UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MAURÍCIO MACHADO SILVEIRA

Simulação e Avaliação de Desempenho de Arquiteturas Paralelas Utilizando a Ferramenta Simics

Trabalho de Graduação.

Prof. Dr. Philippe Olivier Alexandre Navaux
Orientador

Porto Alegre, novembro de 2008.

Dedico este trabalho à minha mãe,
por tudo e sempre.

AGRADECIMENTOS

Agradeço a todas as pessoas que, direta ou indiretamente, contribuíram para a realização deste. Em especial, agradeço à minha mãe, Maria Lúcia, meu alicerce, minha bússola e a principal responsável por eu chegar aonde cheguei. À minha amada namorada, Nathália, por todo carinho, companheirismo e compreensão em todos os momentos, sejam eles quais forem. Agradeço aos meus queridos amigos e colegas, com os quais passei boa parte de minha vida e que traçaram ao meu lado este árduo caminho, sem deixar, mesmo inconscientemente, que eu jamais desistisse da jornada. Muito obrigado ao meu orientador, Philippe Navaux, pela grande ajuda e tutoria no desenvolvimento deste trabalho. Ao Henrique Cota de Freitas, pelas valiosas considerações e revisões, e pela idéia do tema do trabalho. Agradeço ao Marco Antônio Zanata Alves, pela paciência e imensurável apoio para que este trabalho realmente fosse realizado. À Ida Rossi, pelas incansáveis revisões bibliográficas. E a todos que se interessarem em ler este trabalho, meu muito obrigado.

“Nem tão longe que eu não possa ver
Nem tão perto que eu possa tocar
Nem tão longe que eu não possa crer
Que um dia chego lá”

H. Gessinger

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

E/S	Entrada e Saída
CPU	*Central Processing Unit*
FSB	*Front Side Bus*
MESI	*Modified, Exclusive, Shared, Invalid*
MIMD	*Multiple Instruction – Multiple Data*
MISD	*Multiple Instruction – Single Data*
PARSEC	*Princeton Application Repository for Shared-Memory Computers*
RAM	*Random-Access Memory*
SIMD	*Single Instruction – Multiple Data*
SISD	*Single Instruction – Single Data*
SMP	*Symmetric Multiprocessor*
SO	Sistema Operacional
TDP	*Thermal Design Power*
TLB	*Translation Look-Aside Buffer*
VLSI	*Very Large Scale Integration*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Desde os primórdios da computação tem-se ciência de que o processamento paralelo e o paralelismo em geral aumentam o desempenho de execução das aplicações e tornam as máquinas mais rápidas. Desta forma, muitas técnicas e abordagens foram criadas nas últimas décadas para tirar proveito do paralelismo em diversos níveis.

Nos últimos anos, tornou-se muito difundido o conceito de arquiteturas paralelas, principalmente com o advento dos processadores com mais de um núcleo. São capazes de realizar processamentos de forma que vários fluxos de execução possam ocorrer ao mesmo tempo. Mas esta concorrência, por mais que exista um número elevado de elementos de processamento, é limitada por outros componentes, sendo o principal deles a memória *cache*, que "alimenta" os processadores de forma rápida possibilitando um maior paralelismo. Assim, tão importante quanto os processadores, a arquitetura de memória *cache* é fator fundamental nas máquinas paralelas.

Este trabalho visa estudar o desempenho de arquiteturas paralelas comerciais, desenvolvidas por fabricantes distintos, tendo como foco o comportamento da memória *cache* em cada modelo com relação à sua arquitetura. Para isso, será utilizado o simulador Simics para modelar estas arquiteturas e o conjunto de *benchmarks* PARSEC para realizar os testes de desempenho. Ao fim, após avaliar os resultados, pretende-se demonstrar pontos positivos e negativos de cada abordagem, realizando uma comparação entre os modelos estudados.

Palavras-Chave: Arquiteturas paralelas, simulação, teste de desempenho, multinúcleos.

Simulation and Performance Evaluation of Parallel Architectures Using the Simics Tool

ABSTRACT

Since the early days of computing science it’s known that the parallel processing and parallelism in general increase the executing performance of the applications and make the machines faster. Thus, many techniques and approaches have been created in the last decades to take advantage of the parallelism at various levels.

In recent years, has become widespread the concept of parallel architectures, especially with the advent of processors with more than one core. They are capable of processing so that multiple streams of execution may occur at the same time. But this competition, however that there is a large number of processing elements, is limited by other components, the main one being the cache, which "feeds" the processors quickly allowing greater parallelism. Thus, as important as the processor, the architecture of cache memory is a major factor in parallel machines.

This work aims to study the performance of commercial parallel architectures, developed by different manufacturers, with the focus of the behavior of cache memory on each model with respect to its architecture. This will use the simulator Simics to model these architectures and PARSEC suite of benchmarks for the testing of performance. In the end, after analyzing the results, it is intended to demonstrate positive and negative points of each approach, making a comparison between the models studied.

Key-Words: Parallel architectures, simulation, performance test, multicores.

1 INTRODUÇÃO

1.1 Contexto do Trabalho

Com a utilização cada vez maior de processadores com múltiplos núcleos (os chamados *multicores*), novas tecnologias e estratégias de programação estão sendo desenvolvidas visando estas arquiteturas paralelas. Este avanço não se dá apenas no meio industrial e acadêmico, mas também com grande freqüência nos computadores pessoais. Seja para fins comerciais, científicos ou de entretenimento, o emprego da concorrência se faz necessário a fim de alcançar o desempenho requerido. E é justamente nesta necessidade de maior desempenho que surge o emprego do processamento paralelo.

O conceito de processamento paralelo é quase tão antigo quanto o conceito de computação. Estudiosos já projetavam algoritmos para resolver problemas matemáticos de forma paralela antes mesmo da invenção do computador como hoje o conhecemos. Além disso, máquinas para cálculos diferenciais já funcionavam com o conceito de concorrência de execução desde meados da década de 20 (STALLINGS, 2002). Isso demonstra que há muito notou-se os ganhos possibilitados pelo processamento paralelo. Assim, mesmo em períodos onde a tecnologia ainda não permitia a aplicação de mais de um elemento processador por máquina, diversas técnicas foram desenvolvidas com a finalidade de obter ganho a partir de algum nível de paralelismo nas execuções.

Como apenas agrupar processadores não é suficiente para obter um processamento paralelo eficiente, se faz necessário um estudo aprofundado de toda a arquitetura que cerca os processadores, como memórias, registradores, interconexões, entre outros, e talvez mais importante, como estas unidades devem se comunicar. Desta forma, cada fabricante de processadores planeja e monta sua própria arquitetura, a fim de conseguir o melhor desempenho da máquina. Mesmo assim, toda arquitetura possui seus pontos críticos, chamados de “gargalos”. E é justamente nestes pontos que os fabricantes mais divergem quanto às arquiteturas, e investem muito para superar os concorrentes.

Este trabalho visa um estudo sobre estas diferentes arquiteturas paralelas, de variados fabricantes, abordando arquiteturas comerciais, disponíveis ao grande público. Para tanto, foram realizados testes de carga de trabalho a fim de avaliar o desempenho das mesmas, identificar gargalos, comparar os resultados e determinar possíveis soluções e melhorias que poderiam ser implementadas. Com este intuito, foi utilizada a ferramenta Simics de modelagem e simulação de arquiteturas completas. A partir desta ferramenta, é possível obter resultados bastante próximos aos obtidos com as máquinas reais em questão, nunca esquecendo que nenhuma simulação é igual à execução real, sendo estes valores uma aproximação da realidade.

O foco deste trabalho é sobre a arquitetura das memórias *cache*, elemento fundamental no processamento paralelo e um dos principais pontos de discordância entre as implementações existentes. Com mais ou menos níveis de hierarquia, com maior ou menor capacidade de armazenamento, cada fabricante projeta sua arquitetura de memória *cache* com a finalidade de extrair o maior paralelismo possível e executar o maior número de operações em menor tempo.

1.2 O Problema

Determinar o desempenho de uma máquina não é algo trivial, principalmente quando se faz necessária a avaliação de determinado ponto da arquitetura da máquina com maior detalhamento. Além disso, como o intuito deste trabalho é o estudo sobre arquiteturas comerciais, devem ser trabalhadas aquelas de maior utilização neste meio, que são as arquiteturas x86. Estas arquiteturas, mesmo sendo muito difundidas, possuem poucos estudos acerca de desempenho e funcionalidade.

Outros pontos importantes na análise são em relação à simulação e à carga de trabalho que serão utilizadas na mesma. Deve-se ter certeza de que a modelagem descreva com considerável fidelidade a máquina a qual deseja-se simular, pelo menos nos componentes que serão o foco do estudo. Além disso, o simulador deve retratar o mais próximo possível a execução do modelo de forma que o desempenho deste seja equivalente ao da máquina real. Ainda, a carga de trabalho deve ser abrangente e relativamente ampla, a fim de averiguar a funcionalidade do modelo em diversos conjuntos de aplicações.

1.3 Objetivos

Para a realização de uma modelagem eficiente das arquiteturas utilizadas, será efetuado um estudo de suas características e configurações de seus componentes. Tendo como base as arquiteturas x86, serão utilizados modelos de dois e quatro núcleos dos fabricantes Intel Corporation e AMD. Estes modelos são bastante recentes no mercado e facilmente encontrados à venda, por preços bastante acessíveis. Os modelos de dois *cores*, de ambos fabricantes, dizem respeito a computadores portáteis; já os modelos de quatro núcleos são processadores para microcomputadores de mesa e servidores.

As simulações serão realizadas através do simulador Simics, ferramenta altamente reconhecida nos meios acadêmico e comercial para este fim. Muito utilizado por projetistas no mundo todo, o Simics é desenvolvido pela Virtutech AB, empresa que surgiu dentro do Instituto de Ciência da Computação da Suécia, grande centro de estudo e desenvolvimento em computação, principalmente na área de arquitetura de computadores. O Simics simula com fidúcia as arquiteturas modeladas, reproduzindo inclusive os ciclos e passos de cada processador.

Com relação à carga de trabalho, será utilizado o conjunto de *benchmarks* PARSEC, desenvolvido pela Universidade de Princeton em parceria com a Intel. Atualmente um dos mais utilizados e conceituados conjuntos de aplicativos de testes, tem como uma de suas características a inserção de aplicações emergentes, desenvolvidas para o desenvolvimento das próximas gerações de processadores. Estas aplicações, que requerem grande capacidade de processamento e armazenamento, abrangem diversas áreas do estudo científico da computação, o que torna ainda mais representativos os resultados obtidos.

A partir da utilização destas ferramentas e do estudo das arquiteturas que serão analisadas, o objetivo deste trabalho é avaliar o desempenho das arquiteturas de memória *cache* dos modelos. A variedade das abordagens torna o estudo mais interessante, pois a partir dos resultados é possível traçar uma métrica de qual abordagem teve maior destaque e quais são os possíveis motivos para isto.

1.4 Organização do Trabalho

Após este capítulo introdutório, será realizada uma breve revisão sobre conceitos de paralelismo e arquiteturas paralelas. Após, serão apresentados os modelos de arquiteturas a serem estudados, simulados e testados. No capítulo seguinte, que aborda a metodologia utilizada, serão apresentadas as ferramentas básicas deste trabalho: o simulador Simics, responsável pela modelagem e simulação das arquiteturas, o conjunto de *benchmarks* PARSEC, utilizado como carga de trabalho para avaliar o desempenho dos modelos, e a ferramenta CACTI, utilizada para calcular as latências das *caches*. A seguir um capítulo explicando os passos tomados na realização do experimento, como a simulação e a realização dos testes. Em seguida, serão apresentados os resultados obtidos dos testes de desempenho das arquiteturas e as correspondentes análises. Após, um capítulo citando alguns trabalhos relacionados já realizados, envolvendo simulação de memórias *cache*, teste de desempenho de arquiteturas paralelas e outros assuntos correlatos. Por fim, serão apresentadas as conclusões, com as análises e considerações finais acerca deste trabalho.

Ao fim deste, após a conclusão, encontram-se nos apêndices, de A a E, as tabelas com os resultados exatos dos testes de desempenho de cada arquitetura, obtidos através do simulador Simics. Também é possível verificar, no Apêndice F, os *scripts* utilizados para a modelagem das arquiteturas utilizadas para a simulação de cada modelo.

2 PARALELISMO: UMA BREVE REVISÃO

2.1 Visão Geral

Tradicionalmente, o estudo da informática tem se dado sob uma ótica na qual o computador é visto como uma máquina de processamento seqüencial, seguindo algoritmos que especificam exatamente uma ordem de relação entre as instruções. Até hoje o modelo de programação seqüencial continua sendo o mais difundido. Mas o emprego de algum tipo de paralelismo se faz presente nos computadores há bastante tempo, tornando esta visão da computação não totalmente verdadeira.

Desde máquinas como o UNIVAC 1 (ECKERT-MAUCHLY COMPUTER CORP., 1956), que escalonava operações de processamento com operações de E/S – técnica chamada de *overlap* ou sobreposição – os computadores começaram a ser implementados com algum tipo de processamento paralelo a fim de aumentar o desempenho. Como exemplo é possível citar a técnica de *pipeline*, que utiliza concorrência temporal das diversas fases da execução de instruções, onde unidades específicas de *hardware* executam cada subtarefa de uma instrução de forma que diversas instruções possam ser executadas paralelamente. Esta técnica, empregada em meados da década de 50 nos computadores IBM 7030 (também conhecido como Stretch) (BLOCK, 1959) e UNIVAC LARC (ECKERT et al., 1959), até hoje continua sendo a base para a manufatura de processadores mais velozes, como afirmam Hennessy e Patterson (2007).

Apesar da constante evolução na fabricação de componentes eletrônicos cada vez mais rápidos e com maior poder de processamento, chegamos a um momento na história da arquitetura de computadores onde o modelo seqüencial de Von Neumann parece tender a um limite de desempenho, sendo cada vez mais difícil obter ganho de desempenho considerável nestas máquinas (ALMASI; GOTTLIEB, 1994). Faz-se então necessário investir em um novo paradigma que consiga aumentar o desempenho justamente onde as máquinas seqüenciais não conseguem: no paralelismo de instruções.

A motivação básica de uma arquitetura paralela é criar uma máquina poderosa a partir da conexão pura e simples de um conjunto de máquinas menores. Se isto fosse possível, conseguiríamos unir tantos processadores quanto nosso orçamento nos permitisse e teríamos um poder de computação diretamente proporcional ao número de processadores adquiridos. Mas sabemos que isto não é algo alcançável, pois há diversos fatores que degradam esse aumento linear do desempenho, como atraso nas conexões, controle de consistência de memória, e os próprios *softwares*, que muitas vezes não são paralelizáveis. Isto tudo faz com que o *speedup*, que é a taxa de aumento de desempenho conforme o aumento de processadores, em diversos casos, seja limitada.

Para entender como se dá este aumento do desempenho em função do paralelismo, é preciso compreender as diferentes categorias de sistemas de computação e as técnicas de paralelismo empregadas em cada uma delas. Até hoje a classificação mais usual é a de Flynn (1966), dividindo todos os sistemas de computador em quatro grupos baseados no fluxo de instruções e fluxo de dados.

2.2 Taxonomia de Flynn

Flynn (1966) afirma que o processo de computação é essencialmente a execução de uma seqüência de instruções sobre um conjunto de dados. Logo, toda computação depende da multiplicidade ou não dos fluxos de instruções e de dados. Baseado nisso, Flynn determinou as seguintes categorias:

Tabela 2.1: Classificação de Flynn segundo o fluxo de instruções e o fluxo de dados

	SD (*Single Data*)	MD (*Multiple Data*)
SI (*Single Instruction*)	SISD Máquinas Von Neumann convencionais	SIMD Máquinas *Array* (CM-2, MasPar)
MI (*Multiple Instruction*)	MISD Sem representante (até agora)	MIMD Multiprocessadores e multicomputadores (nCUBE, Intel Paragon, Cray T3D)

Fonte: DE ROSE; NAVAUX, 2003. p. 5.

2.2.1 SISD (*Single Instruction – Single Data*)

Neste grupo encontram-se basicamente os computadores monoprocessados, com um único fluxo de instruções atuando sobre um único fluxo de dados armazenados em uma única memória. Aqui enquadra-se o modelo seqüencial de Von Neumann. Nestas máquinas é possível trabalhar com um paralelismo em nível de instrução (ou microinstrução), utilizando técnicas de *pipeline*, tanto em *hardware* (dinâmicas) quanto em *software* (estáticas), como predição de saltos (ACIIÇMEZ; KOÇ; SEIFERT, 2007), especulação e *loop unrolling*.

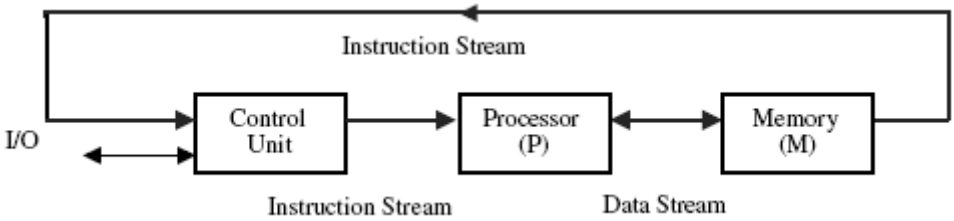

Figura 2.1: Diagrama da classe SISD (EL-REWINI; ABD-EL-BARR, 2005).

Outra técnica muito usada para alcançar maior paralelismo entre as unidades funcionais em sistemas SISD é a execução fora de ordem das instruções. Esta técnica tem a finalidade de diminuir o tempo ocioso do processador, devido a uma operação de E/S ou de instruções com dependência de dados entre si no *pipeline*. Com isto, é

possível aproveitar ao máximo o processador, o que oferece um ganho significativo de desempenho. Há diversos algoritmos para execução fora de ordem, a maioria implementada em *hardware*, mas um dos mais importantes e utilizados é o algoritmo de Tomasulo (1967), desenvolvido para a IBM e implementado pela primeira vez no IBM360/91 (ANDERSON; SPARACIO; TOMASULO, 1967). Neste algoritmo utiliza-se a renomeação de registradores, sendo os valores dos registradores lógicos mapeados para registradores físicos de reserva, que são associados por uma etiqueta ao registrador lógico original. Esta prática elimina as antidependências e dependências de saída. No caso de dependências verdadeiras, não há risco de haver inconsistência devido execução fora de ordem porque o despacho das instruções é feito na ordem em que elas se encontram no programa, sendo a ordem passível de alteração apenas se não houver dependência verdadeira.

A classe dos computadores SISD possui uma vasta gama de representantes, desde o ENIAC (GOLDSTINE; GOLDSTINE, 1946), considerado o primeiro computador da história, até o recente Pentium 4 (HINTON et al., 2001). Também são importantes representantes desta classe o UNIVAC 1 (ECKERT-MAUCHLY COMPUTER CORP., 1956), o PowerPC 601 (MOORE, 1993) e o Macintosh 128K (APPLE INC., 2008).

2.2.2 SIMD (*Single Instruction – Multiple Data*)

A mesma instrução é executada simultaneamente por múltiplos processadores (ou elementos de processamento) sobre diferentes fluxos de dados, sem a utilização de técnicas de confluência. Cada elemento de processamento tem uma memória de dados associada, de modo que cada instrução é executada sobre um conjunto de dados diferente em cada processador. Segundo Flynn (1966), o ganho de desempenho em máquinas desta família é estritamente causado pela utilização de mais unidades, e a comunicação entre estas deve ser restrita a uma determinada vizinhança, seguindo um padrão uniforme.

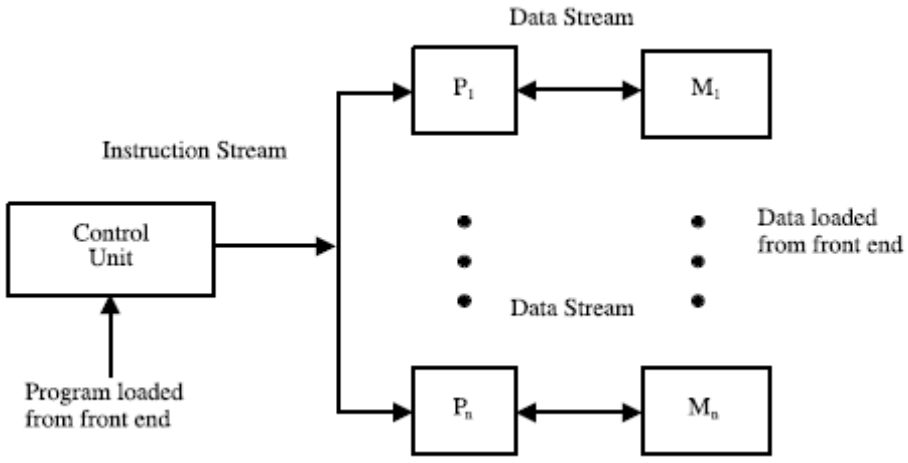

Figura 2.2: Diagrama da classe SIMD (EL-REWINI; ABD-EL-BARR, 2005).

Enquadram-se na categoria SIMD os computadores vetoriais e matriciais, tendo maior destaque o primeiro grupo. Estas máquinas, ao invés de operarem de forma escalar sobre um dado ou um par de dados, atua sobre um vetor de dados, efetuando operações lógicas e/ou aritméticas. Além disso, cada unidade de processamento possui seu próprio registrador de endereços. Desta forma, todos os elementos de

processamento executarão a mesma instrução, porém cada um sobre um endereço diferente do vetor ou em vetores diferentes, obtendo todos os resultados em apenas um ciclo de *clock* e um *program counter*. Segundo Hwang (1993), em geral, o processamento de vetores é mais rápido e mais eficiente que o processamento escalar, pois reduz o *overhead* de *software* para controle de ciclos, reduz conflitos de acesso à memória, além de ser bem relacionado com os conceitos de *pipeline* e segmentação.

Como principais exemplos de máquinas desta família temos o Solomon (GREGORY; MCREYNOLDS, 1963), o Illiac IV (BARNES et al., 1968; DAVIS, 1969) e a Connection Machine 2 (CM-2) (HELIN, 1992).

