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

MARCO ANTONIO ZANATA ALVES

Avaliação do Compartilhamento das Memórias *Cache* no Desempenho de Arquiteturas *Multi-Core*

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Philippe O. A. Navaux
Orientador

Porto Alegre, Março de 2009

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Alves, Marco Antonio Zanata

Avaliação do Compartilhamento das Memórias *Cache* no Desempenho de Arquiteturas *Multi-Core* / Marco Antonio Zanata Alves. – Porto Alegre: PPGC da UFRGS, 2009.

175 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Orientador: Philippe O. A. Navaux.

1. Memória cache. 2. Processador multi-core. 3. Arquitetura de computadores. 4. Processamento de alto desempenho. I. Navaux, Philippe O. A.. II. Título.

“ No quarto Câmbio Abstratoso. Tudo existirá.
Nada estará em contínuo fluxo. Nada correrá. O ser.
A verdade. E o valor. ”
— PRIMEIRA CARTA DE PEDRA. DE JOÃO PEDRA MAIOR.
JOSÉ PAES LIRA – LIROVSKY

AGRADECIMENTOS

Serei eternamente grato à amável Grasciele Fabiana Casagrande Centenaro, meus pais Arlindo Alves e Lizeth Zanata, minha madrasta e amiga Harolda Zanetti e aos demais familiares, tios e avós, por todo amor, compreensão nesses tempos de distanciamento e pelo total apoio nas etapas desse trabalho. Grasciele também ajudou com edição e correção deste e outros trabalhos, sempre com total afinco e apoio.

Inúmeros amigos e colegas ajudaram com leitura, correções e idéias, aumentando a precisão e abrangência do trabalho. Agradeço ao professor e orientador sempre atencioso Philippe Olivier Alexandre Navaux, por suas sabias visões e experiências de vida. Aos colegas do GPPD e UFRGS em especial aos que tiveram contato direto com a dissertação, Henrique Cota de Freitas, Eduardo Rocha Rodrigues, Felipe Lopes Madruga, Roberto Pinto Souto, Rodrigo da Rosa Righi, além dos professores, Nicolas Bruno Maillard, Flávio Rech Wagner, Luigi Carro e Fernanda Lima.

Muitos que ajudaram com palavras de apoio, Marcelo Augusto Zanata Alves, Marcelo Corrêa Yamashita, Marcos Hiroshi Umino, Carlos Farias Fernandes, Eduardo Lino Vieira e outros grandes amigos que mesmo longe fisicamente, estão sempre perto, no coração.

Não menos importante, os professores da FCT-UNESP de Presidente Prudente, os quais me ajudaram no início da caminhada acadêmica, Klaus Schlünzen Junior, Elisa Tomoe Moriya Schlünzen, Erwin Doescher, Milton Hirokazu Shimabukuro e tantos outros amigos do NEC.

A todos amigos e companheiros, um forte abraço e sincero obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ADI	*Alternating Direction Implicit*
AMD	*Advanced Micro Devices, Inc.*
BMT	*Block Multithreading*
CFD	*Computational Fluid Dynamics*
CMP	*Chip Multiprocessor*
CMT	*Chip Multithreading*
COV	*Coefficient of Variation*
DoE	*Design of Experiments*
DRAM	*Dynamic Random Access Memory*
EPIC	*Explicit Parallel Instruction Computing*
FFT	*Fast Fourier Transform*
FIFO	*Fist In, First Out*
GEMS	*General Execution Model Simulator*
GPP	*General Purpose Processor*
IA32	*Intel Architecture 32 bits*
IBM	*International Business Machines Corporation*
IMT	*Interleaved Multithreading*
IPF	*Itanium Processor Family*
L1	*Level 1*
L2	*Level 2*
L3	*Level 3*
LFU	*Least Frequently Used*
LRU	*Least Recently Used*
MFLOPS	Milhares de Operações de Ponto Flutuante por Segundo
MIPS	Milhares de Instruções por Segundo
mm^2	Milímetros Quadrado

MMU *Memory Management Unit*

MPKI *Misses per Kilo Instructions*

MPSoC *Multiprocessor System-on-Chip*

NAS *Numerical Aerodynamic Simulation*

nm Nanômetros

NoC *Network-on-Chip*

NPB *NAS Parallel Benchmark*

ns Nanosegundos

NUCA *Non-Uniform Cache Architecture*

OMP *Open Multi-Processing*

OOO *Out Of Order*

OpenMP *Open Multi-Processing*

OSE *Operating System Embedded*

PAD Processamento de Alto Desempenho

PARSEC *Princeton Application Repository for Shared-Memory* Computers

RTC *Real Time Control*

RTEMS *Real-Time Executive for Multiprocessor Systems*

SMT *Simultaneous Multithreading*

SPEC *Standard Performance Evaluation Corporation*

SPLASH *Stanford Parallel Applications for Shared-Memory*

SRAM *Static Random Access Memory*

SSOR *Symmetric Successive Over-Relaxation*

TLB *Translation Look-Aside Buffer*

TLP *Thread Level Parallelism*

TPC *Transaction Processing Performance Council*

UCA *Uniform Cache Architecture*

VLIW *Very Long Instruction Word*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

No atual contexto de inovações em *multi-core*, em que as novas tecnologias de integração estão fornecendo um número crescente de transistores por *chip*, o estudo de técnicas de aumento de vazão de dados é de suma importância para os atuais e futuros processadores *multi-core* e *many-core*.

Com a contínua demanda por desempenho computacional, as memórias *cache* vêm sendo largamente adotadas nos diversos tipos de projetos arquiteturais de computadores. Os atuais processadores disponíveis no mercado apontam na direção do uso de memórias *cache* L2 compartilhadas. No entanto, ainda não está claro quais os ganhos e custos inerentes desses modelos de compartilhamento da memória *cache*. Assim, nota-se a importância de estudos que abordem os diversos aspectos do compartilhamento de memória *cache* em processadores com múltiplos núcleos.

Portanto, essa dissertação visa avaliar diferentes compartilhamentos de memória *cache*, modelando e aplicando cargas de trabalho sobre as diferentes organizações, a fim de obter resultados significativos sobre o desempenho e a influência do compartilhamento da memória *cache* em processadores *multi-core*.

Para isso, foram avaliados diversos compartilhamentos de memória *cache*, utilizando técnicas tradicionais de aumento de desempenho, como aumento da associatividade, maior tamanho de linha, maior tamanho de memória *cache* e também aumento no número de níveis de memória *cache*, investigando a correlação entre essas arquiteturas de memória *cache* e os diversos tipos de aplicações da carga de trabalho.

Os resultados mostram a importância da integração entre os projetos de arquitetura de memória *cache* e o projeto físico da memória, a fim de obter o melhor equilíbrio entre tempo de acesso à memória *cache* e redução de faltas de dados. Nota-se nos resultados, dentro do espaço de projeto avaliado, que devido às limitações físicas e de desempenho, as organizações 1Core/L2 e 2Cores/L2, com tamanho total igual a 32 MB (bancos de 2 MB compartilhados), tamanho de linha igual a 128 *bytes*, representam uma boa escolha de implementação física em sistemas de propósito geral, obtendo um bom desempenho em todas aplicações avaliadas sem grandes sobrecustos de ocupação de área e consumo de energia.

Além disso, como conclusão desta dissertação, mostra-se que, para as atuais e futuras tecnologias de integração, as tradicionais técnicas de ganho de desempenho obtidas com modificações na memória *cache*, como aumento do tamanho das memórias, incremento da associatividade, maiores tamanhos da linha, etc. não devem apresentar ganhos reais de desempenho caso o acréscimo de latência gerado por essas técnicas não seja reduzido, a fim de equilibrar entre a redução na taxa de faltas de dados e o tempo de acesso aos dados.

Palavras-chave: Memória cache, processador multi-core, arquitetura de computadores, processamento de alto desempenho.

Performance Evaluation of Shared Cache Memory for Multi-Core Architectures

ABSTRACT

In the current context of innovations in multi-core processors, where the new integration technologies are providing an increasing number of transistors inside chip, the study of techniques for increasing data throughput has great importance for the current and future multi-core and many-core processors.

With the continuous demand for performance, the cache memories have been widely adopted in various types of architectural designs of computers. Nowadays, processors on the market point out for the use of shared L2 cache memory. However, it is not clear the gains and costs of these shared cache memory models. Thus, studies that address different aspects of shared cache memory have great importance in context of multi-core processors.

Therefore, this dissertation aims to evaluate different shared cache memory, modeling and applying workloads on different organizations in order to obtain significant results from the performance and the influence of the shared cache memory multi-core processors.

Thus, several types of shared cache memory were evaluated using traditional techniques to increase performance, such as increasing the associativity, larger line size, larger cache memory and also the increase on the cache memory hierarchy, investigating the correlation between the cache memory architecture and the workload applications.

The results show the importance of integration between cache memory architecture project and memory physical design in order to obtain the best trade-off between cache memory access time and cache misses. According to the results, within evaluations, due to physical limitations and performance, organizations 1Core/L2 and 2Cores/L2 with total cache size equal to 32MB, using banks of 2 MB, line size equal to 128 bytes, represent a good choice for physical implementation in general purpose systems, obtaining a good performance in all evaluated applications without major extra costs of area occupation and power consumption.

Furthermore, as a conclusion in this dissertation is shown that, for current and future integration technologies, traditional techniques for performance gain obtained with changes in the cache memory such as, increase of the memory size, increasing the associativity, larger line sizes etc.. should not lead to real performance gains if the additional latency generated by these techniques was not treated, in order to balance between the reduction of cache miss rate and the data access time.

Keywords: Cache memory, multi-core processor, computer architecture, high performance computing.

1 INTRODUÇÃO

“ Would you tell me, please, which way I ought to go from here ? ”
“ That depends a good deal on where you want to get to. ”, said the Cat.
“ I don’t much care where. ”, said Alice.
“ Then it doesn’t matter which way you go. ”, said the Cat.
— ALICE’S ADVENTURES IN WONDERLAND.
LEWIS CARROLL

A busca por desempenho computacional tem aumentado ao longo das décadas, onde diversas técnicas arquiteturais são utilizadas como *pipeline*, superescalaridade e *multithreading* para explorar o paralelismo de execução das aplicações e assim reduzir o tempo de execução em computadores pessoais ou servidores.

O suporte a múltiplas *threads* (UNGERER; ROBIC; SILC, 2003) (UNGERER; ROBIC; SILC, 2002) é uma alternativa de exploração de paralelismo não mais no nível de instruções, mas no nível de fluxo de instruções (*threads*), gerando assim, um aumento na vazão de processamento de *threads*.

Assim, as abordagens *single-threading* tradicionais vêm dando lugar a abordagens diferentes a fim de aumentar ainda mais o desempenho, e ainda algumas vezes diminuir a potência dissipada. O uso de processadores com técnicas *multithreading* e múltiplos núcleos de processamento vem sendo consolidado como um bom método para aumento o do desempenho de computação.

As memórias *cache* para processadores *multi-core* devem acompanhar o ritmo de inovações onde os novos projetos devem planejar o acesso a dados de múltiplas *threads* ao invés de apenas uma *thread* como acontecia nos processadores superescalares. Assim, para garantir a alta vazão de dados para os processadores com múltiplos núcleos de processamento, é necessário que a arquitetura de memória *cache* dê suporte ao fluxo de dados, onde a organização da memória *cache* deva garantir o menor número de faltas de dados, curto tempo de acesso aos dados nas leituras e escritas, e em caso de falta de dados, estes devem ser buscados e disponibilizados rapidamente, evitando assim que os processadores enfrentem longos períodos de latência por espera de dados.

1.1 Problemas

Uma vez que as memórias *cache* impactam no desempenho do sistema final de processadores *multi-core*, a escolha pela melhor arquitetura de memória *cache* em arquiteturas com vários núcleos de processamento é um grande desafio aos projetistas de processador.

Os principais problemas a serem enfrentados durante o desenvolvimento do projeto das memórias *cache* para *multi-core* são as latências de acesso, as contenções pelo limitado número de portas e a organização de compartilhamento de memória entre núcleos de processamento. Esses três fatores estão intimamente ligados, uma vez que grandes blocos de memória *cache* demandam grande tempo de acesso aos dados, porém, esses grandes blocos compartilhados entre diversos núcleos podem trazer grande vantagem em relação ao compartilhamento de dados. Em contrapartida, as latências desses grandes blocos de memória *cache* levam o sistema a perder muitos ciclos por espera de dados a cada busca de informação, o que pode não compensar do ponto de vista de desempenho computacional. Além desses dois fatores, deve-se considerar ainda o balanço entre compartilhamento de memória *cache* e a quantidade de portas de acesso a dados por banco de memória *cache*, pois ambos influenciam na latência de acesso e a área ocupada pelo sistema de memória.

Assim, o principal tema desta dissertação está relacionado ao desempenho influenciado pela latência de acesso e o compartilhamento da memória *cache*, o qual gera contenção para acesso a memória *cache*. Além desses, pode-se considerar ainda outras formas de redução de faltas de dados na memória *cache*, como aumento de associatividade, aumento do tamanho da linha de dados, aumento do número de níveis da hierarquia de memória, entre outros. Esses são os principais problemas abordados, porém, outros problemas relacionados às formas de avaliação de memórias *cache* em *multi-core* também são apresentados e tratados ao longo do trabalho.

1.2 Motivações

Com o surgimento dos processadores *multi-core*, algumas pesquisas e processadores comerciais adotaram o compartilhamento do segundo nível da memória *cache* como uma alternativa para aumentar o desempenho de aplicações paralelas utilizando o modelo de programação por memória compartilhada.

Porém, com as atuais inovações tecnológicas, onde cada vez mais indústrias estão passando a utilizar processadores com múltiplos núcleos de processamento, a hierarquia de memória *cache* a ser adotada para esses processadores multiprocessados ainda é um grande desafio, assim como o modelo de compartilhamento dessas memórias entre os núcleos. Esses questionamentos são os pontos chaves a serem discutidos e estudados neste trabalho.

Atualmente, os processadores do estado da arte (SINHAROY et al., 2005) (KONGETIRA; AINGARAN; OLUKOTUN, 2005) (MCNAIRY; BHATIA, 2005) (CHANG; MEMBER; HUANG, 2007) (PENG et al., 2007), utilizam no primeiro nível de memória *cache*, a organização de uma memória *cache* por processador, sendo a memória *cache* separada para dados e instruções. Já no segundo nível de memória *cache*, alguns projetos utilizam o compartilhamento de uma memória *cache* para dois processadores, ou a *cache* isolada para cada processador, e em alguns casos, é utilizado um terceiro nível na hierarquia de memória *cache*. Desta maneira, nota-se que a hierarquia de memória *cache* a ser adotada em processadores multiprocessados ainda é uma incógnita, assim como o modelo de compartilhamento dessas memórias entre os núcleos. Demonstrando a importância de trabalhos que avaliem mais a fundo a relação entre processadores multiprocessados e organização de memórias *cache*.

1.3 Objetivos

Dados os problemas em relação à memória *cache* para *multi-core*, foram definidos os principais objetivos deste trabalho que são apresentados abaixo.

Avaliação de diferentes compartilhamentos: Avaliar os diferentes compartilhamentos de memória *cache*, modelando e aplicando cargas de trabalho sobre as diferentes organizações, a fim de obter medições significativas sobre as influências dos diferentes compartilhamentos de *cache* entre os núcleos no comportamento do desempenho do sistema.

Avaliação de formas de aumento de desempenho: Avaliar dentre os diversos compartilhamentos de memória *cache* possíveis, outras formas de aumentar os ganhos de desempenho utilizando maior associatividade, maior tamanho de linha, maior tamanho de memória *cache* e também aumento no número de níveis de memória *cache*.

Investigação sobre correlação entre memória *cache* e carga de trabalho: Estudar as cargas de trabalho e analisar os resultados prévios obtidos além de determinar a correlação das diferentes cargas de trabalho com as arquiteturas testadas.

Análise da influência da contenção gerada pelo limitado número de portas: Analisar as possíveis perdas de desempenho e conseqüências pela limitação do número de portas da memória *cache* compartilhada.

1.4 Organização do Texto

No capítulo 2, é apresentado um levantamento teórico e o estado da arte sobre pesquisas acerca do impacto de memória *cache* em processadores *multi-core*, mostrando os principais problemas relacionados às arquiteturas de memória *cache* para os futuros processadores com vários núcleos de processamento.

No capítulo 3, após o levantamento inicial, é apresentada a metodologia para avaliação de impactos de memórias *cache* em *chip multi-core*, descrevendo técnicas de experimento, cargas de trabalho, metodologias de avaliação de desempenho, proposta e modelagem dos experimentos, entre outros métodos necessários para desenvolver esta pesquisa.

No capítulo 4, são apresentados os resultados e análise de impacto das memórias *cache* no desempenho final de sistemas multiprocessados, utilizando um sistema simulado com dezenas de núcleos de processamento. As memórias *cache* foram avaliadas em termos de organização, tamanho, associatividade, tamanho do bloco de dados e hierarquia de memórias *cache*. Além dessas avaliações, uma análise sobre a influência da contenção gerada pela limitação no número de portas das memórias *cache* é apresentado, finalizando a seção de avaliação dos resultados.

No capítulo 5, como finalização deste estudo, são apresentadas as conclusões e análises a respeito de tendências de memórias *cache* nos futuros processadores *multi-core* e *many-core*, além de uma breve apresentação sobre trabalhos futuros.

No apêndice A, um levantamento teórico é apresentado sobre as memórias *cache*, abordando memórias *cache* para processadores com *single-core* e *multi-core*, apresentando também uma breve descrição de interconexões.

No apêndice B, são apresentados resultados e análises complementares sobre o impacto das memórias *cache* no desempenho final de sistemas multiprocessados para os diversos experimentos apresentados.

2 ARQUITETURAS *MULTI-CORE*

" There is nothing so stable as change. "
— BOB DYLAN

O estudo das atuais arquiteturas de processadores *multi-core* disponíveis no mercado é de grande importância para avaliar a direção traçada atualmente pelas industrias de processadores. Além dessas arquiteturas, as atuais pesquisas sobre a influência da memória *cache* nos processadores com múltiplos núcleos de processamento completam a avaliação de estado da arte sobre memórias *cache* em *multi-core*.

Nesse capítulo será apresentado primeiramente um levantamento sobre teórico sobre arquiteturas *multi-core*. Após a base teórica, será apresentado um estudo de casos com os atuais processadores disponíveis no mercado com enfoque no sistema de memória destes processadores. A seção seguinte estará abordando as principais pesquisas sobre formas de compartilhamento de memórias *cache* para *multi-core*.

2.1 Arquiteturas de Alto Desempenho

Vários projetos de arquiteturas de processadores vêm adotando ao longo de décadas técnicas tradicionais (HENNESSY; PATTERSON, 2007) (SMITH; SOHI, 1995) (STALLINGS, 1996) (UNGERER; ROBIC; SILC, 2002) (UNGERER; ROBIC; SILC, 2003) como *pipeline*, superescalaridade e *multithreading* para explorar o paralelismo de execução das aplicações e assim melhorar o desempenho de computadores de propósito geral.

Em um *pipeline* superescalar (SMITH; SOHI, 1995), além do processamento de instruções ser dividido em estágios, é feita uma completa sobreposição das instruções, utilizando para isso, o aumento do número de unidades funcionais e técnicas para solucionar falsas dependências entre as instruções, dentre outras. Desta forma, os processadores superescalares são capazes de aumentar consideravelmente o desempenho na execução de cargas de trabalho com alto paralelismo no nível de instruções.

O suporte a múltiplas *threads* (UNGERER; ROBIC; SILC, 2002) (UNGERER; ROBIC; SILC, 2003) é uma alternativa de exploração de paralelismo não mais no nível de instruções, mas no nível de fluxo de instruções (*threads*). Isso significa um aumento na vazão de *threads*, podendo mais de uma *thread* ser executada ao mesmo tempo, ao contrário da superescalaridade onde a vazão é de instruções de uma única *thread*. Diversas são as técnicas para exploração do paralelismo no nível de *threads*, sendo que a mais conhecida é a SMT (*Simultaneous Multithreading*) que é suportada por uma arquitetura superescalar.

Ao longo de décadas, as técnicas de aumento de profundidade do *pipeline* aliado ao aumento da freqüência de trabalho do processador (ciclo de relógio) foram utilizadas a fim de obter o máximo desempenho a cada nova geração de processador. O custo para esse ganho de desempenho foi o aumento da complexidade da unidade de controle dos processadores, assim como o aumento no consumo de potência e temperatura do sistema computacional.

Entretanto, esta forma tradicional de aumento de desempenho começou a apresentar problemas físicos, relacionados com o alto grau de integração dos componentes como o atraso do fio, consumo de potência estática, entre outros. Assim, essa complexa abordagem de extração de paralelismo vem dando lugar a uma abordagem diferente. A fim de aumentar ainda mais o desempenho, e ainda, algumas vezes diminuir a potência dissipada, o uso de processadores com múltiplos núcleos (CMPs - *Chip Multi-Processors*) (OLUKOTUN et al., 1996) vem sendo consolidada (SINHAROY et al., 2005) (BARROSO et al., 2000) (KONGETIRA; AINGARAN; OLUKOTUN, 2005) (KUMAR et al., 2005) como uma boa abordagem para o aumento do desempenho de computação.

Ainda com a técnica de múltiplos núcleos no mesmo *chip*, nada impede que sejam utilizadas superescalaridade e SMT em cada núcleo, mas nesses casos pode haver um aumento considerável na área do CMP em função da duplicação de registradores além de outros componentes. Em função desta nova abordagem de projeto, os processadores com múltiplos núcleos e que suportam múltiplas *threads* também são conhecidos como CMT ou *Chip Multithreading* (FREITAS; NAVAUX, 2006) (SPRACKLEN; ABRAHAM, 2005).

2.1.1 Processadores *Multithreading*

Nos projetos atuais de processadores, um dos grandes objetivos é a extração máxima do desempenho. Uma das formas está na exploração do paralelismo, seja na execução das instruções ou nos fluxos de instruções. Nesse contexto, podemos considerar que um fluxo de instruções é uma *thread* e que uma *thread* é um processo, ou parte de um programa em execução. Se um processador suporta a execução de múltiplas *threads* (ACOSTA et al., 2005) (KOUFATY; MARR, 2003) (GONÇALVEZ; NAVAUX, 2002) (EGGERS et al., 1997), significa que esse processador é capaz de executar fluxos de instruções diferentes. Nesse caso, cada uma destas *threads*, ou fluxo de instruções, inicia em endereços diferentes de memória.

O suporte à *multithreading* possui duas abordagens (UNGERER; ROBIC; SILC, 2003) (UNGERER; ROBIC; SILC, 2002): *Implicit Multithreading* e *Explicit Multithreading*:

- *Implicit Multithreading*: Exploração do paralelismo existente em programas seqüenciais através de especulação no nível de *thread*. Nessa abordagem, um processador gera múltiplas *threads* especulativas de um único programa seqüencial, dinamicamente, ou estaticamente com ajuda do compilador, e executa todas concorrentemente.

- *Explicit Multithreading*: Exploração do paralelismo existente entre programas de origens diferentes. As *threads* geradas a partir de cada um desses programas podem ser executadas em um mesmo *pipeline*.

Nos dois casos, cada uma das *threads* possui um banco de registradores e contadores de programa específicos, representando cada um dos múltiplos contextos em atividade no processador. A diferença está na uso de execução de *threads* especulativas de um mesmo

programa seqüencial na abordagem *implicit multithreading* ou na execução de *threads* independentes e de programas distintos na abordagem *explicit multithreading*.

Um único núcleo de processamento (escalar ou superescalar) pode suportar múltiplas *threads*. Pequenas modificações na organização interna do núcleo são responsáveis por permitir a execução simultânea ou chaveada das *threads*.

Ainda em se tratando de *multithreading*, podemos ter diversos tipos de exploração de paralelismo no nível de *threads*, como o SMT (*Simultaneous Multithreading*), o IMT (*Interleaved Multithreading*) e o BMT (*Block Multithreading*).

- *Simultaneous Multithreading*: Esse tipo avançado de *multithreading* se aplica a processadores superescalares. Um processador superescalar simples busca instruções da mesma *thread* a cada ciclo do processador. Em um processador SMT, o processador pode buscar instruções de várias *threads* a cada ciclo do processador. Esse tipo de *multithreading* se prevalece do fato de que, para uma única *thread*, o número de instruções paralelas a serem buscadas e executadas é limitado. Assim, buscando instruções de múltiplas *threads*, o processador tenta reduzir o número de unidades funcionais sem uso a cada ciclo de relógio.

- *Interleaved Multithreading*: A execução de cada *thread* é alternada em cada ciclo de relógio do processador. Esse modo de trabalho visa remover todas paradas por dependência de dados de um *pipeline*. Uma vez que uma *thread* é relativamente independente das outras *threads*, existe menos chance de uma instrução precisar esperar por um dado de uma instrução executada anteriormente, uma vez que existirá instruções de outras *threads* intercaladas no *pipeline*.

- *Block Multithreading*: Cada *thread* é executada até que seja bloqueada por um evento que normalmente cria um longo período de espera. Tal evento pode ser uma falta de dados na memória *cache* criando assim uma latência para acesso aos dados. Dessa maneira, ao invés de esperar o evento de alta latência, o processador deverá trocar a execução para outra *thread* que esteja pronta para execução. A *thread* que ocasionou o evento de alta latência só receberá status de pronta para execução assim que sair do estado de espera. Esse modelo de *multithreading* visa esconder as altas latências de acesso à memória, mascarando essas latências com a execução de outro fluxo de instruções.

2.1.2 Processadores *Multi-Core*

Pesquisas sobre as melhores alternativas de projeto de arquiteturas de processadores têm usado basicamente o estudo de cargas de trabalho para entender melhor o comportamento do processador.

Um dos primeiros estudos que identificou o potencial do uso de *chip multiprocessor* foi proposto em (OLUKOTUN et al., 1996), onde foi apresentado um estudo onde dois tipos de arquiteturas foram expostos a um mesmo tipo de carga de trabalho. O estudo procurou definir qual o melhor tipo de arquitetura para cargas onde havia um baixo ou grande paralelismo no nível de *thread*. A Figura 2.1 (OLUKOTUN et al., 1996) apresenta os dois tipos de arquiteturas que foram comparadas, uma arquitetura superescalar com execução de seis instruções simultâneas e uma arquitetura com múltiplos núcleos de processamento suportando duas instruções simultâneas por núcleo de processamento.

Para garantir apenas a influência da carga de trabalho submetida, os dois projetos possuíam as mesmas latências de acessos à memória, principalmente o tempo de acesso à

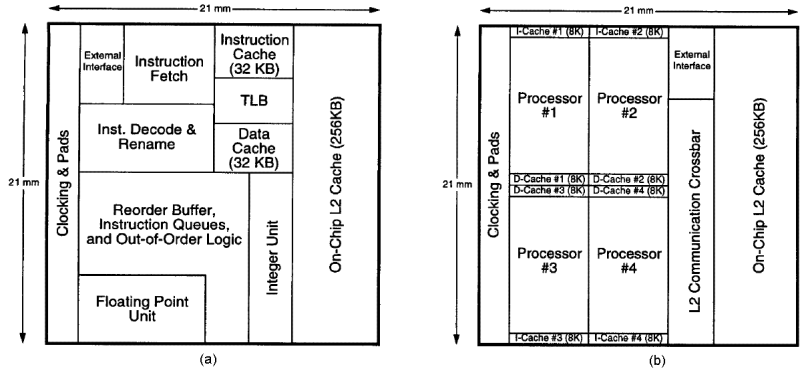


Figura 2.1: Comparativo de processadores, (a) Processador superescalar; (b) Processador *multi-core*; (OLUKOTUN et al., 1996).

memória *cache* e a mesma ocupação de área. As mesmas cargas de trabalho foram aplicadas aos dois projetos com as seguintes características, operações de números inteiros, ponto flutuante, e de multiprogramação.

Os resultados demonstraram que para cargas de trabalho onde as aplicações não são paralelizáveis, o ganho é favorável ao processador superescalar em 30%. Nesse caso existe uma exploração melhor do paralelismo no nível de instrução. Para aplicações onde existe um baixo paralelismo de *threads*, o ganho ainda é favorável à arquitetura superescalar, mas no máximo de 10%. No entanto, onde há um grande paralelismo no nível de *thread*, o ganho passa a ser da arquitetura CMP variando de 50% a 100% em relação ao superescalar.

As cargas de trabalho com grandes níveis de paralelismo em *thread* executam aplicações independentes com processos independentes. Aplicações de visualização e multimídia, processamento de transações e aplicações científicas de ponto flutuante são exemplos destas cargas de trabalho. Esta pesquisa serviu como base para o processador Hydra CMP (HAMMOND et al., 2000), que também gerou resultados para o processador UltraSparc-T1 (KONGETIRA; AINGARAN; OLUKOTUN, 2005).

Atualmente, a grande maioria dos processadores de propósito geral são exemplos de arquiteturas com núcleos homogêneos (iguais) e para um mesmo propósito de funcionamento (aplicações gerais no caso do GPP - *General-Purpose Processor*). No entanto, projetos de processadores *multi-core* para aplicações em sistemas embarcados, freqüentemente possuem núcleos heterogêneos (KUMAR et al., 2004) (KUMAR et al., 2005). Nesse caso, cada núcleo, ou conjunto de núcleos, é responsável por processamentos específicos e distintos dos demais. Uma classe de processadores que representa adequadamente as arquiteturas com núcleos heterogêneos são os processadores conhecidos como MPSoCs (*Multi-Processor System-on-Chip*) (WOLF, 2004) os quais podem apresentar mais de um processador de propósito geral interno ao *chip*, porém cada um desses possui características diferentes, onde um determinado núcleo pode se adequar melhor a um conjunto de aplicações enquanto outro núcleo se encaixa melhor a um segundo conjunto de aplicações.

A Figura 2.2, estendida de (UNGERER; ROBIC; SILC, 2002) apresenta uma comparação entre um superescalar de quatro vias 2.2(a), um processador SMT (*Simultaneous Multithreading*) 2.2(b), um processador IMT (*Interleaved Multithreading*) 2.2(c), um processador BMT (*Block Multithreading*) 2.2(d), sendo todos CMT superescalares de quatro

(a) Superescalar (b) SMT (c) IMT (d) BMT (e) CMP

Figura 2.2: Comparativo entre processadores superescalar, processadores *multithreading* e *multi-core*. Na vertical está ilustrado a linha do tempo de execução, na horizontal encontram-se as vias de execução de cada processador. Pode-se ver ainda, as vias de execução ocupadas com os fluxos de instrução (A, B, C e D).

vias, e por fim, um processador *multi-core* onde cada um dos dois núcleos é um superescalar de duas vias 2.2(e).

Fora o processador superescalar simples, todos os processadores são CMTs que suportam de duas (CMP) até quatro *threads* (SMT, IMT e BMT) simultâneas. No processador superescalar simples apenas uma *thread* está ativa por vez (A), assim apenas instruções de uma *thread* são executadas até que haja uma troca de contexto. Para o processador SMT, quatro *threads* ficam ativas (A, B, C e D), além disso, instruções das quatro *threads* podem ser executadas em um mesmo ciclo de máquina aproveitando todas unidades funcionais. Já no caso do IMT e BMT, apenas instruções de uma *thread* são executadas por ciclo, onde a troca entre *threads* ativas acontece a cada ciclo no caso do IMT e para o BMT a troca ocorre ao executar um evento de alta latência. No caso do *multi-core* ou CMP (*Chip Multiprocessor*) cada núcleo recebe uma única *thread*, mas duas instruções de cada *thread* podem ser executadas ao mesmo tempo. Assim, para que um núcleo passe a operar sobre outra *thread*, deve haver uma troca de contexto, o que é um evento de mais alta latência que a troca entre *threads* ativas.

2.1.3 Memórias *Cache* para *Multi-Core*

Com a contínua demanda por desempenho computacional, as memórias *cache* vêm sendo largamente adotadas nas últimas décadas, nos mais diversos tipos de projetos arquiteturais de processadores (PATTERSON; HENNESSY, 2005). Uma vez incorporada, ao haver necessidade por um dado, esse será procurado primeiramente nas memórias de alto desempenho, ou seja, nas memórias *cache* presentes. Apenas no caso onde o dado não seja encontrado, o processador terá que acessar as memórias de baixo desempenho, ou seja, a memória principal, ou outros dispositivos de armazenamento disponíveis.

Ao longo de décadas, diversas técnicas foram utilizadas a fim de obter os melhores desempenhos com a utilização de memórias *cache* (HENNESSY; PATTERSON, 2007). A maioria dessas tecnologias desenvolvidas leva em conta dois princípios de localidade:

- Princípio de localidade espacial: se um item é referenciado, provavelmente seus vizinhos também o sejam.

- Princípio de localidade temporal: se um item é referenciado, provavelmente ele será referenciado novamente em um curto espaço de tempo.

Assim, através desses dois princípios básicos, muitos avanços foram feitos nas memórias *cache* ao longo de décadas, como por exemplo, os dispositivos de pré-busca, os modos de mapeamento de blocos da memória, as políticas de alocação e substituição de dados da memória *cache*, além das estratégias de escrita, entre outras.

Com a utilização das memórias *cache* nos projetos de arquitetura dos processadores, os esforços passaram a ser a redução das faltas de dados na memória e, além disso, a redução das penalidades causadas por falta de dados. Com a utilização de hierarquias de memórias *cache* esses problemas puderam ser reduzidos.

Porém, com novas tecnologias de integração, o tamanho das memórias *cache* tendem a continuar a crescer assim como a demanda por mais vazão de dados. Atualmente, as memórias *cache* estão organizadas em múltiplos níveis, onde cada nível é formado com um grande bloco que é projetado considerando problemas de latência, largura de banda, interconexão, entre outros. Entretanto, para futuras tecnologias de integração, as memórias *cache* que utilizam o modelo atual de arquitetura irão sofrer com problemas de atraso do fio, gerando assim grande contenção ao processamento. Dado o panorama para as futuras tecnologias, arquiteturas de memória *cache* não-uniformes (NUCA) (KIM; BURGER; KECKLER, 2002) (KIM; BURGER; KECKLER, 2003) têm surgido como promessas para substituir os atuais modelos de arquitetura uniforme de memória *cache* (UCA).

Mesmo com as futuras tendências de uso de memórias NUCA nos processadores com memórias *cache* dominadas por atraso do fio, as memórias UCA ainda devem ser avaliadas a fim de definir os limites de utilização, como limites de compartilhamento, limites de latência, entre outros, de forma que se possa obter dados sobre os domínios favoráveis para o uso de arquiteturas uniformes e saber a partir de qual ponto as arquiteturas não uniformes deverão ser empregadas nos futuros processadores *multi-core*. Considerando ainda as questões sobre qual a melhor forma de programar (AKHTER; ROBERTS, 2006) (CHANDRA et al., 2006) aplicações de alto desempenho nesses novos cenários devem ser consideradas.

2.2 Estudo de Casos

Atualmente, existem no mercado diversos fabricantes de processadores, sendo que podemos citar quatro principais fabricantes de processadores de propósito geral, AMD, Intel, Sun e IBM. Estes fabricantes representam os principais produtores de processadores *multi-core* do estado da arte com diferentes arquiteturas e conjunto de instruções.

Nos dias atuais, o número de núcleos de processamento dentro de um mesmo *chip* varia entre 2 até 8 núcleos de processamento de propósito geral, com indicativos de lançamentos futuros de processadores com até 80 núcleos de processamento (AZIMI et al., 2007) (INTEL CORPORATION, 2007a).

Contudo, sob ponto de vista de organização de memória, os fabricantes apostam em diferentes soluções afim de obter o melhor desempenho computacional, o que fica mais claro ao observar atentamente o subsistema de memória *cache* presente nos principais processadores *multi-core* da atualidade.

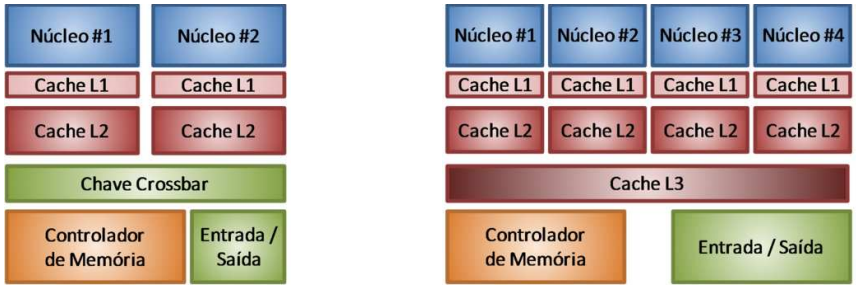
Esta seção apresenta um estudo de casos das principais famílias de processadores dos fabricantes AMD, Intel, Sun e IBM, a fim de fazer um levantamento sobre as arquiteturas

e organizações de memória *cache* dos processadores atuais, com ênfase nas características relacionadas aos dispositivos de memória *cache* desses processadores.

2.2.1 Processadores *Multi-Core* AMD

A indústria de semi-condutores AMD atualmente possui dois principais modelos de processadores com múltiplos núcleos de processamento da linha Barcelona, o modelo com 2 núcleos chamado de Opteron Duo e o modelo Opteron Quad com 4 núcleos.

O modelo com dois núcleos (AMD, 2005) pode ser visto na Figura 2.3(a), onde se apresenta com uma memória *cache* de nível 1 de instruções e dados separadas, sendo neste nível uma memória *cache* por núcleo. Já no segundo nível de memória *cache*, os núcleos de processamento possuem uma memória *cache* privada para cada. A interconexão utilizada entre a memória *cache* e o restante do sistema é a chave *crossbar*.



(a) Opteron Duo (b) Opteron Quad

Figura 2.3: Diagrama de blocos da arquitetura dos processadores AMD da família Barcelona, Opteron Duo e Opteron Quad.

No modelo com quatro núcleos de processamento, apresentado na Figura 2.3(b), a solução adotada é a utilização de memórias *cache* de nível 1 e 2 privadas para cada núcleo. Porém, o fabricante adotou um terceiro nível de memória *cache*, esse terceiro nível de memória *cache* deverá ser o responsável por reduzir a contenção gerada principalmente pela largura de banda da interconexão que liga o processador à memória principal.

Os dois modelos disponíveis do processador Barcelona, com 2 e 4 núcleos de processamento, utilizam *hardware* de pré-busca de dados da memória *cache* para reduzir os tempos de espera dos núcleos por dados e instruções, além disso, estes processadores apresentam um controlador de memória integrado ao *chip* para reduzir a contenção ao acesso à memória principal.

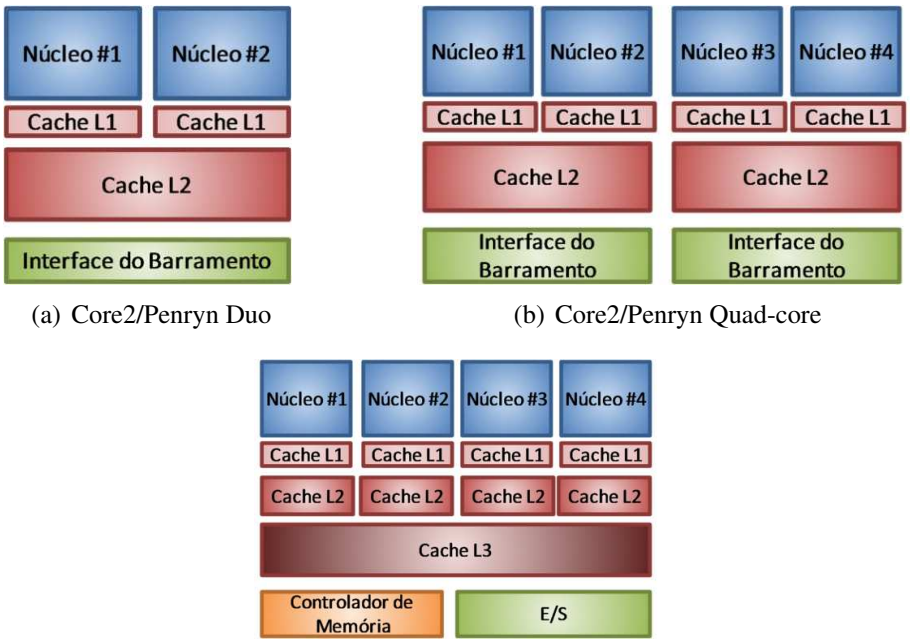
2.2.2 Processadores *Multi-Core* Intel

Atualmente, a fabricante de processadores Intel apresenta dois modelos principais de processadores da família Core2 (INTEL CORPORATION, 2007b), além da nova família Nehalem. Os modelos Core2 são, Core2 Duo com 2 núcleos de processamento e o modelo Core2 Quad-core com quatro núcleos de processamento. Já a família Nehalem apresenta o processador Core i7.

Estes processadores Core2 fabricados originalmente em 65 nm foram relançados também em tecnologia de integração de 45 nm com a nomenclatura Penryn, também conhecidos por Clovertown, porém com a mesma arquitetura dos processadores Core2, apresentando algumas melhorias como maiores memórias *cache* com maior associatividade.

Aqui trataremos os processadores Penryn e Core2 como sinônimos, uma vez que estamos interessados na arquitetura e organização desses, e não na tecnologia de integração e parâmetros de implementação.

Os processadores da família Core2 Duo são formados de dois núcleos de processamento, com as memórias *cache* de nível 1 (dados e instruções) privadas, além da memória *cache* de nível 2 compartilhada entre os dois núcleos, como pode ser visto na Figura 2.4(a).



(a) Core2/Penryn Duo

(b) Core2/Penryn Quad-core

(c) Core i7

Figura 2.4: Diagrama de blocos da arquitetura dos processadores Intel da família Core2/Penryn, Core2 Duo, Core2 Quad-Core e Nehalem, Core i7.

Já a família Core2 Quad-core representada na Figura 2.4(b), mostra que os núcleos também apresentam o primeiro nível de memória *cache* privada, porém, no segundo nível existem no total apenas duas memórias *cache*, cada uma compartilhada entre dois núcleos. Além disso, esses processadores não possuem nenhum outro nível na hierarquia de memória *cache*. Assim, o modelo Core2 Quad-core é basicamente a junção de dois processadores Core2 Duo.

Podemos ainda notar, que de acordo com as figuras, os dois modelos de processadores Core2 utilizam barramentos como interconexões entre as memórias *cache* e o resto do sistema e que o controlador de memória principal fica localizado fora do processador. Esses processadores utilizam técnicas de pré-busca de dados da memória *cache* para reduzir os tempos de espera de dados e instruções pelo processador.

A família Nehalem representada pelo processador Core i7 (INTEL CORPORATION, 2008a) (INTEL CORPORATION, 2008b) apresenta grandes mudanças em relação ao seu antecessor Core2. O processador Core i7 apresenta 4 núcleos de processamento, sendo que cada núcleo possui suporte à SMT de até 2 *threads* por núcleo, trabalhando assim com 8 *threads* ativas no total. Assim, esse processador apresenta quatro núcleos nativos, e não dois encapsulamentos como os processadores Core2 Quad-Core.

O Nehalem Core i7 também apresenta o controlador de memória principal integrado

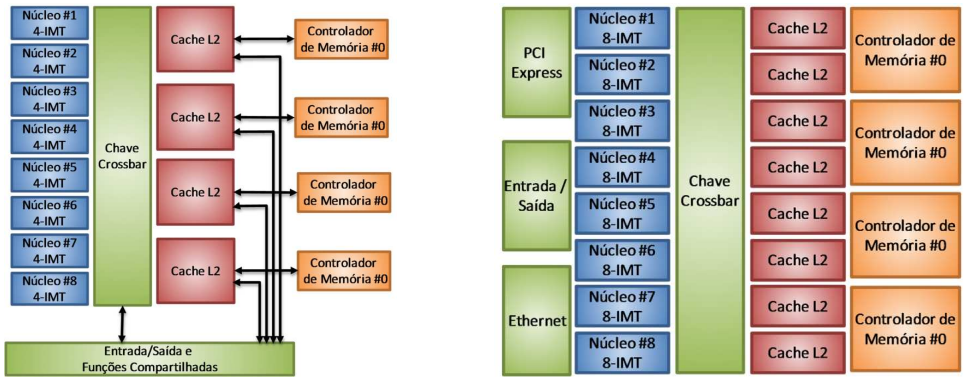
aos núcleos, aumentando assim a vazão de dados entre a memória *cache* e a memória principal, como pode ser visto na Figura 2.4(c). A memória *cache* de primeiro e segundo nível desse processador Core i7 é privada para cada núcleo de processamento, possuindo uma memória *cache* de terceiro nível compartilhada entre todos os núcleos de processamento. Mesmo possuindo memórias *cache* de segundo nível privadas para cada núcleo de processamento, o projeto desse processador determina que qualquer núcleo poderá acessar dados na memória *cache* L2 de outro núcleo a fim de obter melhor do desempenho, assim, pode-se considerar esta memória *cache* sendo de acesso não uniforme.

2.2.3 Processadores *Multi-Core* Sun

A empresa Sun fabricante de processadores, apresenta duas famílias de processadores *multi-core*, a família Niagara e Victoria Falls, também conhecida como Niagara II.

Os processadores da empresa Sun da família Niagara apresentam um conceito diferente para esconder as altas latências ocasionadas pelo tempo de acesso à memória principal do sistema. Esses processadores trabalham com múltiplas *threads* por núcleo. Dessa forma, quando uma *thread* necessita esperar dados da memória, essa *thread* sofre um chaveamento, e assim, uma outra *thread* poderá ser executada.

O primeiro modelo dessa família, chamado de Niagara (KONGETIRA; AINGARAN; OLUKOTUN, 2005), apresenta 8 núcleos de processamento com memórias *cache* de nível 1 privadas, onde cada núcleo suporta até 4 *threads* IMT (*Interleaved Multithreading*), somando então 32 *threads* ativas, podendo ter até 8 *threads* em execução simultaneamente. Todos os núcleos encontram-se conectados a uma chave *crossbar* ligada também às memórias *cache* de segundo nível, onde existem 4 memórias *cache*, e cada memória *cache* é compartilhada entre dois núcleos, como é visto na Figura 2.5(a).



(a) Niagara 1 (b) Niagara 2

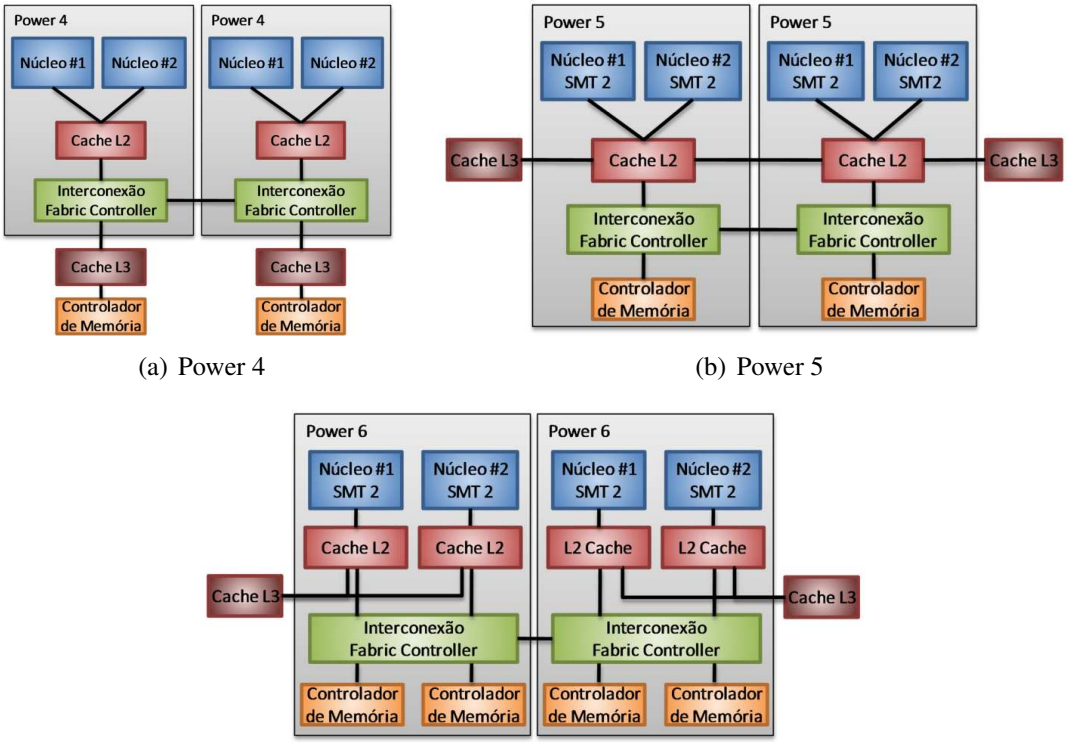
Figura 2.5: Diagrama de blocos da arquitetura dos processadores Sun da família Niagara e Victoria Falls.

O segundo modelo, Victoria Falls (KANTER, 2006), também possui 8 núcleos de processamento, porém com mais unidades funcionais e um esquema mais agressivo de TLP (*Thread Level Parallelism*), uma vez que cada núcleo trabalha com até 8 *threads*, podendo ter 2 *threads* em execução simultânea em cada núcleo. Desta forma, até 64 *threads* podem estar ativas, com até 16 destas em execução simultânea. O modelo deste processador pode ser visto na Figura 2.5(b).

2.2.4 Processadores *Multi-Core* IBM

O fabricante de processadores da linha Power, a IBM, apresenta três principais modelos de processadores *multi-core*, representando a linha evolutiva, que são o Power 4, Power 5 e o Power 6.

O processador Power 4 (DEMONE, 2004) possui dois núcleos de processamento, cada um conta com uma memória *cache* de nível 1 privada conectada a uma memória *cache* de nível 2 compartilhada para os 2 núcleos, como pode ser visto na Figura 2.6(a). Como o fabricante prevê o uso de mais de um processador em um sistema computacional, este apresenta uma interconexão ligada ao nível 2 de memória *cache*, que conecta essa memória ao terceiro nível de memória *cache*, e também a esta mesma interconexão de algum outro processador, se disponível.



(a) Power 4

(b) Power 5

(c) Power 6

Figura 2.6: Diagrama de blocos da arquitetura dos processadores IBM da família Power.

O processador Power 5 (SINHAROY et al., 2005) apresenta uma configuração com duas principais modificações com relação ao seu antecessor Power 4. Estas mudanças podem ser vistas na Figura 2.6(b).

O Power 5 também apresenta dois núcleos compartilhando a mesma memória *cache* de nível 2, porém esses núcleos suportam SMT (*Simultaneous Multithreading*) de duas *threads* por núcleo, ou seja, existem algumas duplicações de unidades de controle e unidades funcionais, a fim de que cada núcleo possa trabalhar simultaneamente com até duas *threads*. Além dessa mudança, o Power 5 apresenta a memória *cache* de segundo nível conectada diretamente à memória *cache* de nível 3 sem passar pelo barramento, além da ligação desta memória *cache* de segundo nível com a conexão que interliga os diversos núcleos ao sistema ou a outros processadores. De acordo com (SINHAROY et al., 2005),

os processadores Power 5 podem apresentar queda de desempenho em algumas aplicações quando ativado a função de SMT dos núcleos de processamento.

As interconexões apresentadas nos dois modelos de processadores Power 4 e Power 5 são barramentos.

Já a linha de processadores Power 6 (LE et al., 2007) apresentada na Figura 2.6(c) é formado de dois núcleos de processamento com suporte à SMT de 2 *threads* cada, suportando então, até 4 *threads* ativas. Cada núcleo de processamento possui uma memória *cache* de primeiro e segundo níveis privada, onde cada memória *cache* de segundo nível possui uma conexão exclusiva para o controlador de memória *cache* de terceiro nível, o qual fica localizado fora do encapsulamento. Além disso, cada memória *cache* de segundo nível possui ligação com a interconexão responsável pela comunicação entre diversos processadores e também responsável pela comunicação entre dois controladores de memória principal disponíveis dentro do *chip*.

Para o aumento de desempenho, o Power 6 (BERRIDGE et al., 2007) aposta na utilização de núcleos de alta freqüência de operação, além dos dispositivos de pré-busca de dados e instrução da memória *cache* para reduzir assim os tempos de latência dos núcleos.

2.3 Trabalhos Correlatos

Com o crescente uso de *chips* multiprocessados, diversos trabalhos abordam alternativas e estudam formas de diminuir os impactos das altas latências de memória em *multi-core*, pesquisando para isso, as possibilidades de hierarquias de *cache* e modos de compartilhamento dessas entre os núcleos de processamento. Nesta seção, citamos alguns trabalhos desta linha de pesquisa, comentando as principais diferenças desta dissertação com cada trabalho correlato apresentado.

O trabalho apresentado por Nayfeh, Olukotun e Singht (1996) faz um estudo de diferentes organizações de memória em um computador multiprocessado, apresentando a avaliação da influência das diferentes organizações de *clusters* de processadores compartilhando memória *cache* sobre o desempenho do sistema. Os resultados obtidos mostraram que, para um sistema multiprocessado de até oito processadores, as contenções geradas pelo barramento são responsáveis por grande parte do tempo total de execução. Sendo assim, o agrupamento dos processadores pode eliminar parte dessa contenção. Além disso, este agrupamento de processadores tende a reduzir os tempos de espera por leitura dos dados, quando utilizado espaço de trabalho compartilhado.

Comparando com o trabalho de Nayfeh, Olukotun e Singht (1996), pode-se notar que existe a diferença que nosso estudo é feito no nível de núcleos de processamento e não no nível de processadores ou máquinas interconectadas, assim, essa diferença de tecnologias pode nos levar a diferentes conclusões, uma vez que as latências de interconexão em nosso caso são bem menores que as latências enfrentadas por *clusters* de processadores apresentado no trabalho correlato.

Em Marino (2006a), um dos focos do trabalho foi testar diferentes organizações de memória. Porém, a simulação foi baseada em um sistema contendo um *chip* com múltiplos núcleos, onde foram simuladas diferentes organizações de agrupamento de núcleos compartilhando memória *cache* L2. Esta simulação foi a partir de um CMP de 32 núcleos, com agrupamentos formados de 1, 2 e 4 núcleos. Segundo o autor, a principal conclusão foi que o compartilhamento da L2 por mais de 2 processadores contribuiu para o ganho de desempenho das aplicações de teste utilizadas. É importante ressaltar que uma das principais diferenças em relação ao trabalho de (NAYFEH; OLUKOTUN; SINGHT, 1996), é

que, neste trabalho as interconexões entre os processadores com a memória *cache* L2 são internas ao *chip*, levando assim a menores latências.

Em Marino (2006b) foi apresentado o estudo do uso de agrupamentos formados por até 4 núcleos compartilhando a *cache* L2 em um *chip* multiprocessado, e ainda, apresenta resultados com variações no tamanho da memória *cache* L2 compartilhada (1MB, 2MB e 4MB). Com relação aos resultados, o aumento do número de processadores por grupo compartilhando memórias *cache* maiores apresentou ganho de desempenho na maioria das aplicações do *benchmark* testado.

Com relação aos trabalhos de Marino (2006a), (2006b), nosso estudo se apresenta como uma extensão da quantidade de agrupamentos, avaliando compartilhamento de 8, 16 e 32 núcleos em uma mesma memória *cache* L2 além de 1, 2 e 4 processadores em mesma memória *cache* L2, que havia sido apresentado nos correlatos. Outra diferença significante é a avaliação que nosso trabalho trás sobre a contenção gerada pela quantidade limitada de portas da memória, uma vez que o trabalho correlato não faz tal avaliação.

Em Kumar et al. (2005) são estudados diversos aspectos relacionados com as interconexões dentro de um *chip* multiprocessado. Este trabalho revela a importância em projetar as interconexões levando em consideração todos os outros aspectos do processador, como tamanho de memória *cache* e agrupamento dos núcleos utilizados. Além disso, o artigo conclui que ao levar em conta os custos de interconexão, nem sempre o compartilhamento de *cache* L2 é vantajoso, sob o ponto de vista de ganho de desempenho e também sob o aspecto do sobrecusto de área do projeto, gerada pela interconexão da memória *cache* com os núcleos.

O trabalho de Kumar et al. (2005) levanta a questão da interconexão *intra-chip*, embora as interconexões sejam de grande importância no contexto de processadores *multi-core*, podemos notar que para a avaliação do compartilhamento de memória *cache*, é necessário mais que uma avaliação da interconexão entre os núcleos para indicar qual organização de memória levará o sistema a obter melhor desempenho. Assim, nosso trabalho, considera além da interconexão, a execução de aplicações de carga de trabalho e assim, a influência do compartilhamento da memória *cache* na comunicação entre diversas *threads*.

O trabalho de Meyer (2007) apresenta uma ferramenta de síntese que utiliza o método de *simulated annealing* levando em conta simultaneamente a síntese do mapeamento de dados, a alocação da memória e o barramento para obter otimizações para sínteses de sistemas multiprocessados em contexto de sistemas embarcados. Como conclusão, o trabalho apresenta reduções de custo para projetos de sistemas de alto desempenho e também para sistemas de baixo custo, em relação a trabalhos que utilizam a síntese de mapeamento de dados e alocação de memória separadas da síntese do barramento.

Considerando o trabalho de Meyer (2007), podemos notar que nossos métodos empíricos utilizando simulação, apresentam resultados sobre o desempenho da execução de cargas de trabalho de aplicações científicas, em um sistema operacional real, revelando dessa maneira uma estimativa de desempenho realística. Além disso, podemos ver que o trabalho correlato apresenta preocupações mais ligadas a sistemas embarcadas, enquanto que em nosso estudo estamos em um ambiente de propósito geral, onde o desempenho do sistema é recebe maior atenção.

A proposta de Zhao et al. (2007) apresenta a arquitetura de *cache* SPS2 para *chip* multiprocessado. A arquitetura proposta define que cada processador terá tanto memória *cache* L2 privada quanto compartilhada, e para isso um novo algoritmo de coerência de *cache* é apresentado. Os resultados de simulação apresentados mostram que essa ar-

quitetura leva vantagem no quesito de acessos fora do *chip* comparado com memória L2 privada. Para os testes feitos, mostra também que esta nova arquitetura de *cache* obtém um desempenho médio melhor que memórias *cache* L2 totalmente compartilhadas ou totalmente isoladas.

Embora o trabalho de Zhao et al. (2007) apresente uma avaliação onde cada núcleo utiliza um modelo de memória *cache* privada e compartilhada, não fica claro as implicações físicas desse modelo, com relação á área, consumo de potência, latências, sobretudo para a implementação desse modelo em processadores com dezenas de núcleos de processamento. Logo, podemos ver que, embora nosso trabalho não avalie propostas de memória *cache* híbrida, são fornecidas através dos experimentos resultados estimados de desempenho, potência, área entre outros a respeito da memória *cache* L2 com diversos graus de compartilhamento para um sistema de 32 núcleos de processamento.

Em Zahran (2003) o trabalho apresentado avalia a hierarquia de memória para *chip* multiprocessado. Assim, o trabalho apresenta quatro organizações de memória *cache*, além de protocolos de coerência, a fim de estudar o comportamento das arquiteturas propostas. Sob a carga de trabalho utilizada, o trabalho conclui que a melhor forma de organização de memória é o uso de *cache* privadas nos níveis L1 e L2 combinado com o uso de um bom protocolo de coerência no nível L2.

Com relação ao trabalho de Zahran (2003), onde o correlato utiliza apenas três aplicações, nosso estudo utiliza uma maior quantidade de aplicações de carga de trabalho, gerando assim, um resultado mais aprofundado considerando aplicações de diversos comportamentos de acesso à memória. Além disso, nosso trabalho trás avaliações considerando as latências de memória modeladas pela ferramenta Cacti, gerando assim, resultados mais realistas a respeito da influência do compartilhamento da memória *cache* L2.

O recente trabalho de Aggarwal et al. (2007) apresenta um enfoque diferente para o uso de recursos compartilhados em um processador *multi-core*, onde leva em conta uma variável de degradação do hardware ao longo do tempo, para fazer as avaliações de desempenho ao longo de anos. Com isso, o autor propõe a utilização de recursos de isolamento reconfigurável, sendo que de acordo com o trabalho, o método proposto apresenta os melhores resultados de desempenho ao longo do tempo considerando a degradação dos dispositivos ao longo dos anos. Mostrando também que, considerando a possível degradação do hardware, a utilização de recursos compartilhados faz decair o desempenho do processador rapidamente ao longo dos anos.

O trabalho de Aggarwal et al. (2007) levanta outra questão sobre a degradação do *hardware*, o que é bastante importante para as atuais e futuras tecnologias de integração, entretanto, o trabalho correlato considera apenas a variável de degradação para pautar o estudo de desempenho, assim, podemos ver que o ponto ideal seria a junção entre as avaliações apresentadas em nosso estudo, com o bom senso de considerar o isolamento dos componentes para que se tenha o melhor desempenho, pelo maior tempo.

O trabalho de Jallel, Mattina e Jacob (2005) descreve uma avaliação de compartilhamento de memória *cache*, focado no último nível da memória *cache*, utilizando cargas de trabalho paralelas de bioinformática. De acordo com os resultados do artigo, a principal característica dessa carga de trabalho é o alto grau de compartilhamento de dados, acima de 95%. Por este motivo, o objetivo do artigo é identificar o impacto das memórias *cache* compartilhadas para o aumento de desempenho. A principal conclusão é relacionada ao aumento do tamanho da *cache* compartilhada de acordo com o comportamento de compartilhamento das aplicações. Dessa maneira, múltiplas memórias *cache* privadas reduzem o comportamento de compartilhamento de dados e o desempenho.

Comparando com o trabalho de Jallel, Mattina e Jacob (2005), podemos identificar facilmente a principal diferença com relação à carga de trabalho utilizada entre nosso trabalho e o correlato. Enquanto que o trabalho correlato apresenta avaliações utilizando uma carga de trabalho paralela de bioinformática, com alto grau de compartilhamento de dados, nosso estudo utiliza aplicações científicas que não favorecem tanto o compartilhamento de memória entre os núcleos de processamento. Dessa maneira, nosso trabalho trás uma avaliação com aplicações mais equilibradas no ponto de vista de compartilhamento de dados.

O trabalho de Hsu, et al. (2005) mostra processadores *many-core* como tendências e aponta a hierarquia de memória *cache* como um problema para suportar um grande número de núcleos e *threads*. O principal objetivo é identificar os efeitos das memórias *cache* L2 e L3, considerando os diferentes tamanhos e pré-busca de endereço de instruções. A metodologia de avaliação foca em carga de trabalho de transações (TPC-C). Os principais resultados do artigo mostram que memória compartilhada pode prover um espaço de melhor eficiência e a pré-busca de instruções consegue aumentar o desempenho, porém, isso não é suficiente para reduzir o *gap* com memórias compartilhadas.

Considerando o trabalho correlato de de Hsu, et al. (2005), notamos que as aplicações avaliadas diferem de nosso trabalho. Além disso, notamos que o correlato trabalhou apenas com variáveis de tamanho da memória *cache* e pré-busca de dados, enquanto que nosso estudo avalia outros parâmetros além do tamanho da memória *cache*, como a influência da associatividade, tamanho da linha de dados, níveis na hierarquia de memória e a influência das contenções geradas pelas portas de acesso a dados das memórias. Assim, fica claro as maior abrangência de nossa avaliação.

Além das principais diferenças citadas com relação a cada trabalho correlato, podemos citar que no presente trabalho foi utilizado cargas de trabalho recentes e abrangentes, houve também, grande abrangência de parâmetros (tamanho da memória, associatividade, tamanho da linha, níveis na hierarquia de memória), modelagem das latências de memória próximas das reais usando a ferramenta Cacti, além das estimativas de ocupação de área e consumo de potência e avaliação da influência das contenções no sistema, sendo esses os principais pontos importantes e inéditos dessa dissertação em relação a maioria dos trabalhos correlatos.

3 METODOLOGIA PARA AVALIAÇÃO DE MEMÓRIAS *CACHE* EM *MULTI-CORE*

" Art is the elimination of the unnecessary. "
— PABLO PICASSO

3.1 Introdução à Metodologia

De acordo com (JAIN, 1991), a utilização de uma correta metodologia para avaliação de desempenho de sistemas computacionais evita diversos problemas, como a falta de objetivos, objetivos viciados, abordagem não sistemática, análise sem compreender o problema, métricas incorretas de desempenho, carga de trabalho não representativa, técnicas erradas para avaliação, descaso com parâmetros importantes, ignorar fatores significantes, projeto experimental inapropriado, entre outros. São diversos problemas que podem ocorrer, caso não haja um correto planejamento dos experimentos.

Para evitar problemas em nosso projeto de avaliação, podemos seguir diversos passos para assegurar que estamos efetuando as análises com rigor acadêmico. As etapas a seguir foram adaptadas de (JAIN, 1991) e aplicadas ao nosso projeto.

3.1.1 Definição do Sistema e Serviços

Esta etapa visa definir os objetivos do estudo delimitando o sistema a ser avaliado. A Figura 3.1 apresenta um diagrama genérico do sistema de memória *cache* a ser avaliado, onde estão definidos por blocos os principais componentes do sistema, como os núcleos de processamento, a interconexão, o sistema de memória *cache* e a memória principal, delimitando ainda o escopo que compreende ao processador, sendo que tais componentes estão em um mesmo *chip*.

Mesmo sem a definição clara da arquitetura e organização do sistema, com a visualização do diagrama do sistema a ser avaliado ficam evidentes os componentes que são relevantes ao estudo, mas não principais (núcleos de processamento, interconexão e memória principal), e o componente principal do estudo que é o próprio sistema de memória *cache*. Assim, o objetivo final do estudo é indicar formas de reduzir o impacto do sistema de memória *cache* e melhorar o desempenho final do sistema.

Os serviços disponíveis aos núcleos de processamento através da interconexão ao sistema de memória *cache* são definidos de forma sucinta como leitura e escrita de dados. Estes serviços podem ser propagados até a memória principal. Os serviços devem oferecer alto desempenho, evitando assim eventuais esperas dos núcleos de processamento.

