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

DANIEL ALFONSO GONÇALVES DE OLIVEIRA

Energy Consumption and Performance of HPC architectures for Exascale

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Philippe Olivier
Alexandre Navaux

Porto Alegre
2013

CIP – CATALOGING-IN-PUBLICATION

Oliveira, Daniel Alfonso Gonçalves de

Energy Consumption and Performance of HPC architectures for Exascale / Daniel Alfonso Gonçalves de Oliveira. – Porto Alegre: PPGC da UFRGS, 2013.

74 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2013. Advisor: Philippe Olivier Alexandre Navaux.

1. HPC. 2. Exascale. 3. ARM processors. 4. GPU Accelerators. 5. Energy Consumption. 6. Performance. I. Navaux, Philippe Olivier Alexandre. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Luigi Carro
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

ABSTRACT

One of the main concerns to build the new generation of High Performance Computing (HPC) systems is energy consumption. To break the exascale barrier, the scientific community needs to investigate alternatives that cope with energy consumption. Current HPC systems are power hungry and are already consuming Megawatts of energy. Future exascale systems will be strongly constrained by their energy consumption requirements. Therefore, general purpose high power processors could be replaced by new architectures in HPC design. Two architectures emerge in the HPC context. The first architecture uses Graphic Processing Units (GPU). GPUs have many processing cores, supporting simultaneous execution of thousands of threads, adapting well to massively parallel applications. Today, top ranked HPC systems feature many GPUs, which present high processing speed at low energy consumption budget with various parallel applications. The second architecture uses Low Power Processors, such as ARM processors. They are improving the performance, while still aiming to keep the power consumption as low as possible. As an example of this performance gain, projects like Mont-Blanc bet on ARM to build energy efficient HPC systems. This work aims to verify the potential of these emerging architectures. We evaluate these architectures and compare them to the current most common HPC architecture, high power processors such as Intel. The main goal is to analyze the energy consumption and performance of these architectures in the HPC context. Therefore, heterogeneous HPC benchmarks were executed in the architectures. The results show that the GPU architecture is the fastest and the best in terms of energy efficiency. GPUs were at least 5 times faster while consuming 18 times less energy for all tested benchmarks. We also observed that high power processors are faster than low power processors and consume less energy for heavy-weight workloads. However, for light-weight workloads, low power processors presented a better energy efficiency. We conclude that heterogeneous systems combining GPUs and low power processors can be an interesting solution to achieve greater energy efficiency, although low power processors presented a worse energy efficiency for HPC workloads. Their extremely low power consumption during the processing of an application is less than the idle power of the other architectures. Therefore, combining low power processors with GPUs could result in an overall energy efficiency greater than high power processors combined with GPUs.

Keywords: HPC. Exascale. ARM processors. GPU Accelerators. Energy Consumption. Performance.

RESUMO

Uma das principais preocupações para construir a próxima geração de sistemas PAD é o consumo de energia. Para quebrar a barreira de exascale a comunidade científica precisa investigar alternativas que possam lidar com o problema de consumo de energia. Sistemas PAD atuais não se preocupam com energia e já consomem GigaWatts. Requisitos de consumo de energia restringirão fortemente sistemas futuros. Nesse contexto processadores de alta potência abrem espaço para novas arquiteturas. Duas arquiteturas surgem no contexto de PAD. A primeira arquitetura são as unidades de processamento gráfico (GPU), GPUs possuem vários núcleos de processamento, suportando milhares de threads simultâneas, se adaptando bem a aplicações massivamente paralelas. Hoje alguns dos melhores sistemas PAD possuem GPUs que demonstram um alto desempenho por um baixo consumo de energia para várias aplicações paralelas. A segunda arquitetura são os processadores de baixo consumo, processadores ARM estão melhorando seu desempenho e mantendo o menor consumo de energia possível. Como exemplo desse ganho, projetos como Mont-Blanc apostam no uso de ARM para construir um sistema PAD energeticamente eficiente. Este trabalho visa verificar o potencial dessas arquiteturas emergentes. Avaliamos essas arquiteturas e comparamos com a arquitetura mais comum encontrada nos sistemas PAD atuais. O principal objetivo é analisar o consumo de energia e o desempenho dessas arquiteturas no contexto de sistemas PAD. Portanto, benchmarks heterogêneos foram executados em todas as arquiteturas. Os resultados mostram que a arquitetura de GPU foi a mais rápida e a melhor em termos de consumo de energia. GPU foi pelo menos 5 vezes mais rápida e consumiu 18 vezes menos energia considerando todos os benchmarks testados. Também observamos que processadores de alta potência foram mais rápidos e consumiram menos energia, para tarefas com uma carga de trabalho leve, do que comparado com processadores de baixo consumo. Entretanto, para tarefas com carga de trabalho leve processadores de baixo consumo apresentaram um consumo de energia melhor. Concluímos que sistemas heterogêneos combinando GPUs e processadores de baixo consumo podem ser uma solução interessante para alcançar um eficiência energética superior. Apesar de processadores de baixo consumo apresentarem um pior consumo de energia para cargas de trabalho pesadas. O consumo de energia extremamente baixo durante o processamento é inferior ao consumo ocioso das demais arquiteturas. Portanto, combinando processadores de baixo consumo para gerenciar GPUs pode resultar em uma eficiência energética superior a sistemas que combinam processadores de alta potência com GPUs.

Palavras-chave: PAD. Exascale. Processadores ARM. Aceleradores GPU. Consumo de Energia. Desempenho.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BMC	*Baseboard Management Controller*
CUDA	*Compute Unified Device Architecture*
DAG	*Directed Acyclic Graph*
GPU	*Graphics Processing Unit*
HPC	*High Performance Computing*
IPMI	*Intelligent Platform Management Interface*
MSRS	*Model-Specific Registers*
MPSoC	*Multiprocessor System-on-Chip*
NPB	*NAS Parallel Benchmarks*
NVML	*NVIDIA Management Library*
OpenCL	*Open Computing Language*
SIMD	*Single Instruction Multiple Data*
SISD	*Single Instruction Single Data*
SoC	*System-on-Chip*
TDP	*Thermal Design Power*
VFP	*Vector Floating Point*

CONTENTS

1 INTRODUCTION

Scientific applications have helped the development of several areas like weather forecast (KRASNOPOLSKY; FOX-RABINOVITZ; BELOCHITSKI, 2010), oil prospection (ZELJKOVIC; MOUSA, 2011), and health care (HO; MITHRARATNE; HUNTER, 2013), to cite a few examples. These applications are time consuming, i.e., processing intensive, stressing the limits of available memory and processing speed. For this reason, the workload of these applications is often a subset of the desired workload. This simplification is mostly done to cope with the hardware constraints of current HPC systems. Responding to this increasing demand of processing speed, we reached recently PFlops HPC systems, i.e., systems capable of processing 10^{15} floating point operations per second. In the near future, about 10 years from today, we expect to reach the exascale era where HPC systems will have 1000 times more processing speed than current systems (EDWARDS, 2010).

Designing exascale HPC systems brings several challenges. Simply scaling the current technology is unfeasible because we are unable to deal with failure probabilities and power consumption, among others. Looking at the past, the HPC community has already faced challenges to reach Petascale. Back at that time, processors had poor multithread support compared to current multicore architectures that feature several processing cores in a single chip. Petascale HPC systems only became feasible through the usage of multicore processing units (PAWLOWSKI, 2010).

Simply scaling the current technology raises problems such as communication and energy consumption. Observing HPC systems on the Top500 list, scaling these machines to exascale would produce machines that consume Gigawatts of energy. To provide this amount of energy would require a medium size nuclear power plant (WEHNER; OLIKER; SHALF, 2009). The DARPA report (BECKMAN et al., 2011) estimates a reasonable peak of electrical power, they say that the maximum power of the future HPC systems must be below 20 Megawatts. Therefore, to reach the next scale of HPC systems, we need alternatives that can cope with energy consumption restraints (BARKER et al., 2009; YOUNGE et al., 2010).

The energy efficiency of future HPC systems, respecting the limit imposed by the DARPA report, would be 50 GFlops/W. On the Top500 list, we have the fastest HPC system, Titan, performing 17 PFlops at a cost of 8 Megawatts, the energy efficiency is 2.1 GFlops/W. Therefore, we have to increase the energy efficiency 25 times to reach exascale.

Using accelerators is one of the approaches to achieve this energy efficiency. Graphics processing units (GPU) are the accelerators commonly used today. Different from CPUs, they feature hundreds of simple processing cores. GPUs were designed to process images and the game industry made this architecture thrive. GPUs can work well with highly parallel algorithms running thousands of threads at the same time. Comparing to normal CPUs, it is possible to obtain speedups higher than 10 for some GPU algorithms (LEE et al., 2010). Another type of accelerators is the Intel XeonPhi, which is based on several x86 cores. XeonPhi promises

almost zero effort to run old algorithms on it, with a performance similar to GPUs.

Low power processors are another approach to help break the exascale barrier. ARM processors, different from traditional CPUs, focus on spending the least amount of energy possible. The main focus of these processors is mobile and embedded devices. One of the focuses of these devices is to make the battery last longer. However, popular products featuring embedded processors, such as smartphones, need more performance, changing the focus of this architecture. Therefore, ARM processors focus on energy consumption and performance, making this architecture a good candidate for exascale HPC systems.

The Mont-Blanc project is a European effort to develop an exascale machine (MONT-BLANC, 2012a). This project bets on the two approaches described above. The idea is to use ARM processors and GPUs together to obtain a high performance machine at a low power consumption. Mont-Blanc expects to build a prototype that can reach 7 GFlops/W by the end of 2014. After that, the project plans to build a supercomputer capable of performing 200 PFlops consuming 10 Megawatts (MONT-BLANC, 2012b; MONT-BLANC, 2012c; VALERO, 2011).

1.1 Research Goals

The goal of this work is to research approaches that guide us to improve the performance of HPC systems while keeping the energy consumption low. In this context, we start with the assumption that heterogeneous systems using accelerators and low power processors may provide a better energy efficiency compared to current systems.

To verify this assumption, we will study three HPC architectures separately. For the accelerator part we will study GPUs, where some applications can greatly benefit of this architecture. Those applications show speedups of up to one order of magnitude compared to traditional CPUs, resulting in a high energy efficiency. GPU architecture is now being studied extensively, with advanced studies like cycle-accurate simulators and load balancers for CPU-GPU systems. For the low power processors, ARM is the most common. It is contained in most smartphones and embedded devices. However, low power processors are new in the HPC context, and their energy efficiency in this area is still uncertain.

Intel processors are the most responsible for the Top500 systems performance. They are the most common architecture in the HPC field, and have decades of performance improvements. This architecture will represent common HPC processors, and we will use it as a baseline for comparison.

The main goal is to compare these architectures to see which is the best in performance, and which is the best in terms of energy consumption. To make this comparison, we will evaluate these architectures with the same HPC benchmarks, analyzing both execution time and energy consumption of each architecture.

1.2 Contributions

The main contribution of this work is the comparison of three different HPC architectures using different benchmarks. We found that the GPU architecture was the fastest and the best in terms of energy efficiency. The low power architecture was the slowest as expected, however its energy efficiency for light-weight workloads was the best.

1.3 Project Insertion

This work is inserted into the context of the project *Green-Grid: Computação de alto desempenho sustentável*, sponsored by FAPERGS and CNPq.

The work is also part of the *Laboratoire International en Calcul Intensif et Informatique Ambiante* (LICIA) between INF/UFRGS and *Laboratoire d'Informatique* (LIG) from the university of Grenoble France.

1.4 Document Organization

The rest of this document is organized as follows:

- Chapter 2 shows the state-of-the-art research regarding HPC architectures
- Chapter 3 details specific architectures for the research presented in this work
- Chapter 4 presents the proposal as well as the methodology used throughout this work, including how the measurements were performed and which platforms were used
- Chapter 5 presents the results of this work, the comparison of the architectures considering performance and energy
- Chapter 6 concludes and shows some perspectives for future work
- Appendix A shows a new method to model GPUs, producing fast and accurate models

2 STATE-OF-THE-ART RESEARCH OF HPC ARCHITECTURES AND MODELS

This chapter presents an overview of research concerning the two main problems introduced in the last chapter. Related to low power processors, we can see several papers analyzing their capabilities. They compare low power processors with other types of processors to see how they perform against current technology, questioning the possibility of using them in future systems. For accelerators, we can see research modeling and simulating them to understand the behavior of scientific applications. They also consider the power perspective, balancing performance and power of accelerator systems to obtain a better energy efficiency. To contextualize these new architectures, we provide an overview of the current multicore architectures. Finally, we also discuss benchmarks that can be used to analyze performance and energy consumption of heterogeneous HPC systems.

2.1 Multicore Architectures

Blake *et al.* (2009) perform a survey of multicore processors. They state that the attempts to increase the performance of single core processors led to complex, unmanageable, and power hungry designs. With multicore, there are advantages like the raw performance and scalability by the number of cores, rather than frequency, limiting the growth of energy consumption. They also say that performance has been the traditional goal. However, power has joined performance as a first-class design constraint.

They say that combining different types of processing elements into a heterogeneous architecture can be advantageous. Heterogeneity can obtain a power advantage without loss of performance. On the other hand, the programming model for this architecture is much more complicated. Two types of processing elements are described, the first one is in-order with a small die area, low power, and is easily combined into large numbers. There are two ways to increase the performance for this processing element, adding multiple pipelines to fetch and issue more than one instruction in parallel, and increase the number of pipeline stages reducing the logic per stage and increasing the clock. The second processing element is the out-of-order processor that requires very complex and power hungry circuits. They say that out-of-order is more suitable for applications that have a wide range of behavior and a high performance need.

To extract even more performance than superscalar architectures, we can use SIMD and VLIW architectures. Blake *et al.* state that SIMD is highly inefficient for general-purpose processing. To avoid the execution of only one instruction at the same time, VLIW can be used. However, the complexity is moved to the compiler that has the task to guarantee no data and control hazards. They say that SIMD and VLIW are power efficient designs, but are well suited only for specific applications.

The memory system was studied and they state that cache is just one part of the system. Consistency models, cache coherence and intrachip interconnections are also part of the system.

They say that consistency models have an effect on performance. Strong models have a great impact on performance, more complex and slower memory systems, but make programming easy. Weak models make the memory system easy to design but require the programmer to ensure the correct behavior.

Cache configurations were also discussed. Automatically tagged caches are the most common approach, transparent to the programmer. However, they are nondeterministic and use die area for storing the tags. Local stores that can be managed by software can provide deterministic performance, they also use the die area occupied by the tags to store program data. However, local stores also rely on the programmer for proper use.

Among the processors that Blake *et al.* covered by the survey there are 3 low power processors (ARM Cortex A9, Intel Atom, and XMOS XS1-G4), 2 high performance processors (Intel Core i7 and Sun Niagara T2), and some application specific processors like GPUs and DSPs.

2.2 Low Power Processors

Three architectures focusing on low power consumption are presented here. Atom processors from Intel are the fastest, and also have the highest power consumption. Freescale QorIQ processors built on the PowerPC architecture, consume a little less power than Atom but their performance is inferior. ARM processors have an extremely low power consumption even when compared to other low power processors.

Many papers compare ARM processors with different processors, including common processors such as Intel Xeon. Some results are reported for energy efficiency while executing different applications. Some results show that GPUs have a better energy efficiency than traditional CPUs and ARM processors have an even better efficiency than GPUs, almost 2 times better.

2.2.1 Analysis of ARM Processors

Padoin *et al.* (2012a) evaluate the performance and energy efficiency of two ARM developer boards with different ARM processors, ARM Cortex A8 and A9. They used the High Performance Linpack to compare the platforms. The ARM A9 platform presented a performance of 755 MFlops while the ARM A8 achieved only 23 MFlops. The peak power of ARM A9 was 8 W and for ARM A8 was only 1 W. Although the peak power of ARM A8 is extremely low, the poor performance presented by this board led to a low energy efficiency.

In (PADOIN et al., 2013), the authors analyze the performance, scalability, and energy efficiency of three ARM platforms. Each platform showed different behaviors. Tegra 3 had the best performance, Snowball the best energy efficiency and worst performance, and PandaBoard the worst scalability.

Furlinger *et al.* (2011) analyze the performance and energy consumption in commodity

devices. They state the energy efficiency will become the single most important factor for the next generation of supercomputers. This constraint can lead to consumer electronic devices becoming the building blocks of future HPC systems.

They studied the performance and energy consumption of an AppleTV cluster, this device features an ARM Cortex A8 processor. Executing Linpack on all four nodes they achieved 160.4 MFlops while consuming 10 W, yielding an energy efficiency of 16 MFlops/W. However, the Cortex A8 SIMD hardware does not support double precision operations. The VFP unit used for double precision operations is not pipelined and each instruction takes 9 to 17 cycles. Therefore, the peak rate for double precision operations would be about 66 MFlops.

They also compare the AppleTV with a BeagleBoard that has the same processor but a slower clock. Both boards showed comparable results on a per-MHz basis. They also compared with a dual core Intel Atom, using only one core the ARM performs better than Atom. However, using the two cores available the Atom outperform ARM.

Dongarra and Luszczek (2012) analyze architecture, ISA, and other aspects of ARM. They showed a complete implementation of a Linpack benchmark for the Ipad 2. Their results were then compared with the latest performance and power specification from various processors, including common multicores and GPUs.

They divided the processors into three tiers. The first one was desktop and server processors that achieved about 1 GFlops/W. The second was accelerators achieving about 2 GFlops/W. The last tier was ARM that achieves a performance of 4 GFlops/W presenting the best energy efficiency.

2.2.2 Comparisons of ARM with other Processors

The work presented in (PADOIN et al., 2012b) shows a comparison between ARM and Intel Xeon processors. They question the possibility of building the next generation of HPC systems using low power processors. To perform the comparison, a subset of the NAS Parallel Benchmarks was executed. They pointed out that for light-weight workloads ARM has a good energy efficiency compared to Intel Xeon, although for heavy-weight workloads the use of ARM is questionable. The results showed that ARM consumes 10% up to 91% less energy than Intel. However, Intel was 9 to 900 times faster.

Roberts-Hoffman *et al.* (2009) compare ARM Cortex A8 and Intel Atom N330 regarding architecture and benchmarks. They say that for ARM to achieve best performance at the lowest power, superscalar architectures and in-order instructions were used. NEON technology, that can be used as a SIMD accelerator, is the most interesting performance increasing feature introduced by the Cortex A8. The Atom is also in-order featuring two ALUs and two FPUs.

ARM uses several power management techniques like dynamic voltage and frequency scaling and dynamic power switching. It also supports several operation modes consuming different amount of power. For example, standby mode consumes 7 mW, the audio and video decode

mode consumes 540 mW, full-on consumes 1.5 W. Atom N330 is a dual core that combines two Atom N230 and doubles the power consumption to 8 W.

The benchmarks showed that Atom has more performance. However, ARM appears to be more power efficient and provides more performance per dollar than Atom.

Stanley-Marbell *et al.* (2011) made a quantitative analysis for architectures that have simple and more complex cores. They state that low power processors have two potential disadvantages. The first one is the execution performance of non-parallel workloads. The second is that the cost of packaging can surpass the cost of the processor.

