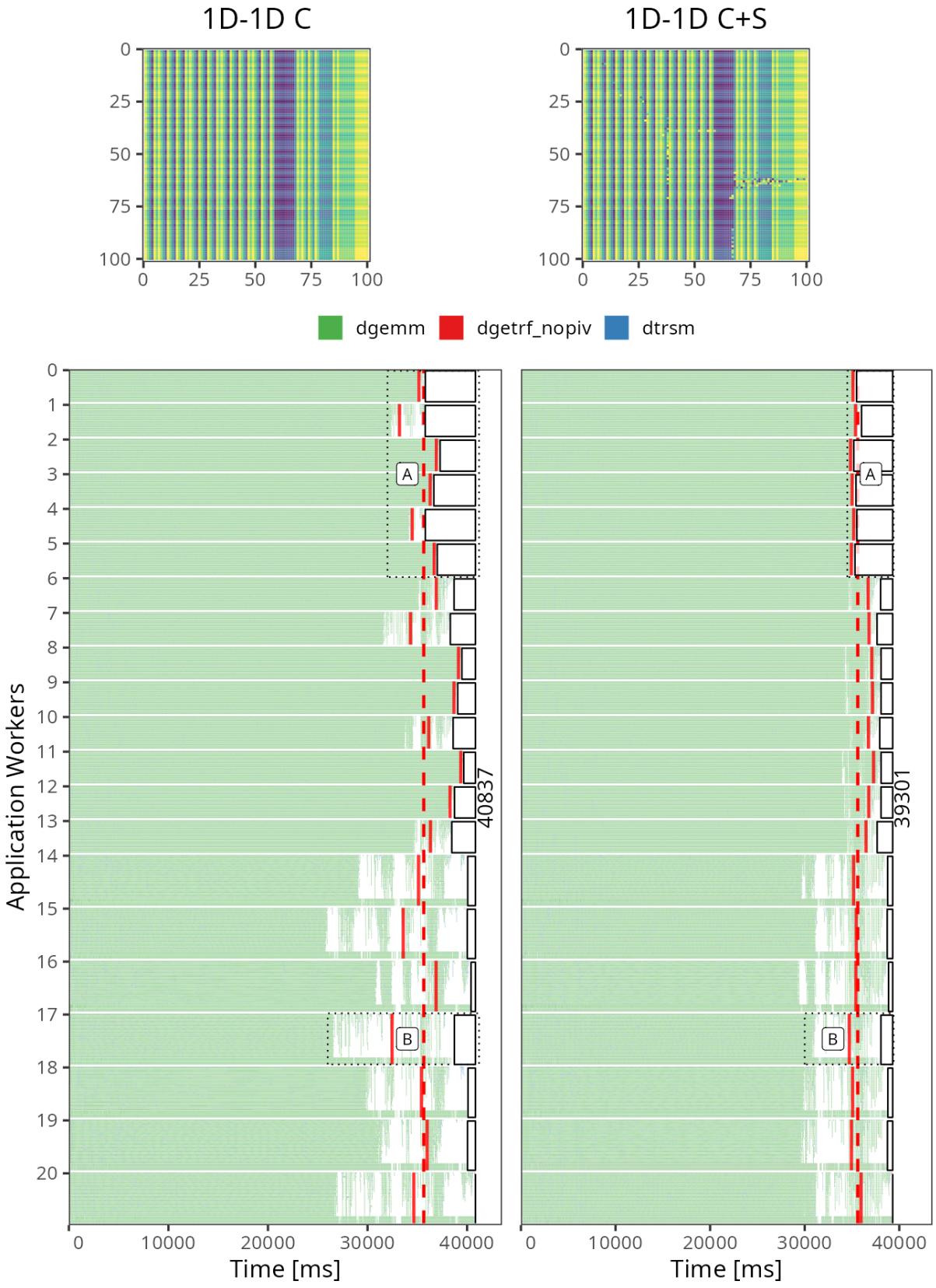


Source: The Author.

In the example of Figure 5.5, it allocates a 100×100 matrix on 14 (slow) Chetemi and 7 (fast) Chifflet, which leads to four sections. The first section corresponds to iterations 100-96 and contains only one Chifflet. Then, the section from iterations 95-85 contains six Chifflet nodes, followed by the section from iterations 84-68 that contains seven Chifflet and eight Chetemi nodes. Finally, the section from iterations 67-1 contains all the nodes. In Figure 5.5, one may note that the CPB varies from one iteration to another, which may seem surprising since the longest path in the sub-DAG corresponding to iteration k remains the same throughout iterations. However, the CPB is per iteration and not cumulative. The MPI communications incur using several nodes justify the first increase of CPB in the second section. The second (and much larger) increase of CPB in the third section appears because of the addition of slow nodes (Chetemi). Small differences in the tasks' execution times between machines cause remaining fluctuations.

The upper left part of Figure 5.6 depicts the final resulting data distribution, called 1D-1D C, which is way less regular than the original 1D-1D distribution (Figure 5.2) and favors the use of the faster nodes toward the end. In contrast, the distribution of the upper parts of the matrix remains relatively similar. The bottom left part of Figure 5.6 depicts the execution obtained with this constrained distribution. A first notable difference with the execution of a 1D-1D distribution (see Figure 5.3) is that now, as indicated by the rightmost white rectangles on each node (A), the slower processors stop working almost instantaneously instead of vainly contributing to the end of the computation. Unfortunately, even though slower nodes finish their

Figure 5.6 – Execution with 14 CPU-only nodes plus 7 GPU-equipped nodes, with the distribution (top) and behavior (bottom) of the 1D-1D C (left) and the 1D-1D C+S (right) runs



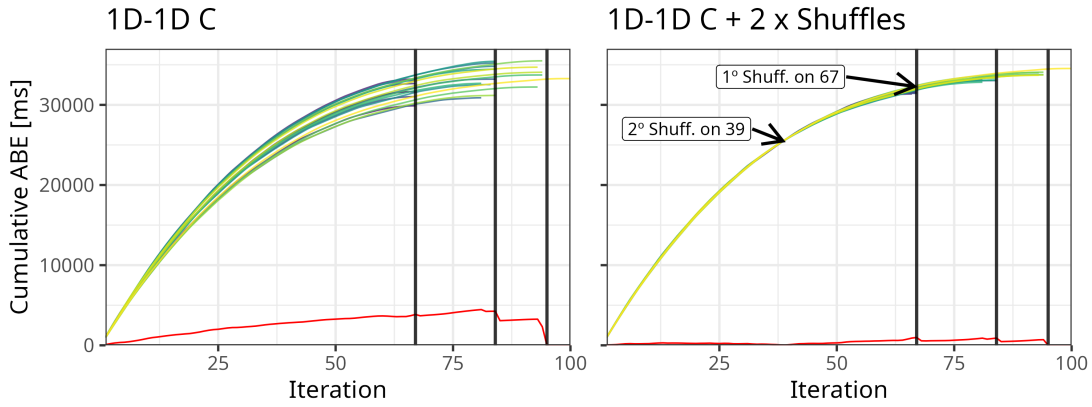
Source: The Author.

work much sooner, the load is still very imperfectly imbalanced (as illustrated by the red lines). The faster nodes still exhibit critical idle periods (B), indicating room for improvement.

5.3.2 Shuffling Blocks

When using the 1D-1D, there were some idle times in the middle of execution, relatively early. Further investigation identified that the work imbalance among nodes in the earlier iterations is the cause of these idle times. Remember that with a runtime like StarPU, there is no synchronization between each iteration. Nevertheless, if some nodes progress faster across iterations than others, they may quickly run out of work. Let us consider the cumulative ABE per node up to iteration k, i.e., $\text{CABE}_p(k) = \sum_{j=1}^{k} \text{ABE}_p(j)$. As illustrated on the left of Figure 5.7, where the group of lines represents the CABE per node, and the red bottom line represents the difference of the $\max_p \text{CABE}_p(k)$ and $\min_p \text{CABE}_p(k)$. There is a significant imbalance for some iterations, so the nodes with less work have to wait for the most loaded ones.

Figure 5.7 – Cumulative ABE (CABE) per node prior and after two shufflings



Source: The Author.

To alleviate this load imbalance, we propose a Shuffling mechanism that moves data blocks around, potentially inducing more communications since node alignments (in the partition) may become broken. Shuffling occurs within a section, say iteration k_1 to k_2, and aims at balancing the total work of this section (remember 1D-1D is only asymptotically optimal), $\text{SABE}_{p,k_1,k_2} = \sum_{j=k_1}^{k_2} \text{ABE}_{p(j)}$. Algorithm 8 presents the shuffling algorithm. The idea is that the fastest node gets the right-bottom most block of the slower node once we determine (lines 5-6) the least and most loaded node. This process repeats (line 4) until one of the two possible conditions becomes true. (i) The difference between the SABE of the nodes falls under a threshold (lines 16-19). In our experiments, we use 20ms. (ii) The fastest node does not have more

Algorithm 8: Shuffling algorithm

```
1  Input: Distribution Base distribution to apply the procedure
           k2 Factorization iteration cap to consider
           k1 Minimum factorization iteration to consider
           max_it Limit of iterations to move blocks
           similar_time Threshold of difference to stop changing
   Result: Distribution
2  SABE ← Compute SABE from k1 until iteration k2 for selected nodes
3  Zone ← List of select blocks between k1 and k2, reversed sorted by min(x, y)
4  moving ← True
5  while moving and max_it do
6      max_it - -
7      slow_node, slow_time ← Select slowest node by SABE and its SABE time
8      fast_node, fast_time ← Select fastest node by SABE and its SABE time
9      target ← mean(SABE)
10     possible_blocks ← Select blocks on Zone owned by slow_node
11     searching ← True
12     if sizeof(possible_blocks) == 0 then
13         # No more blocks to move
14         break
15     end
16     if slow_time - fast_time < similar_time then
17         # Reached the similarity
18         break
19     end
20     while searching do
21         selected_block ← Next block in possible_blocks
22         Duplicate the distribution and change this block to the fast node.
23         new_slow_abe, new_fast_abe ← compute new SABE for these two nodes
             using the duplicated distribution
24         if new_slow_abe > target or new_fast_abe < target then
25             searching ← False
26             Commit differences on Zone, Distribution and ABE
27         end
28         if this was the last block in possible_blocks then
29             searching ← False
30             moving ← False
31         end
32     end
33 end
```

blocks to be moved in this section. The SABE may be exceptionally high because of blocks from previous sections (lines 12-15 and 28-31). Moving blocks from two nodes (line 20) continues until a move results in trespassing the average SABE_{p,k_1,k_2} (line 24). The right of Figure 5.7 shows how cumulative CABE evolves after two shuffling operations are applied, where the red

line represents the difference between the maximum and minimum CABE(k). The top right of Figure 5.6 depicts the resulting irregular distribution, called 1D-1D C+S, with the runtime behavior.

The general proposed methodology first applies this shuffling to the largest section (in the number of iterations it spans). Usually, if the work is sufficient for the total amount of blocks, it will be the first section containing all nodes. It may be the case that after a shuffling, the cumulative load imbalance ($\max_p \text{CABE}_p(k) - \min_p \text{CABE}_p(k)$) remains high, in which case the methodology incurs an additional shuffling at this spike. When applied carefully, this shuffling operation allows smooth progression across iterations and reduces idle periods.

The upper right part of Figure 5.6 depicts the block shuffled distribution, where small disturbed areas (the one between rows 50 and 75, for example) exist and correspond to the few blocks allotted to faster nodes. A noteworthy aspect of the corresponding execution, depicted in the bottom right of Figure 5.6, is that now the ABE of each node is almost perfectly balanced (for example, the comparison of A areas for the first six nodes). In this example, the makespan improvement is insignificant: 39.30 seconds compared to 39.94 for the original 1D-1D distribution. However, the activity period of all slow nodes now matches their ABE, which may allow putting these nodes into a deep sleep mode to save energy or do some other operation. Idle periods (see B areas) on the faster nodes are significantly reduced compared to the distribution without block shuffling. Idling corresponds to the small load imbalance, which is maximum for iteration 84 and visible in Figure 5.7. They appear because of the lack of ready tasks. Our experiences in balancing this load further indicate how difficult it is (there is very little work) and that shuffling should be applied with much care as it involves balancing the total load from the beginning with the remaining one.

5.4 Performance Evaluation

This Section evaluates the efficiency of BC, 1D-1D, 1D-1D C, and 1D-1D C+S distributions in three different scenarios. Section 5.4.1 shows the gains brought by 1D-1D C and 1D-1D C+S when strong scaling on a heterogeneous platform. Section 5.4.2 presents results on a larger heterogeneous cluster. Finally, Section 5.4.3 presents how distributions compare in two setups with different heterogeneous node configurations to depict GFlops when growing the matrix size. Note that these experiments use simulation and employ all available cores, as detailed in Chapter 4.

5.4.1 Strong Scaling in a Heterogeneous Context

This evaluation considers the progressive scenario of adding more nodes. First, it only uses Chifflet nodes as they contain GPUs, and then, gradually, it adds Chetemi nodes, forming a heterogeneous platform at the end. Table 5.1 depicts all the studied configurations.

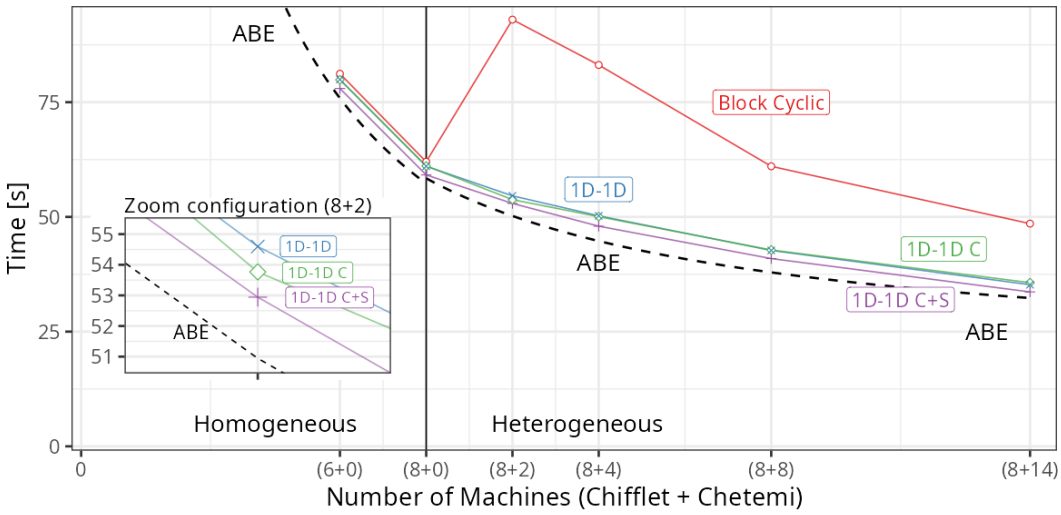
Table 5.1 – Machines configurations used

Case Identification	Chifflet	Chetemi	Total Machines
6 (6+0)	6	0	6
8 (8+0)	8	0	8
10 (8+2)	8	2	10
12 (8+4)	8	4	12
16 (8+8)	8	8	16
22 (8+14)	8	14	22

Source: The Author.

Figure 5.8 presents the performance of every distribution in terms of execution time (Y-axis) as a function of the platform size (X-axis). As expected, when slower nodes participate in the computation, the time required by the BC distribution dramatically increases since the whole execution progresses at the speed of the slower nodes. In contrast, 1D-1D distributions gracefully handle the addition of new nodes and are all very close to the optimal (the ABE with all resources). Although the gain is small (1 to 2 seconds), 1D-1D C and 1D-1D C+S improve systematically upon 1D-1D. The inset zoom with the (8+2) node configuration shows

Figure 5.8 – Execution time (Y-axis) as a function of combinations of number of machines (X-axis) and distributions (lines)

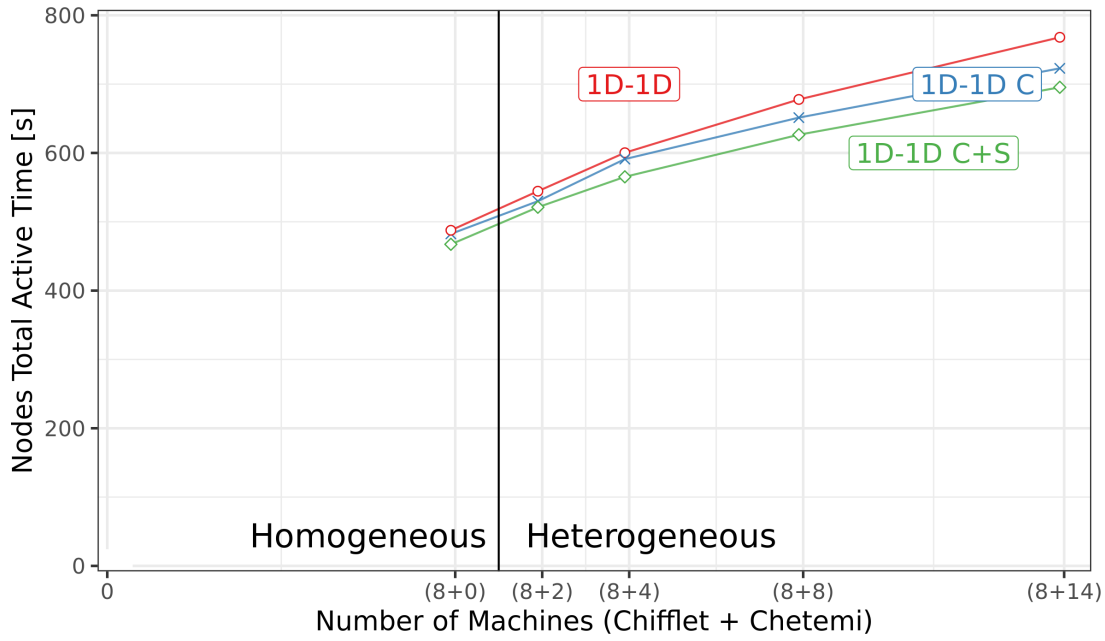


Source: The Author.

differences between strategies. For larger configurations, the difference between 1D-1D and 1D-1D C decreases while 1D-1D C+S maintains a constant gain of about 2 seconds over 1D-1D. This constant gain appears because 1D-1D C+S manages to ensure smooth progress throughout iterations and to optimize the end of the execution (where the gain in this context is limited anyway). Such behavior of constraining the end of the execution will be useful in Chapter 6.

A consequence of the 1D-1D C distribution to use fewer nodes toward the end of the computation allows releasing nodes earlier. Figure 5.9 presents the total utilization time per configuration. In the (8+14) case, the 1D-1D C (695s) and 1D-1D C+S (722s) significantly improve upon 1D-1D (768s). This resource idleness can be further used for two situations. (i) Reducing energy consumption, assuming nodes can enter a deep sleep mode when fully idle. (ii) Advancing other operations, this is the case on multi-phase applications.

Figure 5.9 – Total machine utilization time (Y-axis) for different number of machines (X-axis) and data distributions (lines)



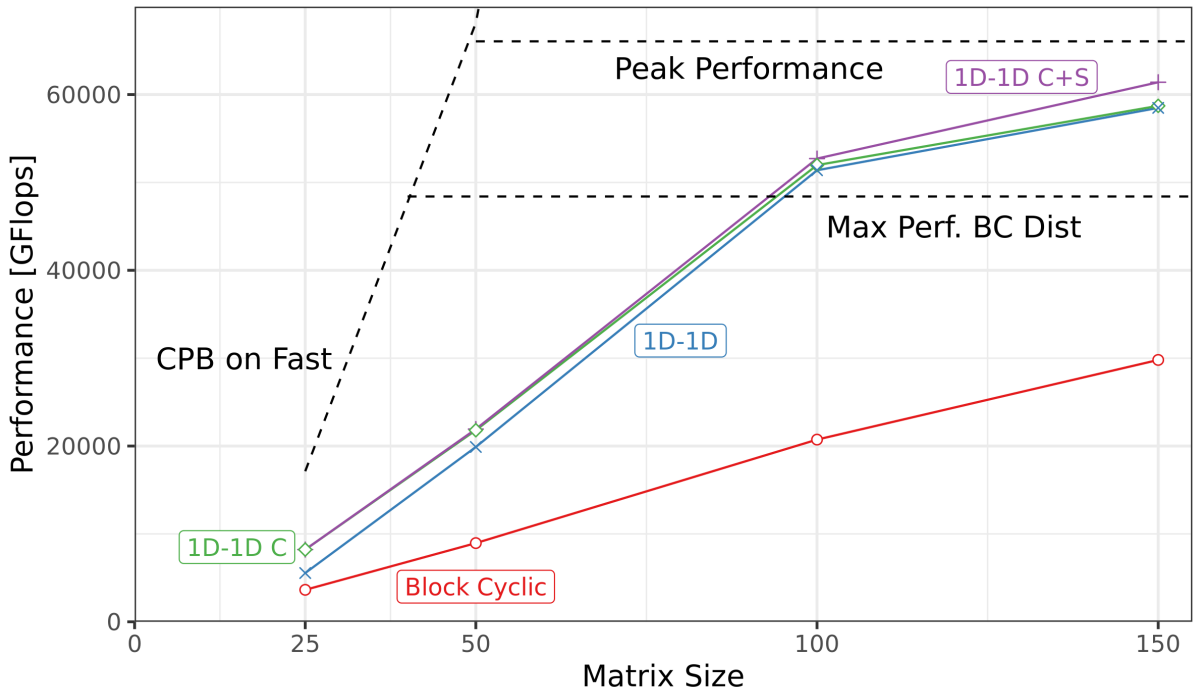
Source: The Author.

Finally, evaluating how much these distributions incur communication is also interesting. Although both strategies are derived from the column-based Peri-Sum partition, the constraining and the shuffling ideas trade off some communications for better load balancing throughout the execution. When factorizing a 100×100 matrix with 8+14 nodes, a pure BC requires transferring $55\,329$ blocks, while the 1D-1D reduces this down to $35\,417$ blocks, and 1D-1D C+S requires the transfer of $42\,632$ blocks. It is thus important to understand that although the 1D-1D C+S allows for better exploitation of computing resources, some communication overhead costs may become problematic at a larger scale.

5.4.2 Performance Gain over a Larger Heterogeneous Cluster

This evaluation now uses a 46-node cluster made of 16 fast Chifflet nodes and 30 slow Chetemis nodes and using discrete matrix sizes. The goal is to evaluate how quickly it reaches the maximum performance (in GFlops). Figure 5.10 presents the performance of each distribution for matrix sizes ranging from 25×25 blocks (which is very small since each node gets less than 14 blocks on average for a BC distribution) to 150×150 blocks. Similarly to the previous evaluation, it depicts the maximum achievable performance computed from the ABE (considering all resources) and the CPB (without communication). As expected, the best performance achieved by a BC distribution is far smaller than the peak performance, and the BC distribution is still far from it even for the 150×150 matrix (about 40% less). The rather unfavorable geometry for BC distributions ($46 = 2 \times 23$) has too many communications, explaining these results. Variants of the 1D-1D distribution remain undisturbed by the prime decomposition of the number of nodes, getting quickly closer to the peak performance (within 6% for the 150×150 matrix).

Figure 5.10 – GFlops performance (Y-axis) for different matrix sizes (X-axis) and distributions (lines) for a case with 16+30 nodes



Source: The Author.

Again, the 1D-1D C+S distribution obtains systematic gains over both 1D-1D C and 1D-1D distributions. It is interesting to note that for small 25×25 matrices, 1D-1D C+S and

1D-1D C distributions are equivalent (there are not enough blocks to shuffle) and significantly improve upon the 1D-1D distribution (32.5% gain). For large 150×150 matrices, 1D-1D C and 1D-1D C obtain a similar performance, which reflects the asymptotically optimal but imperfect load balance. Also, the duration of the factorization is rather significant than the possible critical path gains at the end of the execution. Contrarily, the 1D-1D C+S improves the overall load balancing, resulting in performance gains. Last, regarding communications, the factorization of a 150x150 matrix requires the communication of $129\,164$ blocks for 1D-1D. Constraining the execution on fewer nodes towards the end allows it to decrease this amount to $123\,780$ blocks, but shuffling blocks to achieve a better load balancing increases this amount up to $149\,474$ blocks. This result indicates that the methodology adopted in the shuffling may be too aggressive, and this could become a problem if communication is the main downing factor.

5.4.3 Performance Gain over Different Levels of Heterogeneity

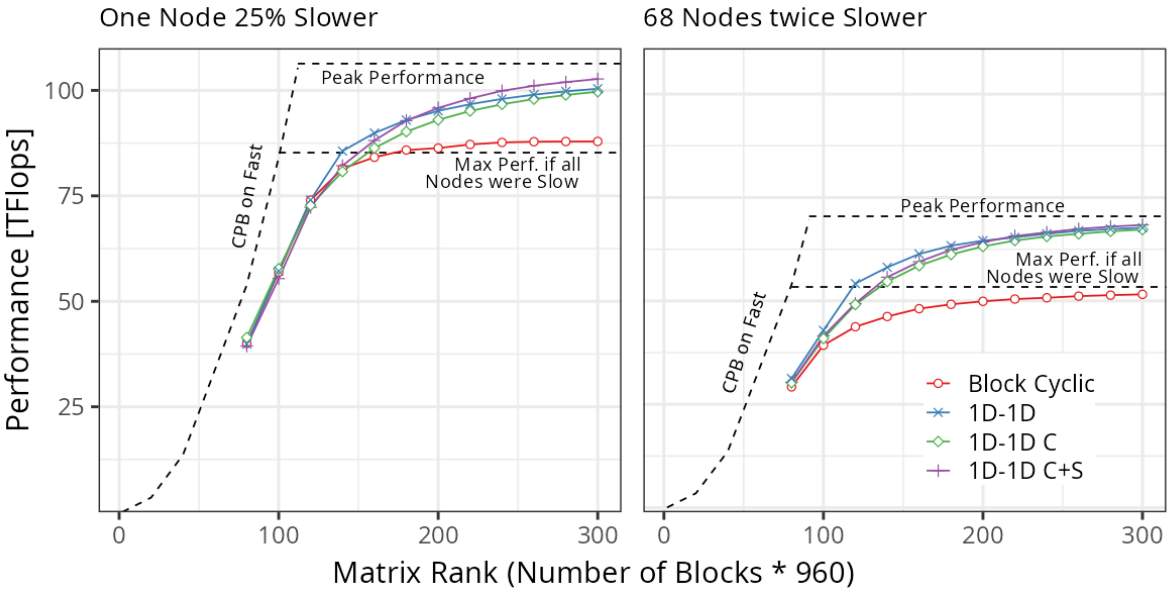
The final analysis presents two cases with Chetemis machines. First, it uses a 100-node cluster where one node is 25% slower on computing `dgemms` (Figure 5.11 left). This scenario is typical of a mild heterogeneity where one node may, unfortunately, create a significant load imbalance. Second, a 100-node cluster where 68 nodes are twice slower for the same operation (right). The second scenario is representative of a strong heterogeneity with an approximate difference of 2× between nodes because of GPUs. In both cases, it gradually increases the matrix size, ranging from 80×80 blocks (which is very small since each node gets 64 blocks for a BC distribution) to 300×300 blocks. The goal is thus to evaluate how quickly we reach the peak TFlops performance.

Similarly to the previous evaluation, the ABE (considering all resources) and the CPB (disregarding communication cost) present the possible peak performance. In both scenarios, the performance achieved by a BC distribution degrades as the matrix size increases, as the slowest machine limits it, and is far smaller than the peak performance. For the 300×300 matrix size and first scenario (a single node is 25% slower), BC is 14.5% slower than 1D-1D and 17.1% more than 1D-1D C+S. For the second scenario, BC is 31.4% slower than 1D-1D and 32.7% slower than 1D-1D C+S. The excessive work on the slower nodes explains these results, as all other nodes are waiting for data dependencies in these nodes. Variants of the 1D-1D distribution remain undisturbed even in the presence of slower nodes, getting much closer to peak performance.

The 1D-1D C+S distribution demonstrates slightly better performance than other distri-

butions for large matrix sizes in both scenarios and shuffling always improves the constrained version. For smaller matrix sizes, 1D-1D C presents lower performance than 1D-1D. For matrix sizes near 100×100, we observe that the 1D-1D is better than the others, which indicates that the methodology adopted for this constrained version is too aggressive in limiting resources at the end of the execution. Last, regarding communications, the factorization of a 300×300 matrix on the first scenario requires the communication of 812 370 blocks for 1D-1D. Constraining the execution on fewer nodes towards the end allows us to decrease this amount to 798 434 blocks, but shuffling blocks to achieve a better load balancing increases this amount up to 904 714 blocks. Still, the concentration of load and better load balancing justify this communication increase.

Figure 5.11 – TFlops performance (Y-axis) for different matrix sizes (X) and distributions (lines) with 100 nodes where one node is 25% slower (left) and 68 nodes are twice slower (right)



Source: The Author.

5.5 Discussion

This Chapter studied static cyclic data distribution techniques for homogeneous and heterogeneous sets of nodes in the context of dynamic task-based runtimes. Experiments with the LU factorization demonstrate how the 1D-1D distributions can be beneficial even for a homogeneous group of hybrid nodes when compared against BC, presenting much more stable scalability. Considering a heterogeneous set of hybrid machines, for which the 1D-1D is also near-optimal, we propose two new strategies. The first uses fewer nodes towards the end of the operation, gradually intensifying the computation in the powerful machines. The second applies additional shuffling of blocks to level out the cumulative work. The constraining technique allows

for optimizing the end of the execution, while block shuffling enables the reach of a quasi-optimal load balance across iterations but increases communication. A careful combination of these techniques provides selective performance improvements against the original 1D-1D, both in load balancing and execution time.

This study also indicates that the 1D-1D distributions are an excellent starting point, which is not so easy to improve. In practice, when only considering one linear algebra operation, it is unclear whether a systematic use of 1D-1D C and 1D-1D C+S is beneficial as it requires a good performance model and may induce communication overhead. However, those two strategies are tools that can be applied to specific situations. When problems at the end of the execution arise because of limited parallelism, excessive communication, or wrong scheduled critical path, the constraining-like strategy may help. Further reshuffling may be beneficial for cases where the balancing provided by the 1D-1D could be improved, and communication is not a problem (in a high-speed, low-latency network, for example). This first study identifies general communication and critical path trade-offs and the consequences of irregular distributions on the overall execution. Also, this study shows that non-perfect balanced distributions (like the constrained version) may present benefits in some situations.

Finally, applications may have several algorithm phases that require different heterogeneous distributions. The interaction between these distributions should be carefully studied when using a fully asynchronous system, like task-based runtimes. The time gained on some resources by deactivating them at the end of the algorithm could be used for the subsequent phases, meaning that imbalance distributions for phases could be compatible in generating a complete, balanced execution. If the idle resources of one phase are intelligently used for the subsequent phases, one could expect performance improvements. Nevertheless, the constrained strategy can be used to optimize the critical path at the distribution end. When having both nodes with and without GPUs, critical path tasks that better exploit GPUs would benefit from an uneven distribution concentrated on nodes with these accelerators. This is further shown in the next Chapter, which studies these cases with multi-phase applications.

This chapter's main contributions and results were published in ICPADS 2020 (NESI; SCHNORR; LEGRAND, 2020).

6 HETEROGENEOUS STRATEGIES FOR MULTI-PHASE APPLICATIONS

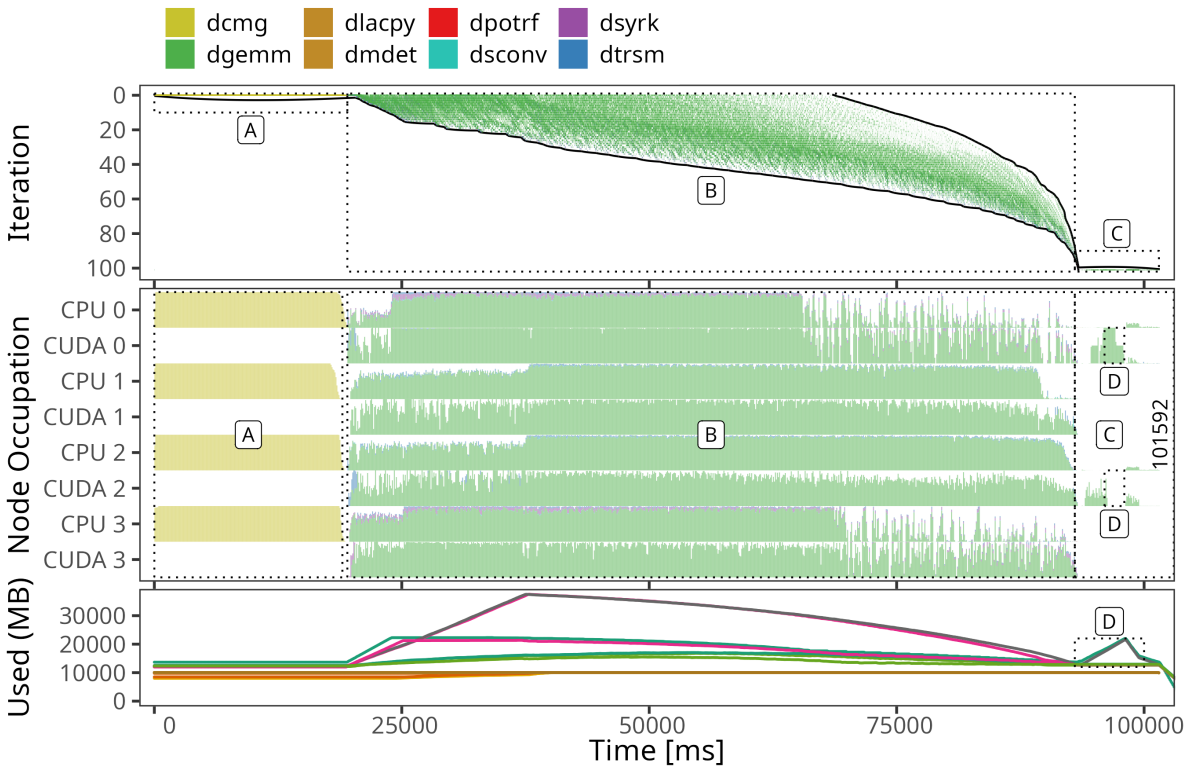
Many applications have different phases, each one having different computational needs. For example, phases comprising data input and generation are generally more suited to CPUs, while compute-intensive operations, such as classical linear algebra kernels, can efficiently exploit accelerators. This inner application heterogeneity can be better balanced when combined with the system heterogeneity, where more distribution options can improve load balance. Because the individual phases do not use all resources equally, an application requires certain freedom to execute the phases concurrently to improve resource usage and performance. The challenges to making phase overlapping possible include programming and algorithmic difficulties. From the programmer's perspective, phase overlapping is usually hard to obtain in traditional bulk synchronous parallel applications. However, in the task-based paradigm, certain asynchronism is seamless if the DAG is correctly structured. But even after the phase overlapping gets implemented, the application would still face the algorithmic challenge of finding an efficient distribution for the available nodes. Combining these challenges makes most applications miss an enormous opportunity to use system-level heterogeneousness. The distinct computational demand of phases makes system heterogeneity an interesting choice to improve load distribution.

This Chapter studies and proposes strategies for distributing these multi-phase applications over resources with system-level heterogeneity. The investigation uses the ExaGeoStat and Diodon applications, discussed in Section 2.4. First, Section 6.1 presents the problems and opportunities when considering multi-phases and the possible multiple distributions. Then, Section 6.2 presents optimizations to improve the task-based asynchronous execution of both applications' phases. In both applications, the same ideas of improving specific DAG structures like priorities, hints, and dependencies lead to performance gains. In ExaGeoStat, five operations overlap, while in Diodon, there is a potential overlap between the reading phase, the Gram operation, and the first matrix multiplication. With the correct overlap, this Section continues and offers strategies for generating a heterogeneous system-level static distribution that considers the phases' computational cost and overlap. The Section also provides an algorithm to infer a distribution of a particular phase by using the following phase distribution, maintaining the regional balance and reducing redistribution overhead. All these strategies are evaluated in Section 6.3, with many different system-level heterogeneous scenarios. Finally, Section 6.4 ends this Chapter with a discussion.

6.1 Problem: Asynchronous Multi-phase distributions

The first step to understanding the opportunities for improvement is to characterize the application's behavior. To this end, we use visualizations from the performance analysis tool StarVZ (GARCIA PINTO et al., 2018). Figure 6.1 presents three panels for one iteration of the synchronous version of ExaGeoStat. In all panels, the X-axis is the time in milliseconds. A Gantt chart is the middle panel, showing the aggregated utilization per resource type per node (for example, CPU 0 is the aggregated utilization in % of all CPUs in node 0). The makespan is the number position on the right end. The upper panel shows the computational intensity of each iteration of the Cholesky factorization on the Y-axis, with generation tasks mapped to 0 and post-factorization mapped to the last subsequent iteration. Two supplementary lines show each iteration's beginning (left) and end (right). This plot depicts how the factorization unfolds and gives a signature to the execution. The last panel shows the resource memory utilization per memory node, where RAM and each GPU have a memory node.

Figure 6.1 – Iteration, Node occupation, and Memory panels for the synchronous version of the ExaGeoStat iteration



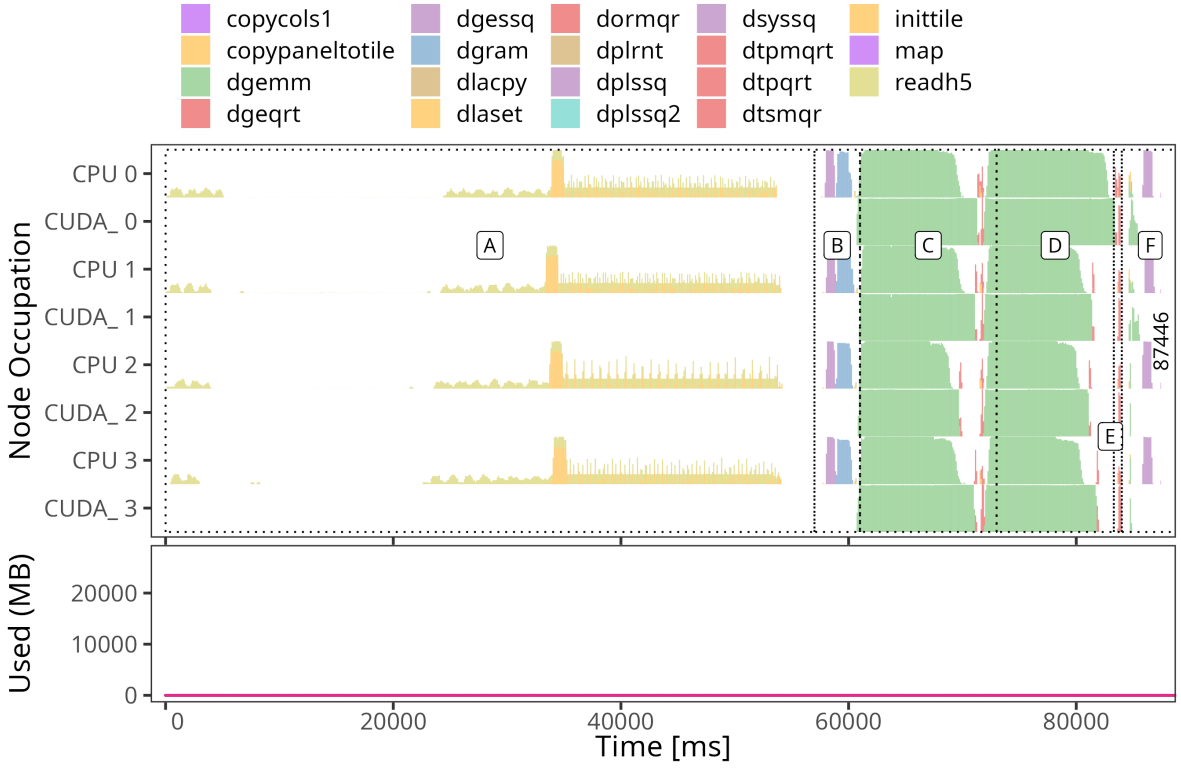
Source: The Author.

In the Figure, there is a distinction between the main three phases of ExaGeoStat. The yellow area (A), in the beginning, reflects the `dcmg` tasks of the *generation* phase that only

run on CPUs because the Matérn covariance function relies on the modified Bessel function whose implementation is complex and which is not available for GPUs. The (B) mostly green area represents the *Cholesky factorization* with its predominant `dgemm` tasks. Finally, the (C) area is the *post-factorization* operations (determinant, solve, and dot product, whose dominant operations are the `dgemm` tasks (in green) from the solve. Considering the whole execution, the resource usage can be improved, especially at the beginning (where only the CPU cores work because of the generation constraint) and toward the end (where there is not enough work for all nodes). Yet, the DAG should enable large amounts of parallelism that could shift a lot of green tasks to the beginning of the execution and improve the execution time.

Diodon's behavior shares similarities with ExaGeoStat. Figure 6.2 presents the StarVZ visualization (similar to the ExaGeoStat one) of this application behavior using the same four homogeneous nodes. The figure also clearly shows all Diodon's phases: the read phase with the yellow IO tasks and the orange `copypaneltotile` tasks (A annotation); the `sum of squares` (purple) and the `gram` tasks (blue) (B); the first and second pair of matrix multiplication (`dgemm`) in green and QR factorization tasks in red (C and D); the external untraced SVD call (E); and the post-processing operations (F). The resource utilization is small in the read phase, as the

Figure 6.2 – Node occupation, and Memory panels for the synchronous version of the Diodon complete execution



Source: The Author.

number of reading workers is limited for IO reasons. Because there is a synchronization point between all phases, GPUs remain completely idle until the matrix multiplication phase starts, very late in the application. Finally, the SVD and post-processing operations for this particular workload size are minimal.

In both applications, the lack of overlap between phases makes the application miss an enormous performance opportunity: powerful resources (GPU) could work more. With the correct execution of the generation phase, earlier GPU tasks could become ready sooner, increasing their utilization. Nevertheless, when the overlap is corrected, the phases have different needs that could exploit different sets of nodes to improve performance. This heterogeneity means that multi-phase distributions must consider this overlap, and a particular phase's distribution must acknowledge the performance of other phases' tasks in the heterogeneous node's resources.

6.2 Multi-phase Partitioning in Heterogeneous Clusters

This Section presents the strategies to partition multi-phase applications load correctly into system-level heterogeneous resources. Before aiming for heterogeneous resources, we had to improve the asynchronous execution of the phases. Section 6.2.1 presents a categorization of phase overlap optimizations that can be applied to task-based applications, following a set of detailed improvements carried out for both applications. With such optimizations, Section 6.2.2 presents the strategies to compute the relative load distribution power per node, considering all major phases into heterogeneous partitions. This power information is the input for traditional heterogeneous distributions. Finally, Section 6.2.3 shows how to compute a distribution for a phase while minimizing communication, using the following phase distribution as a reference.

6.2.1 Improving Application's Phase Overlap

Applications with synchronous phases lose an opportunity to adapt their workload and increase resource usage. In ExaGeoStat, the factorization could start as soon as the matrix upper part is generated and exploit the GPUs while the generation proceeds. A similar situation happens in Diodon since the Gram and matrix multiplication phases can proceed with only a portion of the data. This way, our first optimization removes the synchronization points between all internal operations and makes a **fully asynchronous** execution. This optimization allows StarPU to control the tasks' order and flow. However, a correct phase overlap requires

extra information passed to the runtime, like priorities and the leverage of non-computational operations, including large amounts of communication and allocation costs.

Although these hints and collateral effects may vary depending on the algorithm and application structure, they share the same goals and principles. We globally organize these optimizations into the following categories. **[I] Algorithm asynchronism optimizations** to improve the number of parallel operations and remove possible synchronization points in the program. **[II] Memory optimizations** to improve memory allocation and utilization between phases and the best time to do those operations. **[III] Communication optimizations** to reduce data transfers between phases and enable the earlier start and correct overlap. **[IV] Scheduling hints and artifacts optimizations** to tweak and give information to the scheduler (like task priorities considering the whole application) to maximize phases' overlap and induce the earliest execution of the critical path. The rest of the Section presents each one of the optimizations on each application, indicating their categories.

Considering the **ExaGeoStat** application:

Local solve algorithm [I, III]: The behavior depicted in Figure 6.1 (D annotation) shows an increased memory consumption with associated idle times. Further investigation revealed that a collateral effect of the solve phase caused it. The original Chameleon's solve algorithm performs its `dgemm` operation on the node that owns the solution vector (the right-hand vector) that follows the distribution of the main matrix. Consequently, many matrix blocks are moved between nodes to complete a simple `dgemv` operation, demanding extra allocation and communication between nodes. The number of allocations and communication, especially in a non-optimal order, will cause idle time, which can be considerable depending on the machines' configuration and matrix size. The solution was to replace the Chameleon solve algorithm with a version (Algorithm 9) that improves locality by performing all possible `dgemv` operations locally (that the node already has the data). This is done by accumulating its outputs in a local vector **G** of each node and communicating only it to perform a reduction on **Z**.

Algorithm 9: Local solve algorithm

```
1 for k = 0 up to nb do
2 |   dtrsv(M_(k,k), Z_k)
3 |   for i = k + 1 up to nb do
4 |   |   dgemv(M_(i,k), Z_k, G_(i,node of M_(i,k)))
5 |   end
6 |   foreach updated G_(k+1,j) do
7 |   |   dgeadd(G_(k+1,j), Z_(k+1))
8 |   end
9 end
```

Memory optimizations [II]: In an asynchronous environment, the maximum memory consumption and operations will not be the individual phases but a mixture of all. This mixture can cause problems. We perform four optimizations to mitigate memory management issues by changing runtime functionality alongside the application's DAG and tasks. The first optimization is to transfer the RAM allocation from the task submission function and perform it as other allocations, as a management task in the DAG. The second optimization is to enable cache for the StarPU's chunk memory system when using RAM, causing possible chunk reuse between phases and iterations. The third optimization is related to a slow memory allocation provided by the CUDA API, which GPU workers can originally perform. To maximize GPU throughput, we disable such operations on GPU workers. The last optimization is pre-allocating some memory chunks during application initialization so the first iteration can also use them as a cache.

New priority equations [IV]: One of the possible hints the application can pass to the runtime is priorities, which will play a role in the scheduler decisions. The original Chameleon implementation defined priorities for the Cholesky Factorization tasks ranged from $2n$ to $-n$ with an order following roughly the anti-diagonal. Other operations did not specify priorities, which StarPU interprets as 0. That means that non-defined priorities procedures, like the generation or solve, had effectively priority 0, all preferably executing in the middle of the factorization. For this reason, we propose new priorities for all tasks, considering all phases to guarantee a smoother transition between them all. Such priorities follow the natural DAG critical path with a unit execution cost (i.e., starting from the last tasks and going backward to the first generation's tasks). The new priorities equations for each task are in Equation 6.1, which considers a task that writes on a block of coordinate (i, j) on iteration k. Because the Cholesky DAG is essentially the basis, the generation priorities align with the first factorization iteration ($k = 0$), and to force an earlier execution, the negative component is divided by 2. Because the other operations (determinant and dot product) are leaves of the DAG and can be interchanged without any consequence, they are attributed a priority of 0.

$$
\begin{aligned}
\text{[Generation] } \texttt{dcmg} &= 3nb - \tfrac{i+j}{2} \\
\text{[Cholesky] } \texttt{dpotrf} &= 3\,(nb - k) \\
\text{[Cholesky] } \texttt{dtrsm} &= 3\,(nb - k) - (i - k) \\
\text{[Cholesky] } \texttt{dsyrk} &= 3\,(nb - k) - 2\,(j - k) \\
\text{[Cholesky] } \texttt{dgemm} &= 3\,(nb - k) - (j - k) - (i - k)
\end{aligned}
\tag{6.1}
$$

$$\text{[Solve] } \texttt{dtrsm} = 2\,(nb - k)$$
$$\text{[Solve] } \texttt{dgemm} = 2\,(nb - k) - i$$
$$\text{[Solve] } \texttt{dgeadd} = 2\,(nb - k)$$
$$\text{[Determinant] } \texttt{dmdet} = 0$$
$$\text{[Dot] } \texttt{dgemm} = 0$$

Submission order [IV]: Although the scheduler will try to follow the priorities order, in practice, an artifact can occur and lead to a low-priority task execution before a higher one. Because schedulers cannot foresee the future, tasks will start immediately if the application submits a low-priority task with idle resources. In this way, if the following task submitted has a higher priority, it will have to wait for the previous one to finish. This problem can be easily overcome by submitting tasks following their priorities. Because this artifact mostly appears in the generation phase, only this one had the order modified.