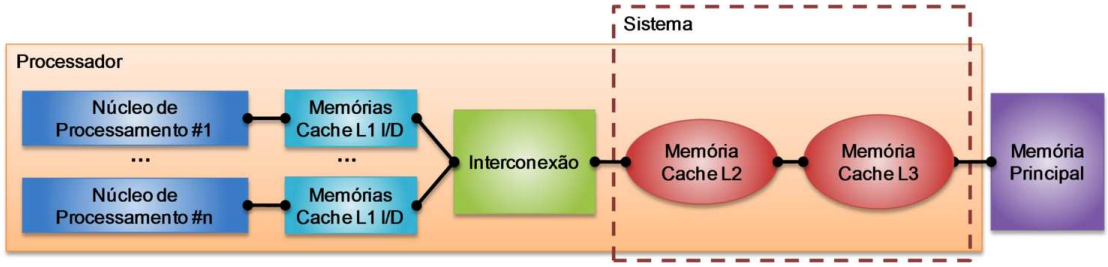


Figura 3.1: Definição do sistema a ser estudado.

3.1.2 Métricas de Avaliação

A escolha das métricas do sistema é importante, uma vez que através dessas métricas as análises serão feitas e assim deve ser possível indicar qual sistema é apropriado para cada situação, ou seja, as métricas devem ser pontos de comparação entre os sistemas avaliados.

Algumas métricas para o sistema de memória *cache* são:

- Métricas de Desempenho
  - Faltas de leituras de dados na memória *cache*.
  - Faltas de escritas de dados na memória *cache*.
  - Porcentagem de faltas de dados na memória *cache*.
  - MPKI - Quantidade de faltas de dados na memória *cache* a cada mil instruções executadas.
  - Total de ciclos para a execução da carga de trabalho.
  - Tempo total de execução da carga de trabalho.
  - Tempo total de espera de dados por requisições de leitura e escrita.
  - Tempo médio de atendimento das requisições de leitura e escrita.
  - MIPS - Número de instruções por segundo.
  - MFLOPS - Número de operações de ponto flutuante por segundo.
  - IPC - Instruções prontas por ciclos de máquina.
- Métricas Físicas
  - Tamanho do sistema de memória dado em área ocupada pela memória *cache*.
  - Consumo de potência dinâmica e estática total do sistema de memória *cache*.

Diversas técnicas estão disponíveis para avaliação de sistemas, de forma que possa nos indicar quão confiável são os dados obtidos, além das que possam servir de métrica para comparação entre os sistemas. Em nossas avaliações usaremos as seguintes métricas: Média; Variância; Desvio padrão; Intervalo de confiança; Coeficiente de variação; Total de ciclos para a execução da carga de trabalho; *Speedup*; MPKI; Consumo de energia e potências. Como o foco principal é o desempenho do sistema, a principal métrica de comparação de desempenho será o total de ciclos para a execução da carga de trabalho, onde eventualmente será decomposta por aplicação para estudar o comportamento de cada aplicação. As definições das métricas utilizadas para avaliação dos sistemas são apresentadas abaixo:

- A média é a métrica básica para agrupar os dados de forma a gerar um valor significativo de toda amostra.

$$\text{Média} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- A variância de uma variável indica quão longe em geral os valores se encontram do valor esperado.

$$\text{Variância} = \sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}^2)$$

- O desvio padrão é obtido pela raiz quadrada da variância. Este valor é o mais comum para denotar a dispersão estatística.

$$\text{Desvio Padrão} = \sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}^2)}$$

- O coeficiente de variação ou apenas COV (*Coefficient Of Variation*) representa a taxa entre o desvio padrão e a média.

$$\text{Coeficiente de Variação} = C.O.V = \frac{\text{Desvio Padrão}}{\text{Média}} = \frac{\sigma}{\bar{x}}$$

- Os intervalos de confiança estatísticos são utilizados para achar um intervalo de valores que possuem uma determinada probabilidade dos valores estarem dentro deste intervalo. A idéia é construir um intervalo de confiança para o parâmetro com uma probabilidade de $1-\alpha$ (nível de confiança) de que o intervalo contenha o verdadeiro parâmetro. Sendo, α o nível de significância, isto é, o erro que estaremos cometendo. Considerando que iremos trabalhar com pequena quantidade de amostras, $z_{1-\alpha/2}$ é obtido da distribuição t de Student. Se $\bar{x}$ é a média de uma amostra aleatória de tamanho n de uma população com variância conhecida σ, o intervalo de confiança $100(1-\alpha)\%$ da média é dado por:

$$\bar{x} - z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}} \leq \text{Intervalo de Confiança} \leq \bar{x} + z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}$$

- O *speedup* de um sistema nada mais é que uma comparação de desempenho. No caso do nosso estudo, os sistemas de um experimento são comparados e avaliados quanto ao desempenho com relação ao primeiro sistema (JOHN; EECKHOUT, 2006), ou seja, a primeira organização de memória *cache* avaliada, mostrando dessa maneira, os eventuais ganhos obtidos com cada abordagem.

$$\textit{Overall speedup} = \frac{\textit{Total time on baseline system}}{\textit{Total time on enhanced system}}$$

- A métrica MPKI (*misses per kilo instructions*) de uma memória *cache* nada mais é que a taxa de faltas de dados a cada mil instruções executadas. Essa métrica ajuda na rápida identificação de faltas na memória *cache*, fornecendo um bom valor de comparação entre diferentes aplicações e organizações.

$$\text{MPKI} = \frac{\text{Faltas de Dados}}{\text{Instruções Executadas}/1000}$$

- A energia dinâmica de um sistema de memória é aquela necessária para efetuar cada operação de leitura ou escrita.

$$\text{Energia Dinâmica}(J) = \text{Energia de Operação}(J) \cdot \text{Número de Operações}$$

- A potência dinâmica, também chamada de potência dinâmica média, é variável de acordo com a aplicação executada, assim, é dada pela energia dinâmica dividida pelo tempo de execução da aplicação.

$$\text{Potência Dinâmica}(W) = \frac{\text{Energia Dinâmica}(J)}{\text{Tempo do Sistema Ligado}}$$

- A energia estática nada mais é que a potência de *leakage* (vazamento) do sistema multiplicado pelo tempo em que o sistema ficou ligado.

$$\text{Energia Estática}(J) = \text{Potência Estática}(W) \cdot \text{Tempo do Sistema Ligado}$$

- A potência estática é a soma de todas taxas de *leakage* (vazamento) do sistema.

$$\text{Potência Estática}(W) = \sum_{i=1}^{n} \text{Potência Estática}_i$$

- A energia ou potência total de um sistema é dado pela soma das energias ou potências dinâmicas e estáticas respectivamente.

$$\text{Energia Total}(J) = \sum_{i=1}^{n} \text{Energia Estática}_i + \text{Energia Dinâmica}_i$$

$$\text{Potência Total}(W) = \sum_{i=1}^{n} \text{Potência Estática}_i + \text{Potência Dinâmica}_i$$

3.1.3 Projeto de Experimentos

O objetivo de um projeto de experimento (DoE - *Design of Experiment*) correto é obter o máximo de informações com o menor número de experimentos. Além disso, uma correta análise desses experimentos também ajuda a identificar vários fatores ou os fatores que mais influenciam no desempenho.

Alguns tipos de design existem e podem ser considerados para o estudo de performance computacional. Alguns deles são (JAIN, 1991):

- *Simple Design*: É considerado um dos mais simples, consiste em variar um fator por vez, e verificar como cada fator influência na performance.

- *Full Factorial Design*: Este modelo utiliza cada combinação de fatores possíveis e o mesmo para cada configuração possível.

- *Fractional Factorial Design*: Este modelo utiliza regras de escolha de combinações, a fim de reduzir o número de experimentos, porém continua gerando um bom nível de detalhamento dos resultados.

O modelo *Fractional Factorial Design* é um dos modelos mais indicados para avaliação de desempenho, principalmente em sistemas simulados onde o número de experimentos deve ser controlado e não muito grande. Dentro deste modelo também existem algumas sub-estratégias de design, como os designs fatoriais 2^k, 2^{k-p}, e os designs fatoriais com replicação $2^k r$.

Após o estudo sobre os tipos de design de experimentos, foi decidido utilizar o modelo *Fractional Factorial Design* que se molda melhor aos requisitos de simulação e teste pretendidos, e a escolha do sub-modelo será feita de acordo com cada experimento pretendido, escolhendo o que mais se adequar às necessidades.

3.2 Avaliação de Desempenho Computacional

A avaliação de desempenho pode ser classificada em modelagem de desempenho e medição de desempenho (JOHN; EECKHOUT, 2006), a modelagem de desempenho é tipicamente dividida em simulação e modelagem analítica.

Assim, a avaliação de sistemas computacionais (JAIN, 1991) pode ser feita de três maneiras diferentes: modelagem analítica, simulação ou medições. A modelagem de desempenho é tipicamente utilizada em estágios anteriores ao processo de projeto, quando os sistemas ainda não estão disponíveis. Desta forma, medições só podem ser efetuadas se algum sistema similar já estiver implementado, mas como não há nenhum protótipo disponível para as memórias em avaliação e pretendemos avaliar as arquiteturas em um nível anterior ao projeto do sistema, apenas o modelo analítico ou a simulação são alternativas válidas.

Em sistemas computacionais como subsistemas de memória, onde existem muitas variáveis, a complexidade é muito alta sendo difícil criar modelos analíticos que representem corretamente o sistema e, ainda assim, modelos analíticos, nesses casos, costumam apresentar baixa precisão.

Assim, simulação se torna a ferramenta mais apropriada para estimar e comparar as características das memórias propostas, mantendo uma boa precisão e boa generalização dos resultados obtidos.

3.2.1 Simulador Simics

O ambiente de simulação utilizado foi o Simics da Virtutech AB (MAGNUSSON et al., 2002), o qual foi escolhido por ser um simulador completo de sistema no nível de conjunto de instruções. Assim, os resultados de tempo de execução são dados basicamente em instruções e ciclos. O número de ciclos é dado pelo número de instruções executadas mais os ciclos em espera gerados pelas latências de todos os componentes modelados.

Uma vez que o simulador Simics não modela todos os componentes de um *hardware* real mas sim o nível de instruções, normalmente a execução no simulador é de uma instrução por ciclo (parâmetro pré-definido pelo IPC), enquanto uma máquina real consegue gerenciar a execução de mais instruções utilizando superescalaridade e outros componentes de desempenho. Mais que isso, diversos componentes como barramentos, pré-busca e comportamentos do sistema, como gargalos de acesso a recursos, não são totalmente simulados. Fica claro que o objetivo do simulador Simics é simular máquinas no nível de conjunto de instruções e não efetuar uma simulação exata de todos dispositivos.

Desta maneira, a simulação não é adequada para a comparação direta entre máquina real e máquina simulada, uma vez que a diferença entre máquina real e simulação de-

pende de diversos fatores, dessa forma haverá diferença de tempos entre os sistemas real e simulado. Entretanto, o simulador é bastante útil para a comparação de cargas de trabalho em diferentes configurações modeladas dentro do próprio simulador, uma vez que as tendências ocorridas nas simulações devem se repetir em sistemas reais. Logo, o simulador Simics fornece um ambiente controlado e com determinismo controlado, propício para avaliação de sistemas computacionais futuros (SIMONG, 2007).

Para simulação de memória *cache*, o Simics provê ferramentas para configurar dispositivos bastante flexíveis. Em uma simulação de memória *cache*, o Simics permite tanto observar informações a respeito de tempo, como a respeito das informações contidas na memória. O simulador possui dois modelos pré-modelados de memórias *cache*: g-cached e g-cached-ooo:

- O modelo g-cached fornece todas as condições para modelagem de uma memória *cache* ligada a um processador executando as instruções em ordem e fornecendo relatórios sobre as atividades realizadas.
- O modelo g-cached-ooo além das funcionalidades apresentadas pela g-cache, provê ainda, a possibilidade de ser utilizada em simulações de processamento fora de ordem (OOO - *Out Of Order*).

Considerando que o principal objeto de estudo é o sistema de memória *cache*, optou-se por não utilizar o modelo g-cache-ooo, uma vez que este modo levaria ao aumento significativo de complexidade de simulação do processador o que acarretaria a problemas de restrição de tempo, tornando proibitiva as simulações de aplicativos inteiros.

Um exemplo de modelagem de um processador com dois núcleos de processamento com memórias *cache* de primeiro e segundo nível privadas para cada um dos núcleos é ilustrado na Figura 3.2.

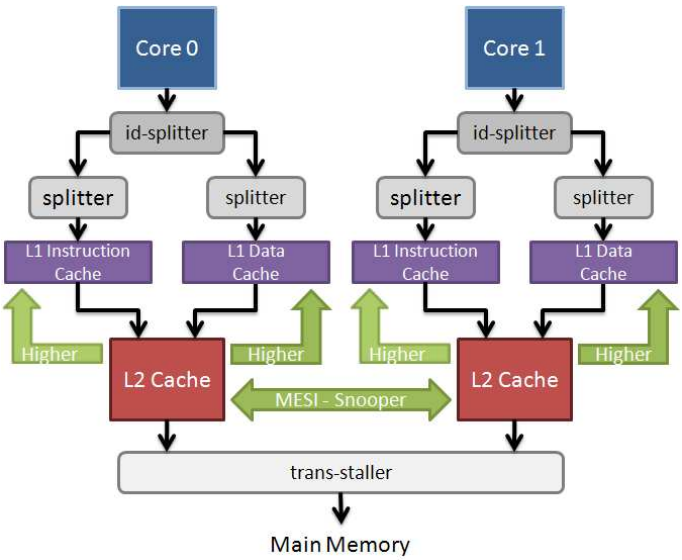


Figura 3.2: Diagrama de modelagem de um *chip multi-core*, adaptado de (VIRTUTECH SIMICS, 2007).

Na Figura 3.2 podemos ver os seguintes componentes internos do Simics responsáveis pela correta simulação da memória *cache*:

- Id-splitter - Utilizado para fazer a separação entre instruções e dados para a memória *cache* correta.
- Splitter - Módulo responsável em particionar os dados, afim de só alocar a quantidade de dados coerente com o tamanho de memória.
- Trans-staller - Dispositivo simples, que simula a latência da memória.

Além dos componentes inerentes à modelagem da memória *cache* no simulador, podemos ver os seguintes componentes referentes ao modelo de controle de coerência de estado das memória *cache* privadas:

- MESI Snooper - Componente definido para indicar quais memórias *cache* de mesmo nível devem ser cuidadas pelo *snooper* a fim de manter coerência.
- Higher Level - Indica ao módulo de coerência do simulador quais memórias *cache*, por exemplo *cache* L1, estão em níveis superiores e que se comunicarão diretamente com cada memória *cache*, por exemplo *cache* L2.

As interconexões utilizadas no simulador são transparentes, assim, pode-se considerar que as conexões entre os componentes como processador e memória *cache* são ponto a ponto. Entretanto, para modelagem de largura de banda, podemos definir uma latência para a interconexão, tornando a modelagem mais precisa.

Seguindo este exemplo de modelagem, os experimentos descritos na próxima seção foram modelados, fazendo as devidas modificações na organização e parâmetros das memórias *cache* de acordo com as características a serem estudadas e avaliadas.

Uma vez que o simulador suporta diversos tipos de máquinas e processadores, segue abaixo uma lista de máquinas e processadores suportados pelo Simics até versão 3.0.30:

- AlphaPC 164LX: Estação de trabalho AlphaPC 164LX monoprocessada com o processador Alpha 21164, oferece suporte nativo ao Linux RedHat 6.0 *kernel* 2.2.5. Apresenta limitações de número de processadores e problemas de arredondamento de ponto flutuante.
- ARM SA1110: Modela um sistema monoprocessado com processador ARMv5 (Intel StrongARM), oferece suporte nativo ao Linux *kernel* 2.4.12. Não apresenta dispositivos de armazenamento modelados, não suporta diversos componentes arquiteturais como hierarquia de memória, MMU (*Memory Management Unit*), unidades de ponto flutuante, entre outras.
- Ebony: Modela a placa de avaliação IBM/AMCC PPC440GP conhecida como Ebony, inclui suporte ao processador PowerPC PPC440GP, funciona com Linux, VxWorks e OSE (*Operating System Embedded*), porém só oferece binários Linux. Oferece suporte a geração de *cluster* de máquinas e memória *cache*, porém o suporte a memória *cache* não é modelado de forma tão robusta como em outras arquiteturas, como x86. Não oferece suporte a arquiteturas *multi-core*. Apresenta diversas instruções não implementadas, e nesses casos o simulador deverá parar a simulação e esperar ação do usuário.
- Fiesta: Modela a estação de trabalho Sun Blade 1500, suporta monoprocessadores baseados em UltraSPARC IIIi, oferece suporte a instalação automática dos sistemas operacionais Solaris 8, 9 e 10. Não oferece suporte a modelagem de *multi-core*.

- IA-64 460GX: Modela máquinas VLIW (*Very Long Instruction Word*) baseadas no processador EPIC (*Explicitly Parallel Instruction Computing*) Itanium da família Intel Itanium, também conhecido como IPF (*Itanium Processor Family*). As máquinas são baseadas no *chipset* Intel 460GX e podem ser configuradas com até 32 processadores. Apenas o sistema operacional Linux é suportado nativamente, oferecendo suporte ao Linux Red Hat 7.1, com *kernel* 2.4.7. Apresenta as limitações: modo IA32 (*Intel Architecture 32 bits*) não implementado, a especulação de dados causa falha na simulação, partes do conjunto de instruções ainda não está implementada e não oferece suporte a latências de componentes.

- Leon2: Modela o processador LEON2 SPARC V8, suporta sistemas operacionais de tempo real (RTEMS - *Real-Time Executive for Multiprocessor Systems*). Atualmente, não é possível conectar componentes de memória *cache* e MMU. Não suporta múltiplos núcleos de processamento.

- Malta/MIPS4kc: Modela a placa de referência Malta para tecnologias MIPS, incluindo suporte a processadores MIPS 4Kc e MIPS 5Kc. Suporta nativamente uma distribuição Linux para embarcados. Não suporta múltiplos núcleos de processamento.

- Niagara: O servidor Sun Fire T2000 é modelado nesta máquina com um processador UltraSPARC T1, suportando apenas o *console* e dispositivos RTC (*Real Time Control*), e não possui nenhum outro componente modelado. Porém, os componentes para outras máquinas Sparc podem funcionar. Oferece suporte de ativação para qualquer número dentre os 32 núcleos virtuais de processamento. Não suporta nativamente múltiplos processadores trabalhando como apenas um *multi-core*, e existem registradores ainda não implementados.

- PM/PPC: A placa Artesyn PM/PPC é modelada incluindo um processador PowerPC 750. Os sistemas operacionais Linux e VxWorks funcionam com esta máquina, porém, apenas os binários do Linux estão disponíveis. O suporte à memória *cache* não é modelado de forma tão robusta. Apresenta limitações conhecidas para compilar novos *kernels*, pontos flutuantes são modelados como aproximações, não apresenta suporte nativo à *multi-core*.

- Simple PPC64: Modela um sistema PPC64 simples com um processador PPC970FX. Oferece suporte ao sistema operacional Linux. Não oferece suporte a *multi-core* e a modelagem de memória *cache* não é detalhada.

- Simple PPC: Esta máquina oferece os processadores PowerPC 603e e PowerPC 440GP. Não apresentando suporte nativo à *multi-core* e a modelagem de memórias *cache* não é robusta como em outras arquiteturas.

- Serengeti: As classes de servidor Sun Fire 3800 - 6800 são modeladas, oferecendo suporte nativo para até 24 processadores UltraSPARC III, UltraSPARC III Cu (III+), UltraSPARC IV, ou UltraSPARC IV+. Oferece diversos componentes PCI modelados. Esta máquina suporta apenas sistemas operacionais Solaris, oferecendo *scripts* para instalação do Solaris 8, 9 e 10.

- SunFire: Modela servidores da classe Sun Enterprise 3500 - 6500, com suporte nativo para até 30 processadores UltraSPARC II. Sistemas operacionais Linux e

Solaris são suportados por esta máquina e também oferece *scripts* para instalação do Solaris 8, 9 e 10.

- x86-440BX: Pode modelar vários sistemas com processadores x86 ou AMD64 baseado no *chipset* 440BX. Suporta os sistemas operacionais Windows, Linux e NetBSD. A BIOS customizada oferece suporte para até 15 processadores, porém, diversas versões de Linux são limitadas a 8 processadores, versões de Windows oferecem suportes variados a processadores dependendo da versão do sistema operacional.

A escolha pela máquina e arquitetura dos processadores a serem simulados foi feita considerando os diversos fatores como: Sistema de propósito geral; Suporte a *multi-core*; Suporte a ponto-flutuante; Suporte avançado à memória *cache* e; Suporte a ilimitados núcleos de processamento. Desta maneira, apenas os sistemas Serengeti, SunFire e x86-440BX apresentam as características necessárias para nossos experimentos.

Após efetuar testes na máquina x86-440BX, foi verificado que não há condições de executar sistemas operacionais com mais de 8 núcleos de processamento, seja por problemas na compilação do *kernel*, seja por problemas internos no simulador.

Avaliando as máquinas SunFire e Serengeti, foi verificada a possibilidade de suporte a até 256 processadores na máquina Serengeti. Comparando as duas máquinas, pode-se verificar que a máquina Serengeti representa máquinas mais recentes, além disso, componentes especializados como GEMS (*General Execution Model Simulator*) formado por módulos especializados de simulação de memória *cache* (Ruby) e de simulação de processador (Opal), que fornecem amplo suporte à máquina Serengeti. Assim, foi escolhida a máquina Serengeti para nossos experimentos, utilizando o modelo de processador UltraSPARC III+ para modelar os núcleos de processamento.

3.3 Definição dos Parâmetros, Fatores e Níveis

Para a avaliação de um sistema computacional devem ser considerados os parâmetros que afetam o desempenho final do sistema. Estes parâmetros podem ser do sistema a ser avaliado e da carga de trabalho utilizada. Após definir os parâmetros que podem influenciar no desempenho, é necessário escolher quais parâmetros serão avaliados, ou seja, quais parâmetros irão sofrer mudanças durante os testes. Os parâmetros a serem variados durante os experimento são chamados de fatores.

Um determinado fator deve ter no mínimo dois níveis, ou seja, dois possíveis valores para aquele fator. Assim, após escolher os fatores, é necessário escolher dentre os diversos níveis quais serão utilizados para cada fator. Embora possam existir muitos parâmetros para um sistema, a lista de fatores deve priorizar os fatores mais representativos.

A escolha por níveis de fatores muitas vezes é infinita, ou seja, os fatores podem aceitar tantos valores quanto se desejar, assim, é inviável a busca pela melhor configuração testando todos os possíveis níveis de um fator. Assim, uma boa estratégia (JAIN, 1991) é começar com poucos fatores e poucos níveis e então ampliar o escopo de acordo com a necessidade e conveniência.

3.3.1 Definição dos Parâmetros

Para a definição dos parâmetros fixos, a primeira etapa foi a definição geral do sistema a ser avaliado, abstraindo detalhes de implementação. Dessa maneira, o primeiro passo foi a escolha do número de processadores e núcleos desses processadores a serem simulados.

Como o objetivo principal é focado na avaliação de desempenho em CMP, foi escolhido apenas um processador a ser simulado, de forma que um maior número de processadores poderia aumentar a complexidade do sistema e assim dificultar a análise dos efeitos da memória *cache*.

Após definido que no sistema haveria apenas um processador, fez-se necessária a escolha do número de núcleos desse processador. Assim, considerando que este estudo deve gerar informações para apontar o futuro em *multi-core* e ou *many-core*, a quantidade de núcleos a ser simulada deve ser maior que o número de núcleos dos atuais processadores. Logo, o sistema avaliado deve ter mais que os 8 núcleos do processador Niagara e/ou Victoria Falls. Assim, seguindo o padrão de aumento de núcleos dos processadores comerciais (potência de dois) um dos valores $2^k, k = \{3, 4, 5, 6, ..., n\}$ deve ser escolhido.

Para a escolha do número de núcleos foi considerado o tempo de simulação estimado, para que assim pudesse ser escolhida a melhor configuração do sistema a ser simulado. Podemos estimar o tempo para execução (MARTY et al., 2005) de aplicativos no simulador Simics através da equação: $TempoSimulacao = TempoReal \cdot 380 \cdot NumProcs$. Assim, considerando 2^k, $k = \{3, 4, 5, 6, ..., n\}$ núcleos, para simular uma carga de trabalho de 15 segundos levará: 13h para $k = \{3\}$, 26h para $k = \{4\}$, 51h para $k = \{5\}$ e 102h para $k = \{6\}$ e assim por diante. Estimando que serão eleitos 5 fatores combinados com k diferentes organizações de memória *cache*, multiplicado por 6 repetições de execução dos experimentos, pode-se calcular o tempo: 1170h (49d), 3120h (130d), 7650h (319d) e 18360h (765d) respectivamente para $k = \{3, 4, 5, 6\}$. Mesmo utilizando diversas máquinas executando os experimentos em paralelo, somente as execuções de 2^k núcleos de processamento, com $k = \{3, 4, 5\}$, são viáveis. Por isso, foi escolhido simular 32 (2^5) núcleos de processamento.

Após definir o primeiro parâmetro, os demais foram escolhidos baseados no processador Intel Clovertown, que é um processador de quatro-núcleos de processamento do estado da arte e que representa uma configuração mediana de processadores de diversas empresas no que diz respeito a tecnologia de fabricação e parâmetros de memória *cache* L1. Uma lista de parâmetros fixos e seus respectivos valores a serem modelados em nossos experimento é apresentado na Tabela 3.1.

As tecnologias de integração consideradas para todas as modelagens foi a tecnologia de 45 nm para dentro do *chip*, assim todos níveis de memória *cache* foram modeladas com 45 nm. Para a modelagem da memória principal considerou-se a tecnologia de integração de 65 nm. Essas tecnologias de integração refletem os processadores comerciais da Intel Clovertown, dessa modo, esses números foram adotados no estudo por refletir tecnologias atuais e por serem de fácil modelagem e de ampla distribuição de materiais de consulta.

A freqüência de operação é utilizada para converter os tempos de acesso das memórias em termos de ns para ciclos de relógio que são utilizadas na simulação. Neste estudo, todas freqüências de operação do processador foram definidas como 2 GHz. Para o caso da memória principal, o tempo de acesso foi convertido para ciclos considerando a freqüência do processador. Desta maneira, a freqüência de operação externa ao processador é considerada a ideal e não causa grande influência nas simulações.

Para as latências de acesso à memória *cache* L1 foram definidos valores para a memória de dados e memória de instruções. No caso das instruções, considerando que todos processadores atuais CMP utilizam técnicas de pré-busca ou técnicas de múltiplas *threads* para esconder as latências de acesso, é justo que para instruções que estivessem na memória *cache* L1 tivessem tempo de acesso nulo para as simulações. Já no caso do acesso a dados da memória *cache* L1, as latências modeladas foram as obtidas pelo Cacti.

Tabela 3.1: Tabela de parâmetros fixos

Sistema Operacional	Tipo	Solaris
	Versão	10.0
Núcleo de Processamento	IPC	1.0
	Pipeline	Não modelado
	Número de núcleos	32 Núcleos
	Modelo dos núcleos	UltraSparc III+
	Conjunto de instruções	Sparc V9
	Freqüência do relógio	2 GHz
	Tecnologia de fabricação	45 nm
Interconexão	Largura de banda	Palavra da memória *cache*
	Vazão de dados	Palavra da memória *cache*
	Tipo	Ponto a ponto (transparente)
	Latência	2 Ciclos
Memória *Cache* L1	Modo de mapeamento de dados	Conjunto associativo
	Política de substituição	LRU
	Política de escrita	*Write-through*
	Tecnologia de fabricação	45 nm
	Tamanho da memória	32 KB (instruções) + 32 KB (dados)
	Latência de acesso a instruções	0 ns
	Ciclos de latência a instruções	0 Ciclos
	Latência de acesso a dados	0,74 ns
	Ciclos de latência a dados	2 Ciclos
	Organização (instruções)	1 Banco por núcleo
	Organização (dados)	1 Banco por núcleo
	Associatividade	2 *Ways set-associative*
	Tamanho da linha	32 Bytes
	Energia Dinâmica (nJ)	0,038
	Potência Estática (W)	0,024
	Área Ocupada (mm^2)	0,228
Memória *Cache* L2	Latência de acesso	Modelado pelo Cacti
	Protocolo de coerência de dados	MESI baseado em *snooping*
	Modo de mapeamento de dados	Conjunto associativo
	Política de substituição	LRU
	Política de escrita	*Write-through / Write-back*
	Tecnologia de fabricação	45 nm
Memória *Cache* L3	Latência de acesso	Modelado pelo Cacti
	Protocolo de coerência de dados	MESI baseado em *snooping*
	Modo de Mapeamento de Dados	Conjunto associativo
	Política de Substituição	LRU
	Política de Escrita	*Write-back*
	Tecnologia de Fabricação	45 nm
	Associatividade	16 *Ways set-associative*
	Tamanho da Linha	64 KB
Memória Principal	Tamanho	1 GB
	Latência de Acesso	38 ns
	Ciclos de Latência	78 Ciclos
	Tecnologia de Fabricação	65 nm
	Energia Dinâmica (nJ)	21,125
	Potência Estática (W)	0,091
	Área Ocupada (mm^2)	739,540

3.3.2 Validação dos Parâmetros

Para validar a escolha dos parâmetros e verificar a relevância dos mesmos, um experimento teste foi executado. O caso teste (ALVES et al., 2007) teve o seguinte título, "Influência do Compartilhamento de *Cache* L2 em um *Chip* Multiprocessado sob Cargas de Trabalho com Conjuntos de Dados Contíguos e Não Contíguos", e foi publicado no Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD'07).

Neste trabalho preliminar foram avaliados os desempenhos de um sistema modelado com 32 núcleos, onde foram variados os compartilhamentos de memória *cache* mantendo o tamanho fixo por banco de memória *cache* em 1 MB , e avaliado o desempenho para duas implementações (dados contíguos e dados não contíguos) da aplicação Ocean presente no *benchmark* SPLASH-2.

A aplicação Ocean estuda movimentos de grande escala de um oceano. As grades utilizadas pela aplicação são particionadas de forma quadrada e a aplicação é implementada em linguagem C utilizando *threads* (pthreads) para a paralelização da aplicação em 32 fluxos de instruções.

Na primeira implementação (contígua), todos os dados são alocados em uma matriz tridimensional. Dessa forma, o primeiro índice da matriz refere-se ao processador pertencente. Assim, todos os dados ficaram alocados em porções contíguas de memória.

Na segunda implementação, cada conjunto de dados pertencente a um determinado processador é alocado em uma matriz bidimensional, evitando dessa forma que os dados de diversos processadores sejam alocados de forma contígua na memória.

A Figura 3.3 apresenta o *speedup* relativo aos experimentos executados no trabalho preliminar, comparando a primeira organização com as demais organizações de memória *cache*, todas as organizações avaliadas com 32 *threads* em execução, 1 *thread* por núcleo de processamento, adaptado dos valores apresentados em (ALVES et al., 2007).

Com relação aos tempos de execução, notamos que houve uma redução de até 0,93% entre a organização com um núcleo por memória *cache* L2 e 32 núcleos com memória *cache* L2 executando a implementação de dados contíguos. A organização com memórias *cache* privadas foi a de pior desempenho, variando gradativamente até a de melhor desempenho com todos núcleos compartilhando a mesma memória *cache*.

Já na implementação de dados não contíguos houve um comportamento diferente, onde notamos na organização de memória *cache* privada uma diferença de aproximadamente 0,11% em relação à máquina de pior desempenho.

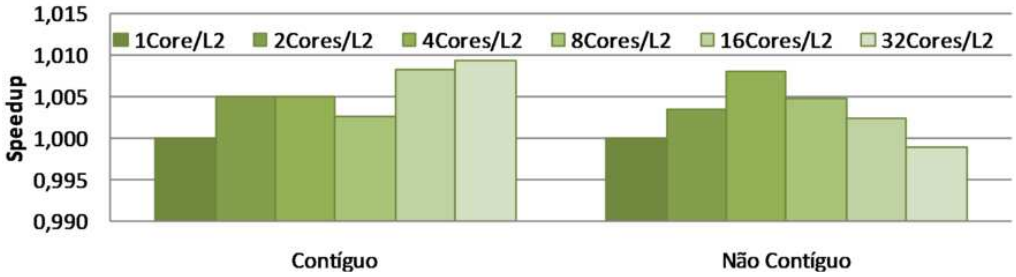


Figura 3.3: Avaliação preliminar do *speedup* da aplicação Ocean sob duas implementações, variando o compartilhamento de memória *cache*, adaptado de (ALVES et al., 2007).

Com estes resultados preliminares pode-se notar a relação entre tipo de aplicação e organização da memória *cache* no desempenho final do sistema. Além disso, nota-

se a relevância dos parâmetros adotados, como o número de núcleos de processamento, justificando a escolha dos mesmos para os experimentos da dissertação, uma vez que esta arquitetura apresenta potencial para avaliação da influência das diferentes organizações de memória *cache*.

3.3.3 Definição dos Fatores

Diversos fatores podem ser avaliados em se tratando de memórias *cache* para *multi-core*. Entretanto, como o principal objetivo da dissertação é a avaliação dos diversos compartilhamentos de memória *cache* a fim de obter melhor desempenho do sistema computacional, o compartilhamento da memória *cache*, gerando diferentes organizações de memória foi o fator principal avaliado nos experimentos propostos, onde o aumento no grau de compartilhamento da memória pode levar a melhora do desempenho das aplicações que utilizam dados compartilhados. A escolha dos demais fatores para este estudo foi baseada nas seis otimizações básicas da memória *cache* para obter melhor desempenho, apresentadas em (HENNESSY; PATTERSON, 2007):

- Tamanho de linha maior para reduzir a taxa de falta de dados. Esse aumento no tamanho da linha da memória *cache* tira vantagem da localidade espacial, reduzindo assim as faltas de dados. Além disso, o tamanho da linha pode reduzir o número de faltas de dados compulsórias.

- Memórias *cache* maiores para reduzir a taxa de falta de dados. Essa otimização obviamente reduz o número de faltas de dados na memória *cache*, reduzindo as faltas por conflito de endereços e de capacidade.

- Altas associatividades para reduzir a taxa de falta de dados. O uso de altas associatividades ataca principalmente as faltas ocorridas por conflitos de endereço.

- Memórias *cache* de múltiplos níveis para reduzir a penalidade durante falta de dados. Com o aumento do nível de integração, as memórias *cache* tem se tornado maiores, levando também a maiores tempos de acesso a dados. O uso de mais níveis na hierarquia visa reduzir a diferença entre processador e memória.

- Prioridade das faltas de leitura sobre as faltas na escrita de dados para reduzir a penalidade durante a falta de dados. Essa técnica é baseada principalmente no uso de *buffer* de escrita de dados, pois essa operação muitas vezes pode ser adiada sem grandes problemas na computação, o que não ocorre no caso de leitura de dados, que ocasiona parada do processador por falta de dados.

- Evitar tradução de endereços durante indexação da memória *cache* para reduzir o tempo de acesso nos acertos de dados, uma vez que as memórias *cache* devem trabalhar em conjunto com a tradução de endereços virtuais do processador para endereços físicos. Ao evitar a tradução de endereços durante o acesso a dados, o sistema perderá menos tempo para acessar os dados requisitados.

Dentre as otimizações básicas propostas, algumas são de difícil implementação e avaliação em um ambiente simulado, assim, apenas quatro das otimizações propostas foram avaliadas: tamanho da linha, aumento da memória *cache*, aumento da associatividade e inclusão de novo nível na hierarquia de memória *cache*. Essas otimizações foram aplicadas aos experimentos, variando o compartilhamento de memória *cache* L2 entre diversos núcleos de processamento.

3.3.4 Definição dos Níveis

Uma vez definidos os fatores a serem avaliados nos experimentos, a escolha por níveis para cada fator é de grande importância no projeto de experimento. Assim, a escolha inicial de níveis para os fatores considerou a importância de cada fator e os possíveis resultados, como segue abaixo:

- Organização da memória *cache*: Este é um dos objetivos principais desse trabalho, por isso, foram escolhidos diversas organizações para a memória *cache*, variando em 2^k o número de núcleos compartilhando a mesma memória *cache* L2, com k valendo 1, 2, 3, 4 e 5, gerando assim, as organizações 1Core/L2, 2Cores/L2, 4Cores/L2, 8Cores/L2, 16Cores/L2 e 32Cores/L2.

- Tamanho da memória *cache* L2: O tamanho da memória *cache* L2 foi planejado para permanecer primeiramente com um tamanho total fixo de 32 MB, para que assim outros fatores possam ser avaliados sem considerar a variação no tamanho da memória *cache*. Em um segundo momento, foi variado o tamanho do banco de memória *cache* L2 compartilhado pelos núcleos, variando entre 1 MB, 2 MB e 4 MB para a avaliar o impacto deste fator no desempenho do sistema. Para a redução no número de níveis possíveis desse fator, foram avaliados somente os tamanhos de memória *cache* inicial, dobrando de tamanho, até chegar ao tamanho de memória *cache* que obteve o melhor desempenho no primeiro experimento.

- Associatividade: A escolha da associatividade base foi considerada a mesma dos processadores Clovertown, ou seja, conjunto de 8 vias associativas. Para a avaliação deste fator, foi duplicado o valor da associatividade e verificado o impacto desse fator no desempenho computacional. Assim, os níveis desse parâmetro utilizados foram conjuntos de 8 e 16 vias associativas. Para esse fator, os níveis definidos foram apenas dois, porém, para reduzir o número de combinações com o primeiro experimento, foram variadas as associatividades da organização inicial até uma organização após obter melhoria no desempenho, tentando assim, aumentar os ganhos de desempenho.

- Tamanho da linha: Para avaliar esse fator, foi definido o tamanho de linha igual a 64 B baseado no processador Clovertown. Assim como a associatividade, para a avaliação desse fator, foram definidos os níveis 64 B e 128 B para o tamanho da linha da memória *cache* L2. Da mesma forma que o experimento anterior, nesse experimento foi variado o tamanho da linha das memórias *cache* partindo da organização inicial até uma organização após a que obteve melhoria no desempenho, tentando assim, aumentar os ganhos de desempenho.

- Níveis na hierarquia de memória *cache*: Para o estudo do impacto do uso de um nível adicional na hierarquia de memória *cache*, foi adicionado o terceiro nível (L3) de memória *cache* ao experimento base. Porém, para a redução de combinações possíveis com o primeiro experimento, fixou-se a organização do primeiro experimento que obteve melhor desempenho, e sobre essa organização foi adicionado mais um nível de memória *cache*. Ainda com este nível de memória *cache* adicional, foram variadas as organizações da memória *cache* L3, a fim de avaliar os possíveis impactos desse novo nível no desempenho final do sistema.

A Tabela 3.2 apresenta um resumo dos fatores e níveis relacionados ao sistema de memória *cache* a serem avaliados neste estudo, lembrando que esses fatores podem também impactar no tempo de acesso a dados da memória *cache* modelada com tais parâmetros, o que será discutido nas próximas seções.

Tabela 3.2: Tabela de fatores e níveis de variação.

Memória *Cache* L2	Organização	1, 2, 4, 8, 16 e 32 núcleos por *cache* L2
	Latência de Acesso	Definido através da modelagem da memória *cache*
	Tamanho da Memória	1 MB, 2 MB, 4 MB, 8 MB, 16 MB e 32 MB por banco
	Associatividade	8 e 16 *Ways Set-Associative*
	Tamanho da Linha	64 KB e 128 KB
Memória *Cache* L3	Tamanho da Memória	16 MB, 32 MB e 64 MB por banco
	Latência de Acesso	Definido através da modelagem da memória *cache*
	Organização	1, 2 e 4 memórias *cache* L2 por memória *cache* L3

3.4 Proposta e Modelagem dos Experimentos

Após definir os parâmetros, fatores e níveis a serem avaliados através de simulação, o próximo passo é a definição e modelagem dos experimentos. Como o simulador trabalha em nível de conjunto de instruções, as penalidades para os diversos componentes simulados são definidos em termos de ciclo de relógio. Desta maneira, uma latência de leitura de dados deve ser modelada em termos de ciclos, considerando a freqüência que se deseja modelar e não em termos de tempo de acesso.

Uma importante ferramenta para estimar parâmetros de memória *cache* e memória principal é o *software* Cacti (THOZIYOOR et al., 2008). Neste estudo, todas as latências foram modeladas considerando as estimativas dessa ferramenta. Para que se possa modelar uma memória corretamente no Cacti, foi considerada a tecnologia de integração, freqüência de operação e outros parâmetros definidos para os experimentos.

Para a modelagem na ferramenta, algumas configurações adicionais devem ser preenchidas. Para configuração de memória *cache* foi utilizada a interface básica (*Normal Interface*) para modelagem, já para modelar a memória principal foi utilizado a interface chamada *Pure RAM Interface*. As configurações para a interface de memória cache são apresentadas na Tabela 3.3, e as configurações da memória principal estão presentes na Tabela 3.3. Tais parâmetros foram validados comparando os resultados obtidos com valores típicos de um sistema (THOZIYOOR et al., 2008), constatando que os valores de tempo de acesso, área e potência obtidos são próximos de componentes reais.

O projeto dos experimentos foi sendo desenvolvido assim que resultados eram gerados, dessa maneira os experimentos foram planejados ao longo da execução do projeto. Entretanto, para um fácil entendimento, as próximas subseções trarão todos os experimentos projetados, para que após os resultados serem apresentados fiquem mais claros os motivos de cada decisão tomada.

Considerando que o simulador Simics não suporta a simulação de contenções que ocorreriam pelo uso de número limitado de portas das memórias, os experimentos foram divididos em duas etapas. Primeiramente (experimentos 1, 2, 3, 4 e 5), todas arquiteturas propostas foram modeladas com ilimitado número de portas, com as latências, consumo de potência e ocupação de área modeladas pelo Cacti considerando apenas uma porta

Tabela 3.3: Descrição da modelagem de memória *cache* na ferramenta Cacti.

Memória *Cache*	
Cache Size	Definido pelo Experimento
Line Size (bytes)	Definido pelo Experimento
Associativity	Definido pelo Experimento
Nr. of Banks	1
Technology Node (nm)	45 nm
Read/Write Ports	0
Read Ports	Definido pelo Experimento
Write Ports	Definido pelo Experimento
Single Ended Read Ports	0
Nr. of Bits Read Out	256
Change Default Cacti Tag	*Yes*
Nr. of Bits per Tag	34
Type of cache *(Normal/Serial/Fast)*	*Normal*
Temperature (300-400 K)	360
RAM cell/transistor type in data array	ITRS-HP (*High Performance*)
Peripheral and global circuitry transistor type in data array	ITRS-HP (*High Performance*)
RAM cell/transistor type in tag array	ITRS-HP (*High Performance*)
Peripheral and global circuitry transistor type in tag array	ITRS-HP (*High Performance*)
Interconnect projection type	*Aggressive*
Type of wire outside mat	*Global*

Tabela 3.4: Descrição da modelagem da memória principal na ferramenta Cacti.

Memória Principal	
RAM Size (bytes)	1.073.741.824 (1 GB)
Nr. of Banks	1
Read/Write Ports	1
Read Ports	0
Write Ports	0
Single Ended Read Ports	0
Nr. of Bits Read Out	512 (64 bytes)
Technology Node (nm)	65 nm
Temperature (300-400 K)	360 K
RAM cell/transistor type in data array	*COMM-DRAM*
Peripheral and global circuit transistor type in data array	ITRS-LSTP (*Low Standby Power*)
RAM cell/transistor type in tag array	COMM-DRAM
Peripheral and global circuit transistor type in tag array	ITRS-LSTP (*Low Standby Power*)
Interconnect projetion type	*Conservative*
Type of wire outside mat	*Semi-Global*

de leitura e escrita. Assim, a primeira etapa foi modelada a melhor situação possível usando o compartilhamento de memória *cache*. Este modelo é necessário considerando as diversas formas de se otimizar os acessos às memórias *cache* que além de não serem disponibilizadas pelos fabricantes também não são suportadas pelo Simics. Portanto, essa escolha visa modelar a arquitetura para um ponto ótimo de funcionamento em função das otimizações que são implementadas durante a fabricação de processadores.

Então, na segunda etapa de avaliação (experimento 6) consiste na modelagem analítica das contenções geradas por limitação na quantidade de portas da memória *cache*, utilizando como dados de entrada os valores obtidos na etapa anterior para modelagem analítica. Além disso, a modelagem analítica será alimentada com dados modelados no Cacti para geração de latências, consumo de potência e ocupação de área relativas a quantidade de portas a ser modelada.

3.4.1 Experimento 1 - Compartilhamento da Memória *Cache*

O primeiro experimento, que serviu como base para os demais experimentos, é o que avaliou o compartilhamento da memória *cache* L2. Neste experimento foi fixado o tamanho total de memória *cache* em 32 MB, e a partir da configuração de um núcleo de processamento por memória *cache*, foi variando o número de núcleos compartilhando o mesmo banco de memória *cache*, como é ilustrado na Figura 3.4

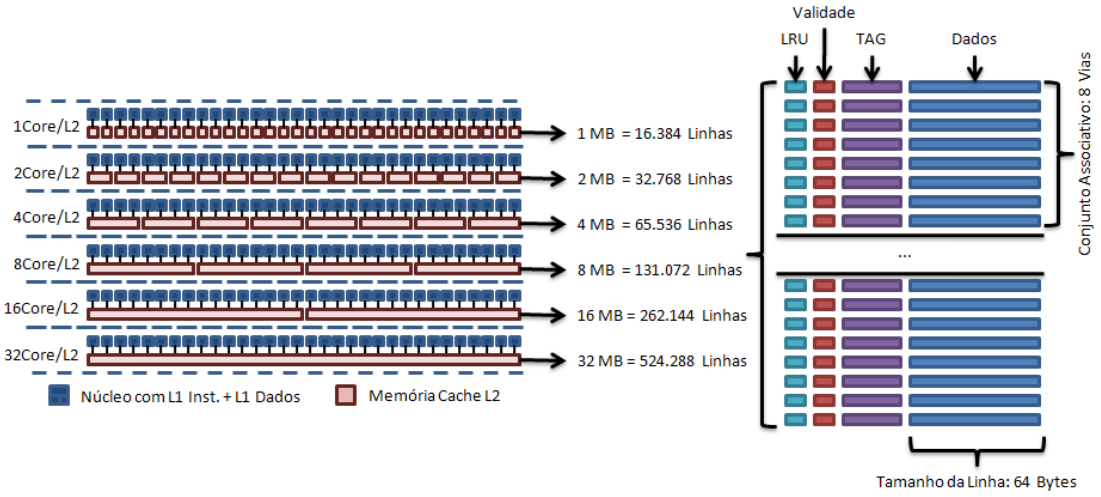


Figura 3.4: Experimento 1 - Compartilhamento da memória *cache* L2.

Este primeiro experimento visa avaliar qual o comportamento do sistema ao agregar mais núcleos compartilhando o mesmo banco de memória *cache*, onde o tamanho total de memória *cache* é mantido. Para que este experimento represente as condições mais próximas da real, o parâmetros das memórias *cache* em todos experimentos foram obtidas pelo Cacti, a Tabela 3.5 apresenta os níveis dos fatores modelados nesse primeiro experimento. A partir deste primeiro experimento, todos os outros experimentos foram planejados, a fim de avaliar outros fatores partindo dos pontos que apresentaram baixo desempenho. Por isso, este é o experimento base para os demais.

Tabela 3.5: Descrição da modelagem da memória *cache* L2 do primeiro experimento.

Tamanho por Banco	1 MB	2 MB	4 MB	8 MB	16 MB	32 MB
Tamanho de Linha	64 B	64 B	64 B	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways	8 Ways	8 Ways	8 Ways
Latência de Acesso	1,77 ns	2,16 ns	2,70 ns	3,78 ns	4,82 ns	7,02 ns
Ciclos de Latência	4 Ciclos	5 Ciclos	6 Ciclos	8 Ciclos	10 Ciclos	15 Ciclos
Energia Dinâmica (nJ)	0,690	0,648	1,008	1,417	2,387	3,602
Potência Estática (W)	0,361	0,648	1,344	2,537	5,473	10,238
Área Ocupada (mm^2)	7,307	10,958	24,528	38,850	81,773	183,353
Portas de Leitura/Escrita	1	1	1	1	1	1

3.4.2 Experimento 2 - Tamanho da Memória *Cache*

O segundo experimento baseia-se no experimento base (primeiro experimento), nesse experimento foram variados os compartilhamentos da memória *cache* L2, variando o tamanho da memória *cache*. Para isso, ao invés de fixar o tamanho total de memória *cache* do sistema, fixou-se o tamanho da memória de cada banco de memória *cache* em 1 MB, 2 MB e 4 MB, ou seja, para cada organização de compartilhamento de memória *cache* L2 igual do primeiro experimento, foram feitos três novos testes com bancos de tamanhos variados, como pode ser visto na Figura 3.5.

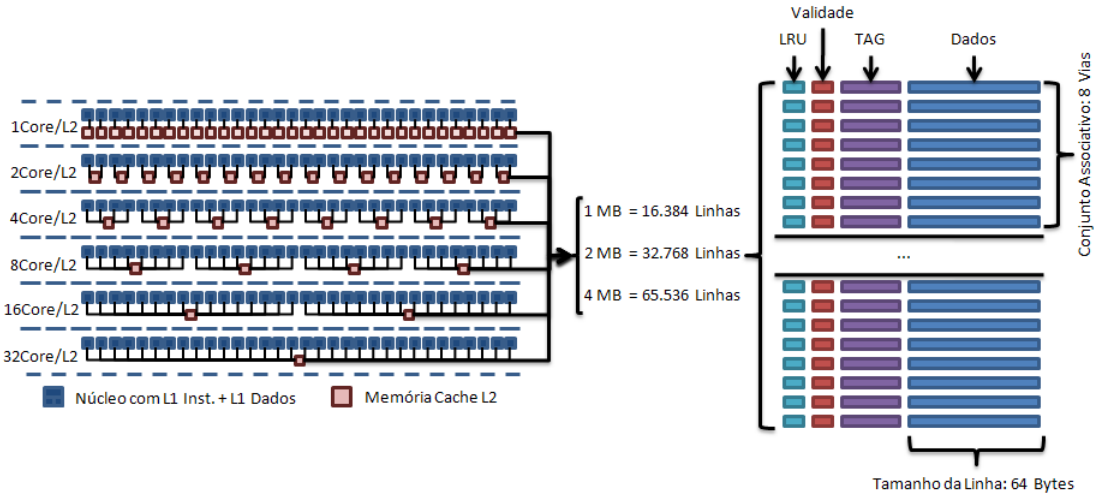


Figura 3.5: Experimento 2 - Tamanho da memória *cache* L2.

Nesse experimento, a variação do tamanho da memória *cache* visa estimar os ganhos que essa técnica pode trazer reduzindo o número de faltas de dados, porém, também se considera que ao aumentar o tamanho da memória *cache*, o tempo de acesso a dados deverá mudar, assim os valores modelados são apresentados na Tabela 3.6 que mostra os níveis dos fatores modelados. Note que os parâmetros da primeira organização 1Core/L2 presente nesse segundo experimento é idêntica a primeira organização do primeiro experimento.

Tabela 3.6: Descrição da modelagem da memória *cache* L2 do segundo experimento

Tamanho por Banco	1 MB	2 MB	4 MB
Tamanho de Linha	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways
Latência de Acesso	1,77 ns	2,16 ns	2,70 ns
Ciclos de Latência	4 Ciclos	5 Ciclos	6 Ciclos
Energia Dinâmica (nJ)	0,690	0,648	1,008
Potência Estática (W)	0,361	0,648	1,344
Área Ocupada (mm^2)	7,307	10,958	24,528
Portas de Leitura/Escrita	1	1	1

3.4.3 Experimento 3 - Associatividade

Considerando o experimento base, nesse experimento foi dobrada a associatividade do sistema a fim de reduzir o número de faltas de dados. Logo, esse experimento visa avaliar a influência da associatividade no desempenho do sistema de memória *cache* L2.

Uma descrição do experimento está ilustrada na Figura 3.6, onde pode-se ver o aumento de 8 para 16 vias do conjunto associativo de cada banco de memória *cache* L2.

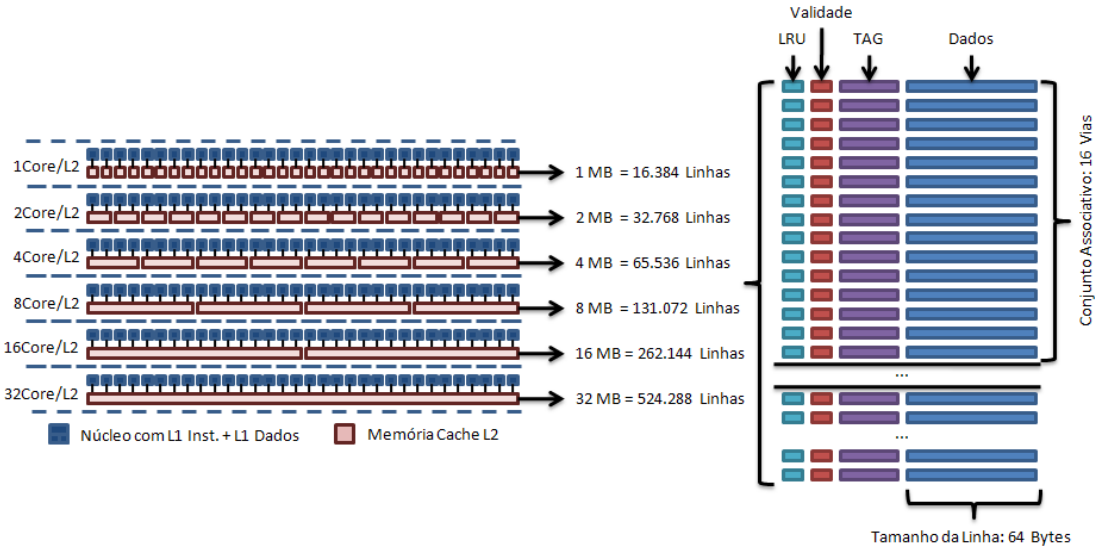


Figura 3.6: Experimento 3 - Associatividade da memória *cache* L2.

Ao aumentar a associatividade das memórias *cache*, espera-se reduzir principalmente o número de falta de dados ocasionados pelo conflito de endereços na memória *cache*, dessa maneira, o aumento da associatividade está relacionado ao aumento da localidade temporal da memória *cache*.

Os valores modelados no terceiro experimento são apresentados na Tabela 3.7 que mostra todos os níveis dos fatores modelados para a memória *cache* L2, além das características dessa memória *cache*.

Tabela 3.7: Descrição da modelagem da memória *cache* L2 do terceiro experimento

Tamanho por Banco	1 MB	2 MB	4 MB	8 MB
Tamanho de Linha	64 B	64 B	64 B	64 B
Associatividade	16 Ways	16 Ways	16 Ways	16 Ways
Latência de Acesso	2,62 ns	2,83 ns	3,08 ns	3,77 ns
Ciclos de Latência	6 Ciclos	6 Ciclos	7 Ciclos	8 Ciclos
Energia Dinâmica (nJ)	0,772	1,417	1,206	1,443
Potência Estática (W)	0,369	0,752	1,295	2,552
Área Ocupada (mm^2)	9,621	14,420	20,806	39,311
Portas de Leitura/Escrita	1	1	1	1

3.4.4 Experimento 4 - Tamanho da Linha

O tamanho da linha é o que define o tamanho da palavra em que a memória *cache* trabalha. Esse experimento avalia a influência do tamanho da linha da memória *cache* L2 no desempenho final do sistema.

Baseado no experimento base, aumentou-se o tamanho da linha de dados de 64 B para 128 B, como pode ser visto no diagrama apresentado na Figura 3.7.

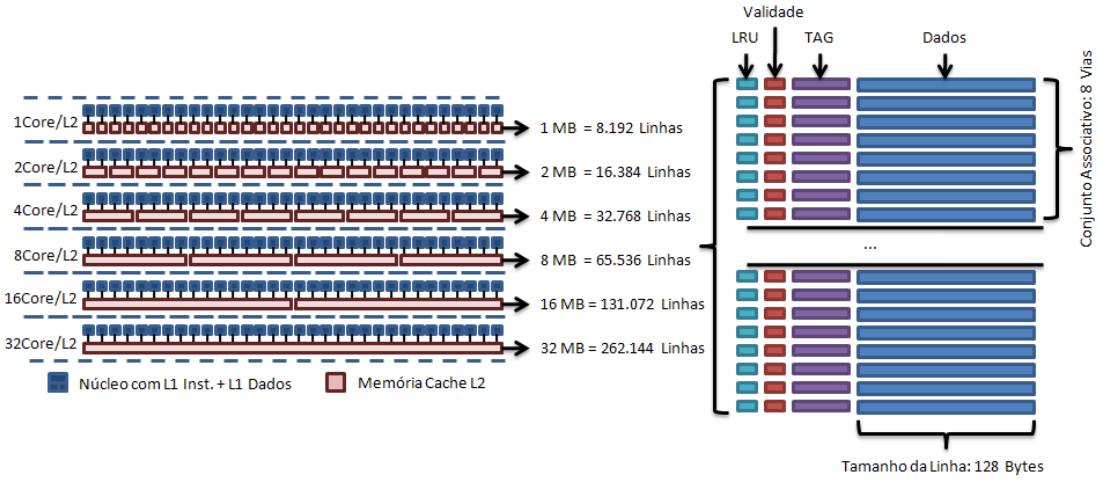


Figura 3.7: Experimento 4 - Tamanho da linha da memória *cache* L2.

Esse experimento tem como objetivo principal o aumento da localidade espacial da memória *cache* L2, para isso, aumentou-se o tamanho da linha da memória *cache* L2. Além disso, o aumento do tamanho da linha de dados pode reduzir algumas faltas de dados compulsórias, uma vez que buscando os primeiros dados, mais dados próximos são trazidos automaticamente proporcionalmente ao tamanho da linha de dados da memória *cache*.

A Tabela 3.8 apresenta os níveis dos fatores da memória *cache* L2 modelados nesse quarto experimento, também estão dispostos nessa tabela as características físicas das memórias modeladas.

Tabela 3.8: Descrição da modelagem da memória *cache* L2 do quarto experimento.

Tamanho por Banco	1 MB	2 MB	4 MB	8 MB
Tamanho de Linha	128 B	128 B	128 B	128 B
Associatividade	8 Ways	8 Ways	8 Ways	8 Ways
Latência de Acesso	2,61 ns	2,75 ns	3,14 ns	3,67 ns
Ciclos de Latência	6 Ciclos	6 Ciclos	7 Ciclos	8 Ciclos
Energia Dinâmica (nJ)	2,383	2,520	1,943	3,442
Potência Estática (W)	0,562	0,854	1,348	2,783
Área Ocupada (mm^2)	16,809	20,741	25,889	56,315
Portas de Leitura/Escrita	1	1	1	1

3.4.5 Experimento 5 - Níveis na Hierarquia

Esse quinto experimento proposto foi projetado para avaliar a influência do nível da hierarquia de memória *cache* no desempenho de um sistema *multi-core*. Ilustrado na Figura 3.8, o experimento aumentou um nível na hierarquia de memória *cache*.

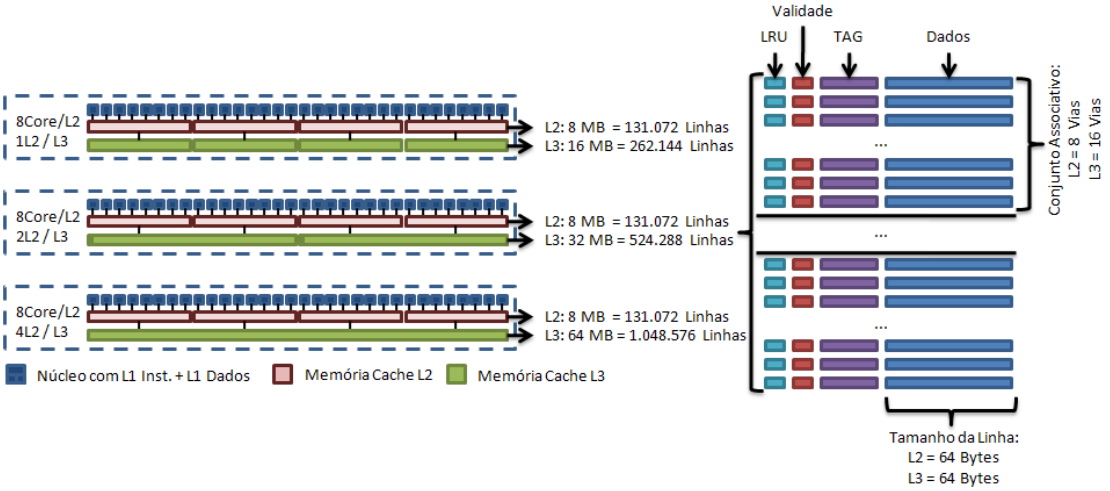


Figura 3.8: Experimento 5 - Níveis na hierarquia de memória *cache*.

O aumento no número de níveis na hierarquia de memória *cache* age reduzindo o tempo de acesso de busca de dados na memória principal, fazendo com que os níveis superiores de memória *cache* e o processador enfrentem menos latência durante faltas de dados. A Tabela 3.9 apresenta os níveis dos fatores modelados e simulados.

3.4.6 Experimento 6 - Contenção e Limitações Físicas da Memória *Cache*

Esse último experimento utiliza dados gerados pelos experimentos anteriores para a avaliação das contenções da memória cache relacionadas à quantidade de portas da memória. Assim, esse experimento avalia as limitações físicas geradas pelas contenções na memória cache.

A quantidade de portas de uma memória influencia no desempenho do sistema de duas formas, primeiramente, a adição de portas reduz as contenções para acesso, liberando o acesso aos dados para mais de uma requisição ao mesmo tempo, porém, esse aumento no número de portas tem implicações físicas, onde o tempo de acesso aos dados

Tabela 3.9: Descrição da modelagem das memórias *cache* L2 e L3 do quinto experimento.

	Cache L2	*Cache* L3		
Tamanho por Banco	8 MB	16 MB	32 MB	64 MB
Tamanho de Linha	64 B	64 B	64 B	64 B
Associatividade	8 Ways	16 Ways	16 Ways	16 Ways
Latência de Acesso	3,78 ns	4,65 ns	6,88 ns	8,76 ns
Ciclos de Latência	8 Ciclos	10 Ciclos	14 Ciclos	18 Ciclos
Energia Dinâmica (nJ)	1,417	2,387	3,602	10,048
Potência Estática (W)	2,537	5,473	10,238	20,985
Área Ocupada (mm^2)	38,850	81,773	183,353	367,330
Portas de Leitura/Escrita	1	1	1	1

será prejudicado conforme a memória apresenta mais portas de comunicação, uma vez que a implementação de múltiplas portas aumenta a latência da memória. Além dessas influências no desempenho, o aumento na quantidade de portas da memória *cache* pode representar um aumento significativo na área ocupada e consumo de potência, sendo que, em alguns casos esses aumentos significam uma limitação física para o circuito.

Como esse sexto experimento é baseado nas medições dos experimentos anteriores, foi definido que apenas as organizações com 1, 2, 3 e 4 núcleos de processamento por memória *cache* L2 seriam avaliadas.

A Tabela 3.10 apresenta os valores modelados para as memórias *cache* com base no primeiro experimento.

Tabela 3.10: Descrição das memórias *cache* L2 do sexto experimento (1)

Experimento 6.1 - Compartilhamento da Memória *Cache*						
Compartilhamento	1Core/L2	2Cores/L2	2Cores/L2	4Cores/L2	4Cores/L2	4Cores/L2
Tamanho por Banco	1 MB	2 MB	2 MB	4 MB	4 MB	4 MB
Tamanho de Linha	64 B	64 B	64 B	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways	8 Ways	8 Ways	8 Ways
Portas de Leitura/Escrita	1	1	2	1	2	4
Latência de Acesso	1,22 ns	1,70 ns	2,41 ns	2,17 ns	3,17 ns	5,17 ns
Ciclos de Latência	3 Ciclos	4 Ciclos	5 Ciclos	5 Ciclos	7 Ciclos	11 Ciclos
Energia Dinâmica (nJ)	0,632	1,039	1,753	1,538	2,413	3,754
Potência Estática (W)	1,504	3,033	4,156	6,073	8,390	14,449
Área Ocupada (mm^2)	9,877	20,853	63,780	42,586	129,856	386,633

Com base nas organizações com memórias *cache* de 1 MB apresentadas no segundo experimento, a Tabela 3.11 apresenta os valores modelados para estimar contenção na quantidade de portas para memória *cache* L2.

A Tabela 3.12 apresenta os valores modelados para a memória *cache* do segundo experimento com memórias *cache* de 2 MB.

Com base no segundo experimento com memória *cache* de 4 MB, a Tabela 3.13 apresenta os valores modelados para memória *cache* L2 com variado número de portas de acesso a dados.

A Tabela 3.14 apresenta os valores para memória *cache*, baseado no terceiro experimento.

Tabela 3.11: Descrição das memórias *cache* L2 do sexto experimento (2), com bancos de memória *cache* de 1 MB

Experimento 6.2 - Tamanho da Memória *Cache* - 1MB			
Tamanho por Banco	1 MB	1 MB	1 MB
Tamanho de Linha	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways
Portas de Leitura/Escrita	1	2	4
Latência de Acesso	1,22 ns	1,83 ns	2,78 ns
Ciclos de Latência	3 Ciclos	4 Ciclos	6 Ciclos
Energia Dinâmica (nJ)	0,632	1,010	1,845
Potência Estática (W)	1,504	2,281	3,996
Área Ocupada (mm^2)	9,877	31,517	109,148

Tabela 3.12: Descrição das memórias *cache* L2 do sexto experimento (2), com bancos de memória *cache* de 2 MB

Experimento 6.2 - Tamanho da Memória *Cache* - 2MB			
Tamanho por Banco	2 MB	2 MB	2MB
Tamanho de Linha	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways
Portas de Leitura/Escrita	1	2	4
Latência de Acesso	1,70 ns	2,41 ns	3,96 ns
Ciclos de Latência	4 Ciclos	5 Ciclos	8 Ciclos
Energia Dinâmica (nJ)	1,039	1,753	2,470
Potência Estática (W)	3,033	4,156	7,163
Área Ocupada (mm^2)	20,853	63,780	192,994

Tabela 3.13: Descrição das memórias *cache* L2 do sexto experimento (2), com bancos de memória *cache* de 4 MB

Experimento 6.2 - Tamanho da Memória *Cache* - 4MB			
Tamanho por Banco	4 MB	4 MB	4 MB
Tamanho de Linha	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways
Portas de Leitura/Escrita	1	2	4
Latência de Acesso	2,17 ns	3,17 ns	5,17 ns
Ciclos de Latência	5 Ciclos	7 Ciclos	11 Ciclos
Energia Dinâmica (nJ)	1,538	2,413	3,754
Potência Estática (W)	6,073	8,390	14,449
Área Ocupada (mm^2)	42,586	129,856	386,633

Por fim, a Tabela 3.15 apresenta os valores baseados no quarto experimento para a memória *cache*.