They analyze performance, power dissipation, and thermal properties for ARM Cortex A8, PowerPC, and Intel Atom architectures. To obtain this data, hardware counters, current measurements, and thermal imaging via a microbolometer array were used.

The performance results show that Atom with dual core and faster clock is undoubtedly faster than the others. Both ARM and PowerPC have floating point support but PowerPC at the same clock performs much worse than ARM.

For power and thermal measurements, ARM showed again an extremely low power consumption in idle state, 3 times lower than the others. ARM also has the best energy efficiency for sequential workloads. However, with parallel workloads the dual core Intel Atom yields the best energy efficiency. The thermal analysis showed that peripherals outside the processor have significant power dissipation, this mean that board optimizations can lead to more energy efficient systems.

Ou, Pang *et al.* (2012) perform a comparison of an ARM processor cluster against an Intel workstation cluster. They say that data centers are build from commodity hardware, and low energy has been a secondary objective of this hardware. The question they address is what happens if we build data centers with low power processors, where the key design has been energy efficiency since the beginning.

They used micro-benchmarks to compare basic operations like bit, add, multiply, divide, and mod. Intel outperforms ARM in every operation from 4 to 14 times. Running a webserver application, ARM shows a small advantage when the utilization level is less than 20%. For utilization levels higher than 20%, the energy efficiency of ARM was 1.2 to 1.4 times greater than Intel. For in-memory databases, ARM achieves even greater energy efficiency than Intel, reaching 3 to 9.5 times more energy efficiency. However, for video transcoding the efficiency of ARM was only slightly higher than Intel.

They conclude that ARM based clusters are advantageous for low performance applications and light weighted workloads, including the total cost to build the data centers. For computing intensive applications, ARM becomes less energy efficient than Intel which reaches comparable efficiency.

Aroca *et al.* (2012) compare ARM against x86 processors. They discuss whether it is possible to use ARM for server and cluster applications. It is verified that the use of low power servers in parallel applications is suitable.

They also executed a webserver and found out that ARM has a better energy efficiency, from 3 to 4 times better, although Intel had a better response time mostly due to a faster memory bus and bigger cache. For SQL databases, the result was the same as for the webserver.

To evaluate floating point performance, they executed the Linpack benchmark. Analyzing pure performance, x86 always outperforms ARM. However, when analyzing energy efficiency, ARM had better energy efficiency for light-weighted workloads. When the matrices are bigger than 500 x 500, x86 becomes more efficient than ARM.

2.3 Accelerators: Graphics Processing Units

This section presents research related to the evolution of GPUs. We can see cycle-accurate and modular functional GPU simulators. We also show some GPU models that use characteristics of the hardware. One paper also considered the power, modeling both performance and energy. Another paper made a characterization of power and performance when dynamically scaling voltage and clock frequency. Finally, we can see a work distribution method to achieve better energy efficiency from a CPU-GPU system.

2.3.1 Power and Performance Characterization

Jiao *et al.* (2010) characterize power and performance of GPU applications. They used dynamically voltage and frequency scaling (DVFS) to vary the frequency of GPU cores and memory, and observe the impact on performance and energy efficiency.

Three applications were tested: matrix multiplication, matrix transpose, and fast fourier transform. The first two applications obtained an improvement in energy efficiency of less than 10%, reducing the memory and core frequency respectively, the last one had no improvements.

2.3.2 CPU-GPU Load Balancing

Wang *et al.* (2010) proposed a power efficient work distribution method for CPU-GPU systems. The method coordinates work distribution and processors DVFS to minimize the whole system energy consumption.

They used a simple linear model to estimate the execution time of the algorithm, and multiply by the power consumption of the hardware at a determined frequency level to estimate the energy of the execution. For a specific workload, they minimize the equations of total energy consumed.

The energy consumption decreased by 14% using their method. They found that applications with smaller differences of energy consumption between CPU and GPU benefit more with the work distribution. Therefore, balancing processors heterogeneity is important for better energy efficiency.

2.3.3 GPU Modeling

The authors in (VELHO et al., 2013) presented a method to model GPUs. They chose analytically modeling of GPUs to produce fast and still accurate models. They based the modeling on the dwarf taxonomy presented in the Berkeley report (BECKMAN et al., 2011). The idea is to divide scientific applications into dwarfs and model them separately. For three dwarfs tested they report a worst error of less than 11%.

Kerr *et al.* (2010) model CPU-GPU workloads. They performed a principal component analysis into several components of the system. Identifying the principal components of the system, they can reduce the amount of information needed to represent the system. With the principal components they built a polynomial model to predict the execution time for similar applications or different hardware.

To execute on CPU they translate the code for GPU using the Ocelot infrastructure. Ocelot is an emulation and compilation infrastructure that implements the CUDA runtime API, and can translate GPU code to CPU. They found that some applications that have similar performance on a GPU can have total different performance on the CPU, for example one can be 25 times faster than the other in CPU.

They observe that the most accurate models are for the same application on another similar architecture. The worst models are when they use data from CPU and GPU architecture to build a more generic multicore model. Most results have an error of about 20% or less. However, some results showed errors from 30% to 80%.

Zhang *et al.* (2011) presents a quantitative analysis model. They develop microbenchmarks for three major components of GPU applications, and a throughput model considering these three components, the instruction pipeline, shared memory access, and global memory access. Their focus was to identify performance bottlenecks in a quantitative way to guide optimizations.

Comparing the time of each component they identify the bottleneck, and can also infer the next component that becomes the bottleneck when the first one is solved. Information to track down the causes of the bottleneck is also provided. Although the main purpose is to guide optimizations, the accuracy is within 5 to 15% for three case studies.

Hong and Kim (2010) create an integrated model for GPU power and performance. They say that some types of applications do not increase the performance using all the cores available due to bandwidth limitations. Therefore, it is possible to achieve better energy efficiency using fewer cores, discovering an optimal number of cores.

To predict the optimal number of cores, they proposed an integrated power and performance prediction system. To model the power, the GPU is decomposed into several components, then using the access rate to each component the power is estimated by empirical data obtained using micro-benchmarks.

Their results showed that for memory bandwidth limited benchmarks, they can save about

10% of energy consumption using fewer cores. For GPUs that implements power gating mechanisms they can save about 25%.

2.3.4 GPU Simulators

Bakhoda *et al.* (2009) analyze CUDA workloads using a GPU simulator, they present GPGPU-Sim that is a cycle-accurate simulator of GPUs. The simulator supports CUDA Parallel Thread Execution (PTX) instruction set, it can run applications without source code modifications, but requires access to the source code. The simulator executes the CPU code on a normal CPU. Therefore, performance estimation are for the GPU code only.

They analyzed 12 benchmarks and validate the simulator against a real hardware, the correlation coefficient found was 0.899. They used the simulator to vary some architectural aspects to analyze the behavior of the benchmarks used. Some of the findings were that most of the applications are more sensitive to interconnect bandwidth than latency, where reducing bandwidth by half is more harmful than raising the latency by 5 times. They also notice that using less threads than possible can increase the performance by reducing contention of hardware resources.

Although cycle-accurate simulators can result in a good time prediction and give hardware insights, this approach requires unfeasible execution time to simulate one simple HPC application.

Collange *et al.* (2010) present another GPU simulator. They developed a modular functional GPU simulator called Barra based on the UNISIM framework. The main purpose of this simulator is to test the scalability of applications before executing them on real hardware.

They propose a modular functional simulator because the long amount of time that cycle-accurate simulator can take, although it still takes orders of magnitude more time than real hardware. A parallelization was also performed showing speedups of 1.9 for 2 cores and 3.53 using 4 cores, showing that Barra is scalable.

2.4 Heterogeneous Benchmark Suites

Rodinia is a benchmark suite presented by Che *et al.* (2009, 2010). The benchmark goal is to help architects study different platforms like GPU accelerators. They aim to support diverse applications, inspired by the Berkeley's dwarf taxonomy, using state-of-the-art algorithms and also providing input sets for testing different situations.

Rodinia targets GPUs and multicore CPUs implementing the benchmarks using CUDA, OpenCL, and OpenMP. So far, Rodinia includes benchmarks from 6 dwarfs that come from several application domains like medical imaging, bioinformatics, fluid dynamics, and others.

Danalis *et al.* (2010) present the Scalable Heterogeneous computing (SHOC) benchmark suite. SHOC includes benchmarks implemented in OpenCL and CUDA. The main goal of

SHOC is to be truly scalable with the capability to test large clusters with large number of devices.

SHOC also divided the benchmarks into 2 categories, stress tests and performance tests. The first category aimes to identify devices with bad memory or other component problems. The second one measures the system performance.

The Parboil benchmarks presented by Stratton *et al.* (2012) is a benchmark suite similar to Rodinia, They include benchmarks from diverse applications domain and implement them using OpemMP, OpenCL, and CUDA.

Valar is a heterogeneous benchmark suite to study dynamic behavior of this systems (MISTRY et al., 2013). This benchmark aimes to study the dynamic behavior of OpenCL applications with host-device interactions. Valar is composed by OpenCL applications with a wide range of host-device behavior and data sharing with data driven examples.

2.5 Conclusion

Although some papers compare ARM with other architectures, with the exception of Dongarra and Luszczek none of them compare them from the HPC perspective. However, with the papers presented here we can see that ARM is unable to match the performance of others processors but has better energy efficiency for some tasks. This energy efficiency of ARM can be incorporated into future HPC systems, building systems that respect the power constraints imposed on them.

On the other hand, GPU architecture is being deeply studied as the most successful accelerator until today. GPUs have both aspects needed for future HPC systems, high energy efficiency and high performance. However, this architecture is suitable only for some applications that can benefit the most of them.

One approach to build the next generation of HPC systems is to combine the high performance and energy efficiency of GPUs with low power processors like ARM. This combination could lead to an overall high energy efficiency, ARM would handle heavy tasks to the GPU and process the other where ARM is more efficient. Therefore, to verify this approach to build future HPC systems we need a deep understanding of ARM in the context of HPC, and better tools to analyze GPUs in the scale expected for these systems.

3 ARCHITECTURES FOR HIGH PERFORMANCE COMPUTING

This chapter describes the architectures that will be studied in this work. We show characteristics of these architectures that make them more suitable for determined types of applications, considering performance and power. For instance, although one processor may have several mechanisms to accelerate different applications, these mechanisms consume more power and could lead to less energy efficiency compared to other architectures.

First, we present one of the current architectures being used in HPC systems, the Intel processors. Following, we analyze low power architectures, more specifically ARM processors. Next, we show one accelerator architecture developed initially to be used for image processing and gaming but got the attention of HPC community. Finally, we detail heterogeneous architectures that combine low power processors with GPUs.

3.1 High Power Processors: Intel

Intel processors are commonly found in HPC systems. The goal of Intel architectures has been performance for many years. Many enhancements like simultaneous multithreading and multilevel caches were added to the architecture to increase its performance. This led to power hungry processors that are capable of delivering high performance for most of the algorithms.

3.1.1 Simultaneous Multithreading

Intel implemented a simultaneous multithreading technology called Hyper-Threading (MARR et al., 2002). This technology makes a processor appear as two logical processors. Threads will be scheduled for each logical processor, the physical processors will share its resources for the threads executing them simultaneously.

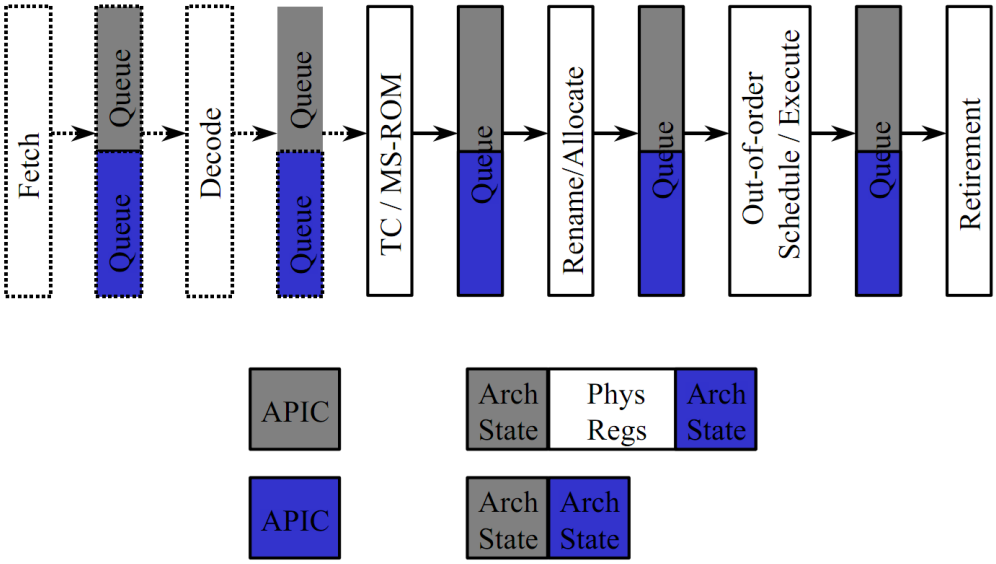
To assure the forward progress of each logical processor, the pipeline blocks inside the physical processors are separated by partitioned or duplicated buffering queues. A high level view of the pipeline is shown in Figure 3.1, where we can see partitioned buffering queues.

3.1.2 Multilevel Caches

The main memory has to be big enough to accommodate all the data of the applications. However, big memories with acceptable price have low speed. The processors need high bandwidth access to memory to be fully occupied, otherwise the processor will be idle waiting for data to be accessed. To serve the data efficiently for the processor, multilevel caches were implemented. Using the temporal and spatial locality, caches can hide the latency to access the main memory.

In the Nehalem architecture, Intel uses three levels of cache, two private caches for each

Figure 3.1 – Intel Xeon Pipeline with Hyper-threading.



Source: The Authors

core and one shared. The level 1 (L1) cache separates instructions and data into 32 KiB each. Level 2 (L2) is a 256 KiB non-inclusive private cache with a write-back policy. Level 3 (L3) is an inclusive 8 MiB shared cache by all cores, with the inclusive cache each line in L1 and L2 is also in L3, this means that traffic due to coherency protocols is minimized.

The Nehalem architecture also introduced a new two-level Translation Lookaside Buffer (TLB). The memory management uses the TLB to accelerate the virtual address translation working as an address translate cache. Nehalem added a second 512 entry TLB level to improve this performance.

3.1.3 SIMD Extension

Processing cores are usually Single Instruction Single Data (SISD), performing one operation on one set of data. However, some algorithms apply the same operation repeatedly on several sets of data. The Single Instruction Multiple Data (SIMD) processing support, where n computing units perform the same operation on different sets of data, could accelerate some computations.

Streaming SIMD Extensions (SSE) instructions in Nehalem can process integer and floating point operations. Nehalem can perform 2 double precision or 4 single precision floating point operations simultaneously.

Separate registers, called MMX and XMM, are used for SIMD instructions. MMX register are 64-bit and used for legacy floating point instructions. The XMM registers are 128-bit and can store up to 4 single precision or 2 double precision floating point operands.

3.1.4 Intel QuickPath Technology

With multicore processors becoming more powerful every day, one bottleneck that arises is the front-side bus. Since processors can execute instructions faster, the rate of instructions and data that have to be fetched increases together with the performance of the processor. Besides fetching the data, in multiprocessor systems there is also communication between the processors to assure the cache coherence. The front-side bus is unsuitable to support all this data traffic.

To provide the high speed interconnection necessary for new multicore processors, Intel developed a new interconnect architecture called QuickPath. This architecture integrates a memory controller into the processor. QuickPath also connects processors and other I/O devices. The Intel QuickPath interconnect in the Nehalem architecture is 20-bit wide and can perform up to 6.4 GT/s or 12.8 GT/s in both directions.

The QuickPath architecture also increased shared memory scalability. Each processor has a dedicated memory that it can access directly. In a multiprocessor system, each processor can access another processor's memory through a high speed QuickPath interconnect.

3.1.5 Turbo Boost Technology

Intel Nehalem can turn off idle cores or some parts of the system as a power saving feature. However, instead of only saving energy, Intel implemented a feature that can boost the performance of single cores when some of them are unused. Intel Turbo Boost Technology turns off idle cores and increases the frequency of the cores in use. Turbo Boost increases the frequency by 133 MHz, 266 MHz, and 400 MHz respecting the thermal and electrical design limits of the processor.

3.2 Low Power Processors: ARM

The ARM Cortex processor family was developed to provide high performance at a low energy consumption (YEUNG et al., 2011). Floating point support together with SIMD capabilities target high processing speed. While several innovations, like shutting down idle cores, take energy consumption into account.

3.2.1 Power Management

With an original focus on embedded and mobile systems, ARM designed low power architectures to last hours or days with a single battery. To do this, some advances in energy saving had to be done.

Micro TLBs reduce the energy consumed translating addresses. Physically addressed caches save energy by reducing the number of cache flushes and refills. Multicore architectures can

save energy by turning off unused cores. ARM can also have some modes between fully operational and shutdown mode called standby and dormant modes, these modes can turn off some units or lower the frequencies to save energy.

3.2.2 big.LITTLE

ARM introduced the technology called big.LITTLE where the multiprocessor combines the highest performance big cores with the most energy efficient little cores. This technology enables a task scheduling focusing on energy consumption. Light and moderate workloads that are not much time limited can be processed by the little cores to save energy. High performance tasks are processed by the big cores, keeping the high performance demanded.

3.2.3 VFP architecture

Vector Floating Point is the ARM Floating Point architecture. It is a coprocessor that provides low cost high performance floating point operations. VFP supports half, single, and double precision formats. This unit is fully compliant with the IEEE 754 standard for binary floating point arithmetic.

In the ARMv7 microarchitecture, the hardware support for vector mode is deprecated due to the NEON technology described in the next section. This unit uses SISD, performing only one operation at a time. The early ARM Cortex A8 architecture uses VFPLite, it provides the same hardware support of a full VFP. However, it requires more clock cycles to perform the same operation.

3.2.4 NEON Technology

To improve the processing performance, the ARM Cortex family also added a SIMD unit called NEON. With the NEON technology, ARM doubles the performance compared with ARMv6 for some applications.

NEON is a 128-bit SIMD unit that can perform 4 simultaneous single precision floating point operations. The technology can also perform operations on 8-bit, 16-bit, or 32-bit operands. Therefore, NEON is able to perform up to 16 operations simultaneously.

For the SIMD instructions, the NEON has 32 64-bits wide registers, they are viewed as 16 registers of 128-bits width. The first processor to use NEON was the ARM Cortex A8.

3.3 Accelerators: Graphics Processing Units

Graphics Processing Units (GPU) arises as an accelerator that can provide high raw performance. Several HPC projects already use GPUs to reach PFlops of performance. GPU started

aiming at rendering massively parallel graphics with substantial investments from the game industry, leading to the development of the current architecture. Nowadays, GPU architectures are widely used to accelerate different scientific applications.

Two main manufacturers, NVIDIA and AMD, offer GPUs with support for HPC. AMD offers GPUs like FireStream 9370 with 1600 cores and SKY 900 with 4096 cores. NVIDIA offers Tesla GPUs that are dedicated for HPC without output ports. The Telsa K20 GPU comes with 2496 cores.

3.3.1 SIMD Architecture

The nature of image applications, where the same operation must be applied to a large set of pixels, led the GPU to be a SIMD architecture. GPUs can execute hundreds of simultaneous operations.