2.2.3 MISD (*Multiple Instruction – Single Data*)

Nesta família, diversos processadores executariam seqüências diferentes de instruções, mas sobre o mesmo conjunto de dados. É como se todas as execuções ocorressem sobre a mesma posição de memória, a qual é lida e escrita por todos os fluxos de instruções.

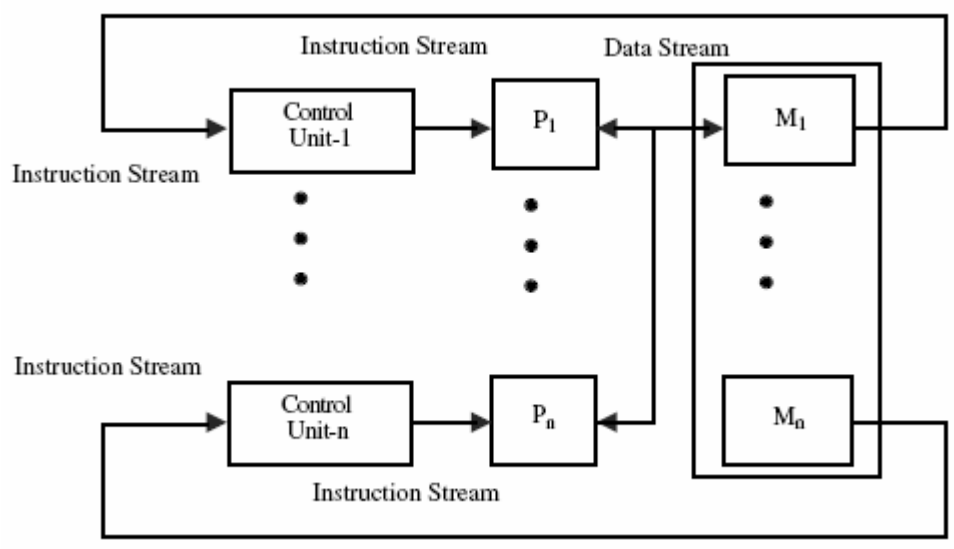

Figura 2.3: Diagrama da classe MISD

Pela dificuldade de imaginar a implementação de uma arquitetura real deste tipo, esta categoria até hoje não possui representante, embora existam estudos de arquiteturas teóricas e multiprocessadores não-comerciais que utilizariam este conceito. Além disso, algumas arquiteturas para reconhecimento de padrões (HALAAS et al., 2004) e tolerância a falhas (CONTESSA, 1988) podem ser consideradas de características MISD.

2.2.4 MIMD (*Multiple Instruction – Multiple Data*)

Em estruturas desta família, um conjunto de processadores executa simultaneamente (de forma assíncrona) seqüências diferentes de instruções, sobre conjuntos de dados distintos. Desta forma, cada processador pode executar seu próprio programa ou parte de um programa sobre seus próprios dados (DE ROSE; NAVAUX, 2003).

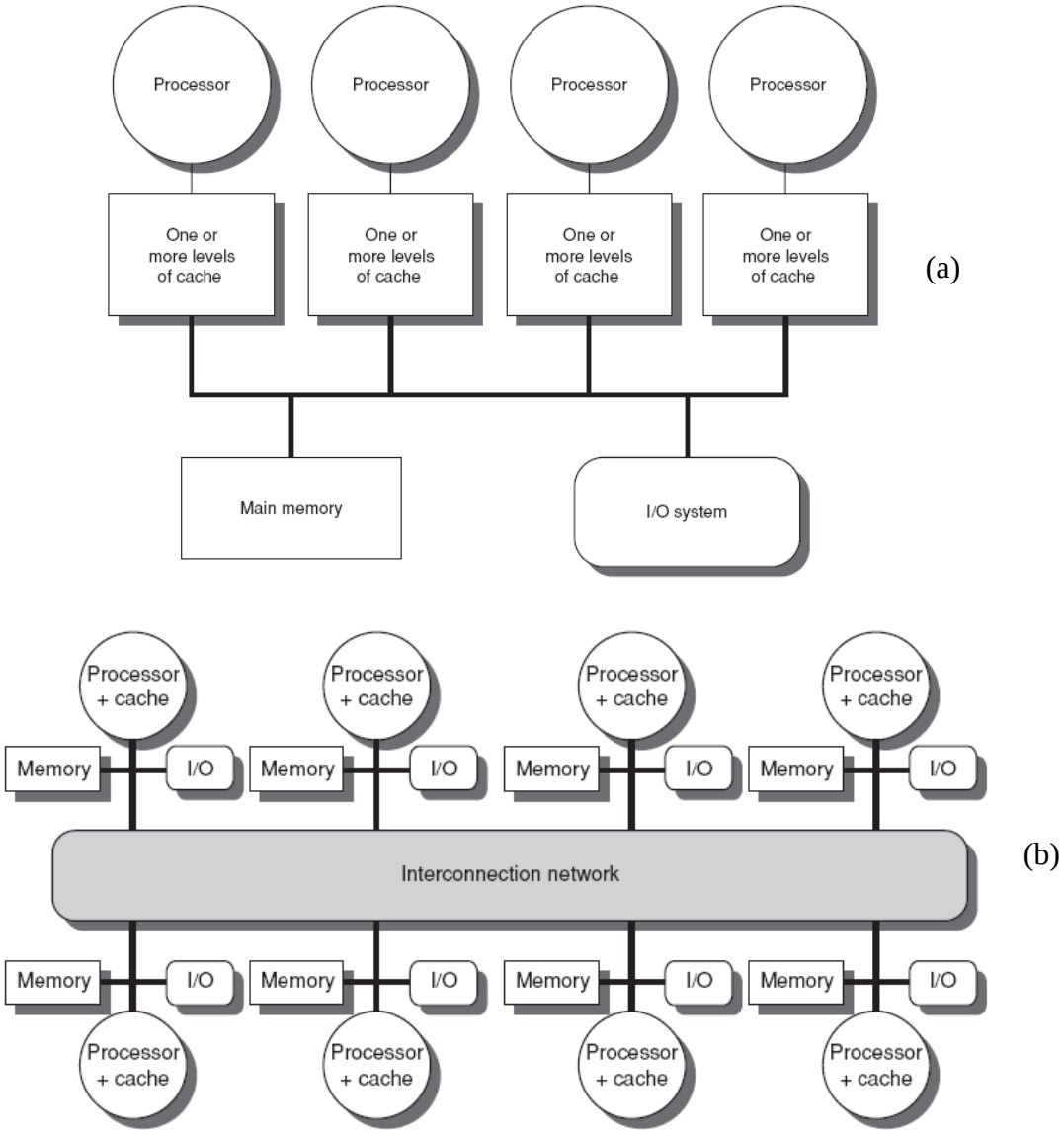

Figura 2.4: Diagramas da classe MIMD com (a) memória compartilhada e (b) memória distribuída (HENNESSY; PATTERSON, 2007).

Os sistemas MIMD podem ser classificados a partir do modo de comunicação entre os processadores. A forma mais usual, e que será o foco deste trabalho, é a utilização de uma memória compartilhada (Figura 2.4a), onde os dados de todos os processadores são lidos e armazenados. Em geral, estes sistemas de memória compartilhada são formados por processadores simétricos, ou seja, não há uma unidade de processamento que coordene as atividades do sistema. Os SMPs, como são chamados, utilizam-se de memória *cache* privada e em vários níveis para diminuir a latência de acesso à memória principal, que por ser grande o suficiente para comportar dados de todos os processadores, torna-se um gargalo quanto ao acesso aos dados. Com esta redução da latência devido o uso da memória *cache*, diversos processadores podem compartilhar a mesma memória principal (HENNESSY; PATTERSON, 2007). Mesmo nos primeiros sistemas SMPs, segundo Bell (1985) a memória *cache* associada aos processadores era responsável por 95% dos acessos a dados (*cache hit*), sendo apenas 5% dos casos necessário acessar a memória principal (*cache miss*).

Outra abordagem de sistemas MIMD utiliza memória distribuída (Figura 2.4b), onde os dados são compartilhados entre os nodos do sistema através de troca de mensagens. O exemplo mais usual deste modelo são os *clusters*, que são sistemas de computadores completos agregados, onde cada máquina coopera com o sistema, mas também pode operar por si própria. Muito utilizados em computação de alto desempenho, a grande vantagem dos *clusters* é a alta disponibilidade, pois no caso de falha em um nodo, o sistema como um todo continua em funcionamento, não havendo perda total de serviço. Fox (1997) cita, além da alta disponibilidade, outras vantagens do *cluster* com relação ao SMP, como escalabilidade, maior capacidade de computação, facilidade de construção do sistema e boa relação custo/desempenho por ser composto de computadores básicos comercialmente disponíveis.

Sobre a arquitetura paralela dos modelos MIMD, deve haver um sistema operacional que efetue o escalonamento de processos e tarefas sobre os processadores. Stallings (2002) afirma que na arquitetura SMP, o SO instalado na máquina deve tratar esta distribuição do fluxo diretamente. Já nos *clusters*, é necessária uma camada de *software* sobre o SO executado em cada computador, que realize o escalonamento e facilite a inclusão ou exclusão de máquinas no sistema.

Como exemplos de arquiteturas MIMD podemos citar a Connection Machine 5 (HILLIS; TUCKER, 1993), o HEP Computer (SMITH, 1978), o IBM S/390 (MAK et al., 1997) e o Intel Core 2 Duo (INTEL CORP., 2008a).

2.3 Aumento de Desempenho e Lei de Amdahl

A utilização do paralelismo, seja em qual nível for, tem a finalidade de executar mais rapidamente uma quantidade de operações de um ou mais programas, aumentando o desempenho e diminuindo o tempo de resposta. Mas como definem Almasi e Gottlieb (1994), toda computação, por maior que seja seu nível de paralelismo, possui uma porção paralela e uma porção serial. Este aumento de desempenho devido ao paralelismo (chamado de *speedup*) é limitado por esta porção seqüencial da execução. Como mostra a Figura 2.5, mesmo que uma aplicação, executando em múltiplos processadores, tenha 95% de sua execução paralelizada, o aumento de desempenho máximo é na ordem de apenas vinte vezes comparado ao desempenho da execução em um único processador.

Amdahl (1967) definiu uma fórmula, conhecida como Lei de Amdahl, com a qual é possível calcular o limite do *speedup* de uma aplicação paralela a partir do número de processadores envolvidos e das porções paralelizável e não-paralelizável da aplicação. Esta fórmula, muito utilizada ainda hoje, é descrita da seguinte forma:

$$Speedup = \frac{1}{r_s + \frac{r_p}{n}} \quad ,$$

onde n representa o número de processadores envolvidos, r_s representa a porcentagem da porção seqüencial do programa, r_p representa a porcentagem da porção paralelizável do programa e $r_s + r_p = 1$.

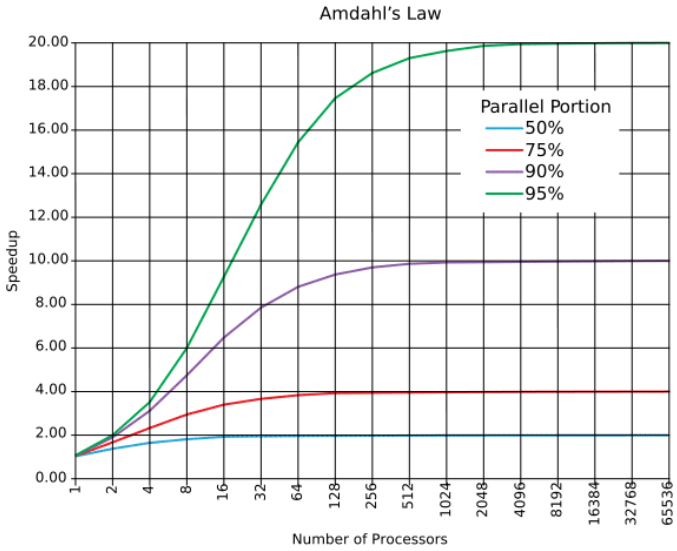

Figura 2.5: Aumento de desempenho em múltiplos processadores (IMAGE, 2008).

Além destes, há mais um elemento limitador de *speedup* que não é representado na Lei de Amdal: o *overhead* de comunicação entre os elementos de processamento. A fórmula anterior ignora este custo excedente, tratando as comunicações como ideais.

O conceito explicitado na Lei de Amdahl é muito importante no estudo do processamento paralelo, pois prova que o aumento do desempenho não é linearmente direto ao acréscimo de processadores. Desta forma, outras abordagens devem ser utilizadas juntamente com o aumento no número de processadores para conseguir um maior *speedup*.

2.4 Utilização de Memória *Cache* em Arquiteturas Paralelas

Com a mesma finalidade de acelerar a busca de instruções e dados que as memórias *cache* têm em sistemas de apenas um processador, em sistemas de múltiplos núcleos a importância das *caches* é ainda maior, dado que quanto maior o número de processadores executando, maior o número de buscas de instruções e dados. Logo, maior o tempo total de espera de dados do sistema, pois nem todas as instruções são paralelizáveis, o que também torna maior o número de ciclos ociosos relativo de cada processador.

O uso de memórias *cache* em sistemas *multicore*, assim como em arquiteturas de processador único, é uma das chaves para o aumento do desempenho desses sistemas. Uma *thread* que estiver executando em um dos processadores pode acessar com maior rapidez um dado que esteja na *cache*, sem deixar este processador ocioso aguardando o dado ser acessado na memória principal, e sem a necessidade de ser escalonada e perder o processador. Isto garante que um maior número de processadores estejam executando a cada ciclo, e portanto um maior número de *threads* também executem. Aspinall (1990) afirma que a memória *cache* de um sistema multiprocessador pode prover um nível de paralelismo transparente às aplicações.

Quando utiliza-se uma hierarquia de memória *cache* em sistemas multinúcleos, principalmente no caso de compartilhamento de módulos de *cache* por dois ou mais processadores, faz-se necessária a implementação de algum tipo de técnica que garanta a coerência dos dados. Existem diversos protocolos de coerência de *cache*, subdivididos em dois grupos: os protocolos de invalidação e os protocolos de atualização. Os dois

grupos diferenciam-se basicamente na política adotada para escritas na *cache*. Um dos protocolos de coerência mais difundidos, sobretudo em computadores pessoais, é o protocolo MESI (IVANOV; NUNNA, 2001), que possui política de invalidação e determina a cada linha da *cache* um dos quatro seguintes estados: *Modified*, *Exclusive*, *Shared* e *Invalid*. O protocolo MESI é implementado pelo Simics em suas simulações, e desta feita, será o protocolo de coerência utilizado neste trabalho.

Dada a grande importância que as *caches* possuem em sistemas de múltiplos processadores, faz-se relevante o estudo do impacto que estes componentes têm sobre o desempenho dos computadores e como elas podem ser implementadas. No próximo capítulo serão apresentados modelos de dois e quatro núcleos da Intel e da AMD, dois dos maiores fabricantes de processadores na atualidade, e como estes utilizam o subterfúgio das memórias *cache* para aumentar o desempenho de suas arquiteturas.

3 ESTUDO DAS ARQUITETURAS

O intuito deste trabalho é a comparação de modelos e desempenho de arquiteturas paralelas de máquinas comerciais, facilmente encontradas no mercado e de preço bastante acessível, através da simulação das mesmas. É interessante ver as soluções empregadas em cada modelo, e notar as diferenças existentes entre modelos "equivalentes" de fabricantes distintos. Cada fabricante investe muito tempo e dinheiro em pesquisa, experimentos, profissionais qualificados etc, a fim de garantir ganho de desempenho com relação aos concorrentes, e obviamente atrair o consumidor. O foco deste trabalho será sobre a arquitetura e hierarquia das memórias *cache*, elemento fundamental no processamento paralelo e um dos principais pontos de discordância entre as implementações existentes. Com mais ou menos níveis, com maior ou menor largura, os projetos de memória *cache* têm a finalidade de extrair o maior paralelismo possível e executar um maior número de operações em menor tempo.

O alvo de estudo deste foram arquiteturas paralelas de dois e quatro núcleos da Intel Corporation e da AMD. Da Intel, foram utilizados os modelos:

- T7800, da família Core 2 Duo;
- T9500, da família Core 2 Duo;
- Q6700, da família Core 2 Quad;
- Q9650, da família Core 2 Quad.

Da AMD, os modelos utilizados foram:

- TL-68, da família Turion 64x2;
- ZM-86, da família Turion 64x2;
- 8360SE, da família Opteron x4;
- 9950, da família Phenom x4.

Mesmo sendo baseados no padrão x86 de 64 *bits*, e portanto com arquiteturas compatíveis, há diversas diferenças entre as implementações dos processadores dos dois fabricantes em questão. Uma das maiores disjunções é quanto à modelagem das memórias *cache*, que como já foi dito, tem papel fundamental nas máquinas multiprocessadas. A Intel mantém uma arquitetura de *cache* mais simples, trabalhando em um compartilhamento de dados eficiente em uma memória geralmente maior, minimizando o problema de coerência de *cache*. Já a AMD implementa uma arquitetura mais complexa e individualizada para cada núcleo, com memórias privadas e maior número de níveis, para diminuir as altas latências associadas aos acessos à memória principal.

3.1 Família Intel Core 2 Duo

A família Core 2, lançada em 2006, foi a sucessora dos consolidados processadores Pentium 4 (HINTON et al., 2001), que eram os modelos topo de linha da Intel desde 2000. Com arquitetura de 64 bits, os processadores Core 2 Duo existem em modelos cujas freqüências variam entre 1,8GHz a 3,3GHz para computadores de mesa e entre 1,6GHz a 2,6GHz para computadores portáteis (LENOVO GROUP LTD., 2008). Para aumentar o desempenho e diminuir o consumo de energia (principalmente nos modelos portáteis), foram implementadas nesta família algumas características relacionadas a acesso à memória (principal e *cache*) e execução paralela, como *Wide Dynamic Execution*, *Intelligent Power Capability* e *Smart Memory Access*, detalhadas por Wechsler (2006).

Este modelo de dois núcleos da Intel utiliza memórias *cache* de nível 1 específicas para cada processador, e uma *cache* de nível 2 compartilhada entre os dois processadores. A Intel implementa um sistema de *cache* chamado *Advanced Smart Cache* (WECHSLER, 2006), que segundo a empresa oferece um compartilhamento dos dados mais eficiente que outros modelos semelhantes, o que resultaria num maior desempenho com menor consumo de energia. Além disso, com uma *cache* L2 única, a necessidade de técnicas para manter a coerência de *cache* é menor do que utilizando *caches* L2 distintas. Também há o fato de que, quando um dos núcleos estiver desativado ou utilizando um mínimo espaço da *cache*, o núcleo em funcionamento pode acessar toda a *cache* de nível 2, e não apenas a metade que lhe seria reservada.

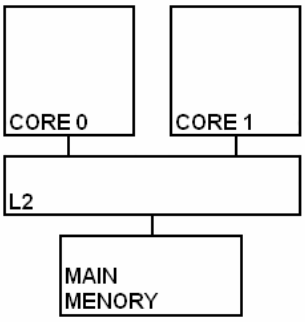

Figura 3.1: Arquitetura Intel Core 2 Duo

Serão utilizados neste trabalho dois modelos de processadores para computadores portáteis da família Core 2 Duo: T7800 e T9500. A escolha por modelos portáteis deve-se ao fato dos modelos equivalentes da AMD que serão utilizados também serem projetados para *laptops*. Para fins de comparação de desempenho de memória *cache*, não haveria a necessidade de avaliar apenas modelos portáteis, dado que a hierarquia de *cache* é a mesma tanto em *laptops* quanto em *desktops*.

3.1.1 Modelo T7800 (LENOVO GROUP LTD., 2008; INTEL CORP., 2008a)

Lançado em julho de 2006, possui o núcleo Merom. Com 65nm e dissipação aproximada de 35W TDP, cada um de seus dois processadores possui uma memória *cache* nível 1 de 64KB (32KB para *cache* de instruções e 32KB para *cache* de dados), com 4 vias de associatividade. Há uma *cache* nível 2 de 4MB, de associatividade de 8 vias. Possui freqüência de 2,6GHz e barramento de 800MHz.

3.1.2 Modelo T9500 (LENOVO GROUP LTD., 2008; INTEL CORP., 2008b)

Lançado em julho de 2008, com o núcleo Penryn. Possui tecnologia de 45nm e consumo de 35W TDP. Também possui uma memória *cache* nível 1 de 64KB (32KB para *cache* de instruções e 32KB para *cache* de dados), com 4 vias de associatividade, para cada processador, mais uma *cache* nível 2 de 6MB, de associatividade de 12 vias. Freqüência de 2,6GHz e barramento de 800MHz.

Tabela 3.1: Configurações dos modelos Intel Core 2 Duo

	T7800	T9500
Núcleo	Merom	Penryn
Tecnologia	65nm	45nm
Consumo TDP	35W	35W
Freqüência	2,6GHz	2,6GHz
Barramento	800MHz	800MHz
Tamanho I-cache L1	32KB (por núcleo)	32KB (por núcleo)
Tamanho D-cache L1	32KB (por núcleo)	32KB (por núcleo)
Associatividade L1	4 vias	4 vias
Tamanho cache L2	4MB (compartilhada)	6MB (compartilhada)
Associatividade L2	8 vias	12 vias

3.2 Família AMD Turion 64x2

Lançados em maio de 2006, os processadores Turion 64x2 (AMD INC., 2006) compõem a primeira família de multiprocessadores para computadores portáteis da AMD. Como a própria nomenclatura sugere, possui arquitetura de 64 *bits*. Os primeiros modelos desta família usam 90nm, mas os mais recentes possuem 65nm. As freqüências dos modelos variam entre 1,6GHz e 2,4GHz. Utiliza a tecnologia *PowerNow!* (AMD INC., 2007a) de gerenciamento de consumo de energia, que varia dinamicamente o consumo em relação ao desempenho requerido pelas aplicações em execução. Além disso, possui implementada a tecnologia *HyperTransport* (AMD INC., 2002) de aumento de desempenho em operações de E/S, reduzindo a latência através de uma comunicação mais eficiente entre processador e outros módulos via barramento com comunicação direta.