O quinto experimento não será avaliado quanto a contenções na memória por motivos práticos, uma vez que esse experimento apresentou nos primeiros resultados desempenho muito abaixo que os demais.

Após definir os fatores e níveis para modelagem do sexto experimento, é necessário

Tabela 3.14: Descrição das memórias *cache* L2 do sexto experimento (3)

Experimento 6.3 - Associatividade da Memória *Cache*						
Compartilhamento	1Core/L2	2Cores/L2	2Cores/L2	4Cores/L2	4Cores/L2	4Cores/L2
Tamanho por Banco	1 MB	2 MB	2 MB	4 MB	4 MB	4 MB
Tamanho de Linha	64 B	64 B	64 B	64 B	64 B	64 B
Associatividade	16 Ways	16 Ways	16 Ways	16 Ways	16 Ways	16 Ways
Portas de Leitura/Escrita	1	1	2	1	2	4
Latência de Acesso	1,49 ns	1,66 ns	2,40 ns	2,25 ns	3,15 ns	5,01 ns
Ciclos de Latência	3 Ciclos	4 Ciclos	5 Ciclos	5 Ciclos	7 Ciclos	11 Ciclos
Energia Dinâmica (nJ)	0,661	0,981	1,415	1,609	2,131	3,977
Potência Estática (W)	1,468	3,028	4,083	6,622	8,178	14,196
Área Ocupada (mm^2)	11,531	18,808	53,890	40,937	108,121	366,299

Tabela 3.15: Descrição das memórias *cache* L2 do sexto experimento (4)

Experimento 6.4 - Tamanho da Linha da Memória *Cache*						
Compartilhamento	1Core/L2	2Cores/L2	2Cores/L2	4Cores/L2	4Cores/L2	4Cores/L2
Tamanho por Banco	1 MB	2 MB	2 MB	4 MB	4 MB	4 MB
Tamanho de Linha	64 B	64 B	64 B	64 B	64 B	64 B
Associatividade	8 Ways	8 Ways	8 Ways	8 Ways	8 Ways	8 Ways
Portas de Leitura/Escrita	1	1	2	1	2	4
Latência de Acesso	1,43 ns	1,59 ns	2,42 ns	2,18 ns	3,02 ns	4,99 ns
Ciclos de Latência	3 Ciclos	4 Ciclos	5 Ciclos	5 Ciclos	7 Ciclos	10 Ciclos
Energia Dinâmica (nJ)	1,488	2,109	3,696	3,880	5,894	10,711
Potência Estática (W)	1,478	2,895	4,050	5,901	7,798	14,093
Área Ocupada (mm^2)	10,184	18,102	58,292	39,886	103,614	382,562

modelar analiticamente as contenções geradas pelo número de portas na memória *cache* L2. Para auxiliar no entendimento das fórmulas analíticas, será utilizada a organização descrita na Figura 3.9 como modelo base. O modelo apresenta 8 núcleos de processamento diretamente conectados à memória *cache* L1 com interconexões e duas memórias *cache* L2.

Com base no modelo apresentado, as seguintes fórmulas analíticas serão utilizadas para modelar a contenção relativa às portas da memória *cache* L2. A fórmula básica de modelagem do tempo total de execução (T_{Total}) é dada pela Equação 3.1.

$$T_{Total} = T_{\text{Max\_Inst}} + T_{\text{L2\_Write\_Back}} + T_{\text{L2\_Faltas}} + T_{\text{L1\_Acertos}} + T_{\text{L1\_Faltas}^p} \tag{3.1}$$

O tempo para execução das instruções ($T_{\text{Max\_Inst}}$) considera o i-ésimo processador ($P^i_{\text{Instruções}}$) que executou a maior quantidade de instruções, conforme descrito na Equação 3.2.

$$T_{\text{Max\_Inst}} = Max \left\{ \bigcup_{i=0}^{7} P^i_{\text{Instruções}} \right\} \tag{3.2}$$

O tempo para executar as operações de *write-back* sofridas pela memória *cache* L2 ($T_{\text{L2\_Write\_Back}}$) está disponível na Equação 3.3, onde considera-se a soma de operações

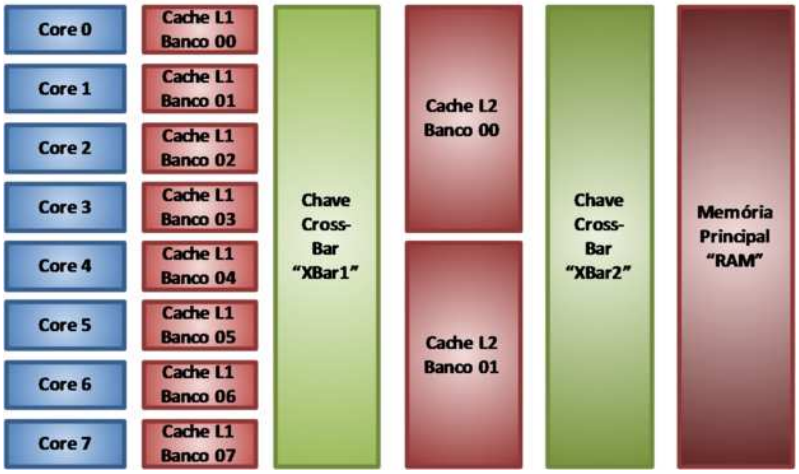


Figura 3.9: Experimento 6 - Modelo base para contenções na memória *cache* L2.

desse tipo de todas as j memórias *cache* L2 ($L2^{j}_{\text{Write\_Back}}$), multiplicada pelas latências do barramento (L_{Xbar2}) e da memória principal (L_{RAM}).

$$T_{\text{L2\_Write\_Back}} = \sum_{j=0}^{1} L2^{j}_{\text{Write\_Back}} \cdot (L_{\text{Xbar2}} + L_{\text{RAM}}) \tag{3.3}$$

O tempo que o sistema levou para acessar a memória principal durante faltas de dados na memória *cache* L2 ($T_{\text{L2\_Faltas}}$) é obtida através da soma de todas as faltas das j memórias *cache* L2 ($L2^{j}_{\text{Faltas}}$), multiplicada pela latência do barramento (L_{Xbar2}) e a latência da memória principal (L_{RAM}), conforme apresentado na Equação 3.4.

$$T_{\text{L2\_Faltas}} = \sum_{j=0}^{1} L2^{j}_{\text{Faltas}} \cdot (L_{\text{Xbar2}} + L_{\text{RAM}}) \tag{3.4}$$

Para se obter o tempo consumido pelos acertos no acesso a dados da memória *cache* L1 ($T_{\text{L1\_Acertos}}$), apresentado na Equação 3.5, consideramos o tempo total gasto para as operações de acerto e acesso a dados na memória *cache* L1, multiplicamos o valor de operações da i-ésima memória *cache* L1 que obteve mais operações com sucesso no acesso a dados ($L1^{i}_{\text{Acertos}}$) pela latência de acesso à memória *cache* L1 (L_{L1}).

$$T_{\text{L1\_Acertos}} = Max \left\{ \bigcup_{i=0}^{7} L1^{i}_{\text{Acertos}} \right\} \cdot (L_{\text{L1}}) \tag{3.5}$$

No cálculo do tempo gasto para as operações de faltas de dados na memória *cache* L1 ($T_{\text{L1\_Faltas}}$) é necessário considerar a quantidade de portas das memórias *cache* L2. Para isso, três fórmulas foram elaboradas a fim de suprir os casos que iríamos ter nos experimentos, com 1 ($T^{1}_{\text{L1\_Faltas}}$), 2 ($T^{1}_{\text{L1\_Faltas}}$) e 4 ($T^{1}_{\text{L1\_Faltas}}$) portas de acesso a dados.

A Equação 3.6 que tráz o cálculo do tempo gasto para as operações de faltas de dados na memória *cache* L1, considerando uma memória *cache* L2 com 1 porta de acesso a dados ($T^{1}_{\text{L1\_Faltas}}$), é obtida somando-se as faltas de dados das memórias *cache* L1 conectadas a cada memória *cache* L2 e então, resgatando o maior somatório. Assim, multiplicando essas operações pelas latências de acesso à memória *cache* L1 (L_{L1}), da chave *cross-bar*

(L_{XBar1}) e da memória *cache* L2 (L_{L2}) é obtido tempo gasto para as operações de faltas de dados na memória *cache* L1.

$$T^{1}_{\text{L1\_Faltas}} = Max \left\{ \sum_{i=0}^{3} L1^{i}_{\text{Faltas}}; \sum_{i=4}^{7} L1^{i}_{\text{Faltas}} \right\} \cdot (L_{\text{L1}} + L_{\text{Xbar1}} + L_{\text{L2}}) \tag{3.6}$$

Para a Equação 3.7, o cálculo do tempo gasto para as operações de faltas de dados na memória *cache* L1, considerando uma memória *cache* L2 com 2 portas de acesso a dados ($T^{2}_{\text{L1\_Faltas}}$), é realizado utilizando a função $F^{n}_{Faltas}(L1_{Ini}, L1_{Fim})$ a qual retorna o n-ésimo maior valor de operações de falta de dados entre as memórias *cache* L1 ($L1_{Ini}$) e ($L1_{Fim}$). Por exemplo, a função $F^{2}(L1^{0}, L1^{3})$ irá retornar entre as memórias *cache* L1 o número de operações da segunda que teve mais faltas de dados. Como a memória *cache* L2 possui 2 portas de acesso a dados, não seria justo colocar as duas memórias *cache* L1 que efetuam mais faltas na mesma porta da memória *cache* L2. Por esse motivo, dividimos em cada uma das memórias *cache* L2, a L1 com maior e menor número de operações de faltas em uma porta e a L1 com segunda maior e terceira maior quantidade de operações de faltas em outra porta de acesso a dados. Após obter o número de faltas total em cada porta de acesso das memórias *cache* L2, basta utilizar a função Max para obter qual porta dominou o número de operações e então multiplicar pelas latências da memória *cache* L1 (L_{L1}) da chave *cross-bar* (L_{Xbar1}) e da memória *cache* L2 (L_{L2}).

$$T^{2}_{\text{L1\_Faltas}} = Max \left\{ \begin{array}{l} (F^{1}_{Faltas}(L1^{0}, L1^{3}) + F^{4}_{Faltas}(L1^{0}, L1^{3})); \\ (F^{2}_{Faltas}(L1^{0}, L1^{3}) + F^{3}_{Faltas}(L1^{0}, L1^{3})); \\ (F^{1}_{Faltas}(L1^{4}, L1^{7}) + F^{4}_{Faltas}(L1^{4}, L1^{7})); \\ (F^{2}_{Faltas}(L1^{4}, L1^{7}) + F^{3}_{Faltas}(L1^{4}, L1^{7})) \end{array} \right\} \cdot (L_{\text{L1}} + L_{\text{Xbar1}} + L_{\text{L2}}) \tag{3.7}$$

Para o cálculo do tempo gasto para as operações de faltas de dados na memória *cache* L1 considerando uma memória *cache* L2 com 4 portas de acesso a dados ($T^{4}_{\text{L1\_Faltas}}$), basta obter a quantidade de faltas da memória *cache* L1 que mais sofreu falta de dados, e multiplicar essa quantidade de operações pelas latências da memória *cache* L1 (L_{L1}) da chave *cross-bar* (L_{Xbar1}) e da memória *cache* L2 (L_{L2}), conforme apresentado na Equação 3.8.

$$T^{4}_{\text{L1\_Faltas}} = Max \left\{ \bigcup_{i=0}^{7} L1^{i}_{\text{Faltas}} \right\} \cdot (L_{\text{L1}} + L_{\text{Xbar1}} + L_{\text{L2}}) \tag{3.8}$$

Seguindo o conceito geral apresentado na formulação analítica, a fórmula foi estendida para o número de núcleos e as organizações de memória *cache* avaliadas.

3.5 Carga de Trabalho

O termo *benchmark* pode ser usado para denotar qualquer tipo de carga de trabalho de teste utilizada em estudos de desempenho, podendo ser sintético, de brinquedo, *kernel* ou real, (JAIN, 1991) (MENASCÉ; ALMEIDA, 2002).

Uma carga de trabalho sintética realiza apenas operações básicas, como multiplicação e adição. Os *benchmarks* de brinquedo são programas muito pequenos, que apresentam problemas clássicos como Torres de Hanói, N-Queens, etc. Os *benchmarks* de *kernel* e reais possuem características semelhantes, onde tenta-se reproduzir as operações normais de um determinado sistema, possuindo ainda a característica de reprodutibilidade. Entretanto os *benchmarks* de *kernel*, como o nome já diz, são aplicações formadas com

apenas o cerne do problema a ser resolvido em uma aplicação real, assim, muitas vezes são eliminadas etapas de inicialização e finalização do programa a fim de obter apenas a parte principal do problema, enquanto os *benchmarks* reais apresentam o funcionamento completo da aplicação.

Considerando que apenas os *benchmarks* de *kernel* e reais apresentam computação próxima de um sistema em uso real, somente esses tipos de *benchmark* foram escolhidos para serem utilizados nesse estudo.

Quanto ao funcionamento, os *benchmarks* podem ser principalmente de dois diferentes tipos (JOHN; EECKHOUT, 2006). Muitos *benchmarks* populares são programas que executam uma quantidade fixa de computação, e assim, de maneira simplista, o sistema que executar a tarefa em menor tempo é considerado o vencedor. Existe também os *benchmarks* de vazão (*throughput*) nos quais não existe o conceito de terminar uma quantidade fixa de trabalho, e sim, medir a taxa em que cada trabalho fica pronto, ou seja, o número de tarefas concluídas em um tempo fixo é usada para comparar os sistemas.

Uma vez que a escolha da carga de trabalho é de grande importância em todo projeto de avaliação de desempenho computacional, o processo de escolha da carga de trabalho a ser utilizada nos experimentos considerou diversos *benchmarks*:

- NPB (NASA) (JIN; FRUMKIN; YAN, 1999) – Conjunto de *benchmark* para avaliação de desempenho de supercomputadores paralelos.
- TPC-C (Wisconsin) (ALAMELDEEN et al., 2003) – Carga de trabalho comercial configurada para servidores multiprocessados.
- PARSEC (Princeton) (BIENIA et al., 2008) – *Benchmark* formado por um conjunto de aplicações paralelas para computadores de memória compartilhada, com foco em cargas de trabalho emergentes.
- SPEC CPU2006 (Spec) (STANDARD PERFORMANCE EVALUATION CORPORATION - SPEC, 2007) – Conjunto de *benchmark* mantido e padronizado para avaliar o desempenho de processadores de alto desempenho.
- SPEC OMP2001 (Spec) (STANDARD PERFORMANCE EVALUATION CORPORATION - SPEC, 2007) – Conjunto de *benchmark* mantido e padronizado para avaliar o desempenho de computadores de alto desempenho, multiprocessados de memória compartilhada.
- SPLASH-2 (Stanford) (WOO et al., 1995) – Conjunto de *benchmark* para avaliação de desempenho de memórias compartilhadas.

A Tabela 3.16 apresenta um quadro comparativo entre os diversos *benchmarks* analisados. A descrição sobre os fatores avaliados na Tabela 3.16 é apresentada abaixo:

- Disponibilidade – Avaliado com relação a disponibilidade do *benchmark* pelo grupo para utilização em nossos experimentos.
- Paralelismo – Diz respeito a existência de versão paralela das aplicações da carga de trabalho.
- Diversidade – A carga de trabalho é considerada diversificada se busca avaliar diversos fatores de uma arquitetura paralela.

Tabela 3.16: Quadro de comparação qualitativa de *benchmarks*

Benchmark	Disponível	Paralelo	Diverso	PAD	Portável	Tamanho	Aplicações
NAS-NPB	SIM	SIM	SIM	SIM	SIM	VAR	09
WISCONSIN TPC	SIM	SIM	SIM	NÃO	SIM	VAR	04
PARSEC	SIM	SIM	SIM	NÃO	NÃO	VAR	12
SPEC CPU2006	NÃO	NÃO	SIM	SIM	-	FIXO	29
SPEC OMP2001	NÃO	SIM	SIM	SIM	-	FIXO	11
SPLASH-2	SIM	SIM	SIM	SIM	NÃO	FIXO	14

- Foco em PAD – Apresenta foco em processamento de alto desempenho, tentando avaliar as novas arquiteturas a respeito de alto desempenho.

- Portabilidade – Representa a portabilidade dos aplicativos para diversas arquiteturas e sistemas operacionais, alguns *benchmarks* não possuem essa avaliação pois não temos acesso aos mesmos para avaliar esse quesito.

- Tamanho – O tamanho do problema a ser resolvido é fixo ou o *benchmark* apresenta diversos tamanhos pré-definidos.

- Aplicações – Quantidade de aplicações disponíveis no *benchmark*.

Após definir e ponderar sobre os fatores para decisão de escolha por determinada carga de trabalho, foi escolhido o conjunto de aplicações NAS-NPB para ser utilizada como carga de trabalho comparativa entre os diversos experimentos.

3.5.1 *Numerical Aerodynamic Simulation - Parallel Benchmark* - NAS-NPB

O *benchmark* de aplicações paralelas NAS apresenta diversas aplicações relacionadas a métodos numéricos de simulações aerodinâmicas para computação científica. Esses aplicativos foram projetados para comparar o desempenho de computadores paralelos, assim os aplicativos são formados por *kernels* e problemas de simulação de dinâmica de fluídos computacionais (CFD) derivados de importantes aplicações das classes aerofísicas, de maneira que as simulações de CFD reproduzem grande parte dos movimentos de dados e computação encontrada em códigos CFD completos (JIN; FRUMKIN; YAN, 1999).

O *benchmark* utilizado neste estudo foi a versão NPB-OpenMP 3.3 que possui as seguintes aplicações: BT (*Block Tridiagonal*), CG (*Conjugate Gradient*), MG (*Multigrid*), EP (*Embarassingly Parallel*), SP (*Scalar Pentadiagonal*), LU (*Lower and Upper triangular system*), IS (*Integer Sort*), FT (*fast Fourier Transform*), UA (*Unstructured Adaptive*) e DC (*Data Cube*). Todas aplicações executam operações de ponto-flutuante de precisão dupla (64 bits) e são programadas utilizando Fortran-90 (BT, CG, MG, EP, SP, LU, FT, UA e DC) ou C (IS), onde não há mistura de códigos Fortran-90 e C. Como o aplicativo DC (Data Cube), que simula operações de mineração de dados, apresentou problemas durante a compilação e execução em nossas simulações, esta aplicação não foi considerada em nossas comparações.

Atualmente, esta versão do NPB conta com diversos tamanhos de problemas, em ordem crescente: S, W, A, B, C e D. A Classe D só está disponível para alguns aplicativos. O tamanho S é descrito como problema de tamanho muito pequeno e deve apenas ser

utilizado para simulações ou testes de funcionalidade de sistemas e simulações. O tamanho W pode ser classificado como problema de tamanho real e embora seja pequeno, esse tamanho já apresenta características reais. Sugere-se que esse problema seja utilizado em máquinas de baixa capacidade ou simulações. O tamanho A é o mais utilizado em testes de máquinas reais, apresenta um tamanho mais robusto e tempo de execução mais elevado. Os demais tamanho B, C e D são recomendados para execução de máquinas reais com alto poder de processamento e alto nível de paralelismo.

Para decidir entre os tamanhos disponíveis, foram feitas algumas avaliações em uma máquina real para estimar o tempo de execução, escalabilidade e variação para os tamanhos S, W e A.

O tempo de execução é uma importante métrica em se tratando de simulações, uma vez que simulações de aplicações muito complexas demandam grande tempo, o que pode tornar o estudo inviável.

A escalabilidade visa mostrar se o tamanho do problema adotado tende a responder bem à medida que se aumenta o poder de processamento (número de núcleos), ou seja, espera-se que o problema tenha tamanho suficiente para obter ganhos (redução do tempo de execução), na medida em que se acrescenta mais núcleos de processamento (aumenta o paralelismo disponível).

A terceira métrica diz respeito à variação do tempo de execução em uma máquina real. Assim, se o tamanho do problema a ser trabalhado na aplicação for muito pequeno a variação devido a fatores externos como sistema operacional tende a ser maior, o que demandará um maior número de simulações até que se obtenha um valor confiável.

Esses experimentos para escolha do tamanho do problema para utilização nas simulações foram executados em um único computador dotado de dois processadores Xeon Quad-Core E5310 Clovertown de 1.6GHz e a carga de trabalho foi compilada com compilador Intel Fortran e C versão 10.0.1, com otimização de compilação -O3.

O tempo de execução do *benchmark* NPB para os tamanhos S, W e A executados na máquina real é apresentado na Figura 3.10. A soma de execução de todas as aplicações também é apresentada no gráfico. A soma dos tempos de execução são $143,89$ segundos para o tamanho A, $14,43$ segundos para o tamanho W e $2,86$ segundos para o tamanho S.

O tempo de execução médio para simulações no Simics é aproximadamente 380 vezes o tempo de uma máquina real, multiplicado pelo número de processadores a serem simulados. Desta maneira, podemos fazer estimativas do tempo de execução das cargas de trabalho no simulador. Considerando a simulação de 32 processadores, podemos estimar o tempo para simulação (MARTY et al., 2005) da execução seqüencial de todos aplicativos do NPB através da equação: $TempoSimulacao = TempoReal \cdot 380 \cdot Num.Procs.$

Pode-se estimar o tempo de execução dos aplicativos em 486 horas para uma execução completa do tamanho A, 50 horas para o tamanho W e 10 horas para o tamanho S. Assim, estimando o tempo de simulação de 6 execuções (1 *warm-up* e 5 medidas) será de aproximadamente 121 dias para o tamanho A e aproximadamente 12 dias para W e 2 dias e meio para o tamanho S.

Considerando que o total de organizações de memória *cache* para 32 processadores será igual a 6 (1, 2, 4, 8, 16 e 32 núcleos na mesma memória *cache*), estima-se que tomará 726 dias para o tamanho A, 72 dias para o tamanho W e 15 dias para o tamanho S. Ainda assim, como estima-se avaliar o experimento base com o impacto de variação de 5 fatores, o tempo total de simulação dos experimentos será aproximadamente 3630 dias para o tamanho A, 360 dias para o tamanho W e 75 dias para o tamanho S. Logo, apenas os tamanhos S e W são factíveis de serem utilizados nos experimentos, uma vez que

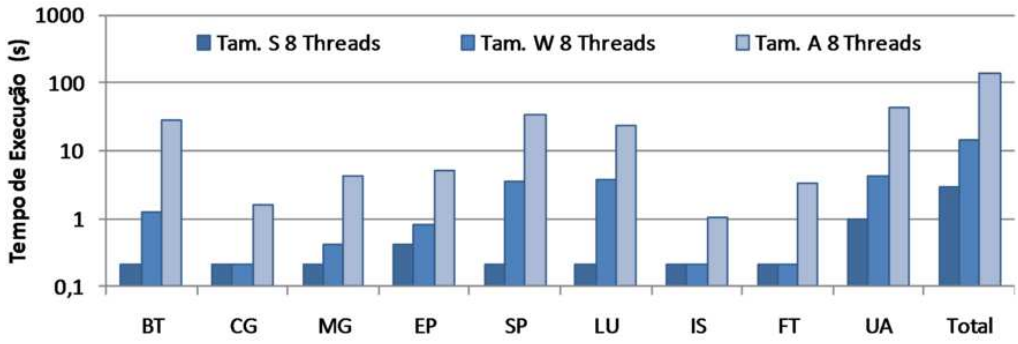


Figura 3.10: Tempo de execução do *benchmark* NPB para diversos tamanhos executado com 8 *threads*.

planeja-se executar os experimentos em 4 máquinas diferentes dividindo assim o tempo de simulação por 3 (120 dias para simular W e 25 dias para simular S).

As figuras 3.11(a), 3.11(b) e 3.11(c) ilustram a escalabilidade dos diversos tamanhos S, W e A, respectivamente, do NPB executado na máquina real.

A Figura 3.11(a) mostra que apenas as aplicações EP, FT e UA escalam conforme aumenta-se o número de *threads*, obtendo *speedup* de $6,40$, $1,99$ e $3,70$ respectivamente, para execução de 8 *threads*, As demais aplicações tiveram *speedup* menor que $1,10$.

Para o tamanho W, apresentado na Figura 3.11(b), as aplicações atingiram *speedup* de $5,48$ para BT, $5,91$ para CG, $2,47$ para MG, $6,21$ para EP, $4,54$ para SP, $5,16$ para LU, $3,03$ para IS, $3,91$ para FT e $6,59$ para UA utilizando 8 *threads*.

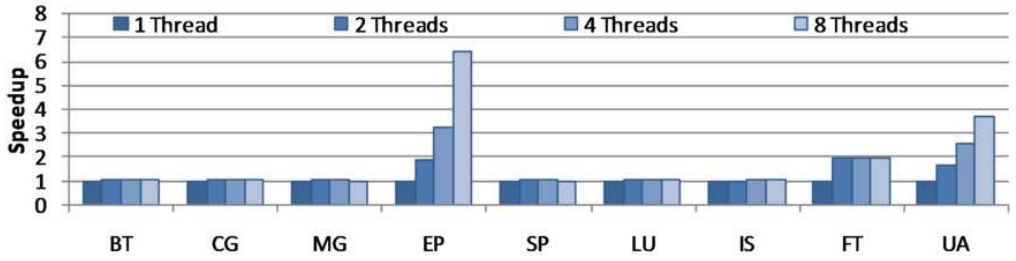
Já no tamanho A, ilustrado no gráfico da Figura 3.11(c), as aplicações atingiram, utilizando 8 *threads*, o *speedup* de $5,67$ para BT, $3,27$ para CG, $2,03$ para MG, $7,75$ para EP, $3,10$ para SP, $6,57$ para LU, $4,76$ para IS, $3,71$ para FT e $5,23$ para UA. Desta maneira, apenas os tamanhos W e A são adequados para serem utilizados nos experimentos.

Os gráficos das figuras 3.12(a), 3.12(b) e 3.12(c) apresentam os resultados sobre variação dos tempos de execução das cargas de trabalho NPB para diversos números de *threads*. Os gráficos apresentam os tempos de execução médios, baseados em 100 medições e além disso o gráfico mostra os valores mínimos e máximos para as diversas execuções.

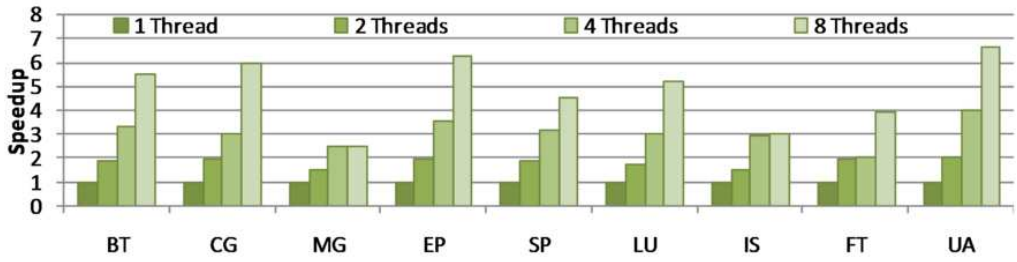
Assim, após os diversos estudos, foi decidido utilizar a carga de trabalho NPB de tamanho W por apresentar um bom balanço entre tempo de execução, escalabilidade e variação. Uma descrição das aplicações de tamanho W, utilizadas nos experimentos estão relacionadas abaixo:

- BT - *Block Tridiagonal*: Soluciona as equações de Navier-Stokes compressíveis 3D com um algoritmo implícito. Baseado no método implícito de direções alternadas (ADI) para solução de diferenças finitas, onde o sistema resultante é um sistema tridiagonal, o qual é solucionado seqüencialmente para cada dimensão. A divisão de trabalho é feita por subdivisão do domínio, com acesso a dados compartilhados nas bordas do domínio. Utiliza aproximadamente $2,7$ MB.

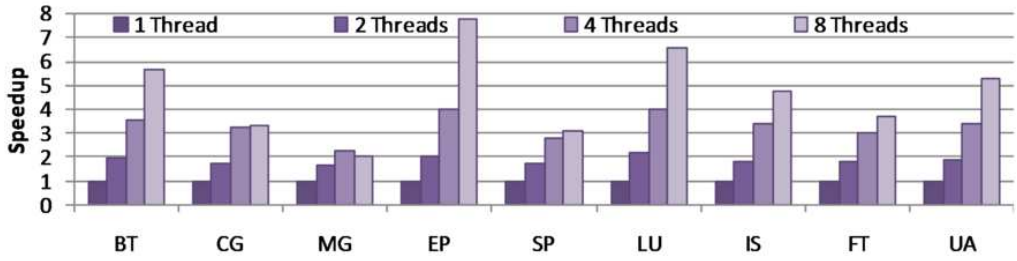
- CG - *Conjugate Gradient*: Computa o menor *eigenvalue* de uma grande matriz esparsa desestruturada, utilizando o método de gradiente conjugado. Exercita computações e acesso a dados compartilhados em grades desestruturadas. Utiliza aproximadamente $13,7$ MB.



(a) Tamanho S



(b) Tamanho W



(c) Tamanho A

Figura 3.11: Escalabilidade do *benchmark* NPB.

- MG - *Multigrid*: Soluciona um sistema escalar tridimensional de equação de Poisson utilizando o método *multigrid* de ciclos de cinco pontos. O algoritmo trabalha entre grades finas e grossas exercitando acessos a dados de grande (acesso esparso) e pequenas distância (acesso linear) em um conjunto de dados contíguo. Utiliza aproximadamente $55,7$ MB.

- EP - *Embarassingly Parallel*: Carga de trabalho nativamente paralela, a qual gera pares Gaussianos randômicos. Objetiva estabelecer um ponto de referência para desempenho de pico para uma determinada plataforma uma vez que apresenta muito processamento independente e quase nenhum compartilhamento de dados. Utiliza aproximadamente $1,3$ MB.

- SP - *Scalar Pentadiagonal*: Aplicação similar a aplicação BT, solucionando um problema de dinâmica de fluídos computacionalmente (CFD). O problema é baseado na fatoração aproximada de Beam-Warming que é decomposta em 3D. O sistema pentadiagonal resultante é solucionado seqüencialmente por cada dimensão. Para solução paralela, o sistema é subdividido, e distribuído, necessitando apenas acesso a dados compartilhados nas bordas de cada domínio. Utiliza aproximadamente $8,7$ MB.

- LU - *Lower and Upper triangular system*: Aplicação de solução de dinâmica de fluídos computacionalmente (CFD) que utiliza o método simétrico sucessivo de sobre-relaxação (SSOR) para solucionar o sistema resultante da discretização de diferenças finitas das equações de Navier-Stokes 3D, separando em sistemas triangulares superiores e inferiores. O uso de dados compartilhados é pequeno comparado com as etapas de processamento, além disso, existe etapas de alto reuso de linhas da memória *cache*. Utiliza aproximadamente $6,6$ MB.

- IS - *Integer Sort*: Esse aplicativo exercita o desempenho de operações com números inteiros e acesso a dados compartilhados. Esse aplicativo é muito útil para métodos de partículas. Utiliza aproximadamente $3,4$ MB.

- FT - *Fast Fourier Transform*: Baseado no método de transformadas rápidas de Fourier (FFT) tridimensional. A transformada rápida executa em cada uma das três dimensões. Essa transformada é implementada para na primeira etapa os dados sejam acessados de forma linear, e em seguida, começam as etapas de acessos a dados compartilhados. Utiliza aproximadamente 20 MB.

- UA - *Unstructured Adaptive*: Exercita mudanças irregulares e contínuas nos acessos à memória medindo seus efeitos. Os resultados de execução dessa aplicação podem ser utilizados para comparação de desempenho de um determinado sistema de memória. Utiliza aproximadamente $16,3$ MB.

Diante das diversas aplicações, podemos dividi-las por comportamento quanto ao acesso a dados, conforme apresentado abaixo:

As aplicações BT, SP e LU tem seu paralelismo formado tipicamente por divisão de domínio espacial, com compartilhamento de dados nas bordas de cada sub-domínio e apresentam acessos de forma linear aos dados.

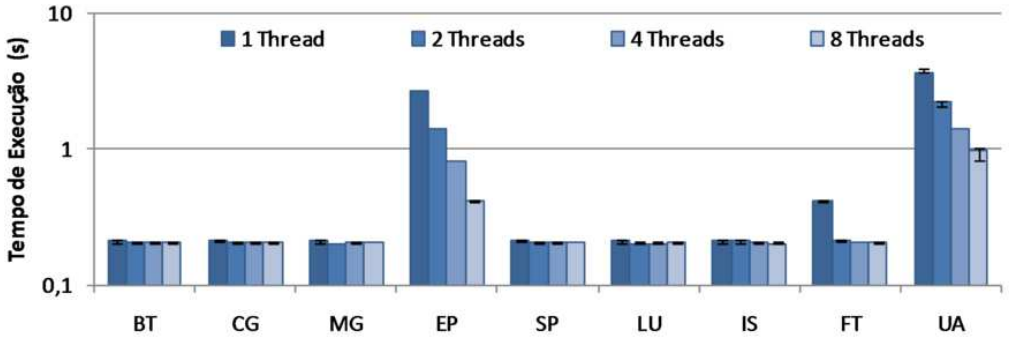
As aplicações CG e UA são aplicações que apresentam característica de acesso randômico a dados, que podem indicar o desempenho exclusivamente das memórias. Sendo que o desempenho da aplicação UA indica o desempenho do sistema de memória podendo servir de base para comparações sobre memórias.

A aplicação MG trabalha sobre dados contíguos, entretanto, em cada etapa de computação o padrão de acesso varia, sendo que nas primeiras etapas o acesso é desalinhado, e conforme a grade é refinada em cada etapa, os acessos se tornam mais lineares. Assim, essa aplicação pode ser considerada do ponto de vista de memória, um misto entre as aplicações BT, SP e LU de acesso linear, e as aplicações CG e UA de acesso irregular.

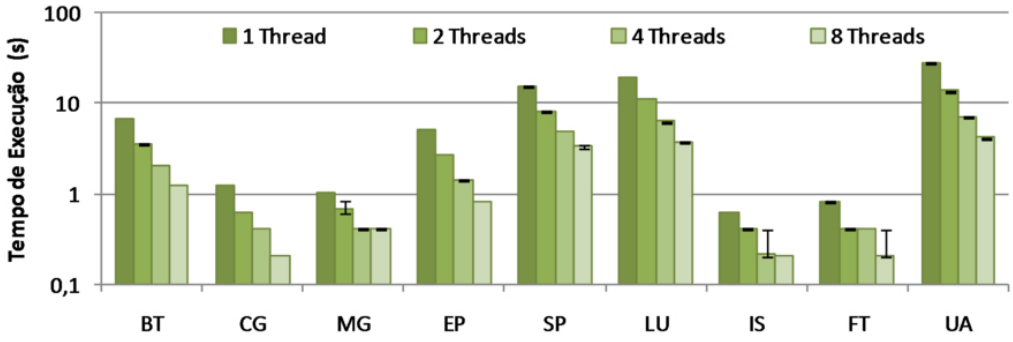
A aplicação EP é nativamente paralela, e possui muito pouco compartilhamento de dados entre as *threads*. O desempenho dessa aplicação pode ser utilizado como referência de desempenho computacional de pico de uma determinada máquina.

A aplicação IS, apresenta comportamento contrário a aplicação EP, nesse, existe muito acesso a dados compartilhados além dos acessos lineares aos dados armazenados.

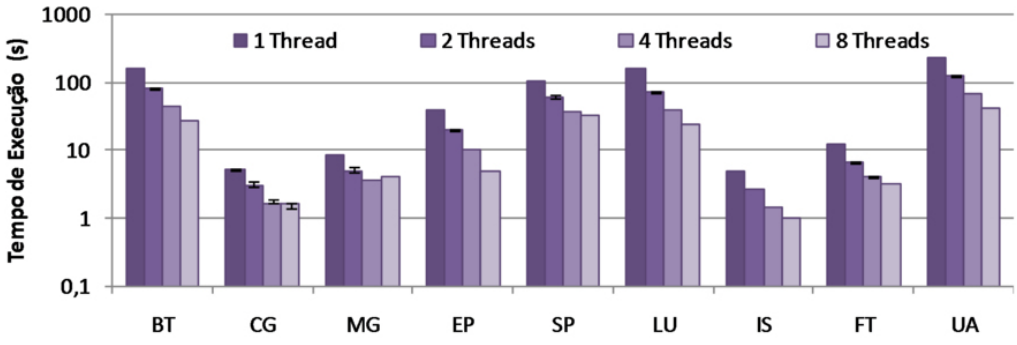
Por fim, a aplicação FT apresenta etapas de acesso linear com etapas de acesso compartilhado entre as diversas *threads*.



(a) Tamanho S



(b) Tamanho W



(c) Tamanho A

Figura 3.12: Variação do *benchmark* NPB.

4 AVALIAÇÃO DOS RESULTADOS

" Measures are not to provide numbers but insight. "
— INGRID BUCHER

A etapa de resultados de uma avaliação de desempenho é uma das mais importantes, uma vez que todos os dados e as análises feitas acerca dos dados devem ser compilados e apresentados de forma clara e objetiva. Justamente nesta etapa é que ocorrem muitos erros ligados à avaliação do sistema computacional (JAIN, 1991), como ignorar a variação das medidas, apresentar os resultados de forma incorreta, assim como omitir as limitações do sistema.

Assim, quando estamos trabalhando em sistemas reais, normalmente são feitas múltiplas medições, e técnicas estatísticas são utilizadas para garantir que as medições são estatisticamente significantes. Logo, o objetivo dessa metodologia é garantir que efeitos causados por fatores não controlados não leve a falsas conclusões. Contudo, ao trabalhar com um simulador determinístico, onde todas as execuções de uma mesma aplicação iniciadas de um mesmo *checkpoint* vão retornar os mesmos resultados, uma vez que o simulador irá executar as mesmas operações e sofrer de mesmas latências do *hardware*, como é o caso do Simics, podemos considerar que temos um resultado plausível a partir de apenas uma medição. Entretanto, devemos considerar que o simulador ser determinístico não garante que a carga de trabalho também será determinística. Ou seja, pequenas variações iniciais do estado do simulador pode levar a carga de trabalho ou o sistema operacional a caminhos de execução diferentes (ALAMELDEEN et al., 2002), dessa maneira, a execução única do experimento pode levar a falsas conclusões.

Por isso, certos cuidados foram tomados na simulação com relação a variação dos resultados obtidos, uma vez que o simulador que estamos utilizando é determinístico. Para inserção de não-determinismo entre as execuções, cada aplicação foi executadas a partir de diferentes *checkpoints*, fazendo assim o sistema agir de forma não-determinística, pelas interrupções causadas pelo sistema operacional, levando assim, as cargas de trabalho à diferentes caminhos de execução. Para isso, após cada execução de uma determinada aplicação, foi criado um *checkpoint* diferente, a partir do qual a próxima execução e medição daquela mesma aplicação seria feita.

Além da preocupação em relação ao determinismo do sistema, planejou-se reduzir possíveis efeitos transientes (ALAMELDEEN; WOOD, 2003) executando previamente cada aplicação da carga de trabalho, a fim de aquecer a memória *cache*, salvando o estado do sistema após essa primeira execução de cada aplicação. Então, a partir desse estado salvo foram executadas e medidas cada aplicação da carga de trabalho.

Desta maneira, se faz necessário o uso de técnicas estatísticas para avaliação da variação das medições, para que assim, possamos obter medidas confiáveis que indiquem o desempenho do sistema. Dessa forma, serão utilizadas as métricas já definidas na seção 3.1.2. Nas medições foi utilizado a métrica de um desvio padrão, para avaliar a confiança do resultado, sendo que o erro igual a um desvio padrão representa 81,30% em uma distribuição T Student. Logo, nos gráficos dos resultados quando estivermos nos referindo ao desvio padrão, estamos nos referindo, ao mesmo tempo, a um intervalo de confiança igual a 81,30%.

Definidos os métodos de avaliação e comparação básicos, resta definir o método de execução dos experimentos, a fim de obter os resultados para avaliação.

A partir da fórmula do intervalo de confiança, pode-se ver que o tamanho do intervalo é inversamente proporcional à raiz quadrada do número de experimentos que fazemos. Uma vez que o objetivo é minimizar o número de repetições dos experimentos a serem executados, essa fórmula pode ser utilizada para determinar quantos experimentos serão necessários para atingir um intervalo de confiança especificado (LILJA, 2004).

Para obter o número de experimentos necessários (n) com confiança ($1 - \alpha/2$) de que os valores medidos estarão com um erro máximo (e), deve-se utilizar uma média de valores medidos para um experimento inicial, obtendo a média ($\bar{x}$) e a unidade de distribuição (z) referente ao intervalo de confiança escolhido com o número de repetições do primeiro experimento, aplicando essas variáveis na seguinte fórmula:

$$\left(\frac{z_{1-\alpha/2}\sigma}{e\bar{x}}\right)^2 = n$$

A unidade padrão de distribuição (z) é tomada sendo uma distribuição T Student uma vez que estimamos ser necessário executar menos de 30 repetições. Para estimar a quantidade de experimentos necessários, a primeira organização de memória 1Core/L2 do primeiro experimento foi repetida 20 vezes, sendo possível assim tirar os valores necessários. A Figura 4.1 apresenta um gráfico de dispersão dos valores obtidos nas 20 execuções.

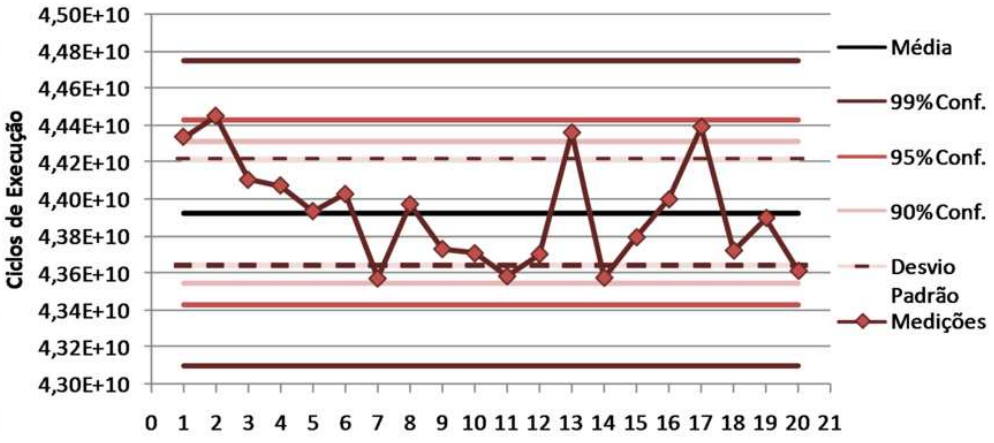


Figura 4.1: Espalhamento das medidas da organização 1Core/L2 do primeiro experimento para estimar número de repetições.

Com base nas 20 medições ($N = 19$), utilizando uma distribuição T Student, podemos afirmar com 95% de confiança ($\alpha = 0,10; 1 - \alpha/2 = 0,95; z = 1,729$), que para obter

uma variação menor que 1% ($+/-0,5\%, e = 0,05$) teremos que executar 5,15 repetições de cada experimento, conforme equação abaixo:

$$\left(\frac{1,729 \cdot (2,88E+08)}{0,005 \cdot (4,39E+10)}\right)^2 = 5,15$$

Para facilitar o entendimento dos resultados, um esquema de nomenclatura foi adotado, considerando o número de núcleos (*cores*) por memória *cache* L2: 1Core/L2, 2Cores/L2, 4Cores/L2, 8Cores/L2, 16Cores/L2 e 32Cores/L2. Para o caso das organizações com memória *cache* L3, foi ainda adotada a descrição do número de memórias *cache* L2 por memória *cache* L3, gerando assim, 1L2/L3, 2L2/L3 4L2/L3. Para os experimentos com limitações na quantidade de portas de comunicação da memória *cache* serão utilizadas as nomenclaturas já apresentadas com adição da informação de portas, como por exemplo a organização 4Cores/L2 com 1, 2 e 4 portas: 4Cores/L2-1P, 4Cores/L2-2P e 4Cores/L2-4P. Lembrando que em todos os experimentos, idependente da organização de memória *cache* as aplicações da carga de trabalho paralela executadas são formadas de 32 *threads*.

Esse capítulo de resultados e avaliação está dividido por experimentos, os experimentos 1, 2, 3, 4 e 5 trazem os resultados divididos nas subseções: Processador *Multi-Core*; Subsistema de Memória; Consumo de Energia e Ocupação de Área; Sumário; Avaliação das Aplicações. As sub-seções de avaliação das aplicações trazem resultados divididos em tipos de aplicação apresentados na seção 3.5.1. O experimento 6, é dividido em: Experimento 1; Experimento 2; Experimento 3; Experimento 4. Onde o nome da subseção diz respeito ao experimento que fornece os dados de entrada utilizado na avaliação.

Parte dos resultados iniciais presentes nos experimentos 2, 3 e 4, são mostrados em (ALVES; FREITAS; NAVAUX, 2009) que foi apresentado no Workshop on Many Core (WMC'2009).

Com relação aos experimentos, teremos na seção 4.1 o experimento base sobre a influência do compartilhamento da memória *cache* L2 entre diversos núcleos de processamento. Na seção 4.2 será visto o segundo experimento, o qual diz respeito à influência do tamanho da memória *cache* no compartilhamento da mesma. A seção 4.3 apresentará os resultados sobre avaliação do aumento da associatividade em *multi-core* com memórias compartilhadas. Na seção 4.4 será abordada a influência do tamanho da linha da memória *cache* para diversas organizações de memória. A seção 4.5 trará resultados a respeito da influência da hierarquia de memória *cache*, avaliando a inserção de mais um nível na hierarquia de memória *cache*. Por fim a seção 4.6 apresentará resultados estimados sobre contenções e limites físicos para a memória *cache*.

Além dos resultados disponíveis nas próximas seções desse capítulo, alguns resultados adicionais dos diversos experimentos estão disponíveis para consulta no Apêndice B. Dessa forma, apenas os resultados mais relevantes serão apresentados nas seções a seguir, deixando a cargo do leitor a consulta no apêndice dos demais resultados sobre: Porcentagem de instruções de acesso a dados da memória executadas; Invalidações MESI da memória *cache* L2; Invalidações MESI da memória *cache* L3; Consumo de energia e potência dinâmica do sistema de memória; Consumo de energia e potência estática do sistema de memória.

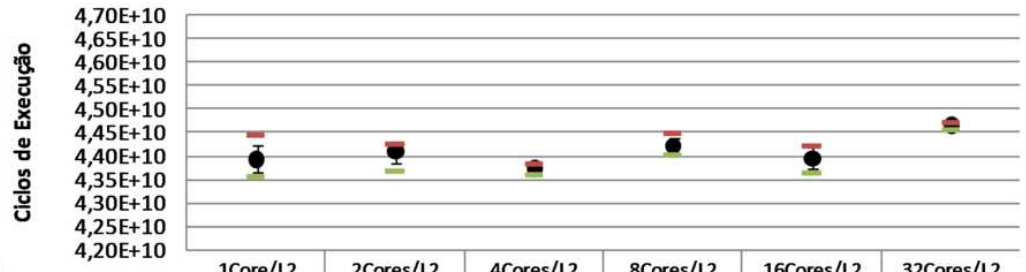
4.1 Experimento 1 - Compartilhamento da Memória *Cache*

Este primeiro experimento visa avaliar a influência do compartilhamento da memória *cache* entre diversos núcleos de processamento. Nesta seção serão apresentados os resultados desse experimento, bem como a avaliação dos resultados obtidos. Para facilitar o entendimento, a apresentação dos dados será dividida em três áreas, dados referentes a execução, dados sobre as memórias, e por fim, dados referentes a métricas físicas como consumo de potência e ocupação de área.

Com o aumento no compartilhamento dos blocos de memória *cache*, as aplicações tendem a usufruir de maior velocidade para o compartilhamento de dados entre os diversos fluxos de execução, reduzindo assim, a quantidade de faltas de dados. Por outro lado, o compartilhamento adotado nesse experimento manteve o total de memória *cache* do sistema, assim, ao aumentar o grau de compartilhamento, os bancos da memória *cache* formadas são maiores, influenciando no tempo de acesso a dados, consumo de potência e área ocupada.

O primeiro resultado desse experimento é sobre o espalhamento das somas de medidas de tempo de execução de todas aplicações da carga de trabalho. A Figura 4.2 apresenta as médias, os valores máximos e mínimos obtidos, além do desvio padrão e o coeficiente de variação para execução da carga de trabalho nas diversas organizações (1, 2, 4, 8, 16 e 32 Cores/L2) de memória *cache* desse experimento. Podemos ver que nas medidas obtidas, o desvio padrão representa no máximo 0,66% da média, sendo uma variação aceitável para os resultados. Entretanto, podemos ver que algumas médias se sobrepõem ao desvio padrão de outras médias e nesses casos, é impossível afirmar que existe alguma diferença no desempenho final entre esses sistemas. Podemos ver que apenas a organização 32Cores/L2 apresenta uma diferença de desempenho final significativa, ficando mais distante das demais.

Mesmo com a equivalência de sistemas, os próximos resultados vão avaliar os diversos sistemas por aplicação, onde as mudanças entre organizações ficam mais simples de serem avaliadas. Além disso, analisando o desempenho por aplicação, podemos entender os motivos que levaram os diferentes sistemas a resultados equivalentes entre si.



	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Média	4,39E+10	4,41E+10	4,37E+10	4,42E+10	4,39E+10	4,46E+10
Máximo	4,45E+10	4,43E+10	4,38E+10	4,45E+10	4,42E+10	4,47E+10
Mínimo	4,36E+10	4,37E+10	4,36E+10	4,40E+10	4,36E+10	4,46E+10
Desvio Padrão	2,88E+08	2,33E+08	9,10E+07	1,93E+08	2,36E+08	6,11E+07
C.O.V.	0,66%	0,53%	0,21%	0,44%	0,54%	0,14%

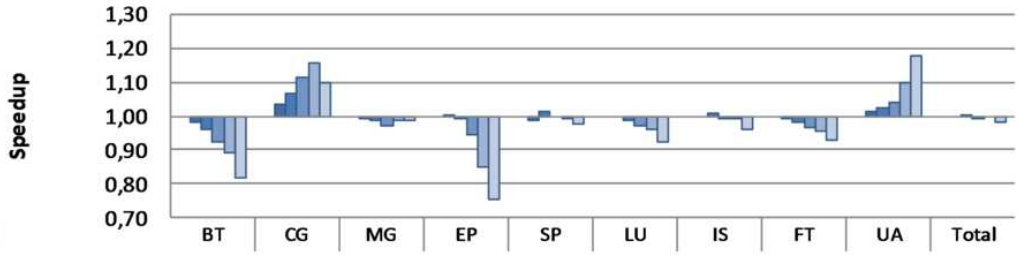
Figura 4.2: Espalhamento das medidas do primeiro experimento.

4.1.1 Processador *Multi-Core*

O resultado apresentado nessa subseção diz respeito ao ganho de desempenho do sistema, que é apresentado na Figura 4.3. Nessa figura pode-se verificar o comportamento por aplicação diante das mudanças no compartilhamento da memória *cache*, onde nota-se claramente que as aplicações CG e UA foram as que obtiveram melhores ganhos (superior a 10%), e as aplicações BT e EP apresentaram redução maior de 10%. No geral, considerando a soma de tempos das aplicações, a organização 4Cores/L2 foi a que apresentou pequeno ganho de desempenho ao compartilhar a memória *cache*.

Podemos notar ainda que a aplicação CG, mesmo apresentando ganhos de desempenho, obteve um pico de desempenho com 16Cores/L2, e então, na organização 32Cores/L2 apresentou um recuo no desempenho.

Através desse gráfico, podemos ver que os ganhos de desempenho em algumas aplicações e perda em outras, foi o que levou a equivalência no desempenho final de alguns sistemas avaliados.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
2Cores/L2	0,979	1,032	0,993	1,004	0,986	0,999	0,997	0,992	1,013	0,996
4Cores/L2	0,959	1,066	0,987	0,993	1,012	0,989	1,006	0,981	1,025	1,004
8Cores/L2	0,923	1,114	0,971	0,942	0,995	0,971	0,990	0,965	1,040	0,994
16Cores/L2	0,890	1,157	0,985	0,851	0,995	0,962	0,992	0,954	1,101	1,000
32Cores/L2	0,818	1,100	0,985	0,753	0,978	0,921	0,959	0,926	1,176	0,984

Figura 4.3: *Speedup* do primeiro experimento.

Como alguns resultados sobre as memórias *cache* L1 e L2 serão dados em MPKI (*misses per kilo instructions*), a Figura 4.4 apresenta o gráfico com dados a respeito do número de instruções executadas pelo sistema nas diversas organizações de memória *cache*. A quantidade de instruções medidas considera tanto as instruções das aplicações como do sistema operacional no momento da medida.

Dessa maneira, notamos que houve pequenas variações na quantidade de instruções executadas, que podem ter sido ocasionadas por influência do próprio sistema operacional, o qual pode ter executado mais instruções enquanto havia espera de dados da memória.

A porcentagem de instruções de acesso à memória com relação ao total de instruções é apresentado na Figura 4.5, mostrando que nas diferentes aplicações, de 20% a 30% das instruções efetuam acesso a dados, seja para escrita ou leitura. A aplicação BT é formada com acima de 30% de instruções de memória, sendo a porcentagem mais alta entre as aplicações, e a aplicação LU apresentado valor próximo a 20% é a aplicação formada com menor porcentagem de instruções de acesso a dados. Devemos lembrar que a porcentagem de instruções deve ser combinada com a quantidade de instruções para obtermos a quantidade total de instruções de acesso à memória. Com esse gráfico, podemos notar que não existe clara correlação entre o desempenho obtido pela aplicação

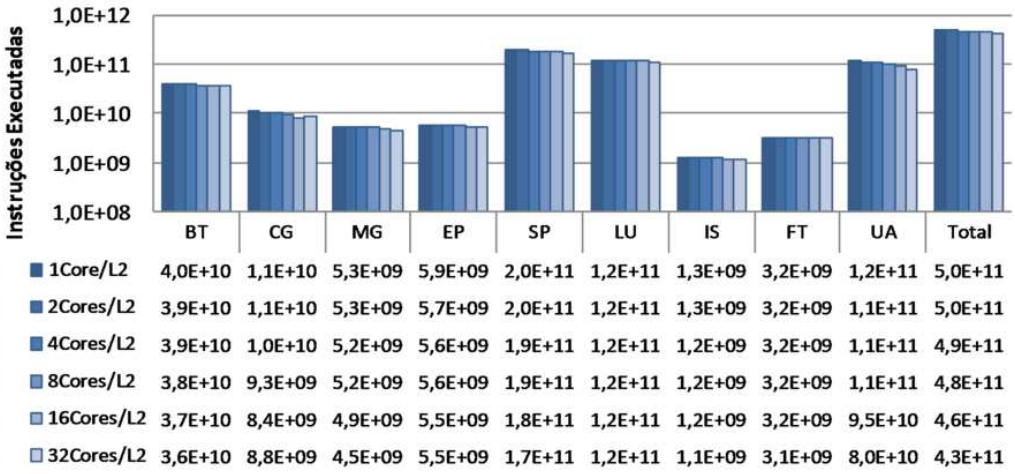


Figura 4.4: Instruções executadas do primeiro experimento.

e a quantidade de instruções de memória executada.

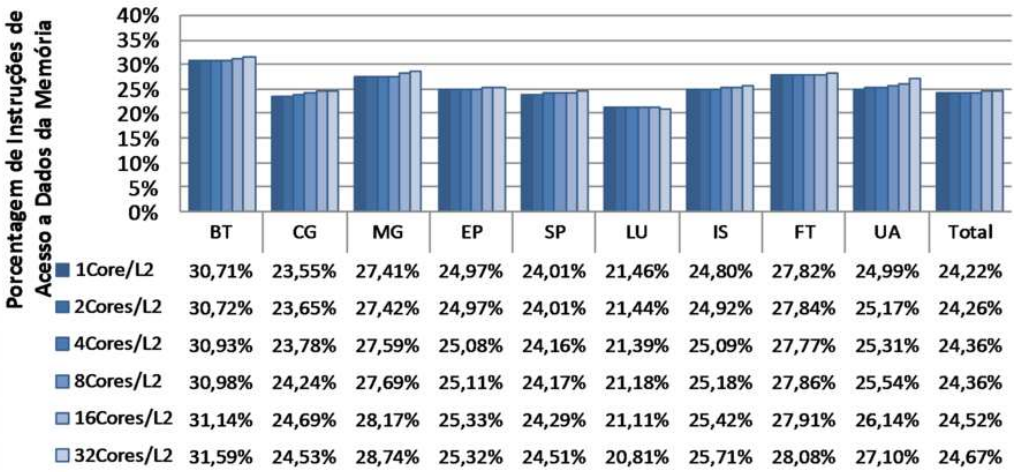
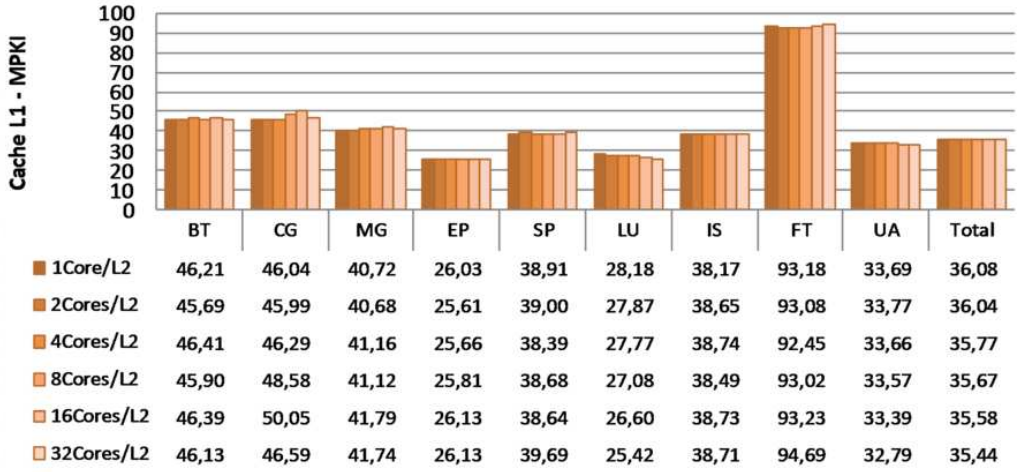


Figura 4.5: Instruções de acesso à memória executadas do primeiro experimento.

4.1.2 Subsistema de Memória

A Figura 4.6 apresenta o gráfico contendo os valores de MPKI para a memória *cache* L1, onde pode-se ver que a aplicação FT, a qual apresentou piora no desempenho, apresenta alta taxa de faltas de dados, comportamento que não é expresso pela aplicação BT, que também havia sofrido grande queda de desempenho. Assim não se pode afirmar com apenas a quantidade de MPKI que uma determinada aplicação irá ter piora no desempenho.

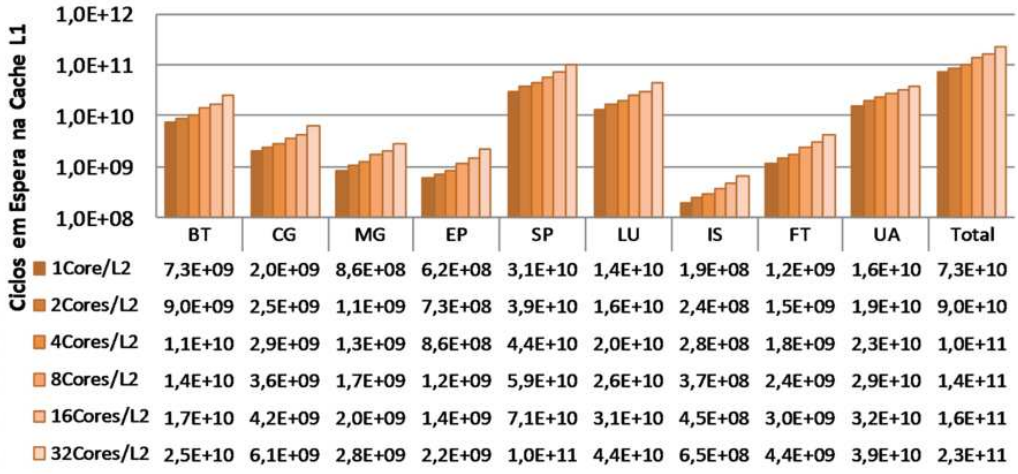
Os ciclos perdidos por espera de dados durante faltas da memória *cache* L1 são apresentados na Figura 4.7. Podemos ver claramente a escalada no número de ciclos perdidos durante faltas de dados na memória *cache* L1 para todas as aplicações, como era de se esperar, uma vez que o aumento no compartilhamento na memória *cache* L2 faz com que as latências, assim como o tamanho dos bancos de memória aumentem. Com base nesses dados, fica claro que, para que uma determinada organização de memória ganhe



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	46,21	46,04	40,72	26,03	38,91	28,18	38,17	93,18	33,69	36,08
2Cores/L2	45,69	45,99	40,68	25,61	39,00	27,87	38,65	93,08	33,77	36,04
4Cores/L2	46,41	46,29	41,16	25,66	38,39	27,77	38,74	92,45	33,66	35,77
8Cores/L2	45,90	48,58	41,12	25,81	38,68	27,08	38,49	93,02	33,57	35,67
16Cores/L2	46,39	50,05	41,79	26,13	38,64	26,60	38,73	93,23	33,39	35,58
32Cores/L2	46,13	46,59	41,74	26,13	39,69	25,42	38,71	94,69	32,79	35,44

Figura 4.6: Memória *cache* L1 MPKI do primeiro experimento.

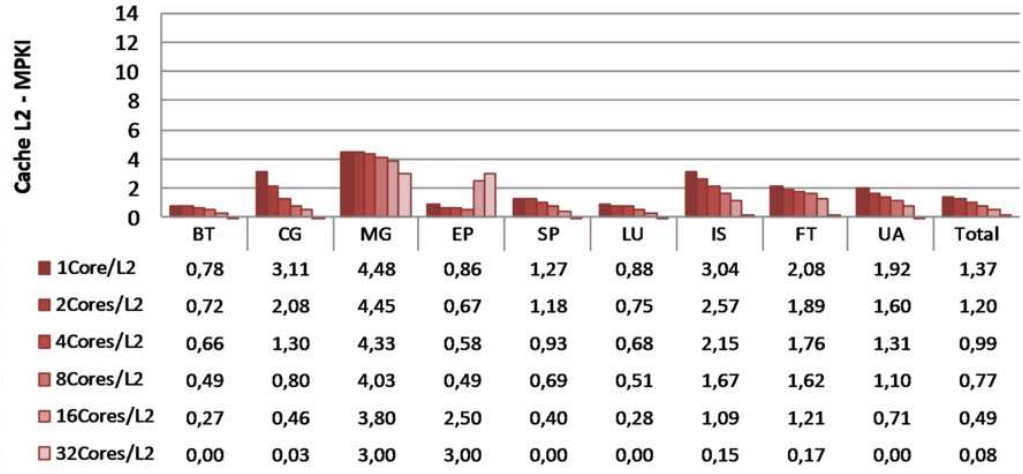
em desempenho final, a redução de faltas nos outros níveis de memória *cache* deve ser suficiente para cobrir esse aumento de ciclos perdidos por espera de dados. Além disso, pode-se concluir que a escalada de ciclos perdidos tem aparência linear entre aplicações, e isso se deve a pouca variação de faltas de dados na memória *cache* L1 combinada com o aumento suave de latência entre as diversas organizações.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	7,3E+09	2,0E+09	8,6E+08	6,2E+08	3,1E+10	1,4E+10	1,9E+08	1,2E+09	1,6E+10	7,3E+10
2Cores/L2	9,0E+09	2,5E+09	1,1E+09	7,3E+08	3,9E+10	1,6E+10	2,4E+08	1,5E+09	1,9E+10	9,0E+10
4Cores/L2	1,1E+10	2,9E+09	1,3E+09	8,6E+08	4,4E+10	2,0E+10	2,8E+08	1,8E+09	2,3E+10	1,0E+11
8Cores/L2	1,4E+10	3,6E+09	1,7E+09	1,2E+09	5,9E+10	2,6E+10	3,7E+08	2,4E+09	2,9E+10	1,4E+11
16Cores/L2	1,7E+10	4,2E+09	2,0E+09	1,4E+09	7,1E+10	3,1E+10	4,5E+08	3,0E+09	3,2E+10	1,6E+11
32Cores/L2	2,5E+10	6,1E+09	2,8E+09	2,2E+09	1,0E+11	4,4E+10	6,5E+08	4,4E+09	3,9E+10	2,3E+11

Figura 4.7: Ciclos de espera de dados na memória *cache* L1 do primeiro experimento.

A Figura 4.8 apresenta os resultados referentes a valores MPKI para memória *cache* L2 entre as diferentes organizações de memória *cache*. Nesse gráfico fica clara a eficácia das memórias *cache* em sistemas computacionais onde as taxas de faltas em média são inferiores a 0,25 para cada mil instruções. Notamos, com esse gráfico que as aplicações que obtiveram pior desempenho apresentam comportamentos diferentes. Para a aplicação BT, a taxa de faltas já é bem baixa na primeira organização, já para a aplicação EP, houve redução até a organização de 8Cores/L2, depois o número de faltas subiu, o que pode ter sido ocasionado pelo aumento no número de conflitos de endereços. Dessa maneira, nenhuma das duas aplicações obteve redução necessária para compensar o aumento da latência de acesso à memória *cache* L2.