Oversubscribe [IV]: Another possible runtime artifact is the delay of high-priority tasks because of longer ones. If all workers have longer tasks, and a high-priority one becomes ready, it may have to wait for an available resource (without preemption procedures). This situation may be a problem if the task is part of the critical path and is responsible for releasing a lot of other tasks. This is the case of `dpotrf`, which can only execute on CPUs that may have to wait because generation tasks (`dcmg`) are much longer. Also, the `dpotrf` tasks are the ones that release many `dgemm` tasks in the Cholesky, tasks that can use powerful resources like GPUs. We use the idea of oversubscribing a CPU core with a dedicated worker for critical tasks over the core that StarPU dedicates to task submission. While tasks are being submitted, their computation will suffer from contention; however, the collateral impact is small relative to the possible gains of advancing the critical path faster.

Considering the **Diodon** application:

Memory optimizations [II]: Data allocation was also a problem at Diodon. Similar optimizations conducted in ExaGeoStat also apply here, including not making all allocations at submission time and turning off slow allocations on GPU workers and critical workers (the workers that perform the read). Another optimization was to pre-allocate the blocks for reading panels. In the default behavior, StarPU would make all allocations in nodes with CUDA GPUs, employing the CUDA API. This API pins the memory to accelerate future transfers. However, these allocations are slow. The read panels will never be transferred to GPU, as Diodon will first break those panels into blocks and redistribute them for the computational phase. This redistribution into other memory regions means that those panels are not required to be pinned.

Asynchronous Gram and Matrix multiplication workflow [I, III]: One of the original operations performed in the Diodon MDS is the Gram one. Diodon makes this operation by first computing the sum of squares to each row and the whole matrix and then applying the `gram` centralizing kernel. However, this situation implies a global synchronization point during the Gram operation as all `gram` tasks require the whole matrix sum of squares. Because the order of operations starts with the read followed by the Gram, and both do not use GPUs, these powerful resources would continue to idle until Diodon computes the whole matrix sum of squares, which requires reading the entire matrix. This synchronization implies a waste of GPU power during matrix reading. To overcome this problem, we propose a new Gram and Matrix multiplication workflow to improve parallelism and start the subsequent phases that can use GPU during matrix reading. We use associativity properties to break and modify the Gram and first matrix multiplication operations. The original Gram operation computes the Gramian block G_{ij} of the original matrix block D_{ij} using c (a constant coefficient), m (number of rows), n (number of columns), SR, SC, and SO, which are the sum of squares of the row, column, and the overall matrix respectively. Then, with a random generation matrix for the rSVD Ω, it performs the matrix multiplication $Y = G\Omega$. The following equations describe this operation in an expanded form for each block of the final Y matrix.

$$G_{ij} = c(D_{ij}^2 - \frac{SR_i^2}{n} - \frac{SC_j^2}{m} + \frac{SO^2}{nm}) \tag{6.2}$$

$$\begin{aligned} Y_{ij} &= \sum_k (G_{ik}\Omega_{kj}) \\ &= \sum_k (c(D_{ik}^2 - \frac{SR_i^2}{n} - \frac{SC_k^2}{m} + \frac{SO^2}{nm})\Omega_{kj}) \\ &= \sum_k (c(D_{ik}^2)\Omega_{kj}) + c(-\frac{SR_i^2}{n} + \frac{SO^2}{nm}) \sum_k (\Omega_{kj}) - c(\sum_k (\frac{SC_k^2}{m}\Omega_{kj})) \end{aligned} \tag{6.3}$$

The goal here is to perform the costlier ($O(n^2 r)$) operation $Y = G\Omega$ earlier. In this way, we reorganize the equations so that the first task only computes the matrix powering square G', followed by multiplying such matrix to omega $Y' = G'\Omega$. The difference of Y' to Y for each column j is $\frac{SR_i^2}{n}$ and $\frac{SO^2}{nm}$ multiplied by the j column sum of the matrix Ω and $\sum_k (\frac{SC_k^2}{m}\Omega_{kj})$. This way, after submitting the generation tasks for the matrix Ω, we submit tasks that will compute the sum of each column of Ω and store it in A_j. Because the generation of Ω and these summations are independent of D, such tasks can run during D's read while the system load is relatively low. We also submit tasks to compute the $\sum_k (\frac{SC_k^2}{m}\Omega_{kj})$ component for each column j and store

it in B_j. Those tasks will gradually execute as SC becomes available. Finally, we submit two new tasks, `gram_gemm_post`, which will correct the value of Y' (and become the original Y) by using A and B, and `gram_post` that will fix the value of G' to G.

$$G'_{ij} = c(D^2_{ij}) \tag{6.4}$$

$$Y'_{ij} = \sum_k (G'_{ik}\Omega_{kj}) \tag{6.5}$$

$$= \sum_k (c(D^2_{ik})\Omega_{kj})$$

$$A_j = \sum_k \Omega_{kj} \tag{6.6}$$

$$B_j = \sum_k \frac{SC^2_k}{m}\Omega_{kj} \tag{6.7}$$

$$Y_{ij} = Y'_{ij} + c(-\frac{SR^2_i}{n} + \frac{SO^2}{nm})A_j - cB_j \tag{6.8}$$

$$G_{ij} = G'_{ij} + c(-\frac{SR^2_i}{n} - \frac{SC^2_j}{m} + \frac{SO^2}{nm}) \tag{6.9}$$

Commutable tasks [I, IV]: Another interesting scheduler artifact is task dependencies that establish a strict task sequence that, in reality, can occur in any order. One example of such behavior is matrix multiplication. A final result block of coordinate (i, j) will be computed by the sum of the $M(i, k)N(k, j)$ blocks. This sum is commutable and can happen in any order. However, during submission time, the default behavior of StarPU is to create a dependency between a later submitted task that uses a block with `RW` (read and write) permission to any previous task that has used that block. When Chameleon submits the `dgemm` tasks from $k = 1$ to nb, the `dgemm` task of $k = 2$ will depend on the $k = 1$ task, when in reality, they can be commuted. The commutability notion is known and possible to StarPU; however, it requires that such tasks be submitted with the permission `RWC`. Such optimization a priori may appear small. However, in our case, without the optimization, the `dgemm` tasks of local M and Ns (already in the local memory of the node) with a larger k would need to wait for the communication of other M and Ns of lower k, reducing possible ready tasks and then resource usage. Thus, the optimization removes this unnecessary dependency, increasing the number of ready tasks.

Read phase distribution [III]: The original Diodon reads the matrix panels in a sequential node order, which means that for x nodes, the first one will receive a continuous $\frac{n}{x}$ area to read. Although this may benefit IO read purposes on specific disks, it can generate too many data transfers when reorganizing the blocks' distributions for the computationally intensive phases.

At this point, this optimization only matches the read phase distribution to the computation phase. However, in the next Section, when discussing heterogeneous nodes, this optimization is replaced with our read distribution calculated from the computation-intensive phases. The latter considers an ideal read distribution while reducing communications to the next phase.

New priority equations [IV]: In the same situation as ExaGeoStat, we have to improve the phases overlap by hinting to the scheduler the tasks that should execute first with priorities. This is important as the submission order differs from the ideal execution order. Because there will be an inevitable synchronization point before the `gram_post` operation, the priorities for this optimization are only defined for the following tasks: `read`, `copypaneltotile`, `gram_pre`, `dgemm`, and tasks related to the sum of squares. Because the generation reads the whole line of each `hdf5` file and the matrix multiplication operation requires a complete line of D to calculate each final block, we follow the order of the rows in all phases, i.e., the block index i of most operations. A high priority task `dlaset` for initializing the block is set to the highest priority n (number of rows). The priority for the matrix multiplication operation follows its intra-loop k, so it will aim to follow the left to right order of the matrix D for the available blocks. The complete list of priorities is on Equation 6.10.

$$
\begin{aligned}
\text{[Generation] } \texttt{inittile} &= nb - i \\
\text{[Generation] } \texttt{read} &= nb - i \\
\text{[Generation] } \texttt{copypaneltotile} &= nb - i \\
\text{[Computation] } \texttt{dlaset} &= nb \\
\text{[Computation] } \texttt{syssq} &= nb - i \\
\text{[Computation] } \texttt{gessq} &= nb - i \\
\text{[Computation] } \texttt{plssq} &= nb - i \\
\text{[Computation] } \texttt{gram\_pre} &= nb - i \\
\text{[Computation] } \texttt{dgemm} &= nb - k
\end{aligned}
\tag{6.10}
$$

The correct overlap of the phases in a smooth way is a requirement for better load distribution thought strategies that consider their interaction. These optimizations will help improve performance locally, and when using non-standard distributions, computational-intensive late DAG tasks with a higher priority can run earlier.

6.2.2 Load Balancing across Application Phases

When distributing asynchronous multi-phase applications into multiple node partitions, we must consider the phases' overlap. This situation happens because essential tasks from a further phase could and should start earlier and use idle resources from previous phases. The standard literature procedure to compute a distribution is to use the relative power of the machines involved, as shown in Section 3.1. In this context, the overlap length should be considered when computing the relative machine powers for a phase. Consider ExaGeoStat, for example; while the two main phases *generation* and *factorization* are relatively long, the generation cannot use GPUs. If the computation of the relative machines' powers for the factorization only considers itself, the actual distribution for GPU machines would be undersized, as *factorization* tasks can start earlier during *generation*. This means CPU-only machines would always work, but GPU machines can receive more load than the exact relative machines' power. Moreover, as powerful as the GPU is, the more load it can receive, alleviating work from CPU-only machines. A similar situation happens in Diodon, as the read phase is IO and performed in the CPUs. At the same time, the next matrix multiplication and Gram can occur asynchronously (the computational intensive phases). However, in Diodon, there will be an inevitable synchronization at the end of the `gram_post` tasks, as the whole matrix is necessary to finalize the operation. In this sense, the strategies of this Section for phase overlap only occur between the read phase, Gram operations, and the first matrix multiplication.

The proposed strategy is to use a linear program to correctly estimate the node groups' powers in task-based applications considering the interaction of two phases. Since phases overlap and depend on each other, the central concept is to divide the *phases* into virtual *steps* and relate the duration of each virtual phase step to dependencies and resource usage. The job scheduling Graham notation (GRAHAM et al., 1979) for the approximate problem is $Rm\,|\text{prec}; p_{ij} \in \{p_j, \infty\}|\,C_{\max}$. Where there are m unrelated machines (in our case, general resources like CPUs and GPUs) and tasks have arbitrary durations p_{ij}, and sometimes not all machines can execute all tasks $p_{ij} \in \{p_j, \infty\}$. Some jobs have precedences; in the same step, we model the dependency between phases. Finally, the goal is to minimize the overall makespan of the application.

The linear program model uses the following notations. The task *type* t corresponds to the different tasks in the application (`dgemm`, `dcmg`, `dgram`). A virtual step s represents a set of approximately independent tasks. In ExaGeoStat, the first phase is the generation, and we decide that each generation step will correspond to an anti-diagonal in the matrix (all the

blocks of coordinates i and j such as $s = \frac{i+j}{2}$), which corresponds to the priorities in Equations of the previous Section. For the second phase, a factorization task of step s was created by a generation task of the same step s. In Diodon, we consider the matrix's natural read order, and each step is composed of the total number of parallel read tasks (based on the maximum of workers that can perform them, in this case, five) and the computationally intensive tasks they release (Gram and matrix multiplication ones). In this way, considering that each row has rb read blocks, and bs rows are grouped into nb blocks, a read block of coordinate i ($1 \leq i \leq nb$) and j ($1 \leq j \leq rb$) is on step s such as $s = \frac{i \times rb + j}{5}$, as five is the number of parallel read workers used in the experiments. Because the generation of the matrix Ω is very low cost and can be done mainly during the first read step, we do not add it to the LP calculation.

The linear programming model takes as an input $N_{s,t}$, the total number of tasks of type t at step s, and their respective processing duration. A resource group r corresponds to, for example, all CPUs of a homogeneous set of nodes. We denote by $w_{t,r}$ the duration that a task of type t takes on resource group r (tasks that cannot execute on a given resource r have their cost set to infinity $w_{t,r} = \infty$). Finally, we will denote possible resources, steps, and task types by $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$. The sets $\mathcal{P}_1$ and $\mathcal{P}_2$ denote the task types of each phase and are disjoint.

The fractional number of tasks $\alpha_{s,t,r}$ of type t of step s placed on resource group r is the main output of the program. Dependencies among phase are accounted by introducing the variables $\tau_s^{(1)}$ and $\tau_s^{(2)}$, which represent the ending times of step s at the first and second phase respectively. The following linear program approximates ExaGeoStat and Diodon behavior, where $\alpha_{s,t,r}$, $\tau_s^{(1)}$, and $\tau_s^{(2)}$ are all positive fractional variables:

$$\text{Minimize} \sum_{s \in \mathcal{S}} (\tau_s^{(1)} + \tau_s^{(2)}) \; s.t. : \tag{6.11}$$

$$\forall t \in \mathcal{T}, \forall s \in \mathcal{S} : \sum_{r \in \mathcal{R}} \alpha_{s,t,r} = N_{s,t} \tag{6.11a}$$

$$\forall s > 1, \forall r \in \mathcal{R} : \tau_{s-1}^{(1)} + \sum_{t \in \mathcal{P}_1} \alpha_{s,t,r} w_{t,r} \leq \tau_s^{(1)} \tag{6.11b}$$

$$\forall s \in \mathcal{S}, \forall r \in \mathcal{R} : \tau_s^{(1)} + \sum_{t \in \mathcal{P}_2} \alpha_{s,t,r} w_{t,r} \leq \tau_s^{(2)} \tag{6.11c}$$

$$\forall s > 1, \forall r \in \mathcal{R} : \tau_{s-1}^{(2)} + \sum_{t \in \mathcal{P}_2} \alpha_{s,t,r} w_{t,r} \leq \tau_s^{(2)} \tag{6.11d}$$

$$\forall r \in \mathcal{R}, \forall s \in \mathcal{S} : \sum_{z \leq s, t \in \mathcal{T}} \alpha_{z,t,r} w_{t,r} \leq \tau_s^{(2)} \tag{6.11e}$$

$$\min_{r \in \mathcal{R}, t \in \mathcal{P}_1} (w_{t,r}) \leq \tau_1^{(1)} \tag{6.11f}$$

The application makespan is given by the last phase two (computationally intensive) ending time $\tau_{nb}^{(2)}$. While the general goal of these optimizations is to reduce application execution time, the objective function for the linear program in Equation 6.11 is more complicated. If the LP used a simple loose objective function like $\tau_{nb}^{(2)}$, the ending of the previous phases' steps $\tau_s^{(2)}$ for $s < n$ could appear as late as possible when the first phase is the bottleneck, which is undesirable. Instead, the objective function minimizes the sum of all $\tau_s^{(1)}$ and $\tau_s^{(2)}$ inducing a simultaneous minimization of all steps endings. In our experiments, giving more weight to $\tau_s^{(2)}$ or adopting a recursive minimization failed to bring any practical improvement compared to this simple sum.

The Equations 6.11a to 6.11f refer to the constraints of the linear program, and their individual goals follow. Equation 6.11a guarantees that the exact number of tasks are used in the resources. The dependencies by consecutive steps inside the same phase are approximated by Equation 6.11b, which states that one phase will end after the previous one ends, plus its own tasks. The next constraint refers to the dependencies between phases. Equation 6.11c guarantees that a step from phase two cannot end earlier than the same step from phase one and its own tasks. Equation 6.11d is similar to Equation 6.11b and enforces that the end of phase two will be after the end of the previous phase plus its tasks. This rule is stricter in the model than in reality because of the completely asynchronous execution; many iterations of the algorithm (that share multiple steps) can be executed in parallel. However, because they essentially share the same cost, it ensures the correct progression between phases without penalizing it too much. To guarantee that two tasks do not overlap in the resources, Equation 6.11e states that for each resource r, phase two ends at a step s be at least the sum of all previous tasks on that resource. The last constraint, Equation 6.11f, improves the approximation in the earlier stages of the execution. As the linear program adopts rational variables, the model allows the "split" of tasks between resources. In ExaGeoStat, the duration of the best implementation of the first phase tasks will be the minimal time for the first step of phase one. In Diodon, because of the step definition and the guarantee that each read step will execute at maximum x parallel tasks, as we have only x parallel workers, this rule can be restricted to $\forall s \in \mathcal{S} : \tau_s^{(1)} + \min_{r \in \mathcal{R}, t \in \mathcal{P}_1}(w_{t,r}) \leq \tau_{s+1}^{(1)}$.

Although the linear program has many constraints, it is fast solved in all our cases. The final duration result remains an excellent approximation and a lower bound even if some rules do not entirely represent reality (Equation 6.11d strictness, for example). In addition, the α variable is an excellent indicator of the number of tasks each resource group should execute in each phase, the information we will use to compute the relative power per machine per phase. Even if α is a fractional number, the relative power computed using it will be a good approximation.

Finally, the distribution algorithms try to approximate their final output concerning the informed power values, meaning that α does not need to be the precise number of tasks for each resource.

The relative power of each node for each phase is an input for most distribution algorithms. The computation of power p_{xy} of node x and phase y uses as input α from the LP, the actual number of tasks in the problem $\sum_{s \in \mathcal{S}} N_{s,t}$, and a task type weight (h_t) to adjust for the most significant tasks. The computation of power of each phase should only consider task types t of that particular phase $\mathcal{T}^{(y)}$. Equation 6.12 shows how to compute p_{xy}.

$$c_t = \frac{1}{\sum_{r \in \mathcal{R}} \frac{1}{w_{t,r}}} \sum_{s \in \mathcal{S}} N_{s,t} \tag{6.12a}$$

$$h_{ty} = \frac{c_t}{\sum_{z \in \mathcal{T}^{(y)}} c_z} \tag{6.12b}$$

$$p_{xy} = \sum_{r \in \mathcal{R}^{(x)}, t \in \mathcal{T}^{(y)}, s \in \mathcal{S}} h_{ty} \frac{\alpha_{s,t,r}}{N_{s,t}} \tag{6.12c}$$

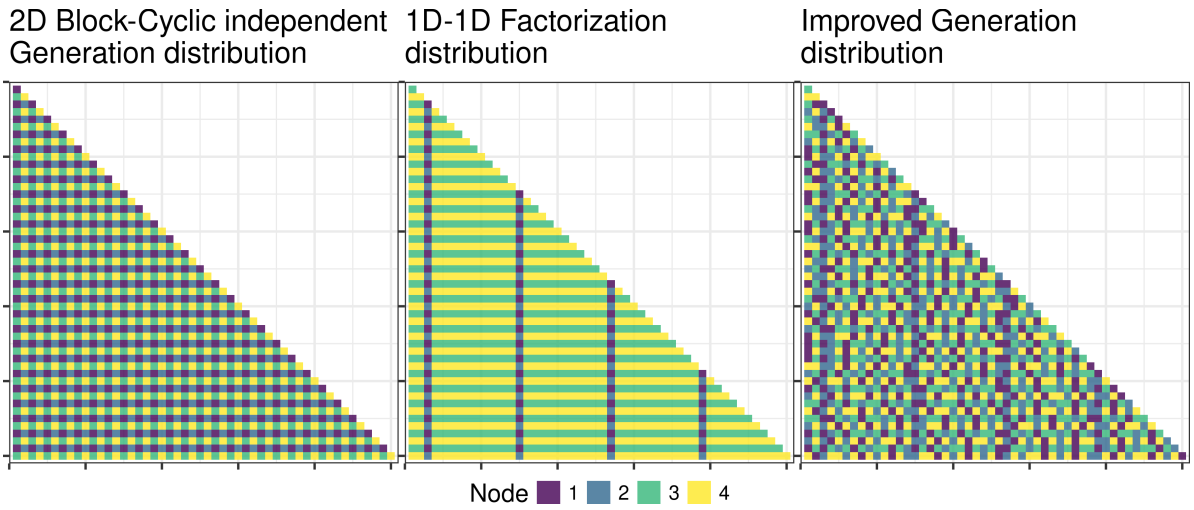
The weight h_{ty} uses c_t, the time to complete all tasks of type t using all resources. The component $\frac{1}{\sum_{r \in \mathcal{R}} \frac{1}{w_{t,r}}}$ express the duration needed to complete one task of type t if it could be hypothetically split in all resources. The weight h_{ty} is then the normalization of c_t, and the normalized values are greater for the task type that requires more time to fully complete. Finally, the power is computed by summing the normalized amount of tasks placed ($\frac{\alpha_{s,t,r}}{N_{s,t}}$) on any resource of that node ($\mathcal{R}^{(x)}$) for each task type, multiplied by the weight of that type.

6.2.3 Multi-Partitioning for distinct phases

Ideally, each phase has a different distribution, as each has a different computational need and resource affinities. In ExaGeoStat and Diodon, while the main tasks of the computationally intensive phase are the `dgemm` that GPUs can accelerate, the main tasks in the first phase (data generation) only execute on CPUs. The linear program output can derive the ideal computation load for each phase. Consider a simple situation with ExaGeoStat with four nodes with the same CPU, but two have GPUs. When considering the problem individually per phase, while the generation distribution will be equally divided, the factorization would mainly use the two faster nodes. Figure 6.3 shows a possible data distribution for the generation in the left (a simple 2D block-cyclic distribution) and for the factorization in the middle (a 1D-1D distribution).

In ExaGeoStat, the 1D-1D distribution ensures a well-balanced factorization with a minimal amount of communication. In Diodon, the problem can be relaxed to a 1D distribution.

Figure 6.3 – Generation and Factorization distributions for two nodes (1, 2) without and two (3, 4) with GPUs



Source: The Author.

However, because of the ratio of work in Diodon, assigning the last rows of the matrix to CPU-only machines during the computational intensive phase would create a huge critical path, as $\frac{r}{bs}$ tasks would need to execute sequentially with the traditional algorithm (other algorithms would require extra memory and a reduction operation). For this reason, we used the idea of a constrained distribution, as discussed in Chapter 5, when analyzing distributions for individual operations. We only consider assigning a row to a node if the global amount of work divided by the resources (expected application duration time) is larger than the critical path (CPB) of assigning that row to that node. Considering the placement of the x row, starting from the last one, the CPB is easily computed as $\frac{r}{bs}$ times the best implementation of the `dgemm` task on that node plus the LP result for the read phase x. At the same time, the amount of work divided by the resources can be estimated by the LP result of the computationally intensive phase at the last step $\tau_{nb}^{(2)}$. We always consider nodes for assignment with the lowest CPB plus a tolerance of two times the number of blocks in milliseconds. This heuristic handles the case for very small workloads and cases where the CPB is marginally different for the nodes. This idea of constraining the end of the distribution can also be applied to ExaGeoStat. We restrict the placement of the last rows of the Cholesky distribution to GPU-only nodes (or a group of fast nodes) while the CPB of an iteration is greater than the area-bound estimator (ABE). These methodologies are much simpler and slighter than the one presented in Chapter 5; however, they solve the problem of the slow and long critical path when there are heterogeneous nodes and multi-phases.

When building the phases (data generation and computationally intensive phases) distributions, if each one is computed independently, the chance is that most of the locations will be

different, requiring more communication during the redistribution between phases. For example, consider ExaGeoStat with a 50×50 matrix and the scenario of Figure 6.3, using the optimal independent partitions results in the communication of 890 blocks between the generation and the factorization phase, i.e., 70% of the total number of blocks.

In these conditions, each node in the data generation should receive roughly the same number of blocks, as the CPU is the same. When considering the phases overlap and possible interference, the linear program states an ideal of [318, 319, 319, 319] blocks per node in the generation and [60, 60, 565, 590] for the factorization. These differences mean that the first two nodes should transfer 517 blocks to the latter ones. This redistribution with 517 communications would be the minimum possible, i.e., 373 ($\approx$42%) fewer transfers than when distributions are independent. Not only are the quantities important for the generation, but the ideal distribution would also maintain it "cyclic", just like the 1D-1D distributions, ensuring that the first generation blocks are spread and processed in parallel over the nodes. This cyclicality helps increase the number of earlier computational intensive tasks ready, as in the factorization, this order matters. In Diodon, such distributions between phases have the same problem of extra communications.

Redistributing the two distributions in both applications can incur extra communication overhead. To minimize it, we propose Algorithm 10, which receives a computationally intensive distribution (a 1D-1D factorization distribution for ExaGeoStat, or a 1D heterogeneous cyclic distribution for Diodon, without or with the constrained rule) and a ideal number of blocks per node in the generation load. The output is a distribution for the generation that respects the factorization cyclically, minimizing the redistribution communication cost.

The algorithm's general idea is to iterate over the computationally intensive phase distribution, computing the current ratio of blocks each node should give or receive by the total number of blocks it has at the moment. The owner of the block changes if a node has more blocks than it should, giving it to the neediest node. If a node has twice as many blocks as it should have, its base ratio is two, and at every two blocks that the algorithms pass through that owner, one block moves to the neediest node. Since the computational intensive distribution in both cases is uniformly spread over the nodes, this cyclic update also ensures a uniform node spread of the generation but respects processing speeds. Figure 6.3 (right) presents the final distribution yield. This distribution minimizes the communications while aiming to reach the ideal number of blocks per node. It is possible to see similarities between this distribution and the factorization one, particularly the vertical stripes for nodes 1 and 2 and the horizontal stripes for nodes 3 and 4.

Algorithm 10: Generation of a target (data generation) distribution ($dist_{(2)}$) from a source (computational intensive) distribution ($dist_{(1)}$)

Input: $dist_{(1)}[1...mb][1...nb]$
$db_{(2)}[1...p]$ *Desired number of blocks per node*
Output: $dist_{(2)}[1...mb][1...nb]$
$db_{(1)}[1...p] \leftarrow$ Number of blocks per node in $dist_{(1)}$
$diff \leftarrow db_{(1)}$ - $db_{(2)}$
$rates$, $base \leftarrow \frac{db_{(1)}}{diff}$
$count[1...p] \leftarrow (0, ..., 0)$
$dist_{(2)} \leftarrow dist_{(1)}$
foreach *(i,j) in (*$1...m$, $1...n$*)* **do**
    $node \leftarrow dist_{(1)}[i, j]$
    if *diff[node] > 0* **then**
        $count[node] \leftarrow count[node] + 1$
        if *count[node]* $\geq$ *rates[node]* **then**
            $neediest \leftarrow$ which.min($diff$)
            $dist_{(2)}[i, j] \leftarrow neediest$
            $diff[neediest]{+}{+}$
            $diff[node]{-}{-}$
            $rates[node] \leftarrow rates[node] + base$[node]
            if *diff[neediest] > 1* **then return**
        end
    end
end

The computation of the ideal number of blocks per node for the first phase comes from the LP result. In ExaGeoStat, the number of tasks is straightforward from the α values. In Diodon, because we use single dimension distributions, we get α, compute the relative power, and apply a 1D distribution again (with the number of blocks as the number of lines, even in situations where there will be more tasks because of the reading grid) and count the number of tasks (rows) per node.

6.3 Performance Evaluation

This section evaluates the strategies for improving phase overlap and multi-phase heterogeneous distributions using solely real experiments. The commits for the software stack used are the following.

ExaGeoStat has the main branch commit `9518886` containing HiCMA, Chameleon, and Stars-H in the corresponding submodules. StarPU developer branch commit `015357bd`, and the NewMadeleine (DENIS, 2019) (the communication layer) main branch commit `51d3bf40`. We

added in ExaGeoStat and Chameleon some functions to load custom distributions. All the phase overlap modifications are dynamically enabled or disabled during execution time. ExaGeoStat authors made available a list of workloads[1], for which we selected three synthetic ones identified by numbers 8, 9, and 10 with $n = 57600$, $n = 96600$ and $n = 122500$, and a subset of workload 27 with $n = 172800$. These present the best workload ratio to our computational power. The evaluation uses 960 as block size and, as a consequence, receives a matrix size (nb) of 60×60, 101×101, 128×128, and 180×180 blocks. These numbers (60, 101, 128, and 180) are used to identify each workload.

Diodon experiments use the official main branch version commit `4e6027e2`. The communication library NewMadeleine is set to master branch commit `51d3bf40` while StarPU is `0fb603d8` and Chameleon `6f185f1`. The hdf5 is the 1.10.7 version. Diodon and Chameleon have modifications for the proposed optimizations and accept heterogeneous distributions. Finally, the workload consists of synthetic datasets generated by computing the distance of n dimension random points with sizes 48k, 64k, and 128k. The execution environment uses a block size of 640, resulting in workloads of 75×75, 100×100, and 200×200 blocks.

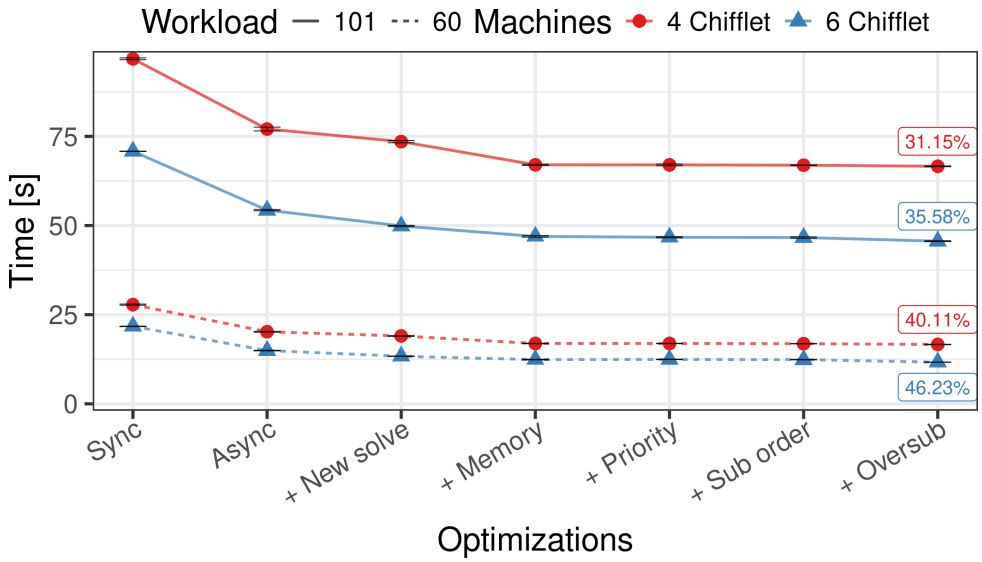
6.3.1 Improving ExaGeoStat Phases Overlap

The phase overlap strategies in ExaGeoStat were evaluated using two workloads (60 and 101), two sets of machines (four and six Chifflets), and ten repetitions. Figure 6.4 depicts the results. The X-axis is the cumulative optimizations enabled (from left to right), while Y-axis is time in seconds with a 99% confidence interval. The percentage values in the far right represent the total gain with all optimizations compacted to the synchronous version. The results indicate that the first three strategies (full asynchronous, new solve algorithm, and memory optimizations) were the ones that brought more improvements. The priority and submission optimizations caused minor improvements in the 101 workload or no improvement in the 60 workload. However, $\approx$10% improvements were further seen in heterogeneous scenarios. The last optimization, over-subscription, brought a slight yet consistent improvement in all cases. All these optimizations represent a total gain from 31% in the 101 workload with four machines up to 46% in the 60 workload with six machines when compared to the synchronous version. The rest of the Section presents a more comprehensive analysis of how these optimizations changed the behavior and improved performance.

The execution visualization of three different optimization cases is present in Figure 6.5

[1] <https://ecrc.github.io/exageostat/md_docs_examples.html>

Figure 6.4 – Performance comparison of our phase overlap improvement strategies against the synchronous version of ExaGeoStat

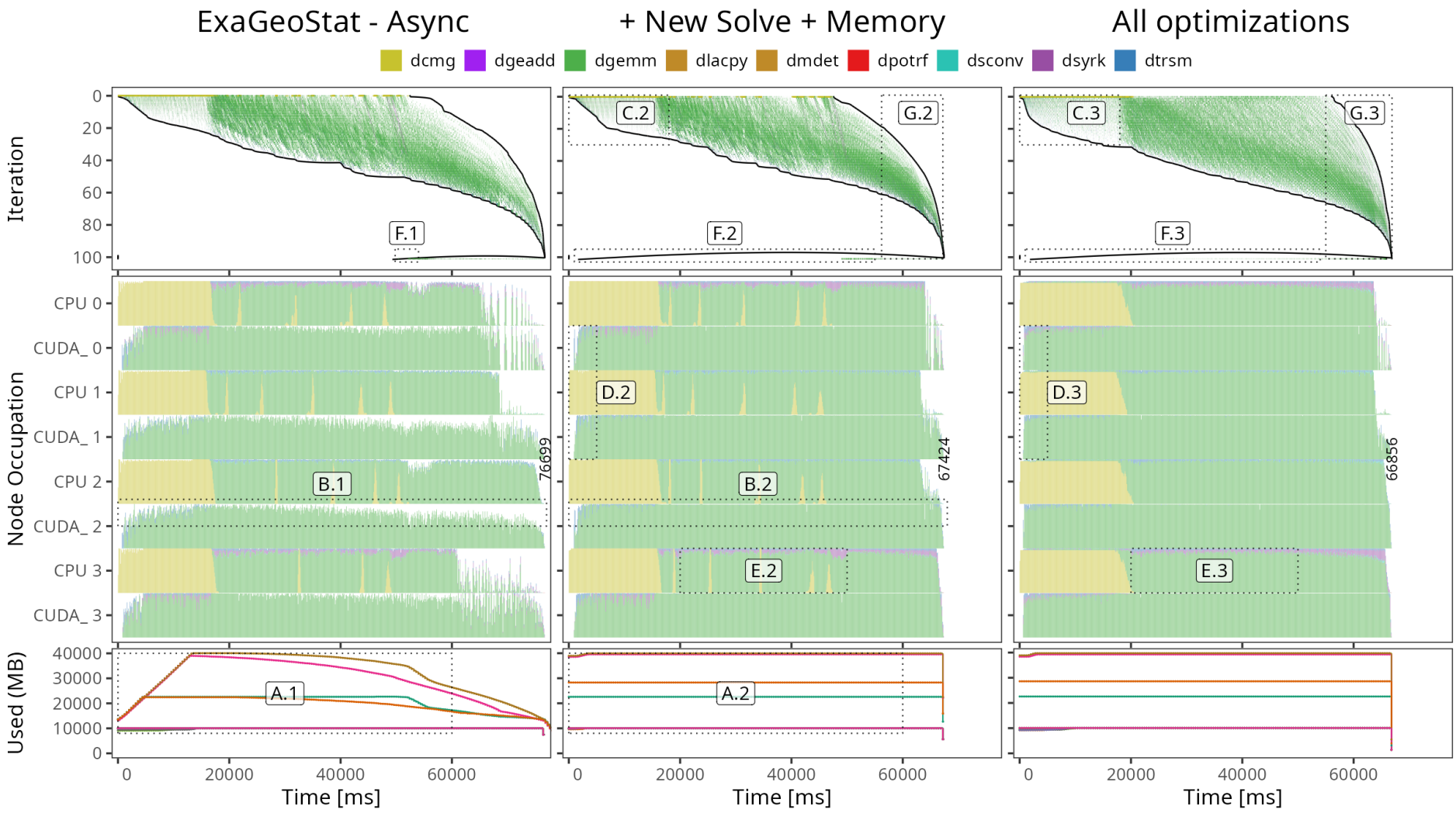


Source: The Author.

for the four machine case with the 101 workload. In the left panel (`Async`), there is the first case **full asynchronous**. The second one in the middle panel (`+ New Solve + Memory`) has two extra optimizations concerning the first one, the **new solve algorithm** and the **memory optimizations** strategies. The last one in the right panel (`All optimizations`) contains all the optimizations. When comparing the simple asynchronous execution with the earlier Figure 6.1 that is the synchronous one, the synchronizations barriers disappear, meaning that factorization tasks (green) are executed in GPUs alongside the generation ones (yellow) in the CPU. However, the simple asynchronous case on the left still has improvement potential, as idle time is present at the beginning of the execution and during the solve phase at the end. As discussed earlier, those problems may be related to communication and memory utilization, even in the asynchronous case. This assumption is confirmed when applying the first optimizations (the new solve algorithm and memory optimizations) present in the center panel of Figure 6.5. There is no gradual memory consumption (allocations are expensive) when comparing annotation A.2 to A.1. Also, resource utilization is almost at 100%, as shown by comparing annotation B.2 to B.1. The total amount of communication, computed from traces, is reduced from 11044MB in the asynchronous version to 8886MB with the New Solve optimization.

The right execution of Figure 6.5 also shows the differences in the behavior of the last three optimizations (Priorities, Submission order, and Over-subscription) when compared to the middle one, though with minor performance gains. The first difference is that the factorization iteration parallelism is higher, as seen in annotations C.3 to C.2, indicating a faster start,

Figure 6.5 – Cholesky Iteration, Node occupation, and Memory utilization panels using 4 Chifflet for one ExaGeoStat iteration in three cases: Asynchronous, Async + New solve + Memory optimizations, All optimizations



Source: The Author.

advancing the critical path sooner and releasing more tasks. This situation is further confirmed by comparing annotations D.3 to D.2, which shows the corrections of the slow start and the full utilization of the resources much earlier. The second main difference is that generation tasks run at the beginning of the execution, without misplaced tasks in the middle, as shown in annotations E.3 with E.2, a result mainly from the defined priorities. A third difference is the start of the late phases. As seen in F.1 to F.2, the solve can start when it's ready; however, most solve tasks in F.2 occur after the middle of the execution. With the priorities, most of the F.3 region tasks occur gradually during the execution. Finally, annotations G.2 and G.3 present the behavior at the end of the execution. While in G.2, there is a more gradual ending, in G.3, with all optimizations, there is an abrupt closure, indicating that it prefers to delay some tasks in the first iterations.

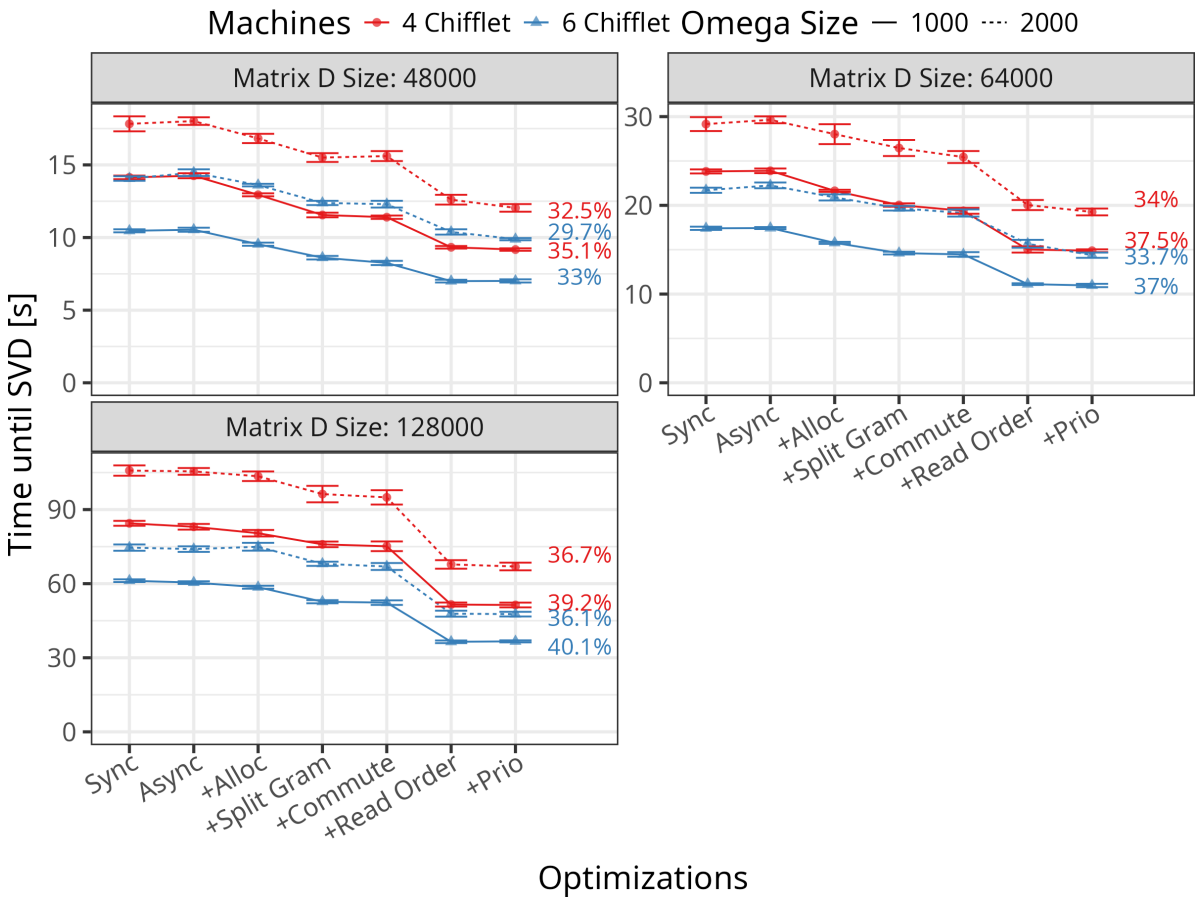
Another metric that indicates the strategies improvements was resource utilization. This metric is the application's complete tasks time divided by the working time of all resources. Runtime overhead is considered idle time, i.e., time not spent in the application. The total resource utilization for each of the three executions is 87.74%, 95.78%, and 96.83%, respectively. Also, when only considering the first 95% of the ExaGeoStat iteration, the metric is 93.01%, 99.55%, and 99.55%. This result indicates that the last possible improvements are at the end of the application when parallelism diminishes and working in the critical path becomes more important (NESI; SCHNORR; LEGRAND, 2020). All the results here indicate that the strategies were able to improve phase overlap in ExaGeoStat and increase resource utilization. These improvements are essential for the multi-phase distribution over heterogeneous systems.

6.3.2 Improving Diodon Phases Overlap

In the Diodon case, Figure 6.6 presents the evaluation of the optimizations, and it consists of using three different workload sizes for the matrix A (facets) and two different sizes for the matrix Ω (line shape). The optimizations are present on the X-axis and the makespan until the SVD is on the vertical axis. Two groups of machines were used, four (red) and six Chifflets (blue). The percentage in the right part presents the final improvement compared to the original sync version. The optimizations improved the performance from at least 29% to up to 40% in these cases.

The following behaviors of individual optimizations are pertinent. An asynchronous flow without overlap consideration impaired the performance in most cases. While the allocation, split Gram, and commute optimizations improved performance by the same amount, a considerable

Figure 6.6 – Performance comparison of our phase overlap improvement strategies against the Diodon synchronous version



Source: The Author.

performance gain appears only when the read order optimization is enabled. We stress that this is caused not only by this optimization alone but also by combining the split Gram, commute, and reading order. Finally, the priorities harvest the last final gains and are more visible in large Ω sizes.

The optimizations modify the behavior of the execution. Figure 6.7 presents the workers' utilization of three different optimizations over four nodes' execution. The left case presents the original asynchronous execution. In this situation, the read phase had problems because of allocation (A.1), and it is possible to observe the hard algorithm synchronization point in the `gram` (blue tasks in B.1). The center case used the allocation and split Gram optimizations. The read tasks are now smooth (A.2), but the `dgemm` did not start earlier than it should be possible (B.2). This behavior results from a poor overlap because of the commute problem, extra communication, and lack of priorities. The right and final case presents the behavior using all optimizations, giving a clear execution and overlap of read and `dgemm` tasks (B.3). This last case shows a makespan of 18.6s compared to the asynchronous naive case of 29.9s. The beginning

of the execution still presents some idle time caused by allocations for the matrix multiplication matrices (A.3). However, even if the `dgemm` tasks could start earlier, the improvement would be limited, as there will be an inevitable algorithm synchronization in the `gram_post` tasks before the QR factorization. The distance between those tasks and the last read tasks is small (the critical path).

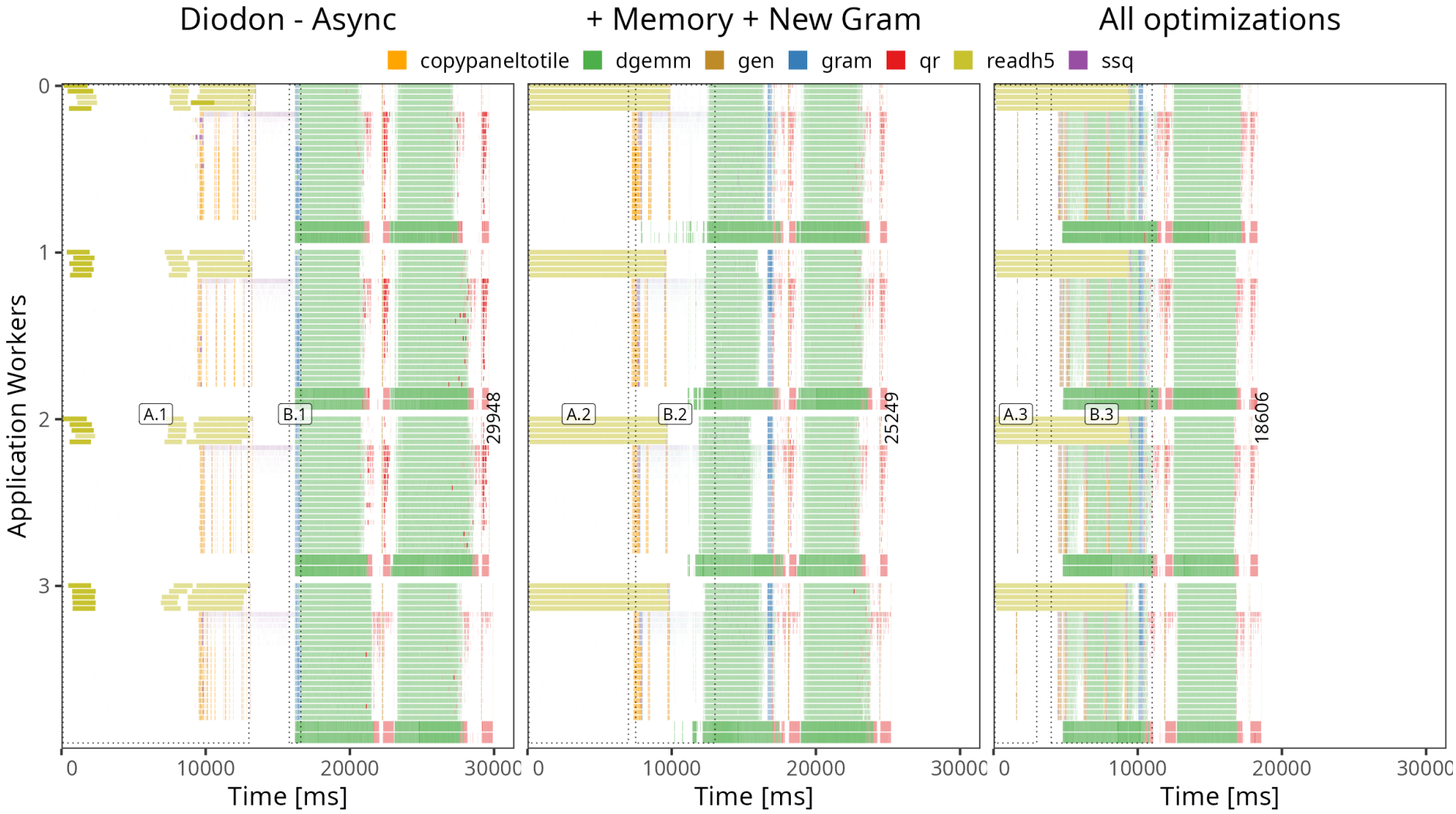
6.3.3 ExaGeoStat phases partitioning in heterogeneous clusters

This Section now considers the heterogeneous distributions' strategies. The experiments use workloads 101, 128, and 180 over 18 different sets of machines combining the available ones on each system, as depicted by the panels of Figure 6.8. These sets of machines demonstrate different system heterogeneity levels. Each panel shows the makespan in seconds (Y-axis) as a function of the distribution strategy (X-axis and colored bars). The first three bars of each panel are our baselines: (1) the homogeneous block-cyclic distribution using all the resources (*BC on All* in red); (2) the homogeneous block-cyclic distribution for the fastest homogeneous subset of nodes (*BC on fast* in blue), as which we consider the standard in the community; and (3) the heterogeneous 1D-1D distribution using the powers of machines computed considering the `dgemm` speed (*1D-1D* in green). These baselines always use the same distribution for the factorization and generation phases. Our proposed distributions, using the linear program and the workflow to compute two distinct distributions for each phase with the factorization distribution using the normal 1D-1D (*LP HetDist* in purple) or the 1D-1D constrained distribution (*LP HetDist Constrained* in orange). The orange bar also presents an inner white bar to represent the ideal makespan obtained by the linear program. The fastest group of machines used is usually the individual fastest homogeneous subset of nodes (used in the block-cyclic distribution in blue). However, in cases with only one Chifflot, the single machine cannot perform this workload well because of high GPU memory utilization. So, for these cases, the *BC on Fast* result indicates the usage of the second powerful partition.