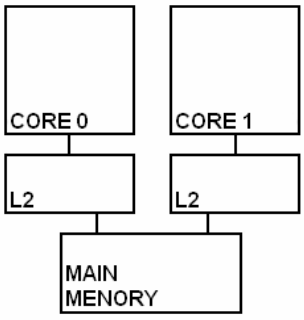

Figura 3.2: Arquitetura AMD Turion 64x2

A hierarquia de *cache* se dá da seguinte maneira: cada processador tem suas memórias *cache* nível 1 e nível 2 privadas, sem acesso compartilhado entre os núcleos (AMD INC., 2007b). Essa arquitetura mais individualizada tem o objetivo de evitar a busca de dados na memória principal, obtendo a maior taxa de *cache hit* possível. Por possuírem *caches* distintas, cada processador terá o mais próximo de si apenas os dados que ele realmente estiver utilizando, o que também diminui o problema de um dado

recém excluído da *cache* ser novamente requerido. Esta proposta de arquitetura tem alguns pontos negativos, como a provável redundância de dados entre as *caches* de nível 2 distintas.

Neste trabalho, serão avaliados os modelos TL-68 e ZM-86, ambos processadores para computadores portáteis. Da mesma forma que os modelos da Intel, as hierarquias de memória *cache* dos modelos *desktop* e *laptop* são as mesmas.

3.2.1 Modelo TL-68 (AMD INC., 2007b; AMD INC., 2008a)

Composto pelo núcleo Tyler, de agosto de 2007, tem tecnologia de 65nm e consumo TDP de 35W. Cada núcleo possui uma *cache* nível 1 de 128KB (64KB de *cache* de instruções e 64KB de *cache* de dados), com associatividade de 2 vias. Cada uma das duas *caches* nível 2 privadas (uma para cada núcleo) possui 512KB de tamanho total e associatividade de 16 vias. Este modelo possui freqüência de *clock* de 2,4GHz e barramento de 800MHz.

3.2.2 Modelo ZM-86 (AMD INC., 2007b; AMD INC., 2008b)

Possui núcleo Lion, de junho de 2008. Com 65nm, consome 35W TDP. Uma *cache* nível 1 de 128KB (64KB de *cache* de instruções e 64KB de *cache* de dados), de associatividade 2, e uma *cache* nível 2 de 1MB, de associatividade 16, por núcleo. Freqüência de 2,4GHz e barramento de 800MHz.

Tabela 3.2: Configurações dos modelos AMD Turion 64x2

	TL-68	ZM-86
Núcleo	Tyler	Lion
Tecnologia	65nm	65nm
Consumo TDP	35W	35W
Freqüência	2,4GHz	2,4GHz
Barramento	800MHz	800MHz
Tamanho I-cache L1	64KB (por núcleo)	64KB (por núcleo)
Tamanho D-cache L1	64KB (por núcleo)	64KB (por núcleo)
Associatividade L1	2 vias	2 vias
Tamanho cache L2	512KB (por núcleo)	1MB (por núcleo)
Associatividade L2	16 vias	16 vias

3.3 Família Intel Core 2 Quad

Família de computadores com quatro núcleos da Intel Corporation, utilizados principalmente em estações de trabalho e servidores (INTEL CORP., 2007a). Com grande poder de processamento, possui modelos com freqüências que variam entre 2,4GHz a 3GHz cada núcleo (LENOVO GROUP LTD., 2008). Possui algumas características em comum à família Core 2 Duo, como a *Wide Dynamic Execution* e a *Advanced Smart Cache*, mas possui uma característica nova chamada *Digital Thermal Sensor*, que oferece controle contínuo de temperatura e ruído dos núcleos.

Os modelos da família Intel Core 2 Quad possuem uma hierarquia de memória *cache* diferenciada: uma memória *cache* nível 1 privada para cada núcleo e duas memórias *cache* nível 2, compartilhadas por cada par de nodos, que podem ser alocadas de qualquer maneira para cada núcleo dentro da sua arquitetura, dependendo da carga de trabalho requerida por cada um (INTEL CORP., 2007a). De forma análoga ao que ocorre na família Core 2 Duo, esse compartilhamento da *cache* – neste caso, par a par –

necessita de menor controle de coerência de *cache* e diminui as chances de uma *cache* ser pouco utilizada por ser específica de um núcleo que esteja ocioso. Por serem, em geral, bastante grandes, estas memórias *cache* nível 2 são bastante eficientes para este compartilhamento (WECHSLER, 2006). Serão utilizados neste trabalho os modelos Q6700 e Q9650 da família Core 2 Quad.

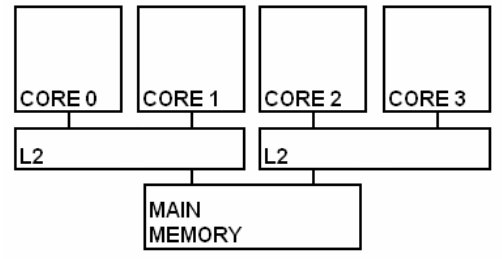

Figura 3.3: Arquitetura Intel Core 2 Quad

3.3.1 Modelo Q6700 (LENOVO GROUP LTD., 2008; INTEL CORP., 2007b)

Este modelo possui o núcleo Kentsfield, produzido em janeiro de 2007. Composto de 65nm, possui consumo médio de 105W TDP. Cada núcleo possui uma *cache* nível 1 de 64KB (32KB para *cache* de instruções e 32KB para *cache* de dados), com 8 vias de associatividade. Cada par de núcleos compartilha um módulo de memória *cache* de nível 2 de 4MB, com associatividade de 16 vias. A freqüência deste modelo é de 2,66GHz e o barramento é de 1066MHz.

3.3.2 Modelo Q9650 (LENOVO GROUP LTD., 2008; INTEL CORP., 2008c)

Modelo bastante recente, possui o núcleo Yorkfield. Tecnologia de 45nm e consumo TDP de 95W. Os núcleos possuem, cada um, uma memória *cache* nível 1 privada de 64KB (32KB para *cache* de instruções e 32KB para *cache* de dados), com 8 vias de associatividade. Da mesma forma que no modelo anterior, cada par de núcleos compartilha uma memória *cache* de nível 2, mas neste caso de 6MB e associatividade 12. Freqüência de *clock* de 3GHz e barramento de 1333MHz.

Tabela 3.3: Configurações dos modelos Intel Core 2 Quad

	Q6700	Q9650
Núcleo	Kentsfield	Yorkfield
Tecnologia	65nm	45nm
Consumo TDP	105W	95W
Freqüência	2,66GHz	3,0GHz
Barramento	1066MHz	1333MHz
Tamanho I-cache L1	32KB (por núcleo)	32KB (por núcleo)
Tamanho D-cache L1	32KB (por núcleo)	32KB (por núcleo)
Associatividade L1	8 vias	8 vias
Tamanho cache L2	4MB (comp. por par)	6MB (comp. por par)
Associatividade L2	16 vias	12 vias

3.4 Família AMD Opteron x4

Primeira família de processadores de quatro núcleos da AMD, assim como os modelos equivalentes da Intel, tem sua maior utilização em servidores e estações de trabalho. Teve modelos de 90nm e atualmente de 65nm (AMD INC., 2005). Assim como outras famílias da AMD, possui as tecnologias *PowerNow!* (AMD INC., 2007a) e

HyperTransport (AMD INC., 2002), de controle de consumo de energia e eficiência de operações de E/S, respectivamente.

A grande diferença desta família de quatro núcleos é na sua hierarquia de memória *cache*. Ela possui uma *cache* nível 1 e nível 2 privadas para cada núcleo, e uma *cache* nível 3 compartilhada entre os quatro núcleos (AMD INC., 2008c). Esta arquitetura prova como realmente a idéia chave da AMD é ter a maior taxa possível de *cache hit*, levando esta hierarquia até o terceiro nível. Além disso, possui uma característica chamada *Memory Optimizer Technology* (AMD INC., 2007c), que oferece tratamento especial à memória em ambiente de execução multitarefa.

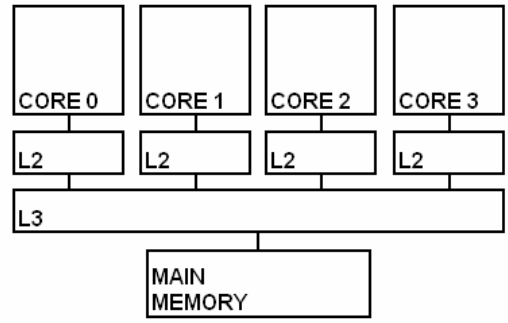

Figura 3.4: Arquitetura AMD Opteron x4 e AMD Phenom x4

3.4.1 Modelo 8360 SE (AMD INC., 2008c; AMD INC., 2008d)

Um dos últimos modelos lançados pela AMD, de abril de 2008, possui o núcleo Barcelona, com 65nm e consumo TDP de 119W. Sua hierarquia de memória *cache* se dá da seguinte maneira: cada núcleo possui uma *cache* nível 1 de 128KB (64KB de *cache* de instruções e 64KB de *cache* de dados), com associatividade 2, e uma *cache* nível 2 de 512KB, de associatividade 16, privadas. Ainda existe uma *cache* nível 3 compartilhada entre os quatro núcleos, de 2MB e associatividade de 32 vias. O *clock* deste modelo tem freqüência de 2,5GHz e o barramento é de 1066MHz.

Tabela 3.4: Configuração do modelo AMD Opteron x4

	Opteron 8360SE
Núcleo	Barcelona
Tecnologia	65nm
Consumo TDP	119W
Freqüência	2,5GHz
Barramento	1066MHz
Tamanho I-cache L1	64KB (por núcleo)
Tamanho D-cache L1	64KB (por núcleo)
Associatividade L1	2 vias
Tamanho cache L2	512KB (por núcleo)
Associatividade L2	16 vias
Tamanho cache L3	2MB (compartilhada)
Associatividade L3	32 vias

3.5 Família AMD Phenom x4

A família Phenom foi a primeira lançada pela AMD unicamente com processadores *multicore*. Contando com representantes de três e quatro núcleos, seus

processadores são mais voltados para modelos *desktop* (AMD INC., 2008e). Suas características são muito próximas às da família Opteron x4.

Sua hierarquia de memória *cache* também é bastante complexa. Para cada núcleo são reservadas uma *cache* nível 1 e uma de nível 2, e ainda há uma *cache* nível 3 compartilhada entre os quatro núcleos (AMD INC., 2007d). Como é possível notar, esta arquitetura é a tendência da AMD no mercado de processadores de quatro núcleos. A Figura 3.4 também representa a arquitetura da família Phenom x4.

3.5.1 Modelo 9950 (AMD INC., 2007d; AMD INC., 2008f)

Muito semelhante ao modelo 8360SE da família Opteron x4. Possui o núcleo Agena, de março de 2008. Tecnologia de 65nm e consumo médio de 140W TDP. Cada núcleo possui uma memória *cache* nível 1 de 128KB (64KB de *cache* de instruções e 64KB de *cache* de dados), de associatividade de 2 vias, e uma *cache* nível 2 de 512KB, de associatividade de 16 vias. Compartilhada entre os núcleos, existe uma *cache* nível 3 de 2MB e associatividade 32. O modelo executa em 2,6GHz de freqüência e possui barramento de 1066MHz.

Tabela 3.5: Configuração do modelo AMD Phenom x4

	Phenom 9950
Núcleo	Agena
Tecnologia	65nm
Consumo TDP	140W
Freqüência	2,6GHz
Barramento	1066MHz
Tamanho I-cache L1	64KB (por núcleo)
Tamanho D-cache L1	64KB (por núcleo)
Associatividade L1	2 vias
Tamanho cache L2	512KB (por núcleo)
Associatividade L2	16 vias
Tamanho cache L3	2MB (compartilhada)
Associatividade L3	32 vias

4 METODOLOGIA

Neste capítulo serão apresentadas as ferramentas utilizadas para a simulação e testes de desempenho realizados. Primeiramente é abordado o ambiente de simulação Simics, cerne dos experimentos deste trabalho, que permite a modelagem das arquiteturas descritas e suas simulações. Na seção seguinte é apresentado o conjunto de aplicativos de testes PARSEC, responsável pela carga de trabalho aplicada na simulação das arquiteturas, permitindo a análise do desempenho das mesmas. Ao fim, apresenta-se a ferramenta CACTI, que fornece dados importantes das *caches*, como a latência dos módulos, utilizada neste trabalho.

4.1 Simics

O Simics é uma plataforma de desenvolvimento, modelagem e simulação de arquiteturas completas de microcomputadores (VIRTUTECH AB, 2008a), produzido pela Virtutech AB, corporação de *softwares* e sistemas embarcados fundada em 1998 no Instituto de Ciência da Computação da Suécia, em Estocolmo. Esta ferramenta, carro-chefe da empresa, provê *hardware* virtual, que executa os mesmos binários de um *hardware* físico. Desta forma, é possível a partir de uma máquina qualquer, desenvolver *software*, realizar simulações e testar desempenho de uma arquitetura virtual configurável. Isto reduz custo e agiliza o processo de desenvolvimento, dado que não é necessário ter o *hardware* físico para realizar estas tarefas, utilizando o ambiente simulado do *hardware*.

Por se tratar de um simulador *full-system*, toda a arquitetura modelada no Simics é programável, como memórias, processadores, conjunto de instruções etc, como afirmam Engblom e Ekblom (2006). Além disso, o pacote original do programa já inclui diversos *targets*, arquiteturas programáveis prontas, como x86, UltraSparc, PowerPc, entre outros, que podem ser modificadas. Com isto, dependendo do que o desenvolvedor necessita simular, é possível modelar uma arquitetura do zero ou alterar algum *target* para construir a arquitetura desejada. Para uma modelagem mais precisa, é possível integrar o Simics a ferramentas próprias para modelagem de memória (Ruby), processador (Opal) e utilizar diretamente linguagens de descrição de *hardware* como VHDL e Verilog (MAGNUSSON et al., 2002).

Além disso, é possível, após modelada uma arquitetura qualquer, instalar um sistema operacional sobre esta arquitetura virtual. O Simics oferece também ferramentas de teste de *software* e ambientes de *debug* determinístico, essencial para desenvolvedores. A ferramenta suporta geração de *checkpointing*, onde o estado completo do sistema, incluindo memórias voláteis, é gravado em disco para ser recuperado futuramente. Também há um interpretador Python na ferramenta. Outros

compiladores, interpretadores e analisadores podem ser utilizados sobre o Simics, como C/C++, Yacc e Lex, utilizando outras ferramentas aliadas, como o Cygwin e o MinGW.

A plataforma Simics é muito interessante e útil, pois uma arquitetura modelada pelo usuário é executada rapidamente como um *script*, que altera a configuração do *hardware* virtual da ferramenta. Há mais de um modo de execução, sendo os mais comuns o modo *normal* (nomenclatura na versão Windows; na versão Linux chama-se *fast*) e o modo *stall*. No modo *normal*, além de ter execução mais rápida devido emulação direta do sistema, as memórias *cache* são desativadas e as latências ignoradas, o que torna a execução otimizada. Além disso, para acelerar a execução, é possível utilizar um mecanismo chamado *Simulator Translation Cache*, que armazena traduções de endereço da *cache* para acesso rápido, funcionando como uma espécie de TLB para a *cache* (MAGNUSSON; WERNER, 1995). O modo *normal* é aconselhado para instalação de SO e *software*, compilação de código e *debug*. Já o modo *stall*, onde as *caches* e latências são incorporadas à execução, é utilizado principalmente em atividades de medição de desempenho e avaliação de arquiteturas.

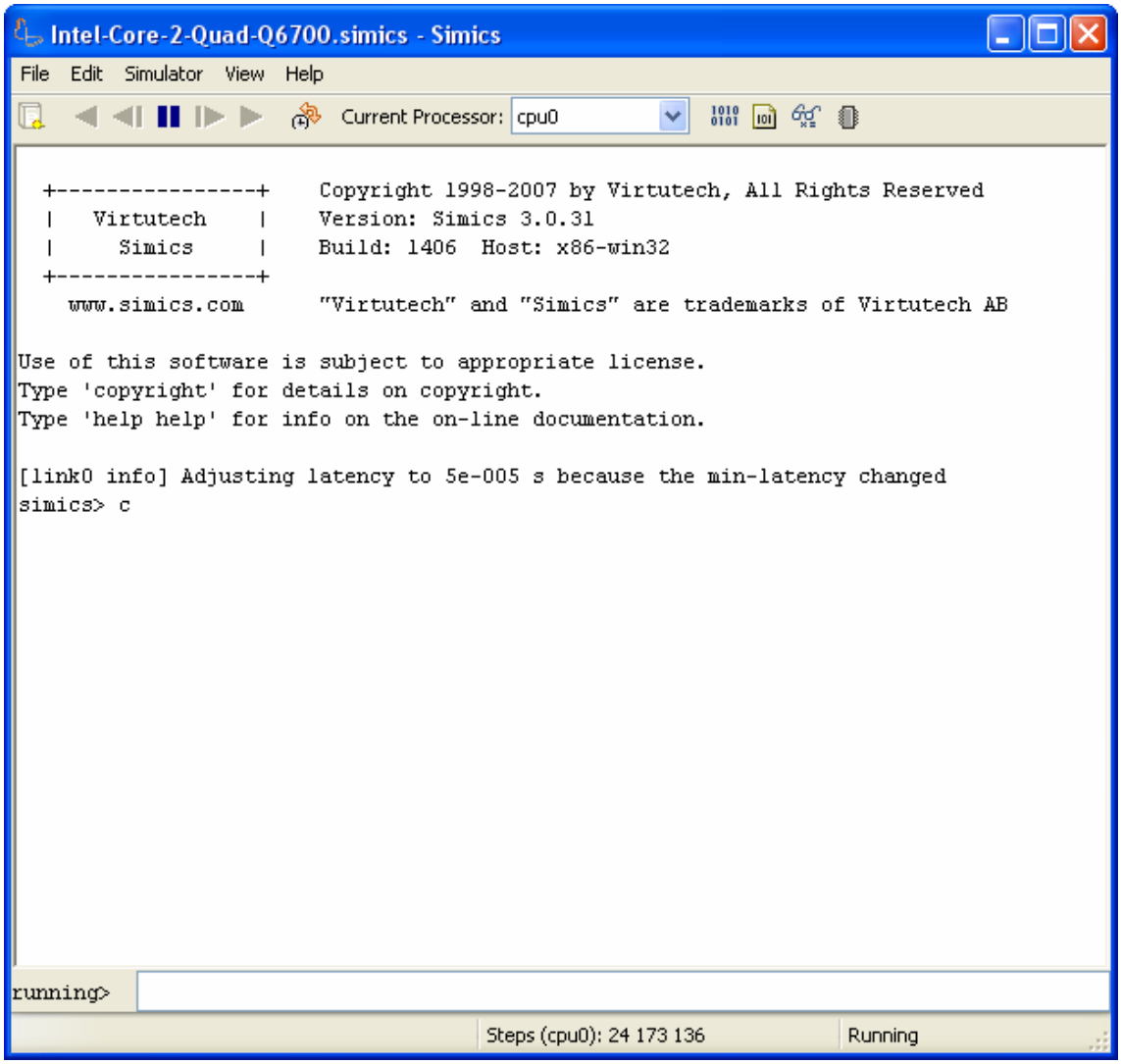

Figura 4.1: Simulador Simics

Por esses motivos, o Simics é uma das ferramentas utilizadas por empresas como Intel, AMD, IBM, Sun e outros fabricantes para realizar simulações e testes de

suas máquinas (VIRTUTECH AB, 2008b). Além disso, algumas destas empresas fornecem aos programadores os *scripts* de novas arquiteturas que estão sendo ainda desenvolvidas, para que estes possam trabalhar com antecedência em programas e ferramentas que irão executar nestas novas arquiteturas.

O Simics foi escolhido como ferramenta deste trabalho justamente por todas as funções que ele provê para modelar e simular as arquiteturas estudadas. Para utilizá-lo foi necessário fazer um cadastro junto a Virtutech, utilizando um endereço de correio eletrônico acadêmico (neste caso, o da UFRGS), e foi encaminhado um pedido de licença acadêmica para utilização da ferramenta. Esta licença dá direito de utilização a um usuário em uma única máquina pré-determinados.

Na primeira etapa do trabalho, houve um estudo das funcionalidades básicas do Simics, como carregar *targets*, modelar e simular as arquiteturas desejadas, criar *checkpoints* etc. Na segunda etapa, foi utilizado um SO sobre as arquiteturas virtuais e testes foram realizados com cargas específicas de trabalho, a fim de determinar as vantagens e desvantagens de cada solução. Para testar o desempenho das arquiteturas foi utilizado o PARSEC, que será explicado em maiores detalhes na próxima seção.

4.2 PARSEC

O PARSEC é um conjunto de *benchmarks* para estudo e avaliação de multiprocessadores, em geral SMPs. Desenvolvido em parceria pela Universidade de Princeton e a Intel Corporation, tem o objetivo de ser uma ferramenta que possa ser usada na construção das próximas gerações de processadores. Para que a avaliação seja relevante, os programas do conjunto possuem características como representar aplicações importantes nas máquinas testadas e utilizar algoritmos e técnicas muito bem fundamentadas e utilizadas na área científica, acadêmica e comercial. Além disso, o conjunto é bastante diverso, possuindo uma gama de aplicações distintas, sendo possível verificar os mais variados comportamentos da máquina em cada aplicação. Afirma-se que os futuros processadores serão desenvolvidos para lidar com cargas de trabalho como estas aplicações, consideradas emergentes. Dessa forma, o conjunto de *benchmarks* deve representar estas demandas de trabalho como um todo (BIENIA et al., 2008).

Composto por doze programas distintos para avaliação, o PARSEC consegue ser altamente abrangente, justamente pela diversidade das aplicações. Os *benchmarks* variam entre cálculo de equações diferenciais, mapeamento e tratamento de imagens, minimização de rotas, busca por similaridade, compressão de vídeo, entre outros. Além disso, são definidos seis conjuntos de entradas para cada *benchmark*, que vão de entradas muito pequenas, apenas para teste (*test* e *simdev*), entradas para simulação (*simsmall*, *simmedium* e *simlarge*), que variam de tamanho e nível de paralelismo, e uma entrada extremamente grande, para execuções em máquinas reais (*native*).