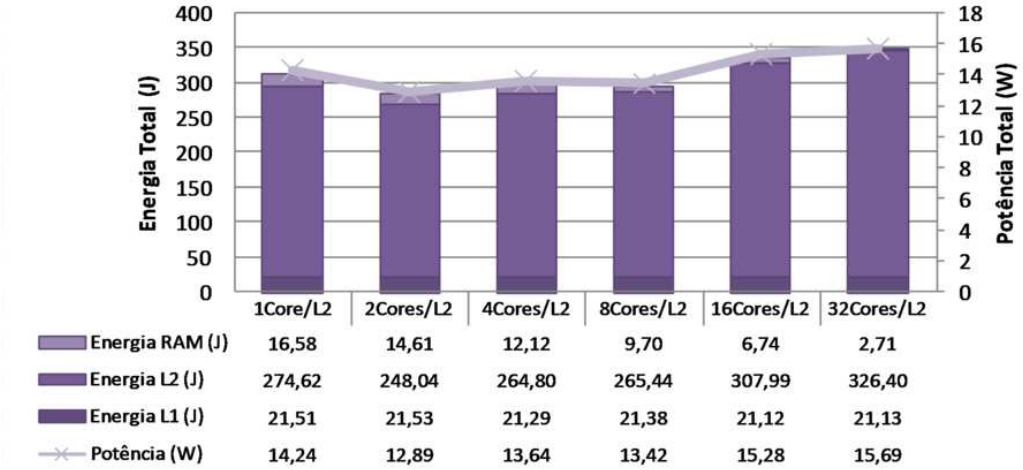
	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	0,78	3,11	4,48	0,86	1,27	0,88	3,04	2,08	1,92	1,37
2Cores/L2	0,72	2,08	4,45	0,67	1,18	0,75	2,57	1,89	1,60	1,20
4Cores/L2	0,66	1,30	4,33	0,58	0,93	0,68	2,15	1,76	1,31	0,99
8Cores/L2	0,49	0,80	4,03	0,49	0,69	0,51	1,67	1,62	1,10	0,77
16Cores/L2	0,27	0,46	3,80	2,50	0,40	0,28	1,09	1,21	0,71	0,49
32Cores/L2	0,00	0,03	3,00	3,00	0,00	0,00	0,15	0,17	0,00	0,08

Figura 4.8: Memória *cache* L2 MPKI do primeiro experimento.

4.1.3 Consumo de Energia e Ocupação de Área

Embora a variação do desempenho final entre os sistemas tenha sido baixa, a variação de consumo de potência entre as organizações sofreram modificações mais acentuadas de comportamento.

A soma de consumo de energia e potência dinâmica e estática está ilustrada na Figura 4.9, apresentando a organização 2Cores/L2 como a de menor consumo de potência total. A suavidade na variação da soma de energias é mantida graças ao aumento do consumo na memória *cache* L2, enquanto houve redução no consumo de potência na memória principal. Esse resultado de potência total é complexo, uma vez que depende do tempo de execução do experimento, quantidade de faltas e tamanho da memória *cache* L2, o que é verificado observando os valores para um desses cada componentes.



	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Energia RAM (J)	16,58	14,61	12,12	9,70	6,74	2,71
Energia L2 (J)	274,62	248,04	264,80	265,44	307,99	326,40
Energia L1 (J)	21,51	21,53	21,29	21,38	21,12	21,13
Potência (W)	14,24	12,89	13,64	13,42	15,28	15,69

Figura 4.9: Consumo de energia e potência total do sistema de memória do primeiro experimento.

A Figura 4.10 apresenta o último resultado referente a métricas físicas, apresentando a ocupação de área total e por banco de memória *cache* L2 para as diversas organizações apresentadas nesse primeiro experimento. A organização 8Cores/L2 foi a que apresentou

a menor ocupação de área total de memória *cache*, seguido pelas organizações 16Cores/L2 e 2Cores/L2. Essa variação não linear na área ocupada pela memória *cache* L2 acontece devido à estrutura interna de organização de sub-blocos de dados e *tag*, além das interconexões dentro da memória *cache*.

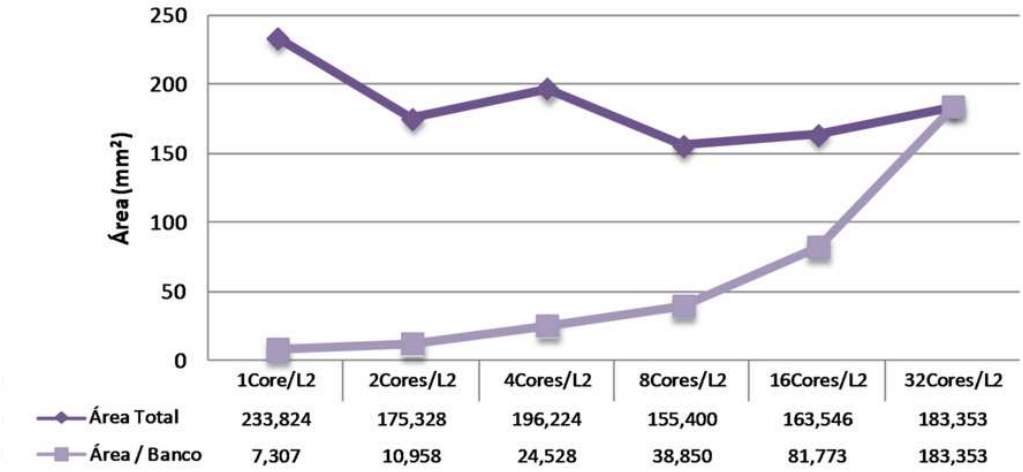


Figura 4.10: Área ocupada pela memória *cache* L2 do primeiro experimento.

4.1.4 Sumário

A Figura 4.11 apresenta um sumário com valores de desempenho, consumo de potência e ocupação de área para esse experimento. Podemos notar pelo gráfico que a organização mais vantajosa em termos de desempenho é a organização 4Cores/L2 onde obtemos ganho de 0,45% no desempenho, ao mesmo tempo que obtém-se redução de 4,74% na potência consumida e 27,08% na ocupação de área.

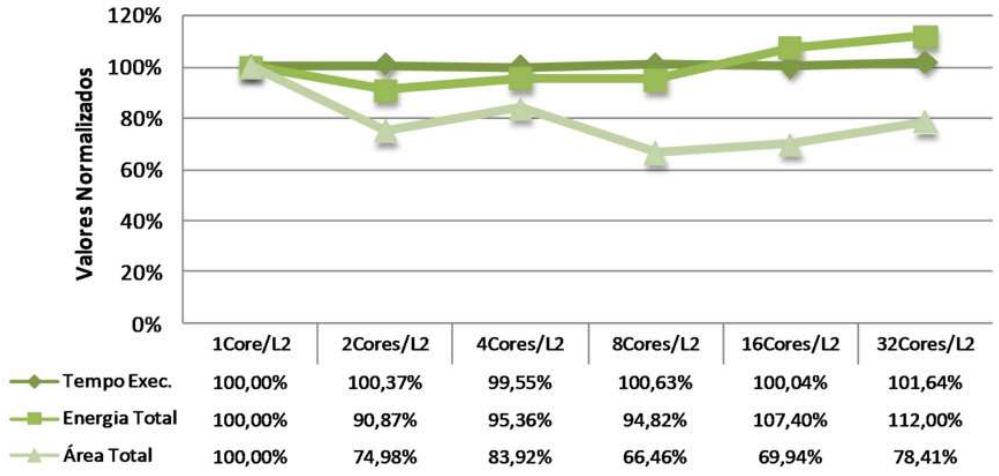
No entanto, o menor consumo de potência foi obtido na organização 2Cores/L2, com redução em 9,13%. Já a menor ocupação de área fica na configuração 8Cores/L2, reduzindo em 34,54% sem perda significativa de desempenho.

4.1.5 Avaliação das Aplicações

Conforme foi possível notar nos resultados obtidos, as aplicações da carga de trabalho utilizada apresentam resultados diferentes entre si quanto ao desempenho, faltas de dados, entre outros. Por isso, esta subseção apresenta avaliações sobre as aplicações e os resultados obtidos nesse primeiro experimento.

Considerando as aplicações BT, SP e LU, o comportamento de desempenho foi próximo entre BT e LU, sendo que, ao aumentar o compartilhamento da memória, o desempenho foi piorando. Já na aplicação SP houve casos de ganho de desempenho. A justificativa para a melhora no desempenho da aplicação SP em alguns casos, é verificado na taxa MPKI da memória *cache* L2, uma vez que essa foi a que sofreu maior taxa de redução de faltas na L2 entre as três aplicações comparadas. Mesmo assim, podemos concluir que apenas o compartilhamento da memória *cache* L2, que gera altas latências de acesso, não fornece bons resultados para aplicações que apresentem pouco compartilhamento de dados.

As aplicações CG e UA apresentaram bom desempenho conforme o compartilhamento de memória *cache* foi incrementado. Essas aplicações apresentam comportamento



	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Tempo Exec.	100,00%	100,37%	99,55%	100,63%	100,04%	101,64%
Energia Total	100,00%	90,87%	95,36%	94,82%	107,40%	112,00%
Área Total	100,00%	74,98%	83,92%	66,46%	69,94%	78,41%

Figura 4.11: Sumário de desempenho, consumo de energia e ocupação de área do primeiro experimento.

de acesso randômico, logo, ao aumentar os blocos de memória *cache*, a taxa de faltas de dados da memória *cache* L2 foi reduzida fortemente, chegando assim, ao desempenho final positivo.

Para a aplicação MG, a variação de desempenho foi sútil. Essa aplicação apresentou um equilíbrio entre ganhos pelo acesso linear a dados e acesso não linear. Entretanto, os ganhos de desempenho pelo compartilhamento da memória *cache* L2 não levaram a melhora de desempenho dessa aplicação.

A aplicação EP não apresenta comportamento de dependência ou compartilhamento de dados, o que pode ser comprovado nos resultados sobre invalidações MESI. Essa aplicação apresentou piora no desempenho à medida que os núcleos de processamento foram compartilhando a memória *cache* L2, sendo que, em relação à quantidade de faltas na memória *cache* L2, houve redução da organização 1Core/L2 até a 8Cores/L2, e nas demais, houve forte aumento de faltas. Esse resultado pode indicar o comportamento que ocorrerá nessas organizações se executarem aplicações distintas, onde o compartilhamento de dados não será benéfico para todas organizações.

A aplicação IS, executando nas diversas organizações desse primeiro experimento, mostrou um ganho de desempenho na organização 4Cores/L2. No entanto, essa aplicação, em relação ao comportamento de acesso a dados compartilhados, deveria ganhar conforme mais memórias *cache* fossem compartilhadas. Porém, verificando a quantidade de instruções executadas e o tempo de espera por dados da memória *cache* L1, fica claro que a aplicação é bastante pequena, e por isso, mesmo com a forte redução nas faltas da memória L2, não houve ganho no desempenho. Entretanto, podemos supor que, com aplicações de mesmo comportamento, porém maiores, poderá haver ganhos com o compartilhamento de memória *cache* L2, aumentando assim, a velocidade de acesso a dados compartilhados.

Por fim, podemos ver que a aplicação FT apresentou degradação no desempenho à medida que a memória *cache* L2 foi compartilhada. Pela quantidade de faltas de dados na memória *cache* L1, podemos ver que essa aplicação apresenta taxas de faltas superiores às demais aplicações, ou seja, sofre maior influência do aumento do tempo de acesso a dados da memória *cache* L2. Esse sobrecusto não é compensado pela redução de faltas da memória *cache* L2, o que leva o sistema a queda de desempenho.

4.2 Experimento 2 - Tamanho da Memória *Cache*

Este segundo experimento tem como objetivo avaliar a influência do tamanho da memória *cache* no compartilhamento da memória *cache* L2 entre os diversos núcleos de processamento. Para esse experimento três tamanhos de memória *cache* foram escolhidos de acordo com os melhores desempenhos apresentados no primeiro experimento (1, 2 e 4 MB). Assim, ao invés de fixar o tamanho total de memória *cache* do sistema, nesse experimento fixou-se o tamanho de cada banco de memória *cache* L2, dessa maneira, a quantidade total de memória varia de organização para organização avaliada.

Com o aumento no tamanho de memória *cache*, o sistema tende a obter menor número de faltas de dados ocasionados por capacidade e conflito de endereços, além disso, as diferentes organizações não irão sofrer aumento no tempo de acesso a dados, a não ser entre os diferentes tamanhos avaliados. Em contrapartida, conforme aumenta-se o compartilhamento, as aplicações aumentam o compartilhamento de memória *cache*, o que pode ocasionar ganhos devido ao rápido compartilhamento de dados, mas também deverão sofrer mais faltas devido a capacidade reduzida da memória *cache*.

Devido ao grande número de resultados desse experimento, essa seção foi dividida em três subseções, uma para cada tamanho de banco de memória *cache* avaliado.

O primeiro resultado a respeito da influência do tamanho da memória *cache* no compartilhamento de memórias é o espalhamento dos resultados obtidos. A Figura 4.12 apresenta os resultados estatísticos para a soma de tempo de execução de todas medições desse segundo experimento, divididos pela organização de memória *cache* L2, onde cada símbolo do gráfico representa um tamanho diferente de bloco de memória *cache*. As medidas apresentam desvio padrão representando no máximo 0,81% da média do experimento, o que pode ser considerado aceitável. Através desse primeiro resultado, podemos verificar o comportamento sem grandes diferenças da organização 1Core/L2 e 2Cores/L2 para os diversos tamanhos de memória *cache*. A partir da organização 4Cores/L2 ficam claras as diferenças de tamanho de memória *cache* no desempenho final do sistema.

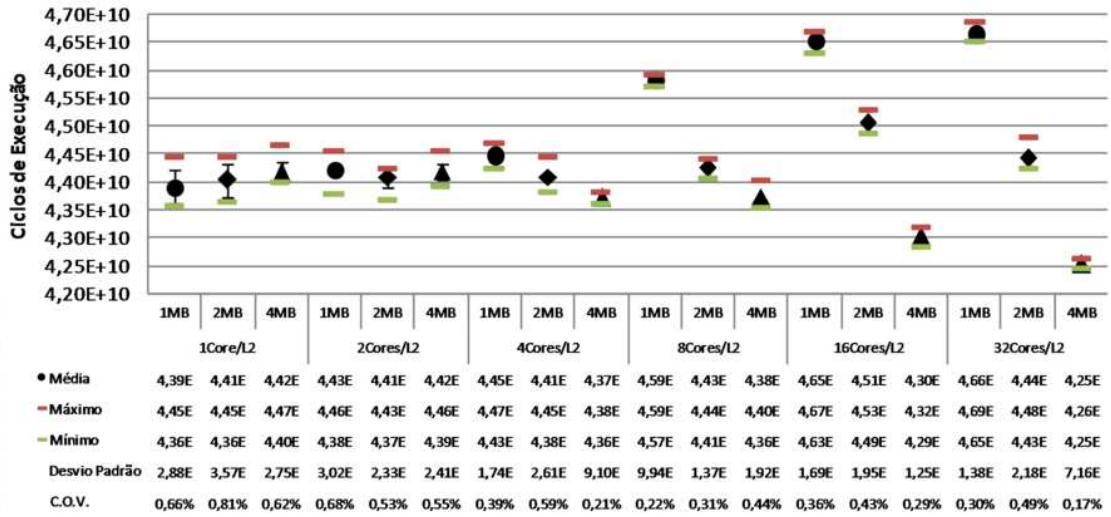


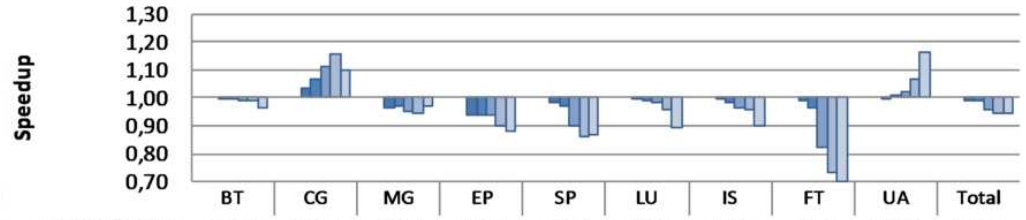
Figura 4.12: Espalhamento das medidas do segundo experimento.

4.2.1 1 MB por Memória *Cache* L2

Essa subseção apresenta resultados sobre a influência do tamanho da memória utilizando bancos de memória *cache* L2 de tamanho igual a 1 MB, variando a quantidade de núcleos de processamento compartilhando o mesmo banco de memória *cache*.

4.2.1.1 Processador Multi-Core

O gráfico apresentado na Figura 4.13 mostra o *speedup* do sistema para as diferentes organizações de memória *cache* calculado com relação à primeira organização do primeiro experimento, executando as diversas aplicações da carga de trabalho. Podemos perceber os ganhos acima de 10% para as aplicações CG e UA e as quedas de desempenho acima de 10% para diversas aplicações (EP, SP, LU IS e FT). Esse comportamento das aplicações levou a um resultado geral ruim do desempenho, de 1 MB por banco de memória *cache* L2, ressaltando a queda em 30% no desempenho da aplicação FT. Tais quedas de desempenho podem ter sido causadas por faltas de dados devido a capacidade da memória *cache*, o que será averiguado nos demais resultados.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2-1Exp	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
1Core/L2	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
2Cores/L2	0,999	1,032	0,967	0,936	0,982	0,996	0,997	0,994	1,006	0,993
4Cores/L2	0,997	1,066	0,968	0,938	0,969	0,990	0,982	0,966	1,011	0,988
8Cores/L2	0,991	1,114	0,952	0,938	0,898	0,985	0,965	0,822	1,022	0,958
16Cores/L2	0,992	1,157	0,946	0,902	0,864	0,957	0,957	0,734	1,066	0,944
32Cores/L2	0,965	1,100	0,968	0,882	0,870	0,892	0,897	0,702	1,163	0,942

Figura 4.13: *Speedup* do segundo experimento (1 MB).

Podemos ver na Figura 4.14 a quantidade de instruções executadas. A quantidade de instruções executadas nesse segundo experimento com 1 MB não apresenta variações bruscas com relação ao primeiro experimento. Pequenas mudanças são esperadas por conta do sistema operacional. Tais resultados podem ser utilizados em conjunto com os demais gráficos para geração de valores absolutos a partir de taxas e porcentagens.

4.2.1.2 Subsistema de Memória

Podemos ver o gráfico com a taxa MPKI da memória *cache* L1 na Figura 4.15, onde nota-se, em média, um comportamento estável. Porém, avaliando por aplicação, temos grandes mudanças na aplicação FT, que mesmo com pequenas variações na porcentagem de instruções de acesso à memória, a taxa MPKI apresentou grandes diferenças. A variação, em comparação ao primeiro experimento foi de 93,18 (1Core/L2) e 94,69 (32Cores/L2) para 93,18 (1Core/L2) e 76,00 (32Cores/L2) nesse segundo experimento com 1 MB por memória *cache* L2. Essas variações de faltas na aplicação FT podem ser explicadas pela variação na quantidade de instruções executadas (e.g. $3,4E+09$ para 4Cores/L2 e $4,3E+09$ para 8Cores/L2), o que pode ter levado a redução da taxa.

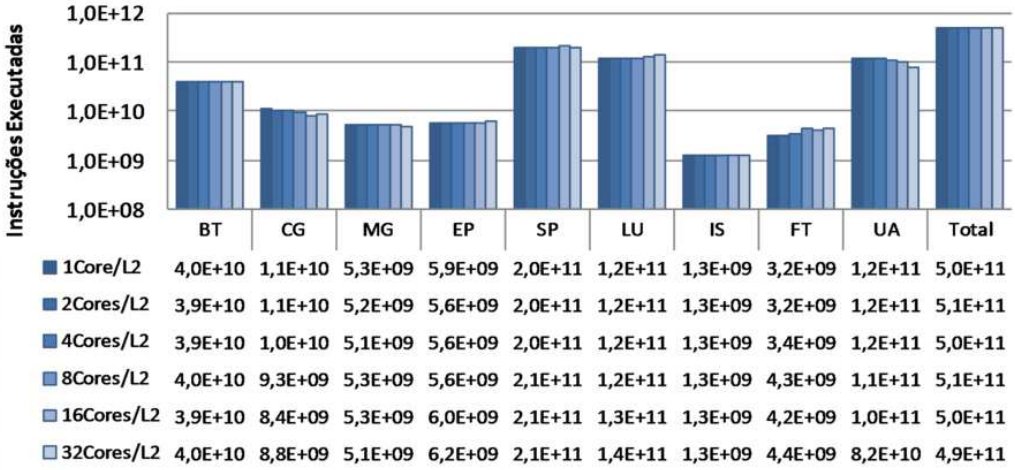


Figura 4.14: Instruções executadas do segundo experimento (1 MB).

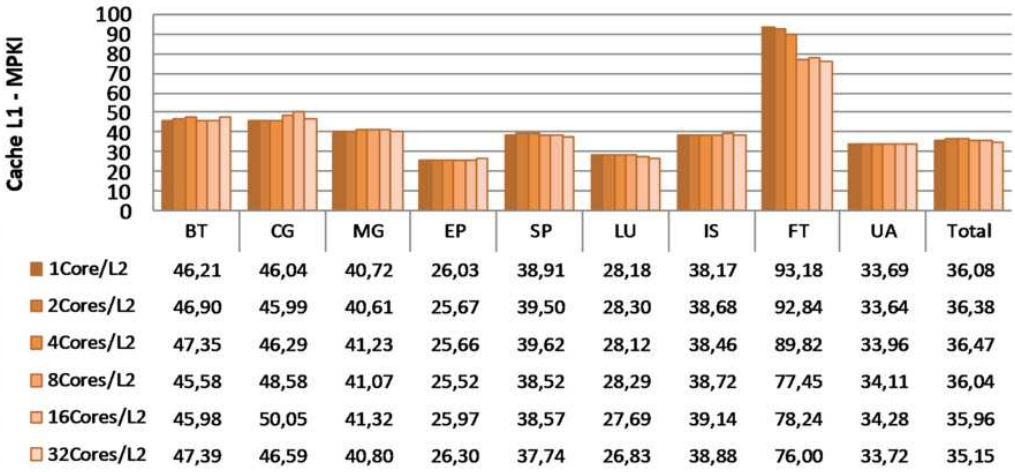
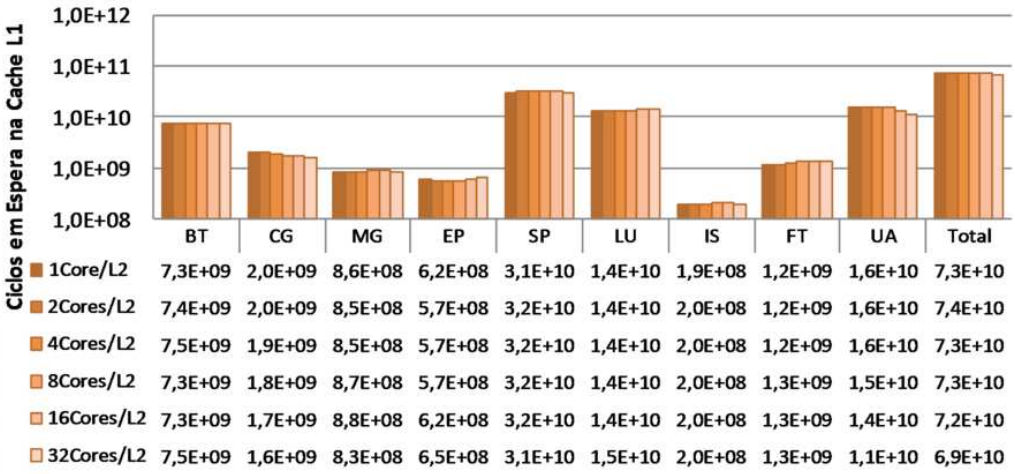


Figura 4.15: Memória *cache* L1 MPKI do segundo experimento (1 MB).

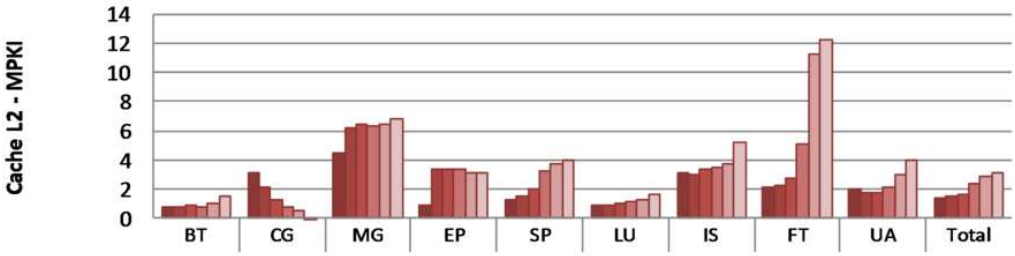
Como era de se esperar, a quantidade de ciclos perdidos por espera de dados durante faltas na memória *cache* L1, ilustrado na Figura 4.16, permaneceu sempre com valores próximos, conforme variou-se a organização de memória e manteve-se o tamanho do banco de memória *cache* fixa. Assim, toda redução de faltas na memória *cache* L2 deverá ser convertida em ganho de desempenho, já que a latência manteve-se fixa. Entretanto, a taxa de faltas é maior que no primeiro experimento, ou seja, houve mais faltas ocasionadas pela redução da quantidade total de memória disponível no sistema. Logo, o equilíbrio entre tamanho de memória *cache* e latência pode levar a ganho de desempenho.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	7,3E+09	2,0E+09	8,6E+08	6,2E+08	3,1E+10	1,4E+10	1,9E+08	1,2E+09	1,6E+10	7,3E+10
2Cores/L2	7,4E+09	2,0E+09	8,5E+08	5,7E+08	3,2E+10	1,4E+10	2,0E+08	1,2E+09	1,6E+10	7,4E+10
4Cores/L2	7,5E+09	1,9E+09	8,5E+08	5,7E+08	3,2E+10	1,4E+10	2,0E+08	1,2E+09	1,6E+10	7,3E+10
8Cores/L2	7,3E+09	1,8E+09	8,7E+08	5,7E+08	3,2E+10	1,4E+10	2,0E+08	1,3E+09	1,5E+10	7,3E+10
16Cores/L2	7,3E+09	1,7E+09	8,8E+08	6,2E+08	3,2E+10	1,4E+10	2,0E+08	1,3E+09	1,4E+10	7,2E+10
32Cores/L2	7,5E+09	1,6E+09	8,3E+08	6,5E+08	3,1E+10	1,5E+10	2,0E+08	1,3E+09	1,1E+10	6,9E+10

Figura 4.16: Ciclos de espera de dados na memória *cache* L1 do segundo experimento (1 MB).

A Figura 4.17 apresenta os valores de MPKI para memória *cache* L2. Nesse gráfico podemos notar o grande aumento da taxa de faltas nos aplicativos SP, LU, IS e FT, o que explica a queda de desempenho dessas aplicações. Notamos que uma suave redução na taxa de MPKI da memória *cache* L1 resultou no ganho de desempenho da aplicação UA, mesmo com aumento na taxa de MPKI para memória *cache* L2.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	0,78	3,11	4,48	0,86	1,27	0,88	3,04	2,08	1,92	1,37
2Cores/L2	0,76	2,08	6,20	3,39	1,54	0,91	2,94	2,28	1,77	1,47
4Cores/L2	0,81	1,30	6,50	3,37	1,96	0,96	3,30	2,76	1,77	1,64
8Cores/L2	0,80	0,80	6,36	3,36	3,24	1,10	3,41	5,14	2,15	2,30
16Cores/L2	1,00	0,46	6,40	3,14	3,78	1,28	3,75	11,29	2,97	2,80
32Cores/L2	1,51	0,03	6,78	3,06	3,93	1,65	5,25	12,28	3,95	3,12

Figura 4.17: Memória *cache* L2 MPKI do segundo experimento (1 MB).

4.2.1.3 *Consumo de Energia e Ocupação de Área*

Podemos notar que a soma de energia e potência dinâmica com estática gera um consumo total reduzido, conforme aumentou-se o compartilhamento de memória *cache* entre os núcleos de processamento. A potência total caiu em 4 vezes partindo da primeira organização até a última, como pode ser visto na Figura 4.18, que trás dados a respeito do consumo total de energia e potência do sistema de memória para esse segundo experimento com blocos de 1 MB de memória *cache*.

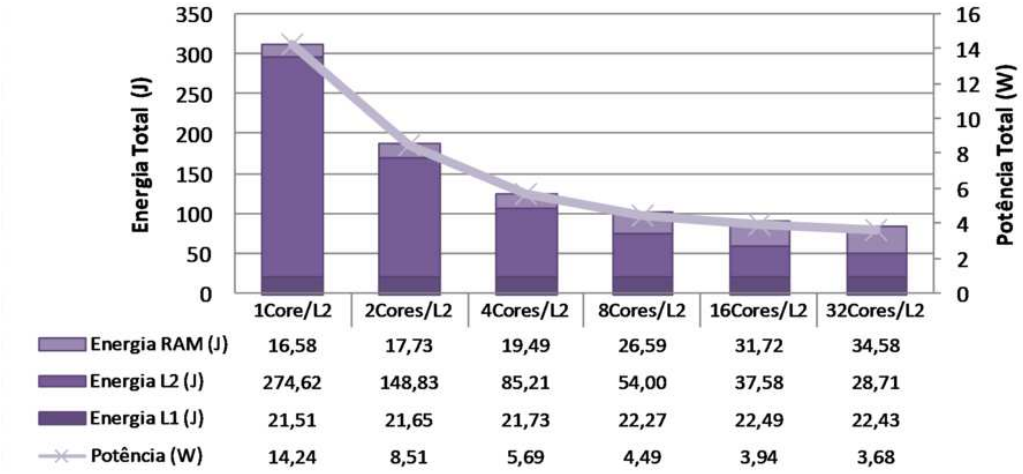


Figura 4.18: Consumo de energia e potência total do sistema de memória do segundo experimento (1 MB).

Os valores para ocupação total de área foi reduzido em 32 vezes, como esperado, uma vez que o tamanho total de memória *cache* L2 do sistema foi reduzido nessa mesma ordem (32 vezes). O gráfico com esse resultado é apresentado na Figura 4.19.

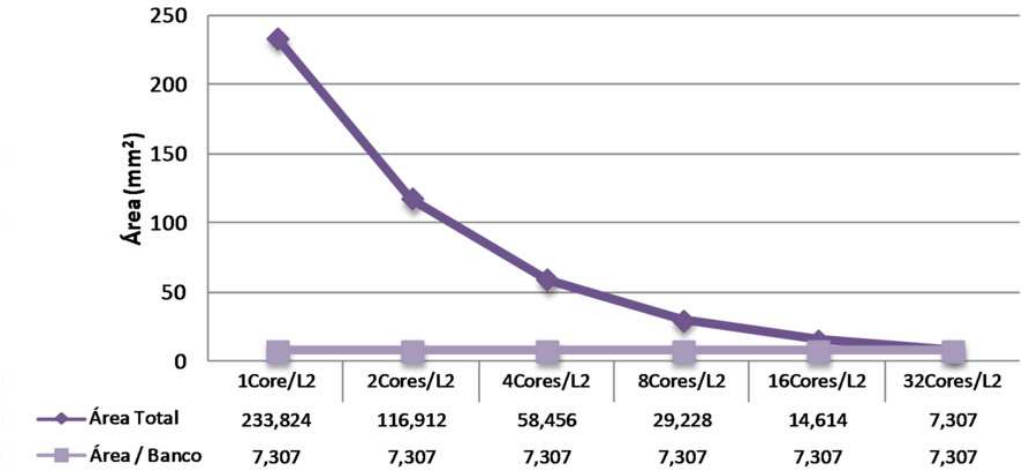
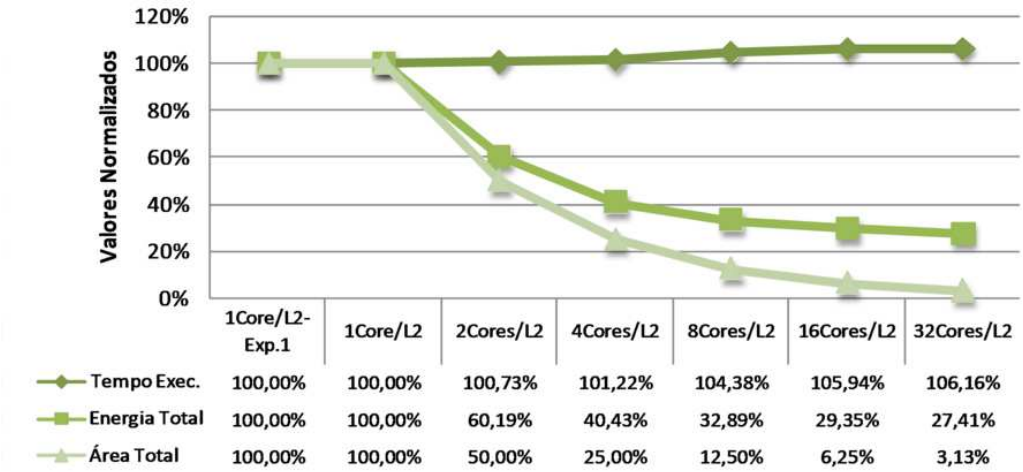


Figura 4.19: Ocupação de área pela memória *cache* L2 do segundo experimento (1 MB).

4.2.1.4 *Sumário*

O sumário contendo valores a respeito do desempenho, energia consumida e área ocupada é apresentado na Figura 4.20. Podemos ver um aumento próximo de 4% no tempo total de execução dos experimentos na organização 32Cores/L2, entretanto, esse valor foi obtido com fortes reduções no consumo de energia total (-72,59%) e ocupação de área (-96,87%).

Assim, pode-se pensar em adoção dessas arquiteturas com memória *cache* reduzida porém compartilhada entre diversos núcleos de processamento para sistemas com requisitos de baixo consumo de potência e pouca ocupação de área pelo sistema de memória *cache*.



	1Core/L2-Exp.1	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Tempo Exec.	100,00%	100,00%	100,73%	101,22%	104,38%	105,94%	106,16%
Energia Total	100,00%	100,00%	60,19%	40,43%	32,89%	29,35%	27,41%
Área Total	100,00%	100,00%	50,00%	25,00%	12,50%	6,25%	3,13%

Figura 4.20: Sumário de desempenho, consumo de energia e ocupação de área do segundo experimento (1 MB).

4.2.1.5 *Avaliação das Aplicações*

Nesse segundo experimento, com 1 MB por banco de memória *cache* L2, fica clara a importância do uso de tamanhos de memória *cache* razoáveis. Nessa subseção serão analisadas as influências de diferentes tamanhos nas aplicações.

Notamos que as aplicações, BT, MG, EP, SP, LU, IS e FT, neste experimento, sofreram perda de desempenho à medida que o compartilhamento de memória aumentou, resultado vindo do aumento de faltas ocorridas na memória *cache* L2. Note que essas aplicações citadas não tiveram o mesmo comportamento no experimento anterior. Entretanto, para esse caso onde houve forte redução de tamanho da memória, os resultados dessas foram praticamente iguais.

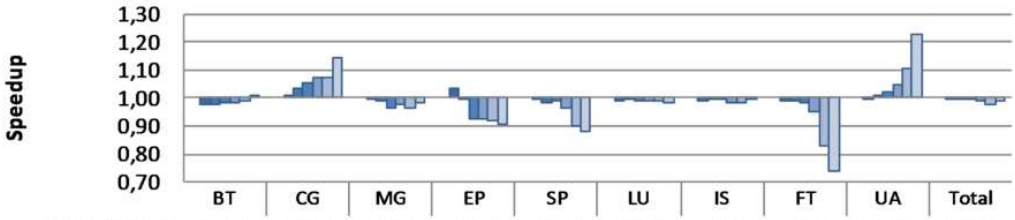
Podemos notar também que, nas aplicações CG e UA, houve ganho de desempenho à medida que o compartilhamento de memória *cache* L2 aumentou até 16Cores/L2. Esse aumento de compartilhamento não resultou em aumento de faltas de dados na memória *cache* L2 como nas demais aplicações. Assim, podemos concluir que essas aplicações de acesso não linear (pseudo randômico) à memória não sofreram grandes prejuízos pela privação de espaço e ainda conseguiram reduzir a latência para acessar dados compartilhados.

4.2.2 2 MB por Memória *Cache* L2

Essa subseção apresenta os resultados referentes ao segundo experimento variando o compartilhamento da memória *cache* L2 utilizando bancos de 2 MB.

4.2.2.1 Processador Multi-Core

O primeiro resultado desta subseção trás a variação de desempenho entre as diversas organizações de memória *cache* listado por aplicação. O gráfico da Figura 4.21 apresenta a variação de *speedup* entre as aplicações em relação à primeira organização do primeiro experimento, onde podemos notar um ganho acima de 10% para as aplicações CG e UA conforme o aumento do compartilhamento da memória *cache*, enquanto as aplicações EP, SP e FT tiveram queda acima de 9% para as modificações na organização de memória.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2-1Exp	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
1Core/L2	0,979	1,008	0,995	1,038	1,002	0,993	0,988	0,992	0,998	0,997
2Cores/L2	0,979	1,032	0,993	1,004	0,986	0,999	0,997	0,992	1,013	0,996
4Cores/L2	0,981	1,053	0,967	0,929	0,988	0,989	1,006	0,987	1,022	0,996
8Cores/L2	0,984	1,075	0,977	0,925	0,962	0,991	0,987	0,953	1,048	0,992
16Cores/L2	0,993	1,076	0,968	0,922	0,898	0,989	0,981	0,828	1,105	0,975
32Cores/L2	1,013	1,145	0,981	0,907	0,883	0,984	1,001	0,742	1,230	0,989

Figura 4.21: *Speedup* do segundo experimento (2 MB).

A variação na quantidade de instruções executadas pelos aplicativos somado ao sistema operacional é apresentada na Figura 4.22. Essa métrica ajuda a avaliar os valores totais dos gráficos de porcentagem de instruções de acesso à memória e MPKI para as memória *cache*.

4.2.2.2 Subsistema de Memória

Assim, como nos testes com 1 MB por banco de memória *cache*, a taxa de MPKI para memória *cache* L1 sofre visível variação para a aplicação FT comparando com o primeiro experimento. Para as demais aplicações existe variação mais suave entre organizações quando comparado com o primeiro experimento. Tais dados podem ser vistos na Figura 4.23, que mostra a taxa MPKI para a memória *cache* L1. Assim, como no experimento anterior, pode-se verificar grande relação entre a quantidade de instruções executadas e a variação na taxa MPKI.

Também como esperado, neste experimento a quantidade de ciclos perdidos por espera durante faltas de dados na memória *cache* L1, apresentado na Figura 4.24, sofreu pouca variação, uma vez que a latência de memória *cache* L2 se manteve estável entre as diferentes organizações de memória. Entretanto, comparando com os resultados de apenas 1 MB por banco de memória *cache*, a espera de dados ocasionou cerca de 1E+10 mais ciclos perdidos, com 2 MB por banco de memória *cache*, justamente pelo aumento na latência da memória *cache*.

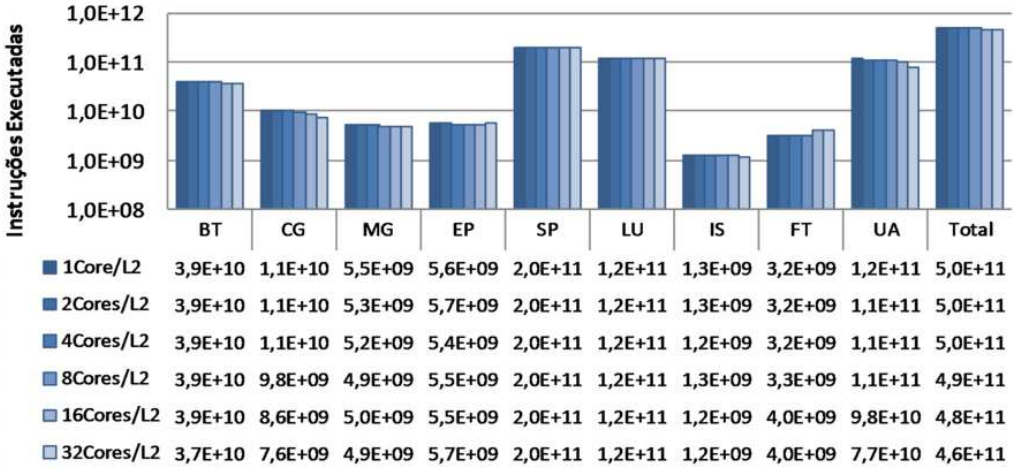


Figura 4.22: Instruções executadas do segundo experimento (2 MB).

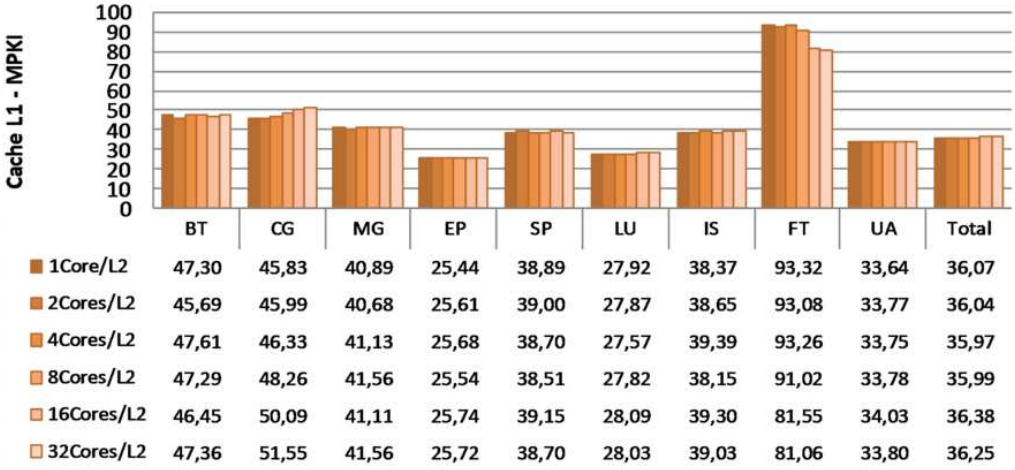


Figura 4.23: Memória *cache* L1 MPKI do segundo experimento (2 MB).

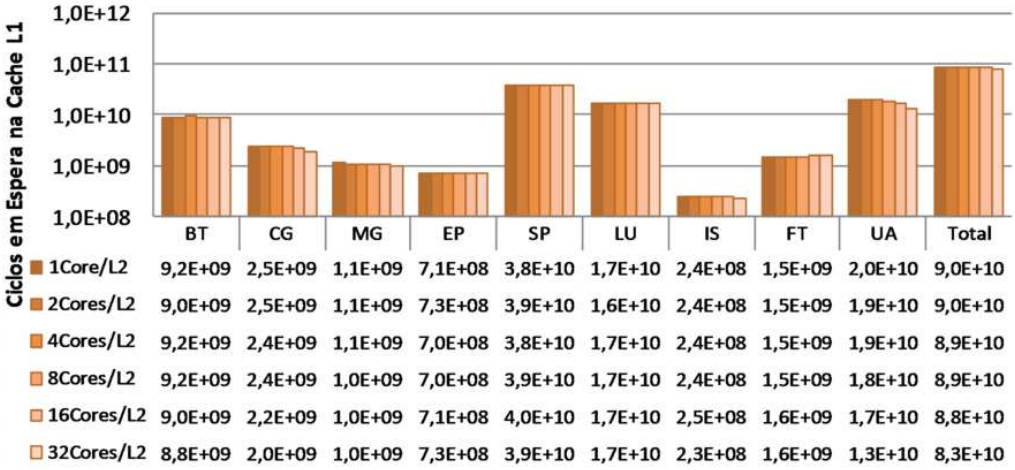
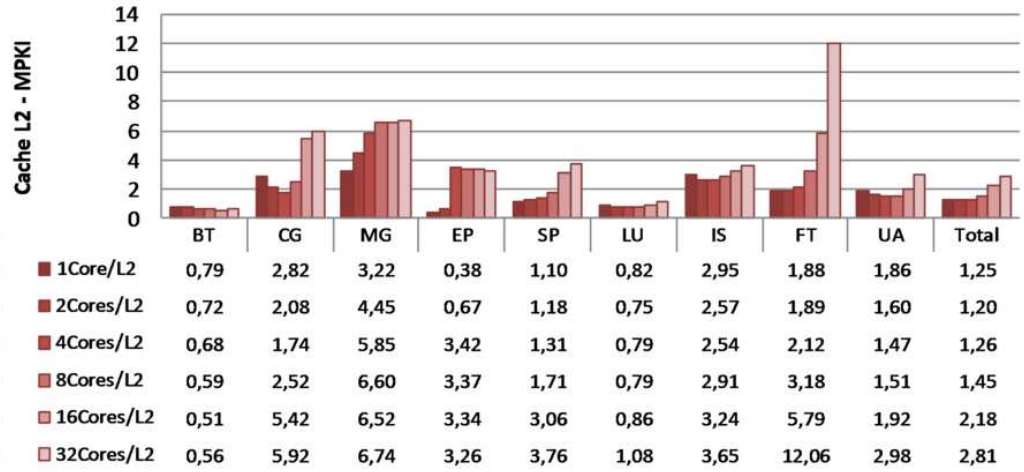


Figura 4.24: Ciclos de espera de dados na memória *cache* L1 do segundo experimento (2 MB).

Avaliando a taxa de faltas de dados na memória *cache* L2, apresentado na Figura 4.25, conforme aumentou a quantidade de núcleos compartilhando o mesmo banco de memória *cache* L2, a taxa de faltas sofreu aumento, variando de 1,25 para 2,81, em média, para as organizações 1Core/L2 e 32Cores/L2, respectivamente. Podemos notar que essas taxas são mais baixas que as apresentadas quando utilizou-se 1 MB por banco de memória *cache*. Assim, este foi o maior motivo para a melhora no desempenho dos experimentos com 2 MB em relação aos executados com apenas 1 MB por banco de memória *cache*.

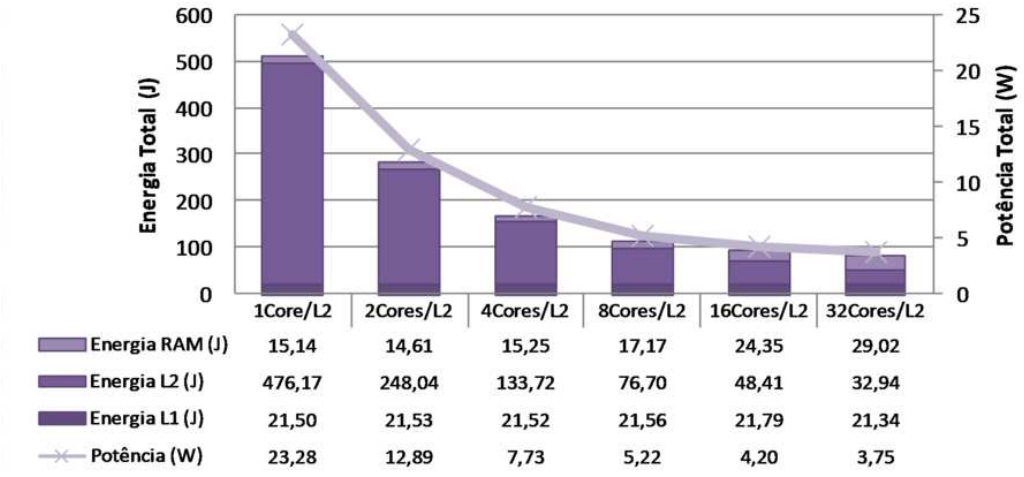


	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	0,79	2,82	3,22	0,38	1,10	0,82	2,95	1,88	1,86	1,25
2Cores/L2	0,72	2,08	4,45	0,67	1,18	0,75	2,57	1,89	1,60	1,20
4Cores/L2	0,68	1,74	5,85	3,42	1,31	0,79	2,54	2,12	1,47	1,26
8Cores/L2	0,59	2,52	6,60	3,37	1,71	0,79	2,91	3,18	1,51	1,45
16Cores/L2	0,51	5,42	6,52	3,34	3,06	0,86	3,24	5,79	1,92	2,18
32Cores/L2	0,56	5,92	6,74	3,26	3,76	1,08	3,65	12,06	2,98	2,81

Figura 4.25: Memória *cache* L2 MPKI do segundo experimento (2 MB).

4.2.2.3 Consumo de Energia e Ocupação de Área

Devido ao aumento de consumo estático, o consumo total de potência foi maior para as organizações com 2 MB se comparados com as que utilizaram 1 MB apenas. Esse comportamento pode ser visto na Figura 4.26 que mostra o consumo de energia e potência total dividido entre memórias *cache* e memória principal.



	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Energia RAM (J)	15,14	14,61	15,25	17,17	24,35	29,02
Energia L2 (J)	476,17	248,04	133,72	76,70	48,41	32,94
Energia L1 (J)	21,50	21,53	21,52	21,56	21,79	21,34
Potência (W)	23,28	12,89	7,73	5,22	4,20	3,75

Figura 4.26: Consumo de energia e potência total do sistema de memória do segundo experimento (2 MB).

A ocupação de área, como esperado, foi reduzido a uma taxa de 32 vezes, partindo da primeira à última organização de memória avaliada, 1Core/L2 e 32Cores/L2 respectivamente, como é visto na Figura 4.27, que mostra o gráfico com ocupação de área pela memória *cache* L2 para os diversos compartilhamentos utilizando bancos de 2 MB por memória *cache*, variando de 64 MB totais na primeira organização (1Core/L2) para apenas 2 MB na última organização avaliada (32Cores/L2).

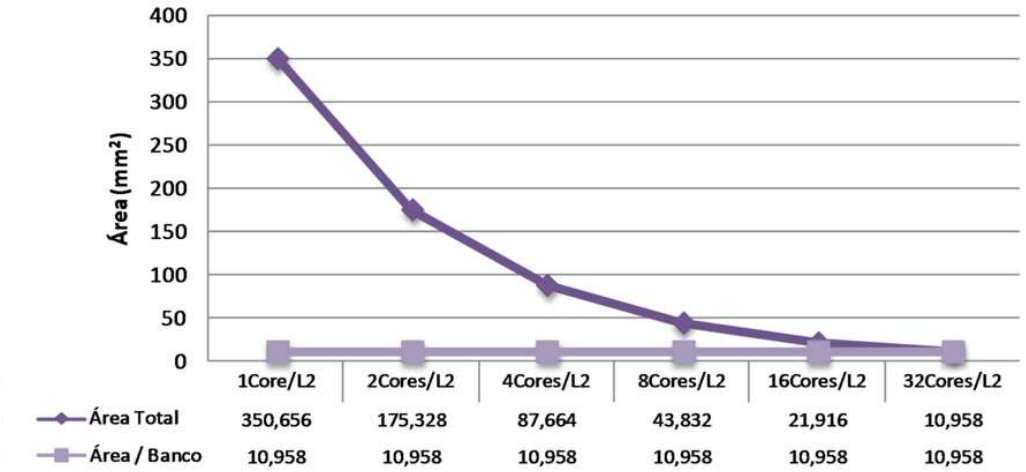


Figura 4.27: Área ocupada pela memória *cache* L2 do segundo experimento (2 MB).

4.2.2.4 Sumário

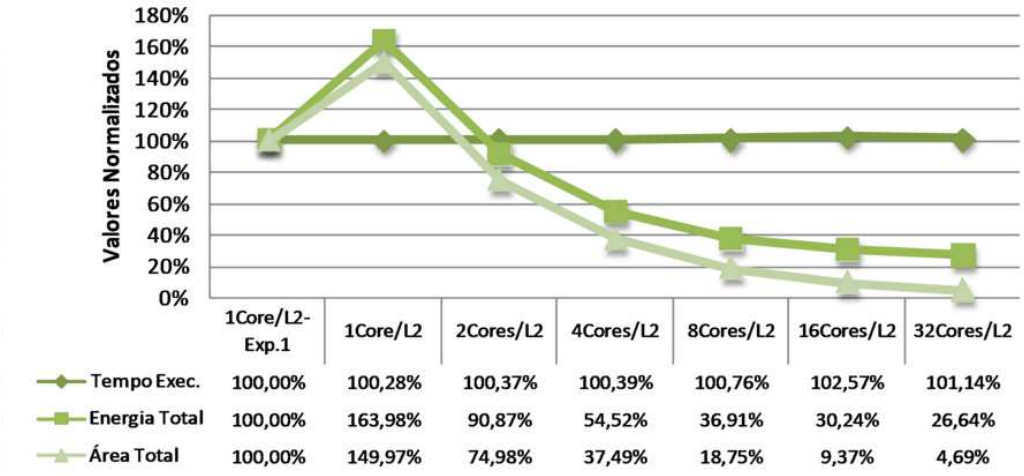
A Figura 4.28 apresenta um sumário sobre os dados de desempenho, consumo de energia e ocupação de área deste experimento. Comparando-se com a organização 1Core/L2 do primeiro experimento, pode-se ver a grande redução de área ocupada (-95,31%) e a redução de -73,36% no consumo de energia total para a organização 32Cores/L2, onde o aumento no tempo de execução da carga de trabalho foi de apenas 1,14% com relação a primeira organização.

Percebe-se com estes resultados, o equilíbrio que deve haver entre faltas de dados e tempo de acesso. Além disso, uma boa configuração dos fatores pode gerar grande redução de área e potência sem grandes sobre-custos no desempenho final.

4.2.2.5 Avaliação das Aplicações

Esse experimento com 2 MB por banco de memória *cache* L2 será avaliado nesta subseção pelo ponto de vista das aplicações da carga de trabalho executadas nas diversas organizações de memória cache.

Começando pelas aplicações BT, SP e LU, podemos notar que as aplicações BT e LU apresentaram pequeno ganho de desempenho na organização 2Cores/L2, em relação à primeira organização. Esse ganho foi gerado pela redução na taxa de faltas de dados na memória *cache* L2, sendo que a aplicação BT continuou com os ganhos para as demais organizações. As aplicações SP e LU, em geral tiveram queda no desempenho à medida que o compartilhamento de memória foi incrementado. Conseguimos verificar que a aplicação difere das outras duas na porcentagem de instruções de acesso a dados. Logo, o compartilhamento de dados auxiliou na redução da taxa MPKI da memória *cache* L2, levando a ganhos de desempenho.



	1Core/L2-Exp.1	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Tempo Exec.	100,00%	100,28%	100,37%	100,39%	100,76%	102,57%	101,14%
Energia Total	100,00%	163,98%	90,87%	54,52%	36,91%	30,24%	26,64%
Área Total	100,00%	149,97%	74,98%	37,49%	18,75%	9,37%	4,69%

Figura 4.28: Sumário de desempenho, consumo de energia e ocupação de área do segundo experimento (2 MB).

As aplicações CG e UA, assim como no experimento utilizando 1 MB por banco de memória *cache*, obtiveram bons ganhos de desempenho para todas as organizações, mostrando que para aplicações com comportamento de acesso a dados esparsos, o compartilhamento de memória compensa, haja visto que essas aplicações não sofrem tanto com conflitos de endereço na memória *cache* e ainda conseguem lucrar com o rápido acesso a dados compartilhados. Entretanto, essas aplicações sofrem reais reduções na taxa de faltas de dados da memória *cache* L2 até a organização 4Cores/L2, e a partir dessa, os ganhos são ocasionados mais por variações na quantidade de instruções executadas que por redução nas faltas de dados.

A aplicação MG, a qual trabalha sobre dados contíguos fazendo acessos lineares e não-lineares, apresentou queda de desempenho com o compartilhamento de memória *cache* L2, ocasionado principalmente pelo aumento na taxa de faltas de dados da memória *cache* L2 por instruções executadas.

Para a aplicação EP, a queda de desempenho foi mais acentuada, uma vez que este aplicativo tende a não utilizar dados compartilhados. Logo, a redução no espaço total disponível gerou queda de desempenho.

A aplicação IS, neste experimento obteve ganhos de desempenho para diversas organizações. Entretanto, assim como nas aplicações CG e UA, essa aplicação apresentou redução de MPKI até a organização 4Cores/L2, e então sofreu com aumento na taxa de MPKI da memória *cache* L2, onde obteve ganhos apenas por variações ocorridas na quantidade de intruções executadas.

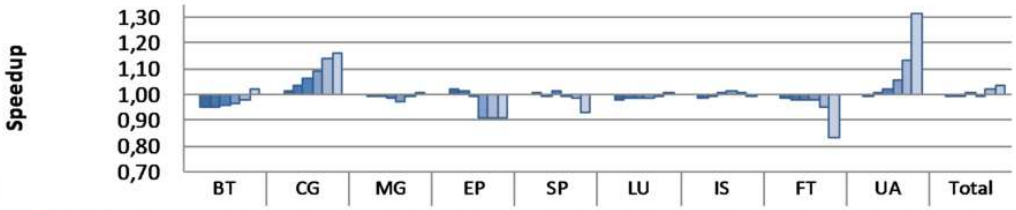
Por fim, a aplicação FT apresentou forte piora no desempenho, sendo ocasionada pelo aumento na taxa MPKI da memória *cache* L2. Porém, até a organização 4Cores/L2 as perdas foram pequenas, seguidas de grandes perdas para as demais organizações. Diversas anomalias nos gráficos de MPKI da memória *cache* L1 foram ocasionadas pelo aumento na quantidade de instruções executadas. Assim, as taxas relacionadas às instruções tiveram seus valores afetados.

4.2.3 4 MB por Memória *Cache* L2

Nesta subseção serão apresentados os resultados acerca do segundo experimento utilizando memória *cache* L2 formada por bancos de 4 MB cada, com organizações de variados compartilhamentos de memória *cache* L2.

4.2.3.1 Processador Multi-Core

O primeiro resultado desta subseção é dado em *speedup* das diversas aplicações executando em diferentes organizações de memória *cache* obtidas em relação à primeira organização do primeiro experimento. Nesse primeiro resultado já podemos ver alguns valores indicando ganho de desempenho em 6 (BT, CG, MG, LU, IS e UA) das 9 aplicações de nossa carga de trabalho, como pode ser visto na Figura 4.29 que apresenta os ganhos acima de 15% para as aplicações CG e UA e perdas de mais de 10% nas aplicações EP e FT. O resultado final é o ganho de desempenho para as organizações com 4 ou mais processadores compartilhando o mesmo banco de memória *cache* L2.



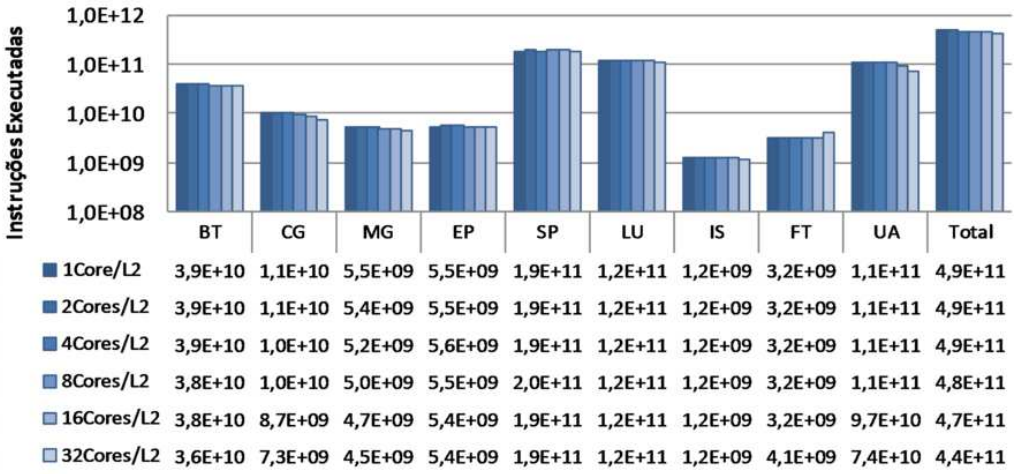
	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2-1Exp	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
1Core/L2	0,954	1,012	0,993	1,023	1,010	0,981	0,984	0,988	0,996	0,994
2Cores/L2	0,955	1,037	0,996	1,015	0,999	0,985	1,000	0,982	1,006	0,994
4Cores/L2	0,959	1,066	0,987	0,993	1,012	0,989	1,006	0,981	1,025	1,004
8Cores/L2	0,969	1,091	0,973	0,907	0,991	0,989	1,013	0,976	1,057	1,004
16Cores/L2	0,979	1,143	0,992	0,907	0,986	0,997	1,008	0,954	1,131	1,021
32Cores/L2	1,024	1,161	1,010	0,913	0,930	1,008	1,003	0,831	1,316	1,033

Figura 4.29: *Speedup* do segundo experimento (4 MB).

A variação no número de instruções executadas pelas aplicações está apresentada no gráfico da Figura 4.30. Podemos notar que existe grande ligação entre *speedup* e número de instruções executadas, uma vez que as aplicações executadas nas diversas organizações de memória *cache* são as mesmas, com dados de entrada idênticos para cada aplicação em todas execuções. Considerando que o simulador não gera instruções durante as latências de memória, podemos concluir que essa variação de instruções segue a tendência do *speedup* por influência pura das instruções adicionais executadas pelo sistema operacional.

4.2.3.2 Subsistema de Memória

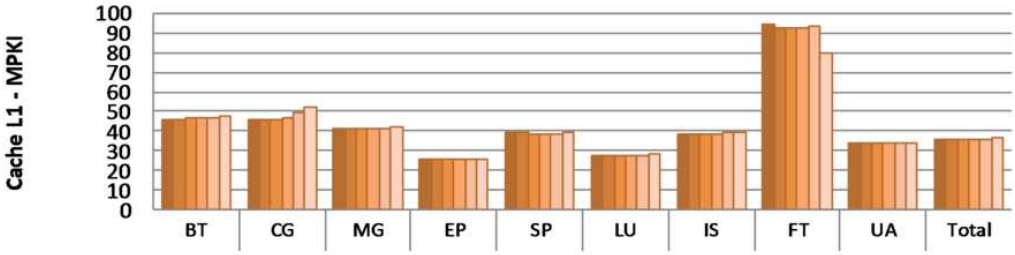
A taxa de faltas de dados na memória *cache* L1 por milhares de instruções executadas é apresentada na Figura 4.31 onde vemos a variação da taxa de faltas entre as diferentes organizações de memória *cache*. Podemos notar que o comportamento anômalo da aplicação FT se repete, mas nesse caso, apenas para a organização 32Cores/L2. Verificando assim que, como no caso 1 MB e 2 MB por banco de memória *cache* L2, existe forte correlação entre instruções executadas, instruções de memória e a taxa MPKI para memória *cache* L1. Entretanto os valores totais sofrem pequenas modificações, o que nos levar a



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	3,9E+10	1,1E+10	5,5E+09	5,5E+09	1,9E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	4,9E+11
2Cores/L2	3,9E+10	1,1E+10	5,4E+09	5,5E+09	1,9E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	4,9E+11
4Cores/L2	3,9E+10	1,0E+10	5,2E+09	5,6E+09	1,9E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	4,9E+11
8Cores/L2	3,8E+10	1,0E+10	5,0E+09	5,5E+09	2,0E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	4,8E+11
16Cores/L2	3,8E+10	8,7E+09	4,7E+09	5,4E+09	1,9E+11	1,2E+11	1,2E+09	3,2E+09	9,7E+10	4,7E+11
32Cores/L2	3,6E+10	7,3E+09	4,5E+09	5,4E+09	1,9E+11	1,2E+11	1,2E+09	4,1E+09	7,4E+10	4,4E+11

Figura 4.30: Instruções executadas do segundo experimento (4 MB).

considerar que tais variações podem vir do sistema operacional assim como a variação na quantidade de instruções executadas.

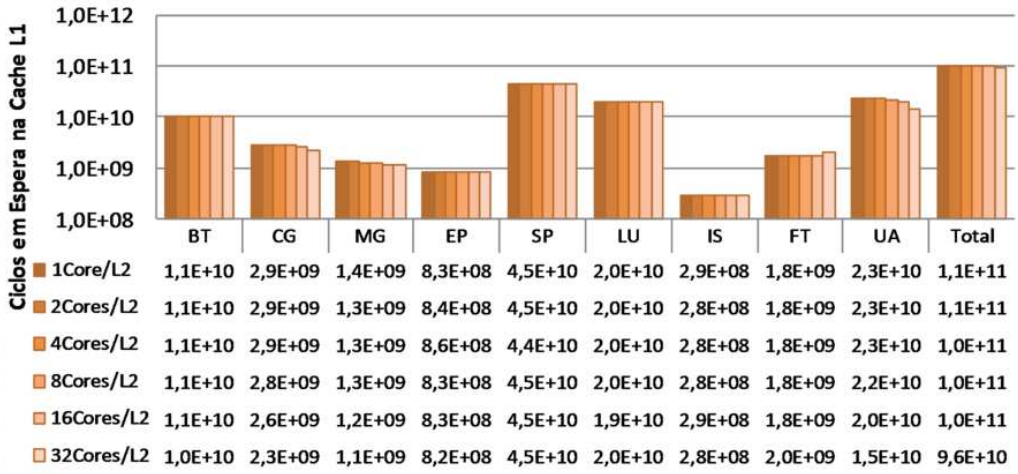


	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	45,61	45,71	41,24	25,44	39,55	27,80	38,81	94,20	33,71	36,16
2Cores/L2	45,86	46,02	41,30	25,34	39,17	27,83	38,79	93,09	33,60	36,04
4Cores/L2	46,41	46,29	41,16	25,66	38,39	27,77	38,74	92,45	33,66	35,77
8Cores/L2	46,65	46,96	41,66	25,44	38,69	27,57	38,90	92,75	33,57	35,87
16Cores/L2	46,39	49,86	41,59	25,54	38,90	27,20	39,18	93,23	33,60	35,92
32Cores/L2	47,50	52,17	42,08	25,47	39,78	28,27	39,25	80,04	33,78	36,79

Figura 4.31: Memória *cache* L1 MPKI do segundo experimento (4 MB).