Figure 6.8 shows that the block-cyclic distributions are never the best result, neither using all the resources (red) nor the fastest homogeneous subset of nodes (blue). The linear program distribution (purple and orange) performs very well. Compared to the homogeneous cases (*BC on Fast*), using the LP is always beneficial: the most significant gain is 69.8% on the 4 Chiclet + 1 Chifflot 101 case, while the lowest gain is 6.5% on the 6 Chiclet + 4 Chifflot 101.

Compared to the single heterogeneous distribution (1D-1D), using the LP is most of the time beneficial or ties, but there are a few situations (6 Che + 6 Chi + 1|2 Cho) where it slightly

Figure 6.7 – Node occupation using 4 Chifflet for Diodon: Asynchronous, Async + Allocation + Split Gram, All optimizations



Source: The Author.

degrades performance. This last situation occurs because the gains by correctly balancing the load are small, and the imbalance compensates for the differences when phases overlap in the heterogeneous nodes. Also, when only using one distribution, there is no cost for redistribution. The minimal difference between the ideal execution time obtained by the linear program and the actual makespan for the good cases shows that the redistribution communication overhead is completely overlapped. However, when using the three levels of heterogeneity with the Chifflot

Figure 6.8 – ExaGeoStat makespan for homogeneous and heterogeneous distributions in 18 machine sets configurations

Source: The Author.

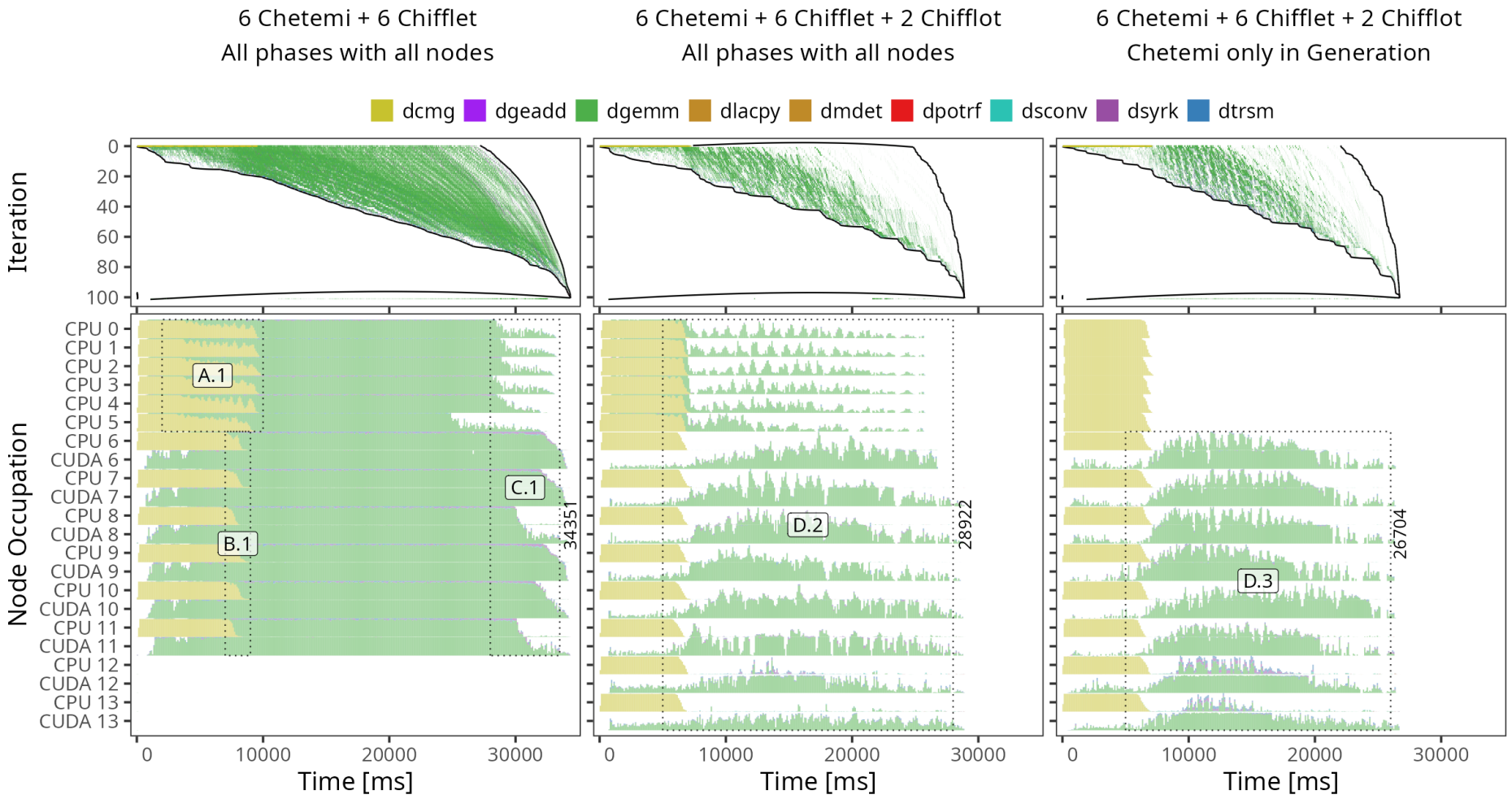
nodes (the cases where the LP distributions are worse than the pure 1D-1D), the execution time is much larger than the LP solution. Further investigation indicates that the problem is the communication of the fast nodes with the slow ones and the ratio of work and power in this configuration. The following section provides further insight into this situation.

6.3.4 Analysis of a case when using too many fast nodes

Using more nodes may not always improve performance. Sometimes, it can impact negatively. Figure 6.9 presents the iteration (top) and Node occupation (bottom) panels for three different cases: 6 Che + 6 Chi (left), 6 Che + 6 Chi + 2 Cho using all the resources in the factorization (center), and 6 Che + 6 Chi + 2 Cho using only nodes with GPUs in the factorization (right). Both the execution with two types of machines and the one shown in Figure 6.5 (right) have low idle times. Moreover, the transition in Figure 6.9 (left) from generation to factorization is more smooth in CPU-only nodes (A.1 annotation) and then in GPU nodes (B.1 annotation). In GPU nodes, the high-priority tasks will execute on the fastest resource, while in CPU, they begin to share the resources with some generation tasks. In this execution, the effect of the constrained version is visible, as the work ends earlier in the CPU-only nodes (C.1).

When adding two Chifflot nodes (represented in the very bottom of the center of Figure 6.9), the P100 GPU computes the `dgemm` task $10\times$ faster than the Chifflet nodes, adding extra heterogeneity, and the need for faster communication. While the overall makespan decreases, considerable idle time is visible (D.2 annotation). Further investigation revealed that communication along the critical path and the small workload for this set of machines are responsible for such high idle times. One problem is that the factorization load is highly different in the nodes; the two very fast nodes receive most of the computation and are helped slightly by the slower nodes. A lot of communication is necessary to propagate all the computation done by the GPU nodes, and even with their faster network, it's not enough, mainly because these nodes share a different subnet in the Lille site. This problem is also why cases 4+4+2, 6+6+1, and 6+6+2 of Figure 6.8 have a distant LP prediction to the result. Another evidence for this network problem is that the pure 1D-1D uses a single distribution across all phases and then requires less communication, trading distribution balance for communication improvements. One possible technique to circumvent the communication problem is limiting the number of nodes during the factorization, which is the phase causing most communication operations. We can easily set such a limitation by excluding the nodes without GPUs from the factorization in the LP constraints. The case in the right of Figure 6.9 depicts the resulting behavior. The idle time

Figure 6.9 – Cholesky Iteration, Node occupation, and Memory utilization panels of the ExaGeoStat iteration using the LP HetDist Constrained distribution for three sets of machines: 6+6, 6+6+2, and 6+6+2 restricting factorization to GPU-only nodes



Source: The Author.

decreases (D.3 annotation), indicating a better usage of the nodes, leading to a slight decrease in the makespan with fewer resources. This case provides a mean makespan of ≈26s. There remains a difference of 50% between the actual makespan and the LP result, so enhancements in communication or a better selection of machines for this workload should improve this even further. Overall, comparing this result against the original synchronous 2 Chifflot homogeneous execution (≈41s), we have a performance improvement of 41%. This result reinforces the necessity of having strategies to decide the correct number of nodes considering communication and, after this inquiry, using such distributions on the correct number of nodes. This will be the topic of Chapter 7.

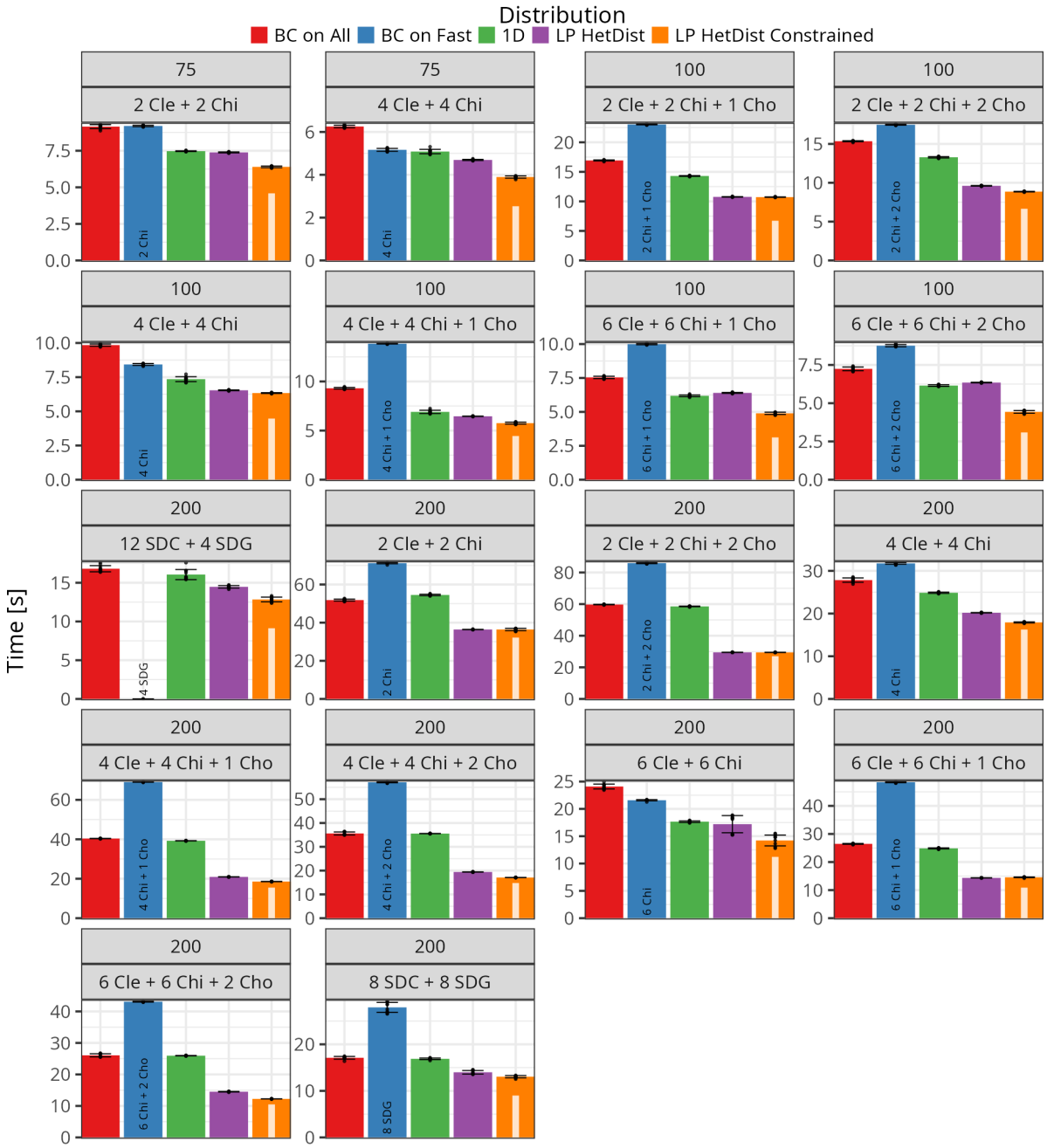
6.3.5 Diodon phases partitioning in heterogeneous clusters

This Section now considers Diodon and the heterogeneous distributions' strategies. The experiments use the workloads with 75, 100, and 200 blocks width (nb) for matrix D and $r = 1000$. Figure 6.10 shows the makespan (the elapsed time between the application start to the last `gram` task) in 18 distinct scenarios of machines with five different distributions.

Once again, the first three bars of each panel are our baselines: 1) *BC on All* uses all the available machines on that configuration but with the traditional block-cyclic distribution (red); 2) *BC on Fast* shows the case of using only the fastest nodes (with GPUs) for both phases with the original homogeneous distribution (blue); 3) The *1D* is the version with the heterogeneous distribution (with one dimension) that considers node heterogeneity for the factorization while the generation phase uses the standard round-robin distribution (green). In all these cases, the read phase does not require heterogeneous awareness as the relative read power is very similar across such nodes. Our proposed distributions are the *LP HetDist* (purple) and *LP HetDist Constrained* (orange). The *LP HetDist* version describes the case when using the LP to compute the ideal capacity of each machine for each phase, considering the overlap. It uses the 1D heterogeneous distribution algorithm for the computational intensive phase and the derivative fewer communication one for the read phase. The *LP HetDist Constrained* version is similar to the *LP HetDist*, but instead of the original 1D heterogeneous distribution for the computation phase, it uses the constrained version that will only assign rows to a node if the CPB is higher than the application expected duration. This situation means that the final rows will be constrained to the fastest nodes, accelerating the critical path. As before, the white bar inside shows the ideal makespan obtained by the linear program.

The results of Figure 6.10 show the improvements when using heterogeneous resources

Figure 6.10 – Diodon time to Gram operation for homogeneous and heterogeneous distributions in 18 machine sets configurations



Source: The Author.

and adequate distributions over them considering phases overlap. In most cases, using the baseline strategy *BC on Fast* presents the worst results, as the reduction in computational power, especially in the read phase, is considerable. The performance sometimes improves by adding different extra nodes using the original homogeneous distribution (*BC on All*). There are positive and negative cases when transitioning to heterogeneous independent distributions (*1D*). However, the performance improves in all cases only when using LP-based versions. In some cases, especially where the workload ratio to machines is lower (less workload per

node), our constrained version (*LP HetDist Constrained*) improves considerably. Also, one extra advantage of using more heterogeneous nodes is that they may handle larger problems in case the application fails to handle large workloads in fewer nodes. This is the situation of the case 12 SDC + 4 SDG, where the workload of 200 was unable to run using only 4 SDG. When comparing the *LP HetDist Constrained* version with the block-cyclic one on the fast nodes only, the greater performance increase was in case 4 Cle + 4 Chi + 1 Cho with 73.1%, and the lower performance increase was in 4 Cle + 4 Chi with 24.6%.

6.4 Discussion

As heterogeneity becomes even more prominent at a system level, when there are distinct nodes, correctly distributing multi-phase applications in such an environment enables them to exploit it and better accommodate their load. This Chapter presented optimizations for improving the overlap and strategies for distributing asynchronous multi-phase task-based applications over heterogeneous system-level resources. Two real scientific applications, ExaGeoStat and Diodon, were used. First, we demonstrate three general optimization categories that improve the overlap of asynchronous phases. These optimizations enable the latter phases' critical tasks to execute earlier and reduce resource idleness, enhancing performance on homogeneous scenarios from 29% to 46%. When considering the heterogeneous case, we present strategies that compute the ideal relative power for each phase on each group of nodes. A linear program models the application's flow considering tasks and resource heterogeneity. The relative power is extracted from the linear program and later used in traditional distribution algorithms for some computationally intensive phases. This Chapter also provides an algorithm to design a distribution of a previous generation phase that maintains data cyclically, reduces redistribution communications, and balances the load for that particular phase. This new distribution is distinct but tightly coupled to the next one. All those heterogeneous distribution strategies enhance the performance by up to 69% compared to a simple homogeneous setup in ExaGeoStat and 24% to 73% in Diodon.

Although these strategies provide a methodology to generate multi-phase distributions considering a given number of heterogeneous resources, the best case may not necessarily be the one with all nodes. The excessive use of nodes is expensive and not necessarily valuable, as performance usually deteriorates because of communication overheads or other unforeseen behaviors. At the same time, modeling when communication occurs exactly in a dynamic runtime is complex, especially with system-level heterogeneity involved. In this way, the next

Chapter considers the problem of dynamically determining the best set of heterogeneous nodes to use during runtime.

This chapter's initial contributions and results, focusing on the ExaGeoStat application, were published in ICPP 2021 (NESI; LEGRAND; SCHNORR, 2021), which received the best paper award. After, the overall contributions and results of the chapter were published in FGCS 2023 (NESI; LEGRAND; MELLO SCHNORR, 2023).

7 LEARNING AND ADAPTING IN COMPLEX HETEROGENEOUS SYSTEMS

This Chapter studies the problem of discovering the ideal set of heterogeneous nodes to use when considering multi-phase applications. Although the task-based paradigm largely mitigates communication overhead, unforeseen effects (e.g., network contention or complex inter-node synchronizations) remain possible and particularly hard to model. In this context, finding an adequate number of computational nodes for each phase can be particularly challenging to anticipate. Hence, methods that dynamically learn and adapt to such complex scenarios and improve performance over time are desirable. The ExaGeoStat application is a good candidate for studying such strategies. It has many iterations where decisions of increasing or decreasing nodes could take place. Many applications have this structure of stable iterations (stationary workload) but whose total duration is difficult to anticipate in some setups, as some phases scale well while others do not. That means the application can actively learn and adapt to the best set of heterogeneous nodes it can access between iterations. Remember that it is possible to inform the runtime about data movement during the submission of tasks, causing the following submitted tasks to change their execution node accordingly. These movements can reflect new distributions, which can be phase-oriented and use more or fewer nodes. The StarPU runtime will move all the data to the right place asynchronously, overlapping with computation.

The structure of this chapter follows. Section 7.1 analyzes the problem showing real cases where varying the number of nodes in just one phase is beneficial. Section 7.2 presents a series of possible candidates to handle this problem of deciding the number of nodes to use. Section 7.3 presents the experimental evaluation of the proposed strategies, including an overview of the results in 16 different scenarios, a detailed step-by-step analysis, the strategy cost overhead evaluation, and possible expansions of the proposal. Experimental results are gathered using real and simulation environments with system-level heterogeneous setups. Section 7.5 discusses the results, concluding the chapter.

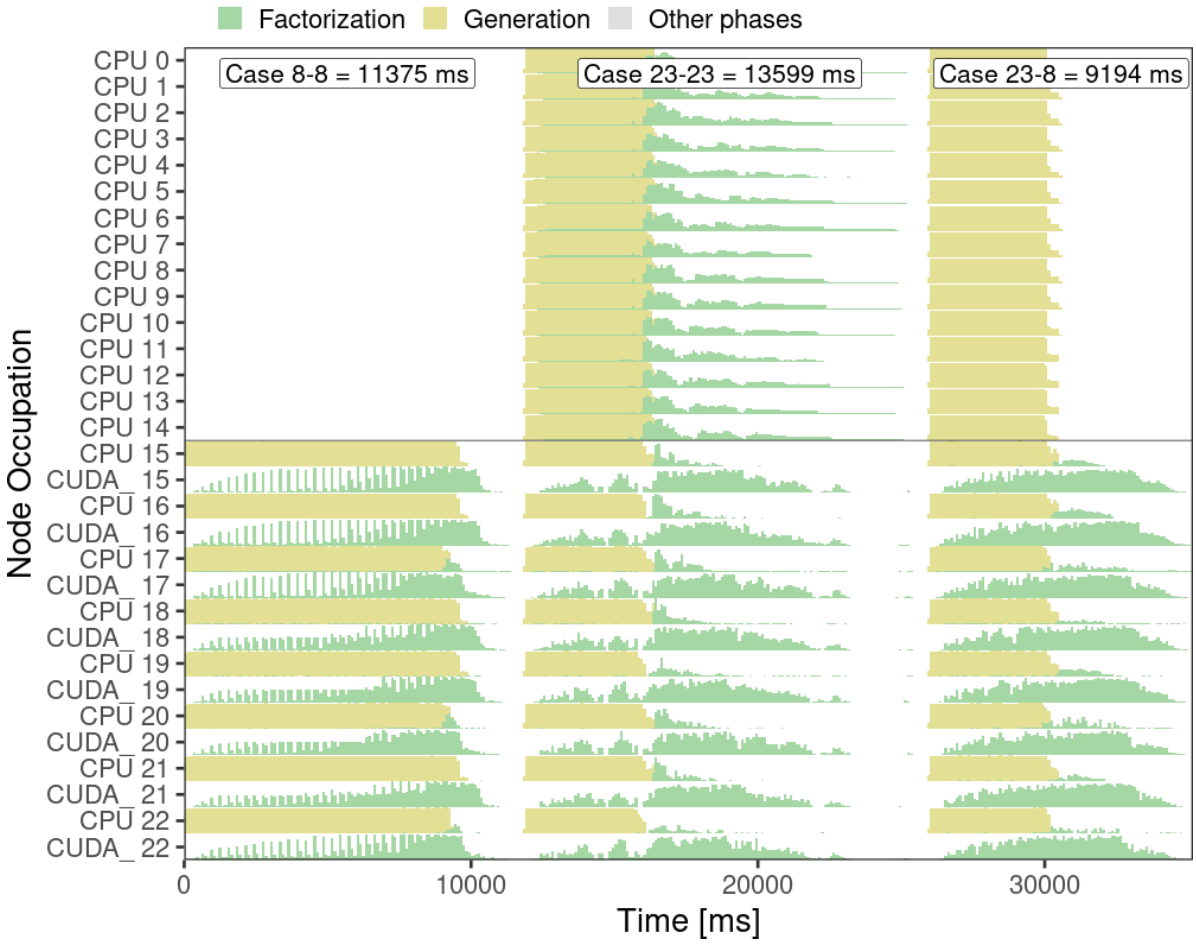
7.1 Problem: Varying Heterogeneous Nodes per Phase

ExaGeoStat's iteration phases have very different computational requirements and resource affinities. While generation only runs on CPUs, the Cholesky factorization can exploit GPUs to accelerate the application. Moreover, the factorization cost is stationary across iterations, as it is only based on the matrix size, and the generation cost is mostly constant between iterations when using adequate parameters. However, the distribution of multiple phases for a

given number of resources is not trivial, as using a different number of nodes for interleaving phases may cause unforeseen network contention and heterogeneous distribution problems. It is also unsatisfactory and costly to manually discover how such a complex application would behave for a certain number of nodes, hardware, and workload. Consequently, it is desirable that this application adapts to any HPC system without extensive analysis or complete executions of all possible configurations. Automatic adaptation for these setups is key to achieving portable performance.

Figure 7.1 depicts three iterations of ExaGeoStat where the X-axis is the time, and the Y-axis has the aggregated resource type utilization per node. The different colors correspond to different phases: the yellow ones are the generation, the green ones are the factorization, and a small number of tasks in gray correspond to the other three phases. Each iteration of Figure 7.1 uses a different number of nodes per phase. The notation of the title, Case A-B denotes the

Figure 7.1 – Three iterations of ExaGeoStat: the first using a eight homogeneous nodes for both phases (8-8). The second increasing the number of nodes (with CPU-only nodes) and using all 23 for both generation and factorization (23-23), the third restricting the factorization to the eight fast nodes (23-8)



Source: The Author.

number of nodes for the generation (A), and the number of nodes for the factorization (B). The first one only uses eight homogeneous nodes for both phases (8-8). The second iteration uses the Chapter 6 strategies and now adds 15 CPU-only nodes to speed up the generation (23-23). The generation phase ends earlier on the second iteration, but excessive communication and critical path problems slow down factorization. However, iteration three presents the best makespan, using all nodes for generation and only the eight faster nodes for the factorization operation (23-8). This execution uses the non-constrained distribution detailed in Chapter 6. For iterative multi-phase applications such as ExaGeoStat, it may be interesting to have nodes that will only be used for some phases, like the generation, and not the others.

Estimating the ideal number of nodes to use per phase is a complex process (NESI; LEGRAND; SCHNORR, 2021), and not only because adding more and more nodes is not the ideal situation. A perfect duration modeling would require anticipating the stochastic behavior of the scheduler, the network conditions, and the distribution's issues. It seems unfeasible to anticipate every possible condition. Figure 7.2 provides three representative examples, named cases (c), (i), and (p), of the ExaGeoStat iteration duration (Y-axis) depending on the number of factorization nodes it uses (X-axis). The application uses all the nodes in the generation step for all scenarios, as this phase is embarrassingly parallel. In the figure, the nodes in the X-axis are sorted by computational power, meaning that we always use the fastest nodes in the set. The nodes are organized in three categories, Large (L), Medium (M), and Small (S), depending on the computing capability of each category (the machines described in Chapter 4 are organized and presented on Table 7.1). The vertical black lines show when the machines' categories change. The dark blue line corresponds to the lower bound provided by the LP (of Chapter 6.2.2). The yellow and green vertical bars represent the duration of the asynchronous (the reason for the inner bar) generation and factorization phases provided by the LP. The jittered crosses represent actual measurements on those configurations, while the red line represents the mean of such measurements. In all cases, the granularity of the workloads is large enough to exploit parallelism in all nodes and present a similar resource usage behavior as in Figure 7.1.

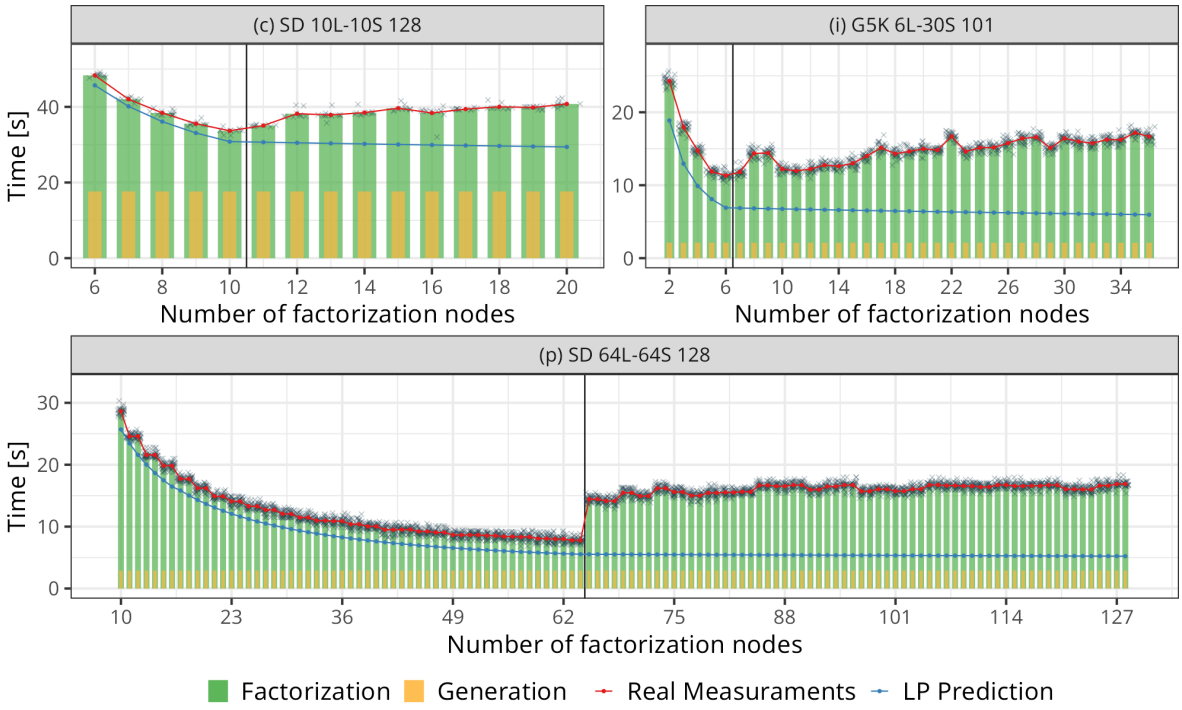
Table 7.1 – Computational nodes used in the performance evaluation

Site	S Machine	M Machine	L Machines
G5K	Chetemi	Chifflet	Chifflot
SD	B715	B715-GPU[1]	B715-GPU

Source: The Author.

[13] Identical to the L machine but with one GPU activated during the experiments.

Figure 7.2 – Behavior using different heterogeneous nodes setups (Table 7.1) by varying the number of factorization nodes



Source: The Author.

It is possible to observe that these are complex scenarios. In all of these cases, using all nodes for all phases is sub-optimal. The main behavior observed is that the addition of new nodes usually forms convex-like shapes. In the beginning, adding new nodes is beneficial by having more processing power. However, there is a point where adding new nodes is no longer useful, and there are some scenarios where the network could get overwhelmed. In these setups with a limited network, the performance starts to deteriorate. Furthermore, there are scenarios with significant breaks, usually related to the heterogeneity and the distribution shape. Sometimes, adding a slow node (especially CPU-only ones), like in scenario (p), creates a critical path that may degrade the overall performance. The observation noise is generally the same for all numbers of nodes, with few outliers. Some scenarios, like (i), have small breaks related to the distribution. Adding new nodes may cause the reorganization of the partition structure, creating more communications and synchronizations. Such cases demonstrate the problem of varying the number of nodes for a particular phase. Combined with the problem that such behaviors are challenging to model upfront, dynamic strategies would be desirable to identify the best case to use during application execution.

7.2 Proposal: Exploration Strategies Candidates

In the case of the studied application, ExaGeoStat, although computation phases can partially overlap thanks to the fine-grain dependencies expressed in StarPU, an iteration (the evaluation of the likelihood of a given value θ) cannot start before the previous one completes. At each iteration, the application may select a different subset of nodes. The optimization goal is to minimize the iteration duration. Instead of exploring all possible node permutations, it can choose n between 1 and N nodes to use and pick the n fastest nodes since trading a slow node for a fast one is always detrimental (when considering performance solely). Therefore, our search space consists of the number of nodes per phase. Although this search space is discrete, it is visible from Figure 7.2 that there is an underlying continuous structure. The variability from one iteration to another for the same configuration adds additional challenges, and the optimization strategy should be able to handle it. This section presents possible methods to explore and optimize such search space.

7.2.1 Naive Heuristics

For comparison purposes, we implement three simple naive heuristics. First, a simple divide and conquer dichotomy (**DC**) to carry out a recursive binary search over the space. At each exploration step, the search space is divided in two, and the middle point of each division is measured. The method selects the division with the lower makespan as the new search space and repeats. This simple heuristic will converge quickly and pick the correct point in simple low-variance curves. The second heuristic considers using all the machines as the best candidate. The heuristic walks from the rightmost option to the left, while the left point presents a lower measurement (**Right-Left**). The final heuristic is similar to the former but moves from the left to the right of the function (**Left-Right**). These two last methods would work when the curve is well-behaved (without huge discontinuities) and with a very small variability. However, excessive resources create extra overhead on the rightmost options.

7.2.2 Classical continuous minimization approaches

Another subset of strategies is the one for classical continuous optimization. The simplest algorithms use gradient, which is unavailable in our problem. Yet, the problem's search space

may have local minima. Since there is only one dimension in our context, a sensible choice is the **Brent** algorithm, which combines the bisection method, the secant method, and inverse quadratic interpolation. Such technique is a priori not robust to noise and does not revise their belief. It is also available on R's `optim` (R Core Team, 2022). We also tried multi-dimension algorithms like Nelder-Mead and BFGS with no better results. We also investigated Stochastic Approximation (CHEN, 2006) and Simulated Annealing (SANN from `optim`), but they achieved bad results because they are not parsimonious, so we refrained from reporting them.

7.2.3 Multi-armed bandits

Multi-armed bandits are a Reinforcement Learning framework that models K possible unrelated choices (SUTTON; BARTO, 2018). Each choice has its distribution and variance. The goal is to maximize its total reward, $\sum_{t=1}^{T} y(n_t)$, where $y(n)$ is the (stochastic) reward of step t (in our case, the opposite of the duration of an iteration when using n_t nodes). The difference between the cumulative reward from choosing the best action upfront is the regret. To minimize regret, one should balance exploration (to discover the best action) and exploitation (selecting the best action to improve the overall reward). A no-regret strategy (i.e., whose regret is $O(\log(T))$, the optimal bound) consists of using the Upper-Confidence-Bound (**UCB**) algorithm (AUER; CESA-BIANCHI; FISCHER, 2002) that selects the action that maximizes the mean empirical reward plus an upper bound that increases over time:

$$x_{t+1} = \underset{x \in A}{\operatorname{argmax}}\ \mu_t(x) + c\sqrt{\ln t / N_t(x)} \tag{7.1}$$

where $\mu_t(x)$ is the mean reward measured so far for action x, c is a adjustment constant, $N_t(x)$ is the number of times action x was selected. Best actions are often selected, while less promising actions are not taken frequently but have their upper confidence bound component increased so that they are eventually selected again after some time.

In our case, each action corresponds to a number of nodes. However, the fact that a similar number of nodes leads to similar performance is not exploited, making large setups have large search spaces requiring a long time to explore. To speed up the exploration, we consider a restricted version (**UCB-struct**) that will only look at multiple complete groups of homogeneous nodes. For example, in a setup with five machines of group A, five of B, and five of C, the only options are 5, 10, or 15 nodes. If the best action is outside these choices, it will never reach the optimal configuration.

7.2.4 Gaussian Process

The Gaussian Process (GP) (GRAMACY, 2020) or Kriging is a surrogate model that assumes a form of smoothness over the data. It uses a Multivariate Normal Distribution (MVN) to model possible realizations over observations. The assumption is optimizing a function f, which is unknown and assumed to have been drawn randomly from the set of smooth functions, $f \sim GP$. Furthermore, this is a noisy scenario, so $f(x)$ cannot be directly observed. The measurements are noisy observations $y(x) = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_N^2)$. Then, based on a set of t observations $D_t = \{x_i, y_i\}$ and given the smoothness of the function which stems from the GP assumption, it is possible to compute $\mu_t(x) = E[f(x)|D_t]$ and $\sigma_t(x) = \sqrt{Var[f(x)|D_t]}$ with standard kriging R libraries.

After the generation of the surrogate model, it can be used to select an action to take. This decision needs to consider the predicted mean of the location, its confidence interval, and the exploration and exploitation trade-off. A possible approach with the same kind of no-regret properties as UCB is to use GP-UCB (SRINIVAS et al., 2010) where the following equations (with β growing logarithmic with iterations) will give the best choice:

$$x_{t+1} = \underset{x \in A}{\text{argmax}}\ \mu_t(x) + \beta_t^{1/2}\sigma_t(x) \tag{7.2}$$

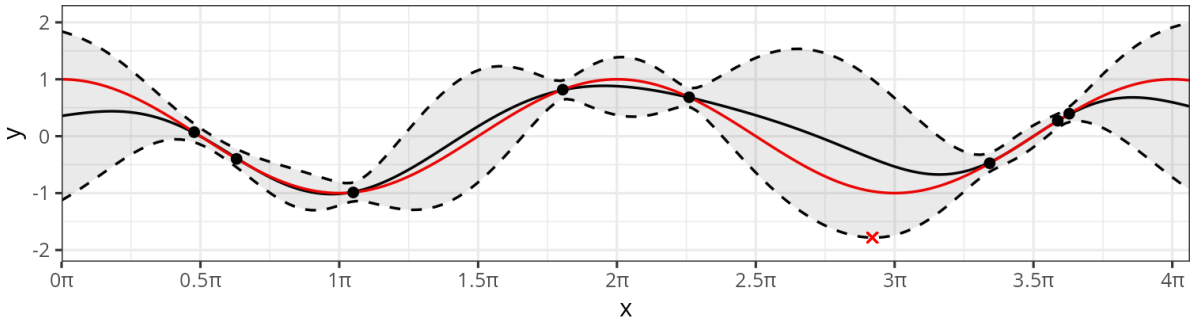
Figure 7.3 illustrates such a process. The red line is the cos function representing a true unknown function that GP wants to model. The black points are real random measurements taken in the function, which will be used as the GP's input. The black line is the predictive mean provided by the GP, while the gray area inside the dashed black lines represents the 95% confidence interval. The mean prediction is very close to the real function in the neighborhood of measured points. Coordinates that do not have measurements have more uncertainty, but the true cos function indeed lies in the 95% confidence region. The red cross is the next point to be evaluated, as it represents the most promising point while considering the current uncertainty.

The GP puts a probability distribution over smooth functions through a covariance function (typically the exp function), which is commonly parameterized as:

$$\Sigma(x, x') = \alpha\ exp\left\{\frac{-||x - x'||}{\theta}\right\} \tag{7.3}$$

It has parameters α, and θ that GP-UCB assumes are known. In practice, they are often estimated from the data with an ML approach (as in ExaGeoStat), but with little data, this can be a problem,

Figure 7.3 – An example of the GP fit with eight measurements over *cos* function



Source: The Author.

as, with bad luck, the algorithm may be overconfident.

In practice, when using GP and building surrogates, it is common to initialize the process with a uniform quasi-random design (e.g., LHS, maximin), where the x_i are uniformly spread over the space as it allows a reasonable estimation of the hyper-parameters. However, this would be too costly in our case, and we need a more parsimonious approach. Therefore, the model selects the actions of the first iterations following a simple procedure. The first iteration will always choose the action with N nodes, which is the application's default behavior. Using all the available nodes would provide the best performance in ideal circumstances. Then, to obtain as much information as possible, standard Bayesian optimization approaches would select the left-most point, i.e., the configuration that uses the minimal number of nodes, which may not be a good idea from an application makespan perspective. Finally, it selects the middle of these two points for the subsequent two iterations.

Any new measurement will bring some information about f but also about α, θ, and σ_N. Measuring two points next to each other provides mostly information about α and θ. While measuring the same location several times provides a lot of information about σ_N. In our case, for example, σ_N is estimated as follows. We consider $S = \{x \in D | N_t(x) > 1\}$, then σ_N^2 can be estimated by:

$$\frac{\sum_{x \in S} \sum_{y(x) \in D} (y(x) - \bar{y}(x))^2}{\sum_{x \in S} N_t(x) - 1} \tag{7.4}$$

The version so far described is **GP-UCB**, and it is reasonable when we do not know a priori the shape and the noise is roughly the same regardless of x. However, the f function may have discontinuities (see Figure 7.2(p)), and the functions are not entirely random as they have a general shape, more or less decreasing than increasing (see Figure 7.2).

The standard GP has no particular trend, which explains why in Figure 7.3, it naturally reverts to 0. If we know something about the shape of f, it should be put in the trend $\mu(x) =$

$\sum_i \gamma_i g_i(x)$ where g_i is a basis function and γ_i the coefficients. But again, the more unknown parameters, the more complex the model and the more measurements we will have to do to reduce uncertainty and explore instead of exploit. In our case, measuring two points far away from each other gives a lot of information about γ if we have a linear model.

This work exploits the properties of its problem scenario. First, the results indicate that the trend of the observations follows a pattern of $\frac{1}{x} + x$. This result follows the intuitive view that the makespan will decrease every time a new node is added. However, adding a new node may cause more communications, and therefore, there is an overhead associated with it. However, the $\frac{1}{x}$ parameters are not very useful as this part is already well estimated in the LP. Hence, the adopted approach is to use the LP as a baseline and use the GP to model the overhead with respect to the LP. In this sense, the trend will be linear x, as the LP captures the $\frac{1}{x}$ behavior. Second, bound the exploration with the LP results. Considering that the first iteration uses all the machines, some configurations with very few resources cannot provide better results when comparing a theoretical lower bound (LP) to the actual first iteration makespan. In G5K 6L-15S or SD 64L, the most left points are inevitably higher than using all nodes as predicted by the linear program. The method to find the most adequated left-point uses the linear programming bound and finds the lower n_l that satisfies $\mathrm{LP}(n_l) < f(N)$. The bound given by the linear program is optimistic and does not consider communications or the critical path. The approach excludes all the left points where their linear programming bound is higher than the real measurement of using all the nodes. This will limit the search space and avoid bad actions.

Finally, the model needs to be fixed because it is too smooth. A local broken continuity results in an artificially inflated estimation of the smoothness parameter α, increasing overall (global) uncertainty. Indeed, some scenarios present a discontinuity of performance when using a new group of machines. Although the GP-UCB will eventually explore all x after a very long time, a strong smoothness assumption may prevent finding the optimal configuration shortly. This situation is sometimes caused by abrupt changes in the partition distribution or because of the critical path. The new nodes may be so less powerful than the others that the few data blocks they receive may cause a synchronous critical path. Many other tasks may have to wait for it, causing a global slowdown. In the case of Cholesky, giving a block of coordinate (i, j) to a node will cause a synchronous sequence of $\min(i, j)$ `dgemm` tasks on that particular node. This can take some time if the node does not have GPUs. Similar to the case of Chapter 6 when using CPU-only nodes.

To model such discontinuities, we introduce dummy variables (JAMES et al., 2013) to the trend model for each group of homogeneous machines. In this way, the trend will be given

by $x + \sum_{g \in G} d_g(x)$ where $d_g(x)$ will be 1 if the node x (last node added) is present on group g and 0 otherwise. The dummy variable allows us to indicate where discontinuities may appear and generally allows it to pool, which is good for estimating parameters. However, this model may get overconfident when the number of measurements is low, given a bad estimate for θ. To overcome this, we set θ to 1 and α to the sample variance. Because this model adds new variables, it requires more initial points for the first fit. Then, each group (after the leftmost point) will have its last point measured once. If this point is already measured, we choose to evaluate the next point. The last group (using all the machines) is already measured and skipped. This is **GP-discontinuous**. Both GP versions are implemented using the R package DiceKriging (ROUSTANT; GINSBOURGER; DEVILLE, 2012).

7.2.5 Summary of Strategies

All presented strategies will behave differently in this particular problem. Table 7.2 presents a list of all discussed algorithms with expected behavior. Considering the technical aspects of the algorithms, our expectations are that **DC**, **Right-Left**, **Left-Right**, and **Brent** were not resilient to noise, and so can be misled by the measurements in this problem. **UCB** would require a full exploration of the search space, which can be bad in applications with few iterations, and because some configurations are particularly bad. **UCB-struct** will only search a fraction of the possibilities. In this way, if the best case is not one of them, it will never discover it. **GP-UCB** does not use the problem particularities and insights to improve its model, which leads to the exploration of bad configurations and lower confidence when finding discontinuities. **GP-discontinuous** is our proposed version that uses knowledge of the problem to solve some of the earlier issues discussed.

Table 7.2 – Summary of exploration strategies and expected behavior

Algorithm	Resilient to noise	Optimal	Fast
DC			×
Right-Left			×
Left-Right			×
Brent			×
UCB	×	×	
UCB-struct	×	(limited exploration)	×
GP-UCB	×	×	
GP-discontinuous	×	×	×

Source: The Author.

7.3 Experimental Evaluation

The experiments use ExaGeoStat on commit `9518886` of its public repository, with other software dependencies present as submodules of the same commit. The StarPU version is the main branch commit `015357bd` with the NewMadeleine library (DENIS, 2019) commit `d6542d72` and backend. The experiments use the 96100 (101×101 blocks) and the 122880 (128×128 blocks) matrices from the ExaGeoStat samples. First, this Chapter presents the behavior of the application on many configurations in Section 7.3.1, followed by a detailed evaluation of the GP versions step-by-step behavior in Section 7.3.2. Then, Section 7.3.3 presents the evaluation of all the candidate strategies. Section 7.3.4 discusses the case of expanding the method to more phases.

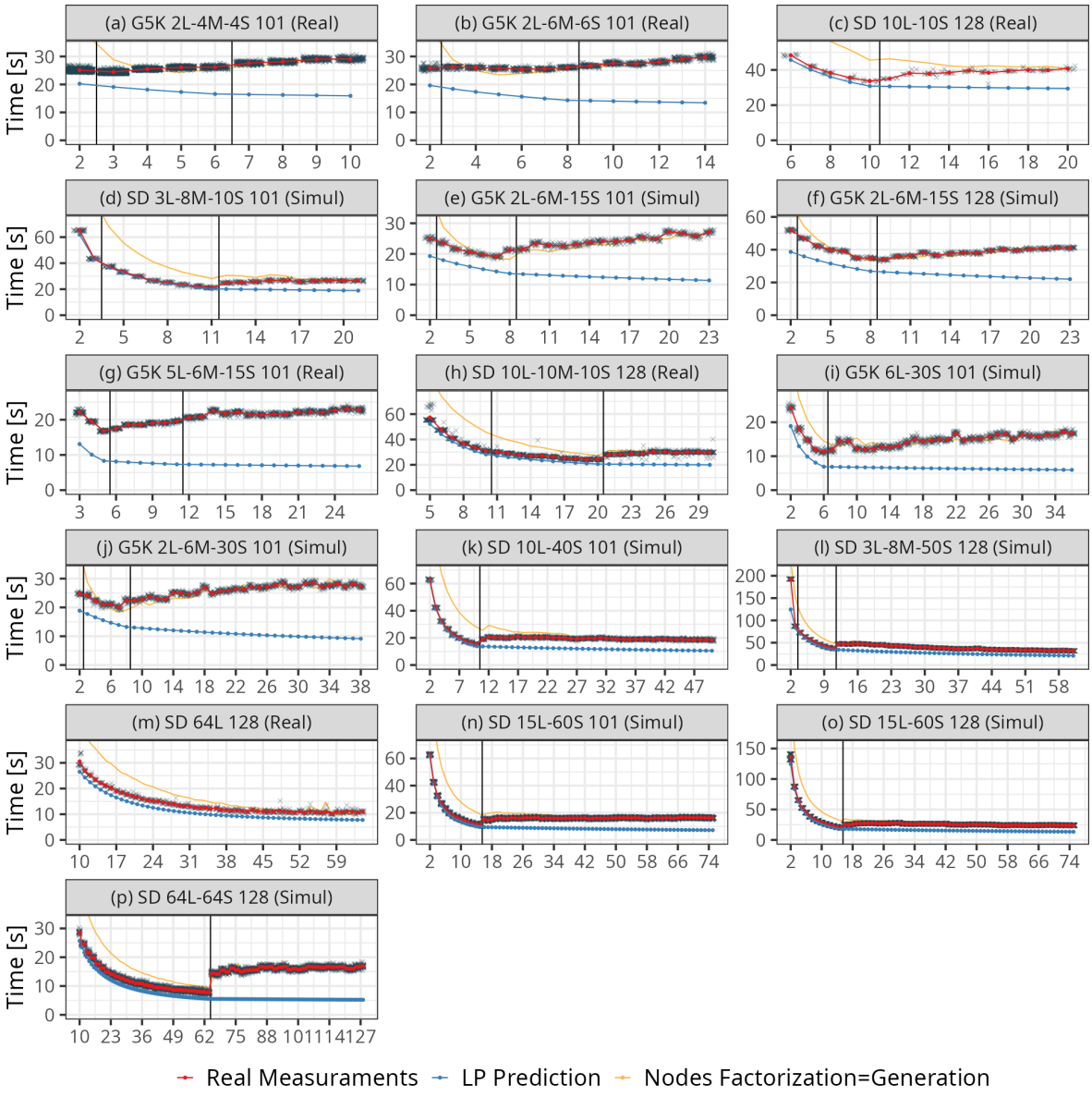
7.3.1 Behavior on different setups

This section presents the behavior of the problem in many setups formulated by varying the machines of Table 7.1. Figure 7.4 presents the behavior (performance when adding nodes) of 16 setups in a similar way to Figure 7.2. One difference is the addition of a yellow line that shows the rigid situation where the same number of nodes is used for both the generation and the factorization steps.