The GPU is composed of hundreds of simple cores that execute the same flow of instructions on different sets of data.

Simple cores are grouped into multiprocessor elements, where the tasks are scheduled. Each core of a multiprocessor executes the same instructions on different sets of data.

3.3.2 Memory and Cache Hierarchy

The memory of GPUs is organized into 2 levels, the global and shared memory. Global memory is slow but visible for all threads of the GPU. Shared memory is fast and visible only to threads in the same multiprocessor.

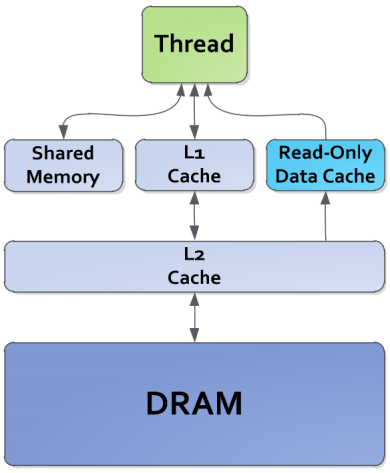
For the NVIDIA Kepler architecture, there are also 2 levels of caches. An unified L2 cache serves all the multiprocessors into the GPU. The L1 cache is private to each multiprocessor and its size is configurable using part of the shared memory. There is also an additional read-only data cache for loading constants. Figure 3.2 shows the memory hierarchy of the Kepler architecture.

3.3.3 SLI and Crossfire

SLI or Crossfire is a technology capable of linking 2 or more GPUs using a direct bus between them. This technology unifies their memory address space, synchronizing data between their memories. This means that one system with 2 GPUs, where each GPU has 1 GB of memory, will appear to have only 1 GB instead of 2 GB when linked together.

GPUs linked together can divide the problem and work together without using the shared PCIe bus. For example, using 2 GPUs for image rendering, each one can work on half of the image, reducing the time to render it. This technology is useful for gaming. However, for general calculations, it is more productive to let them work independently. In this way, the

Figure 3.2 – NVIDIA Kepler Memory Hierarchy.



Source: The Authors

GPUs can work on more data at the same time.

3.4 Performance and Power

This section shows the performance and power for the architectures detailed before. The performance is given by the theoretical double precision operations per second. The power is given by the Thermal Design Power. The theoretical energy efficiency is also given as the performance divided by the power.

Table 3.1 shows the performance, power, and energy efficiency of some latest architectures. Intel Westmere, which is a Nehalem shrink to 32 nm, and AMD Magny-cours are the common processors and show an energy efficiency of about 1 GFlops/W. NVIDIA and AMD GPUs show that accelerators have a much higher performance than common processors without consuming much more power. Accelerators have an energy efficiency from 2 up to 5 GFlops/W.

Low power processors show a very low performance compared to other processors. However, they present an extremely low power consumption. Intel Atom has the highest power consumption, yielding an energy efficiency comparable to common processors. Differently from Intel Atom, ARM processors yields an energy efficiency from 4 up to 11 GFlops/W. Although ARM Cortex A15 performance and power is estimated, ARM Cortex A9 energy efficiency is already comparable to accelerators.

Figure 3.3 shows performance and power of architectures. GPUs are in the top right while low power processors in the bottom left. The ideal spot would be in the top left. However, this spot lacks architectures to fill in. The combination of ARM and GPU could create systems closer to that spot, yielding a high energy efficiency with a high performance.

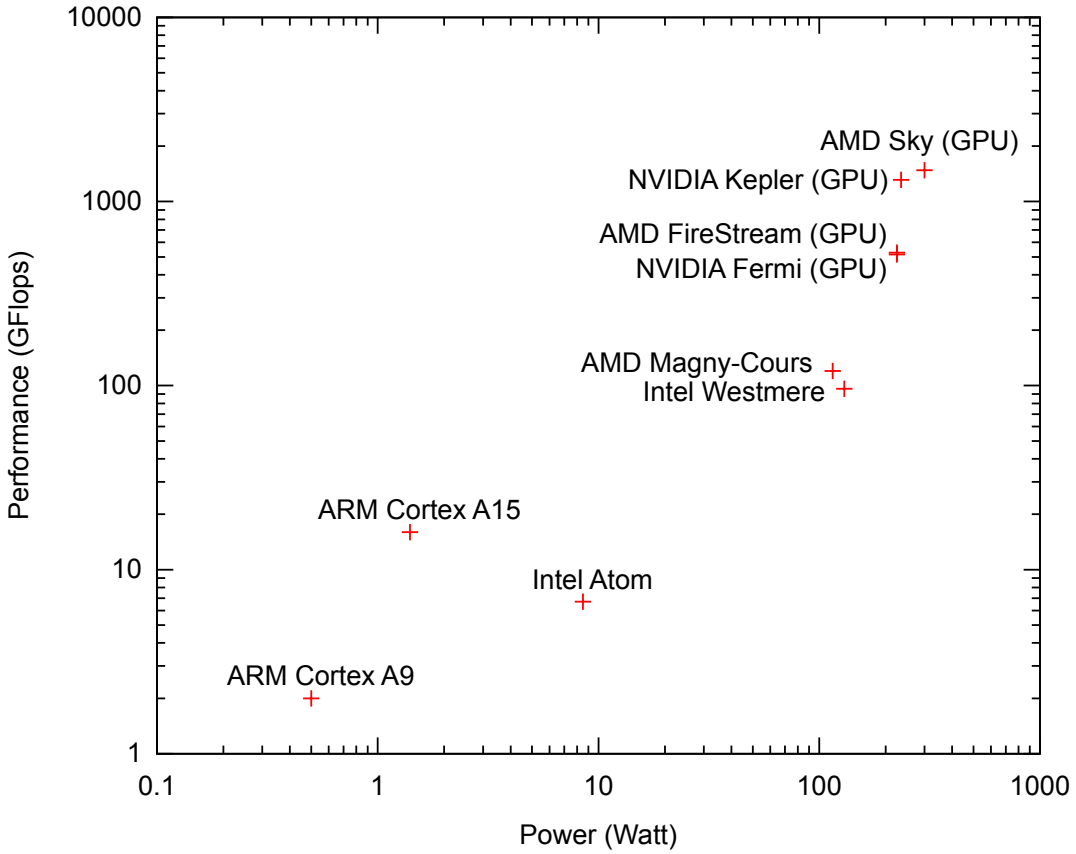
Table 3.1 – Performance, power, and energy efficiency of available architectures.

Processor	Performance (GFlops)	Power (Watt)	Energy Efficiency (GFlops/W)
Inter Westmere E7-8870	96	130	0.74
AMD Magny-Cours 6180SE	120	115	1.04
NVIDIA Fermi GPU M2050	515.2	225	2.29
NVIDIA Kepler GPU K20x	1312	235	5.58
AMD FireStream GPU 9370	528	225	2.35
AMD Sky GPU 900	1478.4	300	4.92
Intel Atom N570	6.7	8.5	0.79
ARM Cortex A9	2	0.5	4
ARM Cortex A15*	16	1.4	11.42

*Estimated

Source: The Authors

Figure 3.3 – Performance and power of architectures.



Source: The Authors

3.5 ARM + GPU

ARM is a low power processor that is getting more performance without compromising power consumption. ARM already has the best theoretical energy efficiency comparing to common processors and GPUs. However, the pure performance is the lowest among these architec-

tures. Moreover, GPUs have an extremely high performance and a better energy efficiency than common processors. Combining ARM and GPU is one solution to build a system that can offer high performance and also high energy efficiency. This section describes two System-on-Chip (SoC) intended for mobile devices that now are offering the latest ARM processors with GPUs capable of general purpose computing.

3.5.1 NVIDIA Tegra

Tegra is a processor built by NVIDIA with mobile computing as the main application of this processor. Devices such as smartphones and tablets feature Tegra processors. Tegra is a SoC integrating ARM, GPU, and other capabilities like audio decoding.

The first Tegra processors are Tegra APX and Tegra 6xx released in 2008 and 2009, this first generation was built with ARM11 processors. Tegra APX was intended for smartphones while Tegra 6xx was for smartbooks and internet devices.

In 2010, NVIDIA released Tegra 2, the second generation of the Tegra family. Tegra 2, shown in Figure 3.5(a), features a dual-core ARM Cortex A9 and an ultra-low power GeForce GPU with 8 cores, delivering high performance for multitasking and games. This generation also supports the Ubuntu Linux distribution, enabling its use for different purposes like web servers and general computation. However, Tegra 2 lacks the NEON SIMD unit, limiting its use for HPC.

Tegra 3, shown in Figure 3.5(b), was announced in 2011 with a quad-core ARM Cortex A9. One of the key innovations introduced by Tegra 3 is the variable SMP, the big.LITTLE technology described before. The fifth core in Tegra 3 is an ARM Cortex A7, this companion core handles the low power tasks. The integrated GPU is an evolution of the ultra-low power GeForce in Tegra 2, the GPU has 12 cores and delivers 3 times the 3D performance compared to Tegra 2.

The GPU of Tegra 3 and Tegra 2 is not capable of general purpose computing. However, there exists the GPU development kit that incorporates the Tegra 3 chip and one separate GPU to build HPC applications.

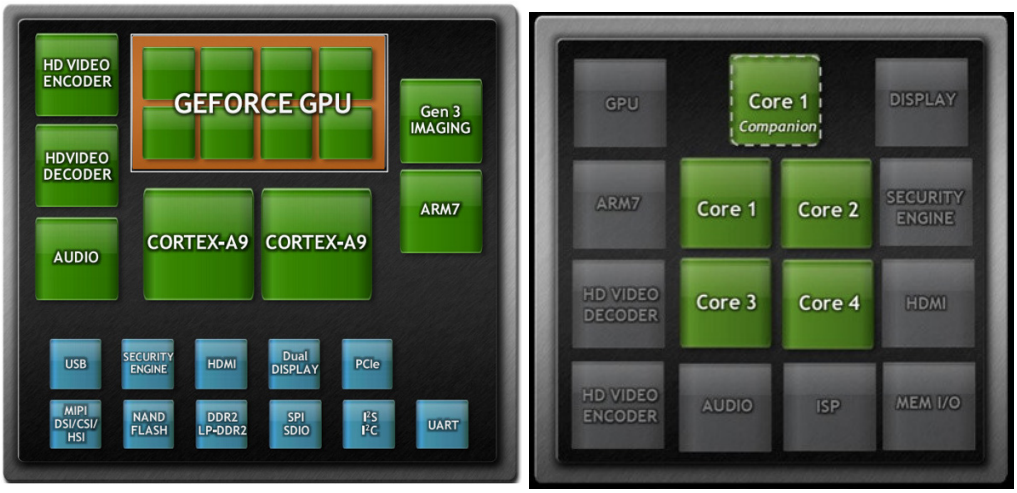
Tegra 4, announced in 2013, is going to be a quad-core ARM Cortex A15. Tegra 4 also has an extra core with the big.LITTLE technology. The GPU featured in the chip has 72 cores and is now capable of general purpose computing.

3.5.2 Samsung Exynos

The Samsung Exynos family is a continuation of the earlier line of SoC that are based on ARM processors. Like NVIDIA, the main application of these SoC is mobile devices.

In 2010, Exynos 3 was released featuring a single core ARM Cortex A8 and a PowerVR GPU. In 2011, Exynos 4 was released, the first one to feature a dual-core ARM Cortex A9

Figure 3.4 – NVIDIA Tegra SoC.



(a) Tegra 2 (b) Tegra 3

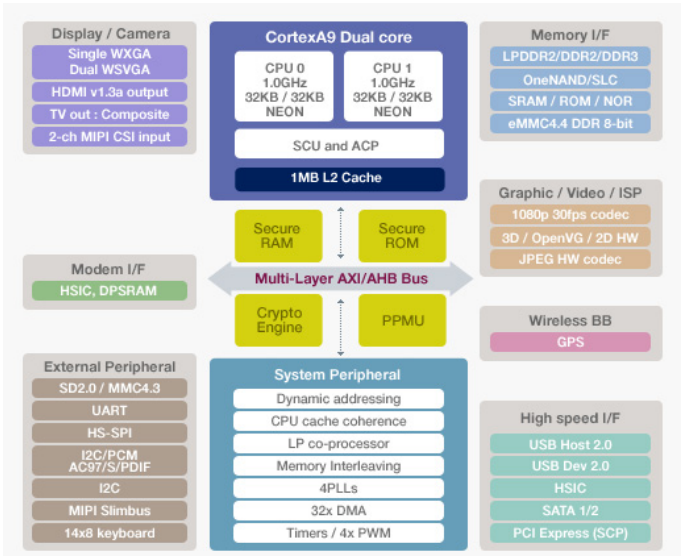
Source: The Authors

with an ARM Mali-400 GPU, shown in Figure 3.6(a). Exynos 4 Quad was released in 2012, featuring a quad-core ARM Cortex A9 and a higher frequency ARM Mali-400 GPU. The GPU in these SoC, like the other ultra-low power GPUs, is not capable of general purpose computing.

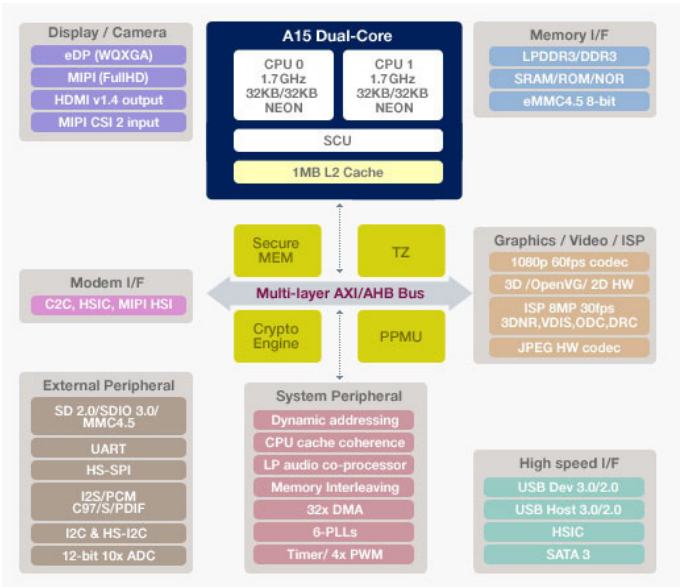
Exynos 5 Dual, shown in Figure 3.6(b) and released in 2012, was the first ARM Cortex A15 featuring a dual-core chip. The GPU is a ARM Mali-T604, capable of general purpose computing and making it more suitable for HPC. In 2013, Exynos 5 Octa was released, featuring a quad-core ARM Cortex A15 with 4 extra ARM Cortex A7 cores using the big.LITTLE technology. The GPU used in Exynos 5 Octa is a PowerVR GPU that is also capable of general purpose computing.

[1] Available in: http://www.samsung.com/global/business/semiconductor/file/product/Exynos4210-0-0.jpg and http://www.samsung.com/global/business/semiconductor/file/product/Exynos5dual-block-diagram-0.jpg. Accessed 2013

Figure 3.5 – Samsung Exynos SoC.



(a) Exynos 4 Quad



(b) Exynos 5 Dual

Source: Samsung Website[1]

4 EVALUATION OF ENERGY CONSUMPTION AND PERFORMANCE

To reach the next step in the evolution of HPC systems, the exascale computer, we need to build systems with architectures that can offer a good ratio between performance and energy consumption. The applications executed in these architectures must be executed fast while spending the least amount of energy possible. Therefore, the energy efficiency of such systems must be high.

Considering this challenge of energy efficiency, this chapter proposes to evaluate the architectures presented in Chapter 3, analyzing performance and energy consumption that these architectures can offer.

4.1 Proposal

The top ranked HPC system in the Top500 list performs 2.1 GFlops/W, to reach the requirements proposed by the DARPA report, it needs to improve the energy efficiency by 25 times. Therefore, to reach exascale HPC systems, we need to find alternatives to achieve the energy efficiency needed.

Two alternatives to achieve the energy efficiency necessary are accelerators and low power processors. Moreover, chips that package low power processors and accelerators into one SoC, may achieve an even better energy efficiency.

To evaluate possible architectures for the exascale system, we select three platforms to represent high power processors, accelerators, and low power processors. We will evaluate the architectures in terms of Time-to-Solution and Energy-to-Solution to focus on the energy efficiency of the platforms.

To represent high power processors, we selected an Intel Xeon E5 processor. Accelerators will be represented by an NVIDIA GPU Tesla K20. Finally, we chose an ARM Cortex A9 as the low power processor.

4.2 Methodology

This section shows the methodology used during this work, describes the architectures and benchmarks chosen to perform the tests, how the measurements were conducted, and the metrics used to evaluate them.

To represent high power processors, an Intel Xeon E5 processor will be used. In the Top500 list of HPC systems from June 2013, 55% of systems use an Intel Xeon E5, and this processor is responsible for 33% of the performance share in the list.

The most common accelerator in the Top500 list are NVIDIA GPUs. However, Intel Xeon Phi is responsible for 18.8% of performance share of accelerators in the list, while NVIDIA GPUs are responsible for 14.4% of performance share. The accelerator used in this thesis will

be an NVIDIA Tesla GPU, Tesla is the most advanced GPU from NVIDIA. Due to lack of access to Intel Xeon Phi, this architecture will not be analyzed.

The most popular low power processor is ARM, present in most of the latest smartphones today, and other embedded devices. ARM is also the targeted architecture of the Mont-Blanc project that intends to build an HPC system with it. The ARM processor available to test is an ARM Cortex A9. However, the last model of ARM processors is the ARM Cortex A15 that is the next generation after ARM A9.

Once the architectures are selected, some benchmarks are needed to test them. The set of benchmarks chosen must be implemented to run natively on all platforms to fairly compare them, this means that each benchmark needs to perform well on each architecture. Therefore, the benchmark suite needs to be implemented using MPI or OpenMP to be executed on the Intel and ARM platforms, and also be implemented using CUDA to be executed on the NVIDIA Tesla GPU.

The benchmarks will then be executed several times, and we will use some metrics to compare the architectures regarding the performance and energy consumed to complete each benchmark. Finally, we will analyze the results of each architecture to reason if some of these architectures have the potential to be used in exascale systems.

4.2.1 Test Environment

In this section, we describe the platforms used during the tests. Table 4.1 summarizes the platforms. In all platforms, we used Ubuntu 12.04 as the operating system, the compiler was gcc version 4.6. For the GPU experiments, we used the CUDA toolkit, version 5.0.

4.2.1.1 High Power Processors

To represent high power processors, we will analyze Intel Xeon processors in a Dell server. The machine is built using two Intel Xeon processors E5-2650 with 8 cores each, the cores are clocked at 2 GHz and the total memory is 32 GB of DDR3 RAM. Thus, this platform offers 16 physical cores and with hyper-threading can execute 32 simultaneous threads. From here on, this test environment will be called Intel Xeon.

4.2.1.2 Accelerators

For the accelerators architecture, we will use an NVIDIA GPU in a computer with a quad-core Intel Core i7 930, operating at 2.80 GHz. The system features a x16 PCIe bus version 2.0, with a speed of 8 GB/s. The GPU card is an NVIDIA Tesla K20c with 2496 CUDA Cores at 706 MHz and 4800 MB of memory. From here on, this test environment will be called Tesla K20.

Figure 4.1 – Intel Xeon Processor.