Assim como nos experimentos com 1 MB e 2 MB por banco de memória *cache*, este experimento com 4 MB também apresenta pouca variação entre as organizações para a quantidade de ciclos perdidos por faltas de dados na memória *cache* L1. Como nos casos anteriores, esse comportamento era esperado, uma vez que a latência da memória *cache* L2 não sofreu modificações entre as organizações. Entretanto, os valores totais sofrem aumentos, com relação aos experimentos 1 MB e 2 MB por banco de memória *cache*, fruto do aumento da latência na memória *cache* L2.

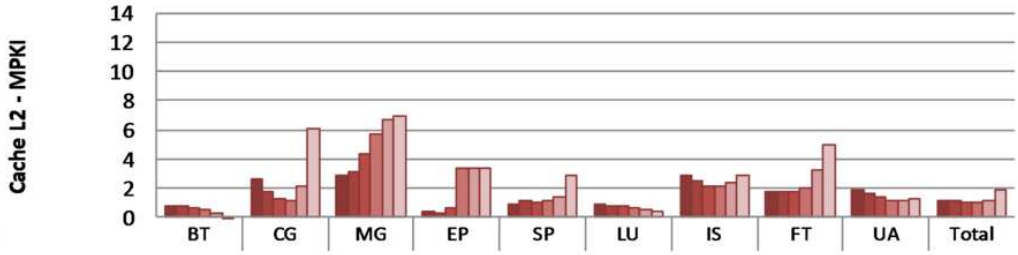
Podemos ver na Figura 4.33 um gráfico com a taxa de faltas de dados na memória *cache* L2. Podemos notar que a taxa de MPKI é de 0,99 para a organização 4Cores/L2, sendo que esta taxa é a menor obtida em todo o segundo experimento, sendo influenciada pelo tamanho da memória *cache*. Além desse bom resultado para a organização 4Cores/L2, as demais organizações também apresentaram bons resultados em comparação com os demais testes efetuados no nesse segundo experimento. Embora as aplicações CG e UA



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	1,1E+10	2,9E+09	1,4E+09	8,3E+08	4,5E+10	2,0E+10	2,9E+08	1,8E+09	2,3E+10	1,1E+11
2Cores/L2	1,1E+10	2,9E+09	1,3E+09	8,4E+08	4,5E+10	2,0E+10	2,8E+08	1,8E+09	2,3E+10	1,1E+11
4Cores/L2	1,1E+10	2,9E+09	1,3E+09	8,6E+08	4,4E+10	2,0E+10	2,8E+08	1,8E+09	2,3E+10	1,0E+11
8Cores/L2	1,1E+10	2,8E+09	1,3E+09	8,3E+08	4,5E+10	2,0E+10	2,8E+08	1,8E+09	2,2E+10	1,0E+11
16Cores/L2	1,1E+10	2,6E+09	1,2E+09	8,3E+08	4,5E+10	1,9E+10	2,9E+08	1,8E+09	2,0E+10	1,0E+11
32Cores/L2	1,0E+10	2,3E+09	1,1E+09	8,2E+08	4,5E+10	2,0E+10	2,8E+08	2,0E+09	1,5E+10	9,6E+10

Figura 4.32: Ciclos de espera de dados na memória *cache* L1 do segundo experimento (4 MB).

apresentem aumento na taxa de MPKI para algumas organizações, esse aumento na taxa advém da redução de instruções executadas e não causou degradação no desempenho.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	0,78	2,56	2,91	0,34	0,92	0,82	2,89	1,78	1,82	1,16
2Cores/L2	0,72	1,80	3,15	0,29	1,06	0,75	2,44	1,76	1,59	1,12
4Cores/L2	0,66	1,30	4,33	0,58	0,93	0,68	2,15	1,76	1,31	0,99
8Cores/L2	0,49	1,09	5,70	3,37	1,07	0,61	2,06	2,04	1,17	1,02
16Cores/L2	0,29	2,05	6,73	3,36	1,36	0,49	2,37	3,16	1,08	1,10
32Cores/L2	0,04	6,04	6,90	3,39	2,91	0,45	2,85	4,92	1,25	1,87

Figura 4.33: Memória *cache* L2 MPKI do segundo experimento (4 MB).

4.2.3.3 Consumo de Energia e Ocupação de Área

O gráfico da Figura 4.34 apresenta valores de consumo de energia e potência total do sistema de memória para as diversas organizações de memória *cache* L2.

Podemos notar a redução acentuada no consumo de potência para todas as organizações, onde mesmo a memória principal consumindo mais potência, as reduções no consumo por parte da memória *cache* L2 fizeram a diferença para o consumo final do sistema.

Como era esperado, tal redução é ocasionada principalmente pela redução no consumo estático, ou seja, da redução no tamanho total da memória *cache* L2.

A ocupação de área do sistema assim como as demais ocupações de área deste segundo experimento, tiveram uma redução de 32 vezes na área total ocupada, uma vez que

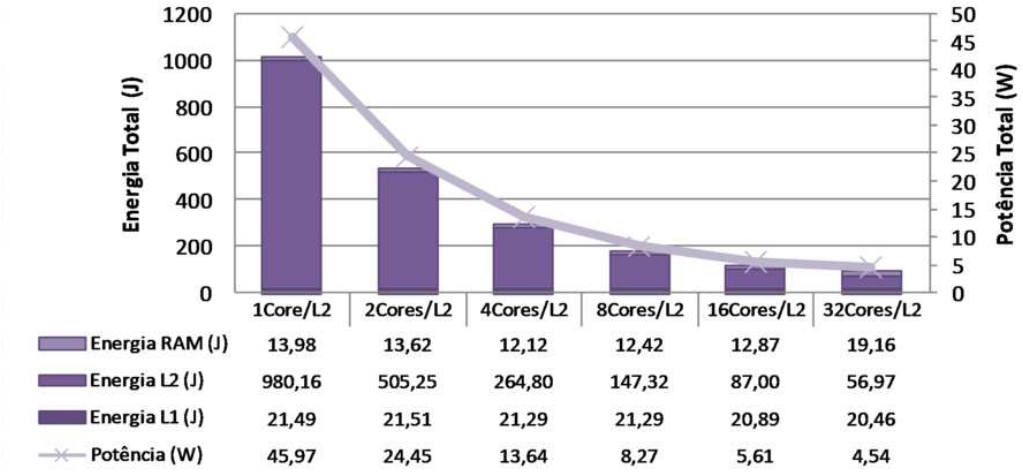


Figura 4.34: Consumo de energia e potência total do sistema de memória do segundo experimento (4 MB).

reduziu-se a área da memória *cache* L2 de 32 bancos de 4 MB na organização 1Core/L2 cada para apenas 1 banco de 4 MB na organização 32Cores/L2.

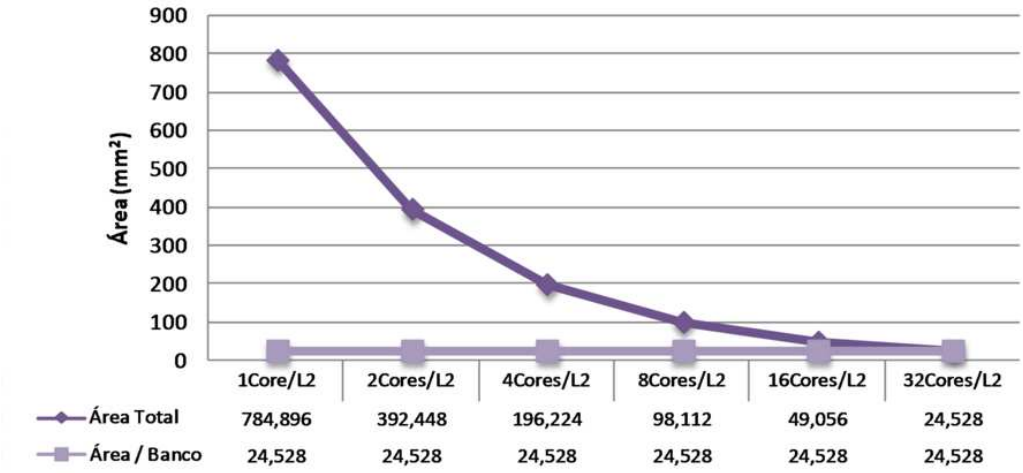


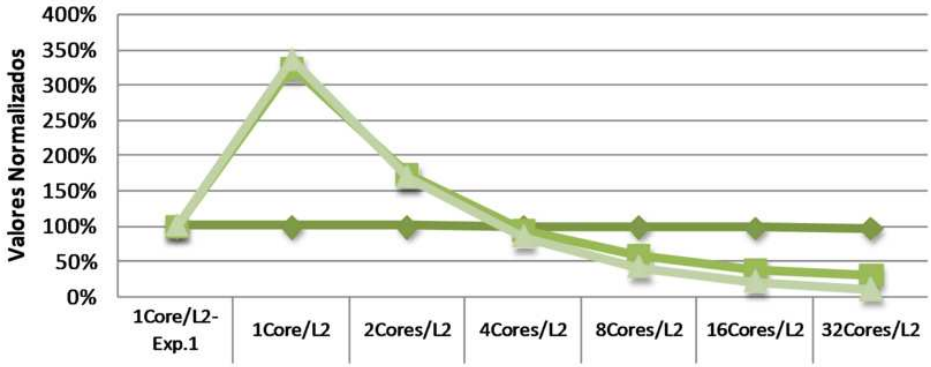
Figura 4.35: Área ocupada pela memória *cache* L2 do segundo experimento (4 MB).

4.2.3.4 Sumário

Os valores de sumário desta seção são apresentados no gráfico da Figura 4.36, onde verficamos a boa combinação entre compartilhamento de memória *cache* e tamanho lógico (4 MB por banco) da memória *cache* L2. Assim, comparando com a primeira organização do primeiro experimento, os valores de tempo de execução apresentaram redução de -3,16% na organização 32Cores/L2, redução de -69,11% no consumo de energia total, e redução de -89,51% na ocupação de área pela memória *cache* L2.

Podemos ainda concluir, que esse experimento apresentou bons resultados pois conseguiu obter um bom equilíbrio entre tamanho de memória *cache*, o que não penalizou o sistema com muitas faltas por capacidade, e o tempo de acesso a dados na memória *cache* L2. Além disso, o compartilhamento da memória *cache* entre os 32 núcleos possibilitou

reduzir ainda mais o número de faltas, o consumo de energia e a área ocupada.



	1Core/L2-Exp.1	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Tempo Exec.	100,00%	100,58%	100,62%	99,55%	99,61%	97,96%	96,84%
Energia Total	100,00%	324,78%	172,80%	95,36%	57,89%	38,61%	30,89%
Área Total	100,00%	335,68%	167,84%	83,92%	41,96%	20,98%	10,49%

Figura 4.36: Sumário de desempenho, consumo de energia e ocupação de área do segundo experimento (4 MB).

4.2.3.5 Avaliação das Aplicações

Essa subseção apresenta avaliações sobre as aplicações executadas nesse experimento com bancos de memória *cache* de tamanho igual a 4 MB.

Neste experimento, as aplicações BT, SP e LU, apresentam ganhos de desempenho pelo menos em uma organização (4Cores/L2), sendo que os ganhos estão presentes em todas organizações executando as aplicações BT e LU. Mesmo que a taxa MPKI varie pouco entre as organizações para a aplicação SP, a última organização sofreu forte aumento na taxa de faltas. Assim, podemos ver que o ganho para essa aplicação ocorreu apenas por queda na taxa de faltas da memória *cache* L1 na organização 4Cores/L2. Logo, podemos concluir que as aplicações de acesso linear a dados que apresentem menor compartilhamento em relação à computação de dados terá queda de desempenho conforme aumenta-se o compartilhamento de memória.

Para as aplicações CG e UA de acesso não linear a dados, os ganhos foram ainda maiores. Podemos ver que o aumento no tamanho do banco de memória *cache* L2 fez com que a organização 8Cores/L2 obtivesse menores taxas MPKI da memória *cache* L2.

A aplicação MG apresentou ganho de desempenho para a segunda (2Cores/L2) e a última organização (32Cores/L2) avaliada, sendo que esses ganhos podem ser relacionados ao compartilhamento de dados, que nesse caso, para compensar o tamanho reduzido de memória *cache*, o compartilhamento deve ser total, reduzindo assim a quantidade de invalidações entre bancos de memória.

Nas aplicações EP e FT notamos comportamento igual aos experimentos anteriores com 1 MB e 2 MB por banco de memória *cache* L2, onde houve perdas de desempenho conforme aumentou-se a quantidade de núcleos em uma mesma memória *cache* L2.

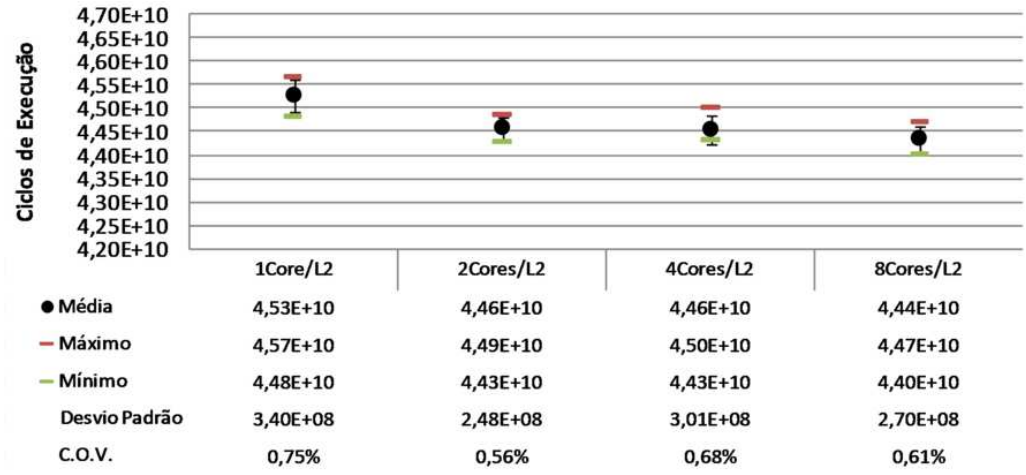
Por fim, a aplicação IS mostrou ganhos de desempenho em todas organizações, sendo a organização 8Cores/L2 a de melhor benefício. Assim, notamos o bom equilíbrio entre tempo de acesso à memória *cache* e tamanho dos blocos. Entretanto, para aplicações nativamente paralelas, como é o caso da EP, o compartilhamento de dados não é benéfico, pois não há ganhos por compartilhamento de dados.

4.3 Experimento 3 - Associatividade da Memória *Cache*

O aumento de associatividade na memória *cache* L2 proposto neste terceiro experimento, tem como objetivo avaliar a influência no aumento da localidade temporal da memória *cache*, além da redução de faltas por conflitos de endereço que são inerentes do aumento da associatividade.

Este experimento foi proposto com base no primeiro experimento. Assim, apenas as organizações 1Core/L2, 2cores/L2, 4Cores/L2 e 8 Cores/L2, são avaliadas uma vez que no primeiro experimento houve ganho na organização 4Cores/L2, por isso, tentamos com o aumento associatividade aumentar o ganho de desempenho para até a organização de 8Cores/L2. Neste experimento, o tamanho total de memória *cache* entre todas organizações foi fixado em 32 MB, assim como no primeiro experimento, entretanto a associatividade passou de 8 vias para 16 vias.

O primeiro resultado refere-se ao espalhamento das medidas e está ilustrado na Figura 4.37, onde aparecem os valores de média, máximo, mínimo, desvio padrão e coeficiente de variação das diversas medidas feitas para as diferentes organizações de memória. Podemos ver que este experimento apresenta valores de médias de tempo de execução acima do experimentos anteriores. Além disso, apenas a primeira organização é claramente diferente das demais, uma vez que os valores das médias com desvio padrão das oganizações 2cores/L2, 4Cores/L2 e 8Cores/L2 são muito próximos.



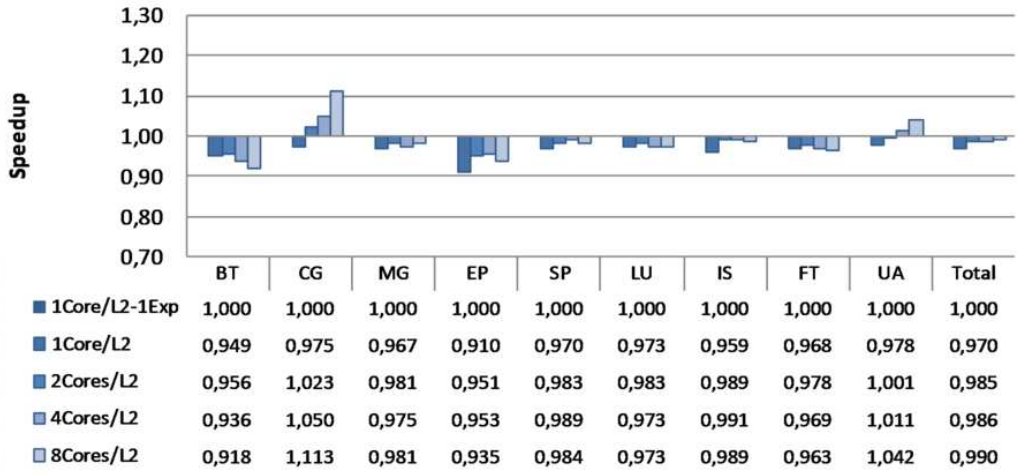
	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2
Média	4,53E+10	4,46E+10	4,46E+10	4,44E+10
Máximo	4,57E+10	4,49E+10	4,50E+10	4,47E+10
Mínimo	4,48E+10	4,43E+10	4,43E+10	4,40E+10
Desvio Padrão	3,40E+08	2,48E+08	3,01E+08	2,70E+08
C.O.V.	0,75%	0,56%	0,68%	0,61%

Figura 4.37: Espalhamento das medidas do terceiro experimento.

4.3.1 Processador *Multi-Core*

O gráfico com o *speedup* das aplicações neste terceiro experimento comparado com o a primeira organização do primeiro experimento é apresentado na Figura 4.38, onde é possível notar o aumento de desempenho da primeira organização de memória 1Core/L2 deste experimento para as demais organizações avaliadas. Entretanto, é necessário ressaltar que, mesmo apresentando ganhos de desempenho com o aumento no compartilhamento da memória *cache*, este experimento obteve desempenho inferior ao primeiro experimento. Outro ponto importante a se notar neste experimento é que, partindo da primeira organização 1Core/L2, a execução de todas as aplicações tiveram ganhos de desempenho até a última organização 8Cores/L2, porém, mesmo com essas melhorias,

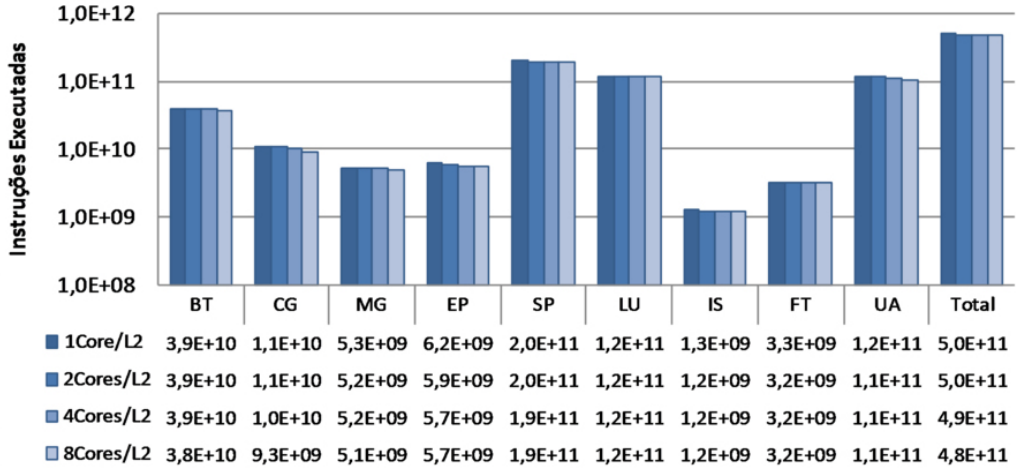
comparado com a primeira organização do primeiro experimento, não houveram ganhos de desempenho final no sistema.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2-1Exp	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
1Core/L2	0,949	0,975	0,967	0,910	0,970	0,973	0,959	0,968	0,978	0,970
2Cores/L2	0,956	1,023	0,981	0,951	0,983	0,983	0,989	0,978	1,001	0,985
4Cores/L2	0,936	1,050	0,975	0,953	0,989	0,973	0,991	0,969	1,011	0,986
8Cores/L2	0,918	1,113	0,981	0,935	0,984	0,973	0,989	0,963	1,042	0,990

Figura 4.38: *Speedup* do terceiro experimento.

A variação na quantidade de instruções executadas nas diversas aplicações, dividida em aplicações e organizações avaliadas é apresentada no gráfico da Figura 4.39. Podemos observar a variação próxima aos experimentos anteriores.



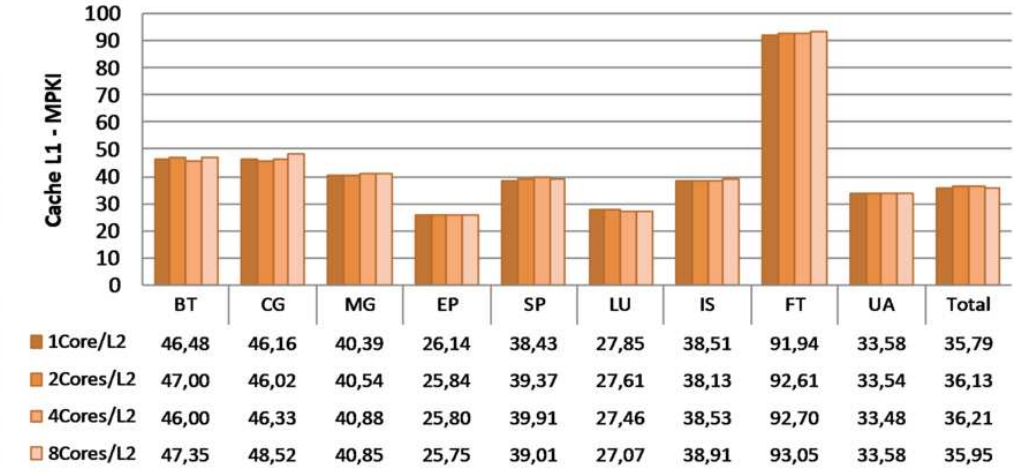
	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	3,9E+10	1,1E+10	5,3E+09	6,2E+09	2,0E+11	1,2E+11	1,3E+09	3,3E+09	1,2E+11	5,0E+11
2Cores/L2	3,9E+10	1,1E+10	5,2E+09	5,9E+09	2,0E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	5,0E+11
4Cores/L2	3,9E+10	1,0E+10	5,2E+09	5,7E+09	1,9E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	4,9E+11
8Cores/L2	3,8E+10	9,3E+09	5,1E+09	5,7E+09	1,9E+11	1,2E+11	1,2E+09	3,2E+09	1,1E+11	4,8E+11

Figura 4.39: Instruções executadas do terceiro experimento.

4.3.2 Subsistema de Memória

O gráfico da Figura 4.40 mostra a taxa de faltas de dados na memória *cache* L1 em relação ao número de instruções executadas. Podemos observar valores muito próximos aos apresentados no primeiro experimento e novamente, as variações são combinadas com o número de instruções executadas e porcentagem de instruções de acesso a dados.

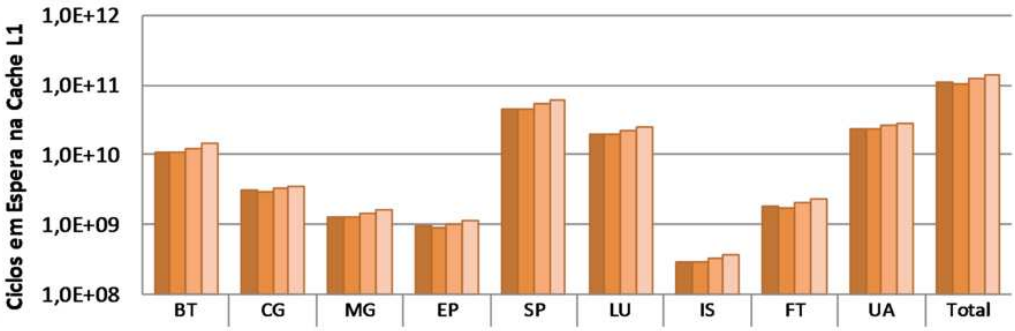
A Figura 4.41 mostra o aumento no número de ciclos perdidos por espera de dados ocasionados por faltas na memória *cache* L1. Podemos ver o crescimento no número de ciclos em espera conforme aumentou-se o número de núcleos compartilhando o mesmo



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	46,48	46,16	40,39	26,14	38,43	27,85	38,51	91,94	33,58	35,79
2Cores/L2	47,00	46,02	40,54	25,84	39,37	27,61	38,13	92,61	33,54	36,13
4Cores/L2	46,00	46,33	40,88	25,80	39,91	27,46	38,53	92,70	33,48	36,21
8Cores/L2	47,35	48,52	40,85	25,75	39,01	27,07	38,91	93,05	33,58	35,95

Figura 4.40: Memória *cache* L1 MPKI do terceiro experimento.

banco de memória *cache* L2. Nota-se que a taxa de aumento da quantidade de ciclos de espera é menor que a taxa apresentada no primeiro experimento, entretanto, os valores absolutos são notoriamente maiores neste terceiro experimento, o que é resultado do sobrecusto da latência de acesso a dados ocasionado pelo aumento na associatividade na memória *cache* L2. Esse sobrecusto por espera de dados durante falta é um dos principais fatores que explicam o maior tempo de execução quando comparado ao experimento base.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	1,1E+10	3,1E+09	1,3E+09	9,7E+08	4,6E+10	2,0E+10	2,9E+08	1,8E+09	2,4E+10	1,1E+11
2Cores/L2	1,1E+10	3,0E+09	1,3E+09	9,1E+08	4,7E+10	2,0E+10	2,8E+08	1,8E+09	2,3E+10	1,1E+11
4Cores/L2	1,2E+10	3,4E+09	1,5E+09	1,0E+09	5,4E+10	2,3E+10	3,3E+08	2,1E+09	2,6E+10	1,2E+11
8Cores/L2	1,4E+10	3,6E+09	1,7E+09	1,2E+09	6,0E+10	2,5E+10	3,8E+08	2,4E+09	2,8E+10	1,4E+11

Figura 4.41: Ciclos de espera de dados na memória *cache* L1 do terceiro experimento.

Notamos, pela Figura 4.42, que a redução na taxa de MPKI na memória *cache* L2, conforme aumentou-se o compartilhamento, foi próxima à obtida no primeiro experimento. Assim, fica claro a desvantagem no desempenho final, conforme aumentou-se a associatividade. Esse aumentou da associatividade levou ao aumento da latência de acesso e não reduziu suficientemente a quantidade de falta de dados. Com isso, podemos ver que as técnicas tradicionais de ganho de desempenho, como a aumento da associatividade, antes só penalizadas pelo aumento na área e consumo de potência, nas tecnologias atuais não funcionam tão bem, considerando a latência extra gerada por essas técnicas.

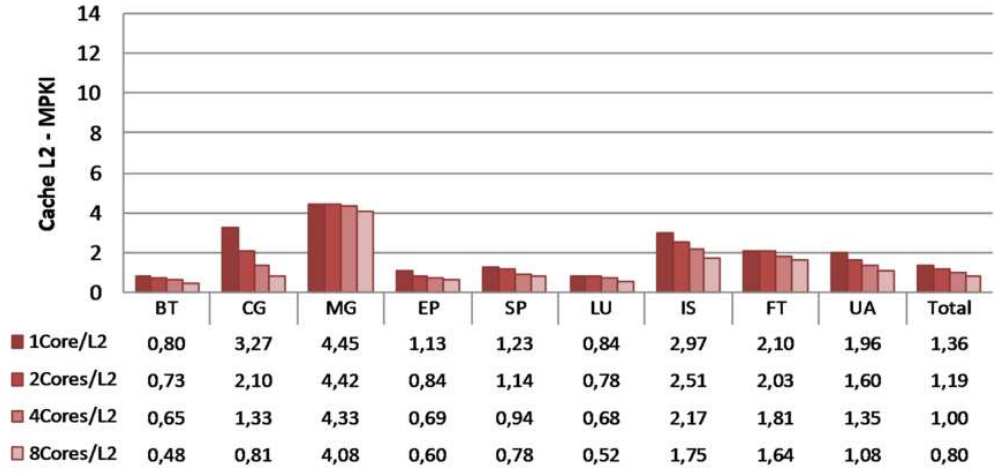


Figura 4.42: Memória *cache* L2 MPKI do terceiro experimento.

4.3.3 Consumo de Energia e Ocupação de Área

Sobre o consumo de energia e potência total do sistema de memória, as organizações 4Cores/L2 e 8Cores/L2 apresentam menores consumos de potência entre as organizações avaliadas neste experimento, como é visto na Figura 4.43, que trás o gráfico de consumo de energia e potência total do sistema de memória avaliado. A redução no consumo de potência é obtida principalmente no sistema de memória *cache* L2.

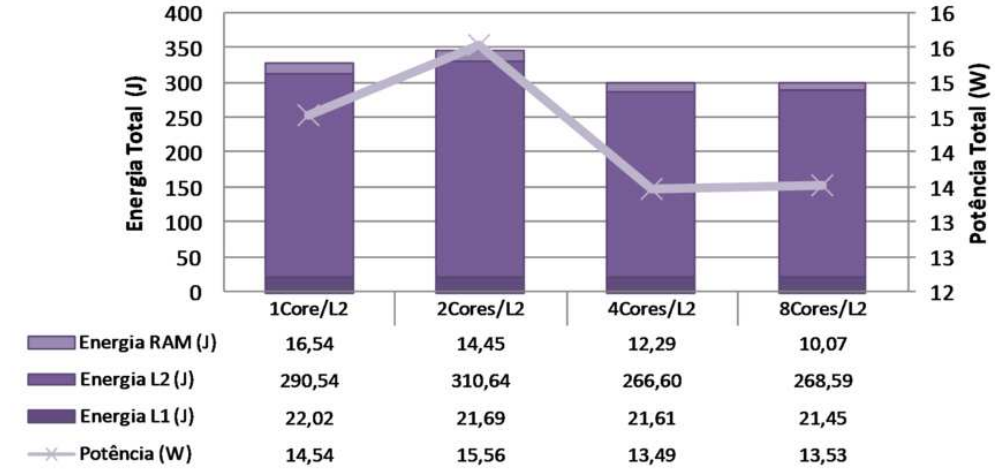


Figura 4.43: Consumo de energia e potência total do sistema de memória do terceiro experimento.

A área ocupada pela memória *cache* L2 também apresentou reduções para as organizações 4Cores/L2 e 8Cores/L2, sendo que, ao utilizar memórias de maior capacidade, a área total ocupada foi sendo reduzida, como apresenta o gráfico da Figura 4.44. A redução na área total ocupada advém do aumento de densidade dos bancos, utilizando um número menor de bancos, mas mantendo a capacidade total.

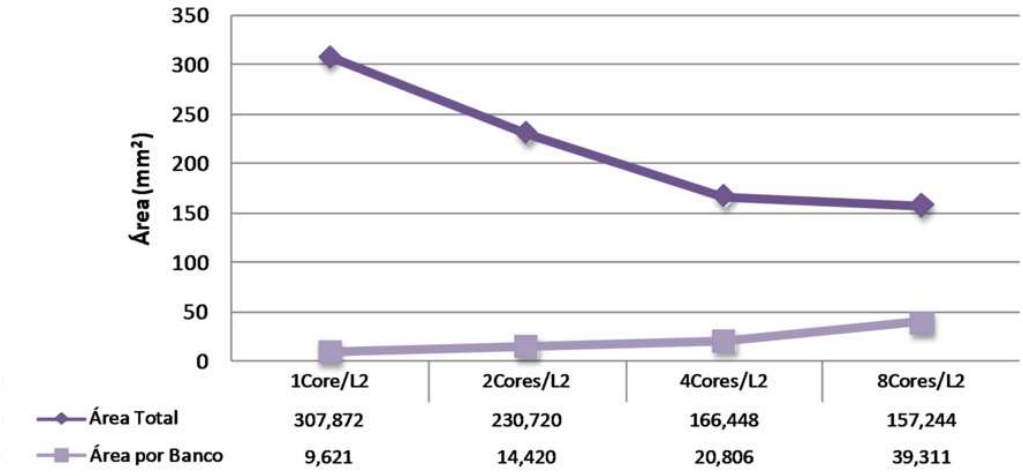


Figura 4.44: Área ocupada pela memória *cache* L2 do terceiro experimento.

4.3.4 Sumário

Podemos concluir com esse experimento que os ganhos obtidos em redução de faltas de dados com o aumento de associatividade não representam grandes ganhos, considerando o aumento na latência da memória *cache* que essa técnica ocasiona. Podemos ver um sumário desse experimento no gráfico da Figura 4.45, que mostra valores comparativos de tempo de execução, consumo de energia e ocupação de área para esse experimento.

Comparando a primeira organização desse terceiro experimento com a primeira organização do experimento base, nota-se claramente todo sobrecusto em termos de desempenho, consumo de energia e ocupação de área que o uso de associatividade ocasiona. Entretanto, conforme aumenta-se o compartilhamento de memória *cache* L2, as perdas de desempenho não são tão grandes (0,99%), obtendo redução no consumo de potência e na ocupação de área pelo sistema de memória.

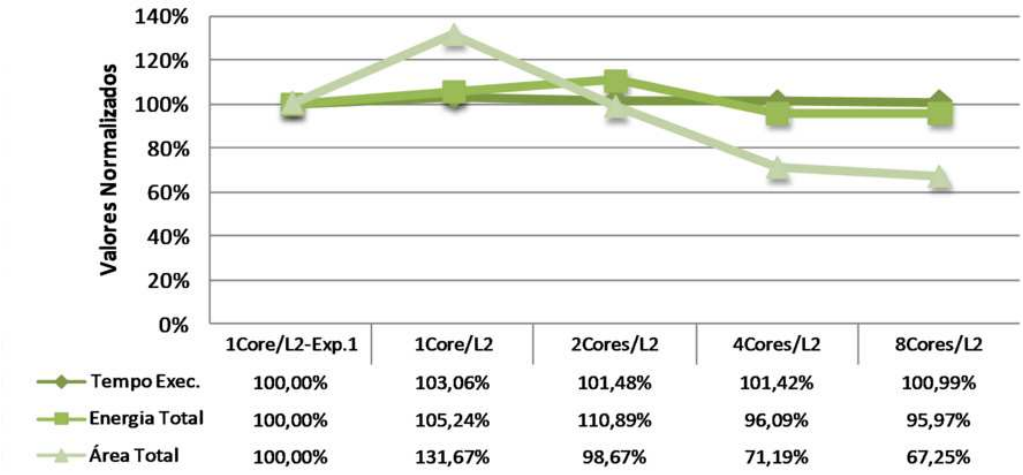


Figura 4.45: Sumário de desempenho, consumo de energia e ocupação de área do terceiro experimento.

4.3.5 Avaliação das Aplicações

As aplicações BT, SP e LU, que fazem o compartilhamento de parte do domínio do problema entre diversos processadores, apresentam comportamentos diferentes neste experimento. A aplicação SP apresenta ganhos com o compartilhamento de memória, considerando o aumento da associatividade deste terceiro experimento. Podemos assim concluir que o aumento da localidade temporal dos dados e redução de conflito de endereços foi a responsável por esse ganho. Para as aplicações BT e LU, o aumento na latência de acesso a dados da memória *cache* L2 resultou em pouco ou nenhum ganho de desempenho.

As aplicações CG e UA, que possuem característica de acesso não-linear aos dados, assim como nos demais experimentos, obtiveram ganhos à medida que o compartilhamento de memória aumentou, e a aplicação CG foi a mais privilégiada com o aumento na associatividade.

A aplicação MG que utiliza acessos lineares e não lineares a dados, mostrou pequenos ganhos de desempenho neste experimento, uma vez que a taxa de faltas na memória *cache* L2 não foi reduzida o suficiente para pagar pelo sobrecusto no tempo de acesso a dados.

Na aplicação EP, o compartilhamento de memória *cache* pôde ser feito sem ocasionar problemas no desempenho. Mas, para isso, o aumento na associatividade teve de ser empregado. Logo, essa estratégia pode ser indicada para sistemas computacionais de uso geral, onde podemos aumentar os ganhos em aplicações paralelas de dados compartilhados sem queda de desempenho em aplicações independentes, executadas em paralelo.

A aplicação IS que compartilha muitos dados entre as diversas *threads*, também apresentou melhora no desempenho conforme aumentamos o compartilhamento da memória *cache* L2 neste experimento. Entretanto, os ganhos não foram proporcionais ao compartilhamento, e isso se deve ao custo de acesso à memória *cache* L2, que sobrepôs os ganhos de redução de faltas de dados.

Para a aplicação FT, o aumento na associatividade, elevando a localidade temporal e reduzindo os conflitos de endereço, levaram à redução nas faltas de dados na memória *cache* L2, mas novamente, o ganho pela redução de faltas não se sobrepôs ao aumento nas latências.

Podemos notar que todas as aplicações obtiveram reduções na taxa faltas de dados na memória *cache* L2 por milhares de instruções executadas, mostrando que o aumento da associatividade é efetivamente válido para a redução de faltas, porém, o sobrecusto em latência leva o sistema a um desempenho pior que o das organizações avaliadas no experimento base, mas a redução de faltas leva a um menor consumo total de potência.

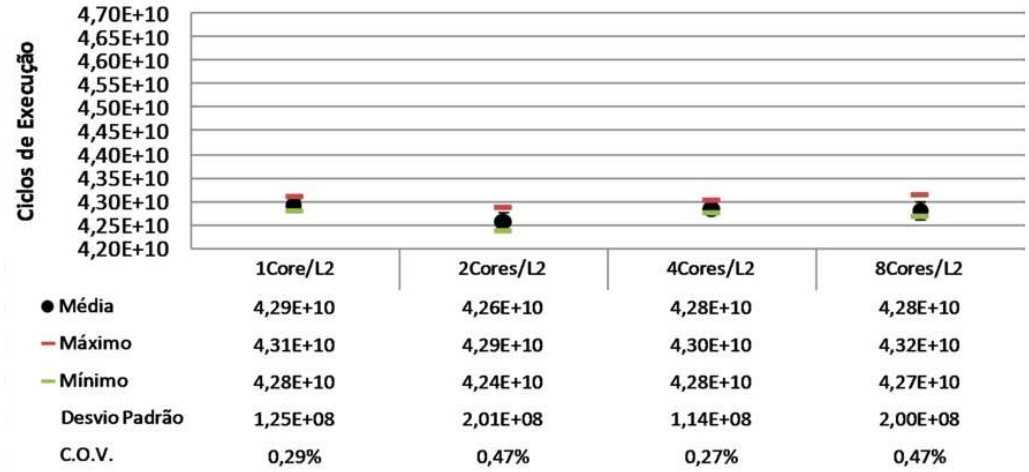
Logo, os custos físicos como, consumo de potência e área, podem ser reduzidos com o aumento no compartilhamento de memória *cache*, porém, nessa tecnologia de integração em que foram modeladas as memórias *cache*, os ganhos de desempenho são diluídos pelas latências de acesso a dados. Pode-se notar também, que novamente as aplicações que possuem acesso a dados não lineares foram as mais privilegiadas com a mudança na memória cache.

4.4 Experimento 4 - Tamanho da Linha da Memória *Cache*

O aumento no tamanho da linha da memória *cache* proposto neste quarto experimento, age na memória *cache* L2 aumentando os ganhos pela localidade espacial dos dados, onde, ao buscar blocos da memória principal, mais dados adjacentes serão buscados ao mesmo tempo. Para isso, o tamanho da linha foi aumentado de 64 bytes para 128 bytes.

Assim como no terceiro experimento, este experimento também só avalia mudanças nos compartilhamentos 1, 2, 4 e 8 núcleos por memória *cache* L2, pelos mesmos motivos, ou seja, como apenas a organização 4Cores/L2 havia apresentado ganhos no primeiro experimento, o objetivo é avaliar a influência do aumento do tamanho da linha no desempenho dos sistemas.

O primeiro resultado desse quarto experimento é apresentado na Figura 4.46, na qual podemos ver as baixas médias de tempo de execução para as organizações avaliadas, onde a organização 2Cores/L2 apresentou melhor desempenho. Sobre as métricas estatísticas, o coeficiente de variação mostra que o desvio padrão representa, no máximo, 0,47% da média obtida. Mesmo com indicação de ganho de desempenho na organização 2Cores/L2, não se pode afirmar, com certeza, que os sistemas diferem no ponto de vista de desempenho, haja visto que os valores das médias colidem entre sí, considerando o desvio padrão das medições.



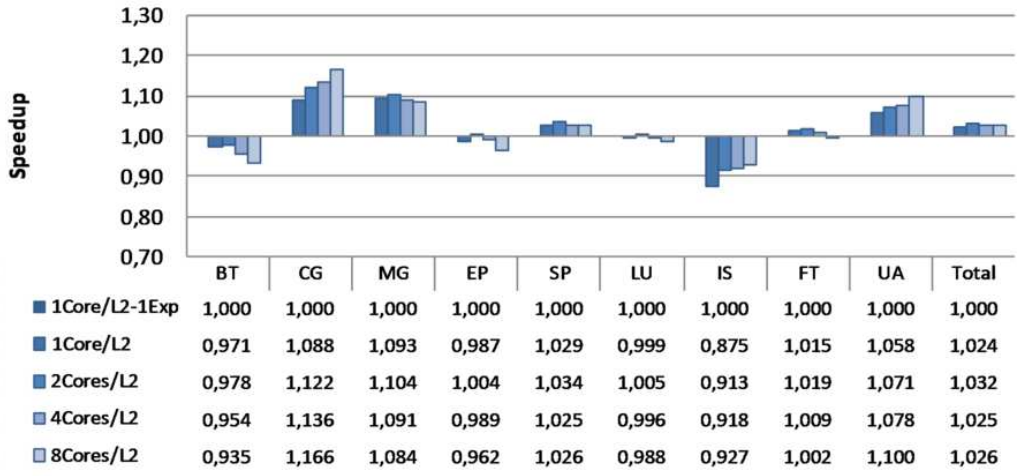
	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2
Média	4,29E+10	4,26E+10	4,28E+10	4,28E+10
Máximo	4,31E+10	4,29E+10	4,30E+10	4,32E+10
Mínimo	4,28E+10	4,24E+10	4,28E+10	4,27E+10
Desvio Padrão	1,25E+08	2,01E+08	1,14E+08	2,00E+08
C.O.V.	0,29%	0,47%	0,27%	0,47%

Figura 4.46: Espalhamento das medidas do quarto experimento.

4.4.1 Processador *Multi-Core*

O gráfico apresentado na Figura 4.47 mostra que na configuração 2Cores/L2, 7 das 9 aplicações obtiveram algum ganho no desempenho, sendo que essa organização foi a que obteve melhor desempenho neste experimento. As aplicações CG, IS e UA foram as maiores beneficiadas, conforme ocorreu aumento do compartilhamento da memória *cache* entre os núcleos de processamento. Mesmo com os ganhos de desempenho na organização 2Cores/L2 de apenas 3,2%, é importante notar que essa organização conseguiu obter bons resultados para a maior parte das aplicações avaliadas, apresentados resultados negativos para as aplicações BT e IS.

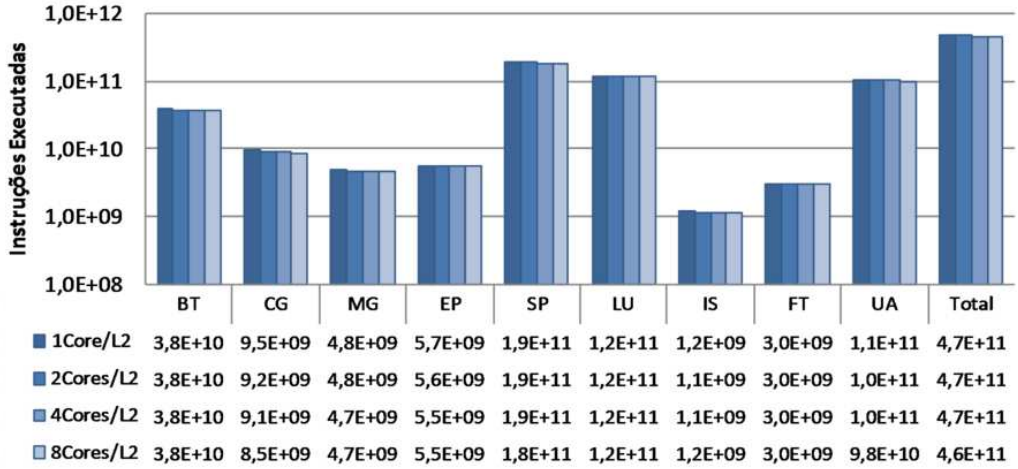
A Figura 4.48 mostra a variação na quantidade de instruções executadas neste quarto experimento. Comparado com o experimento base, houve uma redução na quantidade de instruções executadas, que se deve à redução no tempo total de execução da carga



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2-1Exp	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
1Core/L2	0,971	1,088	1,093	0,987	1,029	0,999	0,875	1,015	1,058	1,024
2Cores/L2	0,978	1,122	1,104	1,004	1,034	1,005	0,913	1,019	1,071	1,032
4Cores/L2	0,954	1,136	1,091	0,989	1,025	0,996	0,918	1,009	1,078	1,025
8Cores/L2	0,935	1,166	1,084	0,962	1,026	0,988	0,927	1,002	1,100	1,026

Figura 4.47: *Speedup* do quarto experimento.

de trabalho, o que permite menos influência do sistema operacional e na quantidade de instruções executadas.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	3,8E+10	9,5E+09	4,8E+09	5,7E+09	1,9E+11	1,2E+11	1,2E+09	3,0E+09	1,1E+11	4,7E+11
2Cores/L2	3,8E+10	9,2E+09	4,8E+09	5,6E+09	1,9E+11	1,2E+11	1,1E+09	3,0E+09	1,0E+11	4,7E+11
4Cores/L2	3,8E+10	9,1E+09	4,7E+09	5,5E+09	1,9E+11	1,2E+11	1,1E+09	3,0E+09	1,0E+11	4,7E+11
8Cores/L2	3,8E+10	8,5E+09	4,7E+09	5,5E+09	1,8E+11	1,2E+11	1,2E+09	3,0E+09	9,8E+10	4,6E+11

Figura 4.48: Instruções executadas do quarto experimento.

4.4.2 Subsistema de Memória

O gráfico contendo as taxas de faltas de dados na memória *cache* L1 (MPKI) é apresentado na Figura 4.49, onde podemos ver, em média, pequenas variações. As taxa obtidas são pouco maiores que as apresentadas no primeiro experimento, o que se explica pela redução na quantidade de instruções executadas.

A Figura 4.50 apresenta o gráfico contendo informações a respeito da quantidade de ciclos perdidos por espera de dados após faltas de dados na memória *cache* L1. Assim como no primeiro e terceiro experimentos, o aumento no tamanho dos blocos de memória *cache*, que ocorre ao aumentar o compartilhamento da memória *cache* L2, foi responsável pelo aumento na latência para acessar os dados na memória *cache* L2. Dessa forma, houve uma escalada na quantidade de ciclos perdidos conforme aumentou-se o compartilhamento de memória *cache*.

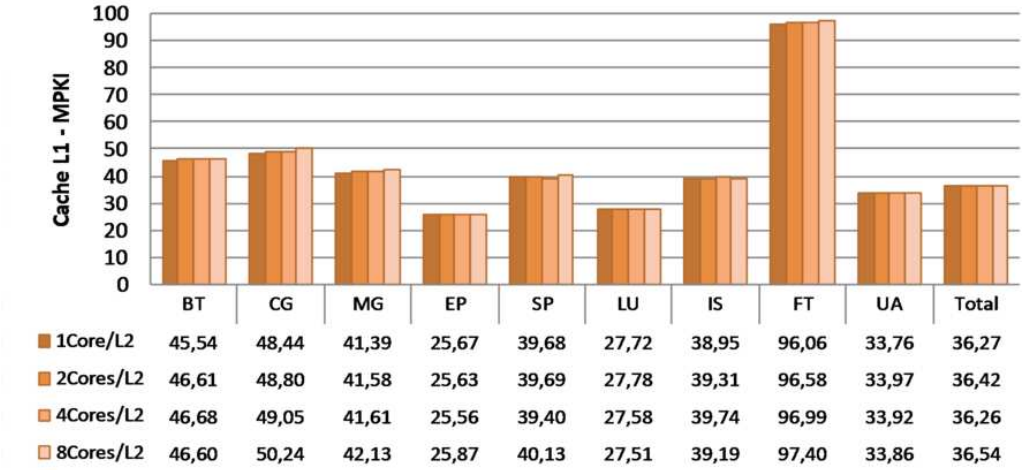


Figura 4.49: Memória *cache* L1 MPKI do quarto experimento.

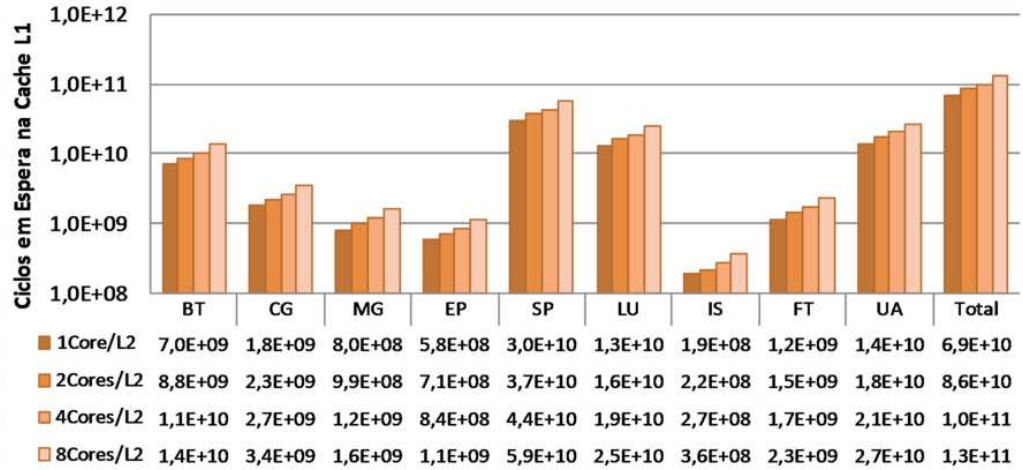
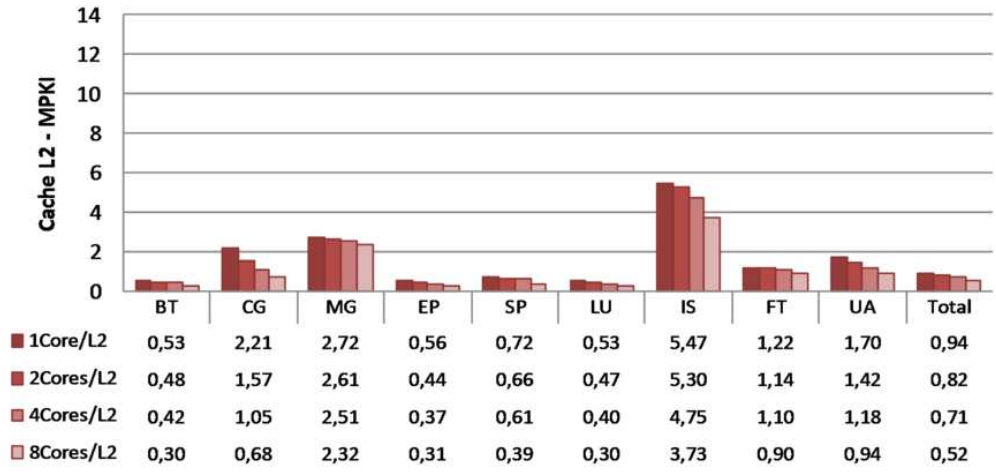


Figura 4.50: Ciclos de espera de dados na memória *cache* L1 do quarto experimento.

A taxa de faltas de dados sobre milhares de instruções executadas é apresentada na Figura 4.51. Podemos notar através desse gráfico, que as taxas de faltas na memória *cache* L2 foram menores que as obtidas no primeiro experimento, ressaltando que isso ocorreu mesmo com a redução no número de instruções executadas, o que tenderia a elevar a taxa MPKI. Tal redução foi verificada para todos os aplicativos da carga de trabalho, sendo que as reduções, em todos casos, partiram da primeira organização (1Core/L2) até a última avaliada (8Cores/L2). Dessa forma, os ganhos com essa técnica de aumento do tamanho da linha de dados ficam mais evidentes.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	0,53	2,21	2,72	0,56	0,72	0,53	5,47	1,22	1,70	0,94
2Cores/L2	0,48	1,57	2,61	0,44	0,66	0,47	5,30	1,14	1,42	0,82
4Cores/L2	0,42	1,05	2,51	0,37	0,61	0,40	4,75	1,10	1,18	0,71
8Cores/L2	0,30	0,68	2,32	0,31	0,39	0,30	3,73	0,90	0,94	0,52

Figura 4.51: Memória *cache* L2 MPKI do quarto experimento.

4.4.3 Consumo de Energia e Ocupação de Área

Com relação ao consumo total de energia e potência, o gráfico da Figura 4.52 mostra o menor consumo na organização 4Cores/L2, sendo que esse baixo consumo é causado, principalmente, por redução no consumo de memória *cache* L2. Esse resultado de consumo total é a combinação dos resultados de consumo dinâmico e estático de energia e potência.

A estimativa de área que seria ocupada por memória *cache* L2 iguais as que foram avaliadas neste quarto experimento está apresentada na Figura 4.53. Nota-se que as duas menores ocupações de área são das organizações 4Cores/L2 e 8Cores/L2, nessa ordem. Esse resultado segue apresentando melhoria no consumo de potência para a organização 4Cores/L2, ou seja, essa organização apresenta boa organização interna dos *arrays* de dados e demais componentes, o que ocasiona redução na área ocupada e consumo de energia.

4.4.4 Sumário

O gráfico presente na Figura 4.54 mostra um sumário de desempenho, consumo de energia e ocupação de área para este quarto experimento. Comparando com a organização 1Core/L2 do primeiro experimento, podemos ver que somente a partir da organização 4Cores/L2 foi possível obter redução na área ocupada pela memória *cache* L2. No entanto, nenhum caso evitou o aumento no consumo de potência que ocorreu na organização 2Cores/L2, mesmo executando a carga de trabalho em menor tempo, em média 3,07% mais rápido que a primeira organização.

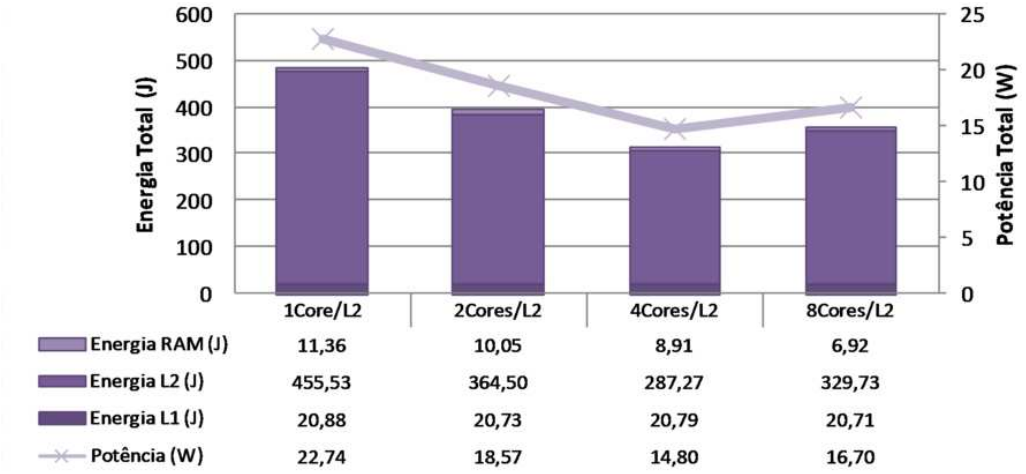


Figura 4.52: Consumo de energia e potência total do sistema de memória do quarto experimento.

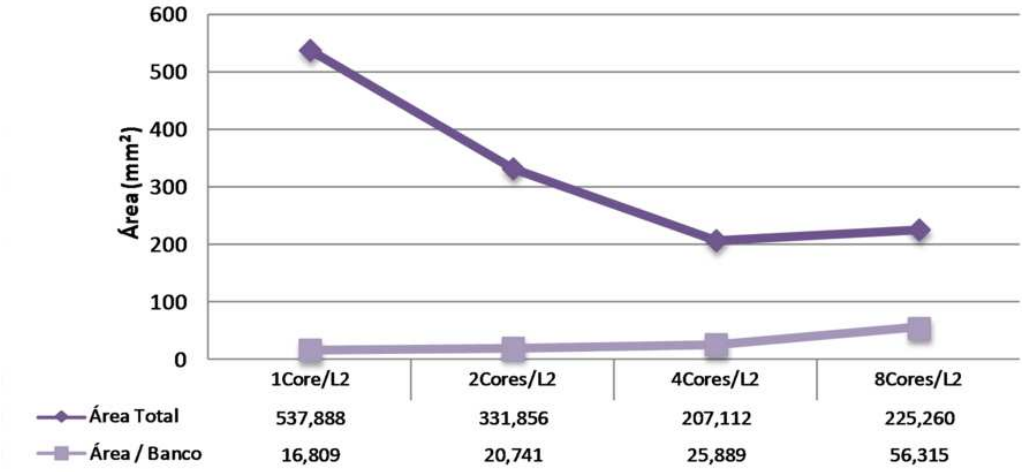
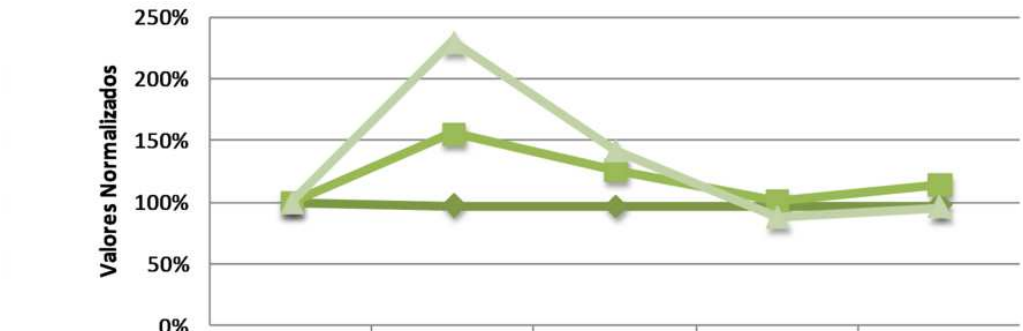


Figura 4.53: Área ocupada pela memória *cache* L2 do quarto experimento.

Podemos concluir por este experimento que existe a necessidade de projetos que considerem questões de organização e arquiteturas acopladas ao projeto físico de memória *cache*, para que então o sistema final ganhe em desempenho, com possibilidades de redução na área e potência.



	1Core/L2-Exp.1	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2
Tempo Exec.	100,00%	97,68%	96,93%	97,53%	97,45%
Energia Total	100,00%	155,98%	126,40%	101,36%	114,28%
Área Total	100,00%	230,04%	141,93%	88,58%	96,34%

Figura 4.54: Sumário de desempenho, consumo de energia e ocupação de área do quarto experimento.

4.4.5 Avaliação das Aplicações

Para as aplicações BT, SP e LU, houve ganhos de desempenho apenas na organização 2Cores/L2, mesmo mantendo a redução de faltas de dados em todas organizações de memória *cache* L2. O comportamento de melhora deve-se principalmente ao aumento na localidade espacial, reduzindo assim o número de faltas de dados em acessos lineares.

As aplicações CG e UA, apresentaram ganhos de desempenho para todas as organizações avaliadas, entretanto, as taxas de faltas de dados não foram reduzidas o suficiente para que o ganho de desempenho chegasse a 10%, como em outros experimentos.

A aplicação IS foi uma das que apresentou maior desempenho neste experimento com o compartilhamento de memória *cache*, uma vez que a localidade espacial dos dados foi beneficiada pelo aumento na linha de dados.

A aplicação EP que representa uma aplicação de paralelismo explícito, apresentou ganho nas organizações 2Cores/L2 e 4Cores/L2, mostrando assim, a importância do aumento não só da associatividade como do tamanho da linha de dados para aplicações independentes.

As demais aplicações, MG e FT, apresentaram reduções nas faltas de dados da memória *cache* L2 conforme aumentamos o compartilhamento de memória *cache*. Entretanto, o comportamento final não representa ganho de desempenho em todas organizações, devido ao aumento na latência de acesso a dados.

Mesmo com os pequenos ganhos de *speedup* deste experimento, comparando com experimento base fica claro o bom desempenho das aplicações executando com memórias *cache* de linhas maiores.

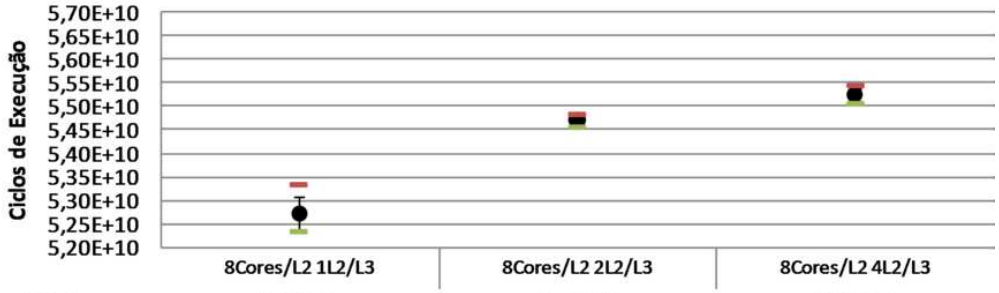
4.5 Experimento 5 - Níveis na Hierarquia de Memória *Cache*

Este quinto experimento tem como objetivo avaliar a influência dos níveis na hierarquia de memória *cache*. Por isso, um terceiro nível foi adicionado para que pudessemos verificar os beneficios e custos adicionais dessa técnica.

Uma vez que diferentes organizações da memória *cache* L3 poderiam ser propostas, baseadas nas organizações do primeiro experimento, neste quinto experimento foi escolhida a organização 8Cores/L2 do experimento base para receber o terceiro nível na hierarquia. Essa organização foi escolhida pois foi a primeira após aquela que obteve ganho de desempenho no experimento base.

Aumentando em um nível a hierarquia de memória *cache*, esperamos reduzir o número de acessos à memória principal, evitando assim as altas latências. Porém, esse aumento deverá causar aumento na área e consumo de potência.

O primeiro resultado é a respeito do espalhamento das medidas de tempo. A Figura 4.55 apresenta os valores médios de tempo de execução nas diversas organizações avaliadas (8Cores/L2 com 1L2/L3, 2L2/L3 e 4L2/L3). Além dos valores médios, são apresentados os valores máximos, mínimos, desvio padrão e o coeficiente de variação, ressaltando que a escala dessa figura é diferente das demais figuras de espalhamento dos experimentos anteriores. Podemos ver as baixas taxas de variação, representando no máximo 0,69% da média. Nesse experimento, as medidas se apresentam bem definidas, sem nenhuma interseção, e por isso, podemos afirmar que os sistemas não são equivalentes.



	8Cores/L2 1L2/L3	8Cores/L2 2L2/L3	8Cores/L2 4L2/L3
Média	5,27E+10	5,47E+10	5,53E+10
Máximo	5,33E+10	5,48E+10	5,54E+10
Mínimo	5,23E+10	5,46E+10	5,51E+10
Desvio Padrão	3,65E+08	1,10E+08	1,31E+08
C.O.V.	0,69%	0,20%	0,24%

Figura 4.55: Espalhamento das medidas do quinto experimento.

4.5.1 Processador *Multi-Core*

O gráfico da Figura 4.56 mostra o *speedup* deste quinto experimento comparado à primeira organização do primeiro experimento. Ao aumentar o compartilhamento de memória *cache* L3, o desempenho da organização final foi pior, onde a queda de desempenho na organização de total compartilhamento da memória *cache* L3 (4L2/L3) foi próxima a 5%. Mesmo assim, podemos ver que apenas as aplicações MG e UA apresentaram algum ganho de desempenho com o compartilhamento total comparado com a primeira organização desse experimento sem nenhum compartilhamento, reforçando a pouca eficiência dessa organização proposta.

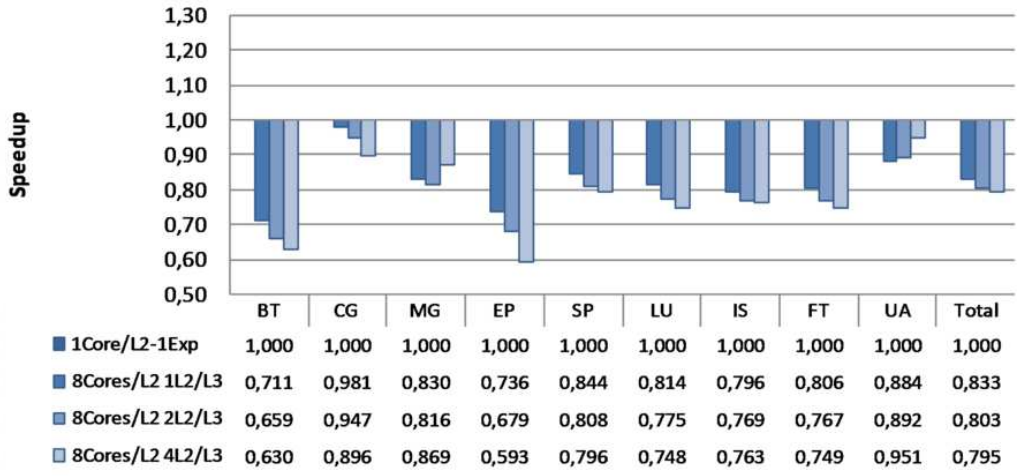


Figura 4.56: *Speedup* do quinto experimento.

A variação na quantidade de instruções executadas neste experimento é apresentada na Figura 4.57. Comparado aos experimentos anteriores, a quantidade de instruções foi maior, sendo essa variação causada pelo aumento no tempo de execução da carga de trabalho. Assim o sistema operacional executou mais instruções.