The data comes from executions that solve the workload entirely and explore all possible numbers of nodes for the factorization phase. These measures enable the estimation of the true variability of the application and the system. Second, we rely on simulations of ExaGeoStat with StarPU-SimGrid, allowing us to quickly and accurately estimate the response of ExaGeoStat even for platforms that are not directly available or would be very time and resource-consuming. However, with the default StarPU-Simgrid configuration, the time for a given iteration and nodes' number is deterministic. Therefore, the simulation evaluation of each configuration is augmented 30 times, assuming a normal distribution with a standard deviation of 0.5s (computed from the real experiments). Real experiments and simulations are marked in the title of each configuration in Figure 7.4.

These setups generally present all the shapes we found, even in unreported situations. Some scenarios show a very smooth behavior, like cases (a), (b), (e), (f), and (m). Most of them are scenarios on G5K that have a limited network compared to SD. Another behavior is the presence of discontinuities when a new group of machines is introduced, like cases (d), (g), (h), (k), (l), (n), (o), and (p). In these cases, adding a very slow node (CPU-only) caused problems on

Figure 7.4 – Behavior using different heterogeneous nodes setups (Table 7.1) by varying the number of factorization nodes.



Source: The Author.

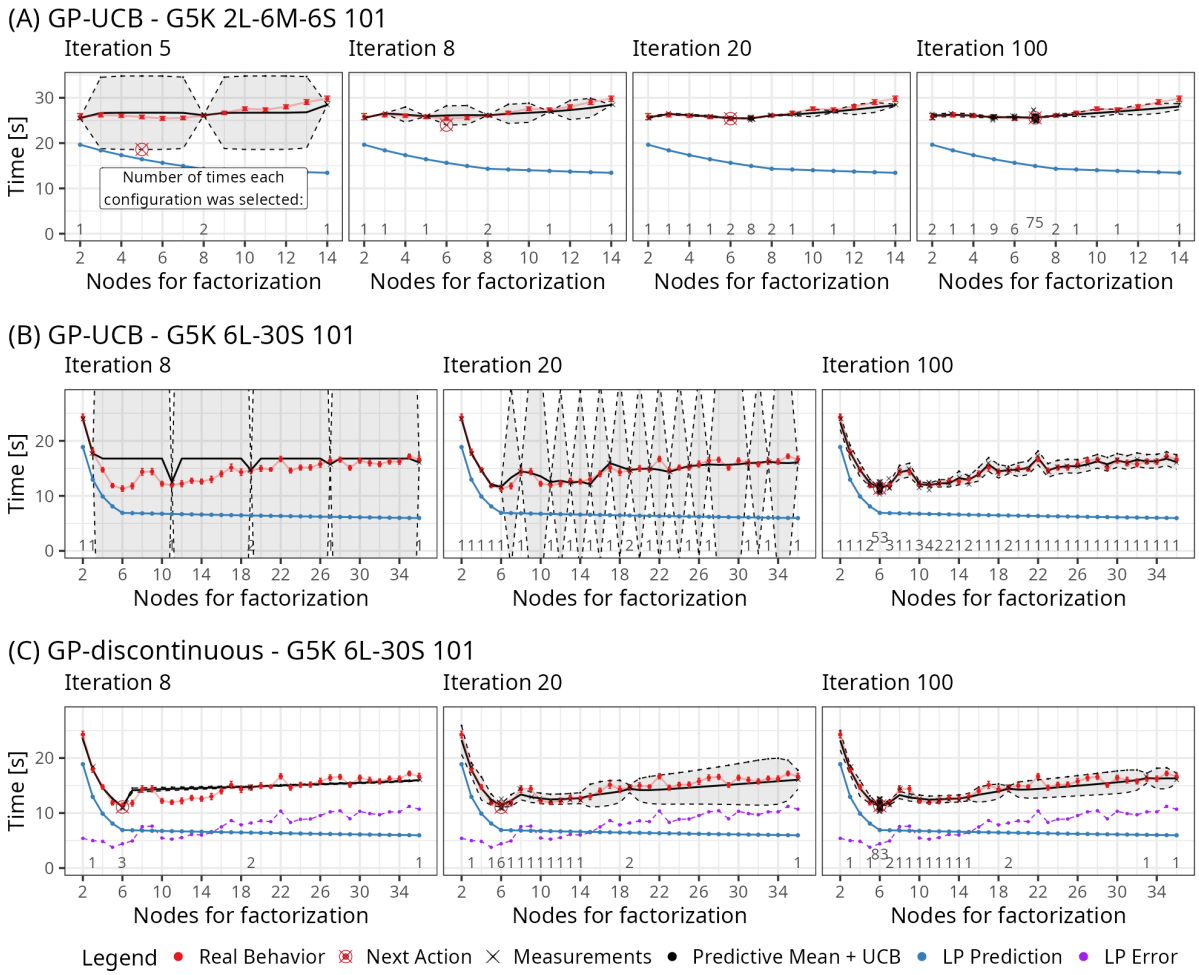
the critical path. Finally, some situations had small breaks related to the heterogeneous partition of that specific number of nodes (and occurring inside groups of the same types of machines), like cases (c), (e), (f), (g), (i), (j), and (p).

7.3.2 Depicting the GP exploration/exploitation step-by-step

Figure 7.5 (A) illustrates the step-by-step process of the **GP-UCB** on the G5K 2L-6M-6S 101 scenario. The graphs show different iterations and the corresponding GP estimation. For

each iteration, the solid red line with error bars shows the real data for each action with a 99% confidence interval. The blue lines show the LP estimation. The black line is the GP mean prediction, and the black dashed lines are the UCB component. The large red cross is the next action to take based on the current situation. The bottom of each graph shows the number of measurements taken with that number of nodes.

Figure 7.5 – Step-by-step of (A) GP-UCB in G5K 2L-6M-6S 101, (B) GP-UCB in G5K 6L-30S 101, and (C) GP-discontinuous in G5K 6L-30S 101



Source: The Author.

Figure 7.5 (A) Iteration 5 has two areas without measurements with much uncertainty in the surrogate model caused by the experimental design's first points. This area is further reduced, as shown in Iteration 8 when middle points were evaluated. At Iteration 20, there is already a convergence in finding the best points (action six or seven, which are very similar). This behavior is maintained until Iteration 100. However, it is interesting to note that GP-UCB keeps exploring as actions 5 and 6 continue to be evaluated sometimes but that some actions (10, 12, 13) are not even tried as they would clearly provide bad performance. In this scenario, the **GP-UCB** is enough to find the best configuration without assessing all the search space.

However, the **GP-UCB** does not behave so well in scenario G5K 6L-30S 101 as shown in Figure 7.5 (B). There is a lot of uncertainty in Iterations 8 and 20 because the measurements scale is large, and the curve has some discontinuities that mislead the smooth GP and overestimate the scale parameters α and θ. Although it does find the best option by Iteration 100, it has explored all the points. Applying the most elaborate version, **GP-discontinuous**, solves these problems as shown in Figure 7.5 (C). Panel (C) has an additional purple line representing the difference between the real data and the LP. At Iteration 8, it is possible to check how this GP version models the discontinuities with the dummy variables. The model presents two distinctive curves, one until action six and another after it, which is a very flat line on this iteration. On Iteration 20, the local minima zone from actions $10 - 14$ is already evaluated, improving the model accuracy. At this iteration, the surrogate model includes all the real values on its UCB component. On Iteration 100, it does find the best action without needing to measure all the points, skipping most of the right zone after action 20.

7.3.3 Results Overview: GP and existing Exploration Strategies

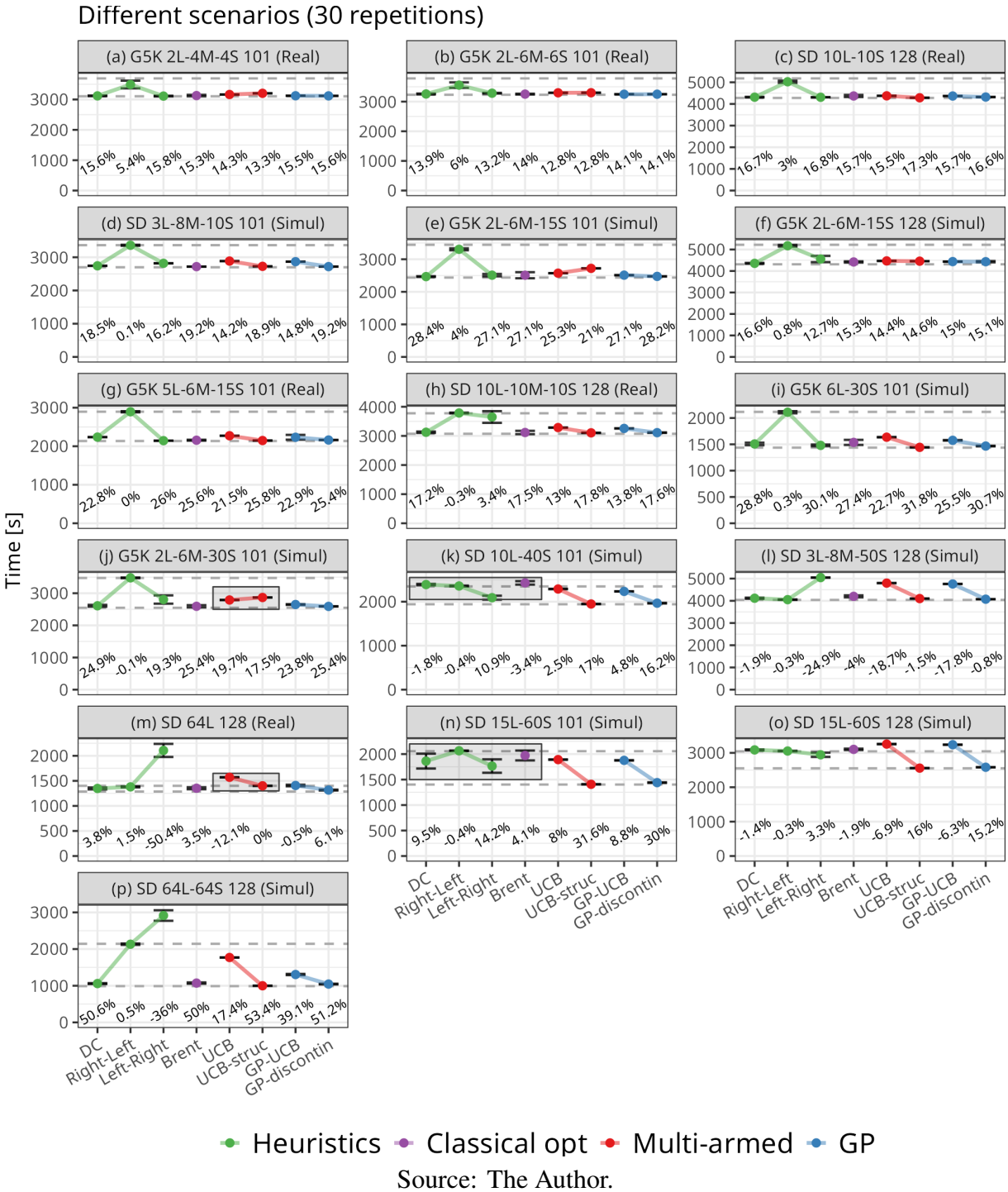
The evaluation of the exploration strategies proposed in Section 7.2 has been conducted using two complementary techniques. First, we collect data from all the iteration durations obtained through real experiments or simulations to obtain a fast and fair comparison. We used R to implement the strategies; every time an action was chosen, we resampled that data. This way, all exploration strategies are compared with the exact same iteration durations (see the Shiny app[2]), and their comparison can be made reliable from a statistical point of view. Finally, we also implemented the GP strategies directly in ExaGeoStat, which lets it control the number of nodes it uses. All the possible distributions were precomputed and accessed when an action is chosen. With this second approach, it is possible to measure the computational cost overhead of the methods, as discussed in Section 7.3.4.

Figure 7.6 depicts a performance evaluation overview of all studied exploration strategies with all scenarios. Each colored line is a group of strategies where the points are the makespan mean of 30 executions after 127 iterations. The top dashed horizontal line describes the performance when using all nodes, and the bottom one is the best option when knowing the best configuration upfront. The percentage for each strategy is the acceleration with respect to using all the machines. In what follows, we discuss the scenarios referencing Figures 7.4 and 7.6.

The **UCB**, **Right-Left**, and **Left-Right** perform poorly in more than half of the scenarios.

[2]Publicly available at: <https://adaphetnodes.shinyapps.io/shiny/>

Figure 7.6 – Comparison of different methods in 16 scenarios



Source: The Author.

UCB requires a full exploration that degrades the overall performance. Bad configurations for this strategy have many nodes or should use most nodes. In scenario (o), it is worse than using all the nodes (**UCB** point is above the top dashed line) because of inevitable bad points exploration. Moreover, **Right-Left** and **Left-Right** fail because they do not explore enough and may get stuck in local minima. For example, in Figure 7.4 (p), using all 128 factorization nodes is better than using 127, so it never explores the best action, which is 64. In (a), variability plays a role,

and some **Right-Left** executions presented better results than others. The algorithm often stops too soon by bad luck because it is not resilient to variability.

The simple divide and conquer (**DC**) algorithm sometimes performs very well, finding the best option. Indeed, using this simple heuristic in most of the Figure 7.4 curves will work correctly. However, the variability of the measurements may trick the decisions, and in some scenarios like that of Figure 7.6 (n), it may perform very well or very badly depending on the observations. In these scenarios, making a wrong decision in any division moment may be enough to never find the best number of nodes. As shown in Figure 7.4 (n), the best point is on the left, but the method will never explore it if the right side is chosen in the first division.

The **Brent** strategy also performs very well in many scenarios. However, in ones with discontinuity like Figure 7.4 (k), (n), and (o), it may not explore enough and get tricked into a local minimum. In such scenarios, because of the scale of the X-axis, it may only try points after the addition of one group of homogeneous machines. Figure 7.4 (n) does have a large plateau corresponding to the 60 nodes of that group. Also, it is subject to the variability of the measurements, like in Figure 7.6 (e), (i), and (l). In these scenarios, it is possible to check that not all executions found the best number of nodes (the confidence interval is larger), resulting in a larger makespan in some executions.

The **GP-UCB** approach works well in more or less half of the scenarios. It does not always find the best option (in the available iterations) or requires a full exploration of the search space, as discussed in Section 7.3.2. It presents good acceleration in Figure 7.6 (a), (b), (c), (e), (f), and (j), which do not present discontinuities and have a small search space. However, scenarios like (k), (l), (m), (n), and (o) indicate that the GP requires extra modeling to handle more complex situations.

However, the **GP-discontinuous** performs very well in all scenarios. The introduction of trend, search space limits, and dummy variables allows it to quickly and dynamically reach the optimal configuration with a minimal overhead compared to the lower bound obtained with a static clairvoyant choice. An example is Figure 7.5 (C), which presents the state of the GP-discontinuous strategy in (i). After the 20 iterations, the method already identified the optimal number of nodes, and the surrogate model has a very good estimate of the response of the whole system as it comprises most of the (true) red line without having ever explored large regions.

UCB-struct also performs very well in almost all scenarios. It has minimal possible actions to check, and its algorithm is resilient to variability. Also, selecting multiple groups of homogeneous machines is, or close to, the best option. However, there are scenarios where the best action to take is far from those points, and because this strategy would never explore

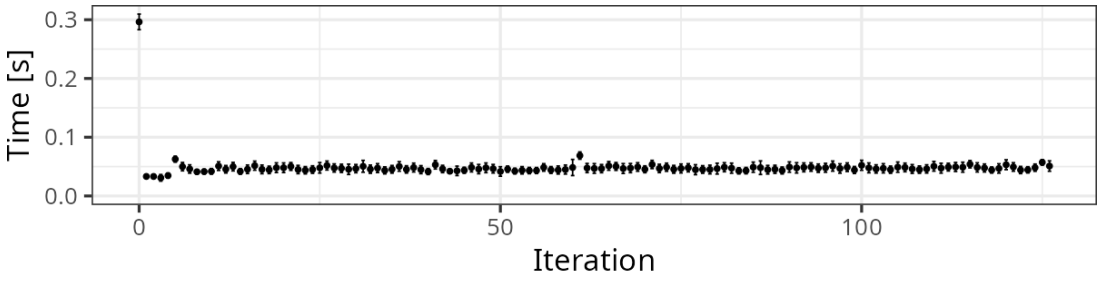
or know these points, it cannot find them. Examples are Figure 7.6 (a), (e), and (j). In these scenarios, the optimal choice (Figure 7.4) does not correspond to any vertical lines that mark the groups' transitions.

In short, the **GP-discontinuous** presents good results in all scenarios, up to 51.2% speedup as shown in Figure 7.6 (p). In scenarios where using all the nodes is the best option (e.g., (l)), the final performance compared to using all notes is superior to -1%, meaning that the exploration overhead is low. All these results corroborate with the Table 7.2 expectations.

7.3.4 GP Computation Overhead Evaluation

The performance gains of using an exploration strategy should be smaller than the overhead of its computation. Figure 7.7 shows the overhead of the **GP-discontinuous** strategy on the scenario (b) G5K 2L-6M-6S, with ten repetitions of a real experiment running the GP online. Each black point represents the average overhead for that iteration. The overhead per iteration is almost constant in this model. The first iteration is longer than the others, and the successive four iterations are less expensive, as they do not perform the GP's computations. In the sixth and subsequent iterations, the DiceKriging package is called, resulting in an almost constant duration. The final overhead per iteration is negligible (0.04s – 0.06s) compared to the typical iteration total duration (10s – 30s). The cost is thus not of great concern compared to the risk of missing a learning opportunity.

Figure 7.7 – Overhead of GP in function of iterations



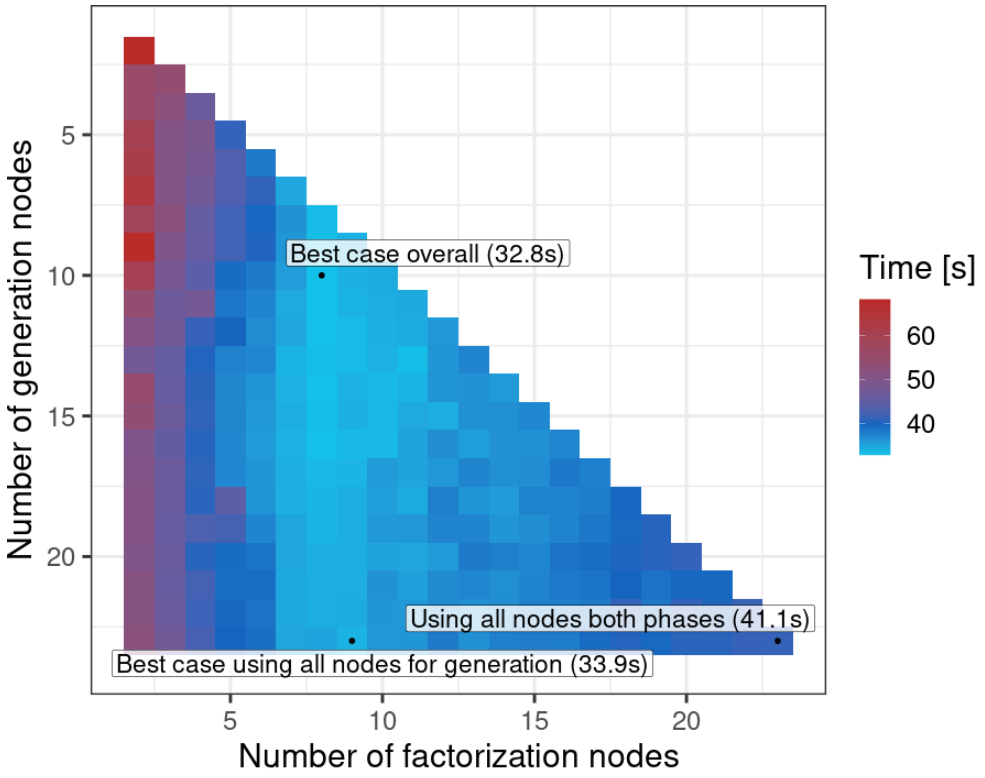
Source: The Author.

7.3.5 Optimizing considering all phases

Using all the machines for the first (generation) phase is the best option for most scenarios. However, one investigated case shows that using too many nodes for the generation can also result

in a slowdown. Figure 7.8 presents the iteration duration (as a colored gradient) when varying both the number of generation nodes and the number of factorization nodes for (f) G5K 2L-6M-15S 128. In this scenario, using 10 generation nodes and eight factorization nodes is better than using all 23 generation nodes (23/9) by 1.1 seconds (≈3% less). In this type of situation, the problem should then be explored in both dimensions. Although the GP exploration should gracefully extend to more dimensions, considering more parameters significantly enlarges the search space and we believe the gain would be limited in practice.

Figure 7.8 – Iteration makespan with different number of generation and factorization nodes



Source: The Author.

7.4 Energy Perspectives

So far, this chapter has proposed and evaluated the Gaussian Process (GP) as a surrogate model focused exclusively on the application performance. At the same time, we wonder how such a model could also consider other observation variables, such as the application's energy consumption when running on a specific set of computational resources. This case is strongly motivated by some situations where the best performance requires many powerful GPU nodes and possibly many additional CPU-only nodes, which are left mostly idle during the factorization phase. This invariably induces idle time in slower resources. The issue here is that these nodes,

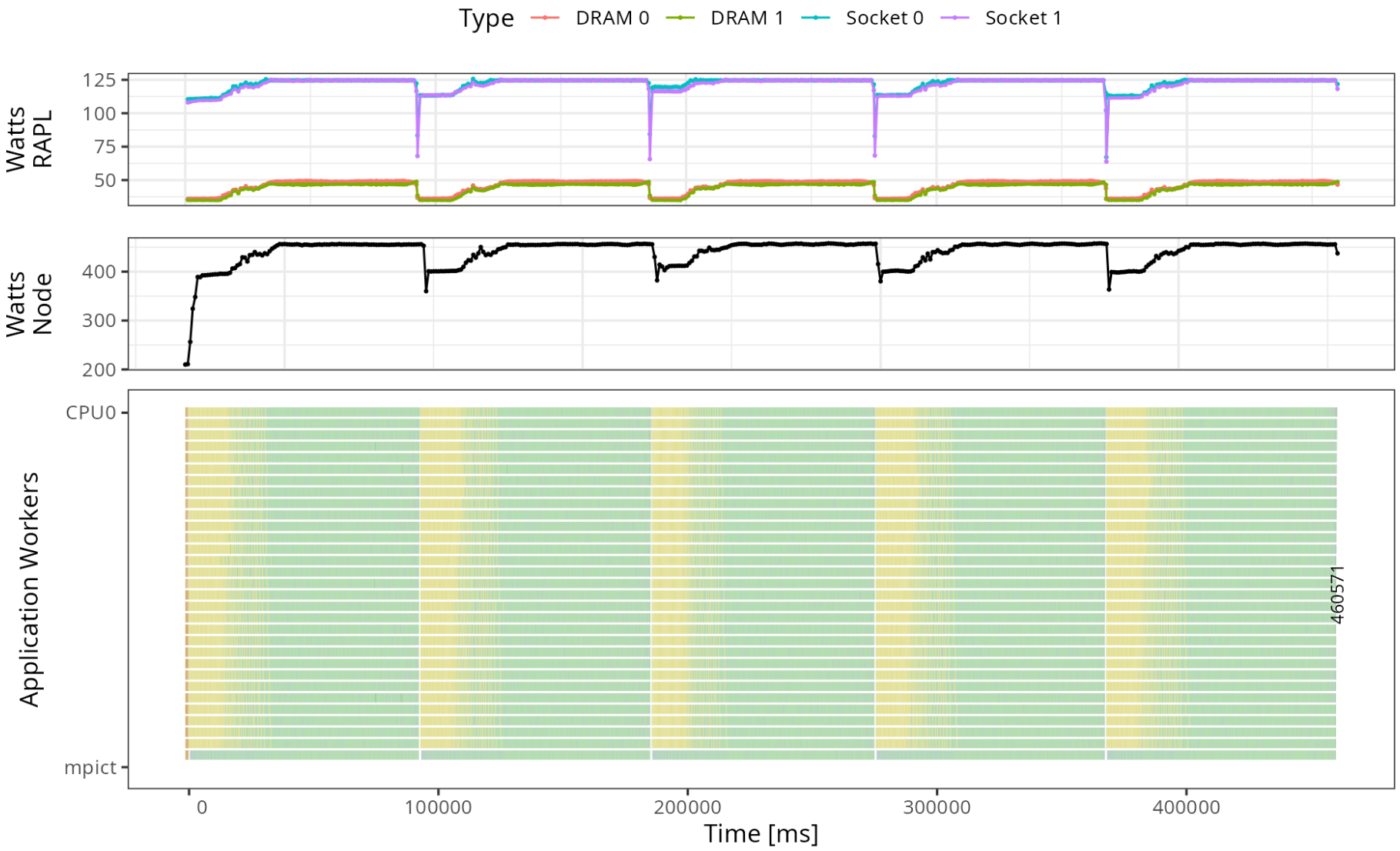
even when idle, do consume energy. A natural and relevant extension of the previous work is thus to automatically identify a good trade-off between makespan and energy consumption. However, some of the previous model's decisions do not apply in this situation. It is no longer possible to summarize a configuration through the number of nodes one phase uses ordered by the fastest nodes. In this case, the search space should include the number of machines from each group in every phase. This entire search space is required because, from an efficiency perspective, a swap from a fast node to a slower one could improve energy efficiency without compromising makespan.

The first step in this investigation is to understand the energy consumption of the applications. ExaGeoStat and StarPU were modified to collect energy metrics via PAPI from RAPL (WEAVER et al., 2012) and NVIDIA NVML (NVIDIA, 2023). Figure 7.9 presents five iterations of ExaGeoStat. The bottom is the traditional Gantt Chart, where the beginning of each iteration can be identified as a yellow phase in the Gantt Chart, represented at the bottom of the figure. The first panel shows four power metrics derived from the RAPL energy counters. Each metric is associated with a socket and can be the power consumption of a CPU or RAM. The second panel presents the power consumption from the wattmeter of the whole machine running the application (troll machine in G5K). The main interesting observation from the figure is that tasks have a heterogeneous power consumption. In the beginning, the generation tasks do not trigger the full power of the CPUs. We believe that such behavior occurs because such tasks are memory-bound. However, as the number of concurrent factorization tasks increases, power gradually increases until it reaches the CPUs' limits. Such results indicate that the energy and power consumption of the application can not be modeled as a function of workers' utilization alone but need to consider the tasks' type and amount.

The following investigation step was to characterize a case varying the number of machines individually for each cluster and each phase. The experimental evaluation considers eight Chiclet machines, five Chifflot machines (with P100), and two Chifflot machines (with V100). It is necessary to vary the number of nodes in all partitions for all phases. The notation used is $n^{\mathrm{S}}_{\mathrm{Gen}} + n^{\mathrm{M}}_{\mathrm{Gen}} + n^{\mathrm{L}}_{\mathrm{Gen}} \mid n^{\mathrm{S}}_{\mathrm{Facto}} + n^{\mathrm{M}}_{\mathrm{Facto}} + n^{\mathrm{L}}_{\mathrm{Facto}}$. A case with notation $8 + 5 + 2 \mid 7 + 5 + 1$ means that the generation used eight Chiclet, five Chifflot (P100), and two Chifflot (V100) for the generation and seven Chiclet, four Chifflot (P100), and one Chifflot (V100) for the factorization. All cases using less than four GPU machines for the factorization were not considered, as they are expected by the LP of Chapter 6 to provide poor performance results.

Figure 7.10 presents the cases of such experimental evaluation where the X-axis is the time to compute one iteration and the Y-axis is the energy used throughout the iteration. Each

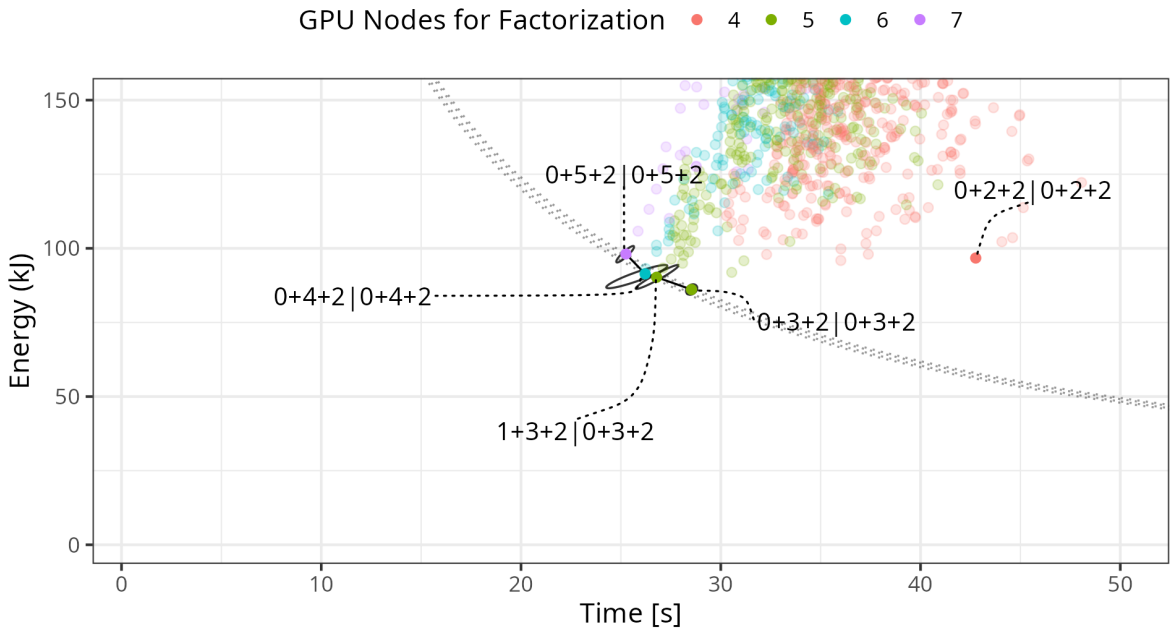
Figure 7.9 – Power consumption from RAPL and Wattmeter in the first two panels, and the traditional Gantt Chart for a one-node ExaGeoStat execution



Source: The Author.

dot is the average of at least two executions of the same case, showing data for one ExaGeoStat iteration. The data is relative to iterations two, four, or five, as it is more representative of further iterations. The colors represent the number of GPU machines used for the factorization. The solid black line is the Pareto Front. Such cases have their notation highlighted. The case $0+2+2\,|\,0+2+2$ is also highlighted but is not part of the Pareto Front, but it is the case using the smaller number of machines of these experiments. All instances of the Pareto Front have a confidence ellipse accounting for the variability of the measurements. The dotted curves are the time× energy product isolines of the Pareto Front elements.

Figure 7.10 – Energy and time of a ExaGeoStat iteration varying the number of machines among eight Chiclet, five Chifflot (P100), and two Chifflot (V100)



Source: The Author.

In this situation, the optimization is multi-criteria, and energy and time should be minimized but have different units. Consequently, aggregating both criteria through a weighted sum or a weighted maximum always raises the question of the weights. Furthermore, both criteria correlate since very long iteration durations typically incur high energy consumption. A common approach is thus to aggregate both criteria with a product, which is unitless. The cases in the Pareto Front do not use all machines available, and there is a case $1+3+2\,|\,0+3+2$ in which the number of nodes from generation to factorization changes. Also, it is interesting to note that these four nodes are somehow equivalent to the product time× energy, but they are quite different from the time and energy perspective. The two configurations that appear to be optimal for the product seem to be $1+3+2\,|\,0+3+2$ and $0+4+2\,|\,0+4+2$. They achieve

an interesting trade-off between time and energy compared to the two other configurations, and further measurements would be required to distinguish them further. A faster case also uses less energy because energy consumption is directly related to the makespan. Another positive side-effect of considering the product is that it should easily discriminate between good and bad configurations (i.e., many configurations with relatively good energy consumption can have wildly different durations and conversely).

Compared to the previous time-only approach, one of the main problems in this situation comes from the dimension of the search space. A good strategy should still quickly find suitable candidates. However, a model that considers all possible numbers of nodes per partition per phase is more complicated (i.e., has more parameters) and typically requires more initial measurements. How to efficiently bootstrap the model, which includes the initial cases to select, is an open question and left as future work.

Another difficulty stems from the computation of the confidence bound, which reinforcement learning approaches like UCB require. Indeed, in our previous work, time is modeled by the GP using the difference between the LP and the actual makespan, with a confidence interval of its own. Modeling energy requires time as an input, and this model would generate another individual confidence bound. When using the time× energy product plus the UCB to select a configuration, it is still unclear how to combine the uncertainties of both models, especially as they are strongly correlated.

Although there are many challenges, we believe such an approach using the GP surrogate (and having a model for all the search space) can be a good strategy for finding energy-aware configurations. This problem is still under investigation and will be pursued as a future work of this thesis. Such a model would enable the application to adapt dynamically to a pool of heterogeneous resources, improving time and energy.

7.5 Discussion

This Chapter proposes and analyzes different exploration methods to find the best collection of heterogeneous nodes for an HPC application phase. It uses the ExaGeoStat application, which has multiple phases and was already optimized for heterogeneous resources (NESI; LEGRAND; SCHNORR, 2021). However, determining the ideal number of nodes per phase was an open and challenging problem. Indeed, the network and the distribution of data over heterogeneous nodes may cause unpredictable and unexpected behavior during the execution of the application on a particular number of nodes. The application should thus explore various

configurations and adapt online to discover the optimal subset of nodes for a given scenario.

Our results show that an informed Gaussian process based reinforcement learning strategy can quickly find the best configuration of nodes for the main phase (factorization). The superior approach, **GP-discontinuous**, uses bounding mechanisms to filter the search space, model the difference of the makespan to a lower bound (and already have some knowledge of the scenario), and use dummy variables and a linear trend to model discontinuities caused by the distributions. This method provided the consistent best results in all 16 studied scenarios. Another (simpler) method that performs particularly well is **UCB-struct**, which only considers specific points. However, this method will only sometimes find the best configuration, as it is constrained not to search all the space but only the multiple complete homogeneous groups. In the end, using all nodes for the generation phase and only a learned subset for the factorization provided up to 51.2% speedup compared to using all nodes for both phases. These results demonstrated that the application could discover during execution the action (number of nodes) it could take to improve performance and adapt to the heterogeneous resources.

This chapter's main contributions and results were published in IPDPS 2022 (NESI; SCHNORR; LEGRAND, 2022).

8 SUMMARIZING APPLICATIONS' BEHAVIOR

The performance analysis of High-Performance Computing applications is a vital step for achieving correct performance. But the complexity of the applications and systems, including the many levels of heterogeneity (HPE et al., 2022), adds considerable challenges to such activity. The performance analysis of HPC applications through visualization is considered an advantageous methodology (GARCIA PINTO et al., 2018), as it enables a facilitated comprehension of large amounts of trace data (SCHNORR; LEGRAND, 2013). However, even with this strategy, there is limited space to plot, and the visualization strategy can give wrong insights. That is the case of Gantt Charts, which focus on resource utilization but can hide the actual progression of the application. Although performance visualizations tailored for an application (MILETTO et al., 2022) or runtime aspects (NESI et al., 2019) can mitigate some of these problems, general strategies that summarize the overall performance behavior would aid in this matter.

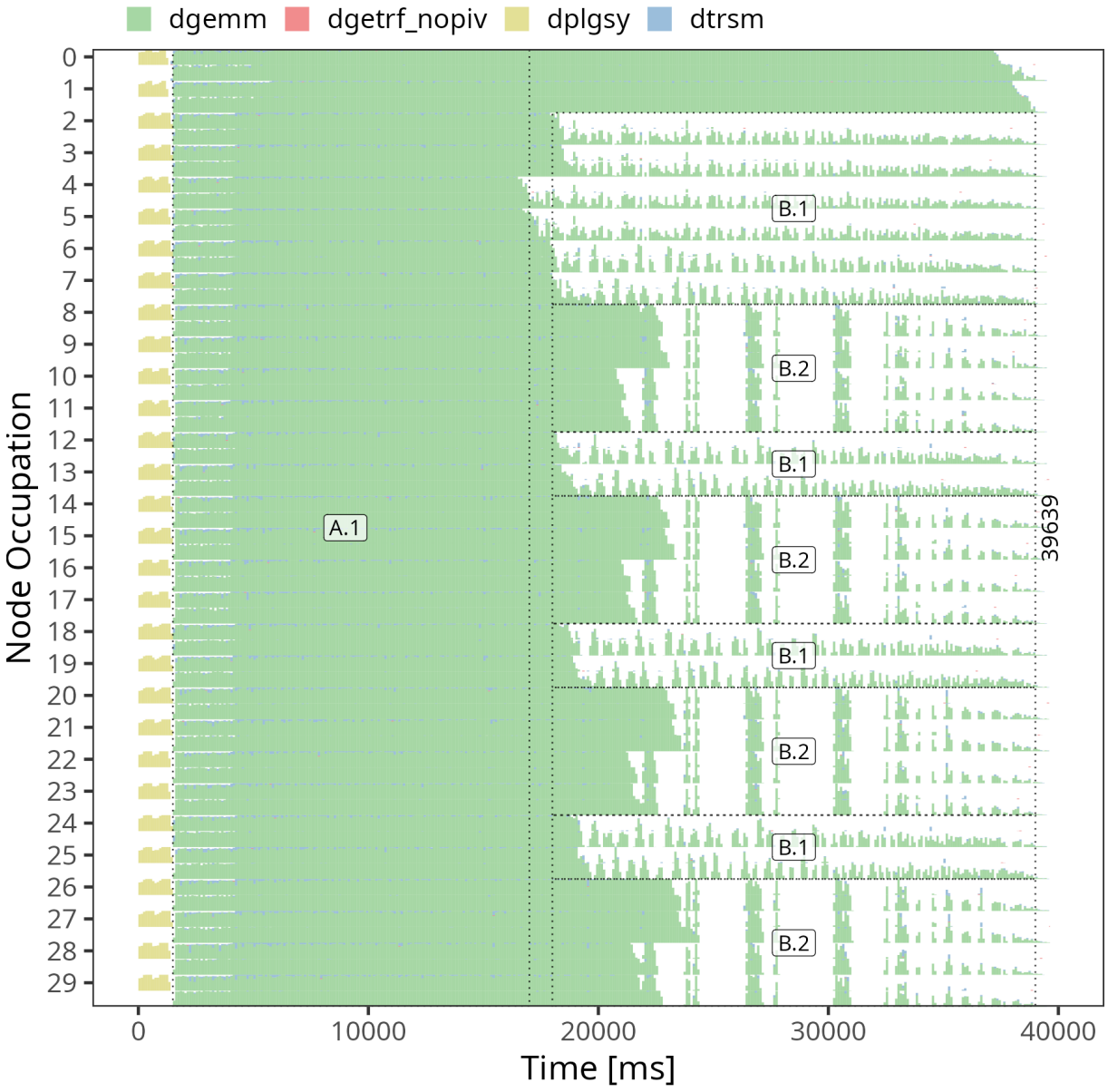
This Chapter proposes a performance analysis methodology through an entry-level visualization. It aims to check the progression of task-based applications on individual nodes to indicate moments and node groups of interest. This methodology comprises three elements: a progression metric per node that employs the structure of task-based applications, a clustering method to classify nodes and reduce the elements to show, and a final entry-level visualization of such components. Section 8.1 presents the general idea of the problem, followed by Section 8.2 detailing the proposed methodology. Section 8.3 presents the evaluation of the methodology on real applications.

8.1 Problem: Limited space to plot complex behaviors

The comprehension of applications' performance may become problematic because of the number of features the system presents. This is particularly the case when using Gantt charts to plot states per node-level resources (cores, GPUs) with large amounts of nodes. Although there are techniques for aggregating and reducing the size of such visualizations (PILLET et al., 1995; KNÜPFER et al., 2008; SCHNORR; STEIN; KERGOMMEAUX, 2013; DOSIMONT et al., 2014), the traditional idea of having system elements as one axis would never scale as their number grows. Moreover, these Gantt chart ideas focus on resource utilization, which does not necessarily reflect how well the application is distributed and progresses. Aggregating many resources with the same utilization would also mask application problems and their progression.

Consider an example of the dense linear algebra Chameleon library with the LU factorization, where when using 30 nodes with two GPUs each, two nodes were faulty and were initialized with only one GPU. Figure 8.1 presents the aggregated Gantt chart per node resource of such execution made with StarVZ. Each node has two rows; the first is the aggregation utilization of the CPUs (four cores per node), while the second is for GPUs (two per node). In this case, it is possible to notice that two nodes (0 and 1) were working more than the others, and already with 30 nodes, such visualizations present scalability problems even with intra-node resource aggregation. The initial yellow tasks represent the generation data phase, and after

Figure 8.1 – Gantt chart with nodes' resources aggregation of the Chameleon LU Factorization execution with 30 nodes where two are misbehaving



Source: The Author.

synchronization, the green area represents the main computational task, the `gemm` operation. In this representation, it is unclear if the two first nodes are problematic or correct concerning behavior. Moreover, the other nodes have two distinct patterns (B.1 and B.2 annotations), but it is also unclear which ones perform better. The only certainty from this visualization is that the resources of nodes 0 and 1 are working all the time.

A critical aspect of the analysis workflow is the progression of the application towards its final result. This aspect is missing in visualizations such as the one depicted in 8.1. Measuring at a given point how far the application is from its goal, from its ideal performance, and how each node behaves compared to each other can lead to performance improvement insights. We argue that an entry-level visualization for the performance analysis workflow that focuses on the progression would complement the Gantt chart by providing an overview. Such entry-level visualization should guide the analyst to points and nodes of interest by serving as a preliminary step before delving into the details of application behavior using the Gantt chart. This visualization should enable the perception of node outliers and handle heterogeneity in its many forms, such as resources, application phases, and tasks.

8.2 Proposal: Node Progression visualization through clustering

When many resources have to be considered, the raw Gantt visualization is inadequate, and other summarizing techniques should be used. Instead of observing the Gantt chart or its variations as the first instrument, this Section presents a methodology for an entry-level tool to provide an overview. This overview can indicate and classify nodes of interest, where the traditional methods (including detailed Gantt charts) could use such groups to focus on non-redundant information.

8.2.1 Progression Metrics

The notion of how much work the application has accomplished at a given time, or how much it requires to finish, can be captured into a progression metric. When improving or analyzing an application's performance, its progression is an aspect of main interest, as if it progresses correctly and fast, the application's execution time will be optimized. In a distributed scenario with many resources, it is desired to understand how each node performs relatively.

We define a progression metric to capture the individual and normalized $[0, 1]$ progression

of a node's work while maintaining the relative comparison possible. This comparison means that their metric should be the same when nodes achieve the same relative progression (i.e., 50% of work).

The important aspect is the quantification of work. It can be directly associated with the application's main algorithm, for example, the iteration number concluded on a particular simulation step. However, it could be agnostic to the application and use information from the programming paradigm, in our case, the task-based one. In this agnostic case, universal information for all applications like tasks, the DAG, and task performance models can be used to measure work.

One very simple progression metric to be defined in the context of the task-based programming paradigm is the number of completed tasks $C_{s,n}$ at a given moment s in a node n as the primary measure of work and normalizing it by dividing with the node's (n) total amount of tasks N_n. The progression of a node n at a given time s, $P_{n,s}$ will be given by:

$$P_{n,s} = \frac{C_{n,s}}{N_n} \tag{8.1}$$

However, this simple metric works when the application has a clear dominant phase and task. Complex applications with multiple phases (NESI; LEGRAND; SCHNORR, 2021) and a variety of significant tasks would require quantifying different tasks costs considering the possible intra-node heterogeneity. This quantification could be achieved by using tasks duration $w_{n,t,r}$ of each heterogeneous resource r (CPU, GPU) in the node n to relativize them between task types t. The proposed metric is present in Equations 8.2 and 8.3. $W_{t,n}$ is essentially how much time a task t would hypothetically take if all node n resources with an available implementation could run it simultaneously, creating a weight per task type. Then, the progression ($P_{n,s}$) can be measured by summing for each task, the number of tasks completed $C_{n,s,t}$ of type t on node n at time s multiplied by the respective weight ($W_{t,n}$), and normalizing in the same way by replacing C by N. Such a metric would detect fewer longer tasks' progression and contribution correctly.

$$W_{t,n} = \frac{1}{\sum_r \frac{1}{w_{n,t,r}}} \tag{8.2}$$

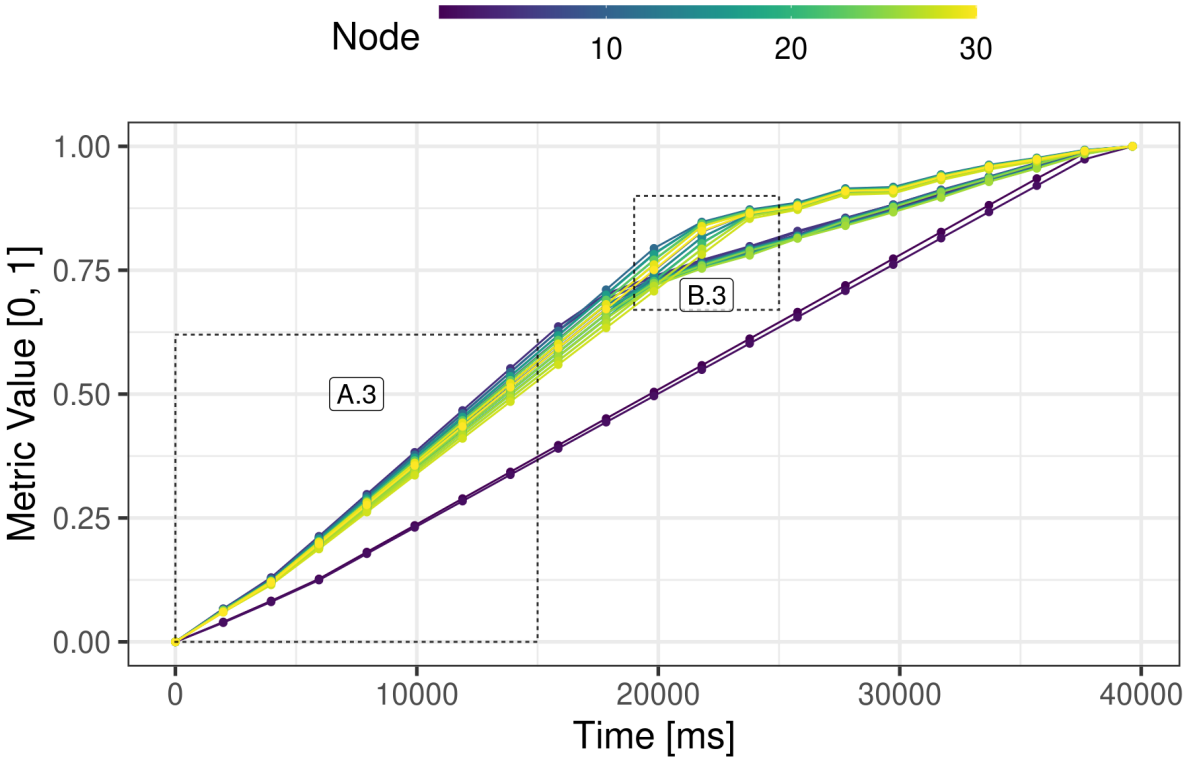
$$P_{n,s} = \frac{\sum_t (C_{n,s,t} W_{t,n})}{\sum_t (N_{n,t} W_{t,n})} \tag{8.3}$$

To illustrate how this metric behaves, consider the case of the last section in Figure 8.1, the LU factorization with 30 nodes where the first two were initialized with one less GPU. Figure

8.2 presents the progression metric that accounts for heterogeneity for each node of Figure 8.1 execution. At the start of the execution, annotation A.3, there is a split in behavior. The below line (actually two overlapping lines) represents the progress of the two slower nodes. The upper lines show the progression of the other nodes. In the middle of the execution, there is also a division of nodes in two distinct behavior, annotation B.3, observed from Gantt chart areas B.1 and B.2.