Source: Intel Website[1]

Figure 4.2 – Tesla K20.

Source: NVIDIA Website[2]

4.2.1.3 Low Power Processors

The ARM platform tested is a PandaBoard that features a dual-core ARM Cortex A9 processor manufactured by Texas Instruments. The platform has 2 cores at 1 GHz and 1 GB of Low Power DDR2 RAM. This board is the successor of the BeagleBoard project that features an ARM Cortex A8. The PandaBoard is intended to offer a prototyping environment for mobile devices with several features and input/output that is unneeded for HPC, like video decoding, HDMI ports, and wireless network controller. From here on, this test environment will be called PandaBoard A9.

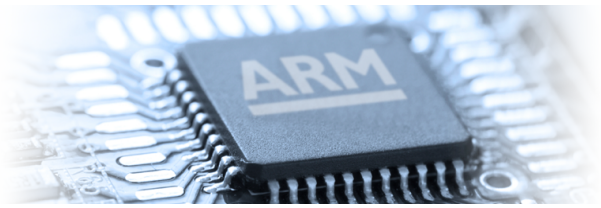
4.2.2 Benchmarks

To fairly compare these architectures, we need benchmarks written in the programming model used natively by them. For the High and Low Power processors, benchmarks written

[1] Available in: http://japan.intel.com/contents/museum/hof/pix/quad_xeon_s.jpg. Accessed 2013

[2] Available in: http://www.nvidia.com/content/tesla/images/TeslaK20_Passive_SC12ShowCover_3Qtr1.png. Accessed 2013

Figure 4.3 – ARM Processor.



Source: ARM Website[3]

Table 4.1 – Test platforms.

Characteristic	Intel Xeon	Tesla K20	PandaBoard A9
# Cores	8	2496	2
Core Clock	2 GHz	706 MHz	1 GHz
Memory	32 GB	5 GB	1 GB
Lithography	32 nm	28 nm	45 nm
Single Processor TDP	95 W	225 W*	0.25 W

* TDP of whole Tesla K20 board.

Source: The Authors

in OpenMP can fully utilize the resources of the architectures. However, the GPU accelerator from NVIDIA uses CUDA that extends languages such as C and needs benchmarks written in CUDA.

Therefore, benchmarks useful for this comparison need to be implemented in OpenMP and CUDA. With the benchmarks presented in Section 2.4, only Rodinia and Parboil are suitable. Both benchmarks suites have recently been updated. However, Rodinia has less compilation and execution complexity and will be used in this work. We describe next the Rodinia benchmark suite.

4.2.2.1 Rodinia Benchmark Suite

Rodinia (CHE et al., 2009) is a benchmark suite for heterogeneous computing developed at the University of Virginia. They aim to help computer architects study platforms such as GPUs and multicore CPUs. Therefore, benchmarks have versions in OpenMP, CUDA, and OpenCL.

Rodinia contains several applications inspired by the Berkeley dwarf's taxonomy described in the scientific report from Asanovic *et al.* (2009). The report points out that scientific applications can be differentiated into 13 types. Therefore, we can expect that a scientific application will behave like one or a composition of up to 13 different types of applications. The Rodinia benchmark suite has several dwarfs covering diverse application domains, from fluid dynamics to bioinformatics, medical imaging, data mining and others. Table 4.2 shows the benchmarks

[3]Available in: http://www.arm.com/images/tpl/our-story-banner.png. Accessed 2013

that compose the Rodinia suite.

Table 4.2 – Benchmarks from the Rodinia Benchmark Suite.

Benchmark	Dwarf	Application Domain
Leukocyte	Structured Grid	Medical Imaging
Heart Wall	Structured Grid	Medical Imaging
MUMmerGPU	Graph Traversal	Bioinformatics
CFD Solver	Unstructured Grid	Fluid Dynamic
LU Decomposition	Dense Linear Algebra	Linear Algebra
HotSpot	Structured Grid	Physics Simulation
Back Propagation	Unstructured Grid	Pattern Recognition
Needleman-Wunsch	Dynamic Programming	Bioinformatics
Kmeans	Dense Linear Algebra	Data Mining
Breadth-First Search	Graph Traversal	Graph Algorithms
SRAD	Structured Grid	Image Processing
Streamcluster	Dense Linear Algebra	Data Mining
Particle Filter	Structured Grid	Medical Imaging
PathFinder	Dynamic Programming	Grid Traversal
Gaussian Elimination	Dense Linear Algebra	Linear Algebra
k-Nearest Neighbors	Dense Linear Algebra	Data Mining
LavaMD	N-Body	Molecular Dynamics
Myocyte	Structured Grid	Biological Simulation
B+ Tree	Graph Traversal	Search

Source: The Authors

Three benchmarks, marked in bold in Table 4.2, from this suite were tested. The benchmarks were selected from different dwarfs to not favor one architecture over another, using algorithms with different characteristics. The benchmarks are described next:

- CFD Solver is a computational fluid dynamics, the benchmark solves a finite volume for the three-dimensional Euler equations for compressible flows. This benchmark, taken from the fluid dynamic domain, represents the unstructured grid dwarf.
- HotSpot is a widely used tool to estimate processor temperature based on an architectural floorplan and simulated power measurements. The thermal simulation iteratively solves a series of differential equations by block. Each output cell in the computational grid represents the average temperature value of the corresponding area of the chip. This benchmark represents the structured grid dwarf taken from the physics simulation domain.
- Needleman-Wunsch is a nonlinear global optimization method for DNA sequence alignments. The potential pairs of sequences are organized in a 2D matrix. In the first step, the algorithm fills the matrix from top left to bottom right, step-by-step. The optimum alignment is the pathway through the array with maximum score, where the score is the value of the maximum weighted path ending at that cell. Thus, the value of each data element depends on the values of its northwest-, north- and west-adjacent elements. In

the second step, the maximum path is traced backwards to deduce the optimal alignment. The dwarf this benchmark represents is dynamic programming and was taken from the bioinformatics domain.

4.2.3 Metrics

To perform the comparisons, two metrics will be used, these metrics are described below.

Time-to-Solution: the value of Time-to-Solution (in seconds) gives an estimation of the amount of time to achieve useful output.

Energy-to-Solution: the value of Energy-to-Solution (in Joules) gives the amount of energy consumed to achieve useful output. It can be calculated as the multiplication of the average power by the Time-to-Solution.

4.2.4 Measurements

This section shows how the time to complete the benchmarks was measured, and how the energy consumed was measured.

4.2.4.1 Time Measurements

The time considered was the time to execute the kernel only. The time to load the input, read files, transfer memory from the CPU do GPU memory in the case of GPU, were all disconsidered. Memory transfer is a step necessary only for accelerators, the other steps are necessary for all platforms. However, this step can be performed in parallel to the accelerator kernel execution, and pipelined implementations where data transfer and kernel execution are mixed can hide this time.

To measure this time, system calls (gettimeofday) from the operating system are performed before and after the kernel execution, giving us the time elapsed to execute the kernel.

4.2.4.2 Energy Measurements

Similarly to time, the energy considered was only the energy consumed during the kernel execution.

Regarding the Intel platform of high power processors, we have 2 measures that can be done. The first one is to measure the power consumed only by the processor. The second one is to measure the power consumed by the mainboard, which includes main memory and any device connected to and powered by the mainboard, this excludes the power consumed by hard disks that are powered directly from the power supply. We will explain next how these two measures

are performed.

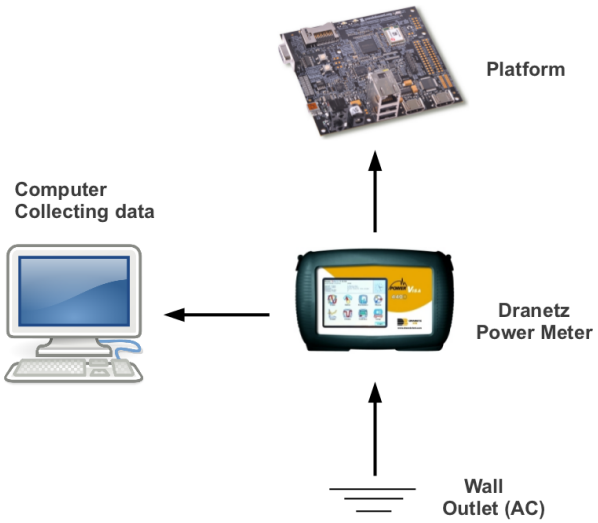
The Intel processor used is from the Sandy Bridge family, this processor includes some Model-Specific Registers (MSRS), with some registers included in this family we can measure the energy in joules consumed by the processor. Therefore, to measure the energy consumed only by the processor we can read this register before and after the kernel execution, the difference of the readings is the energy consumed to execute the kernel of the application.

To measure the energy of the mainboard in the Intel platform we use the Intelligent Platform Management Interface (IPMI), this interface uses the Baseboard Management Controller (BMC) that is a microcontroller embedded in the mainboard. With this we can measure the instantaneous power consumed by the mainboard. Measuring the instantaneous power during the kernel we can have the total energy consumed by the kernel.

The energy consumed by the NVIDIA Tesla can be measured similarly to the energy consumed by the mainboard in the Intel platform. NVIDIA System Management Interface, based on the NVIDIA Management Library (NVML), provides the instantaneous power consumed by the whole Tesla board.

The downside of measuring energy using IPMI and NVIDIA System Management Interface is that we have to be constantly measuring the instantaneous power. This continuous measurement can interfere with the performance of the benchmark, and if the measurement rate is low the error of the energy calculation can be too high. Therefore, we have to verify if the measurement rate is reasonable.

Figure 4.4 – Hardware setup to measure Instantaneous Power (W).



Source: The Authors

PandaBoard is the most challenging platform to measure the energy. Although the other two platforms offer some management interface that provides power information, the PandaBoard platform contains no equivalent. One approach to measure the energy is to use a power meter.

The power meter is connected between the power outlet and the power supply as seen in Figure 4.4.

In the work presented in (PADOIN et al., 2012b), the authors measured the power of PandaBoard during some benchmarks from the NAS Parallel Benchmarks (NPB) using the power meter, the peak power measured was a little less than 8 W for all benchmarks. Therefore, in the experiments performed in this thesis, we will consider the peak power of PandaBoard to be 8 W. The energy consumed will be obtained by multiplying peak power by the time spent to run the benchmarks. This approach is the worst case scenario, where during all the execution time the board is operating at full power.

5 RESULTS AND EVALUATION

This chapter shows the results obtained for each tested benchmark. In all metrics measured, we executed each benchmark at least 30 times and calculated the confidence interval for a confidence level of 95%.

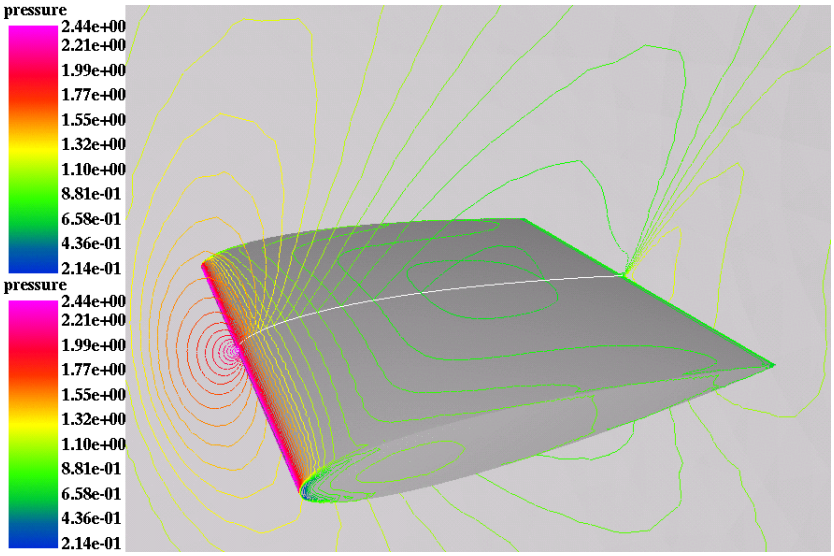
5.1 CFD Solver

This section presents the results obtained for the CFD Solver benchmark. CFD solvers based on unstructured grid come as a challenge for GPUs, the data dependency and irregular access to memory from these algorithms becomes a great bottleneck. The architecture of GPUs was build with a focus on image processing that uses structured grids with no data dependencies and regular memory accesses.

The CFD solver implementation for GPUs used by the Rodinia suite is described in (CORRIGAN et al., 2011), they used several techniques to optimize unstructured grid solver for modern GPUs.

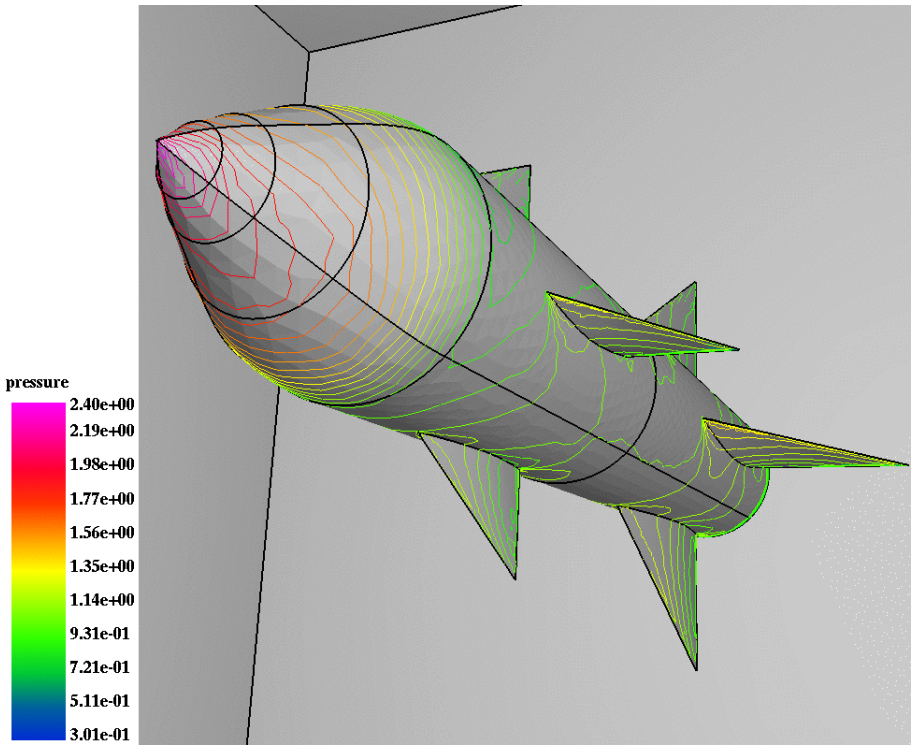
Two workloads are tested under supersonic flow. The first one is a NACA0012 wing for aircraft developed by the National Advisory Committee for Aeronautics. The second one is a missile. Figures 5.1 and 5.2 show the objects and the pressure at the surface of them under supersonic flow.

Figure 5.1 – Pressure at the surface of the NACA0012 wing.



Source: Corrigan (2009, p. 225)

Figure 5.2 – Pressure at the surface of missile.



Source: Corrigan (2009, p. 227)

5.1.1 Time-to-Solution

The Time-to-Solution of each architecture to complete both workloads are presented in Table 5.1. Although unstructured grids are not the best fit for GPUs, Tesla K20 had the best Time-to-Solution. Figure 5.3 shows the Time-to-Solution. **Tesla K20 was 7 to 8 times faster than Intel Xeon and about 600 times faster than PandaBoard A9.** Intel Xeon was about 80 times faster than PandaBoard A9.

Table 5.1 – Time-to-Solution in seconds for CFD solver benchmark.

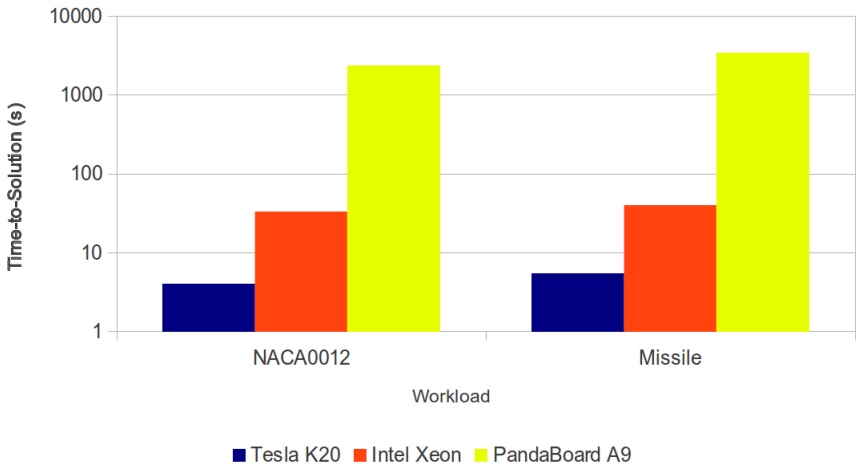
Workload	Intel Xeon	Tesla K20	PandaBoard A9
NACA0012	32.67 ($\pm 8.3 \times 10^{-2}$)	3.98 ($\pm 1.5 \times 10^{-4}$)	2360.75 (± 1.5)
Missile	39.31 ($\pm 1.1 \times 10^{-1}$)	5.38 ($\pm 2.7 \times 10^{-4}$)	3403.70 (± 5.6)

Source: The Authors

5.1.2 Energy-to-Solution

Table 5.2 shows the average power in watts for each architecture and Table 5.3 shows the Energy-to-Solution in joules. For the Intel Xeon architecture we have 2 measures as explained

Figure 5.3 – Time-to-Solution of CFD solver.



Source: The Authors

in the previous chapter, the first one is the energy spent by the main board, and the second is the energy spent only by the two processors.

Table 5.2 – Average Power in watts for CFD solver benchmark.

Workload	Intel Xeon	Tesla K20	PandaBoard A9
NACA0012	196.56 ($\pm$0.74)	91.09 ($\pm$0.65)	8
Missile	205.58 ($\pm$0.65)	84.73 ($\pm$0.73)	8

Source: The Authors

Table 5.3 – Energy-to-Solution in joules for CFD solver benchmark.

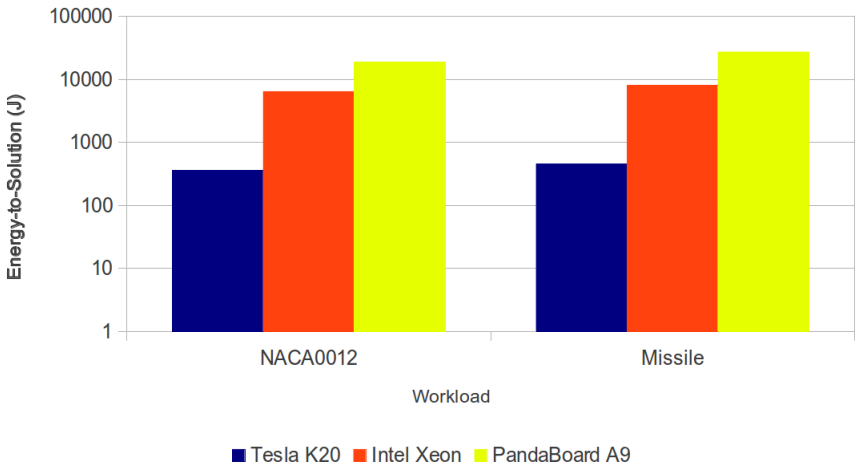
Workload	Intel Xeon	Intel Xeon (processors only)	Tesla K20	PandaBoard A9
NACA0012	6421.62	1981.57 ($\pm$4.83)	362.54	18886.00
Missile	8081.35	2537.71 ($\pm$7.08)	455.85	27229.60

Source: The Authors

Figure 5.4 shows the Energy-to-Solution. Tesla K20 was also the architecture that presented the best energy consumption, **Tesla K20 consumed 18 times less energy than Intel Xeon, and about 5.5 less energy considering only the Intel Xeon processors. Tesla K20 also consumes 52 to 59 times less energy than PandaBoard A9.**

Intel Xeon consumed less energy than PandaBoard A9. However, this consumption was only 3 times less than PandaBoard A9 comparing to a Time-to-Solution 80 times faster. From the processors measurements, we also observed that both processors were responsible for about 30% of the consumption measured for the main board.