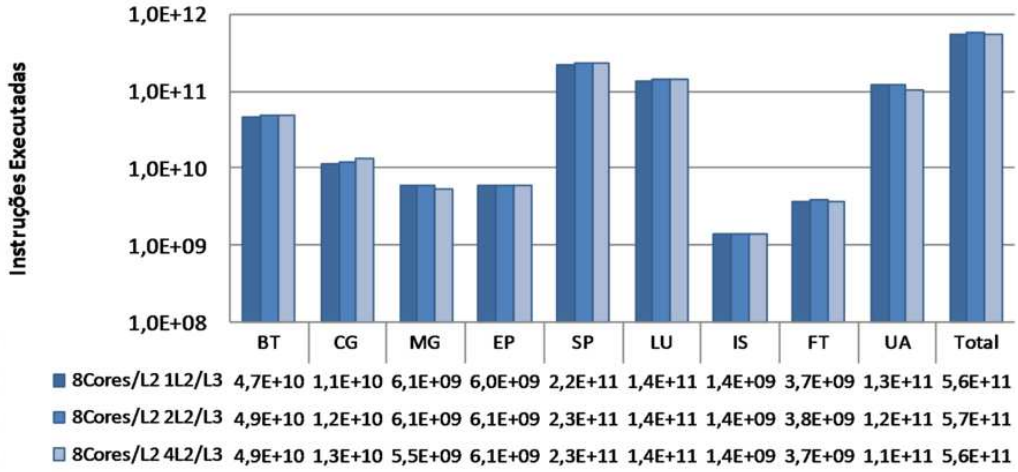


Figura 4.57: Instruções executadas do quinto experimento.

4.5.2 Subsistema de Memória

O gráfico da Figura 4.58 mostra a taxa de faltas de dados na memória *cache* L1 em relação à quantidade de instruções executadas no sistema, os valores estão divididos em aplicações e organizações de memória avaliadas. Podemos notar que as taxas MPKI são inferiores que as do experimento base e isso se deve à variação na quantidade de instruções executadas, sendo que, no total, os valores permanecem em mesmo patamar.

As quantidades de ciclos de espera ocasionados por faltas de dados na memória *cache* L1 são apresentadas na Figura 4.59. Nessa figura, a quantidade de ciclos perdidos por espera de dados não sofre grandes variações entre as organizações de memória, uma vez que a latência da memória *cache* L2 permanece inalterada.

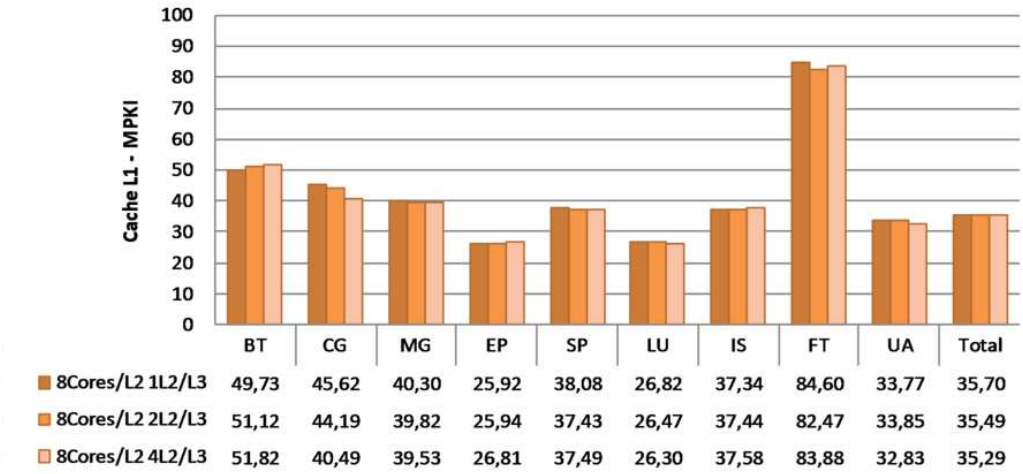


Figura 4.58: Memória *cache* L1 MPKI do quinto experimento.

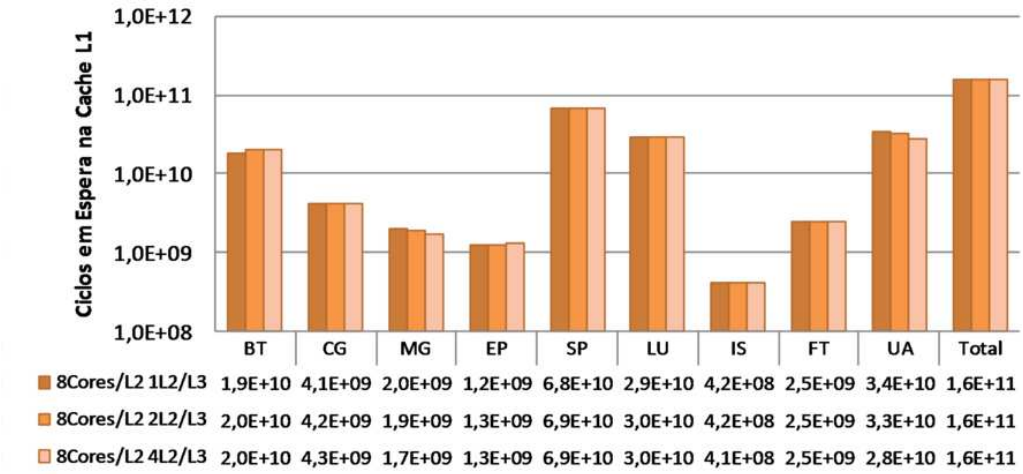


Figura 4.59: Ciclos de espera de dados na memória *cache* L1 do quinto experimento.

A variação na taxa de faltas da memória *cache* L2 está presente na Figura 4.60. Podemos ver que existe certa variação entre diferentes organizações executando a mesma aplicação, como no caso das aplicações CG, MG, EP e FT. Embora seja difícil afirmar com certeza o motivo dessa variação, podemos supor que a causa pode ser a mudança no padrão de alocação de dados gerada conforme mudamos o nível inferior da hierarquia de memória. Observando os valores de MPKI da memória *cache* L2 e o gráfico de *speedup*, não se pode afirmar com certeza que essa variação na taxa de faltas de dados se reflete no desempenho do sistema.

Outro ponto interessante apresentado pelos resultados é que a taxa MPKI apresentada neste experimento é maior que a do experimento base, utilizando a mesma organização 8Cores/L2. Assim, nota-se a fragilidade do sistema, no sentido de que pequenas variações podem levar a diversas variações no comportamento do sistema de memória *cache*.

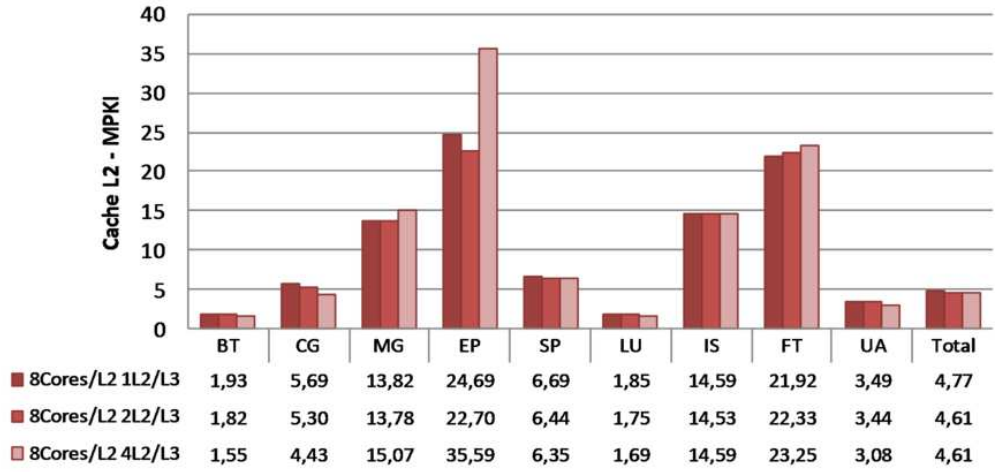


Figura 4.60: Memória *cache* L2 MPKI do quinto experimento.

A Figura 4.61 apresenta a quantidade de ciclos perdidos por espera de dados durante faltas de dados da memória *cache* L2, para as diversas organizações avaliadas. Onde, ao modificar o compartilhamento da memória *cache* L3, os blocos de memória *cache* ficaram maiores, e por isso, mais lentos, sendo esperado o acréscimo na quantidade de ciclos perdidos.

Após averiguar o comportamento das memórias *cache* L1 e L2, foi verificada a taxa de faltas de dados por milhares de instruções na memória *cache* L3, que esta presente na Figura 4.62. Conseguimos verificar nesse gráfico, a redução na taxa de faltas à medida que as memórias *cache* L3 foram compartilhadas por mais núcleos de processamento e memórias *cache* L2.

A taxa de faltas de dados na memória *cache* L3, comparada com a memória *cache* L2 do primeiro experimento é bastante próxima. Assim, pode-se concluir que esse aumento no nível de hierarquias de memória levou a um sobrecusto real no desempenho, já que as taxas da L1 e L3 coincidem com as da L1 e L2 do primeiro experimento, respectivamente. Logo, todas as faltas ocorridas na memória *cache* L2 deste quinto experimento pioraram o desempenho do sistema.

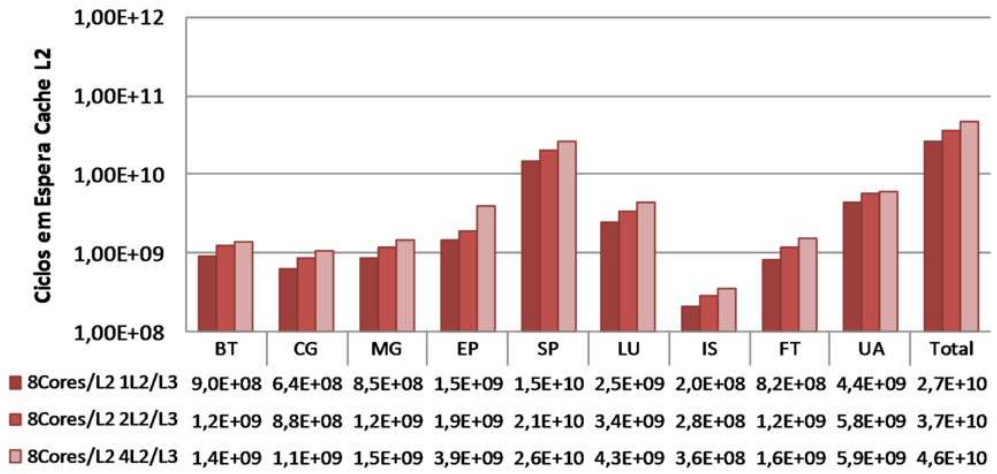


Figura 4.61: Ciclos de espera de dados na memória *cache* L2 do quinto experimento.

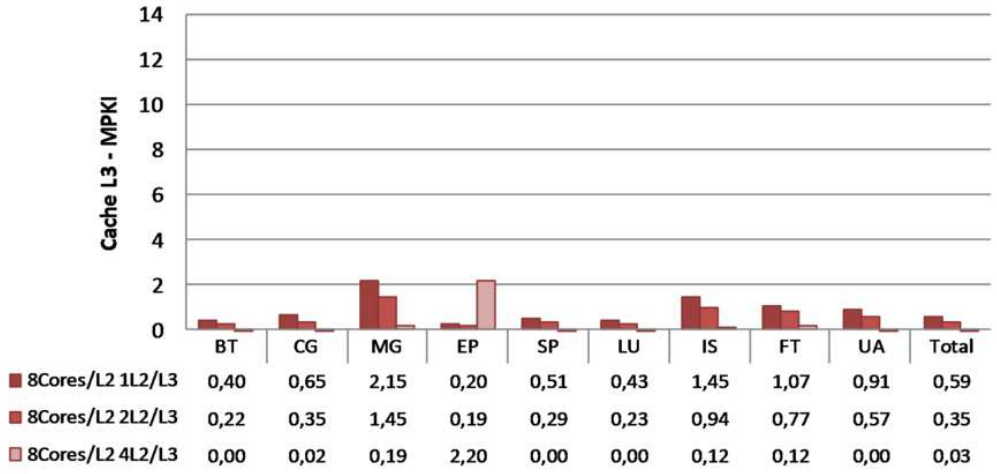


Figura 4.62: Memória *cache* L3 MPKI do quinto experimento.

4.5.3 Consumo de Energia e Ocupação de Área

O consumo de energia e potência total do sistema de memória para o quinto experimento está ilustrado na Figura 4.63, onde pode-se notar que comparando com o primeiro experimento, houve em média 10 W a mais no consumo de potência total.

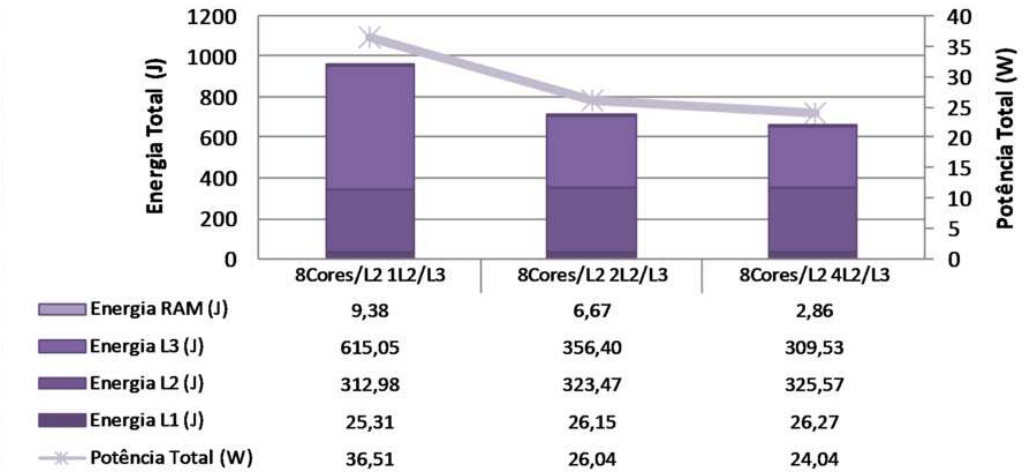


Figura 4.63: Consumo de energia e potência total do sistema de memória do quinto experimento.

O gráfico da Figura 4.64 mostra a ocupação de área da memória *cache* L3 para este quinto experimento. Nota-se um aumento na área total ocupada conforme reduziu-se a quantidade de bancos de memória *cache* L3, mantendo o tamanho lógico total como aconteceu nos demais experimentos. É importante ressaltar que este gráfico apresenta apenas a área da memória *cache* L3.

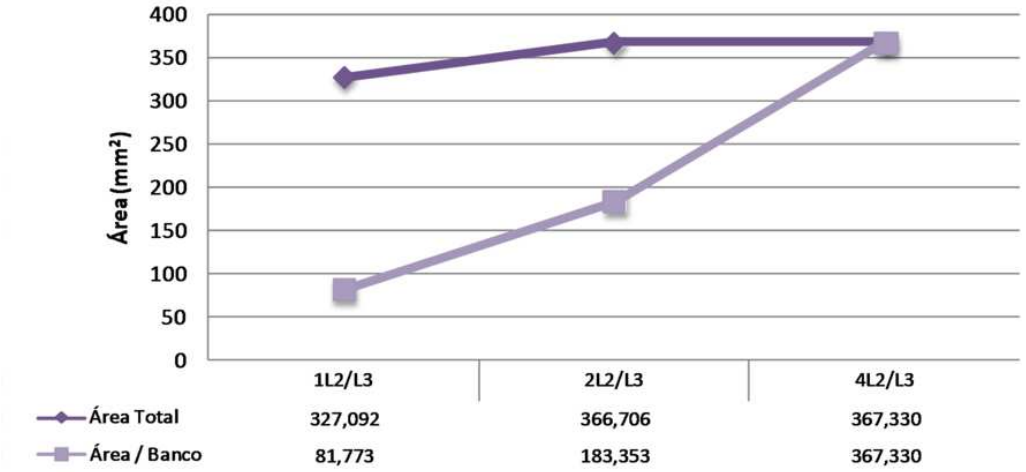


Figura 4.64: Área ocupada pela memória *cache* L3 do quinto experimento.

4.5.4 Sumário

Esse experimento avaliou a inserção de um nível L3 na hierarquia de memória *cache*, o qual obteve resultados de sobrecusto muito elevado, como podemos ver no gráfico de

sumário ilustrado na Figura 4.65. Com relação aos três itens apresentados nesse gráfico, fica claro que a inserção de novo nível na hierarquia de memória *cache* trouxe apenas sobrecusto sem mostrar nenhum beneficio prático. Podemos ver um acréscimo mínimo de aproximadamente 20% no tempo de execução da carga de trabalho, acréscimo de 112% no consumo de energia e 39% na área do sistema comparando as memórias *cache* L2 da primeira organização do experimento base com a área ocupada por apenas a memória *cache* L3 desse experimento.

Notamos que, mesmo com área disponível, a escolha por acrescer a hierarquia de memória pode não trazer benefícios, sendo então, de suma importância a avaliação completa de custos adicionais que essa escolha pode trazer ao desempenho do sistema.

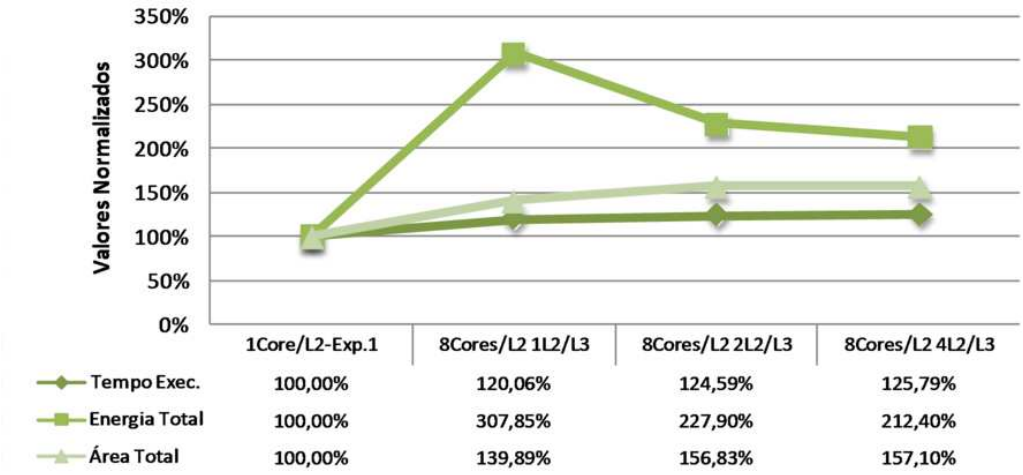


Figura 4.65: Sumário de desempenho, consumo de energia e ocupação de área do quinto experimento.

4.5.5 Avaliação das Aplicações

Neste experimento podemos ver que existe um sobrecusto no desempenho relacionado ao nível adicional na hierarquia de memória. Com base nos resultados apresentados, nota-se que entre os diversos experimentos, este foi o que apresentou piores resultados. Entretanto, nessa subseção as aplicações serão avaliadas a respeito da influência do compartilhamento de memória *cache* em um sistema com três níveis na hierarquia de memória.

As aplicações BT, CG, SP, LU, IS e FT, sofreram queda de desempenho a medida que o terceiro nível da hierarquia de memória *cache* foi compartilhado entre mais blocos de memória *cache* L2. Analisando os gráficos de tempo de espera da memória *cache* L2, podemos ver o aumento de espera à medida que as memórias *cache* L3 foram sendo compartilhadas. Logo, mesmo que as faltas de dados da memória *cache* L3 foram sendo reduzidas para todas essas aplicações, essa redução não foi suficiente para minimizar o custo extra na latência de acesso a dados.

Para as aplicações UA e MG, o compartilhamento de memória *cache* L3 foi benéfico. Esse comportamento se explica pelas fortes reduções na taxa MPKI da memória *cache* L3. Assim, na primeira organização, a taxa de faltas era muito alta, porém, conforme o compartilhamento foi aumentado, as faltas foram sendo reduzidas, minimizando o custo extra de acesso a dados.

A aplicação EP, foi a que mostrou comportamentos mais anômalos neste quinto experimento, sendo que, essa foi a que obteve piores desempenhos conforme aumentou-se o compartilhamento da memória *cache* L3. Analisando a taxa MPKI para memória *cache* L2, é possível ver o comportamento estranho dessa aplicação, porém, esse comportamento só aconteceu devido a variação na quantidade de instruções executadas. Podemos ver pela figura de espera de dados pela memória *cache* L2, que é linear, que o número total de faltas de dados se manteve próximo em todos os testes dessa aplicação EP. No entanto, os resultados sobre MPKI na memória *cache* L3, deixam claro o custo extra que essa memória *cache* adicionou no desempenho final. Logo, podemos afirmar que a adoção de um nível extra na hierarquia de memória deve ser feita com atenção.

4.6 Experimento 6 - Contenção e Limitações Físicas da Memória *Cache*

Este último experimento visa avaliar a influência da contenção gerada pelo número limitado de portas da memória *cache* L2. Assim, como os experimentos anteriores apresentaram resultados sem as contenções geradas por um número limitado de portas físicas, esse sexto experimento modela analiticamente as contenções geradas pelas portas de acesso a dados, utilizando como entrada para as modelagens os dados obtidos nos experimentos anteriores.

Uma vez que a modelagem apresentada nesse experimento é analítica, os resultados obtidos são estimativas e indicam o comportamento geral do sistema. Por esse motivo, não serão analisadas as aplicações separadamente, mas sim, o comportamento geral do sistema.

A nomenclatura adotada para este experimento adiciona a informação da quantidade de portas (e.g. 1Core/L2-1P, 1 *core* por memória *cache* L2, com uma porta de acesso a dados). Além disso, foram estimadas as contenções apenas para as organizações 1Core/L2, 2Cores/L2 e 4Cores/L2, combinando com 1, 2 e 4 portas de acesso a dados. As fórmulas utilizadas para as estimativas são as apresentadas na seção 3.4.

Os resultados desse experimento serão apresentados em subseções, e cada subseção diz respeito ao experimento que gerou os dados, apresentando resultados sobre tempo de execução, consumo de energia e potência e ocupação de área.

4.6.1 Estimativa de Contenção para o Experimento 1

O primeiro resultado sobre estimativas de contenção com base no primeiro experimento é apresentado na Figura 4.66, que apresenta o gráfico com valores de ciclos de execução, ciclos perdidos por espera de dados durante faltas na memória *cache* L1, *speedup* e porcentagem representativa dos ciclos perdidos por faltas na memória *cache* L1 com relação ao tempo total de execução do experimento.

Com relação à restrição do número de portas, podemos ver que a organização com melhor desempenho foi a 1Core/L2-1P, sendo essa a que apresentou menor quantidade de ciclos perdidos durante esperas da L1 e além disso, podemos ver que a esses ciclos perdidos representaram pouco mais de 15% do tempo de execução da carga de trabalho. Nas demais organizações, podemos ver que a contenção gerada pela quantidade de portas reduzida teve um custo maior que as latências de acesso para memórias com maior quantidade de portas. A influência da espera nos casos com apenas 1 porta também foi maior que nos demais casos para as organizações 2Cores/L2 e 4Cores/L2.

O gráfico apresentado na Figura 4.67, mostra valores estimados para consumo de ener-

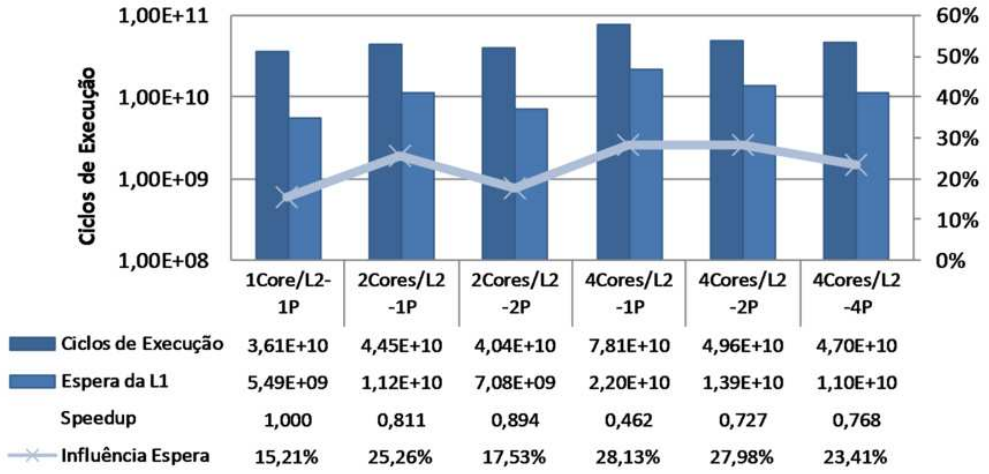


Figura 4.66: Estimativas de execução do primeiro experimento.

gia e potência total do sistema de memória com base nas estimativas de contenção para o primeiro experimento. Pode-se observar que a organização 1Core/L2-1P, 4cores/L2-2P e 4Cores/L2-1P foram as que apresentaram menor consumo estimado de energia total. Entretanto, podemos notar que para o caso 4cores/L2-4P (sistema com 32 portas ao total), o custo extra no consumo de energia gerado à medida que aumentou-se a quantidade de portas, faz com que essa alternativa seja descartada. O mesmo acontece para a memória *cache* 2Cores/L2-2P (sistema com 32 portas ao total). Logo, notamos que para manter um baixo consumo, deve-se manter uma memória *cache* por núcleo, ou então, aumentar o compartilhamento de memória, reduzindo a quantidade de portas total do sistema.

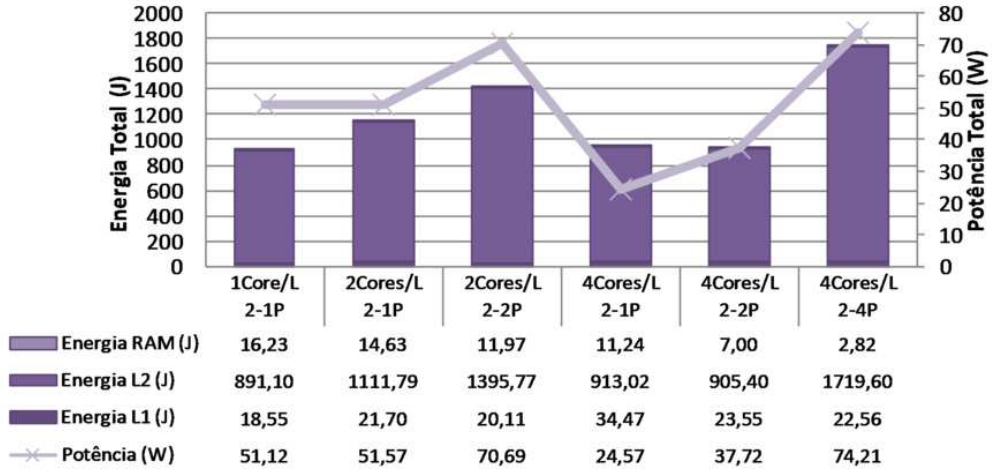


Figura 4.67: Consumo de energia e potência total do sistema de memória do primeiro experimento.

A ocupação de área pela memória *cache* L2 para as organizações e número de portas avaliadas com base no primeiro experimento é apresentada na Figura 4.68. Podemos notar que apenas as organizações 1Core/L2-1P, 2Cores/L2-1P e 4Cores/L2-1P mantém uma ocupação razoável de área da memória *cache* L2. Para os demais casos, existe um acréscimo de mais de 3 vezes, ocasionado pelo aumento na quantidade de portas.

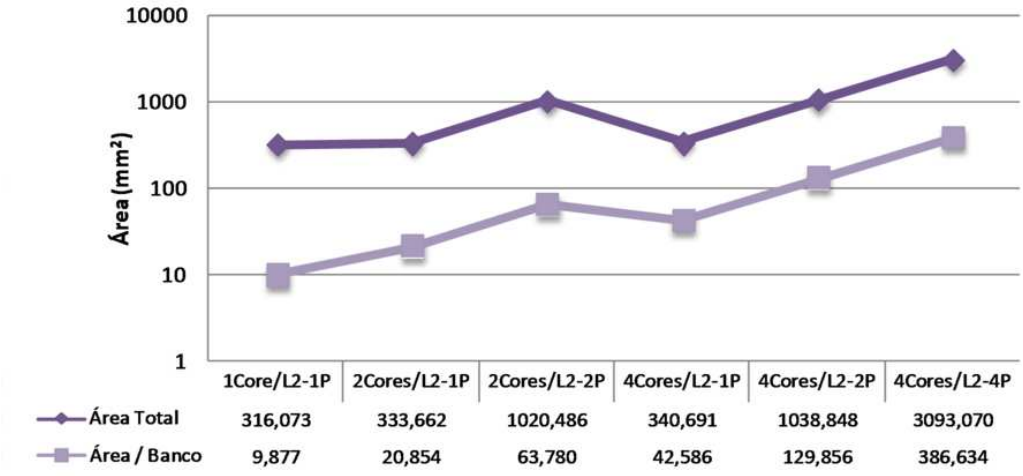


Figura 4.68: Área ocupada pela memória *cache* L2 do primeiro experimento.

Com essas estimativas baseadas no primeiro experimento, podemos notar que quanto ao desempenho, as organizações de melhores resultados foram 1Core/L2, 2Cores/L2-2P e 2Cores/L2-1P. Em relação ao consumo de energia, as melhores organizações foram 1Core/L2-1P, 4Cores/L2-2P e 4Cores/L2-1P. E por fim, as de menor ocupação de área foram 1Core/L2-1P, 2Cores/L2-1P e 4Cores/L2-1P. Assim, combinando os resultados, podemos eleger apenas a organização 1Core/L2-1P como sendo viável para implementação em um sistema computacional.

4.6.2 Estimativa de Contenção para o Experimento 2

Esta subseção apresenta os resultados de avaliação de contenções com base no segundo experimento. Os resultados estão divididos em três partes, 1MB, 2MB e 4MB por banco de memória *cache*.

4.6.2.1 1 MB por Memória Cache *L2*

O gráfico ilustrado na Figura 4.69 apresenta valores estimados com base no segundo experimento com 1 MB por banco de memória *cache* L2. Apresenta estimativas para tempo de execução dado em ciclos de relógio, ciclos perdidos por espera de dados durante faltas na memória *cache* L1, *speedup* com base na primeira organização do primeiro experimento e porcentagem de influência da espera da L1 em relação aos ciclos totais de execução. Observamos, a partir deste gráfico, que nenhuma organização diferente da primeira apresentada com base no primeiro experimento alcançou melhor desempenho. As organizações 2Cores/L2-2P e 2Cores/L2-1P são as que apresentaram segundo e terceiro melhor desempenho.

Com esses resultados baseados no segundo experimento com memória de tamanho fixo, podemos notar que houve aumento no número total de ciclos perdidos durante faltas de dados na memória *cache* L1 conforme aumentou-se o compartilhamento de memória. Neste contexto, podemos notar que o aumento da contenção (4Cores/L2-1P) trouxe pior desempenho que o aumento nas latências combinado com redução nas contenções (4Cores/L2-4P).

A Figura 4.70 apresenta os resultados de consumo de energia e potência total do sistema modelado com contenções no número de portas da memória *cache* L2. O comporta-

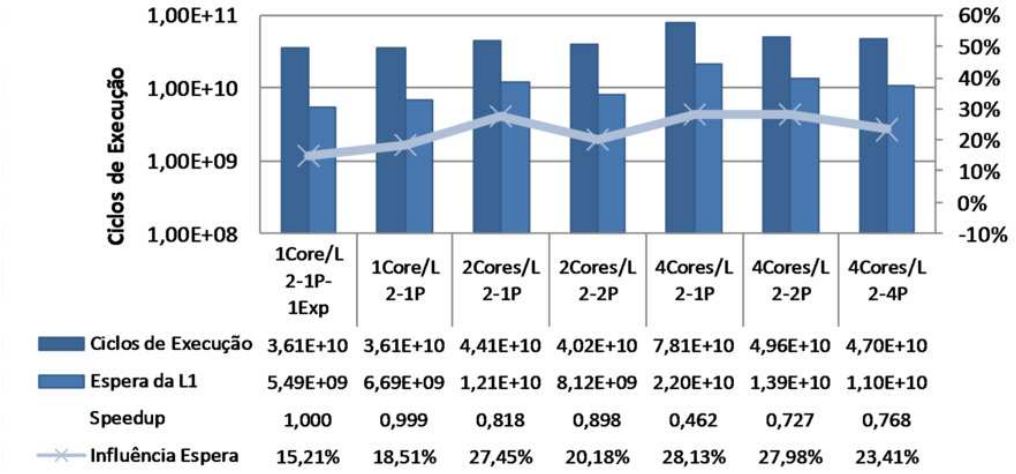


Figura 4.69: Estimativas de execução do segundo experimento (1MB).

mento apresentado nesse gráfico depende de dois fatores: o tamanho total de memória do sistema e a quantidade de portas por memória *cache* L2. Assim, notamos que os menores consumos de energia foram obtidos pelas organizações 4Cores/L2-1P, 2Cores/L2-1P e 4Cores/L2-2P, mostrando que para combater o aumento na energia consumida devido à quantidade de portas do sistema pode-se reduzir a quantidade total de memória *cache* do sistema.

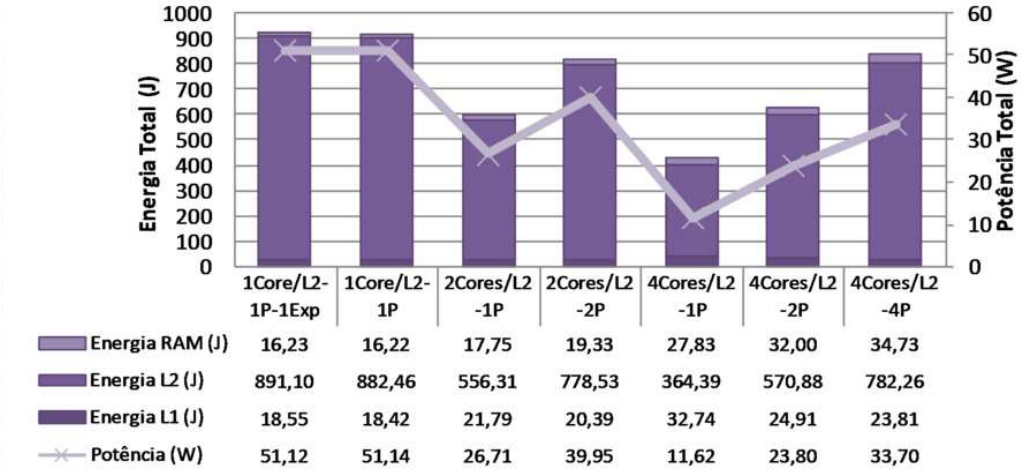


Figura 4.70: Consumo de energia e potência total do sistema de memória do segundo experimento (1MB).

Apresentado na Figura 4.71, o gráfico de ocupação de área mostra as estimativas de área ocupada pela memória *cache* L2 nas diversas organizações, com base no segundo experimento com 1 MB por banco de memória *cache*. Assim como no resultado de potência, as organizações de menor ocupação de área foram as 4Cores/L2-1P, 2Cores/L2-1P e 4Cores/L2-2P, ressaltando o impacto físico da quantidade de portas do sistema. Podemos ver que compensa reduzir a quantidade de compartilhamento de memória de 4Cores/L2-1P para 2Cores/L2-1P e ter um sistema com o dobro de memória *cache*.

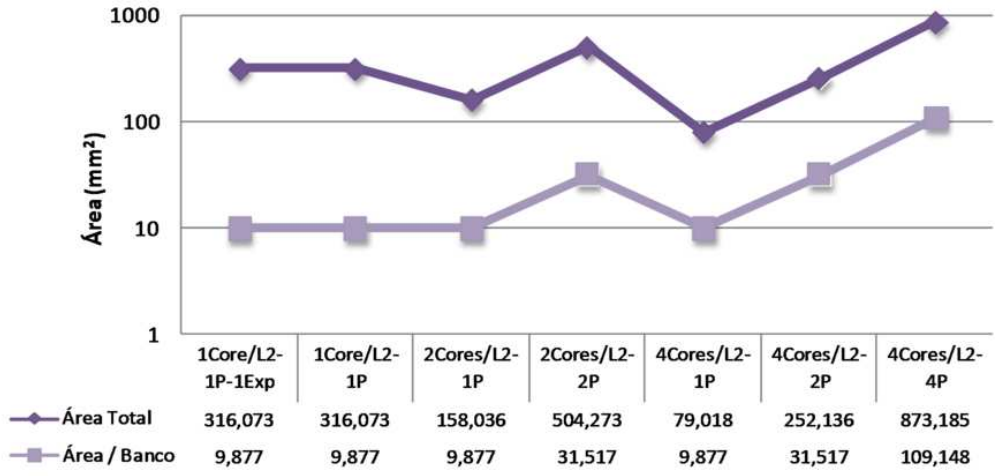


Figura 4.71: Área ocupada pela memória *cache* L2 do segundo experimento (1MB).

4.6.2.2 2 MB por Memória Cache *L2*

A Figura 4.72 apresenta o gráfico com valores de ciclos de execução para memórias *cache* L2 com 2 MB por banco, trazendo valores de tempo de execução, ciclos de espera durante faltas na memória *cache* L1, *speedup* e influência em relação ao tempo de execução dos ciclos perdidos durante as faltas na memória *cache* L1. Através desse gráfico, podemos notar o bom desempenho apresentado pela organização 1Core/L2-1P com 2 MB por banco de memória *cache*, em consequência do equilíbrio entre redução de faltas da memória *cache* L2 e aumento da latência.

No entanto, para as demais organizações o comportamento foi de queda de desempenho. Notamos assim, que o simples aumento no tamanho de memória total do sistema, sem mudança no compartilhamento, levou ao ganho de desempenho final.

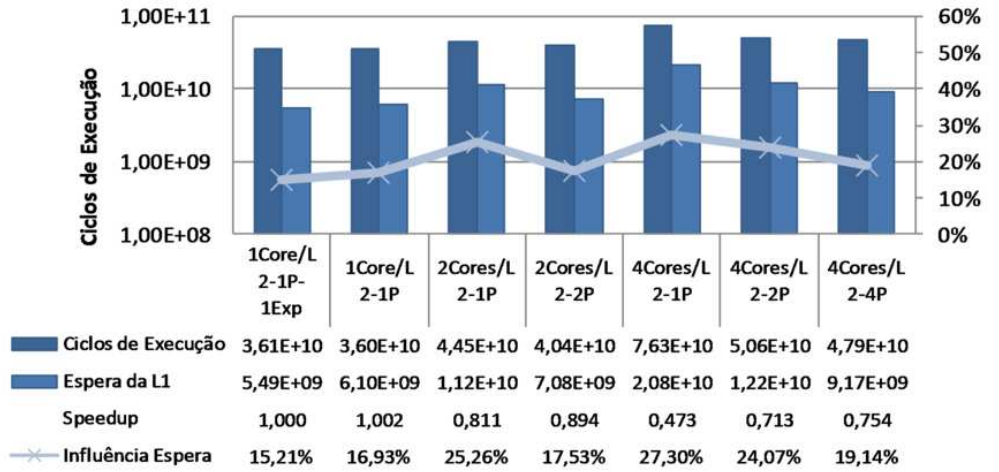


Figura 4.72: Estimativas de execução do segundo experimento (2MB).

O gráfico da Figura 4.73 mostra os valores de consumo de energia e potência para as diversas organizações e quantidades de portas avaliadas com base neste segundo experimento. O consumo de energia aumenta quando ocorre melhora de desempenho, dado

que o sistema dobrou de tamanho de memória e o consumo de energia também dobrou, como podemos observar comparando com a organização 1Core/L2-1P do primeiro experimento.

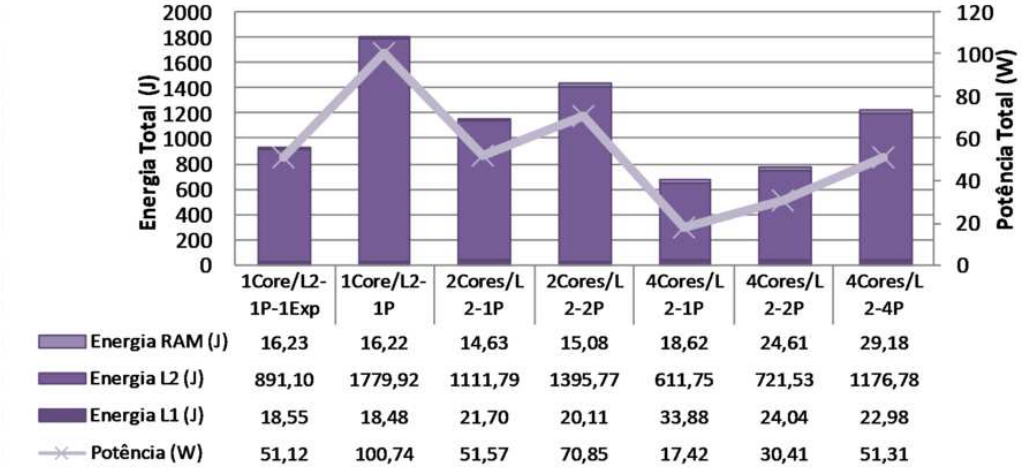


Figura 4.73: Consumo de energia e potência total do sistema de memória do segundo experimento (2MB).

A ocupação de área pela memória *cache* L2 do sistema é apresentada na Figura 4.74, mostrando valores para as diversas organizações avaliadas neste experimento. Como era de se esperar, o tamanho físico da memória *cache* também dobrou para a primeira organização com 2 MB por banco de memória.

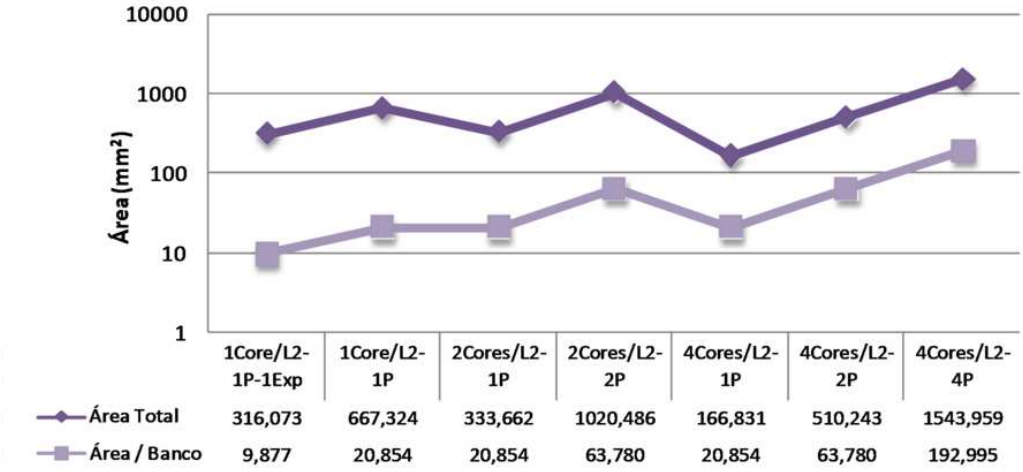


Figura 4.74: Área ocupada pela memória *cache* L2 do segundo experimento (2MB).

4.6.2.3 4 MB por Memória Cache *L2*

Os valores de tempo de execução, ciclos em espera de acesso à memória *cache* L2 (faltas na memória *cache* L1), *speedup* e porcentagem de influência dos ciclos de espera pelo tempo de execução, estimados para o segundo experimento com bancos de memória *cache* de 4 MB, são apresentados na Figura 4.75. Conseguimos novamente perceber com

os resultados, a necessidade de equilíbrio entre tamanho de memória *cache* e latência de acesso. No caso 1Core/L2-1P, o sistema possui 4 vezes mais memória *cache* que o sistema 1Core/L2-1P do primeiro experimento, entretanto, o resultado de desempenho é equivalente, pois, como podemos notar, houve uma redução na taxa de faltas e um acréscimo na latência de acesso, logo, o sistema permaneceu com o mesmo tempo de execução.

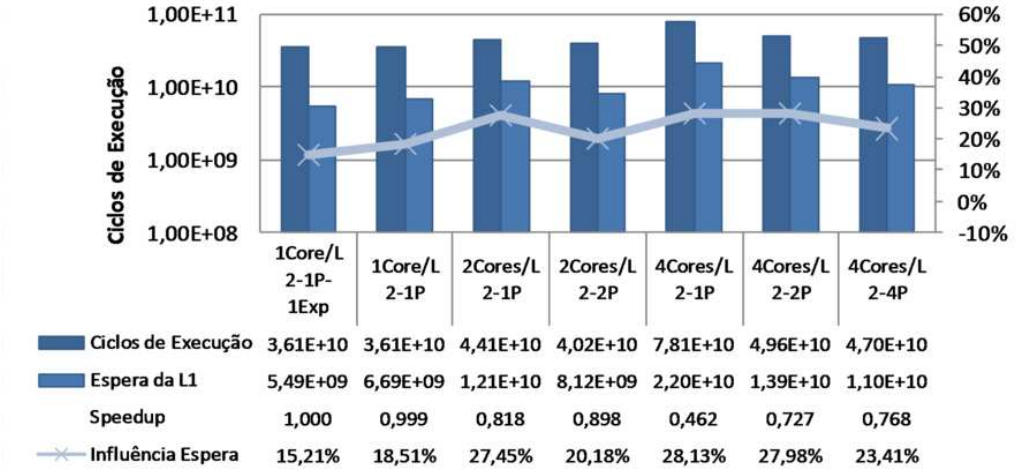


Figura 4.75: Estimativas de execução do segundo experimento (4MB).

O consumo de energia e potência com base neste segundo experimento é apresentado no gráfico da Figura 4.76. Com esse resultado, podemos ver o custo adicionado pelo aumento em 4 vezes do tamanho de memória *cache*, onde o consumo de energia total da organização 1Core/L2-1P foi 4 vezes maior que a mesma organização do primeiro experimento.

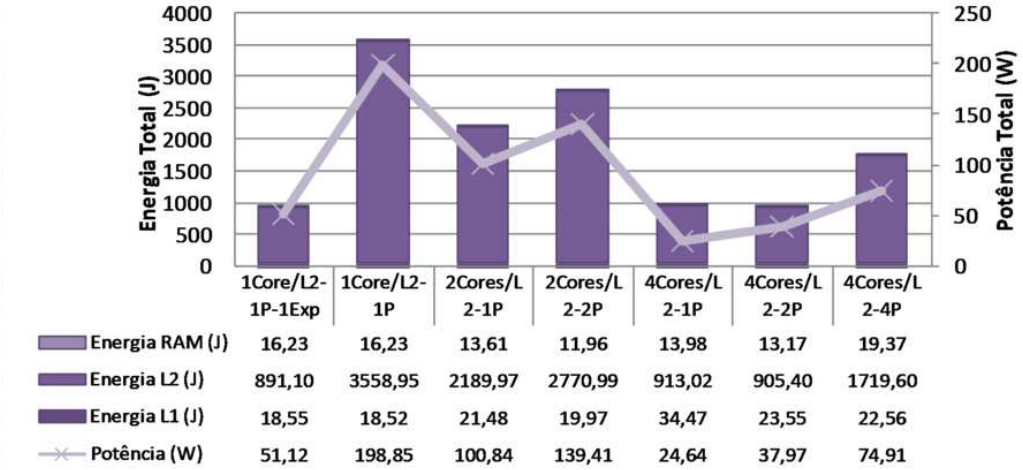


Figura 4.76: Consumo de energia e potência total do sistema de memória do segundo experimento (4MB).

Na Figura 4.77 está ilustrado o gráfico de ocupação de área pelo sistema de memória *cache* L2, o qual apresenta grandes variações de tamanho à medida que mudamos a quantidade de memória total e número de portas.

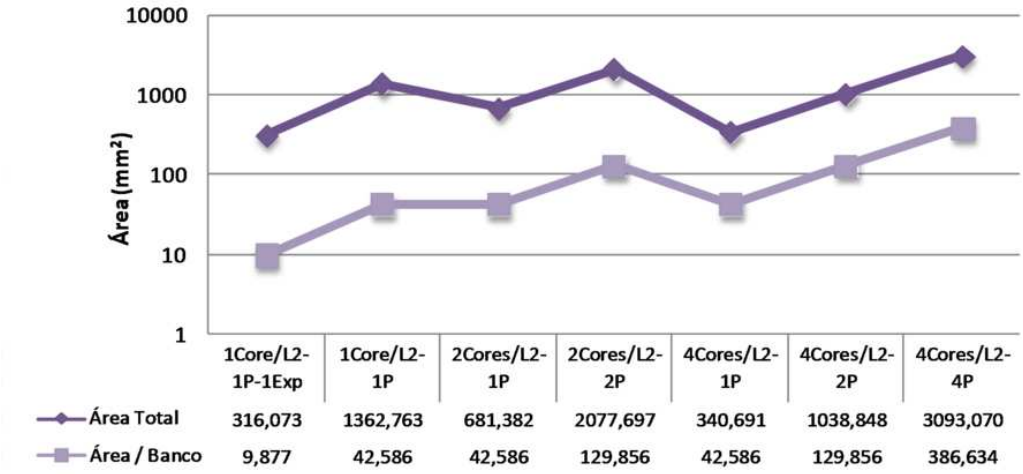


Figura 4.77: Área ocupada pela memória *cache* L2 do segundo experimento (4MB).

A partir das avaliações baseadas no segundo experimento com 1, 2 e 4 MB por banco de memória *cache* L2, podemos concluir que, considerando as contenções de memória *cache*, o aumento no tamanho da memória resulta em ganho no desempenho final, porém, esse aumento no tamanho de memória deve ser feito com cautela, uma vez que o aumento do tamanho em quatro vezes não resultou em ganho de desempenho. Assim, o aumento no tamanho da memória, poderá em alguns casos, levar à melhora no desempenho, porém, sempre resultará em aumentos no consumo de energia e ocupação de área do sistema.

4.6.3 Estimativa de Contenção para o Experimento 3

O primeiro gráfico com resultados sobre estimativas com base no terceiro experimento é apresentado na Figura 4.78. Nessa figura são apresentados valores de tempo de execução, *speedup* e valores de espera por acesso a dados durante faltas da memória *cache* L1, além da porcentagem que esses valores representam no tempo total de execução. Nesse gráfico, nota-se o desempenho equivalente da organização 1Core/L2-1P desse experimento comparado com a mesma organização do primeiro experimento. A segunda melhor organização, 2Cores/L2-2P, apresentou tempos pouco superiores, porém, de melhor desempenho se comparado com a mesma organização do primeiro experimento.

A Figura 4.79 apresenta os valores estimados para consumo de potência e energia com base no terceiro experimento, considerando contenções na quantidade de portas de acesso a dados da memória *cache* L2. Neste gráfico, as organizações 4Cores/L2-2P e 1Core/L2-1P apresentaram menor consumo de energia do sistema, sendo que, a organização 4Cores/L2-2P apresenta redução na energia consumida mesmo com maior tempo para execução da carga de trabalho. Isso se deve à redução na quantidade total de portas do sistema e redução na quantidade de faltas de dados na memória *cache* L2.

O gráfico ilustrado na Figura 4.80 mostra a ocupação de área para as diferentes organizações e quantidades de portas de acesso a dados avaliadas nessa subseção. Mesmo com menor consumo de potência, a organização 4Cores/L2-2P apresenta maior ocupação de área que a primeira organização, mostrando que essa alternativa pode ser interessante apenas do ponto de vista de consumo de potência.

Podemos concluir que o aumento da associatividade nos permite aumentar o compartilhamento de memória *cache* L2 com perdas de desempenho menores que as apresentadas

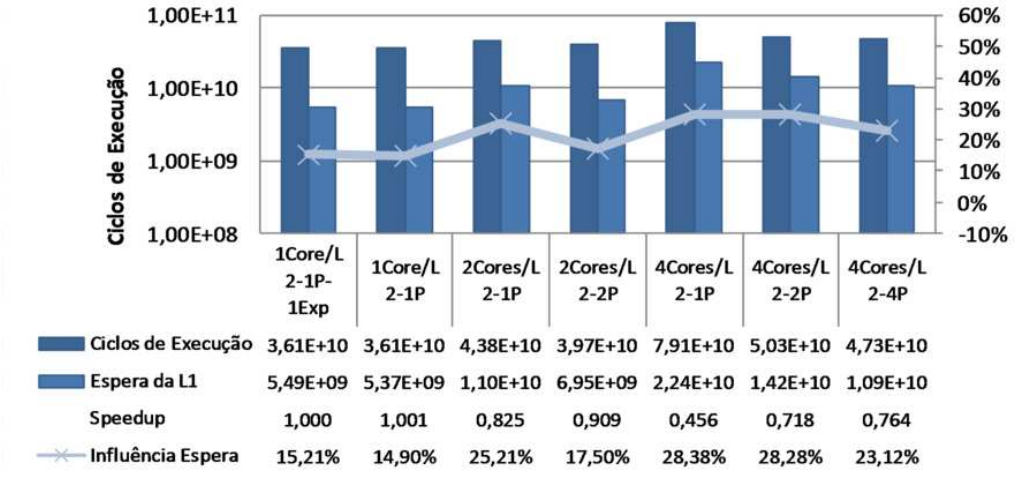


Figura 4.78: Estimativas de execução do terceiro experimento.

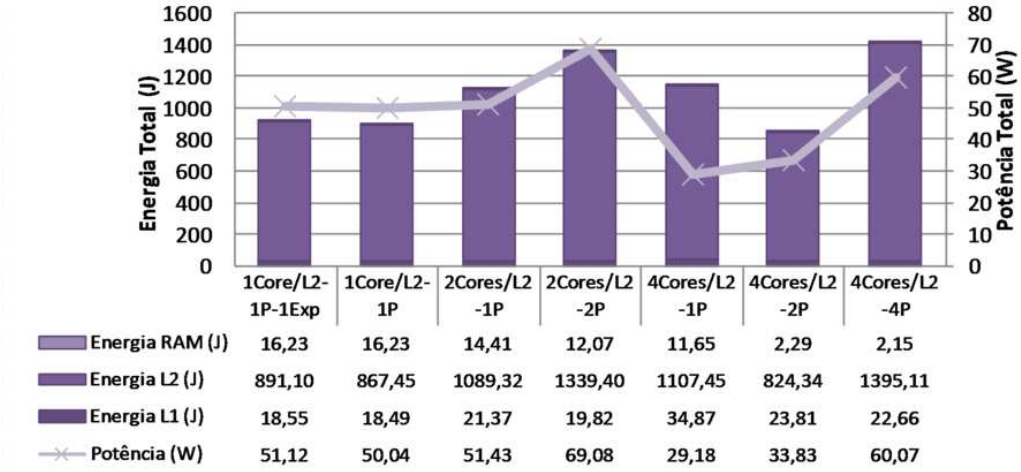


Figura 4.79: Consumo de energia e potência total do sistema de memória do terceiro experimento.

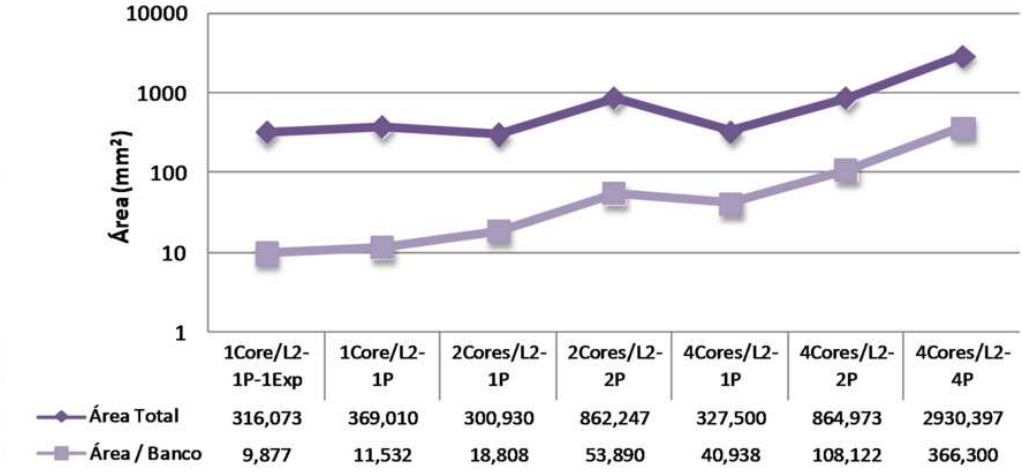


Figura 4.80: Área ocupada pela memória *cache* L2 do terceiro experimento.

no primeiro experimento, entretanto, do ponto de vista de consumo de energia e área, a organização 2Cores/L2-2P apresenta resultados desfavoráveis para ser adotada. Assim, voltamos para a primeira organização que foi a que apresentou melhores resultados de desempenho, com baixo consumo de potência e ocupação de área.

4.6.4 Estimativa de Contenção para o Experimento 4

Presente na Figura 4.81, o gráfico de execução apresenta valores de ciclos de execução, *speedup*, espera durante faltas de dados na memória *cache* L1 e porcentagem representativa dessas esperas no tempo total de execução para as diversas organizações avaliadas, com base no quarto experimento. Os resultados, considerando a contenção de portas, mostra bom desempenho para as organizações 1Core/L2-1P e 2Cores/L2-2P, sendo que a primeira organização obteve *speedup* de 7,9% com relação aos resultados estimados para o primeiro experimento.

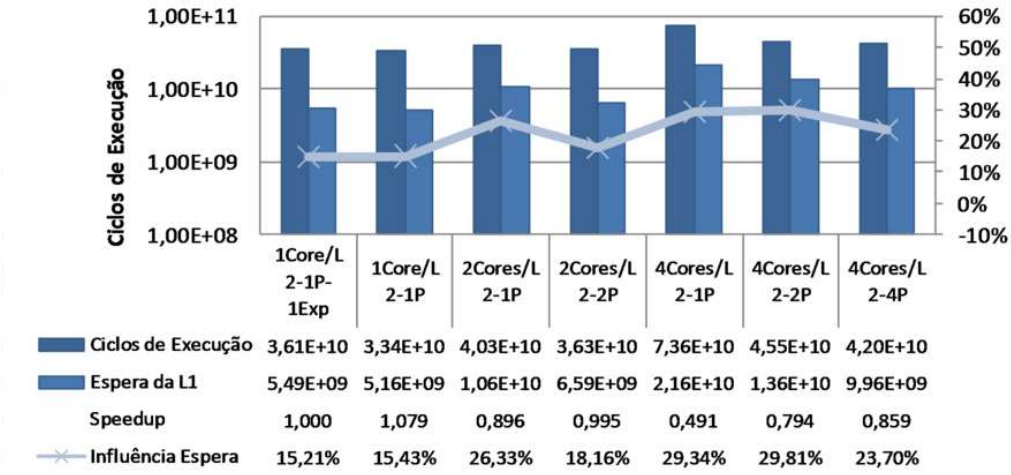


Figura 4.81: Estimativas de execução do quarto experimento.

O consumo de energia e potência total estimado com contenções no número de portas da memória *cache* L2, com base no quarto experimento, é apresentado na Figura 4.82. Podemos ver nessa figura que o consumo de energia da memória *cache* L2 é bastante elevado conforme aumentamos a quantidade de portas e compartilhamento de memória *cache*. Esse aumento advém do consumo estático: uma vez que o tempo de execução da carga de trabalho aumenta, o consumo estático também aumenta, elevando dessa forma o consumo total. Logo, apenas as organizações 1Core/L2-1P, 2Cores/L2-1P e 2Cores/L2-2P apresentam consumo razoável de energia.

Os valores de ocupação de área para as memórias *cache* L2, com restrições na quantidade de portas, baseadas nas organizações do quarto experimento são apresentados na Figura 4.83. Notamos nesse gráfico, o aumento de área ocupada conforme tem-se o aumento na quantidade de portas da memória *cache*. Assim, apenas as memórias com uma porta de acesso a dados apresentam ocupações de área razoáveis.

As estimativas de contenção de portas, baseado no quarto experimento, apresentaram bons resultados de desempenho, consumo de energia e ocupação de área, sendo que a organização 1Core/L2-1P apresentou os melhores resultados dentre todas as avaliadas com contenções. Os bons resultados são ocasionados pela redução nas taxas de falta de dados, combinadas com baixa latência e pouca contenção pelas portas de acesso a dados.

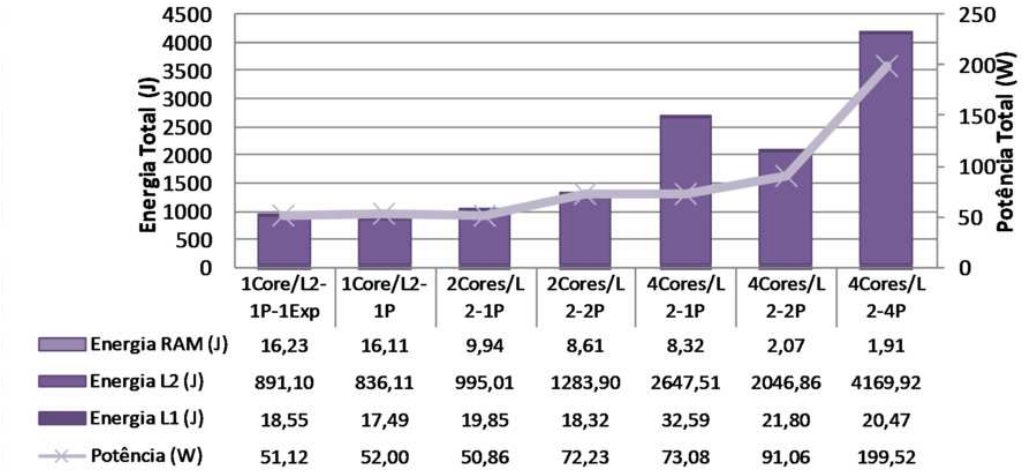


Figura 4.82: Consumo de energia e potência total do sistema de memória do quarto experimento.

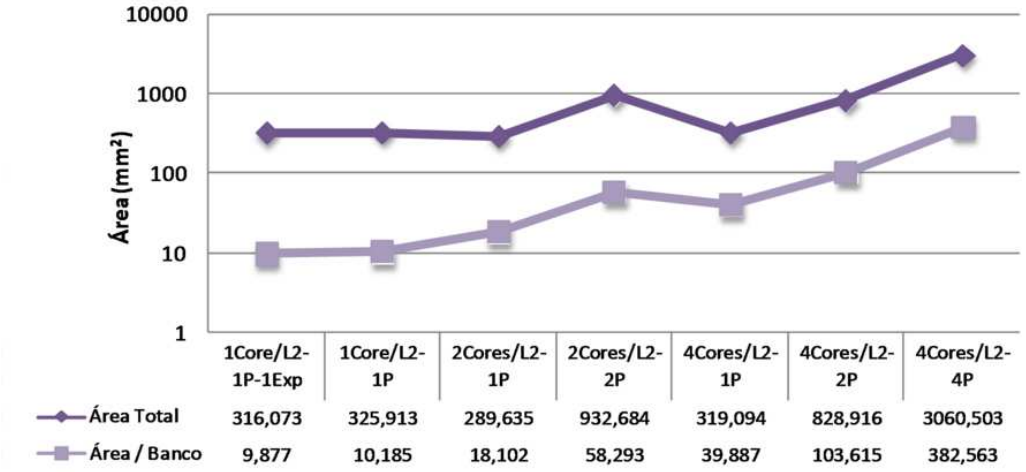


Figura 4.83: Área ocupada pela memória *cache* L2 do quarto experimento.

De forma análoga, considerando os resultados de estimativa de contenção com base nos diversos experimentos, podemos ver que as técnicas tradicionais apresentam ainda bons ganhos de desempenho quando consideramos as contenções, porém, nenhuma técnica apresentou resultados bons o suficiente para que fosse possível aumentar o compartilhamento de memória *cache* mantendo o desempenho. Entretanto, com base nos resultados, podemos afirmar que o compartilhamento de memória *cache* entre núcleos de processamento pode ser útil para sistemas com requisito de baixo consumo de potência e área.

5 CONCLUSÕES

" Pior do que não terminar uma viagem é nunca partir "
— AMYR KLINK

Esta dissertação apresentou um estudo a respeito dos conceitos fundamentais sobre memórias *cache*, além dos principais modelos de processadores disponíveis no mercado. O objetivo foi formar uma base para a proposta e avaliação de memórias *cache* compartilhadas em processadores *multi-core* e *many-core*.

Porém, analisando os atuais processadores *multi-core* e as pesquisas a respeito do compartilhamento de memória *cache*, ainda não é possível apontar uma tendência bem definida sobre o compartilhamento de memória *cache* entre os núcleos. Nesse contexto, esta dissertação foi conduzida com foco na avaliação da influência do compartilhamento da memória *cache* L2. Para isso, foi explorado o espaço de projeto, avaliando a influência do tamanho da memória *cache*, da associatividade, do tamanho da linha de dados e dos níveis na hierarquia de memória, a fim de definir a influência desses fatores no desempenho de sistemas *multi-core*.

Além desse objetivo, este trabalho trouxe o diferencial, inexistente em trabalhos correlatos, que é a modelagem pelo simulador Simics das latências das memórias *cache*, que foram estimadas pela ferramenta Cacti, gerando assim dados de latências de acesso, consumo de potência e ocupação de área próximas as existentes em sistemas reais. Além disso, esse trabalho traz avaliações considerando contenções pelo número de portas da memória *cache*, o que não é considerado nos demais trabalhos, formando assim bases mais sólidas para comparações.

A partir dos resultados apresentados, é possível avaliar e concluir sobre os diversos comportamentos de aplicações e sistemas para os experimentos propostos. De forma geral, os experimentos mostram a importância da integração do projeto de organização de memória *cache* e o projeto físico, a fim de obter a melhor troca entre desempenho e consumo de potência e ocupação de área.

Com relação às avaliações sem contenções no número de portas da memória *cache*, mesmo com pequenas perdas de desempenho, de modo geral, para as cargas de trabalho utilizadas, o compartilhamento de dados reduz o consumo de potência do sistema de memória, além de reduzir a ocupação de área pela memória compartilhada. Além disso, é possível concluir que para aplicações paralelas com acesso a dados compartilhados, o aumento no compartilhamento da memória *cache* L2 pode trazer benefícios como apresentado no primeiro experimento para a organização 4Cores/L2 (+0,50%).