The progression metric, which is not a utilization metric (as in the Gantt chart), indicates that a group of nodes is progressing slower since the beginning, annotation A.3, a situation that is not clear in the Gantt area A.1. The Gantt gives a false impression in area A.1 that things were progression well, where actually there was a problem in the load partition since the start (considering the relative real speed of the machines). In the middle of the execution, the split in behavior is directly associated with the data partition, as will be discussed in Section 8.3. Where B.1 and B.2 are associated respectively with nodes that communicate directly or indirectly with the two problematic nodes. Also, this visualization already has a lot of lines (one per node).

Figure 8.2 – Progression heterogeneous metric applied to the Chameleon simulation of the LU Factorization on 30 nodes



Source: The Author.

8.2.2 Summarizing by Clustering

Analyzing such progression metrics may still be difficult in cases with many nodes having similar behavior, i.e., similar metric values. The identification of nodes or groups of nodes of interest can still be overwhelming. A possible solution is summarizing such metrics by clustering and understanding the behavior of node groups instead of individual ones. We utilize the notion of discrete time-steps, computing the progress metrics and clustering the nodes only for those moments. This clustering per step is particularly useful because some performance degradation may arise at lonely moments of the execution, and while at that particular moment, the node is an outlier, during the rest of the execution, it behaves the same as other nodes.

At a given time-step, the progression may be summarized by clustering nodes that follow the intuitive, though weak and relative, characteristics: (1) Have the same progress behavior but have differences in the progression metric because of system variability, where one would expect a normal distribution; (2) Have genuine, though light, differences in behavior (small variances in workload, for example), but such small differences relatively do not reflect points of interest for the analyst. To comply with such characteristics, we select modal clustering, where the populations and clusters are defined as high regions of density divided by low regions of density (FUKUNAGA; HOSTETLER, 1975; CHEN; GENOVESE; WASSERMAN, 2016; CHACÓN, 2015). Such a definition would encompass both characteristics.

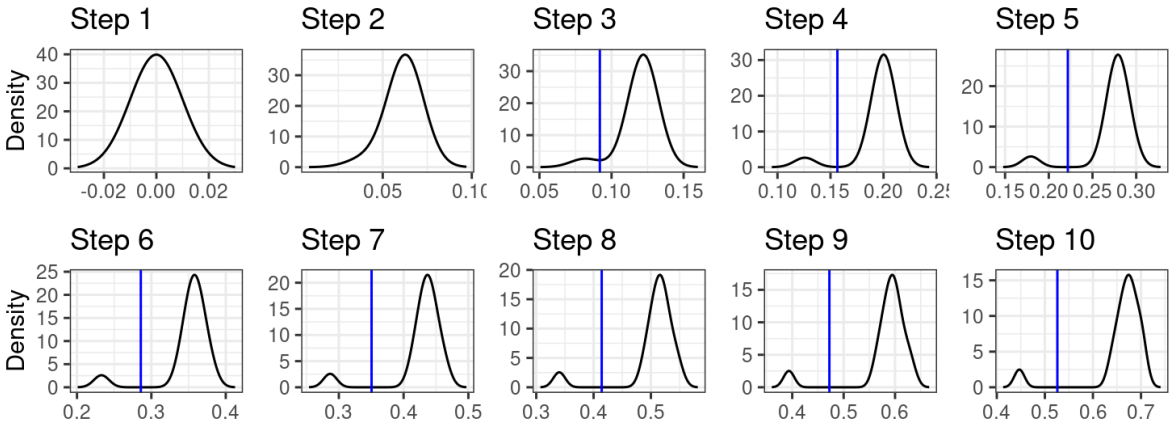
K-Means and Gaussian Mixture Models (GMM) (CHACÓN, 2019) were also tested, but such approaches require determining the number of k clusters or distribution components. One of the most popular approaches to determine this is using Bayesian Information Criterion (BIC); however, it requires the number of observations to be large (KASS; WASSERMAN, 1995; GIRAUD, 2021). In this case, it is also interesting to want to cluster groups with a few outlier nodes, even if this group comprises only one member. Also, for the case of the GMM, which models clusters as a realization of a normal distribution, it is preferable generality that the groups are not necessarily normal distributions.

The modal clustering is performed with the principle of the mean shift algorithm (FUKUNAGA; HOSTETLER, 1975), where data points are moved to the near kernel density modes, and clusters are divided by local minimums (CHEN; GENOVESE; WASSERMAN, 2016; CHACÓN, 2015). For this one-dimensional case with a relatively small number of data points, this can be achieved by computing the density estimation (using the Gaussian kernel in our case) with a bandwidth parameter h and finding the local minimums (CHACÓN, 2015). All data points between the same minimums are classified in the same group as belonging to that mode.

Figure 8.3 presents the density estimations for the first ten time-steps with a bandwidth of 0.01, where the horizontal axis is the progression metric value, the vertical one is the density, and the blue vertical lines are the local minimums. At step one, all nodes share the same progression metric, zero, so it has a unique mode centered at 0. At step two, two nodes start to depart from the other 28; this causes a skewed distribution, with the mode much closer to the metric of the 28 nodes. But still, the distance between points using this bandwidth was not sufficiently far for creating another mode. At step three, there was enough distance, making two local modes, with a local minimum in the middle separating the groups. The left side represents the two slower nodes, while the right side is the other 28 nodes. The same principle continues through the remaining steps.

The advantages of this clustering method are that it works with a small number of data points, has a bandwidth parameter that controls the sensibility and the smoothing factor of the density estimation, enabling the adjustment of the clusters spread, and that clusters may be a complex mixture distribution that admits more complex density shapes than a normal distribution. The disadvantages of this clustering are that it does not capture possible sub-populations of interest that share the same mean but have different variances and the control parameter that must be adjusted. Also, this step-based clustering discards temporal knowledge that could be exploited.

Figure 8.3 – Kernel density estimations of the first ten time-steps with the Gaussian kernel and bandwidth 0.01 for all steps considering metrics of Figure 8.2

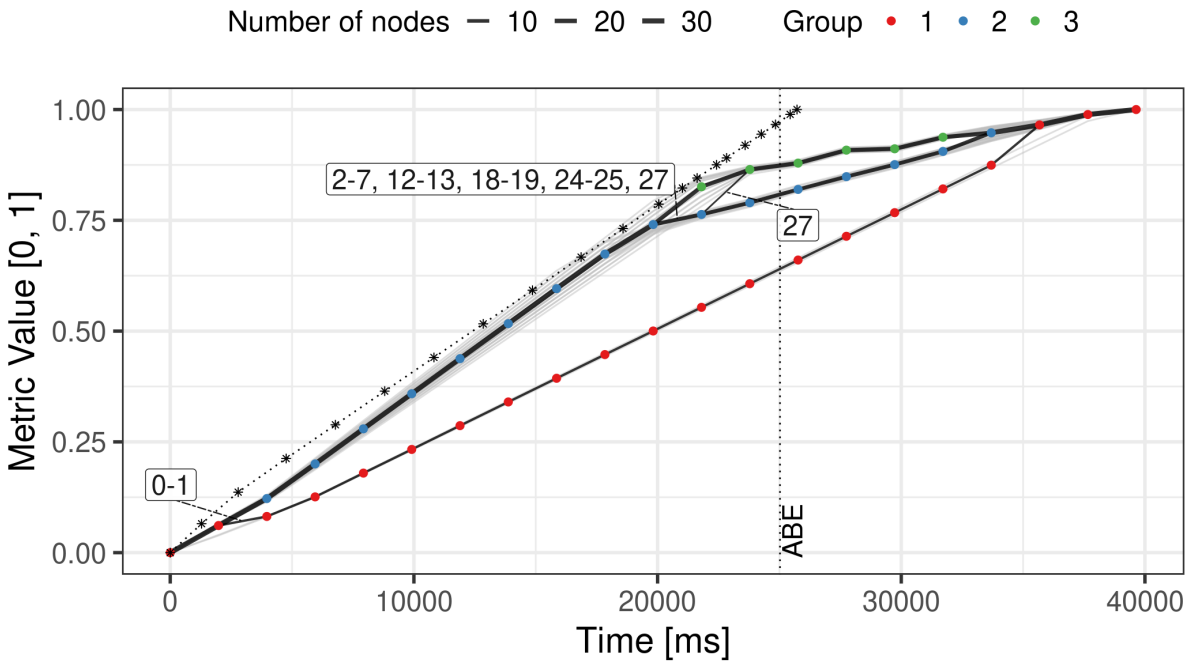


Source: The Author.

8.2.3 Progression Visualization

After having computed the clusters for each time-step, we propose a visualization to show the modal clustering progression. It is then shown in Figure 8.4 for the same data as Figure 8.2. At each time-step, one point is displayed per cluster. A line will connect two points of subsequent steps if they share nodes, and the number of shared nodes determines its thickness. In this way, from time 0 to time ~2000, all nodes are associated with the same cluster and share all nodes. However, at time ~2500, two clusters are detected, and the difference in thickness (thin on the bottom and thick on the top) informs that most of the nodes followed the up path behavior. This is the case as the upper cluster is made of two nodes while the below one is made of 28 nodes. At each disjoint path, the paths with fewer nodes have a label with the nodes that follow it. Also, the visualization keeps the original progression metrics per node (as of Figure 8.2) as background semi-transparent gray lines.

Figure 8.4 – Progression visualization strategy applied to the Chameleon simulation of the LU factorization on 30 nodes



Source: The Author.

However, even if all nodes follow the same behavior, the application does not necessarily achieve the ideal performance. All nodes could share the same problems and be equally late. We plot two additional metrics to aid the observation of such a problem (see Figure 8.4). First, the visualization presents the global ABE as a vertical dashed line. Second, we compute the ABE

per time-step and perform a cumulative sum for each step of their ABE and their previous ones. A series of black points interconnected by a dotted line, usually in the upper part, demonstrates the cumulative per-step ABE. The first point is the ABE per step of the first step, while the second one is the ABE per step of the second step plus the first one, and so on. This metric captures some critical paths and steps' resource restrictions. For example, until step one, there are only tasks that do not utilize GPUs; the ABE of this step will only consider CPUs letting the GPUs idle. This is different from the global ABE, which will try to "pack" all tasks as if they do not have dependencies and compute the bound as if the GPUs were used in the early moments of the execution by future tasks. This difference explains why the per-step ABE may be longer than the global ABE. With those two metrics, one can better perceive the nodes' progress.

8.3 Evaluation on Real Applications

This Section presents experiments on real applications and cases showing how the proposed methodology behaves and helps an analyst.

8.3.1 System and Software

The experiments were conducted with real executions and simulations using Chameleon commit `54e4ec73`, StarPU commit `0fb603d8`, and Simgrid commit `61ee012f`. The version of ExaGeoStat, the application used in Section 8.3.3, is `9518886`. Simulations were used for the LU factorization experiments (Section 8.3.2) with a workload of size 96000×96000, considering 30 machines with eight cores and 2 NVIDIA GTX 1080ti GPUs each (Tupi). For the ExaGeoStat real experiments (Section 8.3.3), two partitions of machines were used with a 96100×96100 workload. First, six nodes with 32 cores each (Chiclet); second, two nodes with 24 cores (2x Intel Xeon 6126), and two Nvidia P100 each (Chifflot). The modal clustering uses the `density` function from `R` 4.2.1 (R Core Team, 2022).

8.3.2 Chameleon predefined abnormal behaviors

Intending to test the methodology's identification power in common problems, we define a set of situations that may happen during the execution of real applications. Such situations are: (a) communications contention problems of one or more nodes; (b) the utilization of a

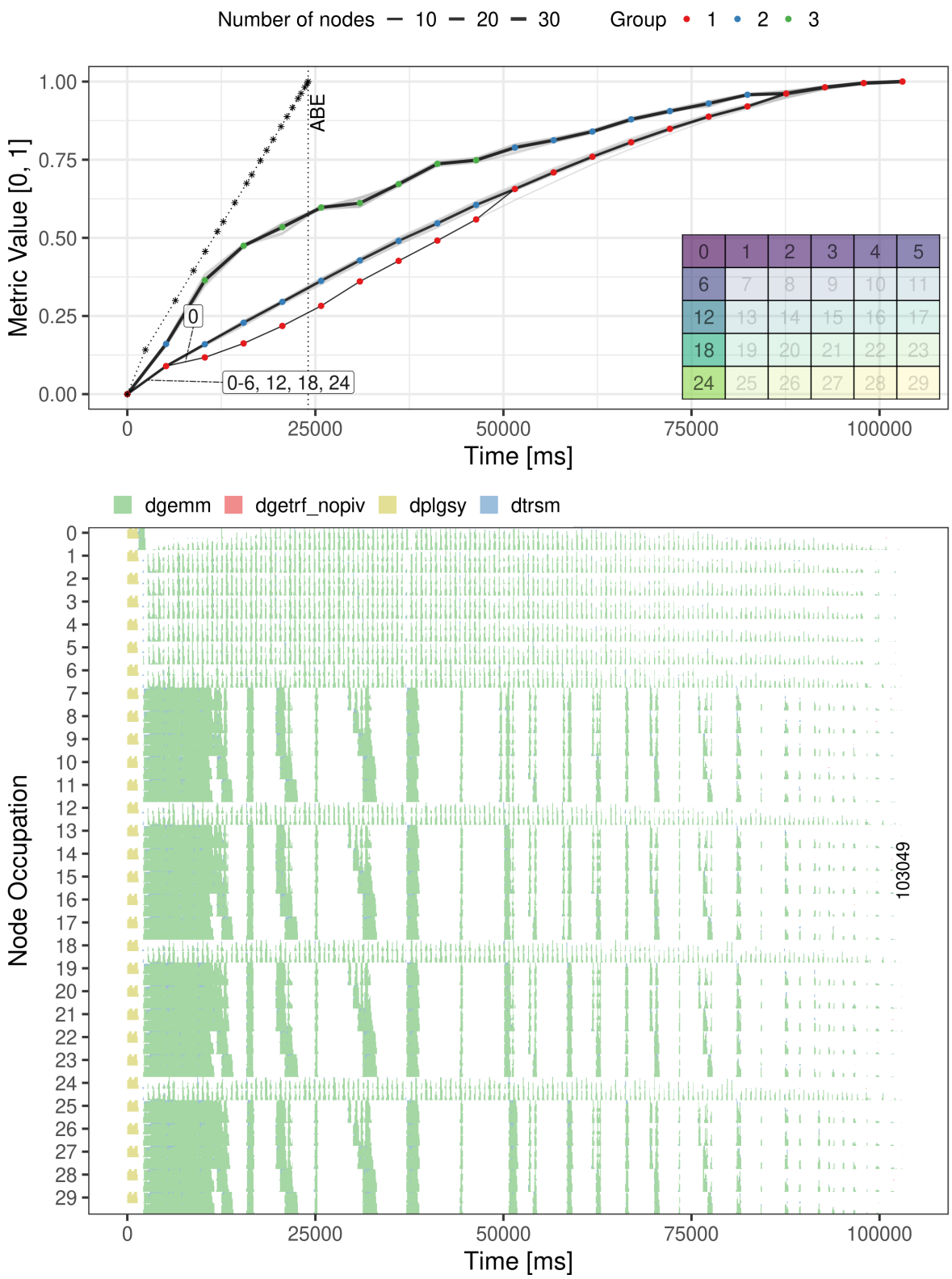
wrong distribution, giving more load to some nodes; (c) global bad behavior that appears to be correct. Those situations were created using StarPU-Simgrid simulations and the Chameleon application. The next Sections detail these situations.

One node with a slower connection. In this case, the first node has only a 1Gb/s network while the others have 25Gb/s. Figure 8.5 shows the progress clustering metric with 20 time-steps on the upper panel and the respective Gantt chart with nodes' resources aggregation on the lower one. Three groups of nodes dominate such execution. The slowest group, comprised of only one node (node zero), is exactly the node with the network reduction. The second slowest group is composed of nodes 1-6, 12, 18, and 24 and is only a little better than group one (the slowest). At the same time, the final and last group is composed of all remaining nodes. The distance from the lower bounds is significant, and while groups one and two started with the slowest progression since the beginning, group three followed the lower bound's progression until time ~15s. The explanation of this group split relies on the application matrix distribution.

The Chameleon library uses the traditional block-cyclic distribution (BLACKFORD et al., 1997) to divide its matrix across many nodes. Considering n nodes, this distribution sets two parameters p and q as $p \times q = n$. These parameters will be used to create a simple partition with p rows and q columns that will be used as a repetitive pattern through all the matrix block distributions. This partition matrix is depicted at the bottom right of the progression metric panel and presents the partition with $p = 5$ and $q = 6$ used in this case. In this distribution, considering the LU factorization kernel, nodes essentially communicate with other nodes that share the same row and columns. The nodes that share the row and columns with the problematic node zero are exactly the nodes of cluster two. This is not necessarily obvious, as one could expect one problematic node to cause a *global* slowdown in the whole system. However, the behavior of nodes that maintain direct communication with node zero is more affected. This observation corroborates that the clustering manages to group significant co-related nodes. From the performance analysis perspective, the nodes with a slower progression would have a priority in the analysis with more detailed tools such as Gantt charts. When observing the Gantt chart solely, it is not clear what is the group of most problematic nodes, though there is a distinction between the two groups in behavior. The progression metric informs such problematic nodes straightforwardly, with the benefit of being a more scalable visualization.

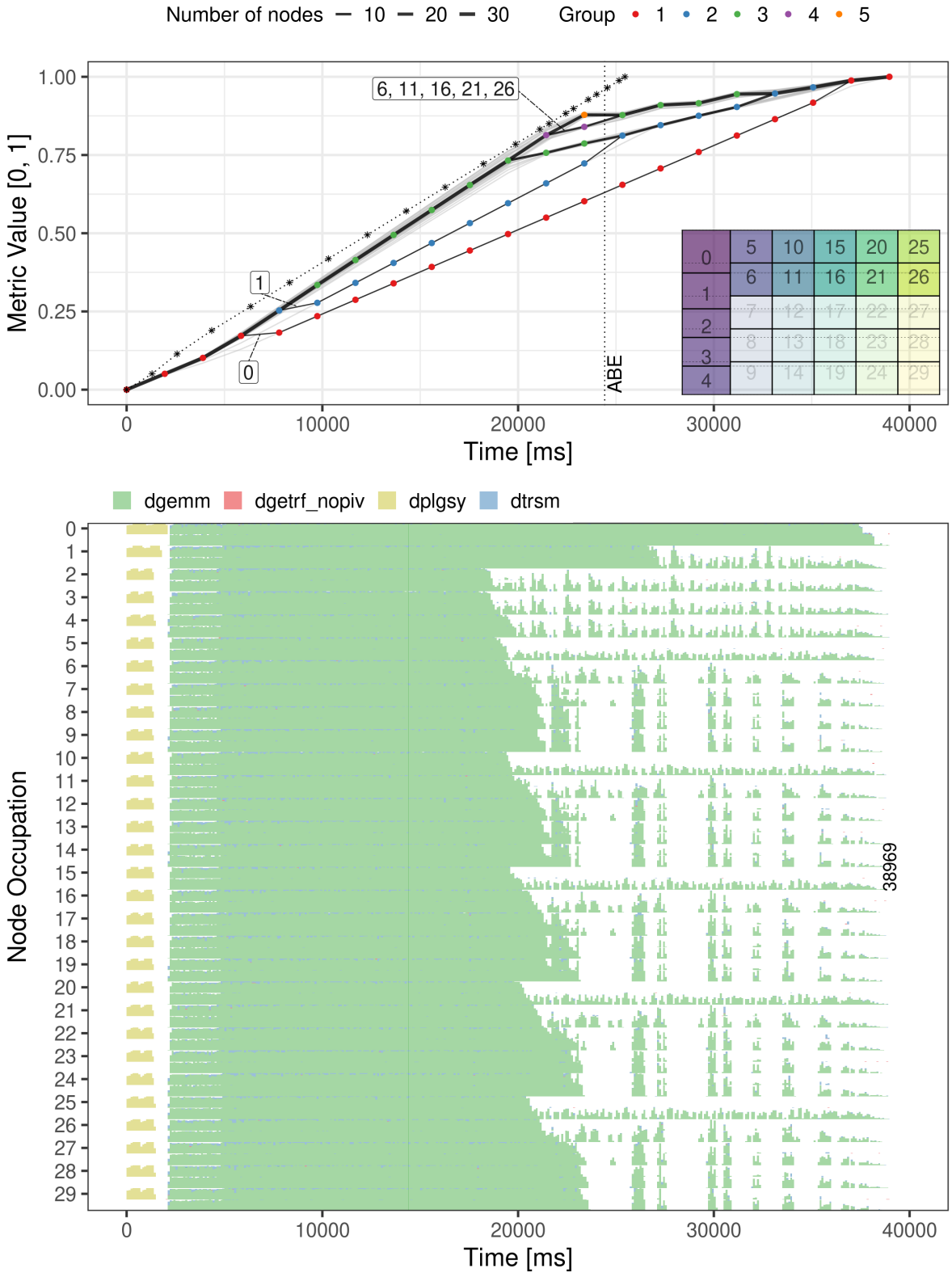
Bad distribution of load across nodes. In this case, we consider the 1D-1D (BEAUMONT et al., 2001a) heterogeneous distribution, giving 50% more load to the first node and 25% more load to the second node. Figure 8.6 presents the progression metrics for this execution in the upper panel with the partition of this heterogeneous distribution on the bottom right. The

Figure 8.5 – Progression visualization strategy and aggregated Gantt chart of Chameleon's LU factorization simulation over 30 nodes where the first node has a slower network



Source: The Author.

Figure 8.6 – Progression visualization strategy and aggregated Gantt chart of Chameleon's LU factorization simulation over 30 nodes where the first and second nodes received 50% and 25% more load, respectively



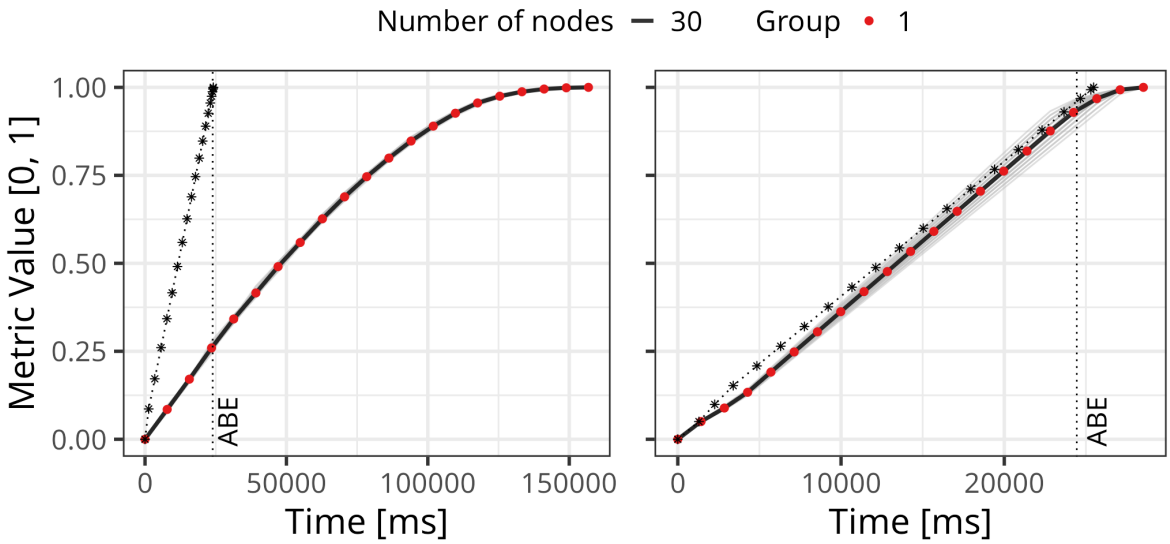
Source: The Author.

bottom panel is the aggregated Gantt chart. Nodes zero and one have larger areas related to the increase in load (though they have the same computational power as the others). There is an expectation that this execution would have problems, but we are interested in checking how the node clusters relate to these problematic nodes.

Groups one and two from time 8s to 24s are solely the nodes zero and one, the problematic ones. There are five clusters at a maximum on time of ~24s. Group three is essentially nodes 2-5, 10, 15, 20, and 25, while group four is nodes 6, 11, 16, 21, 26, and group five is the others. This corresponds to the nodes that directly communicate with the problematic nodes, with a small difference between groups three and four, that the latter communicates more with node one (25% slowdown). While the groups can be recognizable in the Gantt chart, it is unclear to indicate which ones are the most affected. Another situation is that most nodes get affected by the slowdown only at the 20s-25s, progressing until then near the bound.

All nodes with bad network. The network can also globally impact the performance of the application. This case considers a scenario where the infrastructure does not have an optimal network. In this case, all nodes have a 1Gb/s network. Figure 8.7 left presents the clustering metrics for this case. All nodes share the same cluster and present similar behavior. This could lead to the analysis that the execution was well performed. However, the metrics distance to the lower bounds of per-step and global ABE indicates a problem in execution.

Figure 8.7 – Progression visualization strategy of Chameleon's LU factorization simulation over 30 nodes where, on the left, all nodes have slow networks, and on the right, regular networks



Source: The Author.

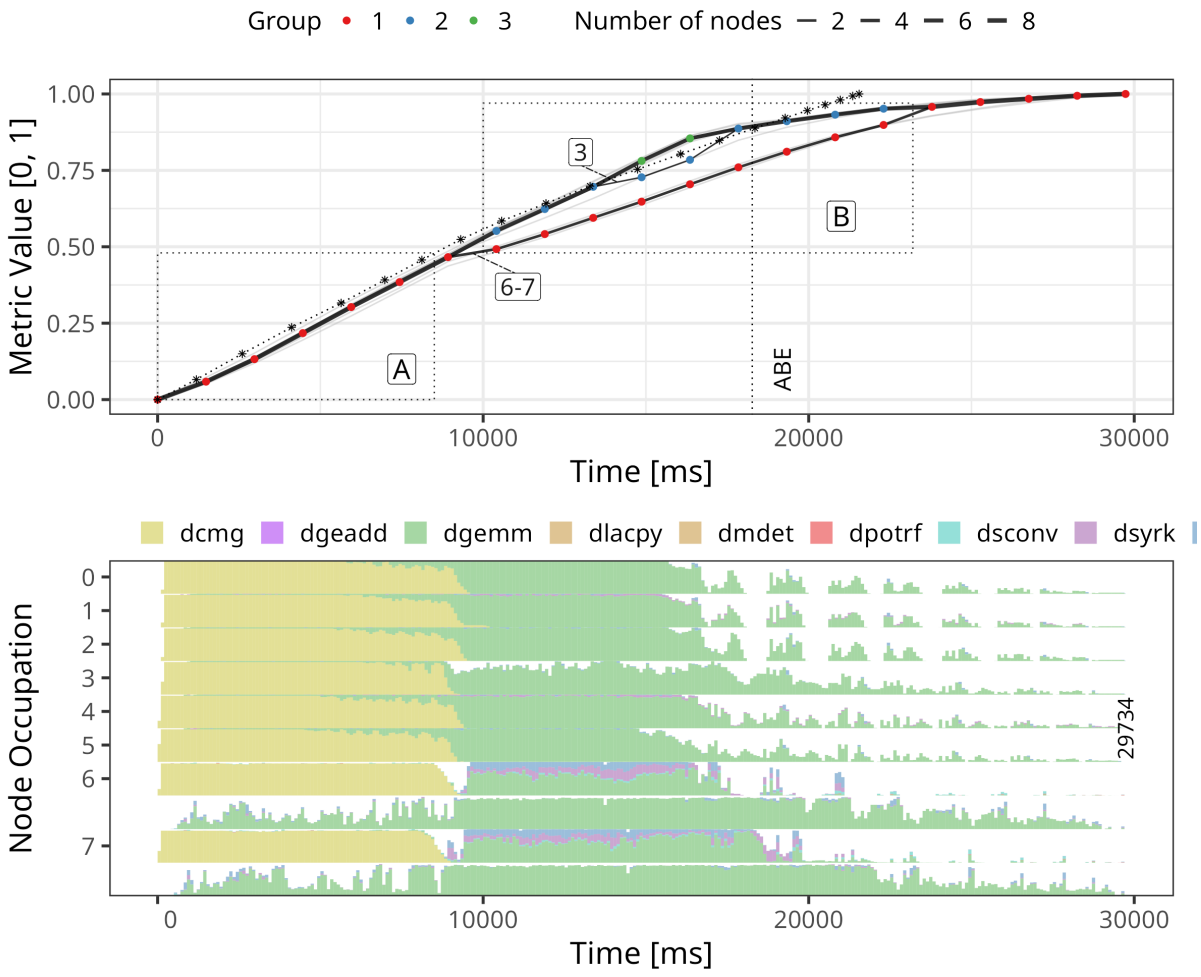
The problem of Figure 8.7 left execution is even more discernible when it is compared

to a correct execution without problems, as shown in Figure 8.7 right side. In the correct case, it is possible to check the proximity of the metric to the lower bound, indicating a well-behaved execution. The correct execution also demonstrates the difficulties of adhering to both ABE bounds at the end, when parallelism diminishes.

8.3.3 A multi-phase application over heterogeneous nodes

In this case, we use eight nodes, where six are CPU-only, and two have CPUs and two GPUs. Figure 8.8 presents the progress cluster visualization on the top panel and the aggregated Gantt chart on the bottom. The first phase is depicted in the Gantt by the yellow tasks and in the A area on the top panel.

Figure 8.8 – Progression visualization strategy (with a bandwidth of 0.15) and aggregated Gantt chart of ExaGeoStat real execution iteration on eight heterogeneous nodes



Source: The Author.

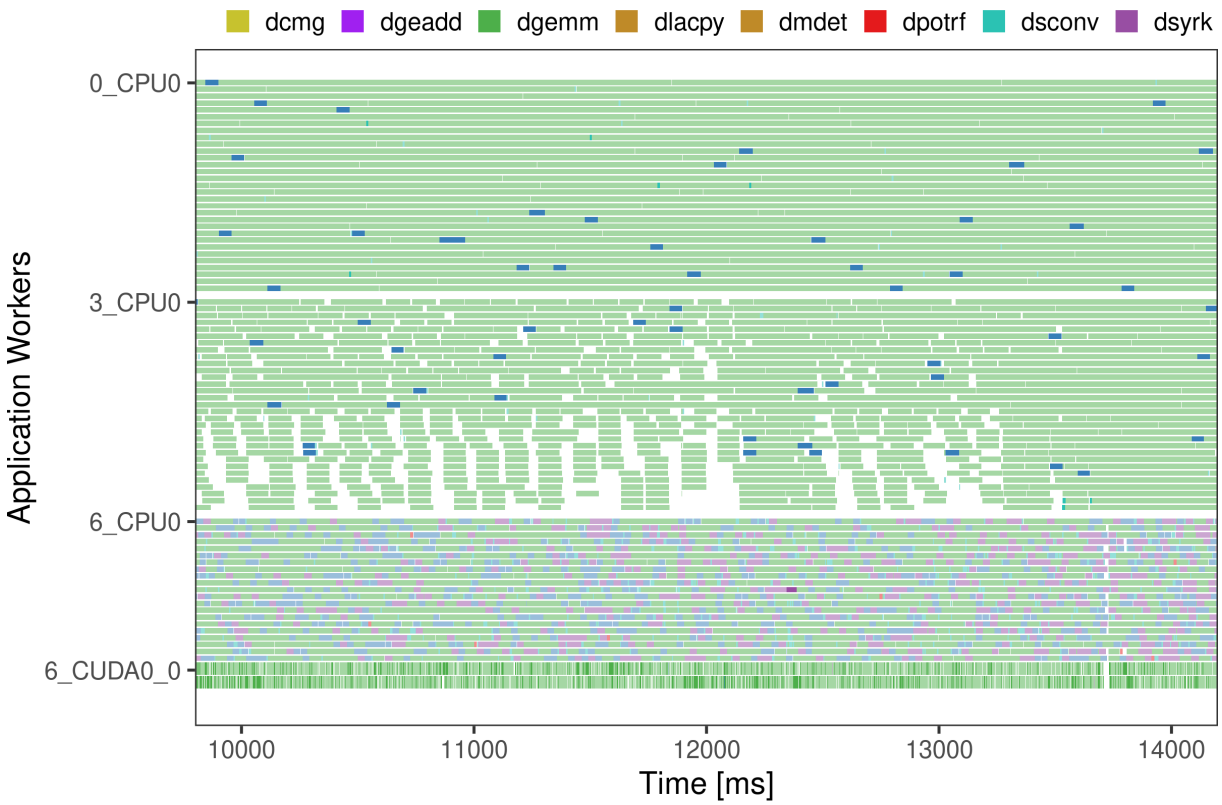
After the first phase, the nodes split into two clusters, B area, and, for a short period, three clusters. The fastest cluster (the top one) comprises six nodes that are exactly the nodes of the first partition, while the slowest cluster (in red) is the two nodes of the second partition (six and seven). With this indication from the progression visualization that some nodes were not performing correctly, further investigation is performed into them. In this case, the slow performance is mainly caused by two problems. First, we used distributions considering the machines' relative power as inputs for this heterogeneous execution. The mean duration of all tasks on all nodes is considered for computing such power. However, when generating such distribution, there was an overestimation of the performance of the main task, `dgemm`, on GPUs by 5%. Second, this power considers a continuous utilization of the GPUs. However, in this case, these workers had short idle periods between many tasks, accounting for 9% idle time. The StarPU scheduler used, `DMDAS`, greedy considers priorities above the locality of data of ready tasks. A task becomes ready when its data is available on that node, i.e., dependencies are met and transferred to that node via MPI. Because of data of high-priority tasks arriving on this node, the scheduler will not be able to pre-fetch such data and will immediately schedule such high-priority tasks above all others, even if their data is still in RAM and has to be transferred to the GPU. Although there is a pipeline of tasks on GPU workers, the transfer duration is higher than the duration of the tasks, and the GPU will wait for the transfer to finish before starting the high-priority task.

This case also illustrates how the Gantt chart can be misleading if used alone. Figure 8.9 shows the Gantt chart for all resources for this execution (of Figure 8.8) for a middle point in the execution for node 0 (Gantt chart behavior equal to nodes 1, 2, 4, 5), node 3, the one that formed a new cluster between the fastest and slowest one, and node 6 (Same behavior as node 7). One may conclude from this Gantt that there is a huge problem on node three because of CPU workers' idle times. However, it is difficult to notice the small GPU idle times that account for a considerable loss in overall performance, as described before. The idle times on node 3 are mainly critical paths of nodes six and seven late tasks that increased communication contention.

8.4 Discussion

Performance analysis of complex HPC applications is the core for improving and accelerating them even more. However, such performance analysis is difficult, as both applications and platforms present many levels of complexity, like heterogeneity, different phases, and sizes. Visualization can aid in this situation by quickly assisting in interpreting the application be-

Figure 8.9 – Traditional Gantt chart of selected nodes of Figure 8.8 execution



Source: The Author.

havior. However, even the best visualizations have limited space to represent such data. Gantt charts are a classical visualization approach for such analysis. However, they are not scalable to the number of nodes and may point in the wrong direction when interpreting results. Although Gantt charts help visually identify resource idleness, they fail to determine or pinpoint the root cause of such inactivity that might be located elsewhere because task dependencies are much more complex in task-based systems.

This Chapter presented a methodology to summarize application behavior, quickly capture the progression of nodes, and indicate problematic ones. It proposes an entry-level strategy to provide an overview of the execution before using other methods, like Gantt charts. This strategy utilizes a progression metric to capture the behavior of the nodes, a clustering of such progression metric to identify node groups of interest and reduce the number of elements to show, and a visualization of such clustering over execution time. We evaluate such strategies over four crafted problematic scenarios with the dense linear algebra library Chameleon, which correctly detected the group of nodes with problems. In a real case with the ExaGeoStat application, it not only handled heterogeneity but indicated the most problematic nodes more straightforwardly than a traditional Gantt Chart. Ultimately, the strategies correctly identified problematic nodes

and provided a new angle that quickly informed the application's progression.

This chapter's main contributions and results were published in PDP 2023 (NESI et al., 2023).

9 FINAL DISCUSSION AND CONCLUSION

Heterogeneity is part of HPC systems, both intra-node with accelerators and inter-node with multiple diverse machines. This system-level (inter-node) heterogeneity may arise because of upgrades over time, handling different applications workloads, or financial decisions. Ultimately, these systems have diverse hardware with vast opportunities for utilization. On the other side, HPC applications are already very complex, having many operations of different behavior. Such applications require modern paradigms that can handle heterogeneous resources, improve application development while allowing systems portability, and avoid unnecessary synchronous barriers between the different operations. The task-based programming paradigm is an example that has such attributes. It uses a dynamic runtime to schedule tasks, and applications are well-defined into a DAG. While this paradigm presents many benefits, there are still challenges when distributing these applications into those system-level heterogeneous resources.

Different problems appear when dealing with the distribution of task-based applications on system-level heterogeneous resources. When considering solely one application operation, one problem is correctly distributing it across a diverse range of computational nodes. While this distribution should consider each resource capability, other behavior aspects, such as critical path and communications, are also important. Nonetheless, a single operation is just one of many parts of the application. There may be many different operations with different behaviors and resource affinities that ideally call for a distinct distribution. Also, in task-based applications, these operations may run asynchronously, overlapping, and each one can choose its relative best resource. In this context, the problem of distributing the application over different resources should now consider multiple overlapping operations adopting various distributions. The number of resources available may also be excessive for a given phase, as sometimes, using fewer resources to reduce some issues is beneficial. However, modeling all behaviors that lead to such problems before the application execution may be challenging, and dynamic adapting during execution is a possible solution. In all these problems, there is a transversal challenge: how to analyze the performance of the applications quickly and thoughtfully. Ultimately, this thesis contributes strategies to all these correlated problems.

This thesis chooses the StarPU task-based runtime and applications of its ecosystem as study subjects. The StarPU runtime has the required flexibility to define distributions, allows asynchronous redistribution, has a simulation feature that can augment experimental scenarios when necessary, and many auxiliary tools for performance analysis. In this sense,

this thesis experiments used the linear algebra library Chameleon, the GeoStatistics machine learning application ExaGeoStat, and the library to analyze large datasets Diodon. The platforms used for the experiments were the Grid5000 infrastructure and the supercomputer SDumont, both providing different levels of heterogeneity. These selected applications present multiple operations that, with the contributions of this thesis, can exploit the system-level heterogeneity to improve performance. The main lines of research and contributions follow.

The first set of contributions, in Chapter 5, focuses on a single application operation distribution. We part from literature algorithms, more specifically, the 1D-1D one. An initial step was to study the behavior of such heterogeneous distributions compared to the classical BC one, even in homogeneous configurations, showing that it can handle an arbitrary number of nodes (including prime numbers). After, this thesis proposes two strategies, inspired by the 1D-1D, that create heterogeneous distributions. The first one considers the critical path and communication alongside the heterogeneous capacities of the resources. It constrains the final workload of the operation to faster resources, decreasing communication and improving critical path. The second strategy performs extra balancing, relaxing previous constraints that some nodes would only communicate with others. A performance analysis compares 1D-1D with the methodology that uses both strategies. The results indicate a slight gain when combining those two strategies in cases where 1D-1D already performed very well. An earlier analysis that predicted a little space for improvement (lower bound) collaborates with such results. However, future problems will reuse the constraining strategy in their cases where the critical path in multi-phase distributions is more critical. In those cases, this strategy presents better results.

The second group of contributions focuses on the problem of applications with multiple phases (operations), tracked in Chapter 6. The Chapter shows that guaranteeing asynchronous execution between operations and tailoring distributions considering their overlap can improve performance, even in homogeneous scenarios. A series of strategies enhance the asynchronous overlap of the phases, improving up to 49% the makespan when considering ExaGeoStat and Diodon applications. Next, it studies the heterogeneous environment. A Linear Program (LP) computes the ideal division of tasks per machine considering heterogeneity and phase interaction overlap. This LP also operates as a lower bound for the application. The relative power of each node for each phase is extracted from the LP's division of tasks result. This power per phase and machine will be the input for the distribution algorithms (of Chapter 5, including the constraining strategy) to compute the distribution of selected operations. Finally, we propose an algorithm to compute a distribution for a precedent phase while minimizing redistribution communications. The algorithms use the following distribution as a reference and the tasks

division of the LP. This methodology improves the performance in the best-studied case by 69% in ExaGeoStat and 73% in Diodon when compared to using a homogeneous distribution strategy on the most powerfully homogeneous cluster (partition) of each scenario. We also present a detailed performance behavior of some cases that suggests that using all available nodes for all phases may be unnecessary.

The third set of contributions (Chapter 7) follows the lead given by the last chapter and focuses on limiting the number of resources in each phase. The problem is that network contention, critical path, or other unexpected behavior may deteriorate the performance more than the possible contribution when adding a computational node. However, modeling such behaviors is challenging in this asynchronous and dynamic scenario. For this reason, this thesis studies the usage of reinforcement learning methods during execution time to model the application's behavior when selecting an arbitrary number of nodes for a given phase. This model serves as a surrogate that the application can consult to guide the next decision. This thesis proposes a method based on the Gaussian Process (GP) with its Upper Confidence Interval (UCB) methodology that assumes a smooth behavior in the search space. The proposal adds HPC knowledge on the method to tune it for this problem, considering limiting the search space, providing an expected trend, and handling discontinuities in makespan behavior. This learning part actuates on a segment of the application, in this case, a long iteration. The ExaGeoStat application presents this structure, where it performs many optimization iterations (with an unavoidable algorithmic synchronization), and each one comprises many asynchronous operations. The surrogate will approximate the duration of one iteration. Before starting each one, the application queries the surrogate and, based on the UCB component, chooses an action that trades exploration and exploitation. In the end, we compare such a method with six others in 16 scenarios, showing that it was the only one that handled all cases, improving the performance up to 51.2% when selecting the number of nodes for the ExaGeoStat factorization phase. It also shows that the overhead of the method is low and that there may be a limited gain (in performance) in optimizing the number of nodes considering all phases.

The final group of contributions comes from a transversal problem to all the others: analyzing the performance of these task-based applications. During the progress of this thesis, any investigation included an extensive and comprehensive analysis of execution traces. Many of these investigations lead to improvements and new features in the StarVZ, a performance analysis workflow package. One example was the addition of the per-node per-resource aggregation Gantt chart. However, in the final stages of this work, the analysis visualization scalability in some experiments was considered a problem. Instead of relying on the Gantt chart directly, which will

never scale as the number of resources increases, another simple visualization would be desirable to point to problematic groups of nodes. That is why Chapter 8 studies a methodology to serve as a visualization summary of the performance behavior of different nodes. It relies on a progression metric sensitive to system-level heterogeneity and distinct tasks (from various operations). The methodology uses this metric for every node in different time steps. Then, because some nodes have similar behavior, it clusters the metric into groups of nodes with similar behavior. The final visualization only shows these groups of nodes, where groups that advance slowly in the progression metric are potentially problematic. This visualization does not need more space as the number of nodes increases. To demonstrate the methodology's usefulness, we craft some bad scenarios with simulation and use real executions of other investigations. Ultimately, the proposed method worked well in all tested cases and helped detect all problematic nodes.

All these contributions are motivated by distributing task-based applications on heterogeneous resources. And in the end, they should be used all together. When an application starts executing, it needs to decide the number of nodes to use per phase, triggering Chapter 7 contributions. This contribution will start by using all nodes in all operations, which requires computing the distributions aware of heterogeneity and asynchronous overlapping operations, leading to using Chapter 6 strategies. Then, those strategies inherently use Chapter 5 to compute the selected phases' final distribution. Afterward, the application's performance can be analyzed using Chapter 8 contributions and the general methodology employed in this Thesis. Finally, one should evaluate if other performance optimizations for the application are worth it.

9.1 Deciding when to stop optimizing

A decision point exists after proposing a solution that works in real experiments: Is this enough? Is the application on the best performance it could get? Is the trade-off between the scientific, engineering, and developing cost to further improvements satisfying? The analysis methodology is important and should consider the cost and gains of possible solutions carefully.

The answer to this question varies depending on many factors, including (i) the application, (ii) the users, (iii) the potential improvements and costs of any change, and (iv) the performance analysis metrics used. This thesis contributes as additional evidence to this question on two fronts.

First, the strategies can be imperfect in some aspects in a practical sense. There will always be idle time in complex applications, minor scheduling decisions that could be corrected when analyzed post-mortem, and the applicability of excessively complex solutions. An example

is that even with a good yet simple model of the multi-phase application in Chapter 6, the final distributions still present some idle time when other unexpected factors happen. In this case, we choose to continue the research and provide Chapter 7 contributions. In the same sense, the strategy in Chapter 7 essentially forces idle time in some resources, as not using all of them is more beneficial in performance. This idle time can be seen as a waste of resources or energy for some. It is complicated to maximize gain in all aspects. However, in this case, this idle time analysis may be worthless and a wrong metric. Although this seems counterintuitive at first look, improving performance by not using some resources can be sufficient (in some cases) to reduce the overall energy consumption compared to the slower scenario that forces the usage in all. Another example is when memory contention has an effect, and using fewer resources to reduce interference is beneficial (MILETTO et al., 2022). Ultimately, it depends on the goal metric, and auxiliary ones (like idle time) may reflect something other than what truly matters.

The second front is that analytical analysis, simulations, and visualization techniques seem a vital ally to deciding when to stop optimizing, as already pointed out by other authors (JAIN, 1991). This situation is further improved by adding diverse scenarios and workloads into consideration. The experiments in Chapter 5 detected a small yet possible improvement when checking the performance distance of the state-of-the-art distribution and the lower bound for the general good case. It is not unexpected that the final improvements of the proposed strategies (in the general case) seemed unsatisfactory for the effort. The first time we analyzed the potential gain, we focused on a specific case that was not representative of the general one. Yet, it is not accidental that the strategy had positive gains when such a case appeared again when considering multi-distributions, a situation that could have been missed if simulations and the analytical analysis had not expanded the view.

Ultimately, this decision, as any other in any domain, has an associated risk. In this case, a risk that the optimization is not worth it. To go blind to uncharted territory trying to fix a "possible problem," focusing on a single non-reproducible experiment seems to have a high chance of leading to wasted effort. However, it is possible to minimize the risk probability by improving the projections of gain and cost with tools like simulation and methods like analytical models. These approaches look appropriate and feasible. All these tools that expand the perspective of the problem, giving new angles, like many of the used visualization techniques, were fundamental for the conduction of this thesis.