O PARSEC possui um aplicativo chamado *PARSEC Management*, com o qual é possível compilar e executar facilmente todos os *benchmarks*, a partir do mesmo diretório e modificando apenas os parâmetros como *benchmark*, compilador a ser utilizado, tamanho da entrada e número de *threads* a executarem. Também é possível compilar ou executar diversos *benchmarks* em apenas um comando, sendo cada um executado imediatamente após o término da execução do anterior.

Para a compilação dos códigos fonte, programados em C++, o PARSEC utiliza o compilador GCC, mas permitindo que se escolha entre quatro modelos de compilação,

para a construção das seguintes versões dos *benchmarks*: paralela, serial, paralela com utilização de *hooks* e paralela compilada com o ICC (versão da Intel do compilador GCC). O PARSEC *hooks* é uma biblioteca de instrumentação do conjunto de *benchmarks*. Ao serem compilados utilizando a opção de *hooks*, em diversos pontos do código, os programas chamarão as funções da biblioteca. Desta forma, é possível modificar funções dos *benchmarks* a partir da biblioteca *hooks*. Além disso, uma mesma função que se queira implementar em todos os *benchmarks*, pode ser codificada na biblioteca e utilizada em todas aplicações.

Tabela 4.1: Comparação do PARSEC com outros conjuntos de *benchmarks*

	Programs	Multi-threaded	Emerging Workloads	Diverse	Not HPC-Focused	Research
PARSEC	12	✓	✓	✓	✓	✓
SPEC CPU2006	29	✗	✗	✓	✗	✗
SPEC OMP2001	11	✓	✗	✓	✗	✗
SPLASH-2	14	✓	✗	✓	✗	✓
ALPBench	5	✓	✓	✗	✓	✓
BioBench	7	✗	✗	✗	✗	✓
BioParallel	5	✓	✗	✗	✗	✓
MediaBench II	12	✗	✗	✗	✓	✓
NU-Minebench 2.0	15	✓	✗	✗	✓	✓
PhysicsBench	8	✓	✓	✗	✓	✓

Fonte: PRINCETON UNIVERSITY, 2008

Em cada execução, é possível determinar o número de *threads* a serem criadas e utilizadas em cada aplicação. Pode-se executar qualquer número de *threads*, mas em geral utiliza-se valores que sejam potências de dois. Não há relação direta entre o número de *threads* com o número de processadores existentes no modelo, apesar de ser recomendado a utilização de um número de *threads* maior ou igual à quantidade de núcleos do sistema.

Considerado um dos mais completos conjuntos de *benchmarks* na atualidade, é largamente utilizado no meio acadêmico e começa a se inserir no meio industrial, no desenvolvimento de processadores. Um estudo de Bienia, Kumar e Li (2008) realiza uma comparação entre o PARSEC e o difundido SPLASH-2 (WOO et al., 1995) em multiprocessadores e comprova que o PARSEC é mais abrangente e diverso, conseguindo realizar análises mais detalhadas sobre o desempenho dos sistemas como um todo. Além disso, os aplicativos do SPLASH-2, por ser um conjunto de *benchmarks* mais antigo, já tendem a uma certa obsolescência, não conseguindo obter resultados tão representativos das máquinas mais recentes quanto o PARSEC.

4.3 CACTI

O CACTI é uma ferramenta de modelagem e organização de sistemas de memória e de *cache* (THOZIYOOR et al., 2008). A partir de informações sobre o tamanho do módulo, tamanho de linha, associatividade e tecnologia de fabricação, a

ferramenta fornece dados como o número de ciclos de latência de acesso, área total, consumo de energia, entre outras informações.

Neste trabalho, a utilização do CACTI deveu-se à necessidade de calcular os ciclos de latência das *caches* de diversos níveis de cada arquitetura. Como estas informações não são fornecidas pelos fabricantes, fez-se uso do CACTI para a obtenção destes dados. A partir desta utilização, foi possível chegar aos valores de latência, apresentados na Tabela 4.2, para cada arquitetura trabalhada.

Tabela 4.2: Número de ciclos de latência de *cache* das arquiteturas trabalhadas

	***Cache* L1**	***Cache* L2**	***Cache* L3**
Intel Core 2 Duo T7800	4	11	-
Intel Core 2 Duo T9500	4	9	-
AMD Turion 64x2 TL-68	3	9	-
AMD Turion 64x2 ZM-86	3	10	-
Intel Core 2 Quad Q6700	6	13	-
Intel Core 2 Quad Q9650	6	10	-
AMD Opteron x4 8360SE	3	9	16
AMD Phenom x4 9950	3	10	17

O CACTI encontra-se atualmente na sua versão 5.3, e é mantido pelos laboratórios de pesquisa e desenvolvimento da HP Company, estando disponível em (HEWLETT-PACKARD COMPANY, 2008).

5 EXPERIMENTO

5.1 Modelagem

Ao utilizar a ferramenta Simics, é possível notar como é tratada a modelagem das arquiteturas a serem simuladas. A partir de um *script*, ou seja, um código que descreve os componentes da arquitetura, o Simics virtualiza a máquina descrita, trabalhando em nível de instrução, como explica Magnusson e colaboradores (2002), com a finalidade de manter resultados fiéis aos que seriam obtidos na máquina verdadeira. O Simics já traz em seu pacote original um grande conjunto de *scripts* que descrevem diversos tipos de componentes como processadores, memórias, placas gráficas etc, além de arquiteturas completas descritas, utilizando os componentes citados, chamadas *targets*.

Como o foco deste trabalho é o desempenho das memórias *cache* relativas às arquiteturas estudadas, a modelagem se restringiu basicamente à hierarquia de *cache* e às ligações existentes entre os diferentes níveis da memória. Outros componentes, como processadores por exemplo, foram utilizados os modelos padrões, sem modificações nos mesmos. As arquiteturas foram modeladas utilizando como base o *script* 'dredd-common', incluso no pacote padrão do Simics. Este *script* é utilizado para a execução de SOs instalados pelo usuário, através de outros dois *scripts*, também originais do simulador e baseados no primeiro, que realizam esta instalação. Assim, como há a instalação de um SO, este *script* foi utilizado como alicerce para as simulações.

Na modelagem das arquiteturas deste trabalho, os *scripts* podem ser divididos em basicamente duas partes: uma que determina componentes essenciais para o funcionamento da máquina, e outra que define detalhadamente a arquitetura de *cache*. Dentre os elementos, são descritos o número de processadores, suas freqüências de *clock*, tamanho de memória principal e tamanho de disco rígido. Quando da utilização de um *script* como base, como é o caso deste, o nível de detalhamento na descrição destes componentes da arquitetura é definido pelo projetista. Devido ao foco do trabalho, os modelos referentes às memórias *cache* foram mais detalhados, descrevendo quantos e quais os níveis existentes na hierarquia, e como se comunicam os módulos de *cache* com processadores, com a memória principal e entre os próprios módulos. Sobre cada módulo, características como tamanho de linha, latências de leitura e escrita, associatividade e políticas de escrita e de troca devem ser bem definidas, para o funcionamento do módulo. Dados como tamanho de linha, tamanho total dos módulos e associatividade foram retirados de manuais, *datasheets* e sítios na Internet dos fabricantes, e foram detalhados no capítulo 3. Já os valores para os ciclos de latência foram calculados utilizando a ferramenta CACTI 5.3, como foi explicado no capítulo anterior.

Utilizou-se em todos os *scripts* a freqüência de *clock* de 20MHz. O motivo disto é que o valor de 20MHz é a freqüência padrão do Simics, e portanto é a que torna a simulação mais rápida; acima deste valor, quanto maior a freqüência de *clock* estabelecida, mais lenta será a simulação, como está indicado em (VIRTUTECH AB, 2008). Porém, pelo fato do simulador trabalhar em nível de instruções, e em termos de ciclos de latência e de execução, a freqüência utilizada na modelagem não afeta os resultados, uma vez que os dados obtidos acerca dos ciclos são fiéis aos modelos originais. Assim, o número de transações efetuadas por cada *cache* independe da freqüência do processador. Desta forma, tendo-se o número de ciclos de execução total e o valor da freqüência de *clock* do processador original, é possível calcular o tempo de execução real dos aplicativos. Quanto à classe dos processadores, todas as arquiteturas trabalhadas são de 64 *bits*. Por isto, os modelos Intel foram definidos como da classe Pentium 4E, e os modelos AMD como da classe Athlon 64.

Durante o estudo das arquiteturas e a modelagem das mesmas, nota-se algumas diferenças de projeto entre os modelos Intel e AMD. Comparando as famílias de dois núcleos das duas empresas, os modelos da Intel utilizam *cache* nível 1 de menor tamanho (32KB) mas maior associatividade (4 vias), o que é interessante para aplicações que executam diversas vezes um mesmo conjunto de instruções e/ou sobre um mesmo conjunto de dados. Enquanto isso, os modelos *dual-core* da AMD possuem *cache* nível 1 de 64KB, mas associatividade de apenas 2 vias, diminuindo a latência e sendo bastante eficiente para aplicações que utilizam diversos dados distintos e necessitam consultar a *cache* com freqüência.

Na comparação entre as *caches* de nível 2, o conceito exposto anteriormente inverte-se: os modelos da Intel utilizam um único módulo de *cache* nível 2, de grande tamanho (4MB no T7800 e 6MB no T9500) e menor associatividade (8 vias no T7800 e 12 vias no T9500), enquanto os exemplos da AMD possuem um módulo de *cache* privado para cada núcleo, de menor tamanho (512KB no TL-68 e 1MB no ZM-86) e maior associatividade (16 vias para ambos os modelos). Importante verificar como a associatividade está ligada ao tamanho total da memória: como afirma-se em (VIRTUTECH AB, 2008), o tamanho de linha deve ser uma potência de dois e o número de linhas dividido pela associatividade também deve ser uma potência de dois.

Maior diferença na modelagem entre as famílias Intel e AMD encontra-se nos modelos *quad-core*. Como já foi visto, a Intel proporciona um módulo de *cache* nível 2 para cada par de núcleos, enquanto a AMD possui *caches* nível 2 privadas e uma *cache* nível 3 compartilhada entre os quatro núcleos. Esta abordagem da AMD torna a modelagem mais complexa, devido ao maior número de módulos e, conseqüentemente, maior número de ligações entre os módulos de *cache*. Além disso, a busca de um dado ou instrução que não se encontra em nenhum nível de *cache* levará maior número de ciclos até alcançar a informação na memória principal. Em contrapartida, a modelagem das arquiteturas Intel necessita a implementação de coerência de *cache*, através do protocolo MESI. Para isto é necessário descrever os chamados *snoopers*, elementos que intermediam a comunicação entre módulos de *cache* de mesmo nível, com a finalidade de manter a coerência.

No apêndice deste trabalho encontra-se, para cada arquitetura utilizada, um *script* de modelagem da arquitetura e outro que, dentre algumas funcionalidades, define as latências da mesma, e é utilizado na execução dos testes de desempenho. Este segundo *script* será melhor explicado mais adiante.

5.2 Instalação do Sistema Operacional

Para que se possa realizar a execução do *benchmark* a ser utilizado, é necessário que exista um SO instalado nas máquinas virtuais simuladas. Para este fim, foi escolhido o Ubuntu 6.06.2 Server com suporte a SMP para ser o SO deste trabalho.

A instalação do SO foi realizada com a utilização de dois *scripts* próprios para este fim, originais do pacote do Simics. O *script* 'dredd-cd-install1', ao ser carregado, cria um disco rígido virtual para a inserção de dados a partir de uma unidade de CD virtual. Como unidade de CD virtual utilizou-se uma imagem do CD de instalação do Ubuntu, disponível em (CANONICAL LTD., 2008). Após a leitura do CD virtual, os arquivos de instalação do Ubuntu são copiados para o disco rígido virtual criado. Neste momento, a simulação é interrompida, sendo necessário salvar o estado atual da configuração inteira (disco e memória principal) para reiniciar a instalação com um segundo *script*, o 'dredd-cd-install2'. Com este, os arquivos de instalação que foram copiados para o disco virtual são agora executados e a instalação do Ubuntu ocorre efetivamente, da mesma forma que seria realizada a instalação do SO em uma máquina real. Ao término da instalação e do devido *login* na máquina, realiza-se o desligamento do sistema e o conseqüente término da simulação. Após, é necessário salvar o estado persistente (ou seja, somente as informações em disco), para que em um momento futuro seja possível, a partir de uma nova simulação, carregar o SO instalado neste disco virtual.

5.3 Simulação

Após a modelagem das arquiteturas e a instalação do SO, é possível realizar as simulações das máquinas escolhidas para o trabalho. Para uma simples simulação das arquiteturas, pode-se utilizar tanto o modo *normal* quanto o modo *stall*, previamente explicados no capítulo 4.1. Mas no momento da execução dos testes de desempenho, onde pretende-se averiguar a atividade da memória *cache*, se faz necessária a utilização do segundo modo. Com o intuito de aumentar a velocidade da simulação em modo *normal*, no *script* que descreve a hierarquia de *cache* as latências são definidas com o valor zero.

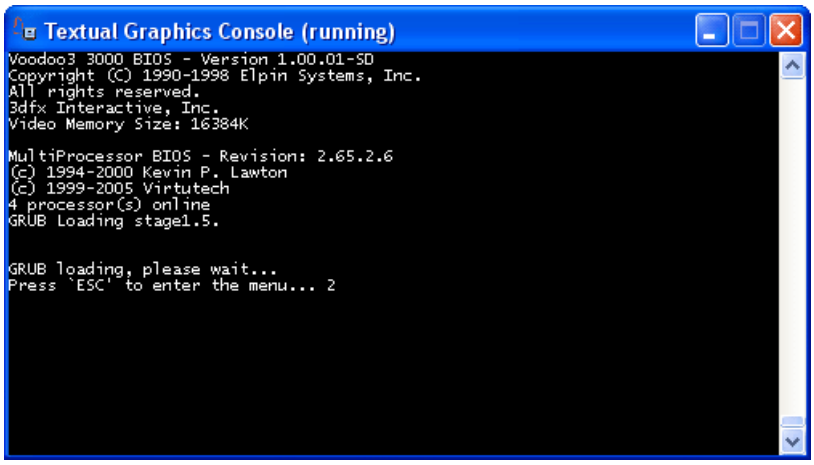

Figura 5.1: Inicialização de sistema com quatro núcleos

Após carregar todas as configurações necessárias do SO e a realização do *login*, tem-se um sistema totalmente operante, equivalente ao mesmo sistema executado em

uma máquina real, sendo possível a instalação e execução de programas, compilação de códigos, criação e conexão de uma rede etc. A Figura 5.2 mostra o resultado de um comando que retorna informações sobre os processadores existentes no sistema, no caso, um AMD Turion 64x2.

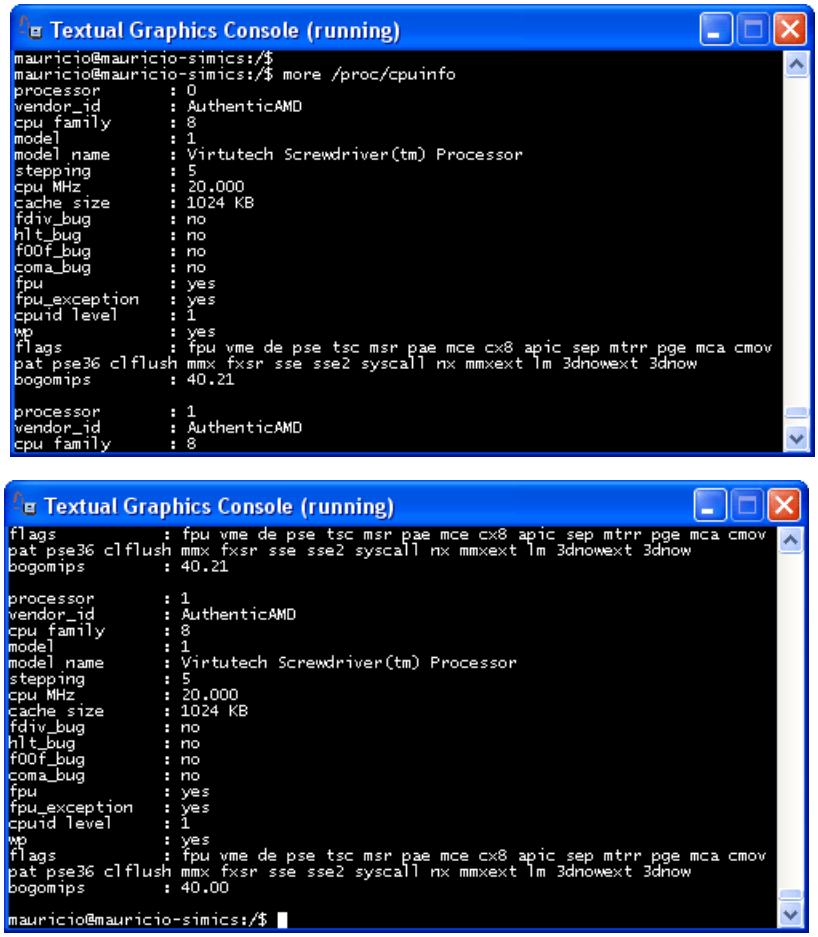

Figura 5.2: Informações sobre os processadores da arquitetura AMD Turion 64x2, detalhando (a) o primeiro e (b) o segundo núcleos

A partir do momento em que a simulação executa de forma correta, com o SO instalado e identificando processadores e memória *cache*, é possível executar os testes de desempenho. Nas próximas seções será explicado como foram realizados estes testes e a forma que os resultados foram obtidos.

5.4 Teste de Desempenho

Para a realização dos testes de desempenho, foi necessário copiar os arquivos fonte e arquivos de entradas do PARSEC para o disco virtual utilizado pelas arquiteturas. Uma vez copiados, realizou-se a compilação dos códigos fonte de cada um dos aplicativos que seriam utilizados. Dos doze *benchmarks* inclusos no conjunto do PARSEC, sete deles foram escolhidos para serem executados a fim de medir o desempenho e funcionalidade das memórias *cache* das arquiteturas. Os sete programas utilizados foram escolhidos devido à importância e abrangência de suas aplicações e por representarem áreas bastante distintas do estudo da computação. Os *benchmarks* utilizados foram os seguintes:

- **Blackscholes**: Desenvolvido pela Intel Corporation, utiliza modelos de equação diferencial parcial a partir do método de Black-Scholes. Como esta expressão não possui uma fórmula fechada, deve ser numericamente computada a cada execução.
- **Bodytrack**: Também desenvolvido pela Intel, este aplicativo modela um corpo humano utilizando múltiplas câmeras, através de uma seqüência de imagens. Utiliza técnicas de análise de vídeo e animação de objetos.
- **Canneal**: Desenvolvido pela Universidade de Princeton, utiliza a heurística de *simulated annealing* para minimizar o custo da rota de ligações em uma placa. Utiliza estratégia de sincronização baseada em recuperação de seqüência de dados.
- **Fluidanimate**: Aplicação da Intel, utiliza um método para simular o comportamento de fluidos. Modela significativas simulações físicas para animação.
- **Streamcluster**: Aplicação da Universidade de Princeton para resolver problemas de agrupamento e classificação de objetos. Possui técnicas de mineração de dados.
- **Swaptions**: Desenvolvido pela Intel, realiza cálculos financeiros de estatísticas de ganhos, utilizando o método Monte Carlo para calcular os valores.
- **x264**: Este aplicativo é um codificador de vídeo do tipo H.264/AVC. O H.264 é o formato de compressão de vídeo com menor perda de qualidade, fazendo parte do padrão MPEG-4. Utiliza técnicas avançadas de compressão de vídeo.

Utilizando o aplicativo *PARSEC Management*, os *benchmarks* selecionados foram compilados utilizando o GCC-Hooks. Após a compilação, utilizando o mesmo aplicativo de gerenciamento, os programas de teste foram executados utilizando quatro *threads* e o pacote de entradas *simsmall*. Apesar de ser o menor dos pacotes de entradas de simulação, ele permite resultados tão precisos quanto os outros pacotes de entradas, e em tempo relativamente curto comparado aos mesmos.

Para realizar as execuções dos testes, primeiramente é necessário interromper a simulação, colocar o simulador em modo *stall* (se já não estiver no mesmo) e executar no simulador o *script* de latências da arquitetura simulada em questão. No *script* de latências estão as informações de ciclos de latência para acesso de cada módulo da *cache*, tanto para leitura quanto para escrita. Além disso, foram inseridos comandos para ativar o monitoramento de busca de instruções por parte das *caches* de instruções de nível 1, para reiniciar as estatísticas de cada módulo da *cache* e para interromper a simulação após o término da execução dos *benchmarks*. Estes comandos garantem a precisão dos resultados obtidos pelos testes. Assim que o *script* de latências é executado e suas informações adicionadas ao modelo, pode-se continuar a simulação e realizar os teste de desempenho. A Figura 5.3 mostra a execução de uns dos *benchmarks* em uma máquina de quatro núcleos.

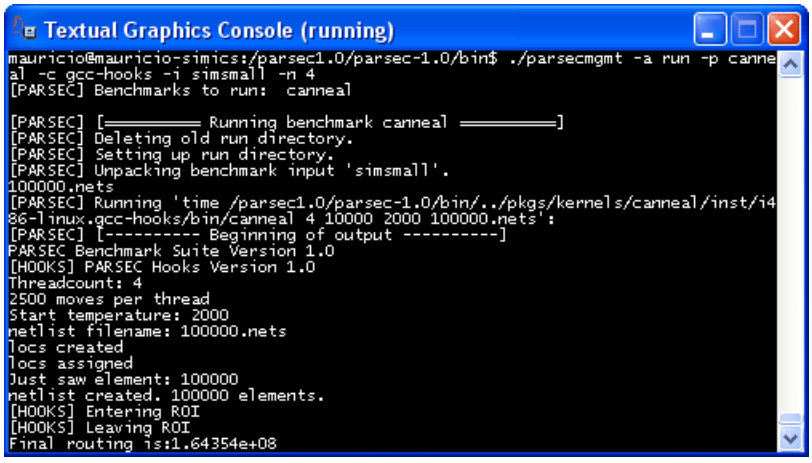

Figura 5.3: Execução do *benckmark* Canneal no Intel Core 2 Quad Q9650

Ao término da execução de qualquer um dos *benchmarks*, o console parecerá com a Figura 5.4, onde são apresentados os tempos gastos para a execução do programa. Como já foi dito, os tempos apresentados no terminal não correspondem aos tempos exatos de execução das máquinas reais, dado que utilizou-se freqüência de *clock* de 20MHz para os processadores. Por este motivo, estes valores não podem ser aproveitados, sendo necessário tomar os tempos de execução a partir do simulador Simics, como é explicado na seção seguinte.