Entretanto, para aplicações, nativamente paralelas, que não apresentam compartilha-

mento de dados ou dependência entre fluxos de execução como a aplicação EP, existirá perdas de desempenho à medida que a memória *cache* L2 for compartilhada por mais núcleos de processamento, devido ao aumento de faltas de dados. Além disso, os ganhos por compartilhamento de dados são nulos, o que resulta apenas no sobrecusto das altas latências de acesso a grandes blocos de dados.

Portanto, para aplicações independentes executando em paralelo em processadores *multi-core*, a melhor hierarquia de memória *cache* será uma memória *cache* L2 para cada núcleo de processamento, enquanto que, para aplicações paralelas com compartilhamento de dados, como a maioria das aplicações científicas avaliadas nesse estudo, o compartilhamento da memória trará benefícios. No entanto, os ganhos para a organização 4Cores/L2 relativos ao primeiro experimento não ocorre quando se considera as contenções de portas da memória *cache* L2, uma vez que a contenção imposta ou a latência gerada pelo aumento no número de portas inibe os ganhos de desempenho.

O segundo experimento sem contenções, que tratou sobre a influência do tamanho da memória *cache* L2, com fatias de memória *cache* iguais a 4 MB, apresentou um bom equilíbrio entre tamanho de memória *cache* e tempo de acesso a dados. Sendo que esse experimento apresentou ganhos para as organizações 4Cores/L2 (+0,45%), 8Cores/L2 (+0,39%), 16Cores/L2 (+2,04%) e 32Cores/L2 (+3,16%). Quando considerado as contenções do sistema de memória, os ganhos para a organização 4Cores/L2 não ocorrem, onde apenas a organização 1Core/L2-1P com 2 MB apresenta ganho de desempenho resultante do equilíbrio entre redução na quantidade de faltas, latência de memória *cache* e a restrição no número de portas do sistema.

No terceiro experimento sem considerar contenções, com o aumento da associatividade podemos ver que as aplicações começaram a apresentar mais ganhos de desempenho com o compartilhamento da memória *cache*, porém, comparando com o experimento base, fica claro que o custo extra no tempo de acesso a dados não compensa o uso dessa técnica para se obter alto desempenho. Se considerada a contenção da memória *cache*, o uso de associatividade não influência no desempenho do sistema, mantendo o sistema em desempenho equivalente ao primeiro experimento.

Sobre o aumento do tamanho de linha da memória *cache*, sem considerar contenções, presente no quarto experimento, podemos concluir que essa técnica apresenta ganho de desempenho ao ser utilizada, apresentando sobrecusto em termos de consumo de potência e ocupação de área. Sendo que, foram obtidos ganhos nas organizações 1Core/L2 (+2,38%), 2Cores/L2 (+3,07%), 4Cores/L2 (+2,47%) e 8Cores/L2 (+2,55%). Logo, a melhor organização foi a 2Cores/L2, que apresentou maior desempenho e ainda obteve ganhos para todas as aplicações da carga de trabalho avaliada. Este quarto experimento, considerando as contenções no número de portas de acesso a dados, apresentou para a organização 1Core/L2-1P ganhos de desempenho e redução no consumo de energia em comparação ao primeiro experimento com contenções deste experimento, com um pequeno aumento na área ocupada pela memória *cache* L2.

A adição de um nível na hierarquia de memória *cache*, mostrada no quinto experimento, sem a modelagem de contenções, gerou mais latências do que o esperado, sendo que nesse experimento os resultados foram ruins do ponto de vista de desempenho, consumo de potência e ocupação de área. Entretanto, devemos considerar que existe grande área de estudos sobre a adição de novo nível na hierarquia de memória, assim, os resultados apresentados mostram uma tendência geral de comportamento. Assim, devemos considerar que mais estudos sobre as formas de adoção dessa memória *cache* L3 em um processador *multi-core* são necessários a fim de definir claramente o comportamento dessa

memória sobre diferentes configurações.

De acordo com os experimentos sem considerações de contenções na quantidade de portas de acesso a dados, podemos ressaltar que as tradicionais técnicas de aumento no tamanho da memória *cache*, aumento na associatividade e aumento nos níveis da hierarquia de memória, antes só penalizadas por custos físicos, não apresentam bons resultados para as atuais tecnologias de integração. Entretanto, o compartilhamento de memória *cache* e técnicas de aumento no tamanho da linha, apresentam potencial para redução no número de faltas de dados.

Considerando as estimativas de custos e contenções do sistema de memória, apresentados no sexto experimento, nota-se que o compartilhamento de memória *cache* impõe várias restrições. Os custos adicionais são em relação a latência de acesso, potência e área para se aumentar a quantidade de portas do sistema, ou no caso, em que se utiliza menor número de portas, a contenção gerada para o acesso a memória faz o sistema enfrentar a espera de muitos ciclos perdidos por espera de dados. No contexto deste experimento, se apresentam eficientes para aumento de desempenho apenas o aumento no tamanho da memória *cache* (+0,20%) e o aumento no tamanho da linha (+7,90%). Logo, para o pior caso, com as contenções de memória *cache*, a melhor técnica a ser utilizada é o aumento no tamanho de linha, que aumentou o desempenho, com redução no consumo de energia e sem grandes custos com relação a ocupação de área.

Avaliando os experimentos com e sem restrições no número de portas de acesso a dados, para não ser injustos com as possíveis otimizações no sistema de acesso a dados, a melhor configuração de memória a ser adotada deverá estar entre as que obtiveram melhores ganhos nos dois casos. Assim, a melhor arquitetura de memória *cache* deve estar entre o uso totalmente compartilhado de um tamanho razoável de memória *cache* (32Cores/L2 do experimento 2 com 4MB por fatia de memória *cache* L2), o aumento no tamanho de linha da memória *cache* com pequeno ou nenhum compartilhamento de memória *cache* L2 (2Cores/L2 do experimento 4) e (1Core/L2-1P do experimento 6 baseado no aumento da linha de dados) respectivamente.

Portanto é possível constatar que a organização 32Cores/L2 do segundo experimento com 4MB sofrerá problemas físicos para sua implementação, uma vez que uma memória de 4 MB com 32 portas de acesso, deverá ocupar 20601 mm^2 segundo modelagem no Cacti. Enquanto a memória da organização 2Cores/L2 do quarto experimento ocupará área 20 vezes menor (386 mm^2). Além disso, apenas a organização 2Cores/L2 do quarto experimento apresentou ganhos de desempenho em todas aplicações o que é de grande importância para processadores de propósito geral. Logo, em um sistema real apenas as organizações 1Core/L2 e 2Cores/L2 com tamanho total igual a 32MB (bancos de 2 MB compartilhados), com tamanho de linha igual a 128 B, permanecem como boa escolha de implementação física em sistemas de propósito geral.

Para processadores *many-core* as memórias *cache* de arquitetura uniforme deverão sofrer ainda mais os problemas associados a latência do fio. Neste cenário futuro, o uso de memórias NUCA (non-uniform *cache* architecture) podem ser consideradas, sendo que, para essas memórias não uniformes o compartilhamento de porções de memória *cache* poderão ser feitas de forma menos custosa, uma vez que não haverá o sobrecusto sobre o acréscimo no número de portas de acesso. Todavia, deverá ser considerado as restrições no compartilhamento de acordo com o conjunto de aplicações a serem utilizadas, conforme apresentado em nossas avaliações de resultados.

Como trabalho futuro podemos ressaltar a importância em extender o trabalho feito sobre a memória *cache* L3 para que possamos obter conclusões mais pontuais a respeito

da adição de um nível na hierarquia de memória. Outro ponto interessante para trabalhos futuros é a adaptação da aplicação para uma dada hierarquia de memória, onde podemos classificar a aplicação de acordo com seu padrão de comunicação e acesso a dados, e a partir desse ponto, moldar esse padrão para que se adapte melhor aos tamanhos de memória, compartilhamento, tamanho de linha, entre outros fatores, a fim de melhorar o desempenho das aplicações paralelas. Além desses, nota-se que são de grande importância os trabalhos que avaliem as próximas gerações de memória *cache*, assim, estudos acerca de memórias *cache* de arquitetura não-uniforme parecem ser bastante promissores, sendo também um foco para trabalhos futuros.

APÊNDICE A SISTEMAS DE MEMÓRIA

“ In theory, there is no difference between theory and practice;
In practice, there is. ”
— CHUCK REID

Desde a criação do computador, a criação de formas de armazenamento de dados é crucial, seja para guardar resultados de computações, seja para armazenar instruções de máquina.

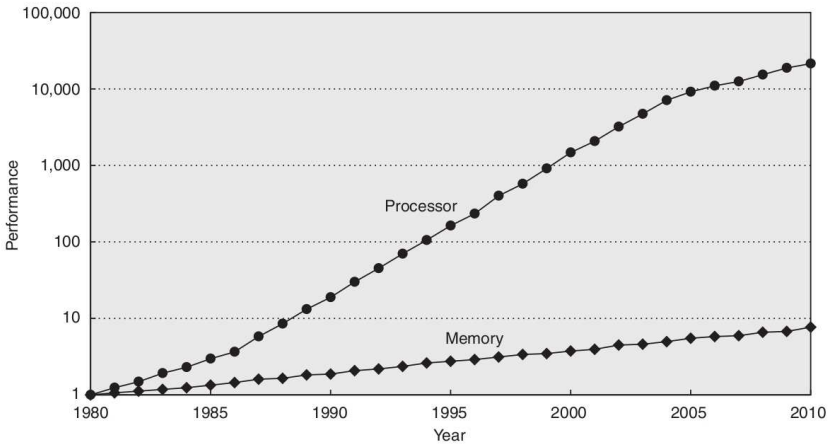


Figura A.1: Distanciamento de desempenho entre processador e memória DRAM (*Dynamic Random Access Memory*) ao longo dos anos (HENNESSY; PATTERSON, 2007).

Mais do que o armazenamento de dados, a partir de 1980, iniciaram-se os estudos sobre formas de aumentar as velocidades dos meios de armazenamento, uma vez que o processamento depende das instruções armazenadas na memória do computador e muitas vezes também depende dos operandos armazenados. Com os agressivos crescimentos de velocidade dos processadores a cada ano, e a velocidade de memória apresentando crescimentos bem inferiores, fez-se necessário a criação de hierarquias de memórias que tiram proveito do custo/desempenho de tecnologias de memória. Podemos ver na Figura A.1 o distanciamento entre as velocidades do processador e da memória DRAM.

A.1 Hierarquias de Memória

Tendo em vista que memórias rápidas possuem custo muito alto, uma hierarquia de memória é organizada em vários níveis, onde as memórias mais rápidas ficam próximas dos elementos de processamento, e então cada nível de memória abaixo é mais lenta. Assim, no topo da hierarquia, mais próximo ao processamento, temos memórias de alta velocidade, porém de tamanho limitado e de alto custo, e em cada nível desta hierarquia temos memórias menos rápidas, de tamanhos maiores e de custo decrescentes.

Geralmente, o conjunto de dados disponíveis na hierarquia mais alta é um subconjunto dos dados do nível inferior e assim por diante. Desta forma, o objetivo é obter um sistema de custo tão baixo quanto o custo da memória de mais baixo nível, e com a velocidade da memória do nível mais alto da hierarquia, escondendo então as latências da memória de armazenamento não volátil através dessa hierarquia.

Atualmente, os níveis de hierarquia de memória são implementados utilizando três tecnologias.

As memórias mais próximas do processador, as memórias *cache*, utilizam tecnologias SRAM (*Static Random Access Memory*). Essa tecnologia oferece altíssimo desempenho para leituras e gravações, porém é a mais cara de toda hierarquia, por isso a sua utilização em pequenos tamanhos, seguindo a tendência de que quanto mais rápida, mais cara e menor. Além disso, esta memória não retem dados quando desligada, necessitando permanecer ligada para que seus dados sejam mantidos.

As memórias principais de trabalho utilizam tecnologias DRAM (*Dynamic Random Access Memory*). Nesse nível empregam-se memórias maiores, porém podemos utilizar memória de menor custo uma vez que os níveis superiores deverão esconder as altas latências de acesso a este nível. Na tecnologia DRAM, além de necessitar permanecer ligada para manter os dados, é preciso o uso de *refresh* dos dados em curtos espaços de tempo.

O nível mais baixo de memória utiliza tecnologias de armazenamento magnético não voláteis. Para este nível o grande ponto chave é o tamanho total de armazenamento em que os dados se mantenham, mesmo quando a energia de alimentação for cortada. Assim, o uso de meios magnéticos se faz necessário em pelo menos uma hierarquia de memória. Este nível de memória tradicionalmente utiliza meios magnéticos que são escritos e gravados por dispositivos mecânicos, tendo assim péssimo desempenho.

A Figura A.2 apresenta uma ilustração com diferentes níveis de hierarquia de memória e os respectivos tempos de acesso e tamanho médio.

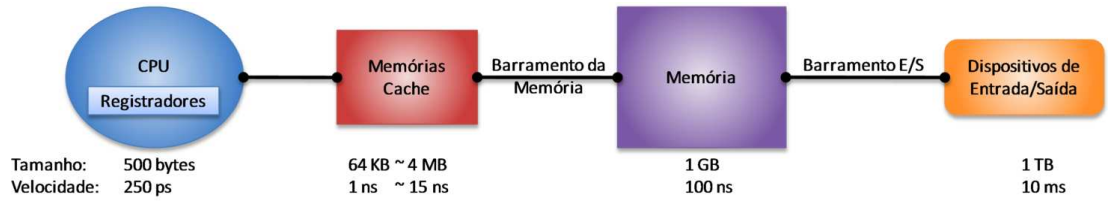


Figura A.2: Diferentes níveis da hierarquia de memória (HENNESSY; PATTERSON, 2007).

A.1.1 Princípios de Funcionamento

A ilusão criada com o uso de hierarquias de memórias, onde os acessos tendem a ser tão rápidos quanto os acessos do nível mais veloz e de tamanho limitado somente pela hierarquia de maior capacidade, é possível graças ao princípio de localidade.

O princípio de localidade pode ser visto como a necessidade do processador durante a execução de uma aplicação, só enxergar parte da memória, ou seja, a parte da memória referente à localização dos dados e instruções necessários para a computação da aplicação, ou necessários para a computação de determinado trecho de código. Ainda, pode-se entender esse princípio como sendo o comportamento dos programas em repetir trechos de código utilizando dados repetidos ou acessar repetidamente dados próximos aos que estão sendo acessados.

Podemos classificar o princípio da localidade em dois tipos diferentes:

- Localidade Temporal: Essa localidade diz respeito à necessidade temporal dos dados: se um item é referenciado, ele tende a ser referenciado novamente em breve. Assim, existe a necessidade de manter os dados recentemente acessados nas hierarquias mais próximas ao processador.

- Localidade Espacial: Essa localidade é referente ao local de armazenamento da memória: se um item é referenciado, os itens de endereços próximos tendem a ser referenciados em breve. Desta forma existe a necessidade de não apenas trazer para hierarquias mais próximas ao processador os dados referenciados, mas sim o bloco de dados, contendo os dados de endereços próximos.

A.1.2 Acertos e Faltas na Busca de Dados

Quando um dado é solicitado a uma determinada hierarquia de memória, se este dado estiver presente e válido nessa hierarquia, então diz-se que houve um acerto na busca de dados. Quando o dado pesquisado não se encontra na hierarquia pesquisada, então fala-se que houve uma falta na busca do dado.

No caso do dado não ser localizado na hierarquia corrente, esse deverá ser trazido das hierarquias mais baixas. Porém, as memórias possuem tempos diferentes no caso de um acerto ou de uma falta de dados. Por se tratar de uma hierarquia com níveis mais lentos, o tempo de acesso de um dado no caso de um acerto na busca é inferior ao tempo de acesso do dado que houve uma falta em sua busca e esse deverá ser trazido de hierarquias mais lentas.

Então, é tratado como tempo de acerto ou tempo de acesso a dados, o tempo necessário para acessar um determinado nível de memória, incluindo o tempo para se determinar se o acesso retornará um acerto ou uma falta. O tempo de falta é o tempo necessário para buscar um bloco de dados do nível inferior para o nível superior da hierarquia de memória, incluindo o tempo de acesso ao bloco, transmití-lo de um nível a outro e inserí-lo no nível que experimentou a falta.

Como a principal funcionalidade da hierarquia é o desempenho, quanto menor for a taxa de faltas na busca de dados, melhor será o desempenho obtido.

A taxa de acertos de busca em um determinado nível da hierarquia nada mais é que a fração dos acessos à memória encontrados naquele nível. A taxa de faltas é a proporção dos acessos à memória não encontrados em tal nível. A taxa de acertos e de faltas são importantes métricas para definir o desempenho em memórias de níveis próximos ao processador.

A.1.3 Fonte de Acertos e Faltas

Para saber as melhorias que possam ser feitas em uma hierarquia de memória, a fim de obter um menor número de faltas de dados, é necessário saber as origens das faltas. Podemos classificar as faltas de dados em quatro categorias simples (HENNESSY; PATTERSON, 2007):

- Compulsórios – O primeiro acesso a qualquer bloco não pode ser realizado pois a *cache* está vazia porque o sistema acabou de entrar em funcionamento, ou é o primeiro acesso àquele endereço. Assim, o bloco pesquisado deverá ser trazido para a *cache*. Essas faltas são também chamadas de faltas de início frio ou de primeira referência.

- Capacidade – Se a *cache* não puder conter todos os blocos de dados necessários durante a execução de um programa, ocorrerão faltas de capacidade (além de faltas compulsórias), pois dessa maneira alguns blocos deverão ser descartados e mais tarde recuperados.

- Conflito – Se a *cache* não puder manter mais dados de mesmo endereçamento pois já possui dados em tal referência para o endereçamento, então ocorrerá faltas de conflito, também conhecidos como faltas de colisão ou faltas de interferência.

- Coerência – Quando a *cache* precisa remover ou limpar alguns blocos de dados a fim de manter em um estado coerente um sistema com múltiplas *caches*, tipicamente em um sistema multiprocessado.

Classificados os quatro tipos simples de faltas, podemos iniciar estudos mais aprofundados sobre como obter melhores desempenhos das hierarquias de memória.

A.2 Memórias *Cache*

Os sistemas de memórias são indispensáveis nos computadores e por isso muitos projetistas dedicam esforços em busca de sistemas de memórias eficientes. As hierarquias de memória que estão dispostas entre o processador e a memória principal de trabalho são conhecidas como memórias *cache*.

O termo *cache* é definido como "um lugar seguro para esconder ou guardar coisas", segundo (LANDAU, 2000). Dessa maneira, o termo memória *cache*, embora seja utilizado para denominar hierarquias próximas aos processadores, pode referenciar qualquer armazenamento usado para tirar proveito da localidade de acesso, ou seja, qualquer nível da hierarquia de memória. Além disso, o termo *cache* também pode ser aplicado sempre que se emprega *buffers* para reutilizar itens que ocorrem comumente.

Neste trabalho trataremos como memórias *cache* somente as hierarquias entre o processador e a memória principal.

As memórias *cache* apareceram primeiramente nos computadores de grande porte dedicados à pesquisa no início da década de 1960 e nos computadores de produção, mais tarde nessa mesma década. Atualmente, todo computador de propósito geral possui pelo menos um nível de hierarquia de memória *cache*.

Ao ser gerado o endereço para acesso de algum dado da memória, podemos definir, de forma simples, o funcionamento de uma memória *cache* em três passos:

- Verificar se a memória *cache* possui cópia da posição de memória correspondente.

- Se possuir, verificar a posição da *cache* onde se encontra o endereço correspondente e retornar a cópia ao processador.
- Caso não possua, trazer o conteúdo da memória principal e escolher a posição da *cache* onde a cópia do dado será armazenada e retornar a cópia ao processador.

Assim, sabendo o funcionamento básico de uma memória *cache*, podemos abordar os assuntos relacionados com seu funcionamento.

A.2.1 Elementos de uma Memória *Cache*

Como uma memória *cache* possui diversos espaços para gravação de dados e cada local da memória *cache* pode armazenar conteúdo refente a diversos endereços da memória principal, algumas informações devem ser adicionadas para sabermos, durante uma pesquisa, se um dado requerido encontra-se na memória *cache*.

A utilização de *tags* junto ao dado copiado à memória *cache* é útil uma vez que a *tag* contêm as informações de endereço necessárias para identificar se a palavra pesquisada corresponde com o dado requisitado.

Assim, podemos observar na Figura A.3, um exemplo de memória *cache* com dados copiados da memória principal e a *tag* de identificação. Podemos ver no caso ilustrado, que a *tag* de dados precisa conter apenas a parte superior do endereço de memória referente para que seja possível sua identificação.

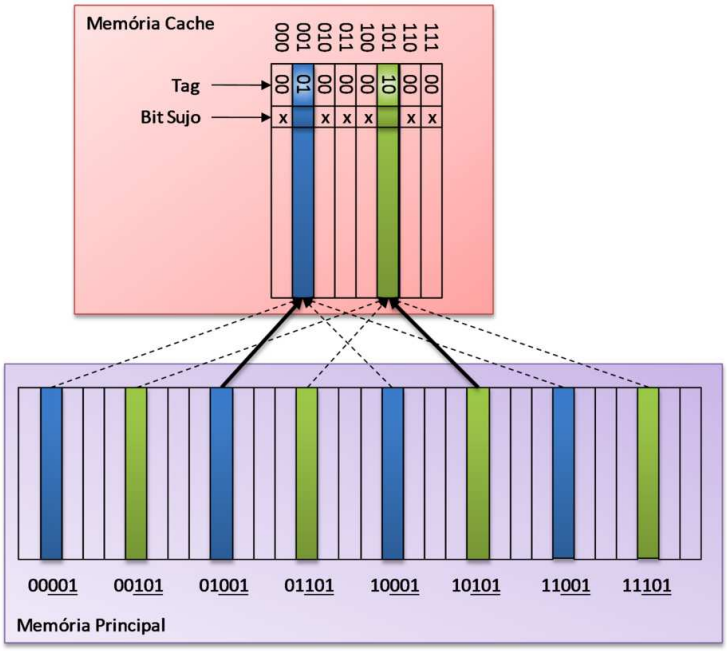


Figura A.3: Diagrama de uma memória *cache* com mapeamento direto, apresentando a *tag* e o *bit* sujo (*bit* de validade ou *dirty bit*), (PATTERSON; HENNESSY, 2005).

Além da *tag* de dados, é necessário uma maneira para saber se um bloco de dados possui informações válidas. Um exemplo claro dessa necessidade é, ao iniciar o sistema, quando a memória *cache* encontra-se apenas com possíveis sujeiras, sem dados reais copiados para si. O método mais comum para saber se a *tag* deve ser ignorada ou não é a inclusão de um *bit* de validade para indicar se a entrada contém um bloco válido.

A.2.2 Tamanho do Bloco de Dados

O tamanho do bloco ou tamanho da linha de dados é a quantidade de dados que serão trazidos da memória principal para a memória *cache* em cada nova busca. Assim, quanto maior o bloco de dados, maior será a quantidade de dados próximos que serão copiados.

Desta maneira, o aumento do tamanho da linha de dados tende a reduzir o número de faltas compulsórias pois privilegiando o princípio de localidade espacial. Porém, o aumento do tamanho dos blocos resultará em uma redução de posições totais da memória *cache*, podendo ocasionar outros tipos de faltas se a memória *cache* for pequena. Além disso, o tamanho do bloco influencia no tempo de penalidade caso ocorra falta de dados. Logo, a definição do tamanho dos blocos também deve levar em conta a largura de banda de comunicação entre os níveis da hierarquia de memória.

A.2.3 Estratégias de Mapeamento de Dados

A estratégia adotada para definir o modo em que os blocos de dados deverão ser posicionados na memória *cache* é chamada de estratégia de mapeamento da *cache*.

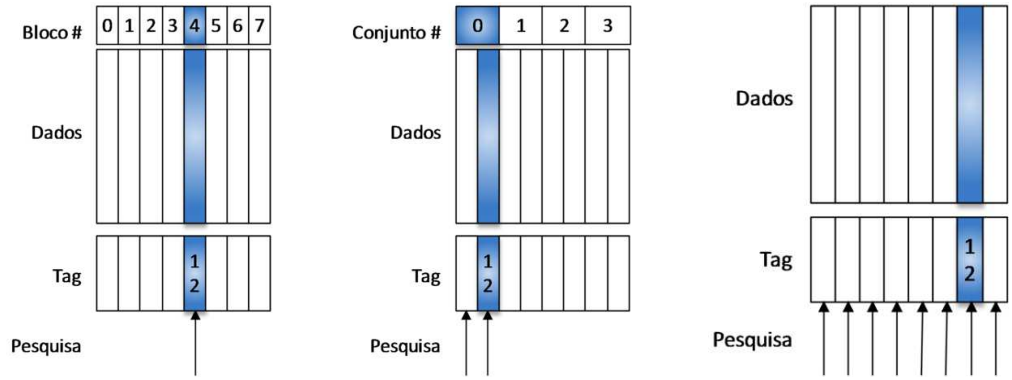
Um modo usual de mapeamento de memória *cache* é o mapeamento direto. Nesse modo de mapeamento cada local da memória é mapeado exatamente para um local na *cache*. Assim, a *tag* trará informações adicionais do endereço base e esse será diretamente associado ao endereço da memória *cache*. Esse modo de mapeamento é atraente pois é de fácil planejamento, uma vez que, se o número de entradas da memória *cache* for uma potência de dois, então o tamanho do endereço base pode ser calculado simplesmente usando os log_2 *bits* menos significativos, e assim os dados podem ser associados à memória *cache* de forma direta.

Outra estratégia de mapeamento de memória *cache* é o chamado mapeamento totalmente associativo. Nesse mapeamento, um bloco pode ser posicionado em qualquer local da *cache*. Porém, nesse mapeamento totalmente associativo, para que um dado seja procurado na memória *cache*, todas as entradas dela precisam ser pesquisadas. Para a implementação desse tipo de mapeamento, é comum a utilização de comparadores paralelos, sendo que, para cada bloco deverá existir um comparador de endereços para a pesquisa de blocos. A inclusão desses comparadores aumentam significantemente o custo de uma memória *cache*, por isso, que esse esquema só é viável em memórias *cache* com pequeno números de blocos.

O ponto de equilíbrio entre os modos de mapeamento é o mapeamento associativo por conjunto. Nesse modo de mapeamento, existe um número fixo de locais (no mínimo dois) onde cada bloco pode ser colocado. Uma determinada memória *cache*, se for associativa por n conjuntos, sendo n o número de locais de mesmo endereço base onde cada bloco pode ser alocado, é chamada de memória *cache* associativa por conjunto de n vias. Para esse modo de mapeamento associativo por conjunto cada bloco é mapeado para um conjunto único na memória *cache* de acordo com o endereço base, e um bloco pode ser alocado em qualquer posição desse conjunto. Logo, uma pesquisa de busca deverá acessar o conjunto referente ao endereço e depois pesquisar entre os elementos do conjunto o endereço requisitado.

A Figura A.4 ilustra diversos modos de mapeamento de memória *cache*, onde temos os blocos utilizando mapeamento direto A.4(a), ou associativo por conjunto A.4(b) e ainda uma memória *cache* totalmente associativa A.4(c).

O número total de blocos da *cache* é igual ao número de conjuntos multiplicados pela associatividade, porém, em uma *cache* associativa por conjunto, o número de compara-



(a) Mapeamento Direto. (b) Conjunto Associativo. (c) Totalmente Associativo.

Figura A.4: Alocamento de um bloco de memória em uma memória *cache* com 8 blocos, em diferentes tipos de mapeamento: A.4(a) mapeamento direto, A.4(b) mapeamento associativo por conjunto e A.4(c) totalmente associativo, adaptado de (PATTERSON; HENNESSY, 2005).

dores de endereço é basicamente igual ao número de vias de associatividade. Assim, o aumento do grau de associatividade de uma dada memória tende a reduzir as faltas de conflito de endereços, mas também reduz a quantidade de endereços base, aumentando o custo relacionado aos comparadores de endereço.

A.2.4 Mecanismos de Busca e Pré-Busca

Os mecanismos de busca e pré-busca, conhecidos também com *fetch* e *pre-fetch*, são responsáveis por buscar um dado em caso de falta (*fetch*) e fazer uma pré-busca de dados (*pre-fetch*) tentando evitar faltas de dados.

Durante uma falta de dados, a busca de dados pode ser feita de duas formas diferentes para fornecer os dados ao processador:

- Palavra Crítica Primeiro – O dado requisitado é buscado na memória e enviado ao processador, assim que disponível. Só então os dados restantes do bloco continuam a ser carregados da memória, enquanto o processador segue com o processamento.
- Reinício Atrasado – O bloco é buscado na memória seguindo a ordem normal dos dados, porém, assim que o dado necessário chegar, esse deve ser enviado ao processador, e então o resto do bloco é carregado.

Essas políticas foram idealizadas após ser identificado que o processador, na maioria dos casos, não necessita de todas as informações buscadas, ou seja, não há necessidade imediata de todo o bloco que está sendo trazido da memória principal. Porém, essas políticas só fazem sentido quando a memória *cache* está trabalhando com blocos de tamanho muito grande e houver grande contenção ao trazer todo bloco de uma só vez.

Além disso, a memória *cache* deverá continuar fornecendo os dados ao processador enquanto o bloco termina de ser recebido.

Assim, enquanto os mecanismos de busca apenas trazem os dados após uma falta ocorrida, os mecanismos de pré-busca tentam trazer os dados antes mesmo deles serem solicitados pelo processador. Tanto instruções quanto dados podem sofrer pré-busca. Assim, o mecanismo de pré-busca básico funciona durante a falta de um dado, buscando,

além do dado requisitado, os dados próximos ao endereço de memória requisitado, contribuindo dessa forma para a redução da taxa de faltas, impulsionado pelo princípio da localidade espacial.

Outra forma de trabalho de mecanismos de pré-busca é sem a necessidade da ocorrência de faltas, sendo que o mecanismo deverá ser capaz de prever os próximos dados necessários e trazê-los da hierarquia de nível mais baixo da memória.

Mesmo com todas as vantagens do uso da pré-busca, esse mecanismo só faz sentido se o processador puder continuar trabalhando enquanto a pré-busca ocorre em paralelo. Além disso, as memórias *cache* devem prosseguir fornecendo dados ao processador enquanto novos dados são trazidos, necessitando assim de memórias *cache* não bloqueantes.

A.2.5 Política de Substituição de Dados

Durante o processamento, quando ocorrer uma falta de dados, ao buscar o bloco de dados para a memória *cache*, esta deverá escolher qual será a posição onde o bloco será gravado, ou seja, qual bloco de dados será substituído. Essa escolha é chamada de política de substituição de dados.

Quando utiliza-se modo de mapeamento direto em uma memória *cache*, ao trazer novos dados não existe escolha a ser feita para a substituição (STALLINGS, 1996), devendo apenas gravar o novo dado no endereço referente. Porém, quando utiliza-se modos de mapeamento completamente associativo ou associativo por conjuntos, existe uma escolha a ser feita sobre qual bloco será substituído para dar lugar ao novo bloco de dados. Existem quatro estratégias principais para política de substituição de blocos:

- Aleatória – Nessa política o bloco a ser substituído será encontrado ao acaso, onde o sistema irá escolher o bloco alvo através de uma escolha pseudo-aleatória.

- Menos Recentemente Usado – Também conhecida como política LRU (*Least Recently Used*), essa política baseia-se em reduzir a chance de descarte dos blocos usados mais recentemente. Desta forma, esta informação sobre quão recentemente cada bloco foi utilizado deverá permanecer junto a cada bloco para a decisão. Desta maneira, pode-se dizer que essa política prevê o futuro baseando-se no passado, e também que esta política baseia-se no princípio de localidade temporal.

- Menos Freqüentemente Usado – Conhecida como política LFU (*Least Frequently Used*), essa política substitui os blocos que foram menos referenciados. Essa informação sobre quão freqüente cada bloco foi referenciado pode permanecer junto a cada bloco para a decisão.

- Primeiro a Entrar, Primeiro a Sair – Como pode ser difícil o cálculo do bloco que não é utilizado por um tempo mais longo, esta política também conhecida como FIFO (*First In, First Out*) baseia-se apenas nas informações sobre os dados mais antigos para achar o bloco alvo a ser substituído.

Podemos notar que a política LRU tende a ser de melhor desempenho em relação as outras, porém, a abordagem aleatória é a de mais fácil implementação, enquanto a política FIFO apresenta maior simplicidade em relação à política LRU, sendo que essa diferença se acentua na medida em que se aumenta o número de blocos a serem controlados.

A.2.6 Políticas de Escrita de Dados

Quando o processador manda a ordem de gravação de algum dado para memória, o processador não precisa saber se está trabalhando em uma hierarquia de memória ou não. Dessa forma, a instrução de gravação de dados é enviada para o nível hierárquico mais próximo ao processador. Entretanto, a memória *cache*, ao receber dados a serem gravados, pode adotar diferentes tratamentos para essa gravação com relação ao restante da hierarquia de memórias. Basicamente, existem duas opções de gravação em memórias *cache*:

- *Write-Through* – Os dados são gravados na memória *cache* e automaticamente atualizados na memória de nível inferior.
- *Write-Back* – Os dados são gravados apenas na memória *cache* e o bloco modificado será gravado na memória principal somente quando for substituído.

O uso da política *write-back* visa reduzir a quantidade de gravações na memória principal enquanto os dados ainda são úteis ao processador. Porém, para reduzir a quantidade de atualizações a serem realizadas durante a substituição de um bloco, comumente utiliza-se a técnica chamada *bit* sujo. Esse *bit* nada mais é que um *flag* que indica se o dado foi modificado enquanto estava na memória *cache*, ou seja, bloco sujo caso tenha sido modificado ou bloco limpo caso esteja inalterado.

A abordagem *write-back* possui a vantagem de que várias gravações ou alterações em um mesmo bloco, só resultam em uma escrita para o nível inferior. Dessa maneira, essa política de escrita utiliza menos largura de banda, além de poupar energias de gravações e utilização de barramento.

Já a abordagem *write-through* é de mais simples implementação, uma vez que não são necessários indicadores de blocos sujos, e além disso, diferentemente do *write-back*, uma falta de leitura nunca irá resultar também em uma escrita no nível mais baixo da hierarquia. Como a utilização de política *write-through* mantém os blocos atualizados em toda hierarquia, essa política torna o controle de coerência de dados em multiprocessadores mais simples.

Quando o processador tem que esperar as gravações se completarem em níveis mais baixos da hierarquia de memória, dizemos que o processador sofreu parada para gravação. Dessa maneira, uma otimização comum para reduzir os tempos de parada por escrita de dados é a utilização de um *buffer* de gravações.

O *buffer* de escrita (*write buffer*) permite ao processador continuar trabalhando enquanto o *buffer* fica responsável por fazer a atualização da memória em paralelo. Mesmo com a utilização de *buffer* de escrita, a latência da memória e a largura de banda da interconexão podem fazer o processador parar durante as escritas. Nesse caso, a alternativa mais simples é aumentar o tamanho do *buffer* de escritas.

A.2.7 Políticas para Faltas de Escrita de Dados

Durante uma falta de escrita o projeto da arquitetura deve definir a estratégia para tratar essa falta. Duas políticas mais conhecidas para tratar as faltas de escrita são a *write allocate* e a *no write allocate* (PRABHU, 2008). Uma descrição das políticas segue abaixo:

- *Write-Allocate* – O bloco é carregado em uma falta de escrita, seguido de uma ação de escrita sem faltas.

- *No Write-Allocate* – O bloco é modificado na memória principal sem que esse seja carregado para a memória *cache*.

Notamos que, para gravação sem alocação, os blocos só entram na memória *cache* no caso de leituras, uma vez que gravações não geram alocações de blocos na memória *cache*.

Porém, mesmo que ambas políticas possam ser utilizadas em conjunto com a política *write-through* ou *write-back*, geralmente a política *write-back* é utilizada em conjunto com *write-allocate*, esperando que escritas subseqüentes para determinado bloco serão realizados na *cache*. Já a política *write-through* costuma utilizar a política *no write-allocate* uma vez que escritas subseqüentes a determinado bloco vão continuar tendo que acessar a memória principal.

Uma descrição das combinações de política de escrita com política de faltas de escrita segue abaixo:

- *Write Through* com *Write Allocate*:

  Durante uma escrita com sucesso, o bloco será gravado na memória *cache* e memória principal. Durante uma falta de escrita, o bloco será atualizado na memória principal e uma cópia será trazida para a memória *cache*;

  Essa combinação de políticas pode não ser muito eficiente em escritas subseqüentes, uma vez que, mesmo trazendo o bloco para a memória *cache*, durante a próxima escrita de dados a gravação será de qualquer maneira feita para a memória principal, seguindo a política *write-through*. Entretanto, se houver uma leitura ao endereço que sofreu escrita previamente, essa combinação de políticas pode garantir que o dado será acessado mais rapidamente.

- *Write Through* com *No Write Allocate*:

  Durante uma escrita, o bloco será gravado na memória *cache* e na memória principal. Durante faltas de escrita, o bloco será atualizado na memória principal sem que o mesmo seja trazido para memória *cache*;

  Escritas subseqüentes em um dado bloco serão feitas sempre na memória principal devido a política *write-through*. Desta maneira, não haverá latência extra durante uma falta de escrita, tendo que trazer blocos para a memória *cache*, uma vez que esse procedimento é desnecessário de qualquer forma. Entretanto, durante uma leitura após escrita, os dados deverão ser buscados da memória principal, podendo causar perdas de desempenho.

- *Write Back* com *Write Allocate*:

  Durante uma escrita, o bloco será atualizado na memória *cache* e o *bit* sujo será marcado sem que a atualização seja feita na memória principal. Durante faltas, o bloco é atualizado na memória principal e o bloco é trazido para memória *cache*;

  Para escritas subseqüentes ao mesmo bloco, se a primeira escrita causar falta, o bloco será atualizado e trazido para memória *cache*. Desta maneira, futuras escritas não irão causar faltas, sendo necessário apenas atualizar o bloco na memória *cache* e definir o *bit* sujo para ativo, o que elimina acessos extras na memória principal de uma forma bem eficiente se comparado às políticas *write through* com *write allocate*. Para leituras após escrita de dados, essa combinação de políticas também

se apresenta como bastante atrativa, uma vez que as chances do bloco estar presente na memória *cache* é grande.

- *Write Back* com *No Write Allocate*:

  Durante escritas o bloco será gravado na memória *cache* e o *bit* sujo será ativo e a memória principal não será atualizada. Durante faltas de escrita, o bloco será atualizado na memória principal sem trazer o bloco para a memória *cache*;

  Nessa combinação de políticas, para escritas subseqüentes ao mesmo bloco, se a primeira escrita causar falta, vão ser geradas faltas para todas as demais escritas subseqüentes, gerando uma execução ineficiente. Além disso, para leitura após uma escrita com falta de dados deverá causar nova falta de dados.

A.3 Memórias *Cache* Compartilhadas e Distribuídas

Durante anos, a adoção de processamento paralelo e distribuído (CHAUDHRY et al., 2005) tornou a utilização paralela de aglomerados de máquinas no principal método para obtenção de processamento de alto desempenho. Porém, com os avanços das tecnologias de integração de circuitos, o futuro aponta para *chips* com múltiplos núcleos de processamento, os chamados CMP (*Chip Multiprocessor*).

Nas áreas de arquitetura e organização de processadores, já são claras as mudanças desencadeadas pelo crescente nível de integração de circuitos, onde, para os atuais e futuros projetos de computadores, os projetistas podem contar com um maior número de recursos físicos em *chip*, dando liberdade para ousados projetos (SINHAROY et al., 2005) (KONGETIRA; AINGARAN; OLUKOTUN, 2005) (INTEL CORPORATION, 2007a) de processadores com inúmeros núcleos de processamento dentro de um único *chip*.

Porém, com essa mudança de paradigma, passando de processadores monoprocessados para multiprocessados, diversos projetos de processadores passaram a contar com diversos núcleos de processamento (OLUKOTUN et al., 1996) de tamanho reduzido. Nesse novo contexto os projetistas começaram a interconectar esses núcleos a uma única memória através de um barramento. Esse esquema de organização é chamado de memória compartilhada simétrica, porque todos os núcleos tem o mesmo relacionamento com a memória compartilhada única, ou seja todos núcleos possuem o mesmo tempo de acesso a memória, desconsiderados os conflitos de acesso ao barramento e os conflitos de acesso à memória.

Devido a esse modelo de memória compartilhada entre vários núcleos, algumas preocupações foram inseridas durante o projeto de uma memória *cache* para os novos processadores com múltiplos núcleos de processamento. Algumas das questões que ganharam mais força nesse novo modelo de memória foram o controle de coerência entre memórias *cache*, estrutura de interconexão, portas de entrada e saída para as memórias, entre outras.

A.3.1 Coerência e Consistência da Memória *Cache*

Ao tratar de coerência de dados em memórias *cache*, estamos envolvendo dois aspectos importantes. O primeiro diz respeito à coerência, que define quais valores devem ser retornados durante a leitura. O segundo aspecto é a consistência, que determina quando um valor gravado será retornado por uma leitura.

Assim, de acordo com (HENNESSY; PATTERSON, 2007), um sistema de memória é considerado consistente se:

- A ordem do programa for preservada. Uma leitura pelo processador P_i em uma posição X após uma gravação por P_j em X, sem a ocorrência de gravações em X por outro processador entre a gravação e leitura por P_i, sempre retorna o valor gravado por P_j.

- Existe uma visão coerente da memória. Uma dada leitura da posição X pelo processador P_i após uma gravação em X por P_j retorna o valor gravado por P_j se a leitura e a gravação estiverem bastante separadas no tempo e não ocorrer nenhuma outra gravação em X entre os dois acessos.

- Gravações na mesma posição são serializadas. Duas gravações na mesma posição por dois processadores quaisquer serão vistas na mesma ordem por todos os processadores.

Mesmo que o modelo ideal seja que, após um processador gravar um dado na memória, esse seja atualizado instantaneamente em todas as suas cópias, esse modelo talvez seja impossível de ser implementado. A noção de quando um dado gravado será visível para leituras é tratada pela consistência da memória.

Assim, podemos notar que a noção de coerência e consistência são complementares, pois enquanto a coerência define o comportamento de leituras e gravações em uma mesma posição, a consistência define o comportamento de leituras e gravações em relação a acessos a outras posições de memória.

A.3.1.1 Protocolos de Coerência

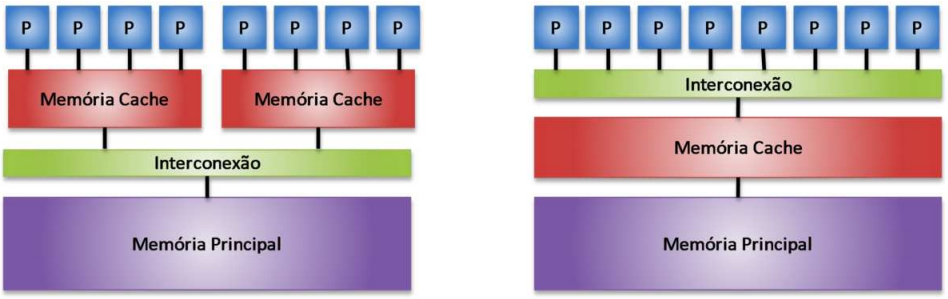
Em um multiprocessador coerente, as memórias *cache* fornecem tanto a migração quanto a replicação de dados compartilhados. Dessa forma, mesmo que existam diversos processadores trabalhando sobre os mesmos dados, cada processador deverá agir sem se preocupar com a existência de outras cópias dos mesmos dados em outros processadores.

As memórias *cache* coerentes proporcionam migração, pois qualquer dado pode ser movido para a *cache* e operado de forma transparente. As memórias *cache* também proporcionam replicação, pois diversas memórias *cache* podem conter cópias de um dado referentes ao mesmo endereço.

É importante ressaltar que, de acordo com (ROSE; NAVAUX, 2003), o problema de coerência de dados apresenta-se apenas em arquiteturas multiprocessadas que associam memórias *cache* a cada um dos processadores do sistema. Assim, em arquiteturas onde a memória *cache* está associada somente à memória principal, não existem problemas de coerência de memória *cache*. Podemos ver um diagrama da arquitetura com memórias *cache* associadas em blocos ou associada somente à memória principal na Figura A.5, onde pode-se observar duas arquiteturas de memória *cache* com problema de coerência de dados A.5(a) e outra sem problemas de coerência de dados A.5(b) (ROSE; NAVAUX, 2003).

Para que uma memória *cache* seja capaz de manter a coerência de dados, é necessário a adoção de um protocolo de coerência para assegurar o controle de compartilhamento de qualquer bloco de dados entre várias memórias *cache* e processadores. Existem duas classes de protocolos atualmente em uso para assegurar a coerência de memórias *cache*:

- Baseado em Diretório – O status sobre o compartilhamento de um bloco de memória é alocado apenas em uma posição chamada diretório.



(a) Com problemas de coerência. (b) Sem problemas de coerência.

Figura A.5: Arquiteturas de memória *cache* com problema de coerência de dados A.5(a) e sem problemas de coerência de dados A.5(b), adaptado de (ROSE; NAVAUX, 2003).

- *Snooping* – O status sobre o compartilhamento de um bloco é replicado em todas as memórias *cache* que possuem a cópia de cada dado. Assim, não existe nenhum diretório centralizador. Geralmente todas as memórias *cache* encontram-se em um mesmo barramento e todos espionam (*snoop*) a movimentação no barramento, no caso de alguma outra memória possuir cópia de blocos solicitados.

Como em diversos projetos de multiprocessadores as memórias *cache* estão conectadas a uma única memória principal interconectadas por um barramento, os protocolos de coerência de memória *cache* mais populares são os baseados na técnica de *snooping*, pois tiram proveito da infra-estrutura pré-existente. Como exemplo desse tipo de protocolo pode-se citar o protocolo MESI (STALLINGS, 1996), onde o nome MESI vêm das inciais dos estados possíveis de um dado durante operação (*modified, exclusive, shared*, e *invalid*).

Os pontos-chave (CULLER; SINGH, 1999) de um barramento que possui suporte à coerência são que todas as transações devem ser visíveis à todos os controladores de memória *cache*. Assim, cabe ao protocolo de coerência garantir que todas as transações de memória apareçam no barramento e que todos os controladores façam as ações necessárias ao verem transações relevantes.

A.3.1.2 Modelos de Consistência

Os modelos de consistência de memória *cache* tratam o grau de consistência que deverá existir no sistema, apresentando soluções de quando um valor atualizado em uma memória *cache* qualquer deverá ser visível aos outros processadores.

Embora a questão sobre a consistência de dados na memória *cache* pareça trivial, é de um elevado grau de complexidade, pois tratar a consistência de dados significa escolher o momento em que as variáveis devam ser atualizadas nas suas cópias. Porém, essa escolha envolve também o algoritmo executado nos diversos processadores, além de ser um tratamento sobre condição de corrida, uma vez que podem haver disputas por recursos.

O modelo mais simples de consistência de dados é o chamado consistência seqüencial, que é implementado de forma simples, exigindo que um processador retarde a conclusão de qualquer acesso à memória até que todas as invalidações causadas por esse acesso se completem.

Outro tipo de modelo também difundido é o modelo de consistência relaxado (PATTERSON; HENNESSY, 2005), que envolve o programador ao gerar programas paralelos

e consiste em permitir que as leituras e gravações se completem fora de ordem, mas requer o uso de operações de sincronização para impor ordenação, fazendo com que o programa seja sincronizado e comporte-se como se estivesse em um processador mono-processado.

A.3.2 Interconexões

Como os diversos dispositivos de um sistema, processadores, memórias, portas de entrada e saída e demais dispositivos, precisam se comunicar, essa é a tarefa das interconexões. Algumas das principais características de comparação entre interconexões são (ROSE; NAVAUX, 2003):

- Escalabilidade – É uma característica desejável em toda interconexão. Diz respeito a adaptabilidade da interconexão ao aumento da quantidade de dispositivos e também de carga total de trabalho. Assim, um sistema escalável será de fácil expansão mantendo as características principais inalteradas.

- Desempenho – O desempenho desejável de uma interconexão é que essa consiga manipular e dar vazão a todos os dados em tempo hábil. Assim, o desempenho indica a capacidade e a velocidade da transferência de dados pela interconexão.

- Custo – O custo de uma interconexão costuma variar proporcionalmente em função do número de dispositivos interconectados e a capacidade de vazão e latência. Em alguns casos, o custo de uma interconexão pode se referir a área ocupada pelo projeto ou pela potência consumida.

- Confiabilidade – Pode ser tratada como a probabilidade da interconexão atender, de forma adequada, aos dispositivos comunicantes. Assim, a existência de caminhos alternativos ou redundantes entre dispositivos aumenta a confiabilidade da interconexão.

- Funcionalidade – Diz respeito às funcionalidades específicas da interconexão agregadas à transferência de dados. Assim, uma interconexão pode implementar outros serviços como *buffers* de entrada e/ou saída, garantia de ordenação na transferência, ou até mesmo roteamento automático.

- Reusabilidade – Trata da capacidade da interconexão se conectar a diferentes tipos de dispositivos, e além disso, a cada nova geração o projeto poder ser em grande parte reutilizado.

Uma interconexão pode ser feita de diversas formas e mesmo assim, durante anos, a solução adotada tem sido a interconexão por barramento. Porém, atualmente existem diversas formas de interconexões que estão sendo estudadas afim de serem implementadas nos processadores multiprocessados, que é o caso das interconexões por matrizes de chaveamento, ou por redes intra-*chip*, também conhecidas como NoC (*Network-on-Chip*) (BJERREGAARD; MAHADEVAN, 2006).

A.3.2.1 Barramentos

Um barramento é um canal de comunicação compartilhado que utiliza um conjunto de fios para conectar diversos dispositivos ou subsistemas. As principais vantagens dessa interconexão é a escalabilidade, reusabilidade e o baixo custo.

A escalabilidade do barramento é considerada a partir do ponto de vista de que novos dispositivos podem ser agregados facilmente sem grandes modificações, porém, o barramento não é totalmente escalável pois o mesmo não se adapta para aumentar a vazão de dados ao adicionar um novo componente interconectado.

Quanto a reusabilidade, como o projeto de barramentos muitas vezes segue a determinados padrões, esses podem ser reusados a cada nova geração, com pouca ou nenhuma modificação. Como o projeto de um barramento pode ser visto de forma simplista como uma forma de compartilhamento de um conjunto de fios, essa interconexão é de baixo custo.

A forma clássica de um barramento apresenta dois conjuntos de linhas de comunicação, um conjunto de controle e outro conjunto de linhas de dados como é ilustrado na Figura A.6. As linhas de controle são usadas para gerenciar as requisições e confirmações, além de informar o tipo dos dados que estão trafegando nas linhas de dados. As linhas de dados são apenas linhas de propagação de informações entre origem e destino.

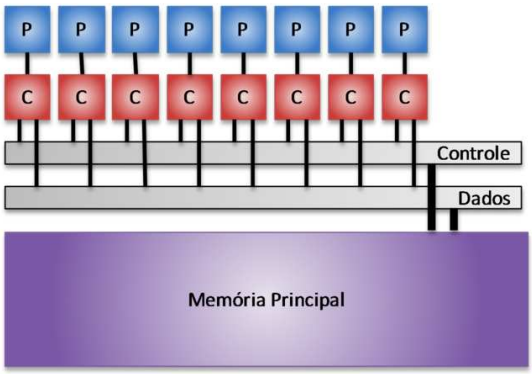


Figura A.6: Diagrama de um barramento com canais de controle e dados interligando processadores e memória *cache* a memória principal.

Como diversos componentes estão compartilhando um mesmo meio físico do barramento, o protocolo do barramento é contido nas linhas de controle, as quais implementam qual será a política de uso do barramento. O barramento também pode ser classificado como rede dinâmica de interconexão, uma vez que a topologia de comunicação não existe *a priori*.

A.3.2.2 Matrizes de Chaveamento

A matriz de chaveamento, também conhecida como *crossbar* ou *crossbar switch* possui um custo elevado, porém, uma boa escalabilidade. Assim, embora uma rede de interconexão possa ocupar grandes áreas do projeto para interligar um grande número de dispositivos, essa interconexão se mantém com uma boa escalibilidade uma vez que ao adicionar novas portas para dispositivos, aumenta-se também a vazão da interconexão total.

Em uma rede de chaveamento pode se interconectar dois dispositivos quaisquer, desde que esses não se encontrem já ocupados. Uma alternativa para o alto custo é a utilização de várias matrizes de chaveamento em formação hierárquica. A Figura A.7 apresenta uma ilustração de uso de uma chave *crossbar* interconectando memórias *cache* de primeiro nível de diversos processadores com memórias *cache* L2 compartilhadas, cada um

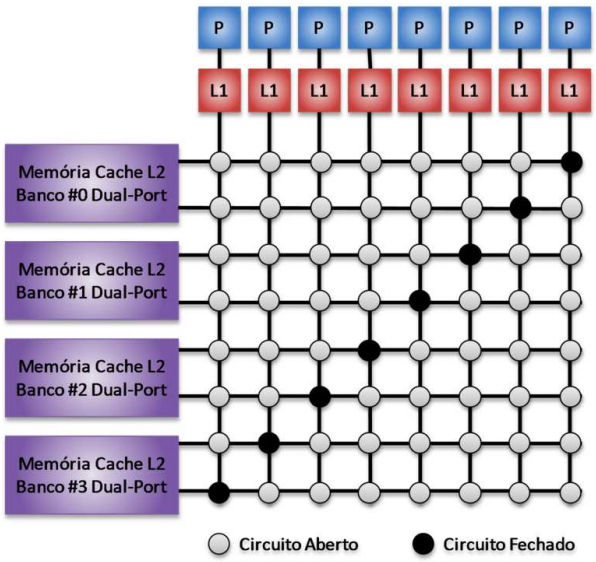


Figura A.7: Diagrama de uma matriz de chaveamento interconectando processadores e memórias *cache*.

possuindo duas portas de comunicação. Dessa maneira, apenas dois núcleos podem ser conectados a cada banco de memória *cache* simultaneamente.

A.3.2.3 Redes Intra-Chip

Com a promessa de rápido aumento do número de núcleos de processamento dentro do processador, aumenta a motivação de estudos sobre formas diferentes de interconexão entre os vários dispositivos dentro do *chip*.

Com o intuito principal de aumentar a escalabilidade e reduzir o custo da interconexão, diversos estudos abordam as redes de interconexão intra-*chip*. Em muitos casos, essa rede de interconexão é formada apenas por diversos roteadores, um em cada dispositivo, interligados. Dessa forma, podem não existir ligações diretas entre todos os dispositivos, sendo necessário então que um pacote trafegue entre os roteadores para chegar ao seu destino.

A política de roteamento da interconexão é a que determina como os pacotes serão chaveados para chegar ao destino. Assim, a forma de chaveamento depende da topologia da rede de interconexão.

A Figura A.8 apresenta um diagrama de utilização de uma rede de interconexão intra-*chip* de tamanho 5x4 (20 roteadores), conectando diversos núcleos de processamento à quatro bancos de memória *cache*. No caso ilustrado, o tempo de comunicação entre núcleo e memória *cache* pode variar dependendo da localização do núcleo na rede de interconexão, onde os núcleos mais próximos irão conseguir acessar a memória *cache* com apenas dois saltos (*hops*), ou seja, passando apenas pelo roteador local e roteador da memória *cache*, enquanto que no pior caso, a comunicação poderá custar de 5 até 8 saltos, dependendo do banco de memória que o núcleo precisar acessar.

A.3.3 Compartilhamento de Memórias *Cache*

Devido ao aumento do número de núcleos de processamento dentro do processador, novos modos de organização da hierarquia de memória estão sendo estudados. As me-

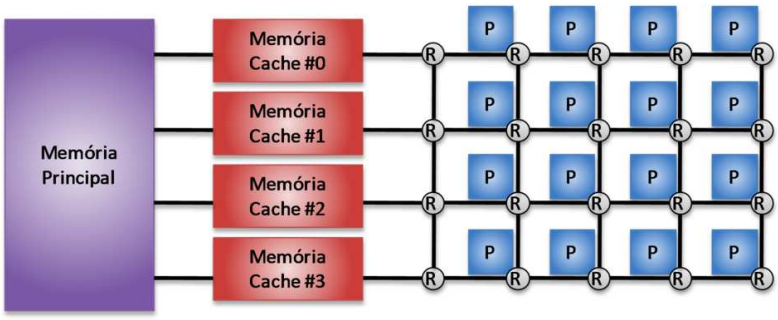
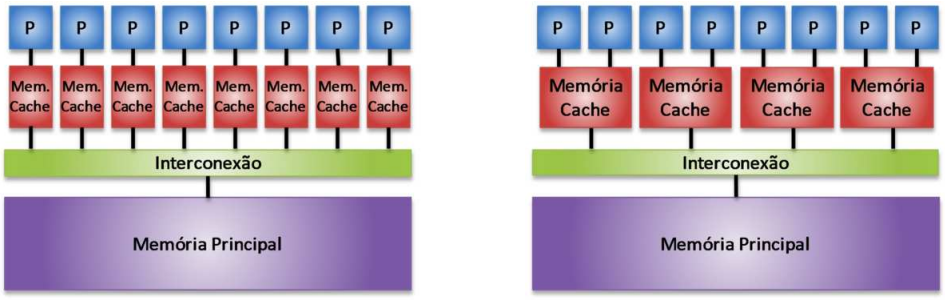


Figura A.8: Diagrama de uma rede de interconexão intra-*chip* (NoC) interligando diversos núcleos de processamento.

mórias *cache* para esses processadores podem ser privadas ou compartilhadas conforme apresenta a Figura A.9.

Em um modelo de organização de memória *cache* privada cada processador possui sua própria hierarquia de memória *cache* totalmente isolada, como apresentado na Figura A.9(a), somente interconectada aos demais núcleos por meio da memória principal.

No modelo de memória *cache* compartilhada, ilustrado na Figura A.9(b), existe pelo menos um nível na hierarquia de memórias *cache* compartilhada entre dois ou mais núcleos de processamento. No caso ilustrado, a memória *cache* está compartilhada a cada dois núcleos de processamento.



(a) Memórias *cache* privadas. (b) Memórias *cache* compartilhada.

Figura A.9: Diagrama representado organizações de memórias *cache*, apresentado memórias *cache* privadas A.9(a) e compartilhadas A.9(b).

A utilização de modelos de memórias *cache* privadas ou compartilhadas influenciam no projeto do sistema, uma vez que questões sobre a implementação do protocolo de coerência e consistência entre as memórias, além da quantidade de portas de entrada e saída necessárias na memória *cache*, largura de banda de interconexão e área do projeto, são afetados pelo modelo de organização privada ou compartilhada.

O desempenho do sistema também pode ser influenciado pelo modo adotado de compartilhamento de memória *cache*, uma vez que o compartilhamento poderá favorecer o desempenho reduzindo o número de faltas quando os processadores que compartilham estiverem trabalhando no mesmo conjunto de dados. Por outro lado, a quantidade de faltas relacionadas ao conflito de endereços e capacidade podem aumentar se os dados não forem os mesmos em uso nos processadores que compartilham a memória *cache*.

APÊNDICE B RESULTADOS ADICIONAIS

" All difficult things have their origin in that which is easy,
and great things in that which is small. "
— LAO TZU

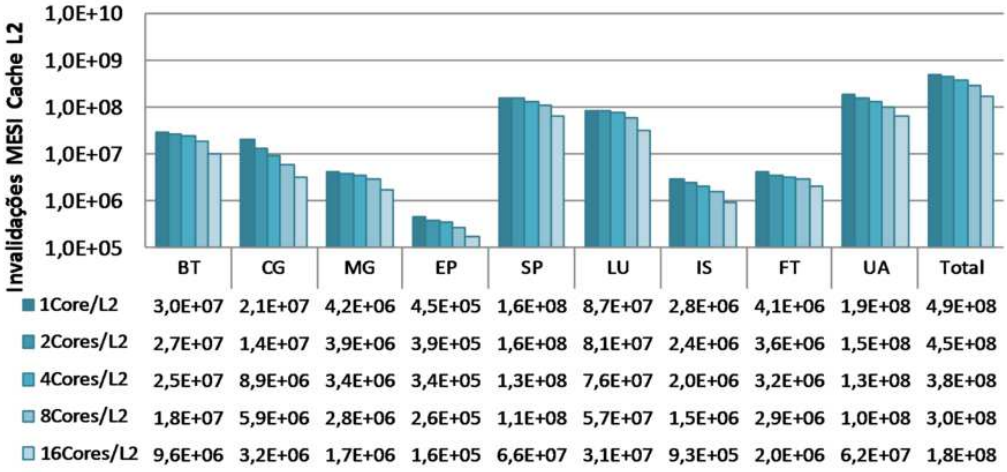
Nesse apêndice, são apresentados resultados e análises complementares sobre o impacto das memórias *cache* no desempenho final de sistemas multiprocessados para os diversos experimentos apresentados no Capítulo 4.

Os resultados complementares estão divididos por experimentos, onde a seção B.1 diz respeito ao experimento base sobre a influência do compartilhamento da memória *cache* L2 entre diversos núcleos de processamento. Na seção 4.2 será visto os resultados adicionais do segundo experimento, o qual diz respeito à influência do tamanho da memória *cache* no compartilhamento da mesma. A seção B.3 apresentará os resultados adicionais sobre avaliação do aumento da associatividade em *multi-core* com memórias compartilhadas. Na seção B.4 trará os resultados adicionais abordando a influência do tamanho da linha da memória *cache* para diversas organizações de memória. Por fim, a seção B.5 trará resultados adicionais a respeito da influência da hierarquia de memória *cache*, avaliando a inserção de mais um nível na hierarquia de memória *cache*.

B.1 Experimento 1 - Compartilhamento da Memória *Cache*

As arquiteturas modeladas apresentam um número variado de bancos de memória *cache* L2 de acordo com o compartilhamento adotado. Assim, para todas as organizações, exceto as que utilizem apenas uma memória *cache* L2 compartilhada por todos os núcleos (32Cores/L2), deverá existir um protocolo de coerência de dados entre esses bancos. O protocolo de coerência MESI baseado em *snooper* foi o protocolo modelado em todos os experimentos. O gráfico B.1 apresenta a quantidade de invalidações de dados entre as memórias *cache* L2. A quantidade de invalidações é uma forma de indicar os ganhos devido ao compartilhamento de dados conforme as organizações forem mudando. Podemos então ver que os ganhos podem variar de aplicação para aplicação, ficando claro quando comparamos a aplicação EP (menor número de invalidações) com as demais aplicações. A aplicação UA apresentou o maior número de invalidações, demonstrando a intensidade do uso e compartilhamento de dados dessa aplicação.

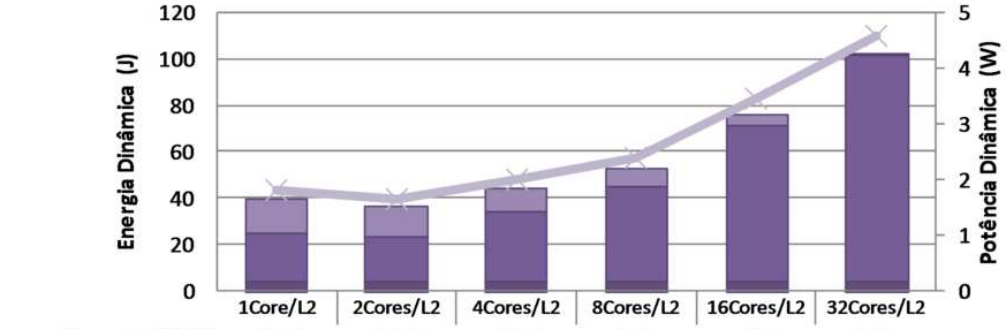
O consumo de energia e potência dinâmica do sistema de memória das diversas organizações avaliadas é apresentado na Figura B.2, onde pode ser visto o aumento de consumo conforme os bancos de memória *cache* ficaram maiores, sendo que a organização



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	3,0E+07	2,1E+07	4,2E+06	4,5E+05	1,6E+08	8,7E+07	2,8E+06	4,1E+06	1,9E+08	4,9E+08
2Cores/L2	2,7E+07	1,4E+07	3,9E+06	3,9E+05	1,6E+08	8,1E+07	2,4E+06	3,6E+06	1,5E+08	4,5E+08
4Cores/L2	2,5E+07	8,9E+06	3,4E+06	3,4E+05	1,3E+08	7,6E+07	2,0E+06	3,2E+06	1,3E+08	3,8E+08
8Cores/L2	1,8E+07	5,9E+06	2,8E+06	2,6E+05	1,1E+08	5,7E+07	1,5E+06	2,9E+06	1,0E+08	3,0E+08
16Cores/L2	9,6E+06	3,2E+06	1,7E+06	1,6E+05	6,6E+07	3,1E+07	9,3E+05	2,0E+06	6,2E+07	1,8E+08

Figura B.1: Invalidações de dados na memória *cache* L2 do primeiro experimento.

de menor consumo foi a 2Cores/L2 e a de maior consumo foi a organização 32Cores/L2. Podemos ver que ao reduzir a quantidade de faltas na memória *cache* L2, aumentando o compartilhamento da mesma, reduzimos o consumo de potência dinâmica da memória principal. Em contrapartida, o consumo de potência dinâmica em blocos grandes de memória *cache* é maior que em blocos pequenos, assim, a redução no consumo dinâmico das memórias principais não cobriram o aumento na memória *cache* L2.



	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Energia RAM (J)	14,59	12,60	10,13	7,69	4,74	0,68
Energia L2 (J)	20,89	19,47	29,68	41,15	67,46	97,84
Energia L1 (J)	4,64	4,60	4,50	4,40	4,24	3,99
Potência (W)	1,83	1,66	2,03	2,41	3,48	4,59

Figura B.2: Consumo de energia e potência dinâmica do sistema de memória do primeiro experimento.