Considering all these points, this thesis achieved its goals. All the enumerated problems were studied, and the thesis proposed new solutions. In this way, in the context of this work, it is the moment to stop optimizing. However, we consider the problems in the next Section worth

investigating in the context of future works and directions.

9.2 Future Works

This work opens further perspectives on many topics. The first one includes adjustments in heterogeneous data distribution during execution, an additional action than the static distributions, and dynamically selecting the number of nodes. This can consist of refining distributions in specific operations while considering the multi-phase complexity. Such a situation could further improve balancing with unpredictable behavior. These improvements are ideally application-tailored, exploiting its algorithm and DAG structure. However, a systematic way of changing such distribution in the context of the STF and distributed computing would present technical and scientific problems. As the model is, after the DAG is unrolled in all nodes, data ownership and task changes would require global awareness to guarantee distributed consistency. In this context, the distribution changes decisions should be equal. In the current model, nodes can even submit different tasks, relying on the application design to submit all the necessary tasks in all nodes, leading to the DAG distributed correctness. In this context, one research topic is determining which information the application should pass to the runtime to make heterogeneous optimizations automatically.

Another open question is how to improve the current multi-phase power model with communication, the LP in Chapter 6. Such extra information could approximate it even more to reality. This information is hard to anticipate because of the dynamic scheduler and general complexity of the operations overlap. In the same context of this model, one current problem is that information about one application is specific to it. Although some applications share the same underlying libraries, like Chameleon, only some tasks using the same sizes could be reused through the history-based performance models.

In the context of finding the best set of nodes to use, one open work is to use the efficiency of the configurations instead of only the makespan. Better preparing applications to adapt from an oversized configuration could improve makespan, energy, and efficiency. Specifically, to the proposed strategies, one situation is considering expanding the model of the GP surrogate when augmenting the search space with all phases. The bootstrap of the model and how to prune the search space are open questions. The flexibility of the GP as a surrogate to model performance and efficiency could be used to explore other parameters, allowing HPC applications to adapt better to their systems on each run actively. One possibility is that the GP approach could model distributions that allow mixed precision blocks or other forms of compression specific to the

input data. With such a strategy, one could determine the regions and the number of blocks with lower precision while modeling the trade-off between performance and accuracy.

For the performance analysis and visualizations, some perspectives include always facilitating the development of such applications, allowing developers to quickly understand how their current implementation is behaving. Specifically to the proposed strategy of summarizing thought clustering, future work includes investigating other progression metrics and clustering techniques. Those could be application or situation-tailored.

Ultimately, many traditional techniques, methods, and strategies for homogeneous cases should be revisited to uncover the multiple opportunities and benefits that arise in this heterogeneous context.

9.3 Publications

The main publications related to the thesis follow. The principal results of Chapter 5 were published in ICPADS 2020 (NESI; SCHNORR; LEGRAND, 2020), including some simulation comparisons. A publication in ICPP 2021, which received the best paper award, includes the first works (with ExaGeoStat) in multi-phase applications of Chapter 6 (NESI; LEGRAND; SCHNORR, 2021). We further extended that work to a journal paper in FGCS 2023 (NESI; LEGRAND; MELLO SCHNORR, 2023). The contributions of Chapter 7 were published in IPDPS 2022 (NESI; SCHNORR; LEGRAND, 2022). A publication in PDP 2023 includes the contributions of Chapter 8 (NESI et al., 2023).

- NESI, L. L.; LEGRAND, A.; SCHNORR, L. M. Asynchronous multi-phase task-based applications: Employing different nodes to design better distributions. **Future Generation Computer Systems**, 2023, 147, 119-135.
- NESI, L. L.; PINTO, V. G.; SCHNORR, L. M.; LEGRAND, A. Summarizing task-based applications behavior over many nodes through progression clustering. In: **31st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2023**. Naples, Italy: Euromicro, 2023.
- NESI, L. L.; SCHNORR, L. M.; LEGRAND, A. Multi-phase task-based HPC applications: Quickly learning how to run fast. In: **36th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022**. Lyon, France, IEEE, 2022. pp. 357-367).
- NESI, L. L.; SCHNORR, L. M.; LEGRAND, A. Exploiting system level heterogeneity to improve the performance of a geostatistics multi-phase task-based application. In: **50th**

International Conference on Parallel Processing, ICPP 2021. Chicago, Illinois, USA: ACM, 2021. **(Best Paper Award)**

- NESI, L. L.; SCHNORR, L. M.; LEGRAND, A. Communication-aware load balancing of the LU factorization over heterogeneous clusters. In: **26th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2020**. Hong Kong: IEEE, 2020. p. 54–63.
- NESI, L. L.; SCHNORR, L. Detection, evaluation and mitigation of resource affinity and communication contention problems in a task-based runtime over heterogeneous clusters. In: **Anais do XXI Simpósio em Sistemas Computacionais de Alto Desempenho**. Porto Alegre, RS, Brasil: SBC, 2020. p. 275–286.
- NESI, L. L.; LEGRAND, A.; SCHNORR, L. Impacto da variabilidade de tarefas nas distribuições de aplicações baseadas em tarefas. In: **SBC. Anais da XXI Escola Regional de Alto Desempenho da Região Sul**. Joinville, Brazil, 2021. p. 113–114.
- NESI, L.; LEGRAND, A.; SCHNORR, L. M. Refinando e balanceando o particionamento de aplicações científicas baseadas em tarefas em plataformas heterogêneas. In: **SBC. Anais da XX Escola Regional de Alto Desempenho da Região Sul**. Santa Maria, Brazil, 2020. p. 137–138.

Collaborations and other publications during the thesis period:

- MILETTO, M. C.; NESI, L. L.; SCHNORR, L. M.; LEGRAND, A. Performance analysis of irregular task-based applications on hybrid platforms: Structure matters. **Future Generation Computer Systems**, 2022, 135, 409-425.
- PINTO, V. G.; NESI, L. L.; MILETTO, M. C.; SCHNORR, L. M. Providing in-depth performance analysis for heterogeneous task-based applications with StarVZ. In: **2021 Heterogeneity in Computing Workshop (HCW) / IPDPSW**. Portland, Oregon USA, 2021.
- SOLÓRZANO, A. L. V.; NESI, L. L.; SCHNORR, L. M. Using visualization of performance data to investigate load imbalance of a geophysics parallel application. In: **Practice and Experience in Advanced Research Computing**. New York, NY, USA: Association for Computing Machinery, 2020. (PEARC '20), p. 518–521. ISBN 9781450366892.
- NESI, L. L; SERPA, M.; SCHNORR, L.; NAVAUX, P. O. A. Advances in GPPD-PCAD management with 12-months analysis and perspectives. In: **XVIII Workshop de Processamento Paralelo e Distribuído**. Porto Alegre, Brazil, GPPD, 2020.
- NESI, L. L.; SERPA, M.; SCHNORR, L. M.; NAVAUX, P. O. A. GPPD-PCAD HPC resources management infrastructure description and 10-month statistics. In: **XVII**

Workshop de Processamento Paralelo e Distribuído. Porto Alegre, Brazil, GPPD, 2019.

Courses developed, with extended material published as a book chapter, and presented during this thesis:

- NESI, L. L.; and SCHNORR, L. M. DevOps para HPC: Como configurar um cluster para uso compartilhado. In: Edson Luiz Padoin; Guilherme Galante; Rodrigo Righi. (Org.). **Minicursos da XXIII Escola Regional de Alto Desempenho da Região Sul**. 1ed. Porto Alegre: Sociedade Brasileira de Computação, 2023, v. 23, p. 38-57.
- PINTO, V. G.; NESI, L. L.; and SCHNORR, L. M. Apresentação de Resultados Experimentais para Processamento de Alto Desempenho em R. In: Arthur Lorenzon; Márcio Castro; Mauricio Pillon. (Org.). **Minicursos da XXII Escola Regional de Alto Desempenho da Região Sul**. 1ed.Porto Alegre: SBC, 2022, v. 22, p. 104-124.
- NESI, L. L.; MILETTO, M. C.; PINTO, V. G.; SCHNORR, L. M. Desenvolvimento de aplicações baseadas em tarefas com openmp tasks. In: CHARÃO, A.; SERPA, M. (Ed.). **Minicursos da XXI Escola Regional de Alto Desempenho da Região Sul**. Porto Alegre, Brazil: Sociedade Brasileira de Computação, 2021. chp. 6, p. 129–150.
- NESI, L. L.; PINTO, V. G.; MILETTO, M. C.; SCHNORR, L. M.; THIBAULT, S. Introdução ao desenvolvimento de aplicações paralelas com o paradigma orientado a tarefas e o runtime starpu. In: BOIS, A. D.; CASTRO, M. (Ed.). **Minicursos da XX Escola Regional de Alto Desempenho da Região Sul**. Porto Alegre, Brazil: Sociedade Brasileira de Computação, 2020. chp. 4, p. 70–88.
- DAGOSTINI, J. I.; PINTO, V. G.; NESI, L. L.; SCHNORR, L. M. Are you root? experimentos reprodutíveis em espaço de usuário. In: CHARãO, A.; SERPA, M. (Ed.). **Minicursos da XXI Escola Regional de Alto Desempenho da Região Sul**. Porto Alegre, Brazil: Sociedade Brasileira de Computação, 2021. chp. 3, p. 70–85.
- PINTO, V. G.; NESI, L. L.; SCHNORR, L. M. Boas práticas para experimentos computacionais de alto desempenho. In: BOIS, A. D.; CASTRO, M. (Ed.). **Minicursos da XX Escola Regional de Alto Desempenho da Região Sul**. Porto Alegre, Brazil: Sociedade Brasileira de Computação, 2020. chp. 1, p. 1–19.

REFERENCES

AALI, S. N.; BAGHERZADEH, N. Divisible load scheduling of image processing applications on the heterogeneous star and tree networks using a new genetic algorithm. **Concurrency and Computation: Practice and Experience**, v. 32, n. 10, p. e5498, 2020. E5498 CPE-18-1079.R2.

ABDULAH, S. et al. Exageostat: A high performance unified software for geostatistics on manycore systems. **IEEE Transactions on Parallel and Distributed Systems**, v. 29, n. 12, p. 2771–2784, 2018.

ACOSTA, A.; BLANCO, V.; ALMEIDA, F. Towards the dynamic load balancing on heterogeneous multi-gpu systems. In: **2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications**. USA: IEEE, 2012. p. 646–653.

ACOSTA, A. et al. Dynamic load balancing on heterogeneous multicore/multigpu systems. In: **2010 International Conference on High Performance Computing Simulation**. [S.l.: s.n.], 2010. p. 467–476.

AGULLO, E. et al. Faster, Cheaper, Better – a Hybridization Methodology to Develop Linear Algebra Software for GPUs. In: HWU, W. mei W. (Ed.). **GPU Computing Gems**. USA: Morgan Kaufmann, 2010. v. 2.

AGULLO, E. et al. Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems. **ACM Tr. Math. Softw.**, ACM, New York, NY, USA, v. 43, n. 2, 2016. ISSN 0098-3500.

AGULLO, E. et al. On the autotuning of task-based numerical libraries for heterogeneous architectures. In: **Parallel Computing: Technology Trends**. Amsterdam, Netherlands: IOS Press, 2020. p. 157–166.

AMESTOY, P. et al. Improving multifrontal methods by means of block low-rank representations. **SIAM Journal on Scientific Computing**, v. 37, n. 3, p. A1451–A1474, 2015.

AUER, P.; CESA-BIANCHI, N.; FISCHER, P. Finite-time analysis of the multiarmed bandit problem. **Machine Learning**, v. 47, p. 235–256, 2002.

AUGONNET, C. et al. StarPU-MPI: Task programming over clusters of machines enhanced with accelerators. In: **Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface**. Berlin, Heidelberg: Springer-Verlag, 2012. (EuroMPI'12), p. 298–299. ISBN 978-3-642-33517-4.

AUGONNET, C. et al. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. **Conc. Comp.: Pract. Exp., SI:EuroPar 2009**, John Wiley and Sons, Ltd., v. 23, 2011.

BALAPRAKASH, P. et al. Autotuning in high-performance computing applications. **Proceedings of the IEEE**, v. 106, n. 11, p. 2068–2083, 2018.

BALLARD, G. et al. Minimizing communication in numerical linear algebra. **SIAM Journal on Matrix Analysis and Applications**, v. 32, n. 3, p. 866–901, 2011.

BEAUMONT, O. et al. Recent advances in matrix partitioning for parallel computing on heterogeneous platforms. **IEEE Transactions on Parallel and Distributed Systems**, v. 30, n. 1, p. 218–229, Jan 2019. ISSN 1045-9219.

BEAUMONT, O. et al. Symmetric Block-Cyclic Distribution: Fewer Communications Leads to Faster Dense Cholesky Factorization. In: **SC 2022 - Supercomputing**. Dallas, Texas, United States: IEEE, 2022.

BEAUMONT, O. et al. Comparison of Static and Dynamic Resource Allocation Strategies for Matrix Multiplication. In: **26th IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2015**. Florianopolis, Brazil: IEEE, 2015.

BEAUMONT, O.; EYRAUD-DUBOIS, L.; VÉRITÉ, M. 2D Static resource allocation for compressed linear algebra and communication constraints. In: **2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)**. USA: IEEE, 2020. p. 181–191.

BEAUMONT, O. et al. Static LU decomposition on heterogeneous platforms. **Int. Journal of High Performance Computing Applications**, SAGE Publications, v. 15, p. 310–323, 2001a.

BEAUMONT, O. et al. Matrix multiplication on heterogeneous platforms. **IEEE Trans. Parallel Distributed Systems**, v. 12, n. 10, p. 1033–1051, 2001b.

BEAUMONT, O. et al. Dense linear algebra kernels on heterogeneous platforms: Redistribution issues. **Parallel Computing**, Elsevier, v. 28, n. 2, p. 155–185, 2002a.

BEAUMONT, O. et al. Partitioning a square into rectangles: NP-completeness and approximation algorithms. **Algorithmica**, v. 34, p. 217–239, 2002b.

BECKER, B. A.; LASTOVETSKY, A. Towards data partitioning for parallel computing on three interconnected clusters. In: **Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07)**. USA: IEEE, 2007. p. 39–39.

BELTRÁN, M.; GUZMÁN, A. How to balance the load on heterogeneous clusters. **The International Journal of High Performance Computing Applications**, v. 23, n. 1, p. 99–118, 2009.

BERNSTEIN, A. J. Analysis of programs for parallel processing. **IEEE Transactions on Electronic Computers**, EC-15, n. 5, p. 757–763, 1966.

BHATELÉ, A.; BOHM, E.; KALÉ, L. V. Optimizing communication for charm++ applications by reducing network contention. **Concurrency and Computation: Practice and Experience**, v. 23, n. 2, p. 211–222, 2011.

BLACKFORD, L. S. et al. **ScaLAPACK User's Guide**. USA: Society for Industrial and Applied Mathematics, 1997. ISBN 0-89871-397-8.

BLUMOFE, R. D. et al. Cilk: An efficient multithreaded runtime system. **Journal of parallel and distributed computing**, Elsevier, v. 37, 1996.

BORRELL, R. et al. Heterogeneous CPU/GPU co-execution of cfd simulations on the POWER9 architecture: Application to airplane aerodynamics. **Future Generation Computer Systems**, v. 107, p. 31–48, 2020. ISSN 0167-739X.

BOSILCA, G. et al. Dague: A generic distributed dag engine for high performance computing. **Parallel Computing**, Elsevier, v. 38, n. 1-2, p. 37–51, 2012.

BOSILCA, G. et al. Parsec: Exploiting heterogeneity to enhance scalability. **Computing in Science Engineering**, v. 15, n. 6, p. 36–45, 2013.

BRAMAS, B. Impact study of data locality on task-based applications through the Heteroprio scheduler. **PeerJ Computer Science**, PeerJ, v. 5, p. e190, may 2019.

BRUEL, P. **Toward transparent and parsimonious methods for automatic performance tuning**. Thesis (PhD) — Université Grenoble Alpes; Universidade de São Paulo (Brasil), 2021.

CABRERA, A. et al. A dynamic multi–objective approach for dynamic load balancing in heterogeneous systems. **IEEE Transactions on Parallel and Distributed Systems**, v. 31, n. 10, p. 2421–2434, 2020.

CAO, Q. et al. Flexible data redistribution in a task-based runtime system. In: **2020 IEEE International Conference on Cluster Computing (CLUSTER)**. USA: IEEE, 2020. p. 221–225.

CASANOVA, H. et al. Versatile, scalable, and accurate simulation of distributed applications and platforms. **Journal of Parallel and Distributed Computing**, Elsevier, v. 74, n. 10, p. 2899–2917, jun. 2014.

CHACÓN, J. E. A Population Background for Nonparametric Density-Based Clustering. **Statistical Science**, Institute of Mathematical Statistics, v. 30, n. 4, p. 518, 2015.

CHACÓN, J. E. Mixture model modal clustering. **Advances in Data Analysis and Classification**, Springer, v. 13, n. 2, p. 379–404, 2019.

CHEN, H.-F. **Stochastic approximation and its applications**. Berlin, Heidelberg: Springer Science & Business Media, 2006.

CHEN, Y.-C.; GENOVESE, C. R.; WASSERMAN, L. A comprehensive approach to mode clustering. **Electronic Journal of Statistics**, Institute of Mathematical Statistics and Bernoulli Society, v. 10, n. 1, p. 210 – 241, 2016.

CIERNIAK, M.; ZAKI, M. J.; LI, W. Compile-time scheduling algorithms for a heterogeneous network of workstations. **The Computer Journal**, v. 40, n. 6, p. 356–372, 1997.

CLARKE, D. et al. Hierarchical partitioning algorithm for scientific computing on highly heterogeneous cpu + gpu clusters. In: **Proceedings of the 18th International Conference on Parallel Processing**. Berlin, Heidelberg: Springer-Verlag, 2012. (Euro-Par'12), p. 489–501. ISBN 9783642328190.

CLARKE, D.; LASTOVETSKY, A.; RYCHKOV, V. Dynamic load balancing of parallel computational iterative routines on highly heterogeneous hpc platforms. **Parallel Processing Letters**, World Scientific, Singapore, v. 21, n. 02, p. 195–217, 2011.

CLARKE, D.; LASTOVETSKY, A.; RYCHKOV, V. Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models. In: ALEXANDER, M. et al. (Ed.). **Euro-Par 2011: Parallel Processing Workshops**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 450–459. ISBN 978-3-642-29737-3.

CODD, E. F. Multiprogram scheduling: Parts 1 and 2. introduction and theory. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 3, n. 6, p. 347–350, jun. 1960. ISSN 0001-0782.

DEFLUMERE, A.; LASTOVETSKY, A. Optimal data partitioning shape for matrix multiplication on three fully connected heterogeneous processors. In: LOPES, L.; OTHERS (Ed.). **Euro-Par 2014: Parallel Processing Workshops**. Cham: Springer International Publishing, 2014. p. 201–214. ISBN 978-3-319-14325-5.

DENIS, A. Scalability of the NewMadeleine Communication Library for Large Numbers of MPI Point-to-Point Requests. In: **19th IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing**. Cyprus: IEEE, 2019.

DENNARD, R. et al. Design of ion-implanted mosfet's with very small physical dimensions. **IEEE Journal of Solid-State Circuits**, v. 9, n. 5, p. 256–268, 1974.

DONGARRA, J.; LASTOVETSKY, A. An overview of heterogeneous high performance and grid computing. In: **Engineering the Grid**. USA: Science Publishers, Inc, 2006.

DONGARRA, J. et al. With extreme computing, the rules have changed. **Computing in Science Engineering**, v. 19, n. 3, p. 52–62, May 2017. ISSN 1521-9615.

DONGARRA, J. J.; LUSZCZEK, P.; PETITET, A. The linpack benchmark: past, present and future. **Concurrency and Computation: Practice and Experience**, v. 15, n. 9, p. 803–820, 2003.

DONGARRA, J. J. et al. Top500 supercomputer sites. **Supercomputer**, ASFRA BV, v. 13, p. 89–111, 1997.

DOSIMONT, D. et al. A spatiotemporal data aggregation technique for performance analysis of large-scale execution traces. In: **2014 IEEE International Conference on Cluster Computing (CLUSTER)**. USA: IEEE, 2014.

DUPROS, F. et al. High-performance finite-element simulations of seismic wave propagation in three-dimensional nonlinear inelastic geological media. **Parallel Computing**, v. 36, n. 5, p. 308–325, 2010. ISSN 0167-8191.

DURAN, A. et al. Ompss: a proposal for programming heterogeneous multi-core architectures. **Paral. Proces. Letters**, World Scientific, v. 21, n. 02, 2011.

EYRAUD-DUBOIS, L. **pmtool: Post-mortem Analysis Tool for starpu Scheduling Studies**. 2019. Accessed January 21, 2021. Available from Internet: <https://gitlab.inria.fr/eyrauddu/pmtool>.

FAN, Y. et al. A heterogeneity-aware data distribution and rebalance method in hadoop cluster. In: **2012 Seventh ChinaGrid Annual Conference**. USA: IEEE, 2012. p. 176–181.

FENTON, W. et al. Supporting machine independent parallel programming on diverse architectures. In: **In proceedings of 1991 International Conference on Parallel Processing**. [S.l.: s.n.], 1991. p. pp.

FOLK, M. et al. An overview of the hdf5 technology suite and its applications. In: **Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases**. New York, NY, USA: Association for Computing Machinery, 2011. (AD '11), p. 36–47. ISBN 9781450306140.

FOSTER, I. **Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0201575949.

FREEH, V. W. et al. Analyzing the energy-time trade-off in high-performance computing applications. **IEEE Transactions on Parallel and Distributed Systems**, v. 18, n. 6, p. 835–848, 2007.

FUKUNAGA, K.; HOSTETLER, L. The estimation of the gradient of a density function, with applications in pattern recognition. **IEEE Transactions on Information Theory**, v. 21, n. 1, p. 32–40, 1975.

GAMMEL, M. et al. **Evaluating the Charm++ Runtimes Ability to Cope with Performance Heterogeneity.** [S.l.], 2017.

GARCIA PINTO, V. et al. A visual performance analysis framework for task-based parallel applications running on hybrid clusters. **Concurrency and Computation: Practice and Experience**, v. 30, n. 18, p. e4472, 2018.

GATES, M. et al. Slate: Design of a modern distributed and accelerated linear algebra library. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.]: ACM, 2019. ISBN 9781450362290.

GIOIOSA, R. et al. The minos computing library: Efficient parallel programming for extremely heterogeneous systems. In: **Proceedings of the 13th Annual Workshop on General Purpose Processing Using Graphics Processing Unit**. New York, NY, USA: Association for Computing Machinery, 2020. (GPGPU '20), p. 1–10. ISBN 9781450370257.

GIRAUD, C. **Introduction to high-dimensional statistics**. [S.l.]: Chapman and Hall/CRC, 2021.

GRAHAM, R. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: HAMMER, P.; JOHNSON, E.; KORTE, B. (Ed.). **Discrete Optimization II**. [S.l.]: Elsevier, 1979, (Annals of Discrete Mathematics, v. 5). p. 287–326.

GRAMACY, R. **Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences**. [S.l.]: CRC Press, 2020. (Chapman & Hall/CRC Texts in Statistical Science). ISBN 9781000766523.

GRIGORI, L.; DEMMEL, J. W.; XIANG, H. Communication avoiding gaussian elimination. In: **SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing**. [S.l.: s.n.], 2008. p. 1–12.

HENNESSY, J.; PATTERSON, D. **Computer Architecture: A Quantitative Approach**. [S.l.]: Elsevier Science, 2017. (ISSN). ISBN 9780128119068.

HERAULT, T. et al. Determining the optimal redistribution for a given data partition. In: **2014 IEEE 13th International Symposium on Parallel and Distributed Computing**. [S.l.: s.n.], 2014. p. 95–102.

HERAULT, T. et al. Generic matrix multiplication for multi-gpu accelerated distributed-memory platforms over parsec. In: **2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)**. [S.l.: s.n.], 2019. p. 33–41.

HERRMANN, J. et al. Assessing the cost of redistribution followed by a computational kernel: Complexity and performance results. **Par. Comp.**, Elsevier, v. 52, 2016.

HOFMANN, M.; RÜNGER, G. Flexible all-to-all data redistribution methods for grid-based particle codes. **Concurrency and Computation: Practice and Experience**, v. 30, n. 13, p. e4421, 2018. E4421 cpe.4421.

HOUSSAM-EDDINE, Z. et al. The hpc-dag task model for heterogeneous real-time systems. **IEEE Transactions on Computers**, p. 1–1, 2020.

HPE, U.-U. H. et al. Heterogeneous high performance computing. **ETP4HPC White Paper**, 2022.

HUMPHREY, A.; BERZINS, M. An evaluation of an asynchronous task based dataflow approach for uintah. In: **2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2019. v. 2, p. 652–657.

Intel Corporation. **Intel oneAPI Threading Building Blocks (oneTBB) Documentation**. Santa Clara, CA, USA: Intel Corporation, 2021. Available from Internet: <https://software.intel.com/content/www/us/en/develop/documentation/onetbb-documentation/>. Accessed in: 01/06/2021.

Intel Corporation. **Optimizing Software for x86 Hybrid Architecture**. [S.l.], 2021.

ISAACS, K. E. et al. State of the art of performance visualization. In: **EuroVis - STARs**. [S.l.]: The Eurographics Association, 2014.

JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. [S.l.]: Wiley, 1991. (Wiley professional computing). ISBN 9780471503361.

JAMES, G. et al. **An introduction to statistical learning**. [S.l.]: Springer, 2013.

JEANNOT, E. et al. Communication and topology-aware load balancing in charm++ with treematch. In: **2013 IEEE International Conference on Cluster Computing (CLUSTER)**. [S.l.: s.n.], 2013. p. 1–8.

JEANNOT, E.; WAGNER, F. Scheduling messages for data redistribution: An experimental study. **The International Journal of High Performance Computing Applications**, v. 20, n. 4, p. 443–454, 2006.

JIN, H. et al. Optimizing HPC Fault-Tolerant Environment: An Analytical Approach. In: **2010 39th International Conference on Parallel Processing**. [S.l.: s.n.], 2010. p. 525–534.

JOUPPI, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In: **Proceedings of the 44th Annual International Symposium on Computer Architecture**. New York, NY, USA: Association for Computing Machinery, 2017. (ISCA '17), p. 1–12. ISBN 9781450348928. Available from Internet: <https://doi.org/10.1145/3079856.3080246>.

KAISER, H. et al. HPX: a task based programming model in a global address space. In: **Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models**. New York, NY, USA: Association for Computing Machinery, 2014. (PGAS '14). ISBN 9781450332477.

KALE, L. V.; KRISHNAN, S. Charm++: A portable concurrent object oriented system based on c++. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 28, n. 10, p. 91–108, oct. 1993. ISSN 0362-1340. Available from Internet: <https://doi.org/10.1145/167962.165874>.

KALE, L. V.; ZHENG, G. Charm++ and ampi: Adaptive runtime strategies via migratable objects. **Advanced Computational Infrastructures for Parallel and Distributed Applications**, p. 265–282, 2009.

KALINOV, A.; LASTOVETSKY, A. Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. **J. of Par. and Distr. Comp.**, v. 61, n. 4, p. 520, 2001. ISSN 0743-7315.

KASS, R. E.; WASSERMAN, L. A reference bayesian test for nested hypotheses and its relationship to the schwarz criterion. **Journal of the american statistical association**, Taylor & Francis, v. 90, n. 431, p. 928–934, 1995.

KERGOMMEAUX, J. C. de; STEIN, B. de O. Flexible performance visualization of parallel and distributed applications. **Future Generation Computer Systems**, v. 19, n. 5, p. 735–747, 2003. ISSN 0167-739X.

KEUPER, J.; PREUNDT, F.-J. Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. In: **2nd WS on Machine Learning in HPC Environments**. [S.l.: s.n.], 2016. p. 19–26.

KHALEGHZADEH, H. et al. A novel data partitioning algorithm for dynamic energy optimization on heterogeneous high-performance computing platforms. **Concurrency and Computation: Practice and Experience**, v. 32, n. 21, p. e5928, 2020.

KHALEGHZADEH, H. et al. Bi-objective optimization of data-parallel applications on heterogeneous hpc platforms for performance and energy through workload distribution. **IEEE Transactions on Parallel and Distributed Systems**, v. 32, n. 3, p. 543–560, 2021.

KHALEGHZADEH, H.; MANUMACHU, R. R.; LASTOVETSKY, A. A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous hpc platforms. **IEEE Transactions on Parallel and Distributed Systems**, v. 29, n. 10, p. 2176–2190, 2018.

KHALEGHZADEH, H.; MANUMACHU, R. R.; LASTOVETSKY, A. A hierarchical data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous multi-accelerator numa nodes. **IEEE Access**, v. 8, p. 7861–7876, 2020.

KLEINBERG, J.; TARDOS, E. **Algorithm design**. [S.l.]: Pearson Education India, 2006.

KLINKENBERG, J. et al. Chameleon: Reactive load balancing for hybrid mpi+openmp task-parallel applications. **Journal of Parallel and Distributed Computing**, v. 138, p. 55–64, 2020. ISSN 0743-7315.

KNÜPFER, A. et al. The Vampir performance analysis tool-set. In: **Proc. of the 2nd Intl. Workshop on Parallel Tools for High Performance Computing**. [S.l.]: Springer, 2008. p. 139–155.

KOMATSU, K. et al. Performance evaluation of a vector supercomputer sx-aurora tsubasa. In: IEEE. **SC18: International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.], 2018. p. 685–696.

LI; YU. New bipartite graph techniques for irregular data redistribution scheduling. **Algorithms**, MDPI AG, v. 12, n. 7, p. 142, Jul 2019. ISSN 1999-4893. Available from Internet: <http://dx.doi.org/10.3390/a12070142>.

LIM, Y. W.; BHAT, P.; PRASANNA, V. Efficient algorithms for block-cyclic redistribution of arrays. In: **Proceedings of SPDP '96: 8th IEEE Symposium on Parallel and Distributed Processing**. [S.l.: s.n.], 1996. p. 74–83.

LYNAR, T. M. et al. Clustering obsolete computers to reduce e-waste. **Int. J. Inf. Syst. Soc. Chang.**, IGI Global, USA, v. 1, n. 1, p. 1–10, jan. 2010. ISSN 1941-868X. Available from Internet: <https://doi.org/10.4018/jissc.2010092901>.

MALIK, T.; LASTOVETSKY, A. Optimal matrix partitioning for data parallel computing on hybrid heterogeneous platforms. In: **2020 19th International Symposium on Parallel and Distributed Computing (ISPDC)**. [S.l.: s.n.], 2020. p. 1–11.

MALIK, T.; LASTOVETSKY, A. Towards optimal matrix partitioning for data parallel computing on a hybrid heterogeneous server. **IEEE Access**, v. 9, p. 17229–17244, 2021.

MANUMACHU, R. R.; LASTOVETSKY, A. L. Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution. **Concurrency and Computation: Practice and Experience**, v. 31, n. 4, p. e4958, 2019.

MENG, C. et al. Training deeper models by gpu memory optimization on tensorflow. In: **Proc. of ML Systems Workshop in NIPS**. [S.l.: s.n.], 2017. v. 7.

MENG, Q.; HUMPHREY, A.; BERZINS, M. The uintah framework: a unified heterogeneous task scheduling and runtime system. In: **2012 SC Companion: High Performance Computing, Networking Storage and Analysis**. [S.l.: s.n.], 2012. p. 2441–2448.

MENON, H.; BHATELE, A.; GAMBLIN, T. Auto-tuning parameter choices in hpc applications using bayesian optimization. In: IEEE. **2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.], 2020. p. 831–840.

MENON, H.; KALÉ, L. A distributed dynamic load balancer for iterative applications. In: **SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2013. p. 1–11.

MILETTO, M. C. **Combining prediction models and visualization techniques for enhanced performance analysis of irregular task-based applications**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul (Brésil), 2021.

MILETTO, M. C. et al. Performance analysis of task-based multi-frontal sparse linear solvers: Structure matters. **Future Generation Computer Systems**, v. 135, p. 409–425, 2022. ISSN 0167-739X.

MOHIYUDDIN, M. et al. Minimizing communication in sparse matrix solvers. In: **Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis**. [S.l.: s.n.], 2009. p. 1–12.

MOORE, G. E. et al. **Cramming more components onto integrated circuits**. [S.l.]: McGraw-Hill New York, NY, USA:, 1965.

MOORE, G. E. et al. Progress in digital integrated electronics. In: MARYLAND, USA. **Electron devices meeting**. [S.l.], 1975. v. 21, p. 11–13.

NAGAMOCHI, H.; ABE, Y. An approximation algorithm for dissecting a rectangle into rectangles with specified areas. **Discrete Applied Mathematics**, v. 155, n. 4, p. 523–537, 2007. ISSN 0166-218X.

NESI, L. et al. Visual performance analysis of memory behavior in a task-based runtime on hybrid platforms. In: IEEE. **2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)**. [S.l.], 2019.

NESI, L. L.; LEGRAND, A.; MELLO SCHNORR, L. Asynchronous multi-phase task-based applications: Employing different nodes to design better distributions. **Future Generation Computer Systems**, v. 147, p. 119–135, 2023. ISSN 0167-739X.

NESI, L. L.; LEGRAND, A.; SCHNORR, L. M. Exploiting system level heterogeneity to improve the performance of a geostatistics multi-phase task-based application. In: **50th International Conference on Parallel Processing**. New York, NY, USA: ACM, 2021. (ICPP).

NESI, L. L. et al. Summarizing task-based applications behavior over many nodes through progression clustering. In: **31st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing**. Naples, Italy: [s.n.], 2023.

NESI, L. L.; SCHNORR, L. Detection, evaluation and mitigation of resource affinity and communication contention problems in a task-based runtime over heterogeneous clusters. In: **Anais do XXI Simpósio em Sistemas Computacionais de Alto Desempenho**. Porto Alegre, RS, Brasil: SBC, 2020. p. 275–286. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/wscad/article/view/14076>.

NESI, L. L.; SCHNORR, L. M.; LEGRAND, A. Communication-aware load balancing of the LU factorization over heterogeneous clusters. In: **26th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2020**. Hong Kong: IEEE, 2020. p. 54–63.

NESI, L. L.; SCHNORR, L. M.; LEGRAND, A. Multi-phase task-based hpc applications: Quickly learning how to run fast. In: **2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.: s.n.], 2022. p. 357–367.

NVIDIA. **CUDA Toolkit Documentation v12.1**. Santa Clara, CA, USA: NVIDIA Corporation, 2023. Available from Internet: <https://docs.nvidia.com/cuda/>. Accessed in: 30/05/2023.

OpenMP Architecture Review Board. **OpenMP Application Program Interface Version 5.1**. 2020. Available from Internet: <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-1.pdf>.

PARK, N.; PRASANNA, V.; RAGHAVENDRA, C. Efficient algorithms for block-cyclic array redistribution between processor sets. **IEEE Transactions on Parallel and Distributed Systems**, v. 10, n. 12, p. 1217–1240, 1999.

PATTON, S. et al. Summagen: Parallel matrix-matrix multiplication based on non-rectangular partitions for heterogeneous hpc platforms. In: **2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.: s.n.], 2019. p. 57–68.

PEI, Y. et al. Evaluation of programming models to address load imbalance on distributed multi-core CPUs: A case study with block low-rank factorization. In: **IEEE/ACM Parallel Applications Workshop, Alternatives To MPI**. [S.l.: s.n.], 2019. p. 25–36.

PILLET, V. et al. Paraver: A tool to visualize and analyze parallel code. In: NIXON, P. (Ed.). **Proceedings of WoTUG-18: Transputer and occam Developments**. [S.l.: s.n.], 1995. p. 17–31. ISBN 90-5199-222-X.

PINTO, V. G. **Performance Analysis Strategies for Task-based Applications on Hybrid Platforms**. Thesis (PhD) — Université Grenoble Alpes; Universidade Federal do Rio Grande do Sul (Brésil), 2018.

PINTO, V. G. et al. Providing in-depth performance analysis for heterogeneous task-based applications with starvz. In: **2021 Heterogeneity in Computing Workshop (HCW)**. [S.l.: s.n.], 2021.

PRYLLI, L.; TOURANCHEAU, B. Efficient block cyclic data redistribution. In: BOUGÉ, L. et al. (Ed.). **Euro-Par'96 Parallel Processing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. p. 155–164. ISBN 978-3-540-70633-5.

R Core Team. **R: A Language and Environment for Statistical Computing**. Vienna, Austria, 2022. Available from Internet: <https://www.R-project.org/>.

RACA, V.; MEHOFER, E. clustercl: comprehensive support for multi-kernel data-parallel applications in heterogeneous asymmetric clusters. **The Journal of Supercomputing**, Springer, p. 1–33, 2020.

RICO-GALLEGO, J. A. et al. Performance evaluation of model-driven partitioning algorithms for data-parallel kernels on heterogeneous platforms. **Computational and Mathematical Methods**, v. 2, n. 1, p. e1017, 2020. E1017 cmm4.1017.

RICO-GALLEGO, J.-A.; LASTOVETSKY, A. L.; DÍAZ-MARTÍN, J.-C. Model-based estimation of the communication cost of hybrid data-parallel applications on heterogeneous clusters. **IEEE Transactions on Parallel and Distributed Systems**, v. 28, n. 11, p. 3215–3228, 2017.

RICO-GALLEGO, J. A. et al. A tool to assess the communication cost of parallel kernels on heterogeneous platforms. **The Journal of Supercomputing**, v. 76, p. 4629–4644, 2019.

ROBINSON, B. H. E-waste: An assessment of global production and environmental impacts. **Science of The Total Environment**, v. 408, n. 2, p. 183–191, 2009. ISSN 0048-9697.

ROBISON, A. D. Intel® threading building blocks (TBB). In: ____. **Encyclopedia of Parallel Computing**. Boston, MA: Springer US, 2011. p. 955–964. ISBN 978-0-387-09766-4.

ROSARIO, V. et al. Fast and low-cost search for efficient cloud configurations for hpc workloads. In: **Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho**. Porto Alegre, RS, Brasil: SBC, 2021. p. 144–155. ISSN 0000-0000.

ROUSTANT, O.; GINSBOURGER, D.; DEVILLE, Y. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. **Journal of Statistical Software**, v. 51, n. 1, p. 1–55, 2012.

SALETORE, V.; JACOB, J.; PADALA, M. Parallel computations on the charm heterogeneous workstation cluster. In: **Proceedings of 3rd IEEE International Symposium on High Performance Distributed Computing**. [S.l.: s.n.], 1994. p. 203–210.

SANDERS, J.; KANDROT, E. **CUDA by example: an introduction to general-purpose GPU programming**. [S.l.]: Addison-Wesley Professional, 2010.

SCHNORR, L.; STEIN, B. de O.; KERGOMMEAUX, J. C. de. Paje trace file format, version 1.2.5. **Laboratoire d'Informatique de Grenoble, France, Technical Report**, 2013.

SCHNORR, L. M.; LEGRAND, A. Visualizing more performance data than what fits on your screen. In: **Tools for High Performance Computing 2012**. [S.l.]: Springer, 2013. p. 149–162.

SHIRAZI, B. A.; KAVI, K. M.; HURSON, A. R. **Scheduling and load balancing in parallel and distributed systems**. [S.l.]: IEEE Computer Society Press, 1995.

SILVANO, C. et al. Autotuning and adaptivity in energy efficient hpc systems: the antarex toolbox. In: **Proceedings of the 15th ACM International Conference on Computing Frontiers**. [S.l.: s.n.], 2018. p. 270–275.

SRINIVAS, N. et al. Gaussian process optimization in the bandit setting: No regret and experimental design. In: **Proceedings of the 27th International Conference on International Conference on Machine Learning**. Madison, WI, USA: Omnipress, 2010. (ICML'10), p. 1015–1022. ISBN 9781605589077.

STANISIC, L. et al. Fast and accurate simulation of multithreaded sparse linear algebra solvers. In: **The 21st IEEE International Conference on Parallel and Distributed Systems**. Melbourne, Australia: [s.n.], 2015.

STANISIC, L. et al. Faithful Performance Prediction of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. **Concurrency and Computation: Practice and Experience**, Wiley, p. 16, may 2015.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

TESSER, R. K. et al. Performance modeling of a geophysics application to accelerate overdecomposition parameter tuning through simulation. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, p. e5012, 2017.

THIBAULT, S. **On Runtime Systems for Task-based Programming on Heterogeneous Platforms**. Thesis (Habilitation à diriger des recherches) — Université de Bordeaux, dec. 2018.

TOPCUOGLU, H.; HARIRI, S.; WU, M.-y. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE transactions on parallel and distributed systems**, IEEE, v. 13, n. 3, p. 260–274, 2002.

UNDERWOOD, K. D.; HEMMERT, K. S.; ULMER, C. D. From silicon to science: The long road to production reconfigurable supercomputing. **ACM Trans. Reconfigurable Technol. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 2, n. 4, sep. 2009. ISSN 1936-7406. Available from Internet: <https://doi.org/10.1145/1575779.1575786>.

VASUDEVAN, R. et al. G-charm: An adaptive runtime system for message-driven parallel applications on hybrid systems. In: **Proceedings of the 27th International ACM Conference on International Conference on Supercomputing**. New York, NY, USA: Association for Computing Machinery, 2013. (ICS '13), p. 349–358. ISBN 9781450321303.

WALKER, D.; OTTO, S. Redistribution of block-cyclic data distributions using mpi. **Concurrency: Practice and Experience**, v. 8, n. 9, p. 707–728, 1996.

WEAVER, V. M. et al. Measuring energy and power with papi. In: **2012 41st International Conference on Parallel Processing Workshops**. [S.l.: s.n.], 2012. p. 262–268.

WIDMER, R. et al. Global perspectives on e-waste. **Environmental Impact Assessment Review**, v. 25, n. 5, p. 436–458, 2005. ISSN 0195-9255. Environmental and Social Impacts of Electronic Waste Recycling.

WILSON, J. M. Gantt charts: A centenary appreciation. **European Journal of Operational Research**, Elsevier BV, v. 149, n. 2, p. 430–437, Sep 2003. ISSN 0377-2217.

WILSON, R. J. **Introduction to graph theory**. [S.l.]: Pearson Education India, 1979.

WU, W. et al. Hierarchical dag scheduling for hybrid distributed systems. In: **2015 IEEE International Parallel and Distributed Processing Symposium**. [S.l.: s.n.], 2015. p. 156–165.

XU, C.; LAU, F. C. **Load balancing in parallel computers: theory and practice**. [S.l.]: Springer Science & Business Media, 1996.

YANG, L.; SCHOPF, J.; FOSTER, I. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In: **SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing**. [S.l.: s.n.], 2003. p. 31–31.

APPENDIX A — RESUMO EXPANDIDO EM PORTUGUÊS

A computação de alto desempenho (HPC) fornece os meios para que aplicações complexas utilizem uma enorme capacidade computacional e sejam concluídas num tempo viável. Até o início da década de 2000, o crescimento do número de transístores por chip seguia a lei de Moore, que estabelecia que este número duplicaria a cada dois anos (MOORE et al., 1965; MOORE et al., 1975). Este crescimento permitiu um aumento exponencial do poder de computação, tal como indicado por Dennard (DENNARD et al., 1974). No entanto, os limites na dissipação de calor travaram o aumento da frequência dos chips na década de 2000 (HENNESSY; PATTERSON, 2017), parando o aumento exponencial do desempenho de núcleos individuais (DONGARRA et al., 2017). Surgiram muitas alternativas para continuar o crescimento do poder computacional, incluindo várias técnicas de CPU e aceleradores para computação especializada.

A combinação de um subconjunto destes recursos resulta num único nó computacional heterogêneo. Esta heterogeneidade intra-nó tem sido amplamente utilizada nos últimos anos em muitas aplicações pela comunidade de HPC (SANDERS; KANDROT, 2010; AUGONNET et al., 2011; MENG et al., 2017; PINTO, 2018). No entanto, a exploração desta heterogeneidade é complexa, uma vez que requer a divisão correta da aplicação nesses múltiplos e diversos recursos, tendo em conta a adaptabilidade e o desempenho do algoritmo em cada um (SANDERS; KANDROT, 2010; AUGONNET et al., 2011).

Do ponto de vista dos programadores, lidar com esta heterogeneidade requer um conhecimento profundo do sistema alvo e a antecipação do comportamento da aplicação. Nos métodos e ferramentas de programação tradicionais, como o MPI e as aplicações BSP, a utilização de recursos heterogêneos é complicada, uma vez que o estilo de programação é inteiramente imperativo. Estas abordagens de programação especificam onde uma carga de trabalho é executada estaticamente, com barreiras e comunicações síncronas para controlar o fluxo das múltiplas operações ou fases da aplicação. Com este nível de sincronismo, um pequeno erro na divisão da carga de trabalho resultaria num desempenho não ideal com recursos inativos. Entretanto, esse erro é quase inevitável, uma vez que tanto os sistemas como as aplicações são complexos. Além disso, garantir a capacidade (portabilidade) da aplicação para ser executada em diferentes sistemas exige modificações extensas no seu código-fonte, aumentando a possibilidade de erros.

No entanto, a disponibilidade de processadores distintos e a diversidade das necessidades das aplicações encorajam a heterogeneidade ao nível do sistema. Assim, é cada vez mais comum que os sistemas HPC incluam muitas máquinas com diversas configurações de hardware organizadas em clusters ou partições homogêneas para melhor acomodar as necessidades das

aplicações. Se utilizada corretamente, esta heterogeneidade permite à aplicação ajustar um comportamento interno específico e melhorar o desempenho, explorando a heterogeneidade em vez de sofrer com ela.

As principais infra-estruturas de HPC, os supercomputadores, estão normalmente localizadas em grandes centros de investigação no meio acadêmico ou industrial. Os locais (as instalações da infra-estrutura) podem acolher sistemas diferentes, como indica a lista TOP500 (DONGARRA et al., 1997) de supercomputadores, oferecendo a heterogeneidade já mencionada. As razões pela heterogeneidade destes locais de supercomputadores podem ser três: (i) por concepção, quando a infra-estrutura possui tais configurações para visar cargas de trabalho diversas, (ii) limitações financeiras, quando, por exemplo, apenas um subconjunto de nós recebe aceleradores devido a restrições orçamentais, e (iii) as atualizações naturais da infra-estrutura ao longo do tempo. A lista TOP 500 apenas considera e classifica um conjunto homogêneo de nós como um sistema. Esta restrição vem de uma limitação do *benchmark* adotado, HPL (DONGARRA; LUSZCZEK; PETITET, 2003), uma vez que ele não funciona adequadamente em configurações heterogêneas ao nível do sistema.

Muitos desafios algorítmicos surgem na transição para cenários heterogêneos (DONGARRA; LASTOVETSKY, 2006; BEAUMONT et al., 2019). Em primeiro lugar, a quantidade ideal de dados por nó será diferente. Dependendo da estrutura de dados da aplicação, esta divisão pode não ser trivial. Embora uma divisão simples seja suficiente em estruturas de dados unidimensionais, caso todas as porções de dados terem um custo computacional idêntico, são necessárias estratégias especializadas para dimensões superiores ou quando determinados dados têm comportamentos ou necessidades de potência computacional diferentes (BEAUMONT et al., 2001a). Além disso, a existência de diferentes quantidades de dados por nó resulta em diferentes quantidades de comunicação entre nós, uma vez que os nós que processam mais dados terão de comunicar mais. Esta diferença de comunicação significa que o compromisso ideal entre comunicação e computação por nó é mais crítico em cenários heterogêneos do que em cenários homogêneos. O mesmo se aplica se a quantidade de computação por elemento de dados diferir, resultando num caminho crítico que as capacidades heterogêneas dos nós computacionais irão influenciar. Por último, existe uma perspectiva técnica de programação. As distribuições mais simples, como as cíclicas, são mais fáceis de implementar em paradigmas tradicionais como o MPI, e as funções de distribuição são mais claramente empregadas. No entanto, surgem padrões de comunicação irregulares quando se lida com distribuições arbitrárias, e podem ser necessários outros paradigmas ou middlewares para reduzir a complexidade de desenvolvimento e a manutenibilidade da aplicação.