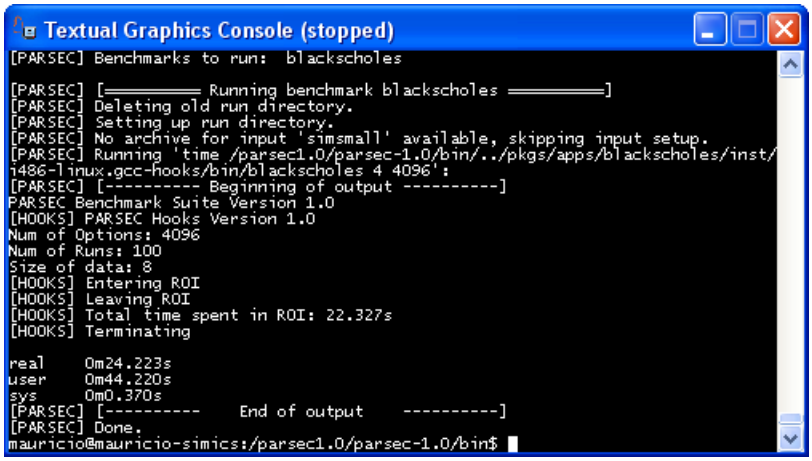

Figura 5.4: Término da execução do *benchmark* Blackscholes

Cada execução de um *benchmark* em uma determinada arquitetura foi realizada da seguinte forma: um primeira execução, para “aquecer” as *caches* e evitar efeitos transientes, e após mais três execuções, a fim de evitar possíveis execuções anômalas, causadas por interferência do SO. Além disso, como explicam Alameldeen e Wood (2003), a variabilidade nos resultados de desempenho de simulações de arquiteturas utilizando cargas de trabalho de processamento paralelo é um grande problema e um dos maiores desafios neste tipo de estudo. Desta forma, o resultado de apenas uma execução possui menor credibilidade do que a média dos valores obtidos nas três execuções, a qual é utilizada como resultado neste trabalho. No capítulo 6 e nos apêndices de A a E serão apresentados somente os resultados médios das execuções de cada *benchmark* nas máquinas utilizadas.

5.5 Obtendo os Resultados

Os resultados obtidos neste trabalho referem-se basicamente ao desempenho da atividade das memórias *cache* de cada arquitetura simulada, como número total de transações, número de *misses*, entre outros valores. Estes dados, como já foi dito anteriormente, são fornecidos pelo simulador, através das estatísticas de cada módulo de *cache* da arquitetura.

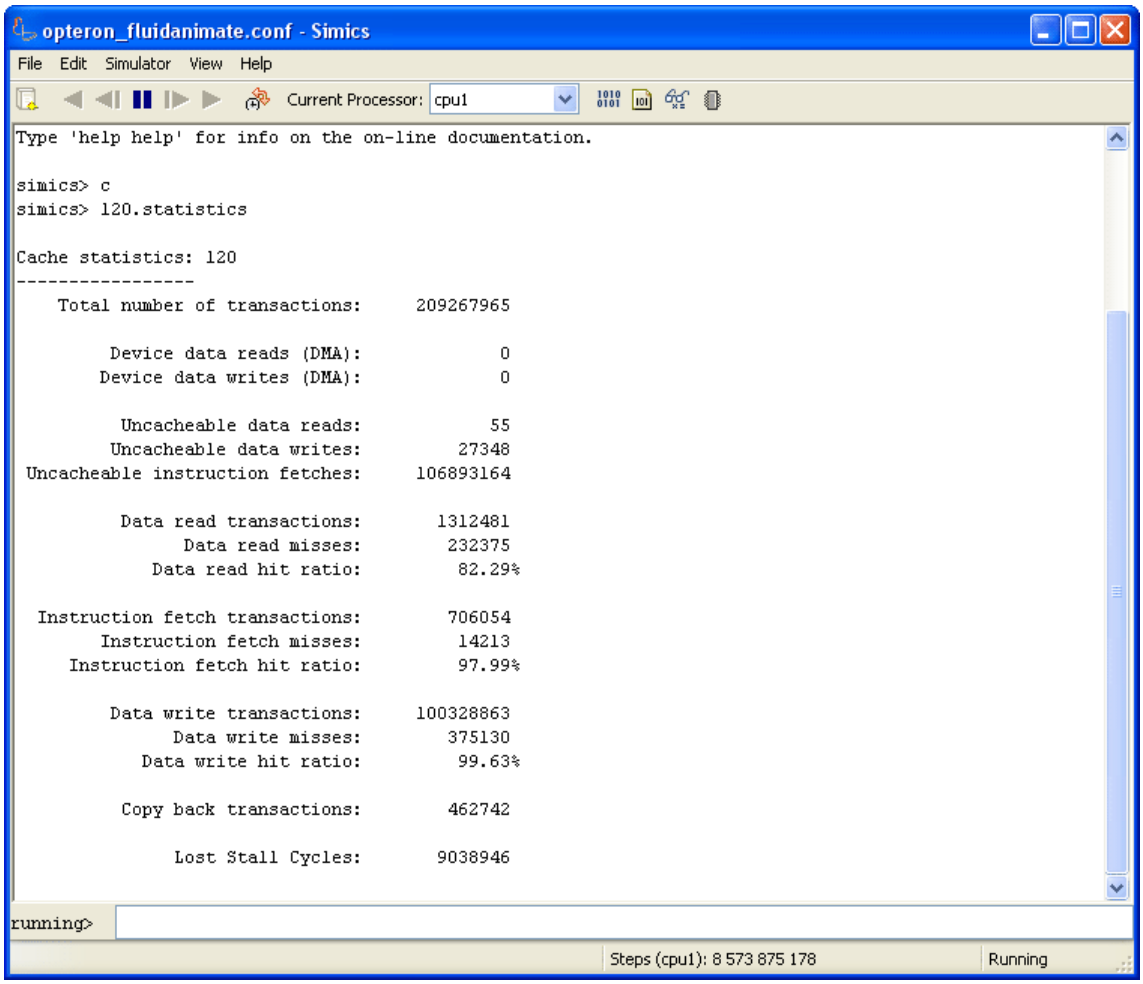

Figura 5.5: Estatísticas de memória *cache* nível 2 no Simics

Ao término de cada execução de um *benchmark*, a simulação é imediatamente interrompida, com o intuito de evitar transações da memória *cache* que não façam parte do conjunto de transações oriundas do *benchmark*. A partir da janela principal do Simics, utilizando o comando *statistics*, é possível obter os resultados de cada módulo de *cache* separadamente, sejam *cache* de instruções ou dados de nível 1, *cache* nível 2 ou até mesmo *cache* nível 3. De forma análoga, no simulador, através do comando *ptime -all*, obtêm-se o número de ciclos de execução utilizados na realização da tarefa, tanto por parte do *benchmark* quanto pelo próprio SO. A partir desse número de ciclos e das informações de freqüência dos processadores, apresentadas no capítulo 3, foi possível calcular o tempo total da execução das cargas de trabalho em cada arquitetura.

6 RESULTADOS E ANÁLISES

Neste capítulo serão apresentados os resultados obtidos na simulação de diferentes arquiteturas paralelas e suas devidas modelagens de memória *cache*. Através da execução de um conjunto relevante de *benchmarks* em cada arquitetura, os dados das transações foram obtidos através do Simics, que durante a simulação monitora a atividade dos módulos de *cache* e armazena estes dados para serem utilizados pelo usuário. Os resultados de tempo foram calculados a partir do número de ciclos de execução de cada *benchmark* nas arquiteturas trabalhadas, valor que também é fornecido pelo simulador. Após isso, será feita uma análise sobre os resultados, e uma comparação do desempenho através dos números apresentados por modelos equivalentes de cada fabricante. Ao mesmo tempo, pretende-se verificar os pontos positivos e negativos de cada abordagem adotada.

Os gráficos apresentados neste capítulo têm o objetivo de demonstrar o desempenho de cada arquitetura sob diversos aspectos, como número de transações das *caches*, número de *misses* de acessos às mesmas, ciclos de latência e tempo de execução. Nos casos de arquiteturas com mais de um módulo do mesmo tipo, os valores apresentados são as somas entre todos estes módulos dividido pelo número de núcleos da arquitetura, uma espécie de "média por *core*". Foi utilizado este método para que os resultados apresentados fossem os mais corretos possíveis, pois comparar valores de *caches* privadas com os de *caches* compartilhadas não oferece um resultado justo. Além disso, desta forma, é possível uma melhor análise do custo de cada núcleo do processador nas *caches* as quais este estiver ligado. No apêndice deste trabalho encontram-se as tabelas com os resultados exatos de cada arquitetura e seus módulos de *cache*.

6.1 Memória *Cache* de Nível 1

As memórias *cache* de nível 1 apresentam bancos distintos para instruções e para dados. Desta forma, cada seção deste subcapítulo será referente a cada um destes diferentes tipos de *cache* L1.

6.1.1 Busca de Instruções

Nas figuras 6.1 e 6.2 é possível verificar a quantidade de transações efetuadas pelos módulos de *cache* de nível 1 de instruções dos modelos de dois e quatro núcleos, respectivamente. Nota-se que as *caches* dos modelos *dual-core* apresentam maior número de transações, pelo menor número de núcleos para a divisão das tarefas.

Entre os modelos de dois núcleos, há um maior equilíbrio no número de transações efetuadas, com exceção do aplicativo Streamcluster, onde o modelo T9500 da Intel teve um número muito inferior de busca de instruções comparado aos outros

modelos, devido este programa realizar repetidas vezes um conjunto pequeno de instruções. Já nos processadores de quatro núcleos, verifica-se uma tendência do modelo Opteron 8360SE da AMD a efetuar menor número de transações deste tipo na comparação com os outros modelos equivalentes.

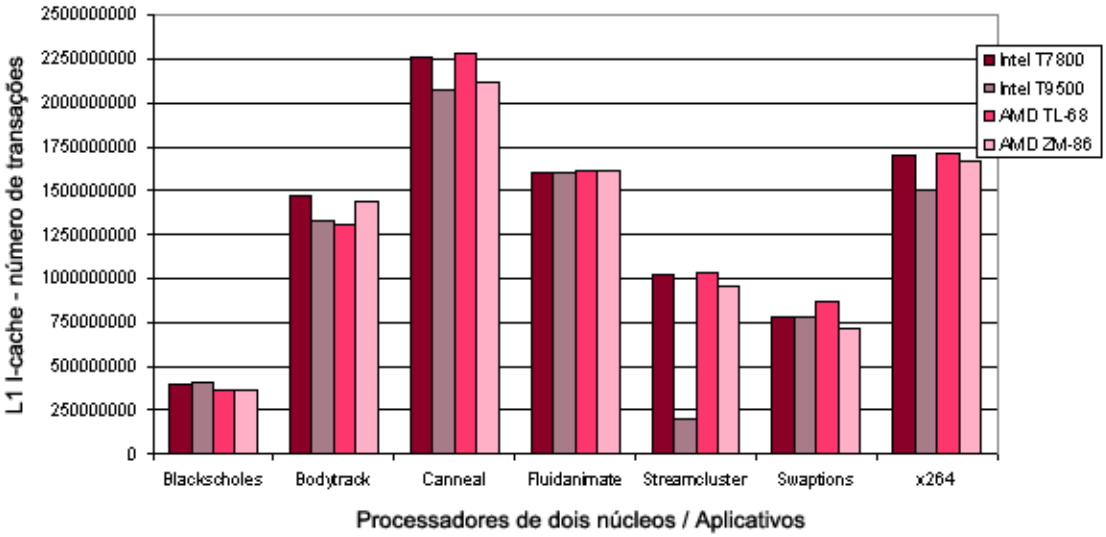

Figura 6.1: Número médio de transações das *caches* L1 de instruções para os processadores de dois núcleos

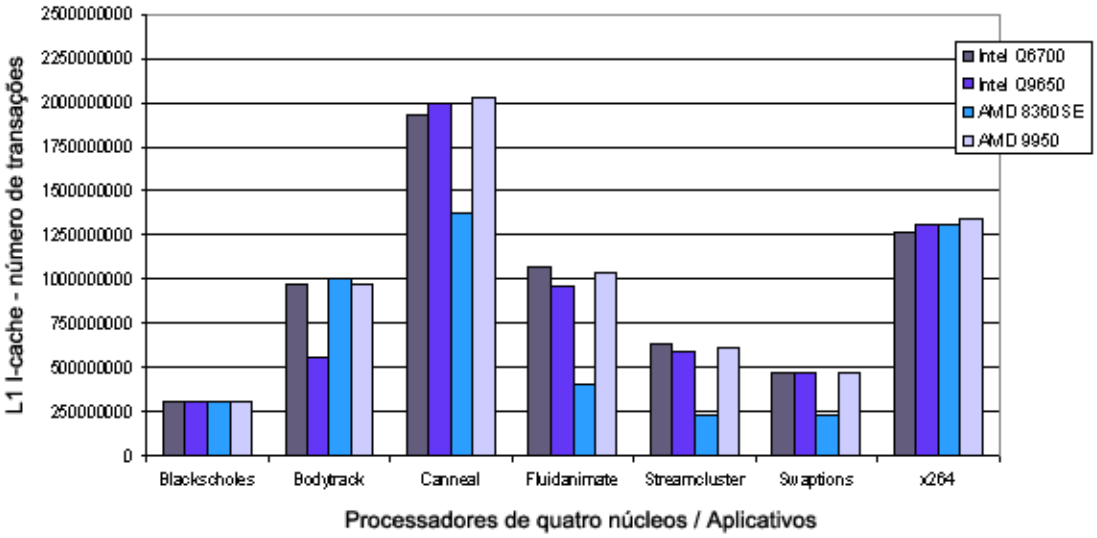

Figura 6.2: Número médio de transações das *caches* L1 de instruções para os processadores de quatro núcleos

Com estes números de transações de cada módulo de instruções das *caches*, e sabendo as latências destes módulos de cada arquitetura, vistos no capítulo 3, calcula-se a latência agregada às buscas de instruções na *cache* L1. Importante frisar que estes modelos utilizam mecanismo de *prefetch* ou pré-busca, que faz com que, quando ocorra uma transação deste tipo, uma linha inteira seja trazida, ao invés de apenas uma instrução. Desta forma, com tamanho de linha de 64 *bytes*, a cada transação são

buscadas dezesseis instruções, dividindo a latência total do sistema. Assim, temos os resultados apresentados nos gráficos 6.3 e 6.4.

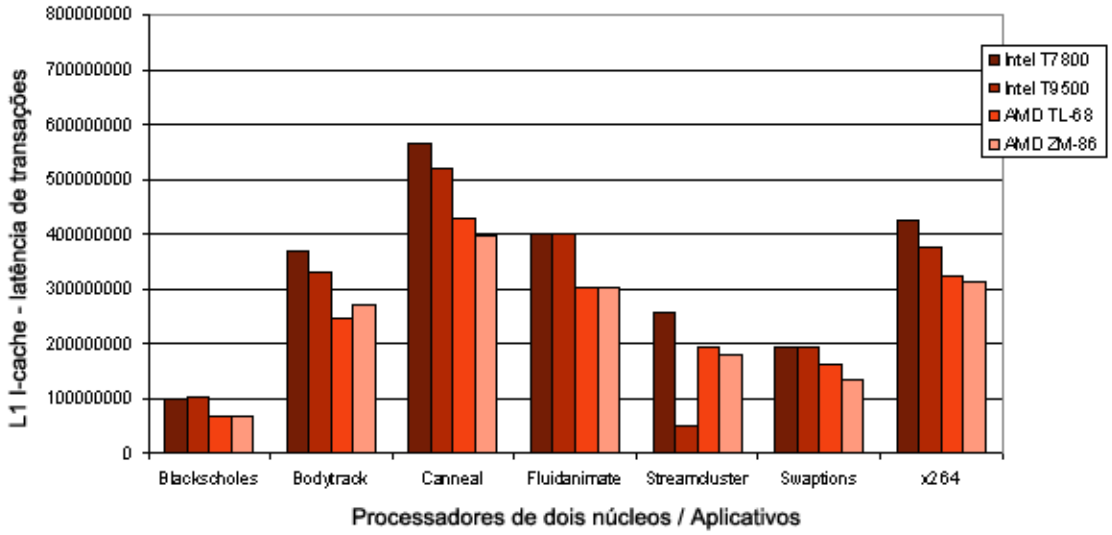

Figura 6.3: Número médio de ciclos de latência das *caches* L1 de instruções para os processadores de dois núcleos

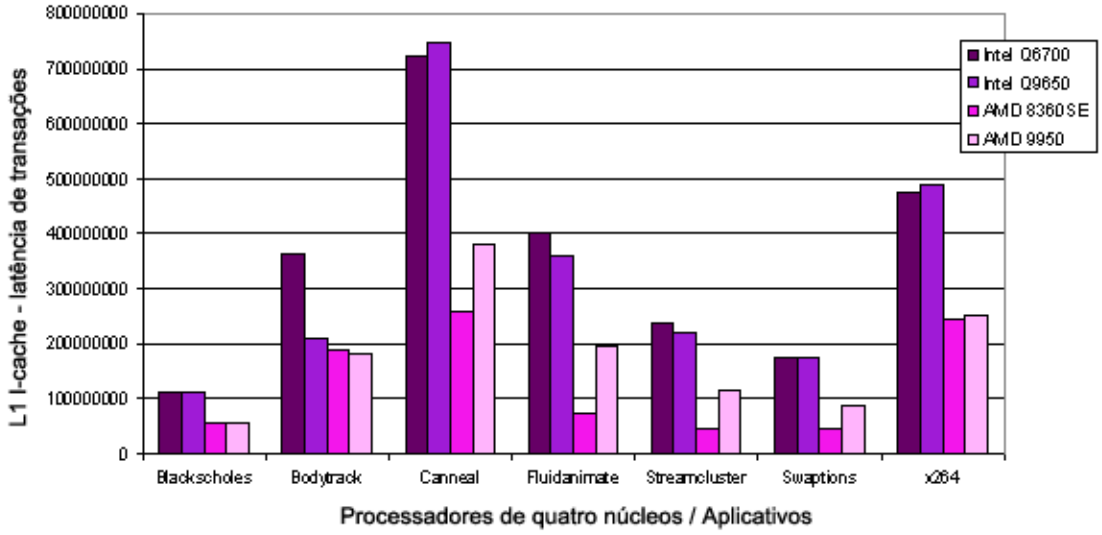

Figura 6.4: Número médio de ciclos de latência das *caches* L1 de instruções para os processadores de quatro núcleos

Verifica-se que os modelos da Intel possuem quantidade de ciclos de latência total significativamente maiores que os da AMD, por terem maior latência agregada aos módulos de *cache* nível 1, devido à maior associatividade destas. Mesmo, em muitos casos, tendo menor número de transações, estas maiores latências acabaram oferecendo resultados piores. Como veremos adiante, a grande maioria das buscas de instruções são atendidas pelas *caches* de nível 1. Deste modo, fica claro que o pior desempenho da Intel neste quesito diminui consideravelmente o desempenho de seus módulos de *cache*, dada a representatividade numérica das transações de busca de instruções

Quanto aos erros na busca das instruções, a Figura 6.5 mostra como os modelos da AMD apresentam menores números de *misses* de instruções, e esse comportamento

se apresenta claramente conforme maior é o número de transações de instruções dos aplicativos.

Nos modelos de dois *cores*, algumas diferenças são bastante claras, chegando a ter-se mais que o dobro de erros em determinado aplicativo (x264). Os programas de menor quantidade de transações tiveram resultados bastante equilibrados. A exceção nesta análise se deu com relação ao *benchmark* Fluidanimate, que possui grande número de transações de instruções, mas quantidades muito pequenas de erros nas buscas.

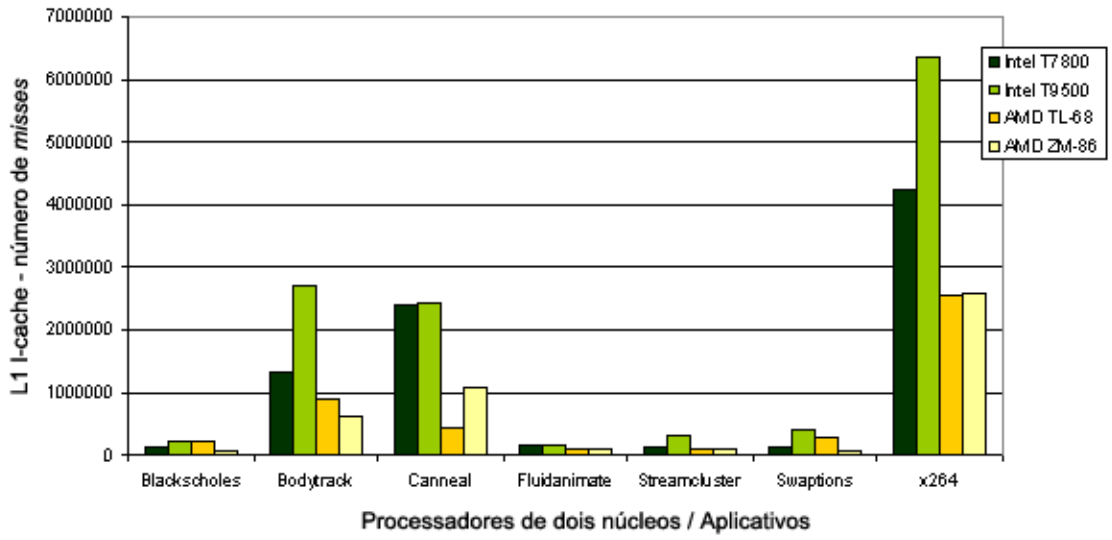

Figura 6.5: Número médio de *misses* das *caches* L1 de instruções para os processadores de dois núcleos

Na mesma análise, nos modelos de quatro núcleos, a Figura 6.6 ilutra um maior equilíbrio entre as quantidades de erros de cada arquitetura. Nota-se resultados muito bons principalmente no modelo Q9650 da Intel, com valores intermediários ou inferiores à média de erros entre os modelos *quad-core*. Também podemos verificar que os modelos Q6700 da Intel e Opteron 8360SE da AMD não tiveram bons resultados, em geral se destacando (no caso, negativamente) com relação aos outros.