Figure 5.4 – Energy-to-Solution of CFD solver.



Source: The Authors

5.2 Hotspot

Hotspot is a structured grid algorithm and the best fitted for GPUs architecture. The input is a matrix representing the processor simulated, three workloads are distributed together with the benchmark in the Rodinia suite, the workloads have square matrices with dimensions of 64, 512, and 1024.

5.2.1 Time-to-Solution

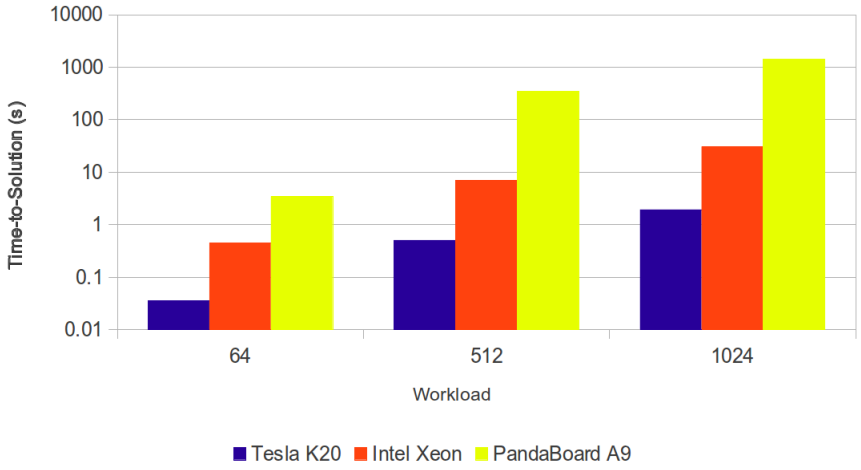
Table 5.4 shows the Time-to-Solution for each architecture. As expected, Tesla K20 was faster than the other architectures. The Time-to-Solution is shown in Figure 5.5. **Tesla K20 was 12 to 15 times faster than Intel Xeon and 94 to 739 times faster than PandaBoard A9.** Intel Xeon was 7 to 49 times faster than PandaBoard A9.

Table 5.4 – Time-to-Solution in seconds for Hotspot benchmark.

Workload	Intel Xeon	Tesla K20	PandaBoard A9
64	0.4458 ($\pm 2.32 \times 10^{-2}$)	0.0359 ($\pm 1.89 \times 10^{-5}$)	3.41 ($\pm 6.16 \times 10^{-2}$)
512	6.9030 ($\pm 3.98 \times 10^{-1}$)	0.5027 ($\pm 4.99 \times 10^{-5}$)	344.09 ($\pm 9.44 \times 10^{-1}$)
1024	30.4220 ($\pm 3.82 \times 10^{0}$)	1.9251 ($\pm 7.93 \times 10^{-5}$)	1423.22 ($\pm 3.21 \times 10^{0}$)

Source: The Authors

Figure 5.5 – Time-to-Solution of Hotspot.



Source: The Authors

5.2.2 Energy-to-Solution

The average power for this benchmark is presented in Table 5.5. In the Table 5.6 and in Figure 5.6, we can see the Energy-to-Solution for each architecture. **Tesla K20 also presented the best energy consumption with 37 to 52 less energy consumption than Intel Xeon, and 11 to 17 less energy considering only the Intel Xeon processors. Tesla K20 consumed 32 to 128 less energy than PandaBoard A9.**

Table 5.5 – Average Power in watts for Hotspot benchmark.

Workload	Intel Xeon	Tesla K20	PandaBoard A9
64	90.50 (±3.35)	23.74 (±1.87)	8
512	161.34 (±2.05)	42.59 (±1.69)	8
1024	180.62 (±1.41)	76.48 (±2.78)	8

Source: The Authors

Table 5.6 – Energy-to-Solution in joules for Hotspot benchmark.

Workload	Intel Xeon	Intel Xeon (processors only)	Tesla K20	PandaBoard A9
64	40.34	12.61 (±0.46)	0.85	27.28
512	1113.73	372.87 (±23.49)	21.41	2752.72
1024	5494.82	1637.19 (±169.78)	147.23	11385.76

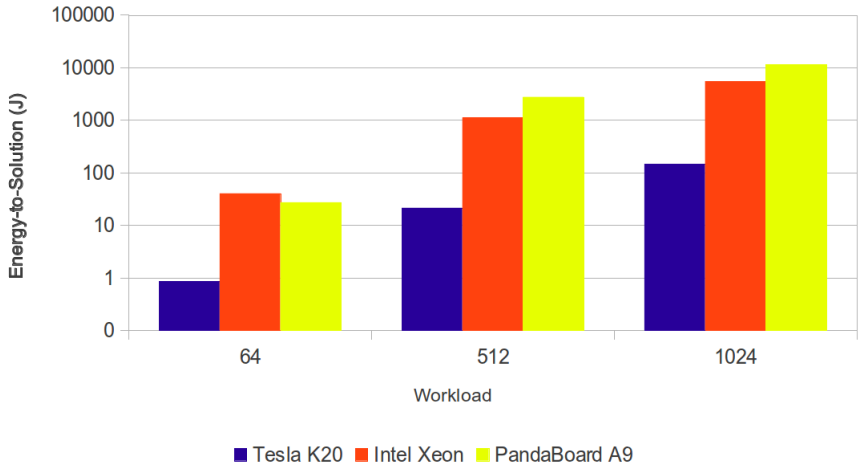
Source: The Authors

Similarly to the CFD benchmark, Intel Xeon processors were responsible for about 30% of the consumption measured for the main board. However, comparing against PandaBoard A9,

Intel Xeon consumed 1.47 more energy for the 64 workload and consumed 2 times less energy for the other two workloads.

Other papers already demonstrated that with small workloads ARM architectures may present better energy efficiency than other architectures. With some under utilization of the other architectures, PandaBoard A9 consumed less than Intel Xeon for the 64 workload result.

Figure 5.6 – Energy-to-Solution of Hotspot.



Source: The Authors

5.3 Needleman-Wunsch

Neddleman-Wunsch is a dynamic programming algorithm and because of data dependencies it may not perform well in GPUs architectures. The workload is also represented by a square matrix with dimensions ranging from 1024 to 4096. However, PandaBoard A9 can only execute until the workload with matrix dimensions of 2560, because PandaBoard A9 gets out of memory for bigger matrices.

5.3.1 Time-to-Solution

Time-to-Solution of each architecture is shown in Table 5.7 and in Figure 5.7. Tesla K20 was again the fastest of them, **Tesla K20 was 6 to 9 times faster than Intel Xeon, and 96 to 218 faster than PandaBoard A9.** Intel Xeon was 16 to 37 times faster than PandaBoard A9.

5.3.2 Energy-to-Solution

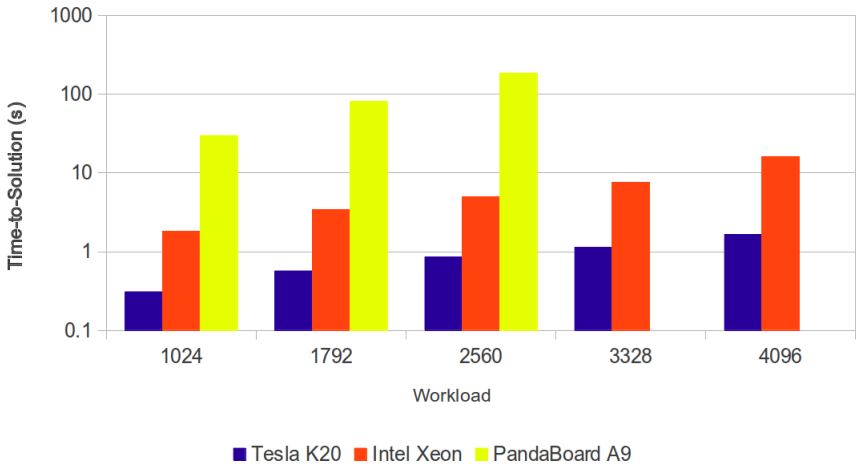
Table 5.8 shows the average power for this benchmark. In Table 5.9 and Figure 5.8 we can see the Energy-to-Solution results. **Tesla K20 consumed 25 to 37 less energy than Intel Xeon**

Table 5.7 – Time-to-Solution in seconds for Needleman-Wunsch benchmark.

Workload	Intel Xeon	Tesla K20	PandaBoard A9
1024	1.80 ($\pm 1.9 \times 10^{-2}$)	0.31 ($\pm 2.8 \times 10^{-5}$)	29.86 ($\pm 2.7 \times 10^{-2}$)
1792	3.40 ($\pm 4.3 \times 10^{-2}$)	0.57 ($\pm 4.8 \times 10^{-5}$)	80.34 ($\pm 2.4 \times 10^{-2}$)
2560	4.98 ($\pm 2.1 \times 10^{-2}$)	0.85 ($\pm 8.2 \times 10^{-5}$)	185.64 ($\pm 1.2 \times 10^{-1}$)
3328	7.55 ($\pm 6.7 \times 10^{-2}$)	1.14 ($\pm 8.4 \times 10^{-5}$)	-
4096	16.09 ($\pm 8.0 \times 10^{-1}$)	1.64 ($\pm 1.0 \times 10^{-4}$)	-

Source: The Authors

Figure 5.7 – Time-to-Solution of Needleman-Wunsch.



Source: The Authors

and 8 to 12 times less energy considering only the Intel Xeon processors, Tesla K20 also consumed 27 to 46 times less energy than PandaBoard A9. Comparing Intel Xeon against PandaBoard A9, we can see again that with small workloads ARM architectures can be better or get close to common architectures, Intel Xeon consumed 1.04 to 1.83 less energy than PandaBoard A9.

Table 5.8 – Average Power in watts for Needleman-Wunsch benchmark.

Workload	Intel Xeon	Tesla K20	PandaBoard A9
1024	127.30 (± 5.32)	28.24 (± 1.40)	8
1792	151.52 (± 3.02)	32.61 (± 1.00)	8
2560	163.07 (± 3.34)	37.45 (± 1.44)	8
3328	172.96 (± 2.36)	42.25 (± 1.57)	-
4096	181.72 (± 1.30)	47.82 (± 1.03)	-

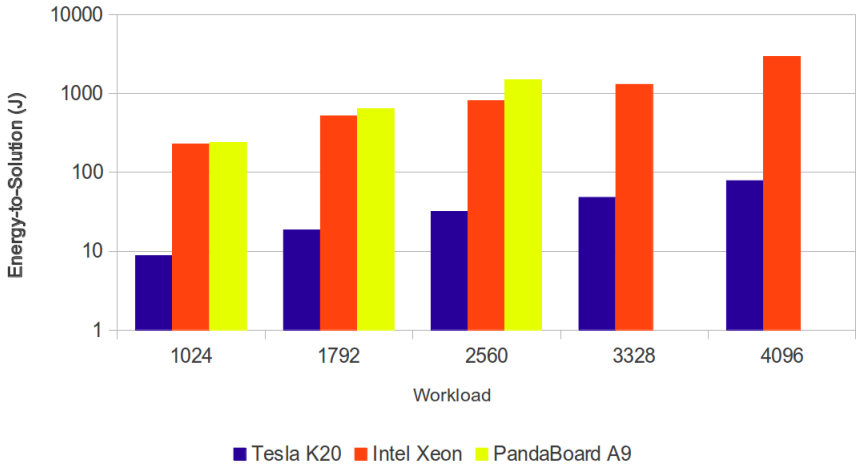
Source: The Authors

Table 5.9 – Energy-to-Solution in joules for Needleman-Wunsch benchmark.

Workload	Intel Xeon	Intel Xeon (processors only)	Tesla K20	PandaBoard A9
1024	229.14	83.62 (±1.24)	8.75	238.88
1792	515.17	175.62 (±3.30)	18.59	642.72
2560	812.09	266.09 (±1.68)	31.83	1485.12
3328	1305.85	405.16 (±3.34)	48.17	-
4096	2923.87	945.34 (±47.34)	78.42	-

Source: The Authors

Figure 5.8 – Energy-to-Solution of Needleman-Wunsch.



Source: The Authors

5.4 Summary of the Results

From the benchmarks tested, we observe that Tesla K20 was the fastest architecture. For all the workloads and benchmarks, Tesla K20 was at least 90 times faster than PandaBoard A9, and 5 times faster than Intel Xeon. For some workloads, Tesla K20 was 600 times faster than PandaBoard A9, and 15 times faster than Intel Xeon. For applications similar to the benchmarks tested, we can assume that the GPU architecture is the best choice considering Time-to-Solution.

Comparing general purpose processors, we can notice the superiority in performance from high power processors. For all the workloads and benchmarks, Intel Xeon performed at least 7 times faster than PandaBoard A9, and was up to 80 times faster. However, PandaBoard A9 is the only platform not meant for intensive computation, PandaBoard A9 is a developer board meant for mobile prototyping while Intel Xeon is a high end processor and Tesla K20 a high end accelerator.

Considering the Energy-to-Solution, Tesla K20 was also the architecture that consumed the least energy. For all the tests, Tesla K20 consumed at least 18 times less energy than In-

tel Xeon, and 25 times less energy than PandaBoard A9. Therefore, in terms of energy efficiency, Tesla K20 is the best architectural choice, providing the best Time-to-Solution as well as the best Energy-to-Solution.

Comparing low power against high power processors, we observe that in the HPC context, low power processors are unable to provide an energy efficiency better than high power processors. Intel Xeon consumed less energy than PandaBoard A9 with the exception of one test case, although PandaBoard A9 consumption was never higher than 3 times the consumption of Intel Xeon.

Table 5.10 shows the peak power consumed by each architecture for all benchmarks and in idle, we can see that the peak power of PandaBoard A9 fully loaded is only half of the Tesla K20 in idle, and only about 9 times less the power of Intel Xeon also in idle.

Table 5.10 – Peak power in watts for each platform and the idle power.

Platform	CFD Solver	Hotspot	Needleman-Wunsch	idle
Intel Xeon	223	198	206	70
Tesla K20	111	121	91	16
PandaBoard A9	8	8	8	-

Source: The Authors

Comparing all the benchmarks, we notice that the GPU is the fastest for all three. For the Hotspot benchmark that is the best suited for GPU, Tesla K20 presented a speedup of 15 times over the Intel Xeon. While the other benchmarks the speedup presented by Tesla K20 over Intel Xeon stayed between 5 to 10 times. For some other algorithms, architectures other than GPU could be the best choice.

6 CONCLUSION AND PERSPECTIVES

This work presented an evaluation of HPC architectures, analyzing Time-to-Solution and Energy-to-Solution considering the energy efficiency of each architecture. Three distinct benchmarks were executed in all architectures to evaluate which architecture is the fastest and which one consumes the least energy.

The results showed that the GPU architecture, for the applications similarly to the three tested, was the fastest one performing at least 5 times faster than Intel Xeon and 80 times faster than PandaBoard A9. Tesla K20 also consumed at least 8 times less energy than the others.

GPU architecture is undoubtedly the best architecture of them for this set of applications. However, as explained in Section 4.2.4, the tests disconsidered the memory transfers needed for the GPU architecture, although this time can be hidden it may diminish the GPU results and the data movement also costs energy that was not measured.

For light-weight workloads, PandaBoard A9 showed a good energy efficiency surpassing Intel Xeon. Therefore, for light-weight tasks that do not require a restrictive time limit to be completed, low power architectures are the best choice in terms of energy consumption. However, for heavy-weight workloads, high power architectures can be dozens of times faster, providing an energy efficiency equal or better than low power architectures.

Another point to notice is that Intel Xeon processors are responsible for about 30 % of the power consumed by the platform. Considering that the ARM processors used by PandaBoard A9 have a TDP of 0.25 W, that would mean that the processor would be responsible for only about 3 % of the board consumption. An ARM platform built for HPC, with no unneeded circuits consuming power, could raise the total energy efficiency of the platform.

From these observations, we conclude that heterogeneous systems with low power processors and GPUs come as an interesting solution to build energy efficient HPC systems. GPU is an accelerator and needs a host system, like the Intel Xeon platform or an ARM platform. Therefore, GPU architectures may be the best choice, however the host system will also consume energy and can impact the energy efficiency of the HPC system. In this context, two possibilities emerge, the first one is to use GPUs together with high power architectures, both architectures would process the solution and the overall energy efficiency will be somewhere in the middle of each efficiency. The second one is to use GPUs together with low power architectures, where GPUs would do all the processing work and low power architectures will only support the GPUs, sending data to GPUs and doing small works to not impact the energy efficiency of GPUs.

To help clarify the last statement we can look into the peak power of each platform tested. We can see that the peak power of PandaBoard A9 is less than the idle power of the other architectures. Therefore, low power architectures together with GPUs may present an energy efficiency close to the one presented by the GPU alone.

6.1 Research Perspectives

Although low power architectures is not a direct match for current high power architectures, the combination of them with GPUs could produce HPC systems with a high energy efficiency. Low power architectures proved to be very efficient for light-weight tasks, and showed a great increase of performance compared to older designs. GPUs also improved their support for scientific applications, increasing double precision performance, moving from the sole purpose of image processing to a more general purpose.

GPUs are now moving into the system-on-chip of low power processors, leading to even more energy efficiency. Recent platforms combine ARM and a programmable GPU into one chip, this is the case of Tegra 4 and Exynos 5 Dual. Heterogeneous systems using these platforms can be the base of future HPC systems, providing high performance and consuming low power.

For the future, we intend to provide fast and accurate GPU models and implement them in simulators like SimGrid (CASANOVA; LEGRAND; QUINSON, 2008) where we can simulate applications in exascale. Platforms combining low power processors and GPUs will be studied to understand the applications behavior and energy efficiency that these architectures can provide.

REFERENCES

AROCA, R.; GONÇALVES, L. G. Towards green data-centers: A comparison of x86 and arm architectures power efficiency. **Journal of Parallel and Distributed Computing**, Academic Press, Inc., Orlando, FL, USA, p. 1770–1780, 2012.

ASANOVIC, K. et al. A View of the Parallel Computing Landscape. **Communications of the ACM**, ACM, New York, NY, USA, v. 52, n. 10, p. 56–67, 2009. ISSN 0001-0782.

BAKHODA, A. et al. Analyzing cuda workloads using a detailed gpu simulator. In: INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS), 2009, Boston, MA, USA. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2009. p. 163–174.