Além disso, as aplicações podem ser compostas por diferentes fases com diferentes necessidades computacionais, possivelmente admitindo diferentes distribuições ideais de dados. Essas fases também podem explorar os recursos dos nós de forma diferente, alterando ainda mais a distribuição ideal para cada uma delas. Por exemplo, as fases que compreendem a geração de dados são tipicamente mais apropriadas para CPUs, enquanto algumas operações de computação intensiva, como os kernels clássicos de álgebra linear, poderiam usar aceleradores para aumentar o desempenho.

As limitações destes paradigmas tradicionais incluem o limitado tratamento da heterogeneidade intra-nó, a baixa eficiência da programação, o sincronismo desnecessário e a portabilidade limitada dos recursos. Em conjunto, elas impulsionam o ressurgimento do paradigma de programação baseado em tarefas (BOSILCA et al., 2013; DURAN et al., 2011; AUGONNET et al., 2011; WU et al., 2015; THIBAULT, 2018). Este paradigma adota uma forma mais declarativa de programação e utiliza um *runtime* para tomar decisões, incluindo o escalonamento dinâmica do trabalho (tarefas) durante a execução. A aplicação descreve tarefas individuais e dependências de dados e estrutura-as num gráfico acíclico dirigido (DAG). Os benefícios desta abordagem incluem a diminuição da complexidade de programação, do tratamento automático de comunicações irregulares e a cooperação de recursos heterogêneos intra-nós.

Um *runtime* é responsável pela programação das tarefas respeitando as dependências, utilizando heurísticas. Esta abordagem também permite o fácil assincronismo de tarefas de múltiplas operações (ou fases). Além disso, alguns *runtimes* efetuam automaticamente a transferências de dados inter e intra-nós com base na estrutura do DAG, reduzindo a carga de desenvolvimento. O programador da aplicação ainda precisa dar muitas dicas ao *runtime* para o ajudar durante a execução, mas uma vez formulado, o código é altamente portátil para vários sistemas. Exemplos de tais *runtimes* são ParSEC (BOSILCA et al., 2013), OmpSS (DURAN et al., 2011), e StarPU (AUGONNET et al., 2011), este último utilizado neste trabalho. Estes *runtimes* modernos fornecem uma abstração de programação de alto nível com a flexibilidade necessária, facilitando a investigação e o desenvolvimento de estratégias sofisticadas de distribuição de dados estáticos em nós heterogêneos. Esta abordagem parece mais adequada para combinar a heterogeneidade ao nível de sistemas e as necessidades das aplicações.

Assim, o paradigma de programação baseado em tarefas oferece muitas oportunidades para implementar estratégias algorítmicas elaboradas e distribuições refinadas necessárias para enfrentar os muitos desafios que surgem da utilização de recursos heterogêneos. Além disso, este trabalho utiliza informações relacionadas com o DAG e tarefas para orientar as decisões das estratégias e a análise do desempenho.

A.1 Contribuições da Tese

As aplicações de HPC exigem um poder computacional considerável fornecido principalmente por supercomputadores. Esses recursos computacionais podem apresentar uma heterogeneidade ao nível de sistema quando há dois ou mais grupos de nós, cada grupo com hardware e poder computacional diferentes. Além disso, as aplicações podem apresentar um comportamento heterogêneo interno devido a operações distintas ou a várias fases que podem ser executadas de forma diferente em cada recurso. O aumento da heterogeneidade entre nós em supercomputadores e a dificuldade de programá-los incentivam o uso de paradigmas robustos de programação paralela, como o baseado em tarefas. Embora esse paradigma ofereça flexibilidade, portabilidade e dinamismo suficientes para lidar com um cenário tão complexo, muitos problemas ainda precisam ser resolvidos. Todas essas circunstâncias tornam o problema da **distribuição dessas aplicações baseados em tarefas entre nós heterogêneos** desafiador, embora surjam muitas oportunidades para melhorar o desempenho e o uso de recursos.

O principal objetivo deste trabalho é fornecer estratégias e métodos para melhorar a distribuição de aplicações baseados em tarefas em recursos heterogêneos no nível do sistema. Nesse contexto, muitos desafios devem ser considerados para uma aplicação atingir o desempenho correto. Os desafios abordados nesta tese são: (1) As distribuições para cada fase da aplicação devem considerar não apenas o balanceamento de carga, mas também a razão entre a comunicação e o caminho crítico; (2) Criar várias distribuições para aplicações multifásicas com necessidades diferentes e, ao mesmo tempo, considerar sua interação de sobreposição; e (3) encontrar o número ideal de nós por tipo para cada fase. A estrutura da tese segue.

Capítulos preliminares. O Capítulo 2 detalha as aplicações baseadas em tarefas, as características dos *runtimes* e o ecossistema StarPU. Ele se concentra na estrutura do DAG das aplicações a partir dos algoritmos e como o *runtime* escalona as tarefas em muitos recursos. O capítulo também apresenta as aplicações usadas no restante da tese. A primeira é a biblioteca de álgebra linear Chameleon. A segunda é a aplicação de geo estatística ExaGeoStat. E a terceira é a biblioteca de análise de grandes conjuntos de dados Diodon.

O Capítulo 3 apresenta trabalhos relacionados. Começando com o problema geral da distribuição de uma aplicação (principalmente as estruturadas de álgebra linear 2D) em cenários homogêneos e heterogêneos. Depois, introduz trabalhos e técnicas clássicas sobre o equilíbrio de carga, seguidos de uma apresentação de trabalhos relacionados com a gestão de aplicações multifásicas, tendo em conta tanto as distribuições múltiplas como a comunicação. Por fim, apresentamos brevemente como alguns algoritmos de aprendizagem automática e de

aprendizagem por reforço são utilizados em problemas de HPC relacionados.

Metodologia. O Capítulo 4 apresenta a metodologia da tese para experimentos controlados e uma análise de desempenho. A metodologia baseia-se em execuções reais com um controle experimental, simulações com uma avaliação da sua fiabilidade no nosso contexto, e análise destes experimentos utilizando métricas analíticas, análise de rastros e visualização.

Contribuições. O Capítulo 5 estuda as distribuições de uma operação possível, a fatoração LU, que poderia ser expandida para outras operações semelhantes de álgebra linear. Ele fornece as seguintes contribuições. **(a)** Uma estratégia que melhora o desempenho das aplicações ao reduzir as comunicações no caminho crítico. Essa situação ocorre quando o paralelismo no DAG diminui. A abordagem restringe a distribuição para usar menos recursos no final do algoritmo. **(b)** Uma estratégia para melhorar o balanceamento da carga computacional de uma determinada distribuição estática, considerando várias tarefas e recursos heterogêneos, aumentando a comunicação. **(c)** Uma metodologia para combinar (a) e (b).

O Capítulo 6 considera o problema de aplicações multifásicas que envolvem possivelmente várias distribuições heterogêneas. O estudo de casos se baseia nas aplicações multifásicas ExaGeoStat e Diodon. O capítulo tem as seguintes contribuições. **(d)** Otimizações para melhorar o assincronismo de fases em aplicações. **(e)** Estratégia para gerar distribuições heterogêneas eficientes para aplicações multifásicas com diferentes afinidades de desempenho de recursos, considerando a sobreposição de fases. **(f)** Uma técnica para derivar outra distribuição de uma distribuição principal que reduz as comunicações ao realizar a redistribuição entre ambas.

O Capítulo 7 apresenta estratégias para a aplicação aprender e se adaptar ativamente aos melhores nós heterogêneos que ela pode acessar. O capítulo tem as seguintes contribuições. **(g)** Uma análise das principais características desse problema (estrutura e ruído) e explicação de por que as técnicas genéricas de otimização e aprendizado provavelmente falharão. Essa análise motiva variações específicas de uma técnica de aprendizado por reforço baseada no Processo Gaussiano. **(h)** Uma avaliação abrangente do desempenho com 16 máquinas heterogêneas e cargas de trabalho diferentes que comparam as soluções propostas com outros métodos de otimização genéricos (Brent, Bandits, GP-UCB). Entre esses vários métodos, a variante baseada em GP é o único método robusto e parcimonioso para alcançar rapidamente a configuração ideal em vários cenários. **(i)** Uma implementação real do método para permitir que a aplicação se adapte durante a execução, demonstrando a baixa sobrecarga.

O Capítulo 8 discute técnicas e métodos para analisar o comportamento de aplicações baseadas em tarefas. Especificamente, **(j)** técnicas com foco extra na heterogeneidade da plataforma e do progresso da aplicação.

O Capítulo 9 conclui esta tese com as principais contribuições, trabalhos futuros e a lista de publicações.

Por fim, um site com materiais complementares para a tese está disponível publicamente em: <https://gitlab.com/lnesi/thesis-companion.git>. Ele inclui os dados, os scripts e os rastros dos experimentos para replicar a análise e as figuras.

A.2 Paradigma de Programação Baseado em Tarefas

O paradigma de programação baseado em tarefas (THIBAULT, 2018), também conhecido como *Data Flow Scheduling* (DONGARRA et al., 2017) ou *Asynchronous Many Task* (AMT) e *runtimes* (HUMPHREY; BERZINS, 2019), usa uma abordagem mais descritiva, não imperativa, para definir uma aplicação. As aplicações expressam seus algoritmos internos com tarefas e dependências sem definir explicitamente onde está o paralelismo e onde e quando essas tarefas são executadas. Um runtime decide o escalonamento e o posicionamento das tarefas durante a execução usando algoritmos internos e heurísticos. Devido a essa flexibilidade e ao acoplamento fraco na plataforma, Dongarra et al. (2017) aponta que a programação baseada em tarefas será o paradigma desejado para os sistemas *exaflop*. Embora a utilização de tarefas e programação dinâmica seja um conceito antigo (CODD, 1960), ele está ganhando popularidade em muitos projetos novos e modernos (DONGARRA et al., 2017; THIBAULT, 2018; HOUSSAM-EDDINE et al., 2020).

No caso do StarPU, o *runtime* usado neste trabalho, o uso sucessivo de dados por meio de várias tarefas cria um fluxo que expressa a ordem de execução da aplicação e constrói o DAG de tarefas, esta estratégia se chama Fluxo de Tarefas Sequenciais (STF) (AGULLO et al., 2016). Nessa abordagem, a aplicação tem apenas uma *thread* e submete as tarefas sequencialmente, construindo o DAG.

Embora os *runtimes* possam suportar tecnicamente a execução em recursos heterogêneos no nível do sistema, seu desempenho depende de como a aplicação é estruturada (DAG, por exemplo) e de como o desenvolvedor a distribui entre os nós. Em ambos os casos, o conhecimento de como uma aplicação com operações assíncronas é executada em recursos heterogêneos no nível do sistema é limitado e tem um grande potencial a ser explorado. Dessa forma, são desejáveis abordagens ou estratégias que ajudem as aplicações baseados em tarefas executarem em recursos heterogêneos no nível do sistema.

A.3 Trabalhos Relacionados: Distribuição de Carga

A divisão de dados e computação para nós distribuídos é um elemento fundamental na programação paralela. A distribuição de carga pode otimizar vários objetivos. Os objetivos mais comuns são melhorar o desempenho da aplicação, diminuindo os tempos ociosos ou equilibrando a carga dos processadores, reduzir a comunicação total para evitar a contenção da rede ou aumentar os cálculos paralelos disponíveis.

As estratégias de distribuição de dados e computação entre os recursos podem ser uma abordagem estática, dinâmica ou híbrida (SHIRAZI; KAVI; HURSON, 1995). A partição estática é realizada uma vez antes de toda a computação e permanece estacionária durante a execução da aplicação. Esse problema de particionamento depende do domínio da aplicação. Por exemplo, as operações lineares de álgebra clássica devem particionar uma matriz 2D entre os recursos. No entanto, a solução ideal para a estratégia 2D estática é um problema NP-completo (BEAUMONT et al., 2002a). As estratégias dinâmicas dependem da distribuição da carga de trabalho durante a execução da aplicação. A maioria dos *runtimes* baseados em tarefas, incluindo o StarPU (AUGONNET et al., 2011), adota essa abordagem para o escalonamento de recursos dentro do nó. Por fim, uma abordagem híbrida tem técnicas estáticas e dinâmicas. Um exemplo híbrido é o módulo StarPU-MPI, que usa escalonamento dinâmico para tarefas dentro do nó e distribuição de dados estáticos entre diferentes nós, principalmente devido a preocupações com a escalabilidade. A aplicação deve informar essa distribuição estática, enquanto um dos muitos algoritmos heurísticos do StarPU executa o escalonamento dinâmico de tarefas dentro do nó.

Assim, estão disponíveis partições que permitem o processamento eficiente de uma única operação. No entanto, é necessária uma distribuição correta de colunas e linhas para obter um balanceamento de carga adequado durante a execução de algoritmos mais complexos, como o algoritmo LU. Beaumont et al. (2001a) propôs um procedimento de distribuição simples (1D-1D), o qual é assintoticamente ótimo, independentemente da partição retangular inicial.

Os problemas de (i) distribuição de uma operação em recursos heterogêneos; (ii) várias distribuições e redistribuições para aplicações de várias fases; e (iii) aprendizado e adaptação a comportamentos inesperados, como escalabilidade de nós, são estudados principalmente com nós homogêneos e separadamente na literatura, sem a oportunidade de combinar todos eles. Este trabalho mostra que esses problemas são desafios ao distribuir aplicações baseadas em tarefas em recursos heterogêneos no nível do sistema. Portanto, é preciso resolver todos esses problemas para melhorar o desempenho da aplicação. Esta tese apresenta suas contribuições de distribuição de carga de uma perspectiva micro para uma macro. Primeira, computando

a distribuição de uma operação (Capítulo 5) que será usada posteriormente ao estudar várias distribuições em várias operações (Capítulo 6). Em seguida, essas estratégias serão novamente empregadas ao ajustar o número de nós a serem usados por fase (Capítulo 7).

A.4 Métodos Experimentais e de Análise

Os métodos experimentais e de análise deste trabalho consistem em três estágios. O primeiro estágio (Metodologia experimental) é a realização de experimentos em ambientes controlados, usando execuções reais e de simulação. A execução gera rastros que descrevem em detalhes cada experimento realizado. O próximo estágio (Metodologia de análise de desempenho) é a análise desses rastros usando métricas, visualizações e ferramentas de ciência de dados. Essa análise leva ao estágio final, a proposta de soluções e métodos para melhorar o comportamento. Uma reinicialização da metodologia ocorre com novos candidatos de otimização. As estratégias oferecidas nesta tese refletem um ciclo iterativo de experimentos, análises e propostas de soluções.

Os experimentos deste trabalho são divididos em duas categorias: execuções reais e simulações. Na primeira, uma plataforma real executa diretamente a aplicação. O programa efetivamente calcula e cumpre seu objetivo, chegando a uma solução de algoritmo real e correta. Diferentemente, o caso da simulação pode substituir alguns cálculos por modelos estatísticos, concentrando-se em aproximar o comportamento da duração e não a solução final do algoritmo. Este trabalho usa o Simgrid (CASANOVA et al., 2014) como meio para simular plataformas HPC distribuídas, trabalhando com o StarPU com seu módulo StarPU+Simgrid (STANISIC et al., 2015b). O StarPU+Simgrid gera rastros que se aproximam fielmente do comportamento da aplicação e da utilização de recursos (STANISIC et al., 2015a). No entanto, o StarPU+Simgrid não executa cálculos (tarefas) e substitui dados por informações nulas.

A base da metodologia de análise são os rastros de execução e os dados complementares que o runtime e aplicação coletam. A tese usa estratégias diferentes para a coleta de dados, o cálculo de métricas de desempenho, a visualização do comportamento do desempenho da aplicação e a investigação direta dos dados. Uma das principais ferramentas usadas é o StarVZ, um worklfow de análise de desempenho para aplicações baseadas em tarefas que utiliza R e Tidyverse.

Com a metodologia de análise, é possível verificar alguns problemas, como a subutilização de recursos em um determinado ponto, o desequilíbrio de recursos, a falta de tarefas prontas, pontos de sincronização em iterações, valores inadequados para qualquer métrica ap-

resentada e operações relacionadas à memória. Todos esses insights de análise permitem que o analista proponha possíveis soluções em algumas entidades de execução (aplicação, runtime, biblioteca, sistema). Essas soluções exigem experimentação, validação da análise e repetição da metodologia. Esse processo foi aplicado a todos os problemas da tese, com os resultados levando ao próximo objeto de estudo, resultando em um processo iterativo.

A.5 Estratégias de Distribuições Heterogêneas para Álgebra Linear

O Capítulo 5 concentra-se em uma única operação e propõe estratégias para gerar distribuições cíclicas estáticas para procedimentos de álgebra linear que precisam equilibrar a carga em muitas iterações. Embora os algoritmos de distribuição cíclica e heterogênea, como o 1D-1D, forneçam resultados assintoticamente ótimos, eles ainda deixam algumas oportunidades para melhorias e para a criação de distribuições mais refinadas. Além disso, ter uma carga de equilíbrio perfeita não garante o melhor desempenho. Devido ao caminho crítico e à heterogeneidade dos nós, às vezes pode ser melhor ter algum desequilíbrio com a carga extra nos nós mais rápidos. As estratégias discutidas neste capítulo atuam em uma única operação, e suas ideias são essenciais ao examinar a interação de várias fases com várias distribuições nos Capítulos 6 e 7.

Este capítulo propõe algoritmos (1D-1D C e 1D-1D C+S) para gerar distribuições considerando os momentos da aplicação com baixo paralelismo, comunicação e equilíbrio de carga. Experimentos com a fatoração LU demonstram como as distribuições 1D-1D podem ser benéficas mesmo para um grupo homogêneo de nós híbridos quando comparadas com BC, apresentando escalabilidade muito mais estável. O capítulo propõe duas novas estratégias para considerar um conjunto heterogêneo de máquinas híbridas, para as quais o 1D-1D também é quase ideal. A primeira usa menos nós no final da operação, intensificando gradualmente a computação nas máquinas potentes. A segunda move adicionalmente blocos para nivelar o trabalho cumulativo. A técnica de restrição permite a otimização do final da execução, enquanto o balanceamento de blocos possibilita o alcance de um equilíbrio de carga quase ideal entre as iterações, mas aumenta a comunicação. Uma combinação cuidadosa dessas técnicas proporciona melhorias seletivas de desempenho em relação ao 1D-1D original, tanto no balanceamento de carga quanto no tempo de execução.

Esse estudo também indica que as distribuições 1D-1D são um excelente ponto de partida, que não é tão fácil de melhorar. Na prática, quando se considera apenas uma operação de álgebra linear, não está claro se o uso sistemático de 1D-1D C e 1D-1D C+S é vantajoso, por requererem

um bom modelo de desempenho e poderem induzir uma sobrecarga de comunicação. Entretanto, essas duas estratégias são ferramentas que podem ser aplicadas a situações específicas. Quando surgem problemas no final da execução devido ao paralelismo limitado, à comunicação excessiva ou ao caminho crítico mal posicionado, a estratégia de restrição pode ajudar. Uma reorganização adicional pode beneficiar os casos em que o balanceamento fornecido pelo 1D-1D pode ser aprimorado e a comunicação não é um problema (em uma rede de alta velocidade e baixa latência, por exemplo). Esse primeiro estudo permite a identificação de quantidades essenciais para o reequilíbrio e as consequências de ter distribuições irregulares na execução geral. Além disso, esse estudo mostra que as distribuições balanceadas não perfeitas (como a versão restrita) podem apresentar benefícios em algumas situações.

Por fim, as aplicações podem ter várias fases de algoritmo que exigem diferentes distribuições heterogêneas. A interação entre essas distribuições deve ser cuidadosamente estudada ao usar um sistema totalmente assíncrono, como nos *runtimes baseados* em tarefas. O tempo ganho em alguns recursos ao desativá-los no final do algoritmo pode ser usado para as fases subsequentes, o que significa que as distribuições desequilíbradas para uma fases podem ser compatíveis na geração de uma execução completa e equilibrada. Se os recursos ociosos de uma fase forem usados inteligentemente pela as fases subsequentes, pode-se esperar melhorias no desempenho. Ainda, a estratégia restrita pode ser usada para otimizar o caminho crítico no final da distribuição. Quando há nós com e sem GPUs, as tarefas no caminho crítico que exploram melhor as GPUs se beneficiariam de uma distribuição desigual concentrada em nós com esses aceleradores.

A.6 Estratégias Heterogêneas para Aplicações Multifásicas

Muitas aplicações têm fases diferentes, cada uma com necessidades computacionais diferentes. Por exemplo, as fases que compreendem a entrada e a geração de dados geralmente são mais adequadas às CPUs, enquanto as operações de computação intensiva, como os kernels clássicos de álgebra linear, podem explorar eficientemente os aceleradores. Essa heterogeneidade interna da aplicação pode ser mais bem adequada quando combinada com a heterogeneidade do sistema, melhorando o equilíbrio da carga ao oferecer mais opções de distribuição. Como as fases individuais não usam todos os recursos igualmente, uma aplicação requer certa liberdade para executar as fases simultaneamente para melhorar o uso de recursos e o desempenho. Os desafios para permitir a sobreposição de fases incluem dificuldades de programação e algorítmicas. Do ponto de vista do programador, a sobreposição de fases geralmente

é difícil de obter em aplicações paralelas tradicionais. No entanto, o assincronismo é possível no paradigma baseado em tarefas se o DAG for estruturado corretamente. Mas, mesmo depois que a sobreposição de fases for implementada, a aplicação ainda enfrentará o desafio de encontrar uma distribuição eficiente para os nós disponíveis. A combinação desses desafios faz com que a maioria das aplicações perca uma enorme oportunidade de usar a heterogeneidade ao nível do sistema. Mesmo que a demanda computacional distinta das fases torne esta heterogeneidade uma opção interessante para melhorar a distribuição da carga.

O Capítulo 6 estuda e propõe estratégias para distribuir essas aplicações de várias fases em recursos com heterogeneidade ao nível de sistema. A investigação usa as aplicações ExaGeoStat e Diodon. Em ambas as aplicações, a falta de sobreposição entre as fases faz com que elas percam uma enorme oportunidade de desempenho: recursos poderosos (GPU) poderiam trabalhar mais. Com a ordem de execução correta na fase de geração, as tarefas para a GPU poderiam ficar prontas mais cedo, aumentando sua utilização. No entanto, quando a sobreposição é corrigida, as fases têm necessidades diferentes que poderiam explorar conjuntos diferentes de nós para melhorar o desempenho. Essa heterogeneidade significa que as distribuições de várias fases devem considerar essa sobreposição. E a distribuição de uma determinada fase deve reconhecer o desempenho das tarefas de outras fases nos recursos e nós heterogêneos.

O Capítulo apresenta otimizações para aprimorar a execução assíncrona baseada em tarefas das fases de ambas as aplicações. Em ambas as aplicações, as mesmas ideias de aprimoramento de estruturas específicas do DAG, como prioridades, dicas e dependências, levam a ganhos de desempenho. No ExaGeoStat, cinco operações se sobrepõem, enquanto no Diodon, a sobreposição de computação ocorre entre o Gram e a primeira etapa de multiplicação de matriz. Com a sobreposição correta, o capítulo continua e oferece estratégias para gerar uma distribuição estática heterogênea no nível do sistema que considera o custo computacional e a sobreposição das fases. Um programa linear modela o fluxo da aplicação considerando as tarefas e a heterogeneidade dos recursos. O pode computacional relativo é extraído do programa linear e, posteriormente, usada em algoritmos de distribuição tradicionais para algumas fases de computação intensiva. O capítulo também fornece um algoritmo para inferir uma distribuição de uma determinada fase usando a distribuição da fase seguinte, mantendo o equilíbrio regional e reduzindo a sobrecarga de redistribuição.

A avaliação de desempenho deste capítulo demonstra que as três categorias gerais de otimização melhoram a sobreposição de fases assíncronas. Essas otimizações permitem que as tarefas das últimas fases sejam executadas mais cedo e reduzem a ociosidade de recursos, melhorando o desempenho em cenários homogêneos de 29% a 46%. Ao considerar o caso

heterogêneo, as estratégias propostas calculam o poder computacional relativo ideal para cada fase em cada grupo de nós. Todas as estratégias de distribuição heterogênea melhoram o desempenho em até 69% em comparação com uma configuração homogênea simples no ExaGeoStat e de 24% a 73% no Diodon.

A.7 Aprendendo e Adaptando em Sistemas Heterogêneos Complexos

O Capítulo 7 estuda o problema de descobrir o conjunto ideal de nós heterogêneos a serem usados ao considerar aplicações multifásicas. Embora o paradigma baseado em tarefas atenue amplamente a sobrecarga de comunicação, efeitos imprevistos (por exemplo, contenção de rede ou sincronizações complexas entre nós) continuam sendo possíveis e particularmente difíceis de modelar. Nesse contexto, encontrar um número adequado de nós de computação para cada fase pode ser particularmente difícil de prever. Portanto, métodos que aprendam e se adaptem dinamicamente a esses cenários complexos e melhorem o desempenho ao longo do tempo são desejáveis. A aplicação ExaGeoStat é uma boa candidata para estudar essas estratégias. Ela tem muitas iterações em que podem ocorrer decisões de aumento ou diminuição de nós. Muitas aplicações têm essa estrutura de iterações estáveis (carga de trabalho estacionária), mas cuja duração total é difícil de prever em algumas configurações, pois algumas fases são bem escaláveis, enquanto outras não. Isso significa que a aplicação pode aprender e se adaptar ativamente ao melhor conjunto de nós heterogêneos que pode acessar entre as iterações.

É possível informar o runtime sobre a redistribuição de dados durante o envio de tarefas, fazendo com que as tarefas enviadas a seguir alterem seu nó de execução de acordo. Essas redistribuições podem refletir novas distribuições e usarem mais ou menos nós. O runtime do StarPU moverá todos os dados para o lugar certo de forma assíncrona, de forma sobreposta à computação.

O capítulo considera sete estratégias diferentes na literatura e apresenta a proposta GP-descontínua. A avaliação experimental inclui uma visão geral dos resultados em 16 cenários diferentes, uma análise detalhada passo a passo, a avaliação da sobrecarga de custos da estratégia e possíveis expansões da proposta. Os resultados experimentais são obtidos usando ambientes reais e de simulação com configurações heterogêneas no nível do sistema.

Os resultados mostram que uma estratégia de aprendizagem por reforço baseada em processo gaussiano com informações extras pode encontrar rapidamente a melhor configuração de nós para a fase principal (fatoração). A abordagem superior, GP-descontínua, usa mecanismos de limite para filtrar o espaço de pesquisa, modelar a diferença do makespan para um

limite inferior (com algum conhecimento do cenário) e usar variáveis fictícias e uma tendência linear para modelar as descontinuidades causadas pelas distribuições. Esse método forneceu os melhores resultados consistentes em todos os 16 cenários estudados. Outro método (mais simples) com desempenho particularmente bom é o UCB-struct, que considera apenas pontos específicos. No entanto, esse método só encontrará a melhor configuração às vezes por estar limitado a não pesquisar todo o espaço, mas apenas os opções que consideram os grupos homogêneos completos. No final, o uso de todos os nós para a fase de geração e apenas um subconjunto aprendido para a fatoração proporcionou um aumento de desempenho de até 51,2% em comparação com o uso de todos os nós para ambas as fases. Esses resultados demonstraram que a aplicação pode descobrir durante a execução a ação (número de nós) que pode tomar para melhorar o desempenho e se adaptar aos recursos heterogêneos.

A.8 Resumindo o Comportamento das Aplicações

A análise de desempenho das aplicações de computação de alto desempenho é uma etapa essencial para obter o desempenho correto. No entanto, a complexidade das aplicações e sistemas, incluindo os vários níveis de heterogeneidade (HPE et al., 2022), acrescenta desafios consideráveis a essa atividade. A análise de desempenho de aplicações HPC por meio da visualização é considerada uma metodologia vantajosa (GARCIA PINTO et al., 2018) por permitir uma compreensão facilitada de grandes quantidades de dados de rastreamento (SCHNORR; LEGRAND, 2013). Entretanto, mesmo com essa estratégia, o espaço para visualização é limitado, e as estratégias podem fornecer percepções errôneas. Esse é o caso do Gantt Chart, que se concentram na utilização de recursos, mas podem ocultar a progressão real das aplicações. Embora as visualizações de desempenho adaptadas para uma aplicação (MILETTO et al., 2022) ou aspectos do *runtime* (NESI et al., 2019) possam atenuar alguns desses problemas, as estratégias gerais que resumem o comportamento geral do desempenho ajudariam nessa questão.

O Capítulo 8 propõe uma metodologia de análise de desempenho por meio de uma visualização introdutória. Seu objetivo é verificar a progressão de aplicações baseados em tarefas em nós computacionais individuais para indicar momentos e grupos de nós de interesse. Essa metodologia é composta por três elementos: uma métrica de progressão por nó que emprega a estrutura das aplicações baseados em tarefas, um método de agrupamento para classificar nós e reduzir os elementos a serem exibidos e uma visualização introdutória desses componentes.

A avaliação dessas estratégias inclui quatro cenários problemáticos criados com a biblioteca de álgebra linear densa Chameleon, onde a estratégia proposta detectou corretamente o

grupo de nós com problemas. Em um caso real com a aplicação ExaGeoStat, a estratégia não apenas lidou com a heterogeneidade, mas indicou os nós mais problemáticos de forma mais direta do que um gráfico de Gantt tradicional. Por fim, as estratégias identificaram corretamente os nós problemáticos e forneceram uma nova visão que informa rapidamente a progressão da aplicação.

A.9 Discussão Final e Conclusão

A heterogeneidade faz parte dos sistemas de HPC, tanto no interior do nó com aceleradores quanto no nível de sistema com várias máquinas diferentes. Essa heterogeneidade ao nível de sistema (entre nós) pode surgir devido a atualizações ao longo do tempo, ao suporte de diferentes cargas de trabalho de aplicações ou a decisões financeiras. Em última análise, esses sistemas têm hardware diversificado com grandes oportunidades de utilização. Por outro lado, as aplicações de HPC já são muito complexas, com várias operações com comportamento diferente. Essas aplicações exigem paradigmas modernos que possam lidar com recursos heterogêneos, melhorar o desenvolvimento de aplicações e, ao mesmo tempo, permitir a portabilidade aos sistemas e evitar barreiras síncronas desnecessárias entre as diferentes operações. O paradigma de programação baseado em tarefas é um exemplo que possui esses atributos. Ele usa um *runtime* dinâmico para escalonar tarefas, e as aplicações são bem definidas em um DAG. Embora esse paradigma apresente muitos benefícios, ainda há desafios ao distribuir essas aplicações nesses recursos heterogêneos ao nível do sistema.

Diferentes problemas aparecem ao lidar com a distribuição de aplicações baseadas em tarefas em recursos heterogêneos ao nível do sistema. Ao considerar apenas uma operação da aplicação, um problema é distribuí-la corretamente em uma gama diversificada de nós de computação. Embora essa distribuição deva considerar cada recurso, outros aspectos comportamentais, como o caminho crítico e as comunicações, também são importantes. No entanto, uma única operação é apenas uma das muitas partes das aplicações. Pode haver diversas operações com comportamentos diferentes e afinidades de recursos que, idealmente, exigem distribuições distintas. Além disso, em aplicações baseadas em tarefas, essas operações podem ser executadas de forma assíncrona, sobrepostas, e cada uma pode escolher seu melhor recurso relativo. Nesse contexto, o problema de distribuir a aplicação em diferentes recursos deve agora considerar as várias operações sobrepostas que adotam distribuições distintas. O número de recursos disponíveis também pode ser excessivo para uma determinada fase, pois, às vezes, é vantajoso usar menos recursos para reduzir alguns problemas. No entanto, a modelagem de

todos os comportamentos que levam a esses problemas antes da execução da aplicação pode ser um desafio, e a adaptação dinâmica durante a execução é uma possível solução. Em todos esses problemas, há um desafio transversal que é como analisar o desempenho das aplicações. Em última análise, esta tese contribui com estratégias para todos esses problemas correlacionados.

Esta tese escolheu o *runtime* baseado em tarefas StarPU e as aplicações de seu ecossistema como objetos de estudo. O *runtime* StarPU tem a flexibilidade necessária para definir distribuições, permite redistribuição assíncrona, tem um recurso de simulação que pode aumentar os cenários experimentais quando necessário e muitas ferramentas auxiliares para análise de desempenho. Nesse sentido, os experimentos desta tese usaram a biblioteca de álgebra linear Chameleon, a aplicação de aprendizagem de máquina ExaGeoStat e a biblioteca para analisar grandes conjuntos de dados Diodon. As plataformas usadas para os experimentos foram a infraestrutura Grid5000 e o supercomputador SDumont, ambos com diferentes níveis de heterogeneidade. Essas aplicações selecionadas apresentam várias operações que, com as contribuições desta tese, podem explorar a heterogeneidade ao nível de sistema para melhorar o desempenho. A seguir, as principais linhas de pesquisa e contribuições são apresentadas.

O primeiro conjunto de contribuições, no Capítulo 5, concentra-se em uma única distribuição em uma operação das aplicações. Ela parte de algoritmos da literatura, mais especificamente, o 1D-1D. Uma etapa inicial foi o estudo do comportamento dessas distribuições heterogêneas em comparação com a distribuição BC clássica, mesmo em configurações homogêneas, mostrando que ela pode lidar com um número arbitrário de nós (incluindo números primos). Em seguida, esta tese propõe duas estratégias, inspiradas no 1D-1D, que criam distribuições heterogêneas. A primeira considera o caminho crítico e a comunicação juntamente com as capacidades heterogêneas dos recursos. Ela restringe a carga de trabalho final da operação a recursos mais rápidos, diminuindo a comunicação e melhorando o caminho crítico. A segunda estratégia realiza um balanceamento extra, relaxando as restrições anteriores de que alguns nós só se comunicariam com outros. Uma análise de desempenho compara o 1D-1D com a metodologia que usa as duas estratégias. Os resultados indicam um pequeno ganho ao combinar essas duas estratégias nos casos em que o 1D-1D já apresentava um desempenho bom. Uma análise anterior que previa um pequeno espaço para melhoria (limite inferior) colabora com esses resultados. No entanto, problemas futuros reutilizarão a estratégia de restrição em seus casos em que o caminho crítico em distribuições multifásicas é mais importante, apresentando melhores resultados.

O segundo grupo de contribuições se concentra no problema de aplicações com várias fases (operações), apresentado no Capítulo 6. O capítulo mostra que garantir a execução

assíncrona entre as operações e adaptar as distribuições considerando sua sobreposição pode melhorar o desempenho, mesmo em cenários homogêneos. Uma série de estratégias aprimora a sobreposição assíncrona das fases, melhorando em até 49% o *makespan* ao considerar as aplicações ExaGeoStat e Diodon. Em seguida, ele estuda o ambiente heterogêneo. Um Programa Linear (LP) calcula a divisão ideal de tarefas por máquina, considerando a heterogeneidade e a sobreposição de fases. Esse LP também serve como um limite inferior para o desempenho da aplicação. A potência relativa de cada nó para cada fase é extraída do resultado da divisão de tarefas do LP. Essa potência por fase e máquina será a entrada para os algoritmos de distribuição (do Capítulo 5, incluindo a estratégia de restrição) para calcular a distribuição das operações selecionadas. Por fim, esta tese propõe um algoritmo para calcular uma distribuição para uma fase precedente e, ao mesmo tempo, minimizar as comunicações de redistribuição. Os algoritmos usam a distribuição seguinte como referência e a divisão de tarefas do LP. Essa metodologia melhora o desempenho no melhor caso estudado em 69% no ExaGeoStat e em 73% no Diodon quando comparado ao uso de uma estratégia de distribuição homogênea no cluster homogêneo mais potente (partição) de cada cenário. O capítulo também apresenta uma análise do desempenho de alguns casos que sugere que o uso de todos os nós disponíveis para todas as fases pode ser desnecessário.

O terceiro conjunto de contribuições (Capítulo 7) segue a conclusão dada pelo último capítulo e se concentra em limitar o número de recursos em cada fase. O problema é que a contenção da rede, o caminho crítico ou outro comportamento inesperado podem deteriorar o desempenho mais do que a possível contribuição ao adicionar um nó de computação. No entanto, a modelagem desses comportamentos é um desafio nesse cenário assíncrono e dinâmico. Por esse motivo, esta tese estuda o uso de métodos de aprendizagem por reforço durante o tempo de execução para modelar o comportamento da aplicação ao selecionar um número arbitrário de nós para uma determinada fase. Esse modelo serve como um substituto que a aplicação pode consultar para orientar a próxima decisão. Esta tese propõe um método baseado no Processo Gaussiano (GP) com sua metodologia de Intervalo de Confiança Superior (UCB) que pressupõe um comportamento suave no espaço de busca. A proposta acrescenta conhecimento de HPC ao método para ajustá-lo a esse problema, considerando a limitação do espaço de busca, fornecendo uma tendência esperada e lidando com descontinuidades no comportamento do *makespan*. Essa parte do aprendizado atua em um segmento da aplicação, nesse caso, uma longa iteração. A aplicação ExaGeoStat apresenta essa estrutura, em que executa muitas iterações de otimização (com uma sincronização algorítmica inevitável), e cada uma delas compreende muitas operações assíncronas. O modelo aproximará a duração de uma iteração. Antes de iniciar cada uma

delas, a aplicação consulta o modelo e, com base no componente UCB, escolhe uma ação que compreende a exploração e o aproveitamento. No final, esse método é comparado com outros seis em 16 cenários, mostrando que ele foi o único que lidou com todos os casos, melhorando o desempenho em até 51,2% ao selecionar o número de nós para a fase de fatoração do ExaGeoStat. Ele também mostra que a sobrecarga do método é baixa e que pode haver um ganho limitado (em desempenho) na otimização do número de nós considerando todas as fases.

O último grupo de contribuições vem de um problema transversal a todos os outros: analisar o desempenho dessas aplicações baseados em tarefas. Durante o andamento desta tese, qualquer investigação incluía uma análise extensa e abrangente dos rastros de execução. Muitas dessas investigações levaram a aprimoramentos e novos recursos no StarVZ. Um exemplo foi a adição do gráfico de Gantt com agregação por nó e por recurso. No entanto, nos estágios finais deste trabalho, a escalabilidade da visualização da análise em alguns experimentos foi considerada um problema. Em vez de depender diretamente do gráfico de Gantt, que nunca será dimensionado à medida que o número de recursos aumenta, seria desejável outra visualização simples para apontar grupos de nós problemáticos. É por isso que o capítulo 8 estuda uma metodologia para servir como uma visualização de resumo do comportamento do desempenho de diferentes nós. Ela se baseia em uma métrica de progressão sensível à heterogeneidade ao nível de sistema e a tarefas distintas (de várias operações). A metodologia usa essa métrica para cada nó em diferentes etapas de tempo. Em seguida, como alguns nós computacionais têm comportamento semelhante, ela agrupa a métrica em grupos de nós parecidos. A visualização final mostra apenas estes grupos de nós, onde os grupos que avançam lentamente na métrica de progressão são potencialmente problemáticos. Essa visualização não precisa de mais espaço à medida que o número de nós aumenta. Para demonstrar a utilidade da metodologia, o capítulo usa alguns cenários problemáticos criados com simulação e usa execuções reais de outras investigações. Em última análise, o método proposto funcionou bem em todos os casos testados e ajudou a detectar todos os nós problemáticos.

Todas essas contribuições são motivadas pela distribuição de aplicações baseados em tarefas em recursos heterogêneos. E, no final, elas devem ser usadas em conjunto. Quando uma aplicação começa a ser executada, ela precisa decidir o número de nós a serem usados por fase, acionando as contribuições do Capítulo 7. Essa contribuição começará usando todos os nós em todas as operações, exigindo o cálculo das distribuições cientes da heterogeneidade e das operações assíncronas sobrepostas, levando ao uso das estratégias do Capítulo 6. Em seguida, essas estratégias usam inerentemente o Capítulo 5 para calcular a distribuição final das fases selecionadas. Depois disso, o desempenho da aplicação pode ser analisado usando as

contribuições do Capítulo 8 e a metodologia geral empregada nesta tese.

Esse trabalho abre novas perspectivas em vários tópicos. O primeiro inclui ajustes na distribuição de dados heterogêneos durante a execução, uma ação adicional às distribuições estáticas, e a seleção dinâmica do número de nós. Isso pode consistir em refinar as distribuições em operações específicas, considerando a complexidade de várias fases. Essa situação poderia melhorar ainda mais o equilíbrio com um comportamento imprevisível. O ideal é que esses aprimoramentos sejam adaptados à aplicação, explorando seu algoritmo e a estrutura do DAG. Entretanto, uma maneira sistemática de alterar essa distribuição no contexto do STF e da computação distribuída apresentaria problemas técnicos e científicos. Depois que o DAG é desenrolado no modelo STF, a propriedade dos dados e as alterações de tarefas exigiriam conhecimento global para garantir a consistência distribuída. Nesse contexto, as decisões de alterações de distribuição devem ser iguais. No modelo atual, os nós computacionais podem até mesmo submeter tarefas diferentes, contando com o design da aplicação para submeter todas as tarefas localmente necessárias em todos os nós, levando ao correto DAG distribuído.

Outra questão em aberto é como melhorar o atual modelo de poder por nó multifásico com comunicação, o LP no Capítulo 6. Essas informações adicionais poderiam aproximá-lo ainda mais da realidade. É difícil prever essas informações devido ao escalonamento dinâmico e à complexidade geral da sobreposição de operações. No mesmo contexto desse modelo, um problema atual é que as informações sobre uma aplicação são específicas a ela. Embora algumas aplicações compartilhem as mesmas bibliotecas subjacentes, como o Chameleon, somente algumas tarefas reutilizadas que usam os mesmos tamanhos podem ser ter os modelos de desempenho baseados em histórico compartilhados.

No contexto de encontrar o melhor conjunto de nós a ser usado, calcular a eficiência das configurações em vez de apenas o *makespan* parece ser a direção correta. Preparar melhor as aplicações para se adaptarem de uma configuração superdimensionada poderia melhorar o *makespan*, a energia e a eficiência. Especificamente, para as estratégias propostas, uma situação é considerar a expansão do modelo do GP ao aumentar o espaço de pesquisa com todas as fases. O *bootstrap* do modelo e como diminuir o espaço de busca são questões em aberto. A flexibilidade do GP como substituto para modelar o desempenho e a eficiência pode ser usada para explorar outros parâmetros, permitindo que as aplicações HPC se adaptem aos seus sistemas em cada execução particular. Um exemplo, é a possibilidade que a abordagem do GP possa modelar distribuições que permitam blocos de precisão mista ou outras formas de compactação específicas para os dados de entrada. Combinando essas estratégias, seria possível determinar as regiões e o número de blocos com menor precisão e, ao mesmo tempo, modelar a compensação

entre desempenho e precisão.

Para a análise de desempenho e as visualizações, algumas perspectivas incluem sempre facilitar o desenvolvimento de tais aplicações, permitindo que os desenvolvedores entendam rapidamente como sua implementação atual está se comportando. Especificamente para a estratégia proposta de resumir o comportamento utilizando agrupamento. Um trabalho futuro inclui a investigação de outras métricas de progressão e técnicas de agrupamento. Elas poderiam ser adaptadas às aplicações e à situação.

Em última análise, muitas técnicas, métodos e estratégias tradicionais para casos homogéneos devem ser revisitadas para descobrir as múltiplas oportunidades e benefícios que surgem neste contexto heterogéneo.

APPENDIX B — RÉSUMÉ DÉTAILLÉ EN FRANÇAIS

Le calcul intensif permet aux applications complexes d'utiliser une puissance de calcul massive et de conclure dans un délai raisonnable. Jusqu'au début de la décennie 2000, la croissance du nombre de transistors par puce suivait la loi de Moore, qui stipulait qu'elle doublerait tous les deux ans (MOORE et al., 1965; MOORE et al., 1975). Cette croissance a permis une augmentation exponentielle de la puissance de calcul, comme le souligne Dennard (DENNARD et al., 1974). Cependant, les limites de la dissipation thermique ont freiné l'augmentation de la fréquence d'horloge des puces dans les années 2000 (HENNESSY; PATTERSON, 2017), stoppant l'augmentation exponentielle des performances des cœurs individuels (DONGARRA et al., 2017). De nombreuses alternatives pour poursuivre la croissance de la puissance de calcul sont apparues, notamment diverses techniques de CPU et des accélérateurs.

La combinaison d'un sous-ensemble de ces ressources aboutit à un seul nœud de calcul hétérogène. Cette hétérogénéité intra-nœud a été largement utilisée ces dernières années dans de nombreuses applications par la communauté HPC (SANDERS; KANDROT, 2010; AUGONNET et al., 2011; MENG et al., 2017; PINTO, 2018). Cependant, l'exploitation de cette hétérogénéité est complexe, car elle nécessite une division correcte de l'application dans ces ressources multiples et variées, en tenant compte de l'adaptabilité et de la performance de l'algorithme dans chaque ressource (SANDERS; KANDROT, 2010; AUGONNET et al., 2011).

Du point de vue des développeurs, la gestion d'une telle hétérogénéité nécessite une connaissance approfondie du système cible et l'anticipation du comportement de l'application. Dans les méthodes et les outils de programmation traditionnels tels que les applications MPI (Message Passing Interface) et BSP (Bulk Synchronous Parallel), l'utilisation de ressources hétérogènes est compliquée, car le style de programmation est entièrement impératif. Ces approches de programmation spécifient l'endroit où une charge de travail s'exécute statiquement avec des barrières et des communications synchrones pour contrôler le flux des multiples opérations ou phases de l'application. Avec ce niveau de synchronisme, une petite erreur dans la répartition de la charge de travail se traduirait par des performances non idéales avec des ressources inactives. Mais une telle erreur est presque inévitable, car les systèmes et les applications sont complexes. De plus, garantir la capacité (portabilité) de l'application à s'exécuter sur différents systèmes nécessite d'importantes modifications de son code source, ce qui augmente le risque d'erreurs.

Cependant, la disponibilité de processeurs distincts et la diversité des besoins des applications encouragent l'hétérogénéité au niveau du système (HPE et al., 2022). Il est donc de

plus en plus courant que les systèmes HPC comprennent de nombreuses machines avec diverses configurations matérielles organisées en grappes ou partitions homogènes pour mieux répondre aux besoins des applications. Si elle est utilisée correctement, cette hétérogénéité permet à l'application d'ajuster un comportement interne particulier et d'améliorer les performances, en exploitant l'hétérogénéité au lieu de la subir.

Les principales infrastructures de calcul intensif, les superordinateurs, sont généralement situés dans de grands centres de recherche universitaires et industriels. Les sites (les infrastructures) peuvent héberger différents systèmes, comme le montre la liste TOP500 (DONGARRA et al., 1997) des superordinateurs, offrant ainsi l'hétérogénéité déjà mentionnée. L'hétérogénéité de ces sites de superordinateurs s'explique par trois raisons : (i) la conception, lorsque l'infrastructure possède de telles configurations pour cibler diverses charges de travail, (ii) les limitations financières, lorsque, par exemple, seul un sous-ensemble de nœuds reçoit des accélérateurs en raison de contraintes budgétaires, et (iii) les mises à niveau naturelles de l'infrastructure au fil du temps. La liste TOP 500 ne prend en compte et ne classe qu'un ensemble homogène de nœuds en tant que système. Cette restriction provient d'une limitation du benchmark HPC adopté, HPL (DONGARRA; LUSZCZEK; PETITET, 2003), qui ne peut pas fonctionner de manière adéquate dans des configurations hétérogènes au niveau du système.