Analisando o número de transações e a quantidade de erros, os modelos da AMD tiveram melhores resultados na comparação entre as *caches* de intruções de nível 1, em especial o modelo Phenom 9950. Isto prova que estas *caches* da AMD, que possuem maior tamanho, ofereceram desempenho melhor, principalmente na questão de número de *misses*, obtendo valores significativamente menores, como nos modelos de dois núcleos. Importante notar que, devido ao imenso número de transações efetuadas por estes módulos de *cache*, mesmo com resultados inferiores, os modelos da Intel ainda assim conseguem taxas de acerto muito altas, como é possível verificar no apêndice do trabalho.

Outro fator que determina o quanto os *misses* afetarão o desempenho do sistema como um todo é o número de ciclos de latência de cada módulo de *cache*. No caso de um erro em uma transação de leitura ou escrita em determinado nível de *cache*, o processador procura ou armazena este dado ou instrução no próximo nível, e o faz sucessivamente, até conseguir realizar a transação sobre a informação, no último caso em memória principal. Desta forma, um *miss* na *cache* de nível 1 resulta em uma busca na *cache* de nível 2, agregando a este *miss* o valor da latência da *cache* superior.

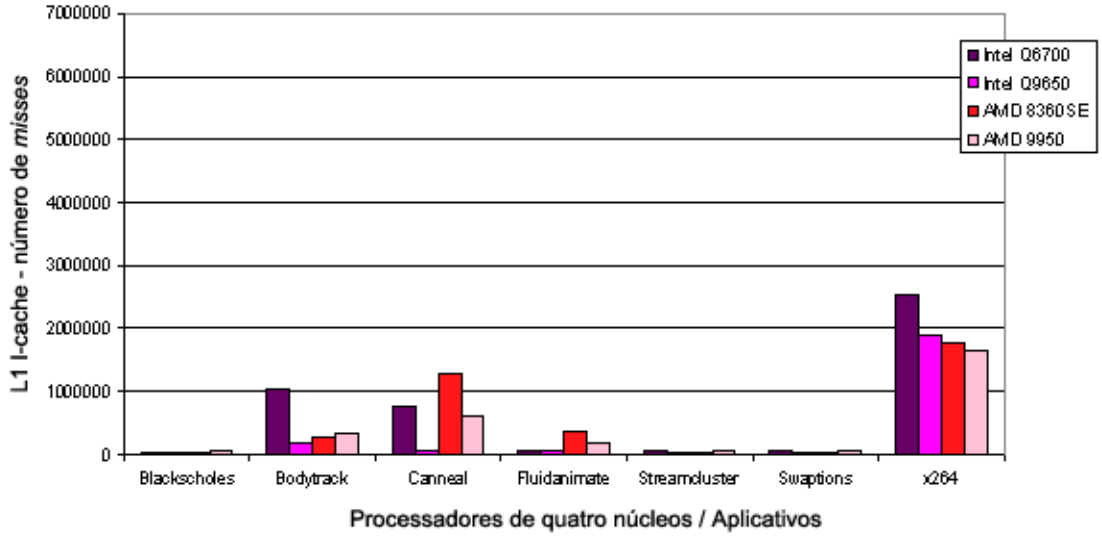

Figura 6.6: Número médio de *misses* das *caches* L1 de instruções para os processadores de quatro núcleos

Assim, para analisarmos o peso agregado aos *misses* da *cache* L1 de instruções, precisamos multiplicar o número de erros pela latência da *cache* L2. Desta forma, obtemos os gráficos representados pelas Figuras 6.7 e 6.8.

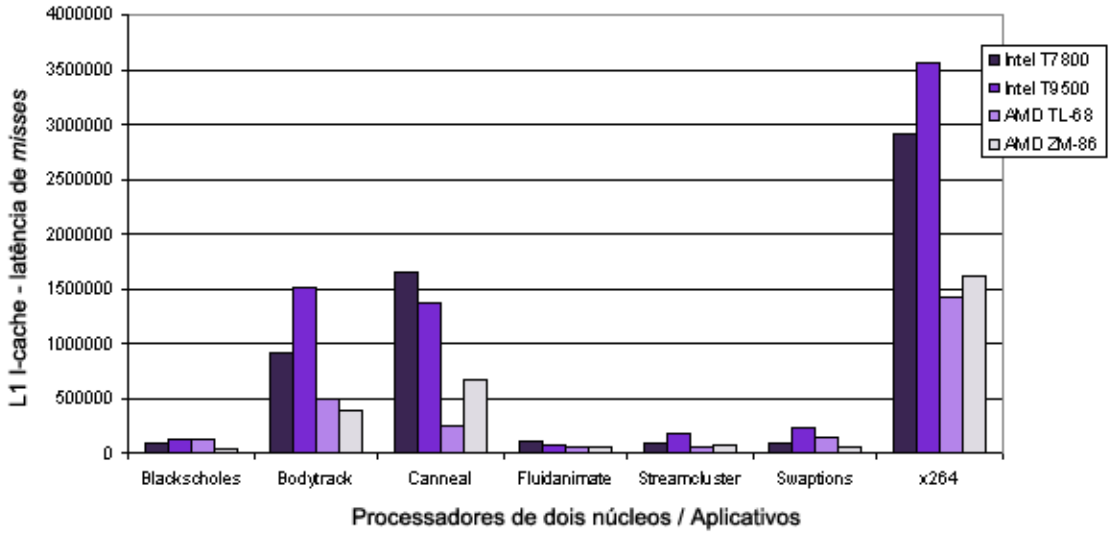

Figura 6.7: Número médio de ciclos de latência de *misses* das *caches* L1 de instruções para os processadores de dois núcleos

A Figura 6.7 mostra que a diferença entre as arquiteturas da Intel e da AMD aumentou, com relação ao gráfico de número de *misses*. Isso deve-se ao fato das *caches* de nível 2 da Intel possuírem, em geral, maior latência que as da AMD, devido à sua maior capacidade de armazenamento. Esta figura deixa clara a relação entre a quantidade de *misses* ocorridos e a queda de desempenho, pois os ciclos de latência representam o tempo total que o sistema utilizou na busca de informações na *cache* L2 por não ter encontrado as mesmas na *cache* L1. Durante estes ciclos, o processador fica ocioso, aguardando a informação ser levada até ele ou ser armazenada na *cache* L2.

Diante disto, os modelos Intel tiveram piores desempenhos médios, o que deve aumentar o tempo de execução destes processadores.

Quanto aos modelos de quatro núcleos, a Figura 6.8 mostra um maior equilíbrio nestas latências entre famílias Intel e AMD, devido à maior paridade de número de *misses* entre as mesmas. Podemos dizer que, de modo geral, os modelos Q6700 da Intel e Opteron 8360SE da AMD não tiveram desempenhos muito satisfatórios, alcançando valores bem acima dos outros processadores em determinados aplicativos.

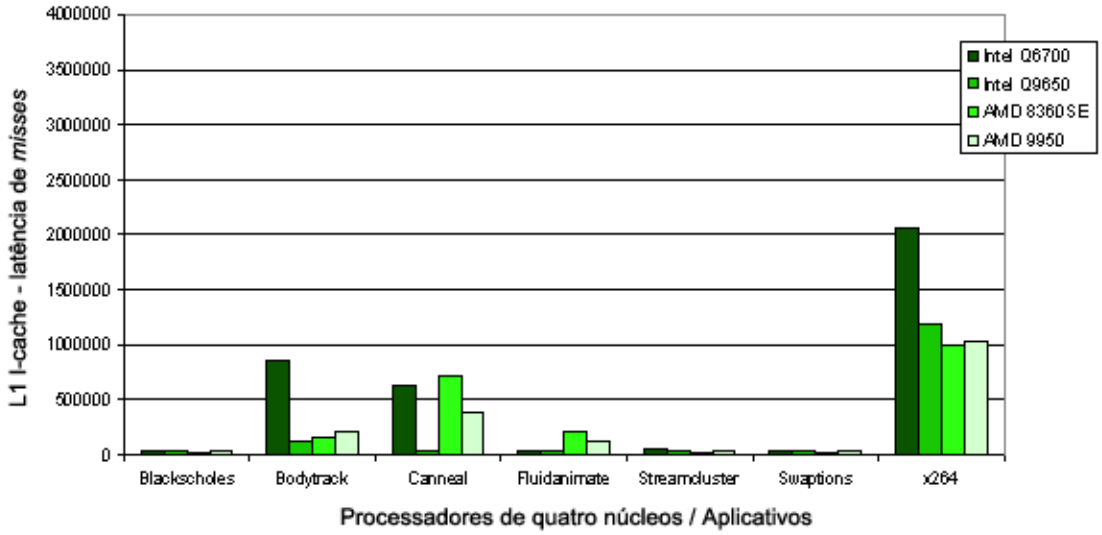

Figura 6.8: Número médio de ciclos de latência de *misses* das *caches* L1 de instruções para os processadores de quatro núcleos

6.1.2 Leitura e Escrita de Dados

Os gráficos a seguir apresentam colunas divididas em duas cores para representar a quantidade total de transações de dados, entre leituras e escritas. Conforme a legenda indica, a parte inferior das colunas representa as transações de leitura, e portanto a parte superior indica a quantidade de escritas destes módulos.

Na figura 6.9, na comparação entre as *caches* de dados de nível 1 dos modelos de dois núcleos, fica claro que a quantidade de leituras de dados é maior que a escrita dos mesmos. Ainda assim, o número de operações de escrita é razoavelmente grande na comparação do número total de transações.

Já nos modelos de quatro núcleos, a desproporcionalidade entre leituras e escritas é menor, como mostra a Figura 6.10. Por efetuarem menor número de transações, o que é visto na comparação com a Figura 6.9, a tendência das *caches* de dados é uma diminuição mais acentuada nas leituras, enquanto nas escritas este decréscimo é menor.

Assim como é possível verificar nos *dual-cores*, os *quad-cores* apresentam um equilíbrio entre os modelos com relação ao número total de transações, sendo difícil distinguir algum modelo que tenha se sobressaído comparado aos outros, tanto positivamente quanto negativamente. Este equilíbrio de transações se estende nas comparações de transações de leitura e escrita.

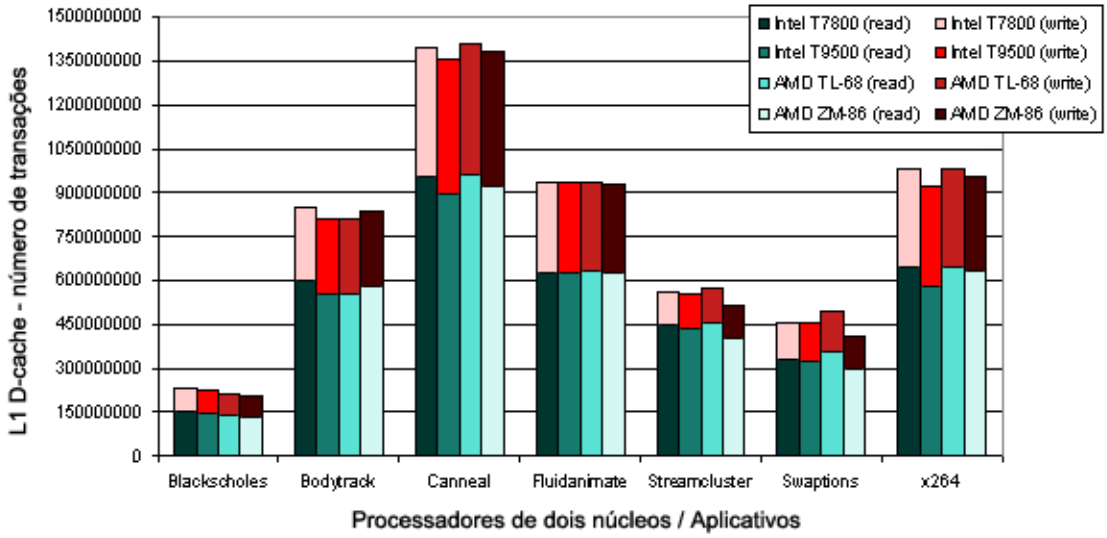

Figura 6.9: Número médio de transações das *caches* L1 de dados para os processadores de dois núcleos

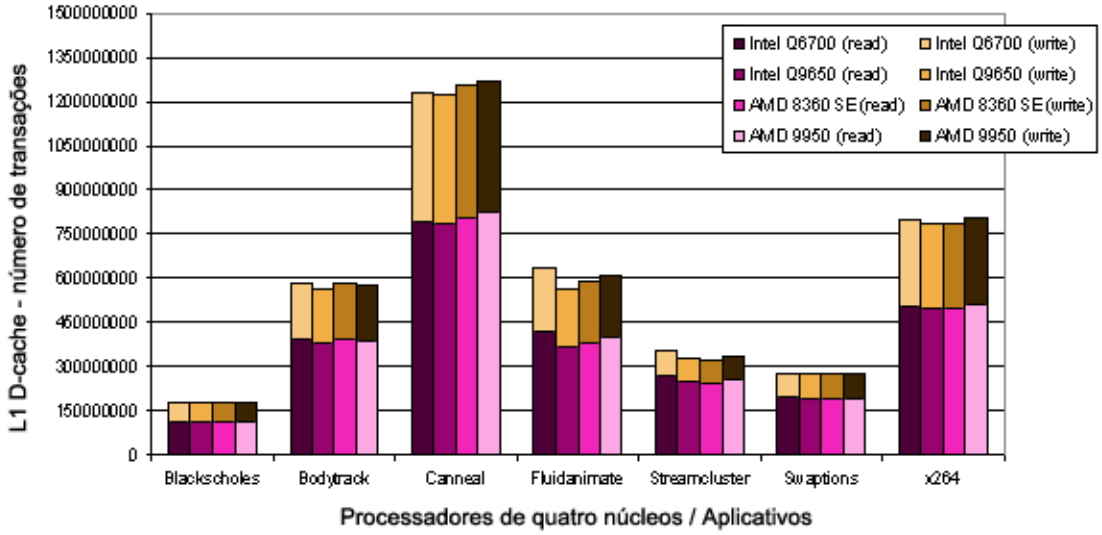

Figura 6.10: Número médio de transações das *caches* L1 de dados para os processadores de quatro núcleos

A partir da avaliação da quantidade de transações efetuadas, o número de ciclos gastos na realização das mesmas irá mostrar qual arquitetura teve melhor desempenho neste aspecto do que as outras. A Figura 6.11 apresenta os resultados das latências das transações da *cache* de dados dos modelos de dois núcleos.

Assim como foi visto nas *caches* de instruções, novamente os modelos da Intel apresentaram maior número de ciclos total, por possuírem módulos com maior latência. É possível notar uma paridade entre os modelos de mesma família, tanto da Intel quanto da AMD. Entre os processadores da Intel, o T9500 obteve número de ciclos levemente menor que o T7800. Já entre os representantes da AMD, o ZM-86 teve melhor desempenho neste quesito.

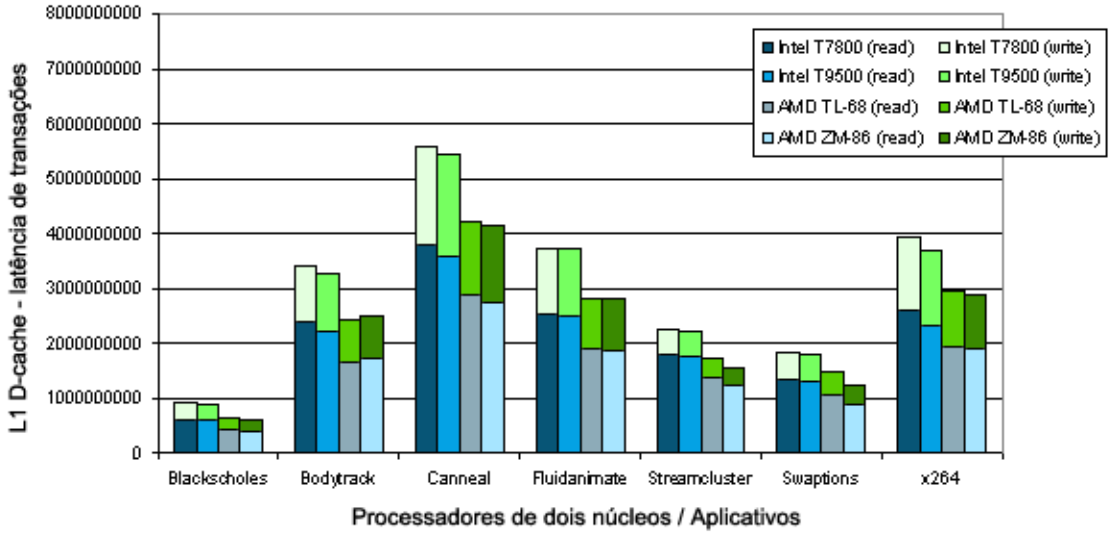

Figura 6.11: Número médio de ciclos de latência das *caches* L1 de dados para os processadores de dois núcleos

Analisando os modelos *quad-cores*, as diferenças de latências de *misses* entre Intel e AMD são ainda maiores. Em todos os casos, a latência de leitura de dados das *caches* da Intel é maior que a latência total das *caches* dos modelos AMD. Novamente nota-se como o desepenho da Intel é prejudicado devido às suas latências. Neste quesito, mesmo a AMD implementando *caches* de menor associatividade, o que em geral aumenta o número de *misses* e de transações de escrita de dados, o menor número de ciclos de latência desta abordagem favorece muito no desempenho.

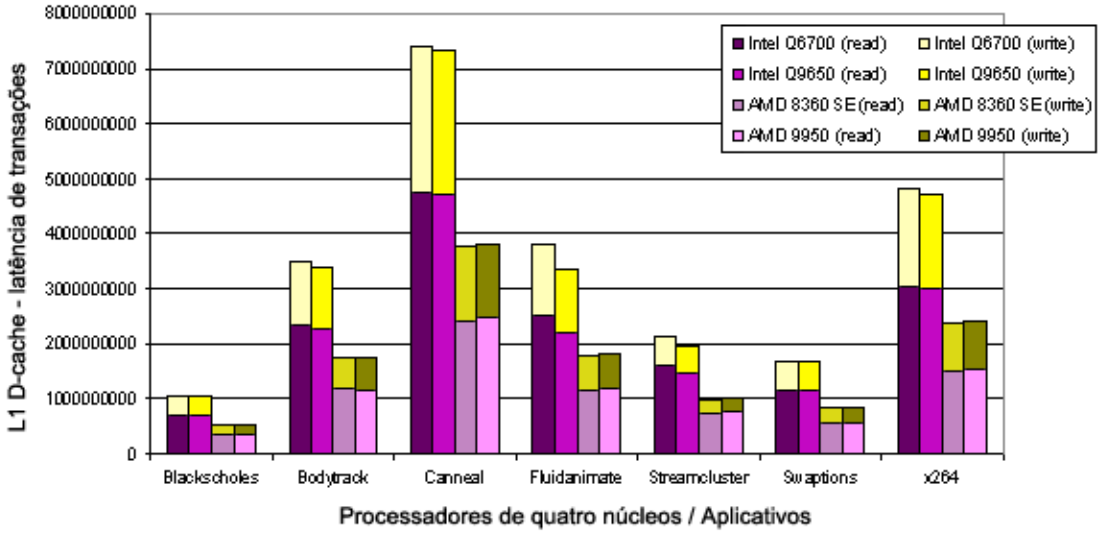

Figura 6.12: Número médio de ciclos de latência das *caches* L1 de dados para os processadores de quatro núcleos

Na avaliação dos *misses* de dados, as figuras 6.13 e 6.14 mostram os resultados das *caches* L1 de dados. Verificando os gráficos, fica claro que, na maioria das aplicações, a escrita dos dados é responsável pela maior parte dos erros. Nos *benchmarks* Blackscholes, Bodytrack e x264 essa diferença é muito grande, sendo as escritas responsáveis por em torno de 75% dos *misses*. De forma antagônica, no

aplicativo Streamcluster, cerca de dois terços do número total de erros deve-se à leitura. Isso mostra como a execução de programas distintos utiliza de formas diferentes as *caches* e obtém resultados distintos.

Nos modelos de dois *cores* verifica-se o desempenho negativo do modelo T9500 da Intel, que conseguiu o pior resultado em todos os testes. Em geral, os modelos da AMD foram os que tiveram as melhores médias de resultados.

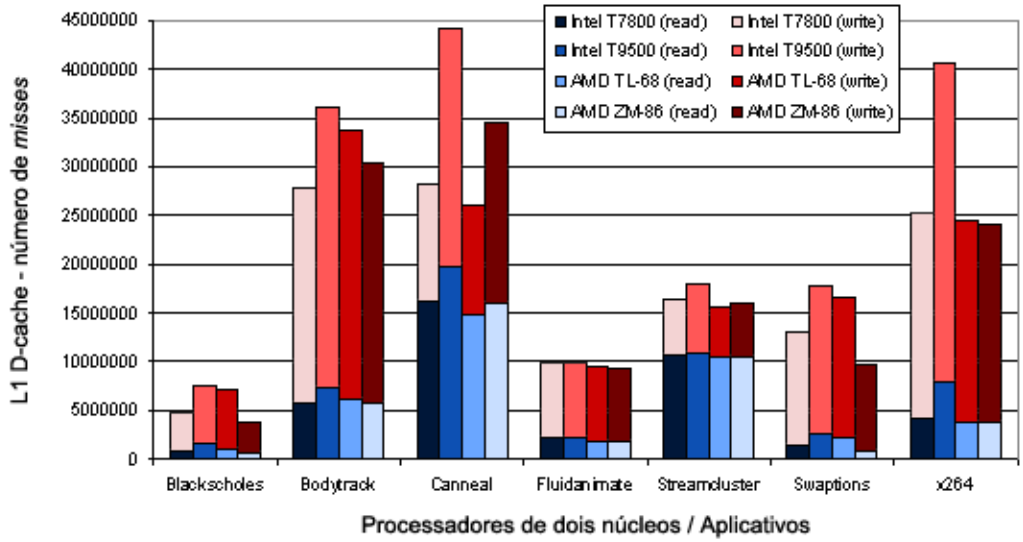

Figura 6.13: Número médio de *misses* das *caches* L1 de dados para os processadores de dois núcleos

Dentre os modelos de quatro núcleos, o que obteve a melhor média de resultados foi o Intel Q9650, que em aplicações como Canneal teve um número de erros bastante inferior aos outros. Em contrapartida, o outro modelo da Intel, o Q6700 não se saiu tão bem, devido a números muito elevados de erros de escrita. Os modelos da AMD tiveram resultados bastante satisfatórios, sendo privilegiados pelo maior tamanho dos módulos de *cache* de nível 1.