BARKER, K. et al. Using performance modeling to design large-scale systems. **IEEE Computer**, v. 42, n. 11, p. 42–49, 2009.

BECKMAN, P. et al. On the road to exascale. **Scientific Computing World**, Europa Science Ltd., n. 116, p. 26–28, February–March 2011.

BLAKE, G.; DRESLINSKI, R.; MUDGE, T. A survey of multicore processors. **Signal Processing Magazine, IEEE**, IEEE Computer Society, v. 26, n. 6, p. 26–37, November 2009.

CASANOVA, H.; LEGRAND, A.; QUINSON, M. Simgrid: a generic framework for large-scale distributed experiments. In: INTERNATIONAL CONFERENCE ON COMPUTER MODELING AND SIMULATION, 2008, Cambridge, UK. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2008. p. 126–131.

CHE, S. et al. Rodinia: A benchmark suite for heterogeneous computing. In: WORKLOAD CHARACTERIZATION, 2009. IISWC 2009. IEEE INTERNATIONAL SYMPOSIUM ON, 2009, Austin, TX, USA. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2009. p. 44–54.

CHE, S. et al. A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads. In: WORKLOAD CHARACTERIZATION (IISWC), 2010 IEEE INTERNATIONAL SYMPOSIUM ON, 2010, Atlanta, GA, USA. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2010. p. 1–11.

COLLANGE, S. et al. Barra: A parallel functional simulator for gpgpu. In: INTERNATIONAL SYMPOSIUM ON MODELING, ANALYSIS SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS (MASCOTS), 2010, Miami Beach, FL, USA. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2010. p. 351–360.

CORRIGAN, A. et al. Running unstructured grid-based cfd solvers on modern graphics hardware. **International Journal for Numerical Methods in Fluids**, Wiley Online Library, v. 66, n. 2, p. 221–229, 2011.

DANALIS, A. et al. The scalable heterogeneous computing (shoc) benchmark suite. In: WORKSHOP ON GENERAL-PURPOSE COMPUTATION ON GRAPHICS PROCESSING UNITS, 2010, Pittsburgh, Pennsylvania, USA. **Proceedings...** New York, NY, USA: ACM, 2010. p. 63–74.

DONGARRA, J.; LUSZCZEK, P. Anatomy of a globally recursive embedded linpack benchmark. **High Performance Extreme Computing, IEEE**, IEEE Computer Society, Waltham, MA, USA, p. 26–37, 2012.

EDWARDS, C. The exascale challenge. **Engineering & Technology**, IET, v. 5, n. 18, p. 53–55, 2010.

FÜRLINGER, K.; KLAUSECKER, C.; KRANZLMÜLLER, D. Towards energy efficient parallel computing on consumer electronic devices. **Information and Communication on Technology for the Fight against Global Warming**, Springer, p. 1–9, 2011.

HO, H.; MITHRARATNE, K.; HUNTER, P. Numerical simulation of blood flow in an anatomically-accurate cerebral venous tree. **IEEE Transactions on Medical Imaging**, v. 32, n. 1, p. 85–91, January 2013.

HOEFLER, T. et al. Performance modeling for systematic performance tuning. In: STATE OF THE PRACTICE REPORTS, 2011, Seattle, Washington. **Proceedings...** New York, NY, USA: ACM, 2011. p. 6:1–6:12.

HONG, S.; KIM, H. An integrated gpu power and performance model. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2010, Saint-Malo, France. **Proceedings...** New York, NY, USA: ACM, 2010. p. 280–289.

JIAO, Y. et al. Power and performance characterization of computational kernels on the gpu. In: INTERNETIONAL CONFERENCE ON GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) & INT'L CONFERENCE ON CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM), 2010, Hangzhou, China. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2010. p. 221–228.

KERR, A.; DIAMOS, G.; YALAMANCHILI, S. Modeling gpu-cpu workloads and systems. In: WORKSHOP ON GENERAL-PURPOSE COMPUTATION ON GRAPHICS PROCESSING UNITS, 2010, New York, NY, USA. **Proceedings...** New York, NY, USA: ACM, 2010. p. 31–42.

KIRK, D. B.; HWU, W. mei W. **Programming Massively Parallel Processors: A Hands-on Approach**. [S.l.]: Morgan Kaufmann, 2010.

KRASNOPOLSKY, V.; FOX-RABINOVITZ, M.; BELOCHITSKI, A. Development of neural network convection parameterizations for numerical climate and weather prediction models using cloud resolving model simulations. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2010, Barcelona, Spain. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2010. p. 1–8.

LEE, V. W. et al. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 38, n. 3, p. 451–460, June 2010.

MARR, D. et al. Hyper-threading technology architecture and microarchitecture. **Intel Technology Journal**, v. 6, n. 1, p. 4–15, 2002.

MISTRY, P. et al. Valar: a benchmark suite to study the dynamic behavior of heterogeneous systems. In: WORKSHOP ON GENERAL PURPOSE PROCESSOR USING GRAPHICS

PROCESSING UNITS, 2013, Houston, Texas, USA. **Proceedings...** New York, NY, USA: ACM, 2013. p. 54–65.

MONT-BLANC. **Mont-Blanc Project Home Page**. [s.n.], 2012. Available from Internet: <http://www.montblanc-project.eu/>.

MONT-BLANC. **Mont-Blanc Project introduction**. [s.n.], 2012. Available from Internet: <http://www.montblanc-project.eu/introduction>.

MONT-BLANC. **Mont-Blanc Project objectives**. [s.n.], 2012. Available from Internet: <http://www.montblanc-project.eu/objectives>.

OU, Z. et al. Energy-and cost-efficiency analysis of arm-based clusters. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2012 12TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2012, Ottawa, Canada. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2012. p. 115–123.

PADOIN, E. L. et al. Evaluating the performance and energy of ARM processors for High Performance Computing. **41st International Conference on Parallel Processing (ICPP 2012) - 1st International Workshop on Unconventional Cluster Architectures and Applications (UCAA 2012)**, IEEE Computer Society, Pittsburgh, PA, USA, p. 1–8, 2012.

PADOIN, E. L. et al. Time-to-solution and energy-to-solution: A comparison between arm and xeon. **24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2012) - 3rd Workshop on Applications for Multi-Core Architectures (WAMCA 2012)**, New York, NY, USA, p. 1–6, 2012.

PADOIN, E. L. et al. Arm-based cluster: Performance, scalability and energy efficiency. **25th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2013) - 4rd Workshop on Applications for Multi-Core Architectures (WAMCA 2013)**, Porto de Galinhas, Brazil, 2013.

PAWLOWSKI, S. S. Exascale science: the next frontier in high performance computing. In: THE 24TH INTERNATIONAL CONFERENCE ON SUPERCOMPUTING (ICS), 2010, Tsukuba, Japan. **Proceedings...** New York, NY, USA: ACM, 2010. p. 1.

ROBERTS-HOFFMAN, K.; HEGDE, P. ARM cortex-a8 vs. intel atom: Architectural and benchmark comparisons. **Dallas: University of Texas at Dallas**, Dallas, TX, USA, 2009.

STANLEY-MARBELL, P.; CABEZAS, V. Performance, power, and thermal analysis of low-power processors for scale-out systems. In: INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING WORKSHOPS AND PHD FORUM (IPDPSW), 2011, Shanghai, China. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2011. p. 863–870.

STONE, J. E.; GOHARA, D.; SHI, G. Opencl: A parallel programming standard for heterogeneous computing systems. **Computing in science & engineering**, IEEE Computer Society, v. 12, n. 3, p. 66–73, 2010.

STRATTON, J. A. et al. Parboil: A revised benchmark suite for scientific and commercial throughput computing. **Center for Reliable and High-Performance Computing**, Urbana-Champaign, IL, USA, 2012.

VALERO, M. Towards exaflop supercomputers. **Conference Center of the University of Patras - High Perfomance Computing Academic Research Network (HPC-net)**, 2011.

VELHO, P. **Accurate and Fast Simulations of Large-Scale Distributed Computing Systems**. Thesis (PhD) — Université de Grenoble, Grenoble, France, 2011.

VELHO, P. et al. Fast and accurate models for gpu applications. **Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR 2013)**, San José, Costa Rica, 2013.

WANG, G.; REN, X. Power-efficient work distribution method for cpu-gpu heterogeneous system. In: INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING WITH APPLICATIONS, 2010, Taipei, Taiwan. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2010. p. 122–129.

WEHNER, M.; OLIKER, L.; SHALF, J. A real cloud computer. **Spectrum, IEEE**, IEEE Computer Society, v. 46, n. 10, p. 24–29, 2009.

YEUNG, G. et al. Low power memory implementation for a ghz+ dual core ARM cortex a9 processor on a high-k metal gate 32nm low power process. In: INTERNATIONAL SYMPOSIUM ON VLSI DESIGN, AUTOMATION AND TEST (VLSI-DAT), 2011, Hsinchu, Taiwan. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2011. p. 1–4.

YOUNGE, A. et al. Efficient resource management for cloud computing environments. In: INTERNATIONAL CONFERENCE ON GREEN COMPUTING, 2010, Chicago, IL, USA. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2010. p. 357–364.

ZELJKOVIC, V.; MOUSA, W. An algorithm for petro-graphic colour image segmentation used for oil exploration. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING AND SIMULATION (HPCS), 2011, Istanbul, Turkey. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2011. p. 498–503.

ZHANG, Y.; OWENS, J. A quantitative performance analysis model for gpu architectures. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTER ARCHITECTURE (HPCA), 2011, San Antonio, TX, USA. **Proceedings...** New York, NY, USA: IEEE Computer Society, 2011. p. 382–393.

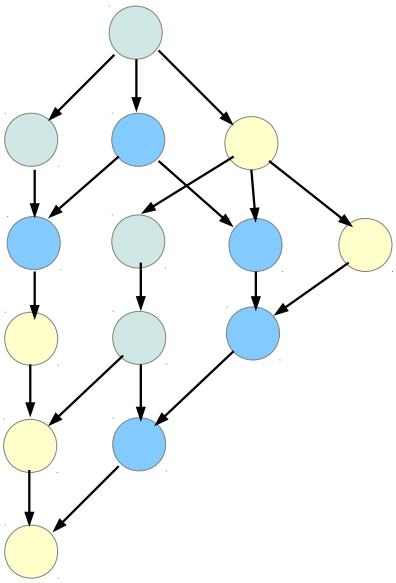
APPENDIX A GPU MODELING

The goal of this appendix is to show a way to model a GPU architecture in a way useful to study exascale systems. We will first analyze applications and the heterogeneous systems available today. Next, we will propose how to model GPU architectures. The methodology used will be detailed. Finally, we analyze the accuracy of the model.

A.1 Applications and Heterogeneous Systems

An application can be a composition of tasks with dependencies between them, each task can be organized in a Directed Acyclic Graph as seen in Figure A.1. For example, an application can execute the Cholesky factorization, matrix multiplication, Fast Fourier Transform, and other functions to calculate the solution. Each function executed by this application has its own characteristics, it could be processing bound, spending most of the time processing a small amount of data, it could be memory bound, depending on the memory to fetch data, it can have different patterns of data accesses and communication. All these characteristics make functions more suitable for one type of architecture. For instance, in the DAG of Figure A.1 we have three colors, each color could represent one type of function. One color can be the tasks suitable for GPUs, other color are the tasks that show great energy efficiency on low power processors, and the last one are tasks suitable for vector machines.

Figure A.1 – Directed Acyclic Graph.



Source: The Authors

A general purpose processor is not the most suitable architecture to achieve the energy efficiency needed for future Exascale systems. A heterogeneous system, where for every type of

task we have the most suitable architecture, would be very expensive, and the work distribution could become unmanageable. However, a heterogeneous systems consisting of few architectures that can sustain an overall high energy efficiency for most of the scientific applications can be the solution for exascale systems.

To study the aspect of performance improvement with GPUs, we propose to analytically model this architecture. The model makes it easy to evaluate large scale GPU architectures and their applications. Real large scale environments suffer from resource failures, long execution time of applications, interference from other applications, and other problems that the analytical model hides.

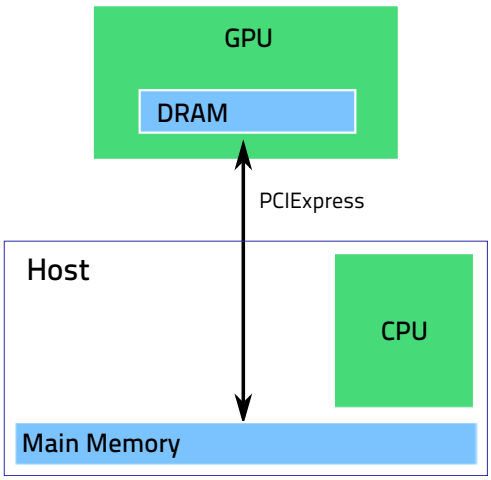
A.2 GPU Modeling

This section describes the programing model used by GPUs. It presents two frameworks, or libraries, used to create programs that can run on GPUs. Afterwards, we discuss how scientific applications are a composition of simpler functions, called kernels, that run on GPUs. Finally, we show a method to model these kernels.

A.2.1 Programming Model

GPU is a coprocessor that is accessed through a host CPU processor. Figure A.2 shows a host CPU processor that has direct access to the main memory with a GPU coprocessor. The GPU features its own internal DRAM memory.

Figure A.2 – The CPU is the host processor which has direct access to the main memory. The GPU is a coprocessor featuring its own internal DRAM memory.



Source: The Authors

To use a GPU for computational intensive algorithms, a specific programming library is necessary. The two main programming frameworks for GPU available today are NVIDIA Compute

Unified Device Architecture (CUDA), and Open Compute Language (OpenCL). Both models offer a set of tools, including compilers and libraries that enable GPU programming. CUDA is very fast, having the support of various libraries. As a drawback CUDA is proprietary and the compiler and some libraries work exclusively with NVIDIA hardware. OpenCL is an open standard with fully available source code. OpenCL targets GPU programming and also general multicore processors.

Independent of the programming framework, the GPU programming model in essence remains the same, independent of the software layer. The GPU as a coprocessor has its own internal memory. Then, to execute an application on the GPU, one needs to move the data and program from the main memory to the GPU internal memory. After the data and applications are loaded, the GPU can start executing. While the GPU is executing, the CPU may do some useful computation in parallel. Finally, when the GPU finishes the execution, the resulting data is available only in the GPU internal DRAM. Therefore, one needs to copy the result from the GPU internal memory back to the main memory to finish the computation. These three steps are the essence of the analytical model developed.

A.2.1.1 CUDA

CUDA is a state-of-the-art programming architecture and environment for NVIDIA GPUs. CUDA means both a SIMD (Single Instruction, Multiple Data) architecture and a programming model that extends languages (such as C) to use these GPUs. A process on the CPU runs a special function, called *kernel*, which executes on the GPU. All data has to be transferred to and from the GPU's memory, which incurs a communication overhead.

The CUDA programming model works with the abstraction of thousands of threads computing in parallel. The CUDA architecture also has a memory hierarchy. All memories inside the Streaming Multiprocessor (SM) have a small size and low latency. The global memory has a large size and a high latency (from 400 to 600 cycles) and can be accessed by all threads. The newer version of this architecture also contains a L1 cache per SM and a shared L2 cache.

A.2.1.2 OpenCL

OpenCL is an open standard for cross-platform parallel programing. With the heterogeneity of multiprocessors available today, OpenCL (STONE; GOHARA; SHI, 2010) provides a common programming model for parallel computation. One algorithm implemented in OpenCL can be compiled and executed on-the-fly on any capable OpenCL device, this approach enables to run the algorithm natively on platforms unavailable at the development time.

The programing interface provided by OpenCL can manage the devices available in the heterogeneous system, and memory allocation and transfers between these devices. OpenCL abstracts the hardware into compute units that contain one or more SIMD processing elements.

The memory hierarchy can be defined into four types, from a large memory with high latency to a small private memory with low latency.

A.2.2 Characterization and Modeling

In this section, we describe how we can characterize scientific applications, breaking them into simpler pieces of code called kernels that appear recurrently in different applications. Then we show one method to model these kernels running on GPUs.

A.2.2.1 Scientific Applications

Several applications of different areas share similar problems, they have some recurrent routines, called kernels. A considerable portion of execution time is spent in these kernels. Asanovic *et al.* (2009) recommend to characterize the execution time of HPC applications through the execution time of the many commonly found kernels. They point out that, besides the large number of scientific applications, only 13 dwarfs can characterize almost all of them. Dwarfs are kernels, algorithm pieces, that characterize data access and computation patterns.

To model a scientific application, we can break it into its kernels and model each kernel. In this way, we can divide the modeling of scientific applications into two parts. The first part is the modeling of the kernels that compose the applications. Lastly, we have to model the application as a set of kernels that are interacting. For example, the DAG in Figure A.1 would be the application model, and the tasks would be each kernel with its specific model that tell us how long this task takes to execute.

To model these kernels, we could use detailed simulation with a cycle-accurate or analytical model. As shown in (HOEFLER et al., 2011), analytical with empirically fitted models can be used to express real application runtime.

Analytical models are also very fast, taking only the time to compute some algebraic computations. This fast way to estimate the time is useful when one needs to simulate large scale applications. According to Hoefler *et al.* (2011), detailed simulation for large scale applications would take much more time, but will not create more accuracy.

To build models for the kernels, we propose the analytical approach. We will use benchmarks for several dwarfs as pointed out by the Berkeley report in (ASANOVIC et al., 2009), execute them on the GPU and extract the empirical data to build the models.

A.2.2.2 Kernel Modeling

As described before, a GPU is seen as a coprocessor by the CPU, featuring an independent internal memory. Therefore, to run a kernel on GPU we first need to transfer the data and the instructions. Afterwards, the kernel starts executing on the GPU. When the kernel finishes,

the result needs to be copied from the internal GPU memory to the main memory. Therefore, running a GPU kernel consists of the three steps described below.

1. **Dispatch**: Before we can run the kernel on the GPU, we need to make the data available in the GPU internal memory. The dispatch step consists of the time needed to transfer the data and programs to the GPU internal memory.
2. **Execution**: Once the data is available, the next step is to run the kernel on the GPU. Hence, the execution step is the time to run the kernel on the GPU.
3. **Collect**: When the GPU kernel finishes, the results from its execution is still in the GPU memory, one needs to copy the results from the GPU internal memory to the main memory. This collect step becomes part of the GPU total computing time.

Our model to estimate GPU computing time follows this approach of three steps. The total computation time is hence the sum of dispatch, execution, and collect times. While the execution time is estimated by the GPU speed, the time for dispatch and collect is estimated by the bandwidth of the PCIe bus. Since the dispatch and collect share the same PCIe bus, we thought that the same model could estimate both times. However, there is an asymmetry between the transmission speeds in both directions (KIRK; HWU, 2010).