De nombreux défis algorithmiques apparaissent lors de la transition vers des scénarios hétérogènes (DONGARRA; LASTOVETSKY, 2006; BEAUMONT et al., 2019). Tout d'abord, la quantité idéale de données par nœud sera différente. En fonction de la structure des données de l'application, cette division peut ne pas être triviale. Bien qu'une simple division soit suffisante dans les structures de données unidimensionnelles, quand toutes les portions de données ait un coût de calcul identique, des stratégies spécialisées sont nécessaires pour les dimensions supérieures ou lorsque des données particulières ont un comportement différent ou des besoins en puissance de calcul (BEAUMONT et al., 2001a). En outre, le fait d'avoir des quantités différentes de données par nœud entraîne différentes quantités de communication entre les nœuds, car les nœuds qui traitent plus de données devront communiquer davantage. Cette différence de communication signifie que le compromis idéal entre la communication et le calcul par nœud est plus critique dans les scénarios hétérogènes que dans les scénarios homogènes. Il en va de même si la quantité de calcul par élément de données diffère, ce qui entraîne un chemin critique que les capacités hétérogènes des nœuds influenceront. Enfin, il existe une perspective technique de programmation. Les distributions plus simples, comme les distributions cycliques, sont plus faciles à mettre en œuvre dans les paradigmes traditionnels tels que MPI, et les fonctions de communication sont plus clairement appliquées. Cependant, des

schémas de communication irréguliers apparaissent lorsqu'il s'agit de distributions arbitraires, et d'autres paradigmes ou intergiciels peuvent être nécessaires pour réduire la complexité du développement et la maintenabilité de l'application.

En outre, les applications peuvent comporter différentes phases ayant des besoins de calcul différents et admettant des distributions de données idéales distinctes. Ces phases peuvent également exploiter différemment les ressources des nœuds, ce qui modifie encore davantage la distribution idéale pour chacune d'entre elles. Par exemple, les phases comprenant la génération de données sont généralement plus adaptées aux unités centrales traditionnelles, tandis que certaines opérations de calcul, telles que les d'algèbre linéaire classiques, pourraient utiliser des accélérateurs pour augmenter les performances.

Les limites du paradigme traditionnel comprennent une mauvaise gestion de l'hétérogénéité intra-nœud, une faible efficacité de programmation, un synchronisme inutile et une portabilité limitée des ressources. Ensemble, ils évoquent la résurgence du paradigme de programmation basé sur les tâches (BOSILCA et al., 2013; DURAN et al., 2011; AUGONNET et al., 2011; WU et al., 2015; THIBAULT, 2018). Ce paradigme adopte un mode de programmation plus déclaratif et utilise un moteur d'exécution pour prendre des décisions, y compris la programmation dynamique du travail (tâches) pendant l'exécution. L'application décrit les tâches individuelles et les dépendances de données et les structure dans un graphe acyclique direct (DAG). Les avantages de cette approche sont notamment de soulager la complexité de programmation liée à la gestion explicite des communications irrégulières et du flux de calcul à l'aide d'un moteur d'exécution et d'améliorer la coopération des ressources hétérogènes à l'intérieur des nœuds.

Le moteur d'exécution est chargé d'ordonnancer les tâches en respectant les dépendances à l'aide de nombreuses heuristiques possibles. Cette approche permet également de faciliter l'asynchronisme des tâches d'opérations simples et multiples. En outre, certains moteurs d'exécution effectuent automatiquement les transferts de données entre les nœuds et à l'intérieur de ceux-ci en fonction de la structure du DAG, ce qui réduit la charge de développement. Le développeur de l'application doit encore donner de nombreuses indications au moteur d'exécution pour l'aider pendant l'exécution, mais une fois formulé, le code est hautement portable sur plusieurs systèmes. Exemples de ces moteurs d'exécution sont ParSEC (BOSILCA et al., 2013), OmpSS (DURAN et al., 2011) et StarPU (AUGONNET et al., 2011), ce dernier étant utilisé dans le cadre de ce travail. Ces moteurs d'exécution modernes basés sur les tâches fournissent une abstraction de programmation de haut niveau avec la flexibilité requise, facilitant la recherche et le développement de stratégies sophistiquées de distribution de données statiques

sur des nœuds hétérogènes. Cette approche semble plus élégante et plus appropriée pour gérer et combiner l'hétérogénéité de l'architecture des systèmes et les besoins des applications.

Par conséquent, le paradigme de programmation basé sur les tâches offre de nombreuses possibilités de mettre en œuvre des stratégies algorithmiques élaborées et des distributions raffinées nécessaires pour relever les nombreux défis qui émergent de l'utilisation de ressources hétérogènes. En outre, ce travail utilise des informations liées au DAG et basées sur les tâches pour guider les décisions relatives aux stratégies et l'analyse des performances.

B.1 Apports de la thèse

Les applications HPC nécessitent une puissance de calcul considérable fournie par les superordinateurs. Ces ressources de calcul peuvent présenter une hétérogénéité au niveau du système lorsqu'il y a deux groupes de nœuds ou plus, chaque groupe ayant un matériel et une puissance de calcul différents. En outre, les applications peuvent présenter un comportement hétérogène interne en raison d'opérations distinctes ou de phases multiples qui peuvent s'exécuter différemment à chaque ressource intra-nœud. L'augmentation de l'hétérogénéité intra-nœud dans les superordinateurs et la difficulté de les programmer encouragent l'utilisation de paradigmes de programmation parallèle robustes tels que celui basé sur les tâches. Bien que ce paradigme offre suffisamment de flexibilité, de portabilité et de dynamisme pour gérer un scénario aussi complexe, de nombreux problèmes doivent encore être résolus. Toutes ces circonstances rendent le problème de la **distribution de ces applications basées sur des tâches entre des nœuds hétérogènes** difficile, bien que de nombreuses opportunités d'amélioration des performances et de l'utilisation des ressources se présentent.

L'objectif principal de ce travail est de fournir des stratégies et des méthodes pour améliorer la distribution des applications basées sur des tâches sur des ressources hétérogènes au niveau du système. Dans ce contexte, de nombreux défis doivent être pris en compte pour qu'une application atteigne une performance correcte. Les défis abordés dans cette thèse sont les suivants : (1) Les distributions pour chaque phase de l'application doivent prendre en compte non seulement l'équilibrage de la charge, mais aussi le compromis entre la communication et le chemin critique ; (2) Créer plusieurs distributions pour des applications multi-phases avec des besoins différents tout en considérant leur interaction chevauchante ; et (3) trouver le nombre idéal de nœuds par type pour chaque phase. La structure principale de cette thèse est la suivante.
Chapitres préliminaires. Le Chapitre 2 présente le contexte des applications basées sur les tâches, les caractéristiques des runtimes, et l'écosystème StarPU. Il se concentre sur la structure

du DAG des applications à partir des algorithmes et sur la façon dont le runtime planifie les tâches dans de nombreuses ressources. Ce chapitre présente également les applications utilisées dans le reste de la thèse. La première est la bibliothèque d'algèbre linéaire Chameleon. La deuxième est l'application de géostatistiques ExaGeoStat. Et la troisième est la bibliothèque d'analyse de grands ensembles de données Diodon.

Le Chapitre 3 présente de nombreux travaux liés à l'état de l'art. Il commence par le problème général de la distribution d'une application (principalement les applications structurées d'algèbre linéaire 2D) sur des scénarios homogènes et hétérogènes. Ensuite, il introduit les travaux et techniques classiques sur l'équilibrage de la charge, suivis d'une présentation des travaux relatifs à la gestion des applications multi-phases en tenant compte des distributions et des communications multiples. Enfin, nous présentons brièvement comment certains algorithmes d'apprentissage automatique et d'apprentissage par renforcement ont été utilisés dans des problèmes HPC connexes.

Méthodologie. Le Chapitre 4 présente la méthodologie de la thèse pour les expériences contrôlées et l'analyse complète des performances. La méthodologie s'appuie sur des exécutions réelles avec un contrôle expérimental, des simulations avec une évaluation de leur fiabilité dans notre contexte, et l'analyse de ces expériences en utilisant des métriques analytiques, l'analyse des traces et la visualisation.

Contributions. Le Chapitre 5 étudie les distributions pour une opération possible, la factorisation LU, qui pourrait être étendue à des opérations d'algèbre linéaire similaires. Il apporte les contributions suivantes. **(a)** Une stratégie qui améliore les performances des applications en réduisant les communications sur le chemin critique. Cette situation se produit lorsque le parallélisme dans le DAG diminue. L'approche contraint la distribution à utiliser moins de ressources vers la fin de l'algorithme. **(b)** Une stratégie pour améliorer l'équilibrage de la charge de calcul d'une distribution statique donnée considérant des tâches multiples et des ressources hétérogènes en augmentant la communication. **(c)** Une méthodologie pour combiner (a) et (b).

Le Chapitre 6 examine le problème des applications multi-phases impliquant éventuellement plusieurs distributions hétérogènes. L'étude de cas repose sur les applications multi-phases ExaGeoStat et Diodon. Les contributions du chapitre sont les suivantes. **(d)** Optimisations pour améliorer l'asynchronisme de phase dans les applications et améliorer les distributions. **(e)** Stratégie pour générer des distributions hétérogènes efficaces pour les applications multi-phases avec différentes affinités de performance des ressources tout en considérant le chevauchement des phases. **(f)** Une technique pour dériver d'autres distributions à partir d'une distribution principale qui réduit les communications lors de la redistribution.

Le Chapitre 7 présente des stratégies permettant à l'application d'apprendre activement et de s'adapter aux meilleurs nœuds hétérogènes auxquels elle peut accéder. Le chapitre contient les contributions suivantes. **(g)** Une analyse des principales caractéristiques de ce problème (structure et bruit) et explique pourquoi les techniques génériques d'optimisation et d'apprentissage échoueront probablement. Cette analyse motive la conception de variantes spécifiques d'une technique d'apprentissage par renforcement basée sur le processus gaussien. **(h)** Une évaluation complète des performances avec 16 machines hétérogènes différentes et des charges de travail qui comparent les solutions proposées avec d'autres méthodes d'optimisation génériques (Brent, Bandits, GP-UCB). Parmi ces différentes méthodes, la variante basée sur GP est la seule méthode robuste et parcimonieuse permettant d'atteindre rapidement la configuration optimale dans de multiples scénarios. **(i)** Une implémentation réelle de la méthode pour permettre à l'application de s'adapter pendant l'exécution, démontrant le faible surcoût.

Le Chapitre 8 traite des techniques et des méthodes permettant d'analyser le comportement des applications basées sur les tâches. En particulier, les techniques **(j)** qui mettent l'accent sur l'hétérogénéité de la plate-forme et de l'application et sur la progression réelle de l'application.

Le Chapitre 9 conclut cette thèse avec les contributions majeures, les prochaines directions, et la liste des publications.

Enfin, un site compagnon pour la thèse est disponible publiquement à l'adresse suivante : <https://gitlab.com/lnesi/thesis-companion.git>. Il comprend les données, les scripts et les traces des expériences pour reproduire l'analyse et les figures.

B.2 Paradigme de Programmation Basé sur les Tâches

Le paradigme de programmation basé sur les tâches (THIBAULT, 2018), également connu sous le nom de Data Flow Scheduling (DONGARRA et al., 2017) ou le Asynchronous Many Task (AMT) paradigme et les runtimes (HUMPHREY; BERZINS, 2019), utilise une approche plus descriptive, et non impérative, pour définir une application. Les applications expriment leurs algorithmes internes avec des tâches et des dépendances sans définir explicitement où se trouve le parallélisme et où et quand ces tâches s'exécutent. Un moteur d'exécution décide de l'ordonnancement et du placement des tâches pendant l'exécution à l'aide d'algorithmes internes et d'heuristiques. En raison de cette flexibilité et du couplage lâche sur la plateforme, Dongarra et al. (2017) souligne que la programmation basée sur les tâches sera le paradigme souhaité pour les systèmes exaflops. Bien que l'utilisation de tâches et l'ordonnancement dynamique

soient des concepts anciens (CODD, 1960), ils gagnent en popularité dans de nombreux projets nouveaux et modernes (DONGARRA et al., 2017; THIBAULT, 2018; HOUSSAM-EDDINE et al., 2020).

Dans le cas de StarPU, le runtime utilisé dans ce travail, l'utilisation successive de données à travers de multiples tâches créera un flux qui exprime la progression de l'application et construit le DAG de tâches, la stratégie du Sequential Task Flow (STF) (AGULLO et al., 2016). Dans cette approche, l'application n'a qu'un seul thread et soumet les tâches de manière séquentielle, construisant ainsi le DAG.

Bien que les moteurs d'exécution puissent techniquement prendre en charge l'exécution sur des ressources hétérogènes au niveau du système, leurs performances dépendent de la manière dont l'application est structurée (DAG, par exemple) et de la manière dont le développeur la répartit sur les nœuds. Dans les deux cas, la connaissance de la façon dont une application avec des opérations asynchrones s'exécute sur des ressources hétérogènes au niveau du système est limitée et a un grand potentiel à explorer. Comme le montrera cette thèse, de nombreux défis apparaissent qui nécessitent des solutions basées sur l'application. Ainsi, les approches ou les stratégies qui aident les applications basées sur des tâches sur des ressources hétérogènes au niveau du système sont souhaitables.

B.3 Travail connexe : Répartition de la charge

La répartition des données et des calculs pour les nœuds distribués est un élément fondamental de la programmation parallèle. La répartition de la charge peut optimiser plusieurs objectifs. Les objectifs les plus courants sont l'amélioration des performances de l'application en réduisant les temps morts ou en équilibrant la charge des processeurs, la réduction de la communication totale pour éviter la contention du réseau, ou l'augmentation des calculs parallèles disponibles.

Les stratégies de répartition des données et des calculs entre les ressources peuvent être statiques, dynamiques ou hybrides. La partition statique est effectuée une fois avant tous les calculs et reste stationnaire pendant l'exécution de l'application. Ce problème de partitionnement dépend du domaine d'application. Par exemple, les opérations linéaires d'algèbre classique doivent répartir une matrice 2D entre les ressources. Cependant, la solution optimale pour la stratégie 2D statique est un problème NP-complet (BEAUMONT et al., 2002a). Les stratégies dynamiques reposent sur la répartition de la charge de travail pendant l'exécution de l'application. La plupart des moteurs d'exécution basés sur les tâches, y compris StarPU (AUGONNET et al.,

2011), adoptent cette approche pour l'ordonnancement des ressources à l'intérieur des nœuds. Enfin, une approche hybride combine des techniques statiques et dynamiques. Un exemple d'approche hybride est le module StarPU-MPI, qui utilise un ordonnancement dynamique pour les tâches intra-nœud et une distribution statique des données entre les différents nœuds, principalement pour des raisons d'extensibilité. L'application doit informer cette distribution statique, tandis que l'un des nombreux algorithmes heuristiques de StarPU effectue l'ordonnancement dynamique des tâches à l'intérieur des nœuds.

Des partitions permettant de traiter efficacement une seule opération de mise à jour sont ainsi disponibles. Cependant, un brassage correct des colonnes et des lignes est nécessaire pour obtenir un bon équilibre de la charge tout au long de l'exécution d'algorithmes plus complexes comme l'algorithme LU. Beaumont et al. (2001a) a proposé une procédure de brassage simple (1D-1D), qui est asymptotiquement optimale, quelle que soit la partition initiale du rectangle.

Les problèmes de (i) distribution d'une opération sur des ressources hétérogènes ; (ii) distributions et redistributions multiples pour des applications multi-phases ; et (iii) apprentissage et adaptation à des comportements inattendus comme l'extensibilité des nœuds ; sont principalement étudiés avec des nœuds homogènes et séparément dans la littérature, manquant ainsi l'opportunité de les combiner tous. Ce travail montre que ces problèmes constituent des défis lors de la distribution d'applications basées sur des tâches sur des ressources hétérogènes au niveau du système. Par conséquent, il faut résoudre tous ces problèmes pour améliorer la performance de l'application. Cette thèse présente ses contributions à la distribution de charge d'un point de vue micro à macro. Nous commençons par calculer la distribution d'une opération (Chapitre 5) qui sera ensuite utilisée pour étudier les multiples distributions dans les multiples opérations (Chapitre 6). Ensuite, de telles stratégies seront à nouveau employées lors du réglage du nombre de nœuds à utiliser par phase (Chapitre 7).

B.4 Méthodes d'analyse et d'expérimentation

Les méthodes expérimentales et d'analyse de ce travail comportent trois étapes. La première étape (méthodologie expérimentale) consiste à mener des expériences dans des environnements contrôlés, en utilisant des exécutions réelles et des simulations. L'exécution génère des traces qui décrivent en détail chaque expérience réalisée. L'étape suivante (méthodologie d'analyse des performances) est l'analyse de ces traces à l'aide de métriques, de visualisations et d'outils de science des données. Cette analyse conduit à l'étape finale, la proposition de solutions et de méthodes pour améliorer le comportement. Un redémarrage de la méthodologie

a lieu avec de nouveaux candidats à l'optimisation. Les stratégies proposées dans cette thèse reflètent une boucle itérative d'expérimentation, d'analyse et de proposition de solutions.

Les expériences de ce travail sont divisées en deux catégories : les exécutions réelles et les simulations. Dans la première, une plate-forme réelle exécute directement l'application. Le programme calcule effectivement et atteint son objectif, en parvenant à une solution algorithmique réelle et correcte. À l'inverse, la simulation peut remplacer certains calculs par des modèles statistiques, en se concentrant sur l'approximation du comportement des performances et non sur la solution finale de l'algorithme. Ce travail utilise Simgrid (CASANOVA et al., 2014) comme cadre pour simuler des plateformes HPC distribuées, combine avec StarPU avec son module StarPU+Simgrid (STANISIC et al., 2015b). StarPU+Simgrid génère des traces qui se rapprochent fidèlement du comportement de l'application et de l'utilisation des ressources (STANISIC et al., 2015a). Cependant, StarPU+Simgrid n'effectue pas de calculs (tâches) et peut remplacer les données par des informations factices.

La base de la méthodologie d'analyse est constituée par les traces d'exécution et les données complémentaires que le runtime et l'application collectent. La thèse utilise différentes stratégies pour la collecte de données, le calcul des métriques de performance, la visualisation du comportement de la performance de l'application et l'investigation directe des données. L'un des principaux outils utilisés est StarVZ, un cadre d'analyse des performances pour les applications basées sur des tâches, construit sur R et Tidyverse.

Grâce à la méthodologie d'analyse, il est possible de vérifier certains problèmes tels que la sous-utilisation des ressources à un moment donné, le déséquilibre des ressources, le manque de tâches prêtes, les points de synchronisation dans les itérations de l'application, les valeurs inadéquates pour toute métrique présentée et les opérations liées à la mémoire. Toutes ces analyses permettent à l'analyste de proposer des solutions possibles dans certaines entités d'exécution (application, runtime, bibliothèque, système). Ces solutions nécessitent une expérimentation, une validation de l'analyse et la répétition de la méthodologie. Ce processus a été appliqué à tous les problèmes envisagés dans la thèse, les résultats conduisant à l'objet d'étude suivant, ce qui donne lieu à un processus itératif.

B.5 Stratégies de distributions hétérogènes pour l'algèbre linéaire

Le Chapitre 5 se concentre sur une seule opération et propose des stratégies pour générer des distributions cycliques statiques pour les procédures d'algèbre linéaire qui doivent équilibrer la charge sur plusieurs itérations. Bien que les algorithmes de distribution cyclique et hétérogène

de pointe comme 1D-1D fournissent des résultats asymptotiquement optimaux, ils laissent encore des possibilités d'amélioration et de création de distributions plus raffinées. En outre, une charge d'équilibre parfaite ne garantit pas les meilleures performances. En raison du chemin critique et de l'hétérogénéité des nœuds, il est parfois préférable d'avoir un certain déséquilibre avec une charge supplémentaire sur les nœuds les plus rapides. Les stratégies discutées dans ce chapitre agissent sur une seule opération, et leurs idées sont essentielles lors de l'examen de l'interaction à phases multiples avec des multiples distributions dans les Chapitres 6 et 7.

Ce chapitre propose des algorithmes (1D-1D C et 1D-1D C+S) pour générer des distributions en tenant compte des moments de l'application avec un faible parallélisme et des compromis de communication et d'équilibre de charge. Des expériences avec la factorisation LU démontrent que les distributions 1D-1D peuvent être bénéfiques même pour un groupe homogène de nœuds hybrides lorsqu'elles sont comparées à BC, présentant une évolutivité beaucoup plus stable. Le chapitre propose deux nouvelles stratégies pour prendre en compte un ensemble hétérogène de machines hybrides, pour lesquelles 1D-1D est également presque optimal. La première utilise moins de nœuds vers la fin de l'opération, en intensifiant progressivement le calcul dans les machines puissantes. La seconde applique un brassage supplémentaire des blocs pour niveler le travail cumulé. La technique de contrainte permet d'optimiser la fin de l'exécution, tandis que le mélange des blocs permet d'atteindre un équilibre de charge quasi optimal entre les itérations, mais augmente la communication. Une combinaison pertinente de ces techniques permet d'améliorer sélectivement les performances par rapport au modèle 1D-1D original, à la fois en termes d'équilibrage de la charge et de temps d'exécution.

Cette étude indique également que les distributions 1D-1D sont un excellent point de départ, qui n'est pas si facile à améliorer. Dans la pratique, lorsqu'on ne considère qu'une opération d'algèbre linéaire, il n'est pas certain qu'une utilisation systématique de 1D-1D C et 1D-1D C+S soit bénéfique, car elle nécessite un bon modèle de performance et peut induire un surcoût de communication. Cependant, ces deux stratégies sont des outils qui peuvent être appliqués à des situations spécifiques. Lorsque des problèmes surviennent à la fin de l'exécution en raison d'un parallélisme limité, d'une communication excessive ou d'un chemin critique mal placé, la stratégie de contrainte peut s'avérer utile. Un remaniement plus poussé peut s'avérer utile dans les cas où l'équilibrage fourni par le 1D-1D pourrait être amélioré et où la communication n'est pas un problème (dans un réseau à grande vitesse et à faible latence, par exemple). Cette première étude permet d'identifier les quantités essentielles au rééquilibrage et les conséquences d'une distribution irrégulière sur l'exécution globale. De plus, cette étude montre que des distributions équilibrées non parfaites (comme la version contrainte) peuvent

présenter des avantages dans certaines situations.

Enfin, les applications peuvent comporter plusieurs phases algorithmiques nécessitant différentes distributions hétérogènes. L'interaction entre ces distributions doit être soigneusement étudiée lors de l'utilisation d'un système entièrement asynchrone, comme les systèmes d'exécution basés sur les tâches. Le temps gagné sur certaines ressources en les désactivant à la fin de l'algorithme pourrait être utilisé pour les phases suivantes, ce qui signifie que les distributions de déséquilibre pour les phases pourraient être compatibles pour générer une exécution complète et équilibrée. Si les ressources inactives d'une phase sont utilisées intelligemment pour les phases suivantes, on peut s'attendre à une amélioration des performances. Néanmoins, la stratégie contrainte peut être utilisée pour optimiser le chemin critique à la fin de la distribution. Lorsque l'on dispose à la fois de nœuds avec et sans GPU, les tâches du chemin critique qui exploitent mieux les GPU bénéficieraient d'une distribution inégale concentrée sur les nœuds dotés de ces accélérateurs.

B.6 Stratégies Hétérogènes pour les Applications Multi-phases

De nombreuses applications comportent plusieurs phases, chacune ayant des besoins de calcul différents. Par exemple, les phases comprenant l'entrée et la génération de données sont généralement plus adaptées aux CPU, tandis que les opérations à forte intensité de calcul, telles que les noyaux d'algèbre linéaire classiques, peuvent exploiter efficacement les accélérateurs. Cette hétérogénéité interne des applications peut être plus équilibrée lorsqu'elle est combinée à l'hétérogénéité du système, ce qui améliore l'équilibre de la charge en offrant davantage d'options de distribution. Comme les phases individuelles n'utilisent pas toutes les ressources de la même manière, une application a besoin d'une certaine liberté pour exécuter les phases simultanément afin d'améliorer l'utilisation des ressources et les performances. Les défis à relever pour rendre possible le chevauchement des phases comprennent des difficultés de programmation et d'algorithme. Du point de vue du programmeur, le chevauchement des phases est habituellement difficile à obtenir dans les applications parallèles synchrones traditionnelles. Toutefois, un certain asynchronisme est possible dans le paradigme basé sur les tâches si le DAG est correctement structuré. Mais même après la mise en œuvre du chevauchement de phases, l'application reste confrontée au défi algorithmique consistant à trouver une distribution efficace pour les nœuds disponibles. La combinaison de ces défis fait que la plupart des applications manquent une énorme opportunité d'utiliser l'hétérogénéité au niveau du système. La demande de calcul distincte des phases fait de l'hétérogénéité du système un choix intéressant pour

améliorer la répartition de la charge.

Le Chapitre 6 étudie et propose des stratégies pour distribuer ces applications multi-phases sur des ressources avec une hétérogénéité au niveau du système. L'étude utilise les applications ExaGeoStat et Diodon. Dans ces deux applications, le manque de chevauchement entre les phases fait manquer à l'application une énorme opportunité de performance : des ressources puissantes (GPU) pourraient travailler davantage. Avec l'exécution correcte de la phase de génération, les tâches GPU antérieures pourraient être prêtes plus tôt, ce qui augmenterait leur utilisation. Néanmoins, lorsque le chevauchement est corrigé, les phases ont des besoins différents qui pourraient exploiter différents ensembles de nœuds pour améliorer les performances. Cette hétérogénéité signifie que les distributions multi-phases doivent tenir compte de ce chevauchement. Et la distribution d'une phase particulière doit tenir compte des performances des tâches des autres phases dans les ressources et les nœuds hétérogènes.

Le Chapitre présente des optimisations visant à améliorer l'exécution asynchrone basée sur les tâches des phases des deux applications. Dans les deux applications, les mêmes idées d'amélioration des structures spécifiques du DAG telles que les priorités, les indices et les dépendances conduisent à des gains de performance. Dans ExaGeoStat, cinq opérations se chevauchent, tandis que dans Diodon, le chevauchement se situe entre le Gram et la première étape de multiplication de la matrice. Lorsque le chevauchement est correct, le chapitre se poursuit et propose des stratégies pour générer une distribution statique au niveau du système hétérogène qui prend en compte le coût de calcul et le chevauchement des phases. Un programme linéaire modélise le flux de l'application en tenant compte des tâches et de l'hétérogénéité des ressources. La puissance relative est extraite du programme linéaire et utilisée par la suite dans les algorithmes de distribution traditionnels pour certaines phases à forte intensité de calcul. Ce chapitre fournit également un algorithme permettant de déduire la distribution d'une phase particulière en utilisant la distribution de la phase suivante, en maintenant l'équilibre régional et en réduisant les frais généraux de redistribution.

L'évaluation des performances de ce chapitre démontre que les trois catégories générales d'optimisation améliorent le chevauchement des phases asynchrones. Ces optimisations permettent aux tâches critiques des dernières phases de s'exécuter plus tôt et réduisent l'inactivité des ressources, ce qui améliore les performances dans les scénarios homogènes de 29 % à 46 %. Dans le cas des scénarios hétérogènes, les stratégies proposées calculent la puissance relative idéale pour chaque phase sur chaque groupe de nœuds. Toutes les stratégies de distribution hétérogène améliorent les performances jusqu'à 69 % par rapport à une configuration homogène simple dans ExaGeoStat, et de 24 % à 73 % dans Diodon.

B.7 Apprentissage et Adaptation dans les Systèmes Hétérogènes Complexes

Le Chapitre 7 étudie le problème de la découverte de l'ensemble idéal de nœuds hétérogènes à utiliser dans le cadre d'applications multi-phases. Bien que le paradigme basé sur les tâches atténue largement la surcharge de communication, des effets imprévus (par exemple, la contention du réseau ou des synchronisations complexes entre les nœuds) restent possibles et particulièrement difficiles à modéliser. Dans ce contexte, trouver un nombre adéquat de nœuds de calcul pour chaque phase peut être particulièrement difficile à anticiper. Il est donc souhaitable de disposer de méthodes qui apprennent et s'adaptent dynamiquement à ces scénarios complexes et qui améliorent les performances au fil du temps. L'application ExaGeoStat est un bon candidat pour étudier de telles stratégies. Elle comporte de nombreuses itérations au cours desquelles des décisions d'augmentation ou de diminution des nœuds peuvent être prises. De nombreuses applications ont cette structure d'itérations stables (charge de travail stationnaire) mais dont la durée totale est difficile à anticiper dans certaines configurations, car certaines phases s'échelonnent bien et d'autres non. Cela signifie que l'application peut apprendre activement et s'adapter au meilleur ensemble de nœuds hétérogènes auquel elle peut accéder entre les itérations.

Il est possible d'informer le moteur d'exécution des mouvements de données pendant la soumission des tâches, ce qui amène les tâches suivantes à modifier leur nœud d'exécution en conséquence. Ces mouvements peuvent refléter de nouvelles distributions et utiliser plus ou moins de nœuds. Le moteur d'exécution StarPU déplacera toutes les données au bon endroit de manière asynchrone, en chevauchant le calcul.

Le chapitre examine sept stratégies différentes dans la littérature et présente la proposition GP-discontinu. L'évaluation expérimentale comprend une vue d'ensemble des résultats dans 16 scénarios différents, une analyse détaillée étape par étape, l'évaluation des coûts indirects de la stratégie et les extensions possibles de la proposition. Les résultats expérimentaux sont recueillis à l'aide d'environnements réels et de simulation avec des configurations hétérogènes au niveau du système.

Les résultats montrent qu'une stratégie d'apprentissage par renforcement basée sur un processus gaussien informé peut rapidement trouver la meilleure configuration de nœuds pour la phase principale (factorisation). L'approche supérieure, GP-discontinu, utilise des mécanismes de bornage pour filtrer l'espace de recherche, modélise la différence du makespan par rapport à une borne inférieure (et a déjà une certaine connaissance du scénario), et utilise des variables fictives et une tendance linéaire pour modéliser les discontinuités causées par les distributions.

Cette méthode a donné les meilleurs résultats dans les 16 scénarios étudiés. Une autre méthode (plus simple) particulièrement performante est UCB-struct, qui ne prend en compte que des points spécifiques. Cependant, cette méthode ne trouvera que parfois la meilleure configuration, car elle est contrainte de ne pas chercher dans tout l'espace, mais seulement dans les multiples groupes homogènes complets. Finalement, l'utilisation de tous les nœuds pour la phase de génération et uniquement d'un sous-ensemble appris pour la factorisation a permis une accélération de 51,2 % par rapport à l'utilisation de tous les nœuds pour les deux phases. Ces résultats démontrent que l'application peut découvrir pendant l'exécution l'action (nombre de nœuds) qu'elle peut prendre pour améliorer les performances et s'adapter aux ressources hétérogènes.

B.8 Résumé du Comportement des Applications

L'analyse des performances des applications de calcul à haute performance est une étape essentielle pour obtenir des performances correctes. Mais la complexité des applications et des systèmes, y compris les nombreux niveaux d'hétérogénéité (HPE et al., 2022), ajoute des défis considérables à cette activité. L'analyse des performances des applications HPC par le biais de la visualisation est considérée comme une méthodologie avantageuse (GARCIA PINTO et al., 2018), car elle permet une compréhension facilitée de grandes quantités de données de trace (SCHNORR; LEGRAND, 2013). Cependant, même avec cette stratégie, l'espace de représentation est limité et la stratégie de visualisation peut donner des indications erronées. C'est le cas des diagrammes de Gantt, qui se concentrent sur l'utilisation des ressources mais peuvent masquer la progression réelle de l'application. Bien que les visualisations de performance adaptées à une application (MILETTO et al., 2022) ou à des aspects du runtime (NESI et al., 2019) puissent atténuer certains de ces problèmes, des stratégies générales qui résument le comportement global de la performance seraient utiles dans ce domaine.

Le Chapitre 8 propose une méthodologie d'analyse des performances par le biais d'une visualisation introductive. Elle vise à vérifier la progression des applications basées sur des tâches sur des nœuds individuels afin d'indiquer les moments et les groupes de nœuds d'intérêt. Cette méthodologie comprend trois éléments : une métrique de progression par nœud qui utilise la structure des applications basées sur les tâches, une méthode de regroupement pour classer les nœuds et réduire les éléments à montrer, et une visualisation introductive de ces composants.

L'évaluation de ces stratégies comprend quatre scénarios problématiques élaborés avec la bibliothèque d'algèbre linéaire dense Chameleon, qui a correctement détecté le groupe de

nœuds présentant des problèmes. Dans un cas réel avec l'application ExaGeoStat, elle a non seulement géré l'hétérogénéité, mais a également indiqué les nœuds les plus problématiques plus directement qu'un diagramme de Gantt traditionnel. En fin de compte, les stratégies ont permis d'identifier correctement les nœuds problématiques et ont fourni un nouvel angle d'approche qui a permis d'éclairer rapidement la progression de l'application.

B.9 Discussion Finale et Conclusion

L'hétérogénéité fait partie des systèmes HPC, à la fois à l'intérieur d'un nœud avec des accélérateurs et au niveau du système avec plusieurs machines différentes. Cette hétérogénéité au niveau du système (entre nœuds) peut être due à des mises à niveau au fil du temps, à la gestion de charges de travail d'applications différentes ou à des décisions financières. En fin de compte, ces systèmes disposent d'un matériel diversifié offrant de vastes possibilités d'utilisation. D'autre part, les applications HPC sont déjà très complexes, avec de nombreuses opérations aux comportements différents. Ces applications nécessitent des paradigmes modernes capables de gérer des ressources hétérogènes, d'améliorer le développement d'applications tout en permettant la portabilité des systèmes, et de s'affranchir des barrières synchrones inutiles entre les différentes opérations. Le paradigme de la programmation basée sur les tâches est un exemple qui présente de telles caractéristiques. Il utilise un *runtime* dynamique pour planifier les tâches, et les applications sont bien définies dans un DAG. Bien que ce paradigme présente de nombreux avantages, il reste des défis à relever lors de la distribution de ces applications dans ces ressources hétérogènes au niveau du système.

Différents problèmes apparaissent lorsqu'il s'agit de distribuer des applications basées sur des tâches sur des ressources hétérogènes au niveau du système. Lorsque l'on considère uniquement une opération d'application, l'un des problèmes consiste à la répartir correctement sur une gamme variée de nœuds de calcul. Bien que cette répartition doive tenir compte de la capacité de chaque ressource, d'autres aspects du comportement, tels que le chemin critique et les communications, sont également importants. Néanmoins, une opération unique n'est qu'une des nombreuses parties de l'application. Il peut y avoir de nombreuses opérations différentes avec des comportements différents et des affinités de ressources qui appellent idéalement une distribution distincte. De plus, dans les applications basées sur des tâches, ces opérations peuvent s'exécuter de manière asynchrone, se chevaucher, et chacune d'entre elles peut choisir sa meilleure ressource relative. Dans ce contexte, le problème de la répartition de l'application sur différentes ressources devrait maintenant prendre en compte de multiples opérations se

chevauchant et adoptant diverses répartitions. Le nombre de ressources disponibles peut également être excessif pour une phase donnée, car il est parfois avantageux d'utiliser moins de ressources pour réduire certains problèmes. Cependant, la modélisation de tous les comportements qui conduisent à de tels problèmes avant l'exécution de l'application peut s'avérer difficile, et l'adaptation dynamique au cours de l'exécution est une solution possible. Dans tous ces problèmes, il y a un problème transversal qui est comment analyser la performance des applications rapidement et de manière réfléchie. En fin de compte, cette thèse apporte des stratégies à tous ces problèmes corrélés.

Cette thèse choisit comme sujet d'étude le moteur d'exécution StarPU basé sur les tâches et les applications de son écosystème. Le moteur d'exécution StarPU possède la flexibilité nécessaire pour définir des distributions, permet une redistribution asynchrone, possède une fonction de simulation qui peut augmenter les scénarios expérimentaux si nécessaire, et de nombreux outils auxiliaires pour l'analyse des performances. Dans ce sens, les expériences de cette thèse ont utilisé la bibliothèque d'algèbre linéaire Chameleon, l'application d'apprentissage automatique géostatistique ExaGeoStat, et la bibliothèque d'analyse de grands ensembles de données Diodon. Les plateformes utilisées pour les expériences étaient l'infrastructure Grid5000 et le supercalculateur SDumont, tous deux offrant différents niveaux d'hétérogénéité. Ces applications sélectionnées présentent de multiples opérations qui, grâce aux contributions de cette thèse, peuvent exploiter l'hétérogénéité au niveau du système pour améliorer les performances. Les principaux axes de recherche et les contributions sont les suivants.

Le premier ensemble de contributions, dans le Chapitre 5, se concentre sur la distribution des opérations d'une seule application. Nous partons des algorithmes de la littérature, plus particulièrement de l'algorithme 1D-1D. Une première étape a été d'étudier le comportement de ces distributions hétérogènes par rapport à la distribution BC classique, même dans des configurations homogènes, en montrant qu'elle peut gérer un nombre arbitraire de nœuds (y compris des nombres premiers). Ensuite, cette thèse propose deux stratégies, inspirées de la 1D-1D, qui créent des distributions hétérogènes. La première considère le chemin critique et la communication en même temps que les capacités hétérogènes des ressources. Elle contraint la charge de travail finale de l'opération vers des ressources plus rapides, diminuant la communication et améliorant le chemin critique. La seconde stratégie réalise un équilibrage supplémentaire, en assouplissant les contraintes précédentes selon lesquelles certains nœuds ne communiqueraient qu'avec d'autres. Une analyse des performances compare la méthode 1D-1D à celle qui utilise les deux stratégies. Les résultats indiquent un léger gain en combinant ces deux stratégies dans les cas où la méthode 1D-1D était déjà très performante. Une analyse antérieure

qui prévoyait une petite marge d'amélioration (limite inférieure) va dans le même sens que ces résultats. Cependant, les problèmes futurs réutiliseront la stratégie de contrainte dans leurs cas où le chemin critique dans les distributions multi-phases est plus critique. Dans ces cas, cette stratégie présente de meilleurs résultats.

Le deuxième groupe de contributions se concentre sur le problème des applications à phases multiples (opérations), abordé dans le chapitre 6. Ce chapitre montre que la garantie d'une exécution asynchrone entre les opérations et l'adaptation des distributions en fonction de leur chevauchement peuvent améliorer les performances, même dans des scénarios homogènes. Une série de stratégies améliorent le chevauchement asynchrone des phases, améliorant jusqu'à 49 % le makespan en considérant les applications ExaGeoStat et Diodon. Ensuite, il étudie l'environnement hétérogène. Un programme linéaire (LP) calcule la division idéale des tâches par machine en tenant compte de l'hétérogénéité et du chevauchement des interactions de phase. Ce programme linéaire sert également de limite inférieure pour l'application. La puissance relative de chaque nœud pour chaque phase est extraite du résultat de la division des tâches du programme linéaire. Cette puissance par phase et par machine sera l'entrée des algorithmes de distribution (du chapitre 5, y compris la stratégie de contrainte) pour calculer la distribution des opérations sélectionnées. Enfin, cette thèse propose un algorithme pour calculer une distribution pour une phase précédente tout en minimisant les communications de redistribution. Les algorithmes utilisent la distribution suivante comme référence et la division des tâches du LP. Cette méthodologie améliore la performance dans le cas le plus étudié de 69 % dans ExaGeoStat et de 73 % dans Diodon par rapport à l'utilisation d'une stratégie de distribution homogène sur le cluster homogène le plus puissant (partition) de chaque scénario. Le chapitre présente également un comportement détaillé des performances de certains cas qui suggère que l'utilisation de tous les nœuds disponibles pour toutes les phases puisse être problématique.

La troisième série de contributions (Chapitre 7) suit l'exemple donné par le dernier chapitre et se concentre sur la limitation du nombre de ressources dans chaque phase. Le problème est que la contention du réseau, le chemin critique ou d'autres comportements inattendus peuvent détériorer la performance plus que la contribution possible lors de l'ajout d'un nœud de calcul. Cependant, la modélisation de tels comportements est un défi dans ce scénario asynchrone et dynamique. Pour cette raison, cette thèse étudie l'utilisation de méthodes d'apprentissage par renforcement pendant le temps d'exécution pour modéliser le comportement de l'application lors de la sélection d'un nombre arbitraire de nœuds pour une phase donnée. Ce modèle sert de substitut que l'application peut consulter pour guider la décision suivante.

Cette thèse propose une méthode basée sur le processus gaussien (GP) avec sa méthodologie d'intervalle de confiance supérieur (UCB) qui suppose un comportement lisse dans l'espace de recherche. La proposition ajoute des connaissances HPC sur la méthode pour l'adapter à ce problème, en considérant la limitation de l'espace de recherche, la fourniture d'une tendance attendue et la gestion des discontinuités dans le comportement du makespan. Cette partie d'apprentissage agit sur un segment de l'application, dans ce cas, une longue itération. L'application ExaGeoStat présente cette structure, où elle effectue de nombreuses itérations d'optimisation (avec une synchronisation algorithmique inévitable), et chacune d'entre elles comprend de nombreuses opérations asynchrones. Le substitut correspond approximativement à la durée d'une itération. Avant de commencer chaque itération, l'application interroge le substitut et, sur la base du composant UCB, choisit une action qui échange l'exploration et l'exploitation. En fin de compte, le chapitre compare cette méthode à six autres dans 16 scénarios, ce qui montre qu'elle est la seule à traiter tous les cas, améliorant la performance jusqu'à 51,2 % lors de la sélection du nombre de nœuds pour la phase de factorisation d'ExaGeoStat. Cela montre également que le surcoût de la méthode est faible et qu'il peut y avoir un gain limité (en termes de performance) en optimisant le nombre de nœuds en tenant compte de toutes les phases.

Le dernier groupe de contributions provient d'un problème transversal à tous les autres : l'analyse de la performance de ces applications basées sur des tâches. Au cours de l'avancement de cette thèse, toute investigation comprenait une analyse extensive et complète des traces d'exécution. Beaucoup de ces investigations ont conduit à des améliorations et à de nouvelles fonctionnalités dans StarVZ, un package d'analyse de performance. Un exemple a été l'ajout du diagramme de Gantt avec d'agrégation par nœud et par ressource. Cependant, dans les dernières étapes de ce travail, l'extensibilité de la visualisation de l'analyse dans certaines expériences a été considérée comme un problème. Au lieu de s'appuyer directement sur le diagramme de Gantt, qui ne s'adaptera jamais à l'augmentation du nombre de ressources, une autre visualisation simple serait souhaitable pour indiquer les groupes de nœuds qui posent un problème. C'est pourquoi le Chapitre 8 étudie une méthodologie pour servir de résumé de visualisation du comportement de performance de différents nœuds. Elle s'appuie sur une métrique de progression sensible à l'hétérogénéité du système et aux tâches distinctes (provenant de diverses opérations). La méthodologie utilise cette métrique pour chaque nœud à différents pas de temps. Ensuite, comme certains nœuds ont un comportement similaire, elle regroupe la métrique en groupes de nœuds ayant un comportement similaire. La visualisation finale ne montre que ces groupes de nœuds, où les groupes qui progressent lentement dans la métrique de progression sont potentiellement problématiques. Cette visualisation ne nécessite pas plus

d'espace à mesure que le nombre de nœuds augmente. Pour démontrer l'utilité de la méthode, le chapitre utilise des scénarios problématiques élaborés par simulation et des exécutions réelles d'autres enquêtes. En fin de compte, la méthode proposée a bien fonctionné dans tous les cas testés et a permis de détecter tous les nœuds problématiques.

Toutes ces contributions sont motivées par la distribution d'applications basées sur des tâches sur des ressources hétérogènes. Et en fin de compte, elles devraient être utilisées toutes ensemble. Lorsqu'une application commence à s'exécuter, elle doit décider du nombre de nœuds à utiliser par phase, ce qui déclenche les contributions du Chapitre 7. Cette contribution commencera par l'utilisation de tous les nœuds pour toutes les opérations, ce qui nécessite de calculer les distributions en tenant compte de l'hétérogénéité et des opérations asynchrones qui se chevauchent, ce qui conduit à utiliser les stratégies du Chapitre 6. Ensuite, ces stratégies utilisent intrinsèquement le Chapitre 5 pour calculer la distribution finale des phases sélectionnées. Ensuite, la performance de l'application peut être analysée en utilisant les contributions du Chapitre 8 et la méthodologie générale employée dans cette thèse.

Ces travaux ouvrent d'autres perspectives sur de nombreux sujets. La première concerne les ajustements de la distribution des données hétérogènes pendant l'exécution, une action supplémentaire par rapport aux distributions statiques et la sélection dynamique du nombre de nœuds. Il peut s'agir d'affiner les distributions dans des opérations spécifiques tout en tenant compte de la complexité des multiples phases. Une telle situation pourrait encore améliorer l'équilibrage avec un comportement imprévisible. Ces améliorations sont idéalement adaptées à l'application, en exploitant son algorithme et la structure du DAG. Cependant, un moyen systématique de modifier cette distribution dans le contexte du STF et de l'informatique distribuée poserait des problèmes techniques et scientifiques. En l'état actuel du modèle, une fois que le DAG est déroulé dans tous les nœuds, la propriété des données et les changements de tâches nécessiteraient une prise de conscience globale pour garantir la cohérence de la distribution. Dans ce contexte, les décisions de modification de la distribution devraient être égales. Dans le modèle actuel, les nœuds peuvent même soumettre des tâches différentes, en s'appuyant sur le contrôle de l'application pour soumettre toutes les tâches localement nécessaires dans tous les nœuds, ce qui conduit à la correction distribuée du DAG.

Une autre question ouverte est de savoir comment améliorer le modèle actuel puissance multi-phase avec communication, le LP du Chapitre 6. De telles informations supplémentaires pourraient le rapprocher encore plus de la réalité. Ces informations sont difficiles à anticiper en raison du planificateur dynamique et de la complexité générale du chevauchement des opérations. Dans le même contexte de ce modèle, l'un des problèmes actuels est que les informations

relatives à une application sont spécifiques à celle-ci. Bien que certaines applications partagent les mêmes bibliothèques sous-jacentes, comme Chameleon, seules certaines tâches utilisant les mêmes tailles ont pu être réutilisées grâce aux modèles de performance basés sur l'historique.

Dans le contexte de la recherche du meilleur ensemble de nœuds à utiliser, le calcul de l'efficacité des configurations au lieu du seul makespan semble aller dans la bonne direction. Mieux préparer les applications à s'adapter d'une configuration surdimensionnée pourrait améliorer la portée, l'énergie et l'efficacité. En ce qui concerne les stratégies proposées, une situation consiste à envisager d'étendre le modèle GP lors de l'augmentation de l'espace de recherche avec toutes les phases. Le démarrage du modèle et la manière d'élaguer l'espace de recherche sont des questions ouvertes. La flexibilité du GP en tant que substitut pour modéliser la performance et l'efficacité pourrait être utilisée pour explorer d'autres paramètres, ce qui permettrait aux applications HPC de mieux s'adapter à leurs systèmes à chaque exécution. Une possibilité est que l'approche GP puisse modéliser des distributions qui permettent des blocs de précision mixtes ou d'autres formes de compression spécifiques aux données d'entrée. Avec une telle stratégie, on pourrait déterminer les régions et le nombre de blocs de moindre précision tout en modélisant le compromis entre performance et précision.

En ce qui concerne l'analyse et la visualisation des performances, certaines perspectives consistent à toujours faciliter le développement de ces applications, en permettant aux développeurs de comprendre rapidement comment se comporte leur implémentation actuelle. En ce qui concerne la stratégie proposée pour résumer le comportement des applications, les travaux futurs comprennent l'étude d'autres mesures de progression et de techniques de regroupement. Celles-ci pourraient être adaptées à l'application ou à la situation.

En fin de compte, de nombreuses techniques, méthodes et stratégies traditionnelles pour les cas homogènes devraient être revues pour découvrir les multiples opportunités et avantages qui se présentent dans ce contexte hétérogène.