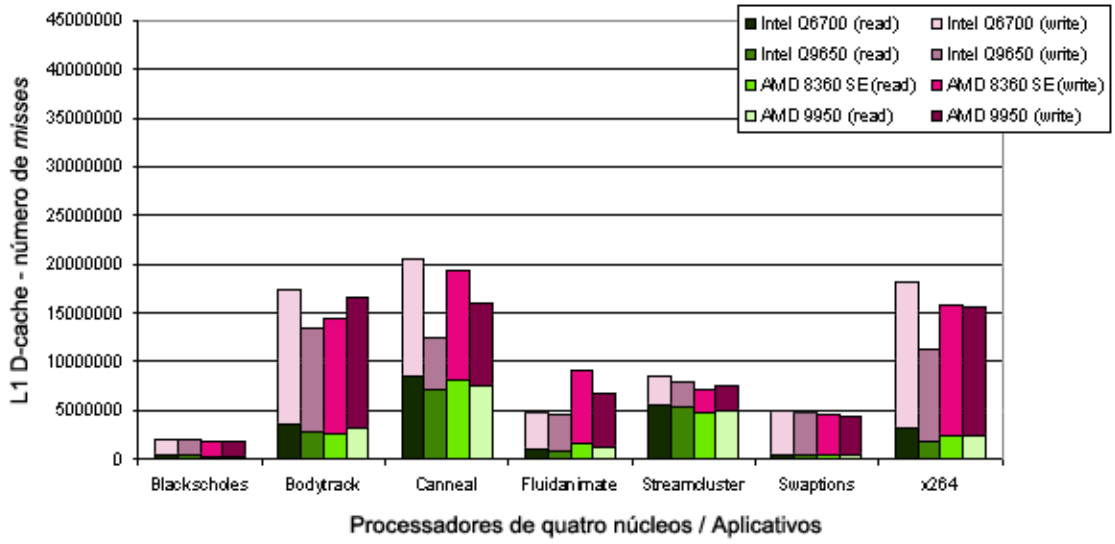

Figura 6.14: Número médio de *misses* das *caches* L1 de dados para os processadores de quatro núcleos

Analisando os números das transações de dados e a quantidade de *misses* ocorridos, verifica-se que dentre as arquiteturas de dois núcleos, os modelos da AMD levam certa vantagem, por efetuarem, em geral, menor número de transações e possuírem menos erros. Já nas famílias de quatro processadores, o modelo Q9650 da Intel acaba tendo um ligeiro ganho sobre os outros, principalmente pelo fato de realizar menor número de transações.

De forma análoga ao que foi feito na análise dos *misses* de busca de instruções, deve-se calcular os ciclos de latência de cada arquitetura com relação aos erros de leituras e escritas na *cache* nível 1 de dados. Neste caso, assim como os outros gráficos sobre transações de dados, os resultados de leituras e escritas apareceram empilhados, para que se possa ter uma idéia da quantidade total de ciclos de latência dos *misses* de dados.

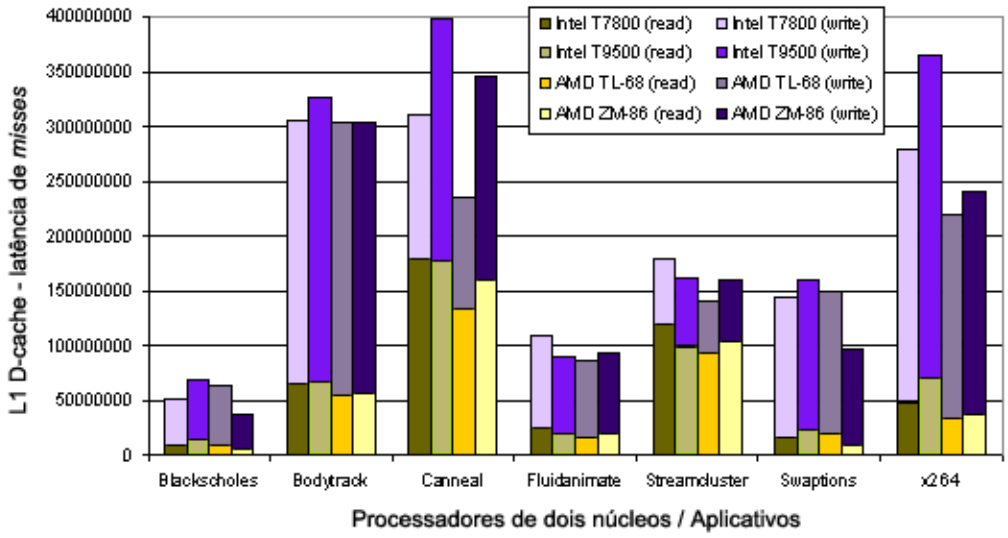

Figura 6.15: Número médio de ciclos de latência de *misses* das *caches* L1 de dados para os processadores de dois núcleos

A Figura 6.15 apresenta resultados parecidos com os de números de erros das transações sobre dados, pois as *caches* L2 dos modelos de dois núcleos possuem latências próximas, como é possível ver no capítulo 3, o que propicia um aumento mais proporcional. Mas é possível verificar um pior resultado dos modelos T7800 da Intel e ZM-86 da AMD, que possuem latências levemente maiores e acabaram se aproximando dos valores obtidos pelos outros modelos, embora tenham executado número de *misses* significativamente menores.

Já na Figura 6.16 verifica-se claramente uma tendência do Intel Q6700 de ter os piores resultados, por possuir latência significativamente maior que os outros modelos, inclusive o Q9650. Apesar de ter número de *misses* semelhante aos outros *quad-cores*, o número de ciclos foi seriamente penalizado. Em contrapartida, o modelo Opteron 8360SE da AMD, que obteve número de erros de transações relativamente altos, conseguiu ter melhor desempenho na relação dos ciclos de latência, devido à latência de reduzido valor.

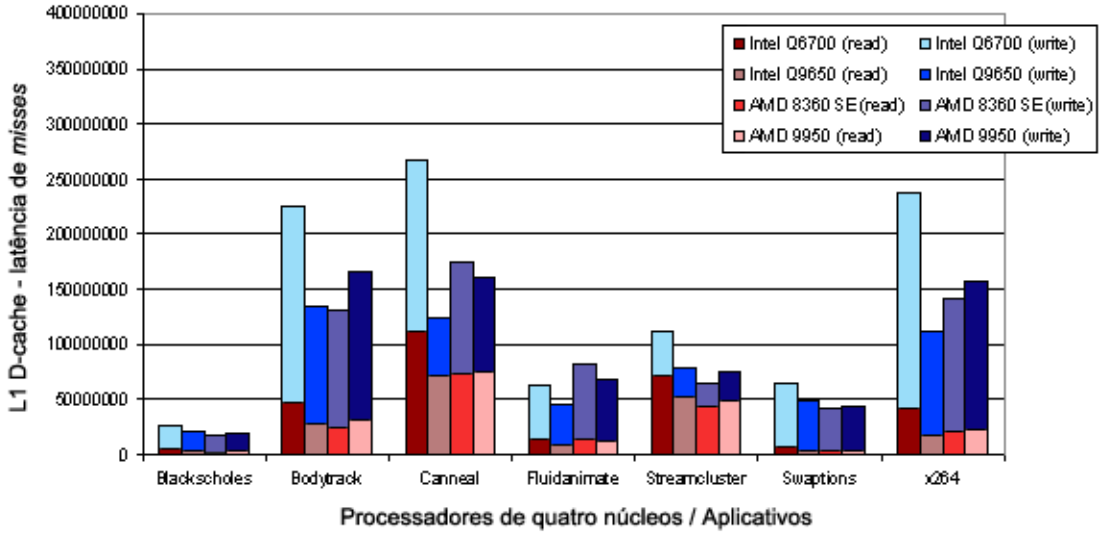

Figura 6.16: Número médio de ciclos de latência de *misses* das *caches* L1 de dados para os processadores de quatro núcleos

Foi possível analisar como as *caches* de nível 1 agilizam as consultas de instruções e dados, representando a imensa maioria das transações realizadas pelos processadores. Um dos pontos cruciais no desempenho das *caches* é a latência a elas associada, que determina a quantidade de ciclos que o sistema esperará pelo dado ou instrução a ser buscada. A utilização de *caches* com latências elevadas tende a reduzir consideravelmente o desempenho do sistema. Além disso, o número de *misses* é outro agravante nas *caches* de nível 1. Em geral, por terem tamanho reduzido, acabam ocasionando grande número de erros de consulta, o que faz com que seja necessário acessar *caches* de nível superior ou até mesmo a memória principal. Neste ponto, deve-se encontrar um consenso entre tamanho de *cache*, associatividade e latência, para que o modelo escolhido possa ter o melhor desempenho possível, realizando um número razoável de *misses* e possuindo latência de poucos ciclos. Analisando as abordagens da Intel e da AMD, a utilização de uma menor latência através do uso de uma baixa associatividade, implementada pela AMD, mostrou ser mais eficiente.

Como será visto no próximo subcapítulo, a utilização de *caches* de nível 2 de grande capacidade é muito interessante, com o intuito de diminuir o número de *misses* e aumentar as taxas de acerto, de leitura e principalmente de escritas. Mas uma das desvantagens desta abordagem, como já foi observado, é o aumento da latência destas *caches*, o que prejudica o desempenho no caso de transações que efetuam *miss* na *cache* L1 e precisam acessar a *cache* L2.

6.2 Memória *Cache* de Nível 2

6.2.1 Busca de Instruções

Verificando o gráfico representado pela Figura 6.17, é visível a diferença do número de buscas de instruções realizadas pelas arquiteturas de dois núcleos da Intel e da AMD. Nas aplicações de maior execução, e que por isso realizam maior número de transações de instruções, a diferença entre as famílias dos dois fabricantes chega a mais de cem por cento. Em aplicações menores, os valores são mais próximos, mas ainda assim os modelos da Intel se sobressaem negativamente. Pelo fato das arquiteturas da

AMD utilizarem memórias *cache* L2 privadas por núcleo, era esperado que elas tivessem menor número de transações, mas com uma menor diferença. Lembrando que os valores apresentados nos gráficos são as médias por número de núcleos de cada sistema. Se forem comparadas as médias por número de *caches*, a diferença entre Intel e AMD torna-se ainda maior.

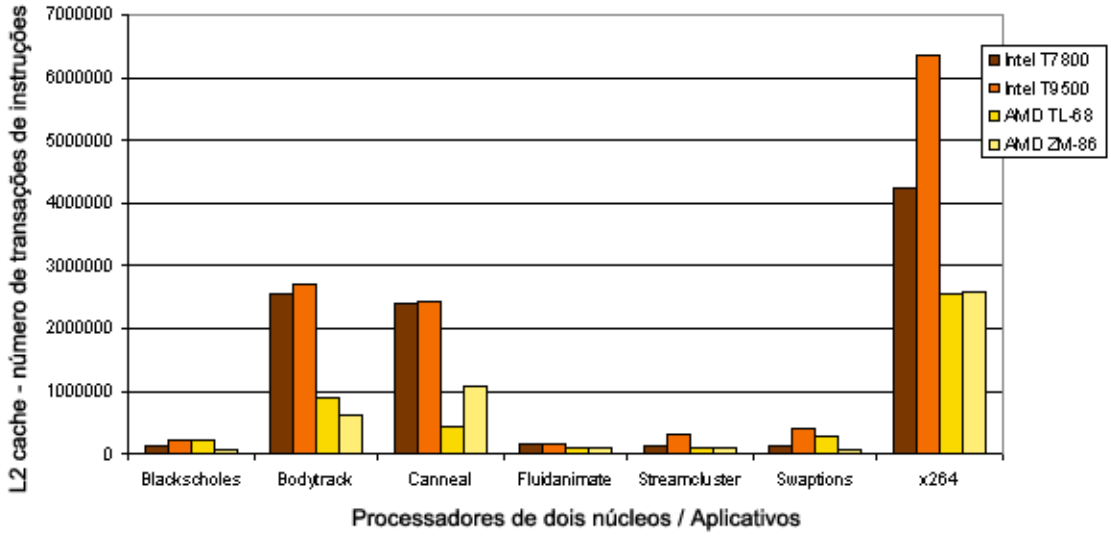

Figura 6.17: Número médio de transações de instruções das *caches* L2 para os processadores de dois núcleos

Na comparação entre os modelos de mesma família, o Core 2 Duo T9500 teve maior número de transações comparado ao T7800. Os modelos Turion tiveram resultados muito parecidos, mas na maioria das aplicações houve maiores valores no modelo TL-68, apesar de a maior diferença entre eles encontrar-se no *benchmark* Canneal, com número maior de transações efetuadas pelo ZM-86.

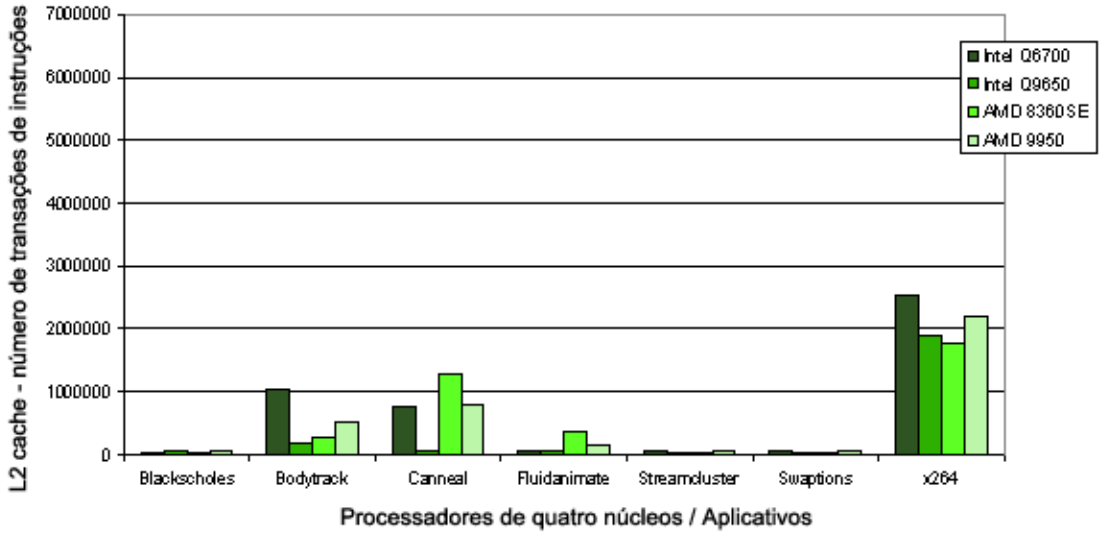

Figura 6.18: Número médio de transações de instruções das *caches* L2 para os processadores de quatro núcleos

Dentre as arquiteturas de quatro núcleos, além de menores números de transações, observa-se maior paridade entre os mesmos. Em certos aplicativos, alguns

modelos obtiveram resultados significativamente maiores que os outros, como é o caso do Q6700 da Intel nos *benchmarks* Bodytrack e x264. Analisando com maior afinco as aplicações de maior execução, nota-se que novamente os modelos da AMD tiveram menores números de buscas de instruções, agora numa proporção mais adequada ao esperado anteriormente. Interessante notar que no *benchmark* Fluidanimate os modelos Intel de quatro núcleos tiveram expressiva redução do número de transações, enquanto os equivalentes da AMD tiveram aumento significativo, comparados aos modelos de dois núcleos. Neste caso, a maior individualização das *caches* não teve o efeito desejado.

Se por um lado a grande capacidade de armazenamento das *caches* nível 2 e suas altas associatividades aumentam as taxas de acerto de escritas e leituras, por outro lado tendem a ter latências altas. Esta análise dos ciclos de latência efetuados pode ser feita a partir das figuras 6.19 e 6.20.

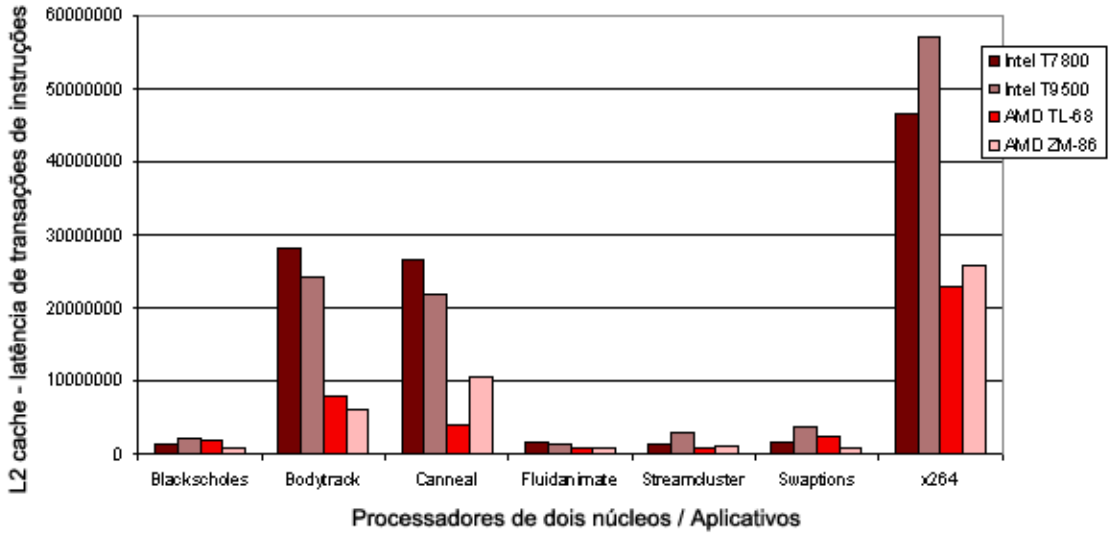

Figura 6.19: Número médio de ciclos de latência de instruções das *caches* L2 para os processadores de dois núcleos

A Figura 6.19 mostra como as arquiteturas Intel mais uma vez têm seus desempenhos comprometidos. Além de suas *caches* L2 realizarem maior número de transações, possuem maior latência que as arquiteturas AMD em ambos os modelos. Esta soma de fatores contribui para os altos números de ciclos obtidos pelos processadores Intel nas cargas de trabalho utilizadas.

Nas arquiteturas *quad-core*, assim como o número de transações, as latências tornam-se mais próximas. Mas ainda assim há um aumento no número de ciclos da Intel maior que da AMD, por conta de seus valores de latência.

Como já havia sido observado na questão de *misses* das *caches* L1, mais uma vez fica claro que um dos grandes problemas das arquiteturas Intel são os altos valores de latência de suas *caches*. Nas de nível 1, por utilizar associatividade de diversas vias, e nas de nível 2 basicamente pelo grande tamanho dos módulos. Nitidamente, a intenção da Intel é diminuir ao máximo o número de consultas sem sucesso, com *caches* L2 bastante largas que possam minimizar acessos à memória principal.

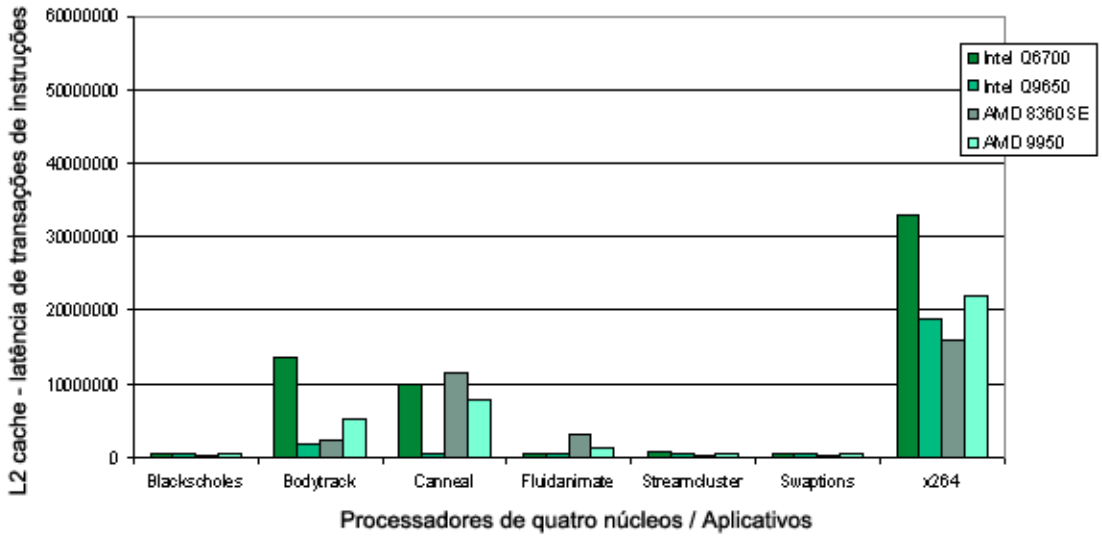

Figura 6.20: Número médio de ciclos de latência de instruções das *caches* L2 para os processadores de quatro núcleos

Na comparação de *misses* de busca de instruções das *caches* L2, tem-se resultados contrários aos obtidos até agora. As arquiteturas da AMD, tanto de dois quanto de quatro *cores* tiveram números de erros muito maiores que os modelos da Intel, como mostram as figuras 6.21 e 6.22. Aqui, o reduzido tamanho dos módulos de *cache* nível 2 das famílias AMD contribuíram para estes resultados negativos, tendo então vantagem as características empregadas pela Intel.

Os resultados dos processadores de dois núcleos mostram como os modelos da AMD são prejudicados por esta abordagem: o TL-68 teve valores muito acima dos obtidos pelos Core 2 Duo, e abaixo do ZM-86 em apenas um aplicativo. Já os modelos da Intel alternam-se entre qual possui menor número de erros em cada aplicação, o que mostra que a diferença de 4MB para 6MB – existente entre o T7800 para o T9500 – não garantiu melhor desempenho.