To create the model, we assume that the time in each one of the steps is given by a linear function of workload. Therefore, the model is based on the hypotheses below:

(a) Dispatch time is a linear function of the input data ($W_{dispatch}$), i.e., the amount of data to copy from CPU memory to GPU memory, divided by the dispatch bandwidth, plus the error $\beta_{dispatch}$ of the linear regression;

$$T_{dispatch}(W_{dispatch}) = \frac{W_{dispatch}}{dispatchBandwidth} + \beta_{dispatch} \tag{A.1}$$

(b) Execution time is a linear function of workload (W_{exec}) with coefficient $\frac{1}{GPUSpeed}$, plus the error β_{exec} of the linear regression;

$$T_{exec}(W_{exec}) = \frac{W_{exec}}{GPUspeed} + \beta_{exec} \tag{A.2}$$

(c) Collect time is a linear function of the output data ($W_{collect}$), i.e., the amount of memory to copy from GPU to CPU when the kernel has finished, divided by the collect bandwidth, plus the error $\beta_{collect}$ of the linear regression;

$$T_{collect}(W_{collect}) = \frac{W_{collect}}{collectBandwidth} + \beta_{collect} \tag{A.3}$$

(d) Total time of computing the GPU kernel can be obtained by the sum of the time estimated in the three steps.

$$T_{GPU} = T_{dispatch} + T_{exec} + T_{collect} \tag{A.4}$$

A.3 Methodology

To create the models, we use 30 samples for random sizes of workloads. From these samples, one third (10 samples) were used to calibrate the model and infer the transmission rate of the PCIe bus and the speed rate of the GPU. The other 20 samples were used to evaluate the accuracy of the model.

A.3.1 Environment

The test platform was a computer with a quad-core Intel Core i7 930 operating at 2.80 GHz, the system features a x16 PCIe bus version 2.0 with a total capacity of 8 GB/s. The GPU card is an NVIDIA GTX 480 with 1536 MB of global memory and 480 cores, each core operates at 1.40GHz.

The operating system running was Ubuntu 11.04. The compiler was gcc in the version 4.4 and the CUDA was in the 4.0 version.

A.3.2 Kernels

To perform the modeling and tests, we chose three kernels commonly found in scientific applications.

- **Matrix Multiplication**: this kernel is commonly found in solvers for linear equation systems and algebraic applications.
- **Fast Fourier Transform**: also known as FFT, this kernel is useful in signal processing.
- **Needleman-Wunsch** : A dynamic programming kernel for sequence comparison commonly found in Bioinformatics.

A.3.3 Error Metric

In our experiments, we use the error metric described in Equation A.5 for the reasons shown in (VELHO, 2011). We use this metric because it respects the properties of symmetry and triangular inequality. This metric can also be easily transformed into a discrepancy percentage using Equation A.6.

$$\epsilon = \|ln(a) - ln(b)\| \tag{A.5}$$

$$\text{Discrepancy percentage} = \epsilon^{\text{Error}} - 1 \tag{A.6}$$

A.4 Results

We evaluate the quality of the models to verify in which conditions the models are accurate. For the sake of clarity, we group results by evaluating each one of the hypotheses.

A.4.1 Hypothesis a) Dispatch Time

Table A.1 presents the dispatch models for each application. The average transfer speed for all models is about 5.6 GB/s. This value is less than the nominal bandwidth of the PCIe bus 8 GB/s (for v2.x with 16 lane slot in each direction) but higher than half. This happens because the GPU is designed to stream video a fluid way, so the bus bandwidth is split to seamlessly transfer data in both direction simultaneously. This guarantees the fluidity of graphics rendering. So, the sum of dispatch and collect bandwidth should reach near the nominal bandwidth of 8 GB/s.

Table A.1 – Models for the three application to estimate the dispatch time of one GPU application. The bandwidth seems regular and close to the nominal bandwidth of the PCIe bus 8 GB/s (for version 2.x).

	$T_{dispatch}(W_{dispatch})$	$Max(\epsilon)$	$Mean(\epsilon)$
Matrix Multiplication	$\frac{W_{dispatch}}{5.68GB/s} + 0.000721$	20.66 %	0.65 %
FFT	$\frac{W_{dispatch}}{5.44GB/s} + 0.050160$	10.39 %	0.14 %
NeedleMan-Wunsch	$\frac{W_{dispatch}}{5.68GB/s} + 0.050476$	10.41 %	0.15 %

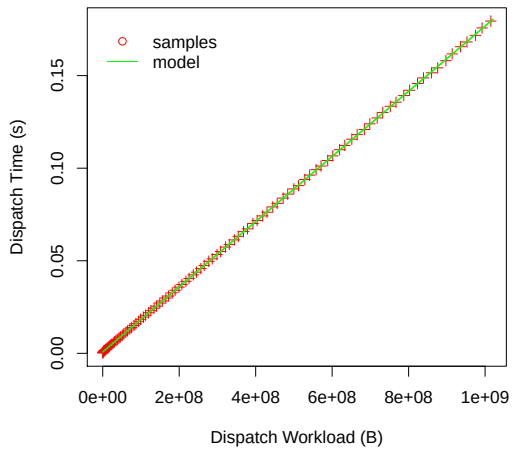
Source: The Authors

The next step now is to evaluate the error of the model and investigate which conditions make the model inaccurate. In Figures A.4(a), A.4(b) and A.4(c) we show the dispatch graph for Matrix Multiplication, FFT, and NeedleMan-Wunsch.

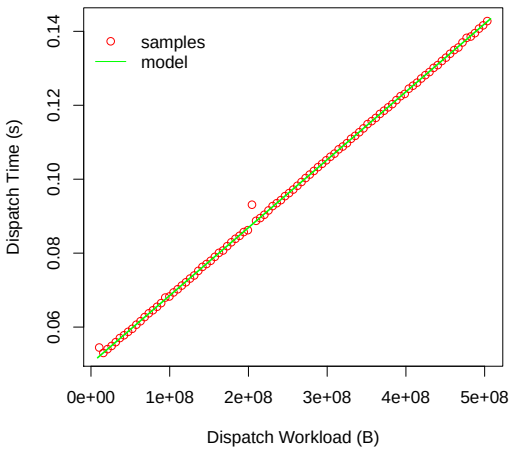
Visually, we can see that the linear regression fits well. Table A.1 presents the maximum error ($Max(\epsilon)$), a worst case error for all test samples, and the mean error ($Mean(\epsilon)$). Analyzing deeper the model for the Matrix Multiplication we have a mean error of 4.5%. However, looking the maximum error we got a maximum of 502 % discrepancy. This maximum error is because very small workloads, below 1MiB. In these cases the model is non-linear. Ignoring these small transmissions we can reduce the maximum error. In typical scientific application is reasonable to assume that the workload will be high (KIRK; HWU, 2010).

The maximum error obtained ignoring small workloads is about 20 %. The other benchmarks have larger workloads. Therefore, they presented a smaller error of about 10.41 %.

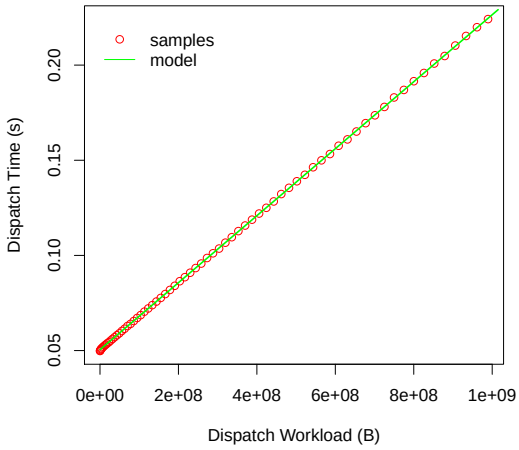
Figure A.3 – Dispatch time, in seconds, in function of the dispatch workload, in bytes with the three tested kernels: Matrix Multiplication, FFT, and NeedleMan-Wunsch. The model values, straight line, are shown aside with the test samples.



(a) Matrix Multiplication



(b) FFT



(c) NeedleMan-Wunsch

Source: The Authors

A.4.2 Hypothesis b) Execution Time

The model for the execution step is presented in Table A.2. For the execution time, the workload is described in giga floating points operations and the coefficient is the GPU processing speed (GFlops). The Figures A.5(a), A.5(b) and A.5(c) shows the models to estimate the execution time for Matrix Multiplication, FFT and NeedleMan-Wunsch. We can see that the linear regression model fits well, especially for the FFT that presents a very regular behavior. On the NeedleMan-Wunsch benchmark, the behavior is less regular but has a good fit of the

linear regression. In general, we see here a pronounced difference on the GPU speed achieved by each application. However, we still have to analyze the errors obtained using these models.

Table A.2 – Models for the three application to estimate the execution time of one GPU application. The computing speed highly depends on the computing kernel.

	$T_{exec}(W_{exec})$	$Max(\epsilon)$	$Mean(\epsilon)$
Matrix Multiplication	$\frac{W_{exec}}{241.10GFlops} + 0.001033$	16.19 %	1.11 %
FFT	$\frac{W_{exec}}{0.0004GFlops} + 0.0002$	4.60 %	0.11 %
NeedleMan-Wunsch	$\frac{W_{exec}}{1.15GFlops} + 0.0032$	201.26 %	8.47 %

Source: The Authors

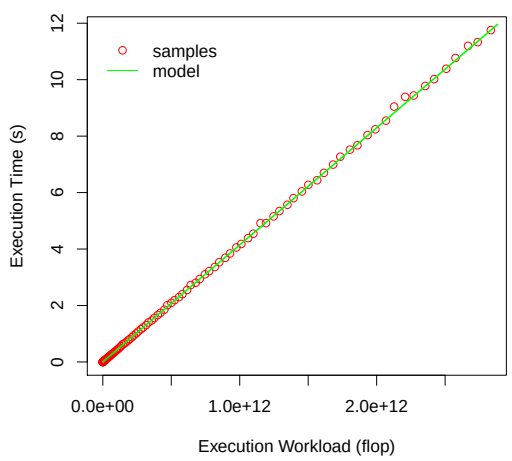
The biggest error was with NeedleMan-Wunsch, the maximum error for this benchmark was 6375 % while the mean error is 16 %. However, analyzing the workload and the speed performance, floating operations per seconds, we can notice that with some small workloads the GPU is unable to fully occupy its hardware. Removing these workloads, the maximum error found is 201 %. For Matrix Multiplication, the maximum error is 2692 % and the mean error is 11 %. Ignoring the small workloads the maximum error is reduced to 16 %. FFT has the best behavior of all, the maximum error is 13 % and without the small workloads is 5 %.

A.4.3 Hypothesis c) Collect Time

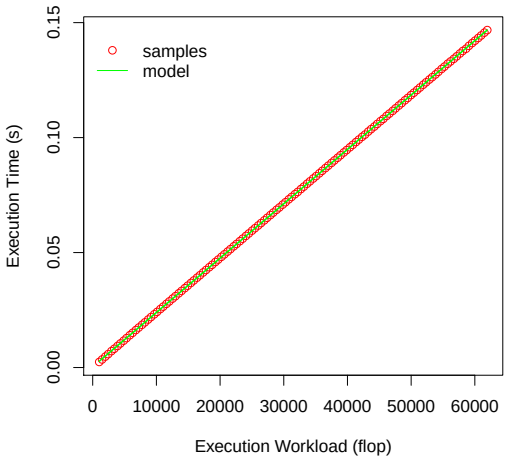
The model for the collect step appears in Table A.3. Analyzing the graphics on the Figures A.6(a), A.6(b) and A.6(c) we show, as previously, the collect model aside with the sampled values. All three applications present a regular behavior with bandwidth around 2.13 GB/s. The bandwidth for transferring data from GPU to CPU is smaller as expected. Now we can see that the sum of dispatch and collect bandwidth is close to the nominal bandwidth of 8 GB/s.

Evaluating the maximum error, NeedleMan-Wunsch is the worst case with 1249 %, Matrix Multiplication has 999 %. Nevertheless, the mean error for both of them is still acceptable 4 %. FFT presented a small maximum error of about 10 %. This happens because FFT data size is bigger. Ignoring small transfers, we can get a maximum error smaller than 11 % with Matrix Multiplication and FFT, although with NeedleMan-Wunsch the maximum error is nearly 53 %. Similar as in the dispatch, we verified the possibility of having one single model regardless of the application.

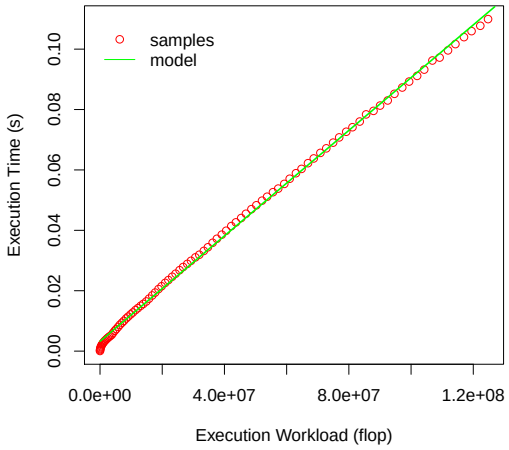
Figure A.4 – Execution time, in seconds, in function of the workload, in Flop with the three tested kernels: Matrix Multiplication, FFT, and NeedleMan-Wunsch. The model values, straight line, are shown aside with the test samples.



(a) Matrix Multiplication



(b) FFT



(c) NeedleMan-Wunsch

Source: The Authors

A.4.4 Hypothesis d) Total Execution Time

Our experiments indicate that the times for dispatch, execution and collect are well estimated using the linear model with good accuracy (with realistic input values). Even though we have good accuracy for each one of the three steps, we still assume that the sum of these three is a good approximation to the GPU overall computing time. This assumption is uncertain because the sum of the three may create compensation effects that might drastically increase the error. Therefore, we need to numerically evaluate if the sum of times accurately estimates the total

Table A.3 – Models for the three application to estimate the execution time of one GPU application. The computing speed highly depends on the computing kernel.

	$T_{collect}(W_{collect})$	$Max(\epsilon)$	$Mean(\epsilon)$
Matrix Multiplication	$\frac{W_{collect}}{2.13GB/s} + 0.000174$	10.61 %	0.28%
FFT	$W_{collect}2.14GB/s + 0.000163$	10.21 %	0.17 %
NeedleMan-Wunsch	$\frac{W_{collect}}{2.14GB/s} + 0.000175$	53.05 %	0.40 %

Source: The Authors

GPU computing time. To do this, we will observe the maximum error for the GPU overall computing time.

Here we use the sum of models presented in Tables A.1, A.2, and A.3. Equations (A.7), (A.8), and (A.9) show the final model. Here the input parameters of workload correctly specify a different workload depending on the application kernel.

- **Matrix Multiplication**

$$T^{MM}(W^{MM}_{dispatch}, W^{MM}_{exec}, W^{MM}_{collect}) = \\ \frac{W^{MM}_{dispatch}}{5.68GB/s} + \frac{W^{MM}_{exec}}{241.10GFlops} + \frac{W^{MM}_{collect}}{2.13GB/s} + 0.001928 \tag{A.7}$$

- **FFT**

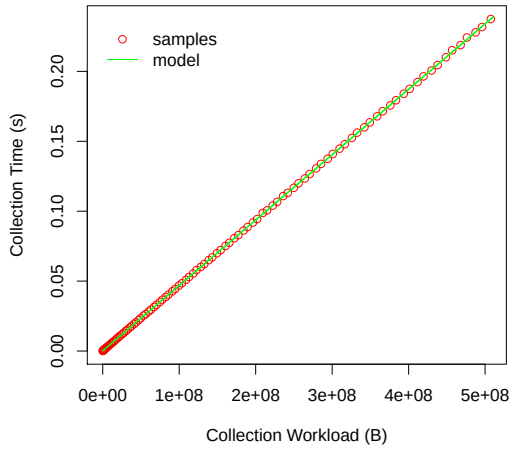
$$T^{FFT}(W^{FFT}_{dispatch}, W^{FFT}_{exec}, W^{FFT}_{collect}) = \\ \frac{W^{FFT}_{dispatch}}{5.44GB/s} + \frac{W^{FFT}_{exec}}{0.0004GFlops} + \frac{W^{FFT}_{collect}}{2.14GB/s} + 0.050523 \tag{A.8}$$

- **NeedleMan-Wunsch**

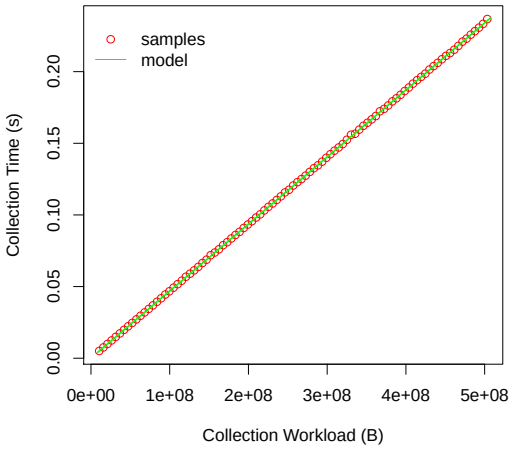
$$T^{NW}(W^{NW}_{dispatch}, W^{NW}_{exec}, W^{NW}_{collect}) = \\ \frac{W^{NW}_{dispatch}}{5.68GB/s} + \frac{W^{NW}_{exec}}{1.15GFlops} + \frac{W^{NW}_{collect}}{2.14GB/s} + 0.053851 \tag{A.9}$$

Evaluating the final model hypothesis we have a very good model accuracy. The maximum error was 1010 % with the Matrix Multiplication kernel, removing the small workloads the error found is reduced to less than 11 %. For FFT and NeedleMan-Wunsch, the maximum error for the sum was less than 8 %. Therefore, the final model has an outstanding accuracy.

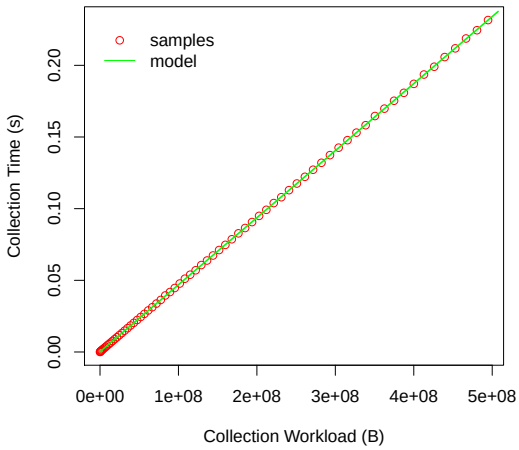
Figure A.5 – **Collect time**, in seconds, in function of the collect workload, in bytes with the three tested kernels: Matrix Multiplication, FFT, and NeedleMan-Wunsch. The model values, straight line, are shown aside with the test samples.



(a) Matrix Multiplication



(b) FFT



(c) NeedleMan-Wunsch

Source: The Authors

APPENDIX B SUMMARY IN PORTUGUESE

In this appendix, we present a summary of this master thesis in the portuguese lan- guage, as required by the PPGC Graduate Program in Computing.

Neste apêndice, é apresentado um resumo desta dissertação de mestrado na língua portuguesa, como requerido pelo Programa de Pós-Graduação em Computação.