Nas atuais e futuras tecnologias de integração, o consumo de potência estática é fundamental nas avaliações de potência total. A Figura B.3 apresenta o gráfico contendo os valores de consumo de energia e potência estática para as diferentes organizações de memória. A variação apresentada nesse gráfico é mais suave e não linear, onde o consumo mínimo foi apresentado pela organização 8Cores/L2. O consumo de potência estática da memória *cache* L2 foi predominante para o consumo estático final do sistema de memória.

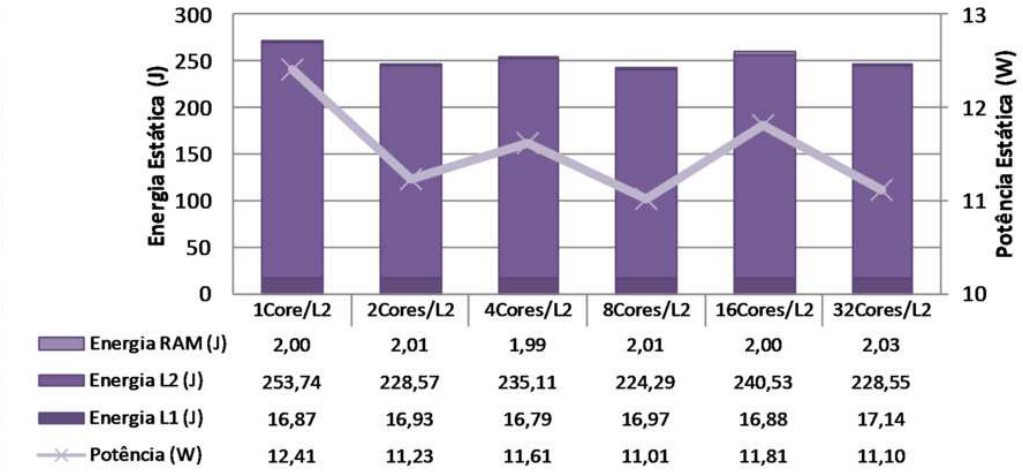


Figura B.3: Consumo de energia e potência estática do sistema de memória do primeiro experimento.

B.2 Experimento 2 - Tamanho da Memória *Cache*

A porcentagem de instruções de acesso à memória por aplicação, de forma geral, permaneceu próxima das porcentagens do primeiro experimento, apresentando pequenas variações como pode ser visto na Figura B.4. Avaliando as porcentagens, também concluímos que não existe clara dependência entre os resultados de desempenho e porcentagem de instruções de acesso a dados executados, uma vez que o compartilhamento de dados ou tipo de aplicação cria maior dependência com os tempos de execução.

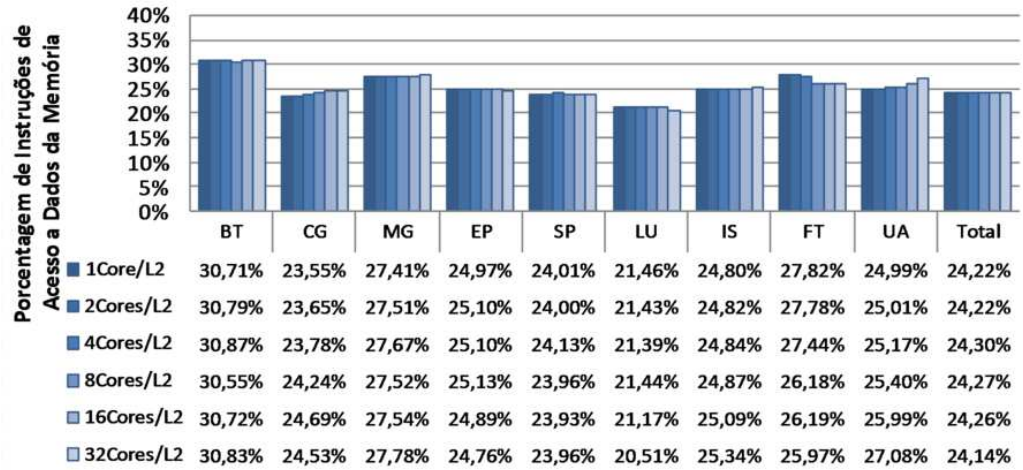


Figura B.4: Instruções de acesso à memória executadas do segundo experimento (1 MB).

Mesmo reduzindo o tamanho total de memória *cache* L2 conforme aumentou-se o compartilhamento dessa, o gráfico da Figura B.5 mostra a redução na quantidade de invalidações de dados conforme maior o compartilhamento da memória *cache* L2. Entretanto, devemos ressaltar que essa quantidade de invalidações pode sofrer redução também quando o dado não está presente em nenhuma outra memória *cache*, ou seja, nesse caso, essa métrica é fortemente influenciada pela quantidade de faltas na memória *cache* L2.

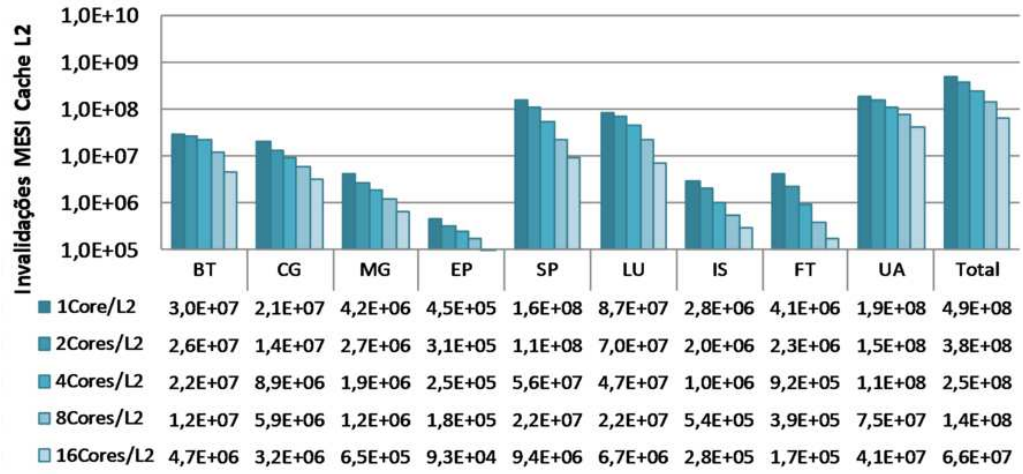


Figura B.5: Invalidações de dados na memória *cache* L2 do segundo experimento (1 MB).

O consumo de energia e potência dinâmica desse experimento é apresentado na Figura B.6. Podemos ver o crescimento no consumo de potência dinâmica devido, principalmente, ao aumento no número de operações na memória principal que foi ocasionado pelo aumento na quantidade de faltas na memória *cache* L2. Quanto às memórias *cache* L1 e L2, houve poucas variações no consumo de potência dinâmica.

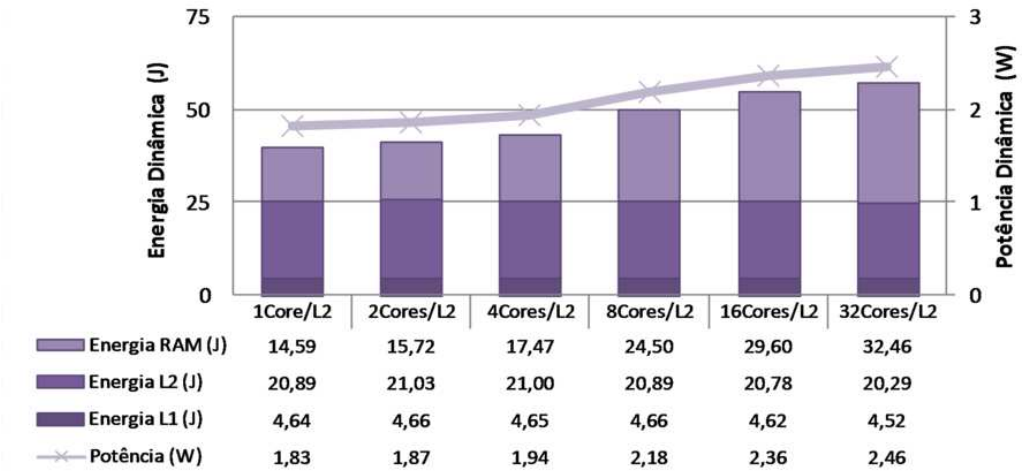


Figura B.6: Consumo de energia e potência dinâmica do sistema de memória do segundo experimento (1 MB).

Considerando a potência estática, houve uma redução à medida que a quantidade total de memória do sistema foi reduzida, como pode ser visto na Figura B.7. Para as memórias *cache* L1 e memória principal, as modificações de consumo de energia variaram apenas por motivos de variação no tempo de execução das aplicações.

B.2.1 2 MB por Memória *Cache* L2

A Figura B.8 mostra a porcentagem de instruções que efetuam acesso a dados para as diferentes aplicações e organizações de memória *cache* neste segundo experimento,

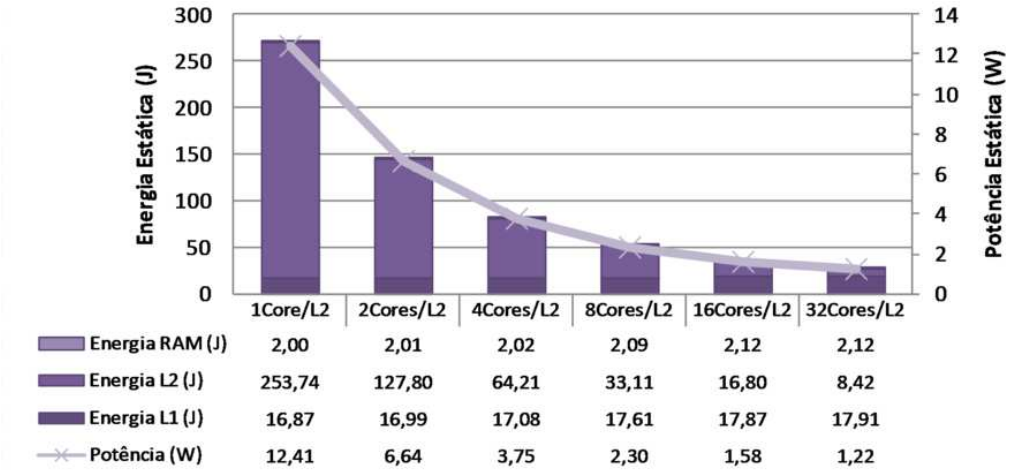


Figura B.7: Consumo de energia e potência estática do sistema de memória do segundo experimento (1 MB).

avaliando que não existe relação clara entre a porcentagem de instruções que acessam dados e o desempenho da aplicação.

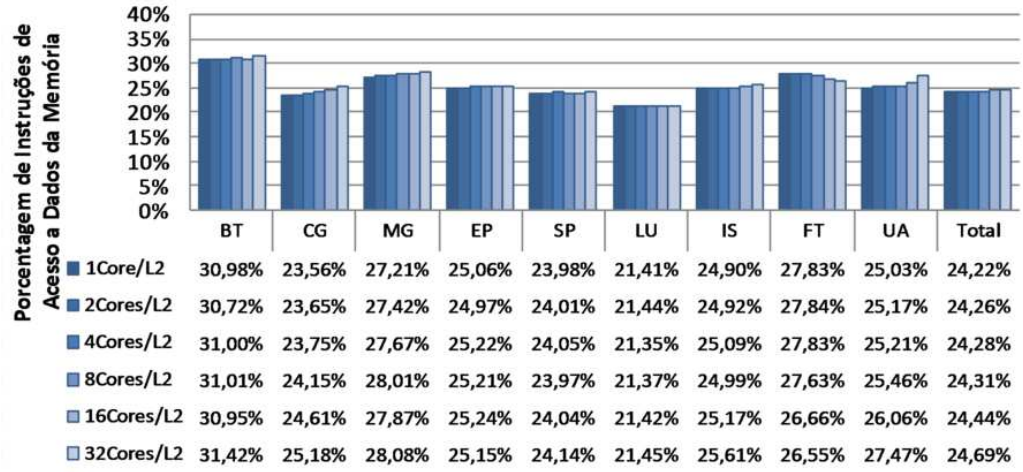


Figura B.8: Instruções de acesso à memória executadas do segundo experimento (2 MB).

Assim como no caso de 1 MB por banco de memória *cache*, houve uma redução no número de invalidações de dados da memória *cache* L2 conforme aumentou-se o compartilhamento dessa memória entre os núcleos de processamento, o que pode ser visto na Figura B.9. Porém, como já foi dito, essa redução pode ser ocasionada por faltas de dados, ou seja, o dado acessado não estava presente em nenhuma outra memória, e também por influência do próprio compartilhamento de memória *cache*.

Para a potência dinâmica, podemos verificar na Figura B.10 que houve um comportamento próximo de consumo entre as organizações 1Core/L2, 2Cores/L2 e 4Cores/L2. Todas organizações consumiram menos que as organizações utilizando apenas 1 MB por banco de memória *cache*, o que foi ocasionado pela redução no consumo dinâmico da memória principal, ou seja, menor número de faltas na memória *cache* L2, resultando em um número menor de operações na memória principal.

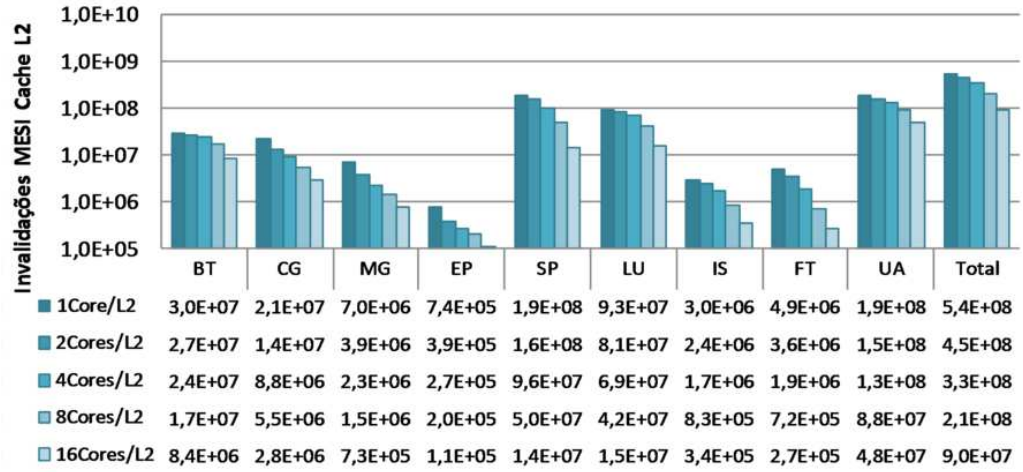


Figura B.9: Invalidações de dados na memória *cache* L2 do segundo experimento (2 MB).

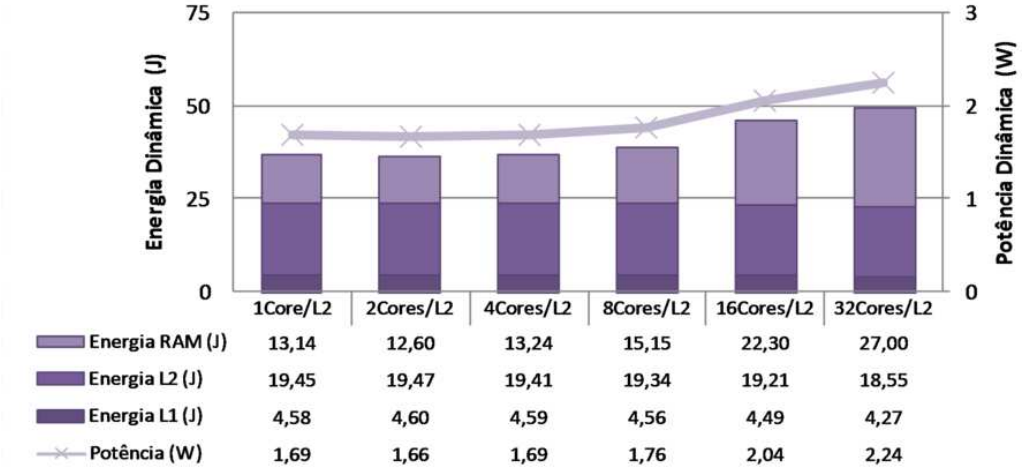


Figura B.10: Consumo de energia e potência dinâmica do sistema de memória do segundo experimento (2 MB).

Para o caso de consumo de energia estática, o comportamento foi bastante parecido com as organizações utilizando 1 MB e 2 MB de memória *cache*. Entretanto, no caso apresentado na Figura B.11, o consumo de energia e potência estática foi maior no caso de 2 MB. Isso foi ocasionado principalmente pelo aumento no consumo estático de potência pela memória *cache* L2 que aumentou de tamanho.

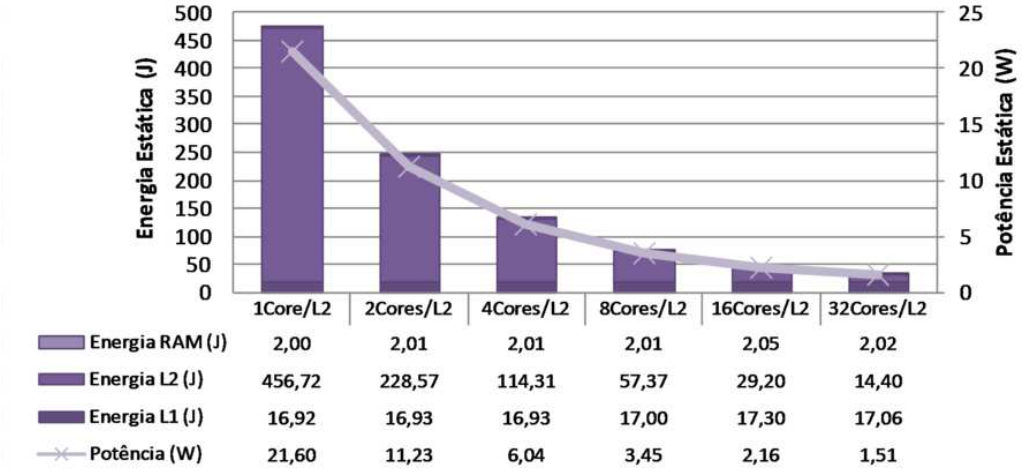


Figura B.11: Consumo de energia e potência estática do sistema de memória do segundo experimento (2 MB).

B.2.2 4 MB por Memória *Cache* L2

A Figura B.12 apresenta a porcentagem de instruções de acesso à memória das diversas aplicações executadas neste experimento. Nota-se que o total de instruções de acesso à memória permanece em um mesmo patamar para as diversas organizações de memória *cache* L2.

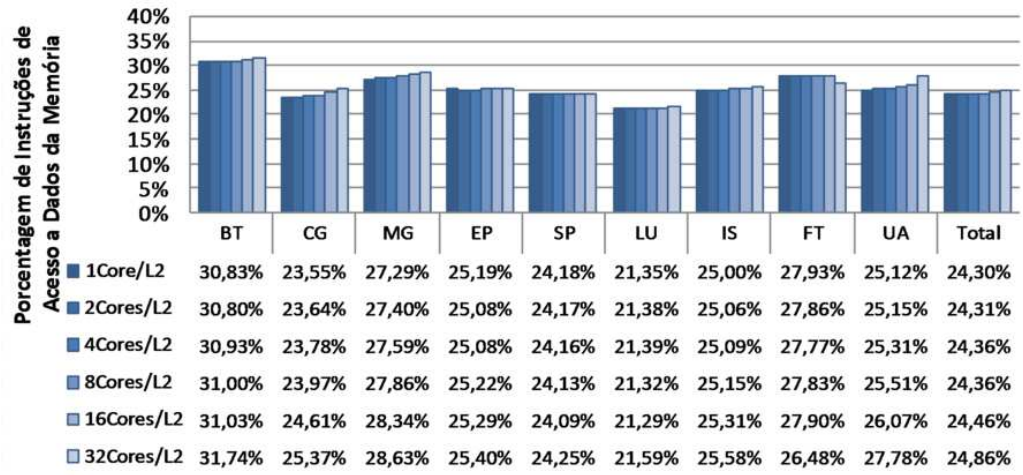
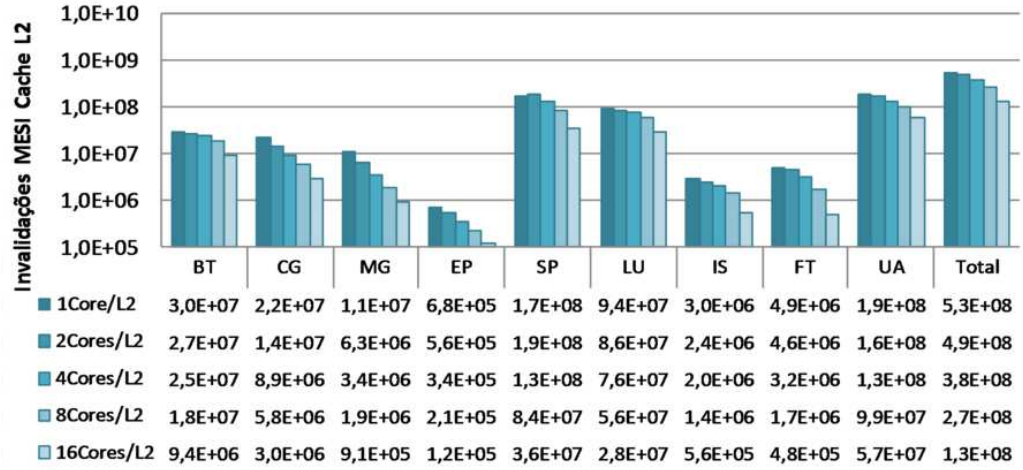


Figura B.12: Instruções de acesso à memória executadas do segundo experimento (4 MB).

Os números de invalidações ocorridas na memória *cache* L2 estão dispostos na Figura B.13, onde podemos ver que a queda de invalidações utilizando 4 MB por banco de

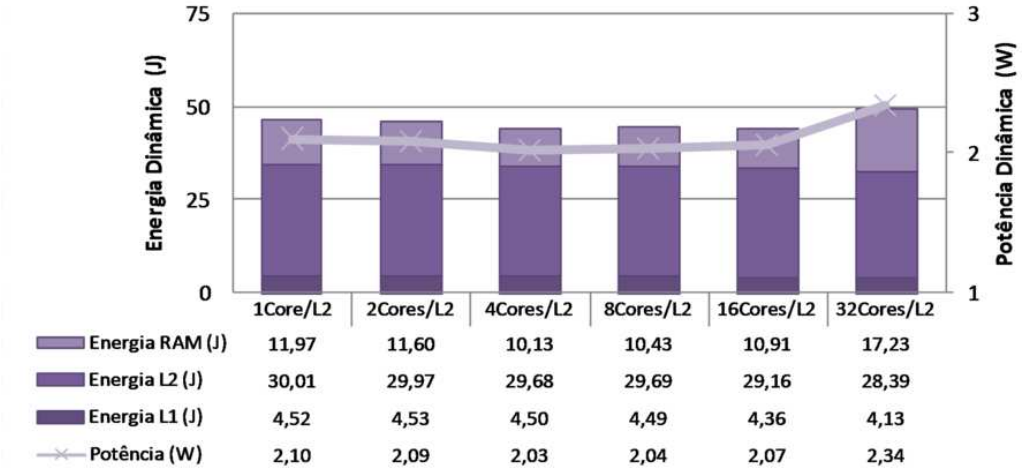
memória *cache* L2 foi menos acentuada que nos casos utilizando 1 MB ou 2 MB, o que confirma a teoria de que parte da redução de invalidações é devida ao aumento do número de faltas de dados nas memórias para que um determinado dado possa ser invalidado.



	BT	CG	MG	EP	SP	LU	IS	FT	UA	Total
1Core/L2	3,0E+07	2,2E+07	1,1E+07	6,8E+05	1,7E+08	9,4E+07	3,0E+06	4,9E+06	1,9E+08	5,3E+08
2Cores/L2	2,7E+07	1,4E+07	6,3E+06	5,6E+05	1,9E+08	8,6E+07	2,4E+06	4,6E+06	1,6E+08	4,9E+08
4Cores/L2	2,5E+07	8,9E+06	3,4E+06	3,4E+05	1,3E+08	7,6E+07	2,0E+06	3,2E+06	1,3E+08	3,8E+08
8Cores/L2	1,8E+07	5,8E+06	1,9E+06	2,1E+05	8,4E+07	5,6E+07	1,4E+06	1,7E+06	9,9E+07	2,7E+08
16Cores/L2	9,4E+06	3,0E+06	9,1E+05	1,2E+05	3,6E+07	2,8E+07	5,6E+05	4,8E+05	5,7E+07	1,3E+08

Figura B.13: Invalidações de dados na memória *cache* L2 do segundo experimento (4 MB).

A variação do consumo de energia e potência dinâmica está apresentada no gráfico da Figura B.14, onde temos uma queda de consumo para as organizações com 2, 4, 8 e 16 núcleos por memória *cache* em relação a primeira configuração (1Core/L2), sendo que os menores valores ficaram para 4Cores/L2 e 8Cores/L2.



	1Core/L2	2Cores/L2	4Cores/L2	8Cores/L2	16Cores/L2	32Cores/L2
Energia RAM (J)	11,97	11,60	10,13	10,43	10,91	17,23
Energia L2 (J)	30,01	29,97	29,68	29,69	29,16	28,39
Energia L1 (J)	4,52	4,53	4,50	4,49	4,36	4,13
Potência (W)	2,10	2,09	2,03	2,04	2,07	2,34

Figura B.14: Consumo de energia e potência dinâmica do sistema de memória do segundo experimento (4 MB).

No caso de energia e potência estática consumida pelo sistema de memória, temos um comportamento similar às configurações com 1 MB e 2 MB por memória *cache* L2, sendo que houve redução desse consumo em todas organizações, partindo da primeira (1Core/L2) até a última (32Cores/L2). O principal responsável pela redução de consumo foi a memória *cache* L2, que teve seu tamanho lógico reduzido em 32 vezes, gerando assim menos consumo de potência de vazamento.

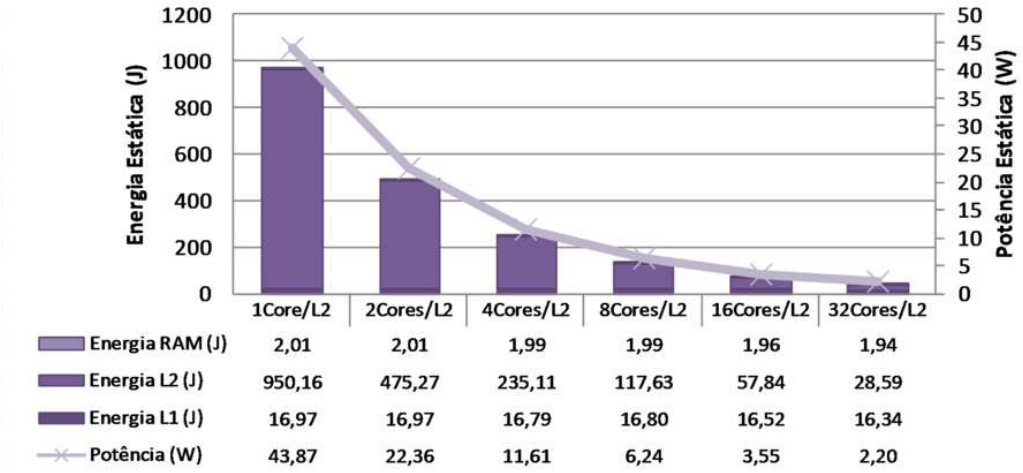


Figura B.15: Consumo de energia e potência estática do sistema de memória do segundo experimento (4 MB).

B.3 Experimento 3 - Associatividade da Memória *Cache*

A porcentagem de instruções de acesso a dados relacionada ao total de instruções executadas neste experimento é ilustrada na Figura B.16. É possível notar que há variação nas porcentagens a medida que a quantidade de instruções varia, porém, como nos demais experimentos, a porcentagem total de instruções de acesso a dados permanece com pouca variação

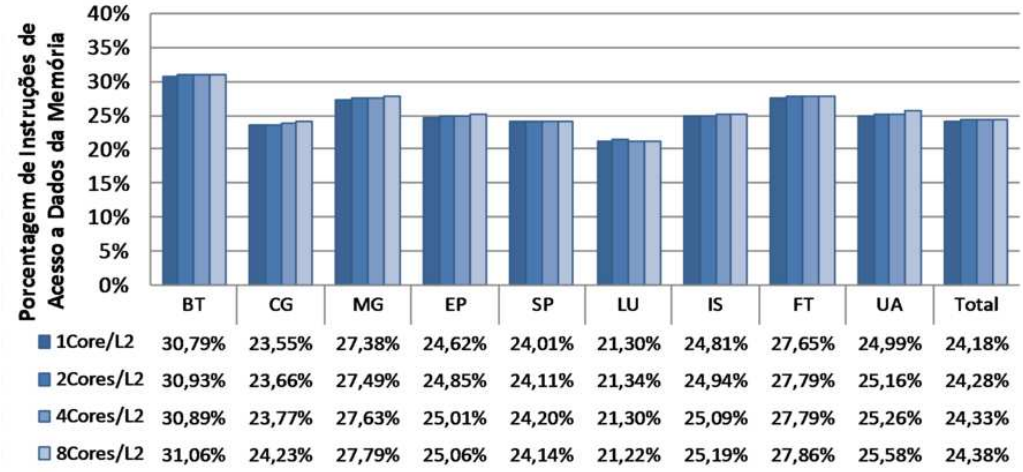


Figura B.16: Instruções de acesso à memória executadas do terceiro experimento.

A quantidade de invalidações geradas pelo protocolo de coerência de dados é apresentado no gráfico da Figura B.17, onde vemos a redução de invalidações conforme aumentamos o compartilhamento de memória entre os núcleos de processamento. Comparado com o experimento base, a quantidade de invalidações permanece em patamar pouco superior, mas com comportamente de redução muito próximo.

Após avaliar os gráficos ligados à execução e métricas da hierarquia de memória, o resultado apresentado na Figura B.18 mostra os valores de consumo de energia e potência

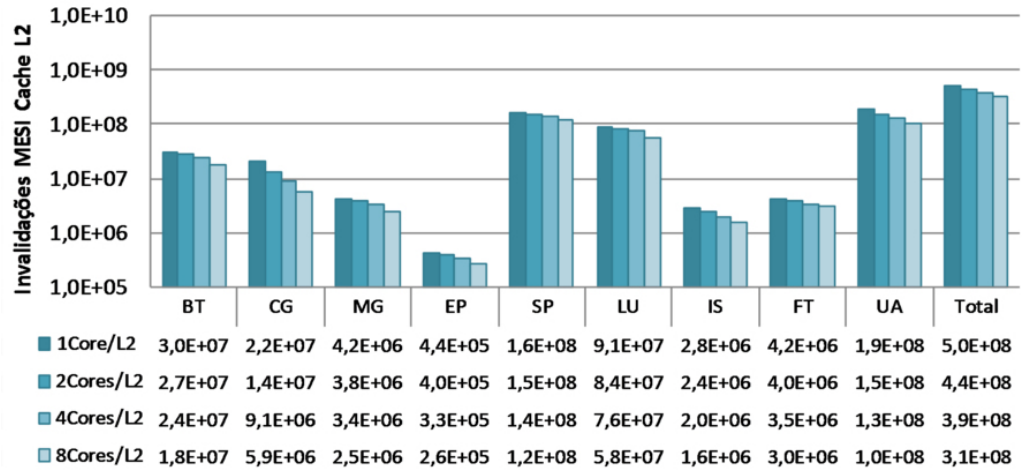


Figura B.17: Invalidações de dados na memória *cache* L2 do terceiro experimento.

dinâmica pelas organizações de memória do terceiro experimento. Notamos um comportamento de redução na potência consumida pela memória principal combinada com o aumento no consumo da memória *cache* L2, sendo que a organização 1Core/L2 apresentou menor consumo de potência dinâmica para este experimento. O consumo de potência desse experimento depende mais da organização interna da memória *cache* modelada, uma vez que o crescimento de potência não é linear.

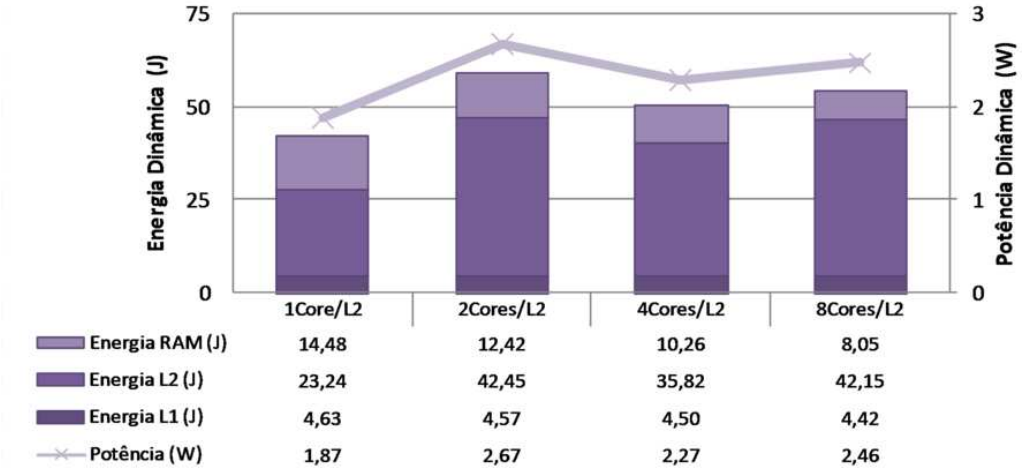


Figura B.18: Consumo de energia e potência dinâmica do sistema de memória do terceiro experimento.

O consumo de energia e potência estático apresentado na Figura B.19 mostra uma redução no consumo para as organizações 4Cores/L2 e 8Cores/L2, geradas principalmente pela redução no consumo estático de potência da memória *cache* L2.

B.4 Experimento 4 - Tamanho da Linha da Memória *Cache*

A porcentagem de instruções de acesso a dados em relação ao total de instruções está ilustrado na Figura B.20, onde notamos pequenas variações. Assim como nos experimen-

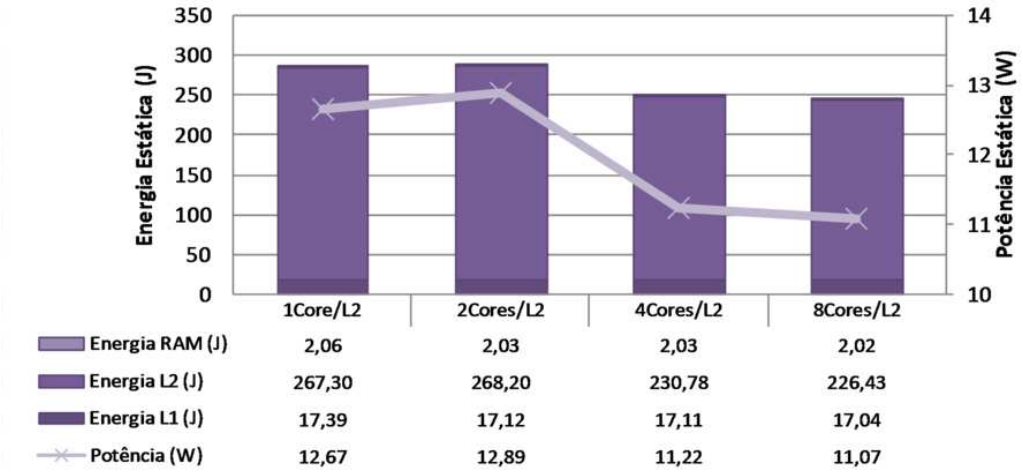


Figura B.19: Consumo de energia e potência estática do sistema de memória do terceiro experimento.

tos anteriores, tais variações podem ter sido geradas por influência do próprio sistema operacional.

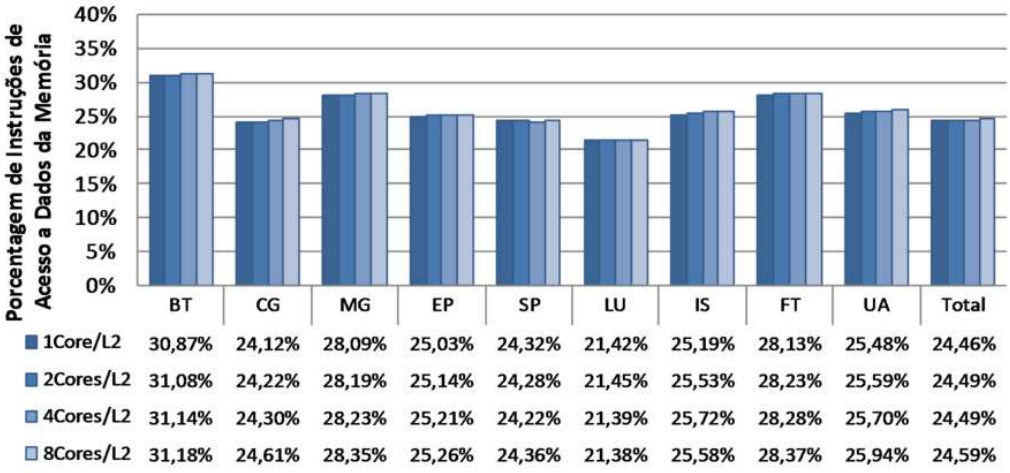


Figura B.20: Instruções de acesso à memória executadas do quarto experimento.

A quantidade de invalidações de dados na memória *cache* L2 para as organizações neste quarto experimento foram menores, se comparado ao primeiro experimento, o que pode ser visto na Figura B.21.

O consumo de energia e potência dinâmica gerado pelas operações na hierarquia de memória está apresentado na Figura B.22. Podemos notar um menor consumo na organização 4Cores/L2, sendo essa a que obteve menor consumo também na memória *cache* L2. A redução no consumo de potência da memória *cache* L2 pela organização 4Cores/L2 se deve à própria redução no consumo por operação, quando utilizado bancos de 4 MB com linha de 128 Bytes, como a modelada nessa organização. Quando comparada com as demais memórias modeladas neste experimento, a de 4 MB é a que apresenta menor consumo dinâmico por operação.

Presente na Figura B.23, o gráfico de consumo de energia e potência estática apre-

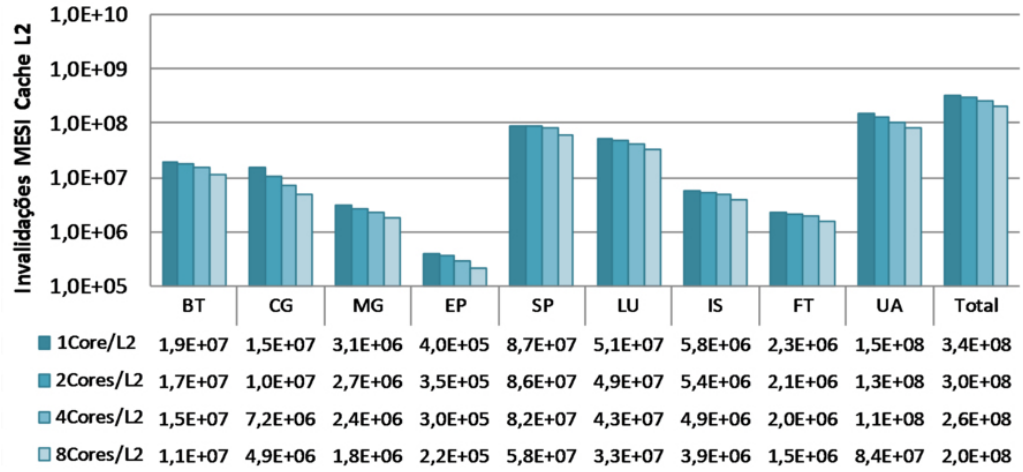


Figura B.21: Invalidações de dados na memória *cache* L2 do quarto experimento.

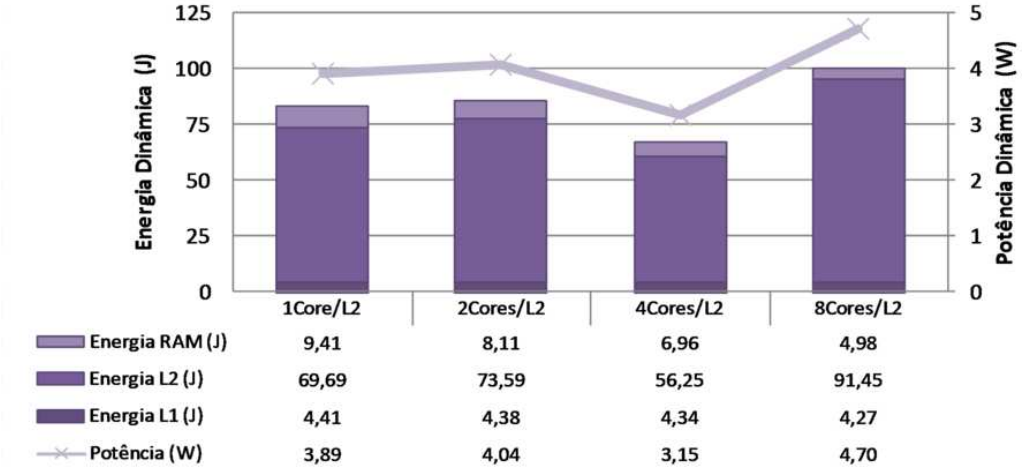


Figura B.22: Consumo de energia e potência dinâmica do sistema de memória do quarto experimento.

senta uma redução no consumo dessa potência a medida que mais núcleos compartilham o mesmo banco de memória *cache* L2. Tal redução é propiciada principalmente pela memória *cache* L2, na qual a densidade dos blocos aumenta, o que leva o consumo estático à redução que foi observada.

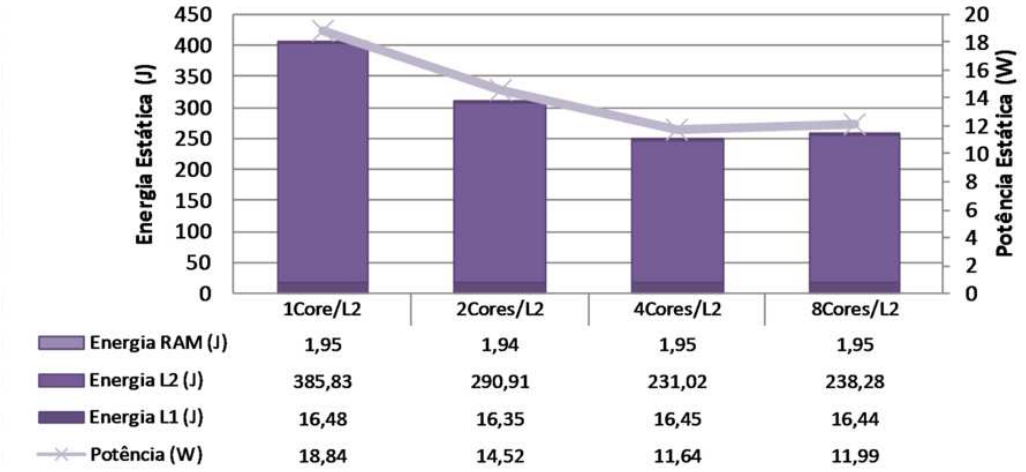


Figura B.23: Consumo de energia e potência estática do sistema de memória do quarto experimento.

B.5 Experimento 5 - Níveis na Hierarquia de Memória *Cache*

A porcentagem de instruções de acesso a dados executadas está presente na Figura B.24, onde podemos ver a variação entre as aplicações. Tal qual os experimentos anteriores, a variação na porcentagem de intruções de acesso a dados é inversa à variação da quantidade de instruções executadas, ou seja, a quantidade dessas instruções se mantém, enquanto a porcentagem sofre variação.

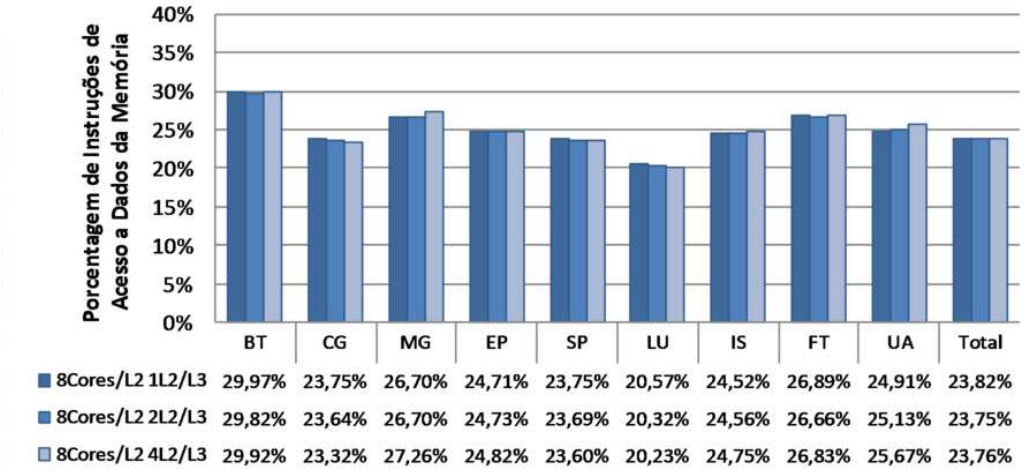


Figura B.24: Instruções de acesso à memória executadas do quinto experimento.

A quantidade de invalidações sofridas pela memória *cache* L2 é apresentado no gráfico da Figura B.25, onde pode-se ver a redução nas invalidações para as organizações 1L2/L3

e 2L2/L3. Essas invalidações são inferiores às apresentadas no primeiro experimento, e a causa disso é a mudança na hierarquia e no modo de controle de coerência.

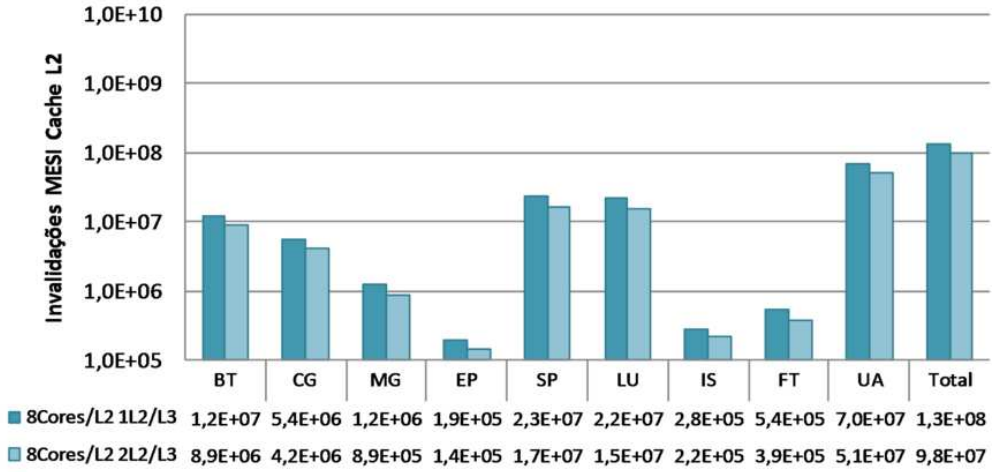


Figura B.25: Invalidações de dados na memória *cache* L2 do quinto experimento.

A variação na quantidade de invalidações MESI para a memória *cache* L3 é ilustrada na Figura B.26. Podemos ver a baixa taxa de redução final, sendo que, para algumas aplicações houve aumento de invalidações conforme aumentou-se a quantidade de memória *cache* L2 compartilhando a mesma memória *cache* L3. Esse aumento de invalidações ocorre devido ao aumento de capacidade da memória *cache* L3, sendo que, ao aumentar o compartilhamento, houve menos faltas e assim, mais invalidações.

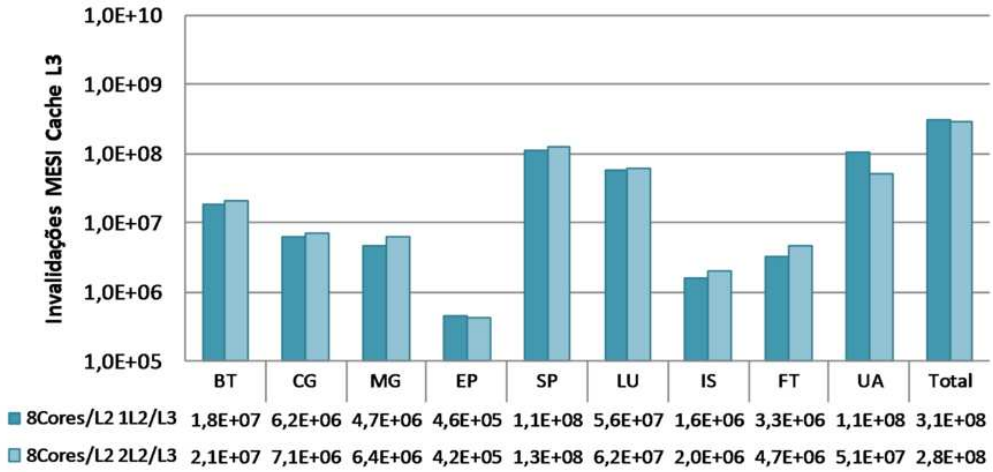


Figura B.26: Invalidações de dados na memória *cache* L3 do quinto experimento.

Avaliando a Figura B.27, podemos notar um aumento no consumo de energia e potência dinâmica conforme aumentou-se o compartilhamento da memória *cache* L3. O aumento da energia consumida pela memória *cache* L3 foi o principal responsável por esse aumento no consumo de energia que ocorre devido ao aumento de faltas na L2 conforme aumentamos o compartilhamento de memória.

Através do gráfico da Figura B.28, notamos a redução no consumo total de energia e potência estática pelo sistema de memória, sendo que neste caso, a redução foi ocasionada

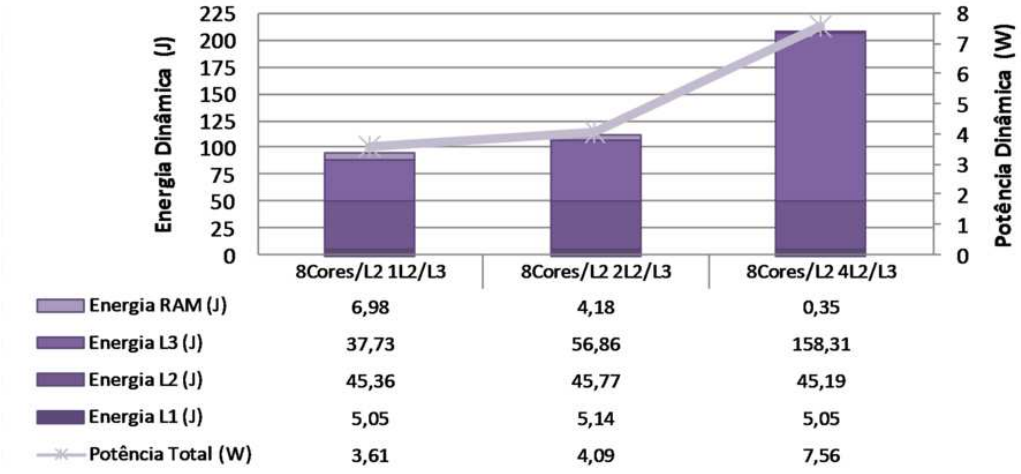


Figura B.27: Consumo de energia e potência dinâmica do sistema de memória do quinto experimento.

principalmente pela memória *cache* L3. Assim como apresentado em vários experimentos, ao utilizar blocos maiores de memória *cache* ao invés de um maior número de blocos menores, a densidade e organização interna dessas memórias *cache* faz o consumo estático decair, como apresentado.

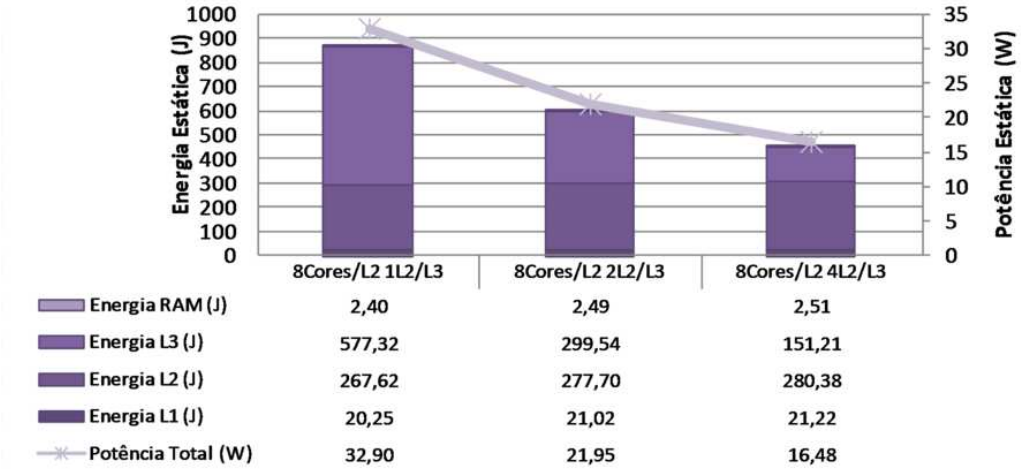


Figura B.28: Consumo de energia e potência estática do sistema de memória do quinto experimento.

REFERÊNCIAS

ACOSTA, C. et al. A complexity-effective simultaneous multithreading architecture. In: INT. CONF. ON PARALLEL PROCESSING, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.157–164.

AGGARWAL, N. et al. Isolation in Commodity Multicore Processors. **Computer**, [S.l.], v.40, n.6, p.49–59, 2007.

AKHTER, S.; ROBERTS, J. **Multi-Core Programming, Increasing Performance through Software Multi-threading**. [S.l.]: Intel Press, 2006.

ALAMELDEEN, A. et al. Simulating a $2M commercial server on a $2K PC. **Computer**, [S.l.], v.36, n.2, p.50–57, Feb 2003.

ALAMELDEEN, A. R. et al. Evaluating Non-deterministic Multi-threaded Commercial Workloads. **Computer Architecture Evaluation using Comercial Workloads**, [S.l.], 2002.

ALAMELDEEN, A. R.; WOOD, D. A. Variability in Architectural Simulations of Multi-threaded Workloads. In: HPCA: INT. SYMP. ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE, 2003. **Proceedings...** [S.l.: s.n.], 2003.

ALVES, M. A. Z.; FREITAS, H. C.; NAVAUX, P. O. A. Investigation of Shared L2 Cache on Many-Core Processors. In: WORKSHOP ON MANY-CORE, 2009, Berlin. **Proceedings...** VDE Verlag GMBH, 2009. p.21–30.

ALVES, M. A. Z.; FREITAS, H. C.; WAGNER, F. R.; NAVAUX, P. O. A. Influência do Compartilhamento de Cache L2 em um Chip Multiprocessado sob Cargas de Trabalho com Conjuntos de Dados Contíguos e Não Contíguos. In: WSCAD: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 2007. **Anais...** [S.l.: s.n.], 2007.

AMD. AMD Opteron Processor for Servers and Workstations. **Advanced Micro Devices**, [S.l.], 2005.

AZIMI, M. et al. Integration Challenges and Tradeoffs for Tera-scale Architectures. **Intel Technology Journal**, [S.l.], v.11, 2007.

BARROSO, L. A. et al. Piranha: a scalable architecture based on single-chip multiprocessing. In: ISCA: INT. SYMP. ON COMPUTER ARCHITECTURE, 2000. **Proceedings...** IEEE, 2000. p.282–293.

BERRIDGE, R. et al. IBM POWER6 microprocessor physical design and design methodology. **IBM Journal of Research and Development**, [S.l.], v.51, n.6, 2007.

BIENIA, C. et al. The PARSEC Benchmark Suite: characterization and architectural implications. **Technical Report - TR-811-08**, [S.l.], 2008.

BJERREGAARD, T.; MAHADEVAN, S. A Survey of Research and Practices of Network-on-Chip. **ACM Computing Surveys**, [S.l.], v.38, p.1–51, 2006.

CHANDRA, R. et al. **Parallel Programming in OpenMP**. [S.l.]: Morgan Kaufmann, 2006.

CHANG, J.; MEMBER, S.; HUANG, M. The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series. **Journal of Solid-State Circuits**, [S.l.], v.42, n.4, 2007.

CHAUDHRY, S. et al. High-Performance Throughput Computing. **IEEE Micro**, Los Alamitos, USA, v.25, n.3, p.32–45, 2005.

CULLER, D. E.; SINGH, J. P. **Parallel Computer Architecture**: a hardware/software approach. San Francisco, USA: Morgan Kaufmann, 1999.

DEMONE, P. Sizing Up the Super Heavyweights. **Real World Technologies**, [S.l.], 2004. Disponível em: <http://www.realworldtech.com/page.cfm?ArticleID=RWT090406012516>. Acesso em: maio 2007.

EGGERS, S. J. et al. Simultaneous Multithreading: a platform for next generation processors. **IEEE Micro**, [S.l.], p.12–19, 1997.

FREITAS, H. C.; NAVAUX, P. O. A. **Chip Multithreading**: conceitos, arquiteturas e tendências. 2006. Trabalho Individual — Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

GONÇALVEZ, R.; NAVAUX, P. O. A. Improving SMT performance scheduling processes. In: EUROMICRO, WORKSHOP ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING, 2002. **Proceedings...** [S.l.: s.n.], 2002. p.327–334.

HAMMOND, L. et al. The Stanford Hydra CMP. **IEEE Micro**, Los Alamitos, CA, USA, v.20, n.2, p.71–84, 2000.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture**: a quantitative approach. 4.ed. USA: Elsevier, Inc., 2007.

HSU, L. et al. Exploring the cache design space for large scale CMPs. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.33, n.4, p.24–33, 2005.

INTEL CORPORATION. Advancing multi-core technology into the tera-scale era. **Intel Teraflops Research Chip**, [S.l.], 2007. Disponível em: <http://www.intel.com/go/terascale>. Acesso em: janeiro 2008.

INTEL CORPORATION. Intel 64 and IA-32 Intel Architectures Software Developer's Manual: basic architecture. **Intel Corporation**, [S.l.], 2007.

INTEL CORPORATION. Core i7 Product Brief. **Intel Technology Journal**, [S.l.], 2008.

INTEL CORPORATION. Intel Core i7 Processor Extreme Edition Series and Intel Core i7 Processor. **Intel Data Sheet**, [S.l.], 2008.

JAIN, R. **The art of computer systems performance analysis**: techniques for experimental design, measurement, simulation, and modeling. New York, USA: J. Wiley, 1991.

JALEEL, A.; MATTINA, M.; JACOB, B. Last Level Cache (LLC) Performance of Data Mining Workloads on a CMP: a case study of parallel bioinformatics workloads. In: INT. SYMP. ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE, 2005. **Proceedings...** IEEE, 2005. p.88–98.

JIN, H.; FRUMKIN, M.; YAN, J. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. In: NAS TECHNICAL REPORT NAS-99-011, 1999. **Proceedings...** NASA Ames Research Center, 1999.

JOHN, L. K.; EECKHOUT, L. **Performance Evaluation and Benchmarking**. Boca Raton, USA: Taylor & Francis Group, 2006.

KANTER, D. Niagara II: the hydra returns. **Real World Technologies**, [S.l.], 2006. Disponível em: <http://www.realworldtech.com/page.cfm?ArticleID=RWT090406012516>. Acesso em: maio 2007.

KIM, C.; BURGER, D.; KECKLER, S. Q. An adaptive, non-uniform cache structure for wire-delay dominated on-chip-caches. In: INT. CONF. ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 2002. **Proceedings...** IEEE, 2002. p.211–222.

KIM, C.; BURGER, D.; KECKLER, S. W. Non-Uniform Cache Architetures for Wire-Delay Dominated On-Chip Caches. **IEEE Computer**, [S.l.], 2003.

KONGETIRA, P.; AINGARAN, K.; OLUKOTUN, K. Niagara: a 32-way multithreaded sparc processor. **IEEE Micro**, [S.l.], v.25, n.2, p.21–29, 2005.

KOUFATY, D.; MARR, D. T. Hyperthreading technology in the netburst microarchitecture. **IEEE Micro**, [S.l.], v.23, n.2, p.56–65, 2003.

KUMAR, R. et al. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In: SIGARCH COMPUT. ARCHIT. NEWS, 2004, New York, NY, USA. **Proceedings...** ACM, 2004. v.32, n.2, p.64.

KUMAR, R. et al. Heterogeneous chip multiprocessors. **IEEE Computer**, [S.l.], v.38, n.11, p.32–38, 2005.

KUMAR, R.; ZYUBAN, V.; TULLSEN, D. Interconnections in Multi-core Architectures: understanding mechanisms, overheads and scaling. In: ISCA: INT. SYMP. ON COMPUTER ARCHITECTURE, 2005. **Proceedings...** IEEE, 2005. p.408–419.

LANDAU, S. I. **Cambridge dictionary of American English**. USA: Cambridge University Press, 2000.

LE, H. Q. et al. IBM POWER6 microarchitecture. **IBM Journal of Research and Development**, [S.l.], v.51, n.6, 2007.

LILJA, D. J. **Measuring Computer Performance**. Cambridge: Cambridge University Press, 2004.

MAGNUSSON, P. et al. Simics: a full system simulation platform. **IEEE Computer Micro**, [S.l.], v.35, n.2, p.50–58, Feb 2002.

MARINO, M. D. 32-core CMP with multi-sliced L2: 2 and 4 cores sharing a l2 slice. In: SBAC-PAD: INT. SYMP. ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 2006. **Proceedings...** IEEE, 2006. p.141–150.

MARINO, M. D. L2-cache hierarchical organizations for multi-core architectures. In: ISPA: INT. SYMP. ON PARALLEL AND DISTRIBUTED PROCESSING AND APPLICATIONS, 2006. **Proceedings...** IEEE, 2006. p.74–83.

MARTY, M. et al. General Execution-driven Multiprocessor Simulator. In: ISCA TUTORIAL, 2005. **Proceedings...** IEEE, 2005.

MCNAIRY, C.; BHATIA, R. MONTECITO: a dual-core, dual-thread itanium processor. **IEEE Micro**, [S.l.], v.16, 2005.

MENASCÉ, D. A.; ALMEIDA, V. A. F. **Planejamento de Capacidade para Serviços na WEB**. Rio de Janeiro, Brazil: Editora Campus, 2002.

MEYER, B. H.; THOMAS, D. E. Simultaneous synthesis of buses, data mapping and memory allocation for MPSoC. In: CODES+ISSS: INT. CONF. ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS, 2007, New York, USA. **Proceedings...** ACM, 2007.

NAYFEH, B. A.; OLUKOTUN, K.; SINGHT, J. P. The Impact of Shared-Cache Clustering in Small-Scale Shared-Memory Multiprocessors. In: HPCA: SECOND INT. SYMP. ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE, 1996. **Proceedings...** IEEE, 1996. p.74–84.

OLUKOTUN, K. et al. The Case for a Single-Chip Multiprocessor. In: ASPLOS: INT. SYMP. ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 1996. **Proceedings...** IEEE, 1996. p.2–11.

PATTERSON, D. A.; HENNESSY, J. L. **Organização e projeto de computadores**. 2.ed. Rio de Janeiro, Brasil: Editora Campus, 2005.

PENG, L. et al. Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: a case study. In: IPCCC: INT. PERFORMANCE, COMPUTING, AND COMMUNICATIONS CONF., 2007. **Proceedings...** [S.l.: s.n.], 2007.

PRABHU, G. M. Computer Architecture Tutorial. **Iowa State University of Science and Technology**, [S.l.], 2008. Disponível em: <http://www.cs.iastate.edu/ prabhu/Tutorial/title.html>. Acesso em: setembro 2008.

ROSE, C. A. F. D.; NAVAUX, P. O. A. **Arquiteturas Paralelas**. Porto Alegre RS, Brasil: Editora Sagra Luzzatto, 2003.

SIMONG. **Inaccurate Simulation**. [S.l.]: Simics Forum, 2007. Disponível em: <https://www.simics.net/mwf/topic_show.pl?pid=55029>. Acesso em: agosto 2008.

SINHAROY, B. et al. POWER5 system microarchitecture. **IBM Journal of Research and Development**, [S.l.], v.49, n.4/5, 2005.

SMITH, J. E.; SOHI, G. S. The Microarchitecture of Superscalar Processors. **IEEE**, [S.l.], v.83, n.12, p.1609–1624, 1995.

SPRACKLEN, L.; ABRAHAM, S. Chip Multithreading: opportunities and challenges. In: HPCA: INT. SYMP. ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE, 2005. **Proceedings...** IEEE, 2005. p.248–252.

STALLINGS, W. **Computer Organization and Architecture**: designing for performance. 4.ed. USA: Prentice Hall, 1996.

STANDARD PERFORMANCE EVALUATION CORPORATION - SPEC. **SPEC CPU2006**. [S.l.: s.n.], 2007. Disponível em: <http://www.spec.org>. Acesso em: fevereiro 2008.

THOZIYOOR, S. et al. A comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies. In: ISCA: INT. SYMP. ON COMPUTER ARCHITECTURE, 2008. **Proceedings...** [S.l.: s.n.], 2008. p.51–62.

UNGERER, T.; ROBIC, B.; SILC, J. Multithreaded Processors. **British Computer Society**, [S.l.], v.45, n.3, p.320–348, 2002.

UNGERER, T.; ROBIC, B.; SILC, J. A Survey of Processors with Explicit Multithreading. **ACM Computing Surveys**, [S.l.], v.35, n.1, p.29–63, 2003.

VIRTUTECH SIMICS. **Simics 3.0 – User Guide for Unix**. [S.l.: s.n.], 2007. Disponível em: <http://www.simics.net>. Acesso em: junho 2008.

WOLF, W. The Future of Multiprocessor System on Chip. In: DAC: DESIGN AUTOMATION CONF., 2004. **Proceedings...** [S.l.: s.n.], 2004.

WOO, S. et al. The SPLASH-2 programs: characterization and methodological considerations. In: ISCA: INT. SYMP. ON COMPUTER ARCHITECTURE, 1995. **Proceedings...** [S.l.: s.n.], 1995.

ZAHRAN, M. M. On cache memory hierarchy for Chip-Multiprocessor. **SIGARCH Computer Architecture News**, New York, USA, v.31, p.39–48, 2003.

ZHAO, X. et al. Split Private and Shared L2 Cache Architecture for Snooping-based CMP. In: ICIS: INT. CONF. ON COMPUTER AND INFORMATION SCIENCE, 2007. **Proceedings...** IEEE, 2007. p.900–905.