B.1 Introdução

Aplicações científicas têm ajudado o desenvolvimento de várias áreas, como previsão do tempo (KRASNOPOLSKY; FOX-RABINOVITZ; BELOCHITSKI, 2010), prospecção de petróleo (ZELJKOVIC; MOUSA, 2011) e saúde (HO; MITHRARATNE; HUNTER, 2013), para citar alguns exemplos. Estas aplicações são demoradas, como por exemplo processamento intensivo, atingindo os limites da memória disponível e velocidade de processamento. Por esse motivo, a carga de trabalho dessas aplicações são muitas vezes uma parte da carga de trabalho desejada. Esta simplificação é feita principalmente para lidar com as limitações de hardware dos sistemas de alto desempenho atuais. Respondendo a essa demanda crescente de velocidade de processamento, que chegou recentemente a sistemas petascale, ou seja, sistemas capazes de processar 10^{15} operações de ponto flutuante por segundo. Num futuro próximo, cerca de 10 anos a partir de hoje, esperamos alcançar a era exascale onde sistemas de alto desempenho terão mil vezes mais velocidade de processamento do que os sistemas atuais (EDWARDS, 2010).

Projetar sistemas de alto desempenho para exascale traz vários desafios. Simplismente escalar a tecnologia atual é inviável porque somos incapazes de lidar com problemas como o alto consumo de energia, entre outros. Olhando para o passado, a comunidade de alto desempenho já enfrentou desafios para alcançar petascale. Naquela época, os processadores possuíam fraco suporte a multithread em comparação com arquiteturas multicore atuais que possuem vários núcleos de processamento em um único chip. Sistema de alto desempenho em petascale só se tornou viável através da utilização de unidades de processamento multicore (PAWLOWSKI, 2010).

Ao escalar a tecnologia atual, problemas como a comunicação e consumo de energia surgem. Tomando como exemplo os sistemas de alto desempenho na lista Top500 , ao escalar essas máquinas para exascale resultaria em máquinas que consomem Gigawatts de energia. Para fornecer essa quantidade de energia exigiria uma usina de energia nuclear de tamanho médio (WEHNER; OLIKER; SHALF, 2009). O relatório da DARPA (BECKMAN et al., 2011) estima um pico razoável de energia elétrica, eles dizem que a potência máxima dos próximos sistemas de alto desempenho deve estar abaixo de 20 Megawatts. Portanto, para alcançar a próxima escala de sistemas de alto desempenho, precisamos de alternativas que possam lidar com as restrições de consumo de energia (BARKER et al., 2009; YOUNGE et al., 2010).

A eficiência energética dos sistemas de alto desempenho futuros, respeitando o limite im-

posto pelo relatório DARPA, seria de 50 GFlops/W. Na lista Top500 , temos o sistemas de alto desempenho mais rápido, Titan, executando 17 petaflops a um custo de 8 Megawatts, a eficiência energética é de 2,1 GFlops/W. Portanto, temos de aumentar a eficiência energética em 25 vezes para atingir exascale.

Usar aceleradores é uma das alternativas para alcançar esta eficiência energética. Unidades de processamento gráfico (GPU) são os aceleradores comumente usados hoje. Diferente de CPUs, elas apresentam centenas de núcleos de processamento simples. GPUs foram projetadas para processar imagens e a indústria de jogos fez esta arquitetura prosperar. GPUs são bastante eficientes com algoritmos altamente paralelos executando milhares de threads ao mesmo tempo. Comparando com CPUs normais, é possível obter ganhos de até 10 vezes para alguns algoritmos executados na GPU (LEE et al., 2010). Outro tipo de acelerador é o Intel XeonPhi, que se baseia em vários núcleos x86. XeonPhi promete quase zero de esforço para executar algoritmos escritos para arquiteturas normais, com um desempenho semelhante ao das GPUs.

Processadores de baixa potência são uma outra abordagem para viabilizar sistemas exascale. Processadores ARM, diferente de CPUs tradicionais, concentraram-se em gastar o mínimo de energia possível. O foco principal desses processadores são dispositivos móveis e embarcados. Um dos focos desses dispositivos é fazer com que a bateria dure por mais tempo. No entanto, produtos populares apresentando processadores embarcados, tais como smartphones, precisam de mais desempenho, mudando o foco desta arquitetura. Portanto, processadores ARM se concentram no consumo de energia e desempenho, fazendo com que esta arquitetura seja uma boa alternativa para sistemas exascale de alto desempenho.

O projeto de Mont-Blanc é um esforço europeu para desenvolver um sistema exascale (MONT-BLANC, 2012a). Este projeto aposta em duas alternativas descritas acima. A idéia é usar processadores ARM e GPU em conjunto para obter uma máquina de alto desempenho com um baixo consumo de energia. Mont-Blanc espera construir um protótipo que pretende alcançar 7 GFlops/W até o final de 2014. Depois disso, o projeto prevê a construção de um sistema capaz de realizar 200 petaflops, consumindo um total de 10 Megawatts (MONT-BLANC, 2012b; MONT-BLANC, 2012c; VALERO, 2011).

B.2 Arquiteturas para alto desempenho

As arquiteturas apresentadas aqui serão utilizadas nesse trabalho. Considerando desempenho e energia, características serão apresentadas que fazem com que uma arquitetura seja mais apropriada para um determinado tipo de aplicação. Primeiro será apresentada uma arquitetura atualmente usada nos sistemas de alto desempenho. Depois, analisaremos uma arquitetura de baixo consumo, mais especificamente a arquitetura ARM. Em seguida, apresentamos a arquitetura de uma GPU, um acelerador usado inicialmente para jogos e processamento de imagens que chamou a atenção da comunidade de alto desempenho. Por fim, detalhamos uma arquitetura heterogênea que combina processadores de baixo desempenho e GPUs.

B.2.1 Processadores Intel

Processadores Intel são comumente encontrados em sistemas de alto desempenho. O objetivo dessa arquitetura tem sido somente desempenho por vários anos. Muitas melhorias foram adicionadas para incrementar o desempenho, isso levou a uma arquitetura de alto consumo que é capaz de prover um alto desempenho para a maioria dos algoritmos.

As melhorias adicionadas incluem multithreading simultâneo, que faz com que um processador físico seja visto como dois processadores virtuais. Outra melhoria foi adicionar cache com vários níveis, onde um processador possui de dois a três níveis de cache, com o objetivo de trazer os dados que o processador necessita de forma mais rápida. Extensões SIMD foram acrescentadas para que algoritmos possam executar instruções repetidas simultaneamente. Por fim, a Intel também adicionou a tecnologia Turbo Boost, na qual em vez de um processador economizar energia quando alguns núcleos de processamento estão desligados, o processador pode aumentar a frequência de um único núcleo aumentando o desempenho sequencial.

B.2.2 Processadores ARM

Processadores da família ARM Cortex foram desenvolvidos para prover alto desempenho por um baixo consumo de energia (YEUNG et al., 2011). Várias inovações foram feitas tanto para desempenho como para consumo de energia.

Com um foco inicial em sistemas embarcados, ARM desenvolveu arquiteturas de baixo consumo para durar horas ou dias com uma única bateria. Para isso, algumas melhorais foram feitas para economia de energia. Entre essas melhorias temos Micro TLBs e até mesmo a tecnologia multicore onde alguns núcleos são desligados quando não estão processando. Também foi introduzida a tecnologia big.LITTLE criando chips heterogêneos, adequando melhor o consumo e desempenho para cada aplicação.

Também foi incluída a arquitetura VFP, que é um coprocessador para operações de ponto flutuante com um alto desempenho e baixo consumo energético, melhorando assim tanto o consumo de energia como o desempenho. A tecnologia NEON foi introduzida na família Cortex para melhorar seu desempenho, NEON é uma unidade SIMD para executar operações de forma simultânea.

B.2.3 Aceleradores GPU

Unidades de processamento gráfico aparecem como um acelerador para prover um alto desempenho. Vários projetos de alto desempenho já estão usando GPUs para alcançar petaflops de desempenho. A arquitetura GPU foi inicialmente desenvolvida para a industria de jogos, levando ao desenvolvimento da arquitetura atual. Hoje em dia GPUs são amplamente usadas para acelerar diferentes aplicações científicas.

GPU é uma arquitetura SIMD, composta por centenas de núcleos de processamento mais simples, GPUs podem então executar centenas de threads simultaneamente. A memória da GPU é organizada em dois níveis, memória global e compartilhada. A memória global é lenta porém visível por todas as threads, a memória compartilhada é mais rápida mas visível apenas para um grupo de threads. Para a arquitetura Kepler da NVIDIA, a GPU possui ainda 2 níveis de cache. A Figura 3.2 mostra a hierarquia de memória da GPU Kepler.

B.2.4 ARM + GPU

Tegra é um processador desenvolvido pela NVIDIA voltado para computação móvel. Dispositivos como smartphones e tablets usam Tegra como seus processadores. Tegra é um System-on-Chip integrando ARM, GPU e outras capacidades como decodificador de áudio.

Os primeiros processadores Tegra foram lançados no ano de 2008 e 2009. Em 2010 NVIDIA lançou o Tegra 2, mostrado na Figura 3.5(a), composto de um ARM Cortex A9 dual-core e uma GPU GeForce de baixo consumo. Tegra 3, mostrado na Figura 3.5(b), foi anunciado em 2011 com um ARM Cortex A9 quad-core, como inovação foi incluído um quinto núcleo de processamento usando a tecnologia big.LITTLE para economia de energia.

A GPU nos processadores Tegra 2 e Tegra 3 não são capazes de de computação de propósito geral. Entretanto, Tegra 4, anunciado em 2013 serâ composto de um ARM Cortex A15 e uma GPU com 72 núcleos onde é possível utilizâ-la para comutação de propósito geral.

A família Samsung Exynos de processadores são semelhantes ao Tegra. Seu útimo lançamento, Exynos 5 Dual, é composto de um ARM Cortex A15 e uma GPU ARM Mali-T604 que pode ser utilizada para computação de propósito geral.

B.3 Avaliação de consumo de energia e desempenho

Para alcançar o próximo passo na evolução de sistemas de alto desempenho, precisamos construir sistemas com arquiteturas que possam oferecer uma boa relação entre desempenho e consumo de energia. As aplicações executadas nessas arquiteturas devem ser executadas rapidamente, enquanto o consumo de energia deve ser o menor possível. Portanto, a eficiência energética de tais sistemas deve ser alta.

Considerando este desafio da eficiência energética, esta seção se propõe a avaliar as arquiteturas apresentadas anteriormente, analisando o consumo energético e o desempenho de tais arquiteturas.

B.3.1 Proposta

Os melhores sistemas de alto desempeho na lista Top500 atingem 2,1 GFlops/W, para atender os requisitos propostos pelo relatório DARPA, esses sistemas precisam melhorar a eficiência

energética em 25 vezes. Duas alternativas para atingir a eficiência energética são os aceleradores e os processadores de baixo consumo. Além disso, chips que incluem tanto processadores de baixa consumo como aceleradores, podem conseguir uma eficiência energética ainda melhor.

Para avaliar as possíveis arquiteturas, selecionamos três plataformas para representar processadores de alta potência, aceleradores e processadores de baixo consumo. Vamos avaliar as arquiteturas em termos de tempo para solucação e energia para solução.

Para representar os processadores de alta potência, selecionamos um processador Intel Xeon E5. Aceleradores seram representados por uma GPU NVIDIA Tesla K20. Finalmente, optamos por um ARM Cortex A9 como o processador de baixo consumo de energia.

B.3.2 Metodologia

Para representar os processadores de alta potência, um processador Intel Xeon E5 será usado e a partir daqui será chamado de Intel Xeon. Os aceleradores mais comuns na lista Top500 são as GPUs NVIDIA. Portanto, será usada uma GPU NVIDIA Tesla que será chamada de Tesla K20. O processador de baixo consumo de energia mais popular é o ARM, presente na maioria dos mais recentes smartphones hoje em dia, ARM Cortex A9 será o processador usado nos testes e chamado de PandaBoard A9.

O processadores testados durante essa dissertação estão descritos em detalhes na Tabela 4.1.

Algoritmos do *Rodinia Benchmark Suite* serão utilizados para comparar essas arquiteturas. Esses algoritmos foram implementados de forma eficiente para as três arquiteturas usadas nessa dissertação.

B.3.2.1 Algoritmos testados

Rodínia (CHE et al., 2009) é uma seleção de algoritmos para computação heterogênea desenvolvido na Universidade da Virgínia. O objectivo é ajudar arquitetos de computadores a estudarem plataformas, como GPUs e CPUs multicore. Portanto, os algoritmos têm versões em OpenMP, CUDA e OpenCL. Três algoritmos foram selecionados.

- CFD é um algoritmos de dinâmica dos fluidos, ele representa uma classe de algoritmos conhecida como grade não estruturada.
- HotSpot é uma ferramenta amplamente utilizada para estimar a temperatura do processador. A simulação térmica de forma iterativa resolve uma série de equações diferenciais. Ele representa a classe de algoritmos conhecida como grade estruturada.
- Needleman-Wunsch é um método de otimização global não-linear para alinhamentos de sequências de DNA. A classe de algoritmos que ele representa é a de programação dinâmica.

B.3.3 Resultados

Este capítulo mostra os resultados obtidos para cada algoritmo testado. Executamos cada algoritmo pelo menos 30 vezes e calculamos o intervalo de confiança para um nível de confiança de 95%.

B.3.3.1 CFD

Algoritmos baseados em grade não estruturada, como o CFD, são um desafio para GPUs, a dependência dos dados e acesso irregular a memória desses algoritmos torna-se um grande gargalo. A arquitetura de GPUs foi construída com foco em processamento de imagens que usam grades estruturadas, sem dependência de dados e acessos regulares a memória.

A implementação do CFD para GPUs usado pelo Rodínia é descrito em (CORRIGAN et al., 2011), eles usaram várias técnicas para otimizar algoritmos de grade não estruturada para GPUs modernas.

Duas cargas de trabalho são testadas sob fluxo supersônico. A primeira é NACA0012, uma asa de avião. A segunda é um míssil. Figuras 5.1 e 5.2 mostram os objetos e a pressão na superfície deles sob fluxo supersônico.

O tempo para solução de cada arquitetura é apresentado na Tabela 5.1. Embora grades não estruturadas não são a melhor opção para GPUs, Tesla K20 obteve o melhor tempo. Figura 5.3 mostra o tempo para solução. Tesla K20 foi de 7 a 8 vezes mais rápida do que Intel Xeon e cerca de 600 vezes mais rápida do que PandaBoard A9. Intel Xeon foi cerca de 80 vezes mais rápida do que PandaBoard A9.

A Tabela 5.2 mostra a potência média em watts para cada arquitetura e a Tabela 5.3 mostra a energia para solução em joules. Para a arquitetura Intel Xeon temos duas medidas, a primeira é a energia gasta pela placa mãe, e a segunda é a energia gasta apenas pelos processadores.

Figura 5.4 mostra a energia para solução. Tesla K20 também foi a arquitetura que apresentou o melhor consumo de energia, Tesla K20 consumiu 18 vezes menos energia do que Intel Xeon, e cerca de 5,5 menos energia considerando apenas os processadores Intel Xeon. Tesla K20 também consomiu de 52 a 59 vezes menos energia do que PandaBoard A9.

Intel Xeon consumiu menos energia do que PandaBoard A9. No entanto, esse consumo foi de apenas 3 vezes menor comparando com um tempo para solução 80 vezes mais rápido. Observamos também que ambos os processadores foram responsáveis por cerca de 30% do consumo medido para a placa principal.

B.3.3.2 Hotspot

Hotspot é um algoritmo de grade estruturada e melhor adequado para a arquitetura de GPUs. A entrada é uma matriz que representa o processador simulado, três cargas de trabalho com

dimensões de 64, 512 e 1024 foram testadas.

A Tabela 5.4 mostra o tempo para solução de cada arquitetura. Como esperado, Tesla K20 foi a mais rápida das arquiteturas. O tempo para solução também é mostrado na Figura 5.5. Tesla K20 foi de 12 a 15 vezes mais rápida que Intel Xeon e de 94 a 739 vezes mais rápida que PandaBoard A9. Intel Xeon foi de 7 a 49 vezes mais rápido do que PandaBoard A9.

A potência média para este algoritmo é apresentado na Tabela 5.5. Na Tabela 5.6 e na Figura 5.6, podemos ver a energia para solução de cada arquitetura. Tesla K20 também apresentou o melhor consumo de energia com 37 a 52 menor consumo de energia do que Intel Xeon, e 11 a 17 menos energia considerando apenas os processadores Intel Xeon. Tesla K20 consumiu de 32 a 128 menos energia do que PandaBoard A9.

Da mesma forma que o algoritmo CFD, processadores Intel Xeon foram responsáveis por cerca de 30% do consumo medido para a placa mãe. No entanto, comparando-se contra PandaBoard A9, Intel Xeon consumiu 1,47 mais energia para a carga de trabalho de 64 e consumiu 2 vezes menos energia para as outras duas cargas de trabalho. Outros trabalhos já demonstraram que com pequenas cargas de trabalho arquiteturas ARM podem apresentar uma maior eficiência energética do que outras arquiteturas.

B.3.3.3 Needleman-Wunsch

Neddleman-Wunsch é um algoritmo de programação dinâmica e por causa da dependência dos dados pode apresentar um fraco desempenho em arquiteturas GPUs. A carga de trabalho é representada por uma matriz quadrada com dimensões que variam de 1024 até 4096. No entanto, PandaBoard A9 só pode executar até a carga de trabalho com dimensões da matriz de 2560, uma vez que PandaBoard A9 não possui memória suficiente para alocar cargas de trabalho maiores.

O tempo para solução de cada arquitetura são mostrados na Tabela 5.7 e na Figura 5.7. Tesla K20 foi novamente a arquitetura mais rápida, Tesla K20 foi de 6 a 9 vezes mais rápida do que Intel Xeon, e 96 a 218 mais rápida do que PandaBoard A9. Intel Xeon foi 16 a 37 vezes mais rápido do que PandaBoard A9.

A Tabela 5.8 mostra a potência média para este algoritmo. Na Tabela 5.9 e Figura 5.8 podemos ver os resultados de energia para solução. Tesla K20 consumiu 25 a 37 vezes menos energia do que Intel Xeon e 8 a 12 vezes menos energia considerando apenas os processadores Intel Xeon, Tesla K20 também consumiu 27 a 46 vezes menos energia do que PandaBoard A9. Comparando Intel Xeon contra PandaBoard A9, podemos ver mais uma vez que com pequenas cargas de trabalho arquiteturas ARM podem ser melhores ou chegar perto de arquiteturas comum, Intel Xeon consumiu 1,04 a 1,83 vezes menos energia do que PandaBoard A9.

B.4 Conclusão

Este trabalho apresentou uma avaliação das arquiteturas de alto desempenho, analisando tempo e energia para solução considerando a eficiência energética de cada arquitetura. três algoritmos distintos foram executados em todas as arquiteturas para avaliar qual arquitetura é a mais rápida e qual consome o mínimo de energia.

Os resultados mostraram que a arquitetura GPU, para as aplicações de semelhantes as três testadas, obteve o maior desempenho, pelo menos 5 vezes mais rápida do que Intel e 80 vezes mais rápido do que ARM. A GPU também consumiu 8 vezes menos energia do que as outras.

Para cargas de trabalho leves, a arquitetura ARM mostrou uma boa eficiência energética superando Intel. Portanto, para tarefas com carga de trabalho leve, que não necessitam de uma restrição de tempo para ser concluída, arquiteturas de baixa potência como ARM é a melhor escolha em termos de consumo de energia. No entanto, para cargas de trabalho mais pesadas, arquiteturas de alta potência podem ser dezenas de vezes mais rápida, proporcionando uma eficiência energética igual ou melhor do que as arquiteturas de baixa potência.