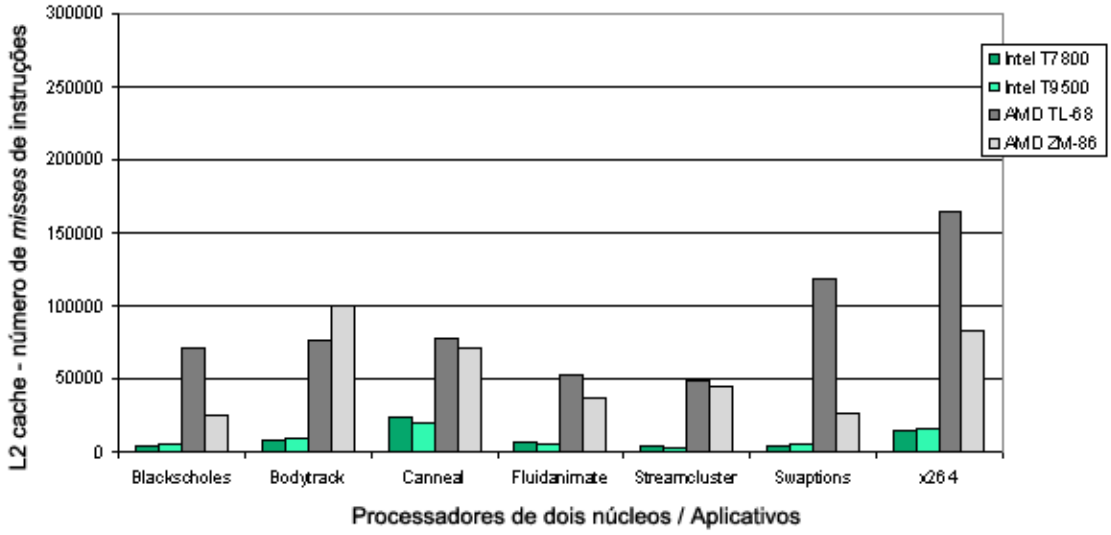

Figura 6.21: Número médio de *misses* de instruções das *caches* L2 para os processadores de dois núcleos

De forma análoga ao ocorrido com os *dual-cores*, os processadores de quatro núcleos da AMD também tiveram valores de *misses* significativamente maiores. Novamente, nenhum resultado da AMD foi melhor que os resultados da Intel, devido ao tamanho das *caches* L2, que possuem somente 512KB de capacidade de armazenamento. Importante notar como o Opteron 8360SE teve péssimos resultados, comparado aos demais, nos *benchmarks* Canneal e x264. Nos demais programas, o Phenom 9950 apresentou os maiores valores.

Dentre os modelos da Intel, o Q9650 teve os melhores desempenhos, com valores em alguns aplicativos bem abaixo até do outro modelo da Intel, o Q6700, que também obteve resultados satisfatórios. Neste caso, há sim um ganho com relação ao aumento do tamanho dos módulos de *cache*, de 4MB para 6MB, entre o Q6700 e o Q9650.

Os modelos AMD tiveram menor número de transações, mas uma quantidade significativamente maior no número de *misses*. Isso prova que, se por um lado *caches* pequenas são interessantes para diminuir a latência, por outro lado prejudicam o sistema pelo aumento de erros agregados a esta abordagem.

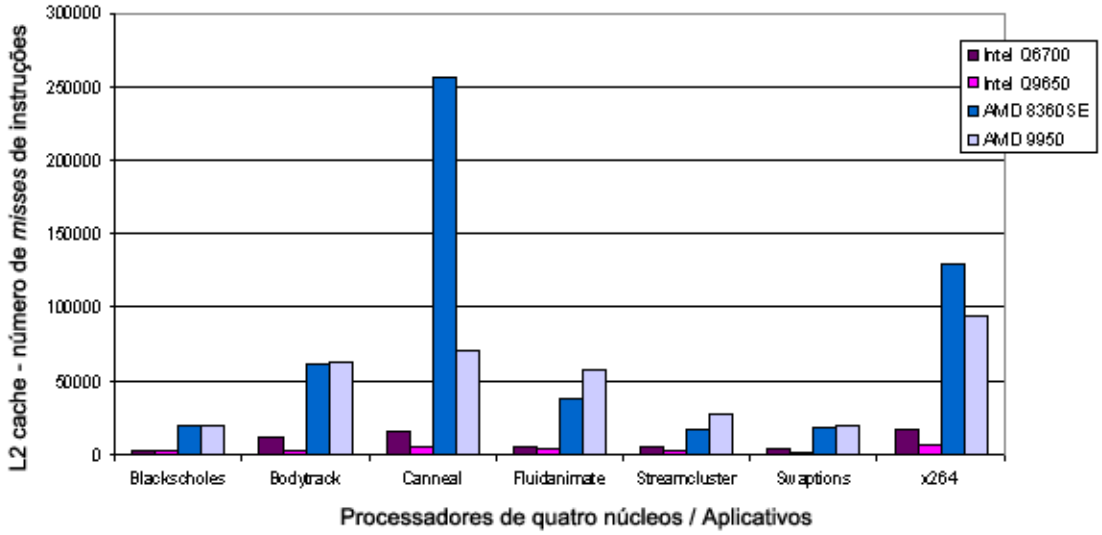

Figura 6.22: Número médio de *misses* de instruções das *caches* L2 para os processadores de quatro núcleos

No caso dos *misses*, as arquiteturas da AMD tiveram número de ciclos de latência muito maiores que as arquiteturas Intel, devido ao grande número de erros nas buscas de instruções das *caches* L2 dos modelos AMD. É possível verificar que, comparado ao gráfico de número de *misses*, a diferença entre Intel e AMD apresentada na Figura 6.23 aumentou razoavelmente, pois ambos as famílias de dois núcleos, ao gerar um *miss* na *cache* L2, realizam acesso à memória principal, de alta latência.

Na Figura 6.24 há uma nítida diminuição na diferença entre Intel e AMD com relação aos ciclos de latência de *miss*. Isto deve-se à utilização da *cache* de nível 3 por parte dos modelos de quatro núcleos da AMD, o que reduz muito a latência. No caso da ausência desta *cache* L3, os *quad-cores* da AMD teriam latências ainda maiores comparadas aos equivalentes da Intel.

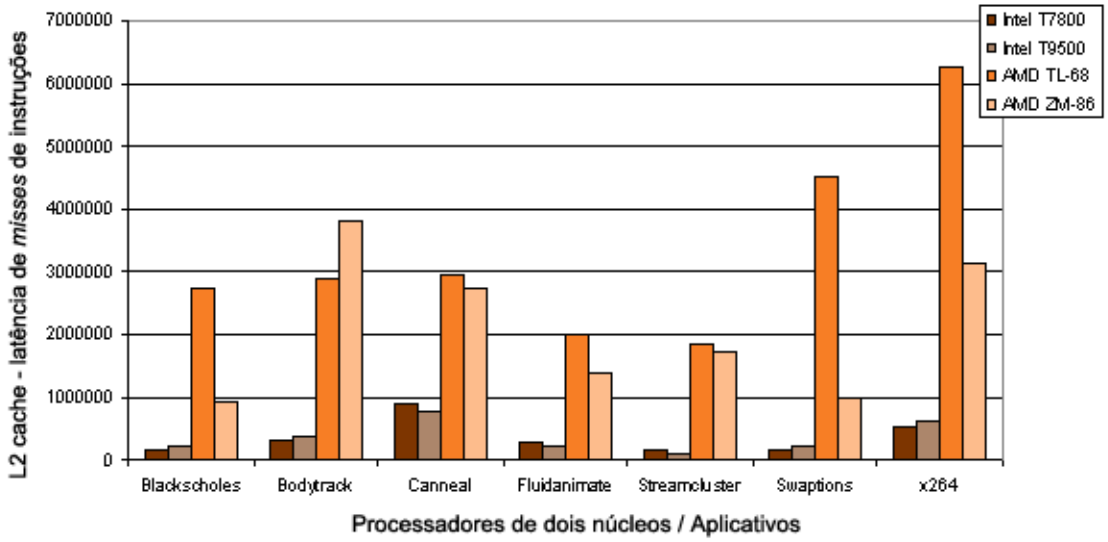

Figura 6.23: Número médio de ciclos de latência de *misses* de instruções das *caches* L2 para os processadores de dois núcleos

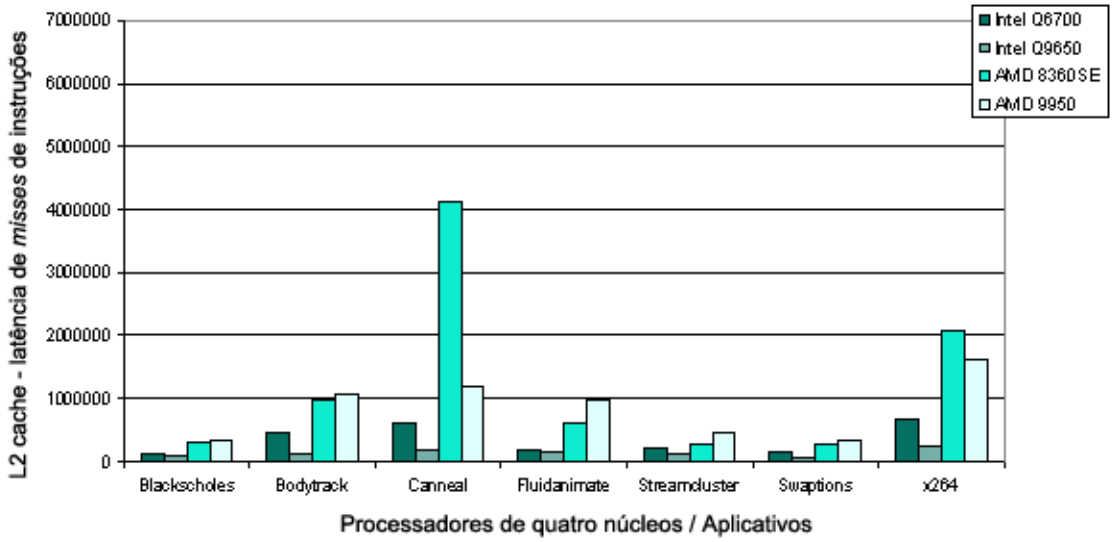

Figura 6.24: Número médio de ciclos de latência de *misses* de instruções das *caches* L2 para os processadores de quatro núcleos

Nota-se que a relação entre quantidade de transações, número de *misses* e latência não é tão trivial. A arquitetura Intel, com elevado número de transações e latência, efetua muito menos *misses* que os modelos AMD. Claramente, a abordagem de *cache* dos *dual-core* AMD mostra-se muito ineficiente quanto aos *misses*. Já a Intel, por possuir um módulo grande de *cache* L2, consegue tratar estes erros com maior eficácia. Por este motivo, como será visto adiante, a utilização das *caches* L3 nos modelos *quad-core* Opteron e Phenom é um trunfo na abordagem de *caches* privadas da AMD.

6.2.2 Leitura e Escrita de Dados

Visivelmente, as *caches* de nível 2 realizam muito mais transações sobre dados do que sobre instruções. Destas, a maioria é de escrita de dados, oriundos das *caches* de nível 1 e dos processadores. A Figura 6.25 deixa clara a disparidade entre transações de

leitura e de escrita nas *caches* L2. Desta forma, e ao contrário do que ocorre nas *caches* L1, é correto imaginar as *caches* L2 mais como um componente de armazenamento de dados eliminados das *caches* L1 ou que necessitaram de atualização por controle de coerência, do que propriamente um banco de consulta dos mesmos.

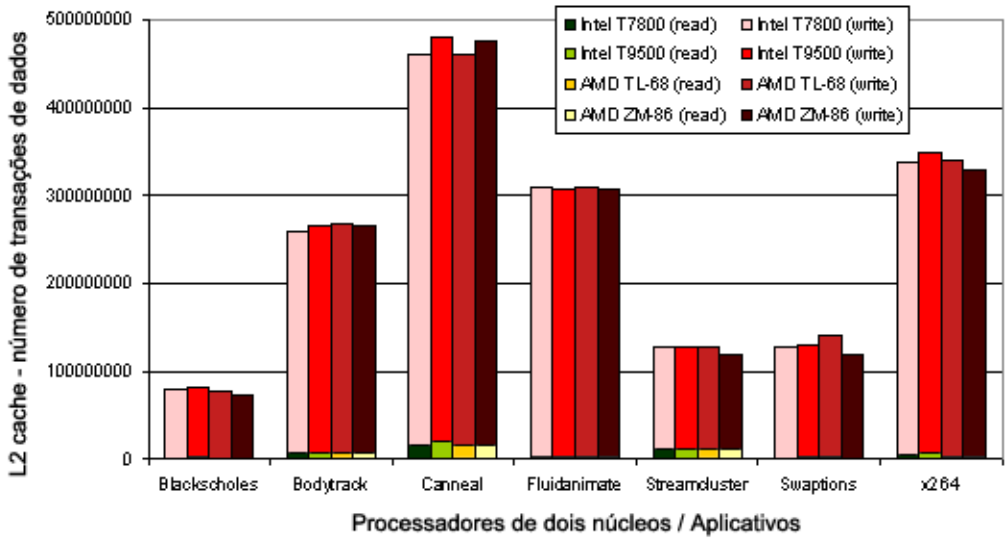

Figura 6.25: Número médio de transações de dados das *caches* L2 para os processadores de dois núcleos

Diferentemente das transações de instruções, há um grande equilíbrio no número de transações de dados das arquiteturas de dois *cores* da Intel e da AMD. Destas, como já foi dito, a imensa maioria é de escrita de dados na *cache*. Não se pode esquecer que a *cache* L2 dos modelos Intel é compartilhada pelos dois núcleos, sendo os valores apresentados no gráfico uma média das transações que cada núcleo realiza no módulo de *cache*. Outro fator importante é o menor tamanho das *caches* L1 dos modelos Intel, o que obriga a escrita de dados na *cache* L2 com maior freqüência. Mesmo assim, os números das famílias Core 2 Duo e Turion foram muito próximos.

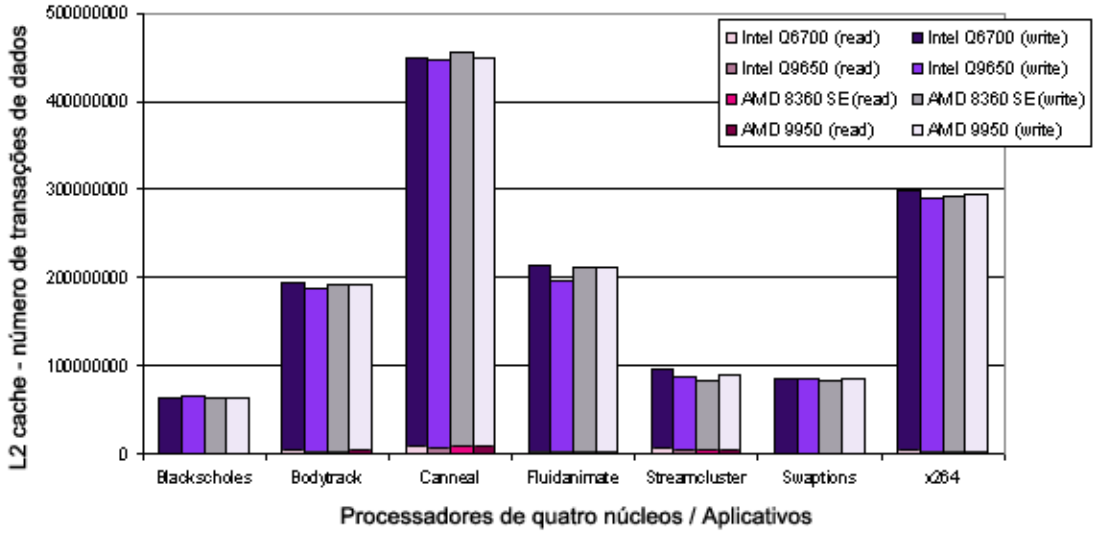

Figura 6.26: Número médio de transações de dados das *caches* L2 para os processadores de quatro núcleos

Quanto às famílias de quatro núcleos, os resultados foram semelhantes aos obtidos pelos modelos *dual-core*, como pode ser visto na Figura 6.26. As arquiteturas de ambos fabricantes obtiveram grande paridade nos resultados. É possível notar que o número de transações de leitura de dados diminuiu razoavelmente dos modelos de dois núcleos para os de quatro processadores, possivelmente pela maior divisão das tarefas e maior associatividade das *caches* nível 1.

Com valores equilibrados de transações, os números de ciclos de latência das transações de dados entre as arquiteturas tendem a manter este equilíbrio, já que o fator que irá diferenciar os resultados será as latências das *caches* de cada modelo. Como é possível ver na Figura 6.27, as latências também mantiveram-se equilibradas, com os maiores valores justamente nos modelos de maior latência, o T7800 da Intel e o ZM-86 da AMD. Além disso, interessante verificar a grande paridade existente entre o Intel T9500 e o AMD TL-68. De forma análoga ao número de transações, a latência das transações de leitura é ínfima comparada ao número de ciclos das transações de escrita.

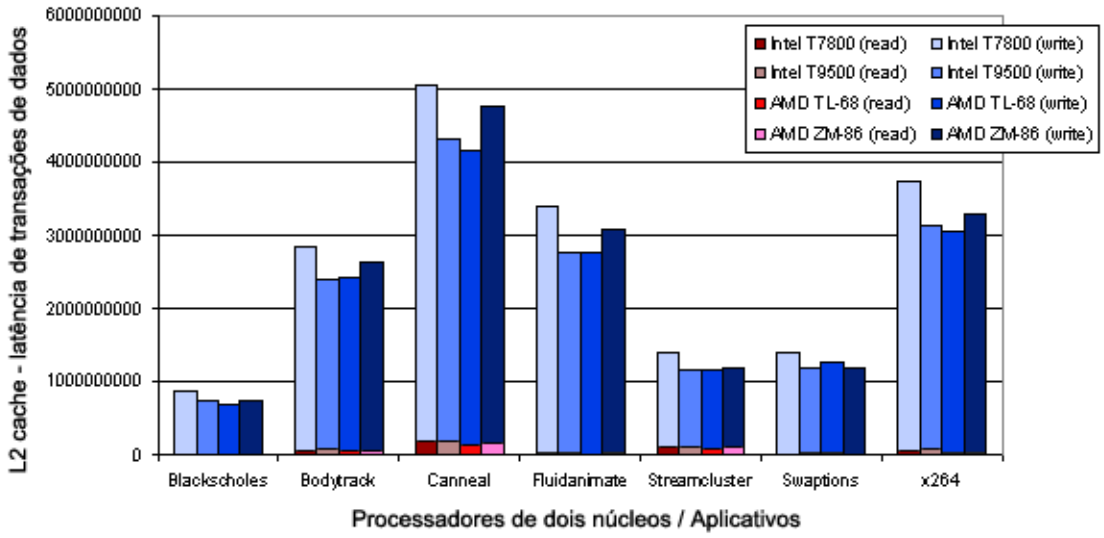

Figura 6.27: Número médio de ciclos de latência de dados das *caches* L2 para os processadores de dois núcleos

Já nas arquiteturas de quatro núcleos, o modelo Q6700 da Intel destacou-se com relação ao número de ciclos comparado aos outros três modelos, devido a sua latência significativamente maior que as dos outros. Em contrapartida, o AMD Opteron 8360SE, que possui a menor latência entre os *quad-core* estudados, conseguiu os melhores resultados, mesmo nos *benchmarks* em que obteve maior número de transações.

Outro ponto importante a se destacar é o maior número de ciclos de latência do modelo Q6700 da Intel comparado aos modelos de quatro núcleos em alguns aplicativos, em especial o Canneal. Mesmo com uma maior divisão das tarefas e menor número de transações, a alta latência agregada aos seus módulos de *cache* L2 torna o seu desempenho nos *benchmarks* 38,78% pior que os outros modelos de quatro núcleos.

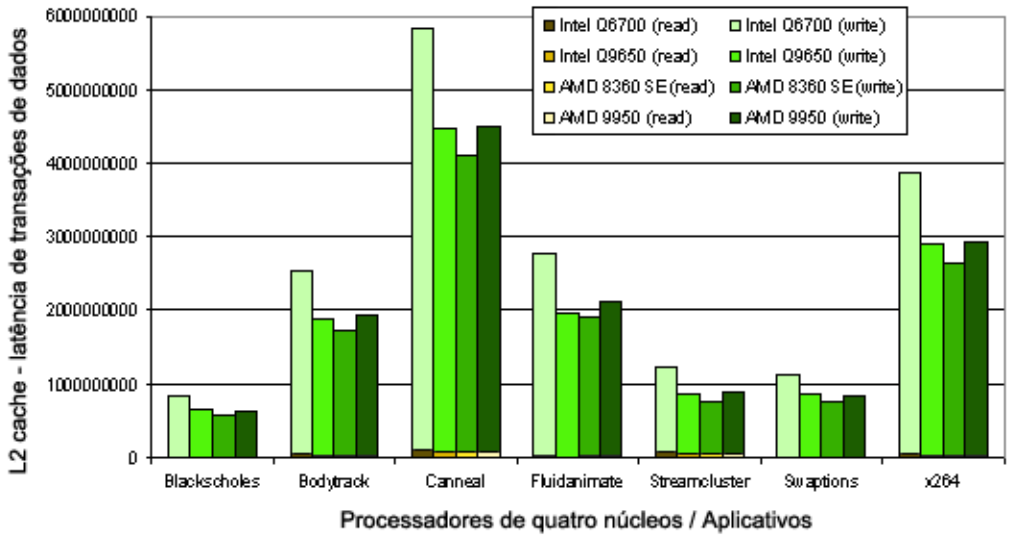

Figura 6.28: Número médio de ciclos de latência de dados das *caches* L2 para os processadores de quatro núcleos

Na relação dos *misses* de leitura e escrita de dados, temos resultados contrários aos de quantidade de transações. Em geral, as arquiteturas apresentam maior número de *misses* de leituras do que de escritas, o que é esperado. Esta relação faz com que as taxas de acerto de leitura sejam bastante inferiores às taxas de escrita.

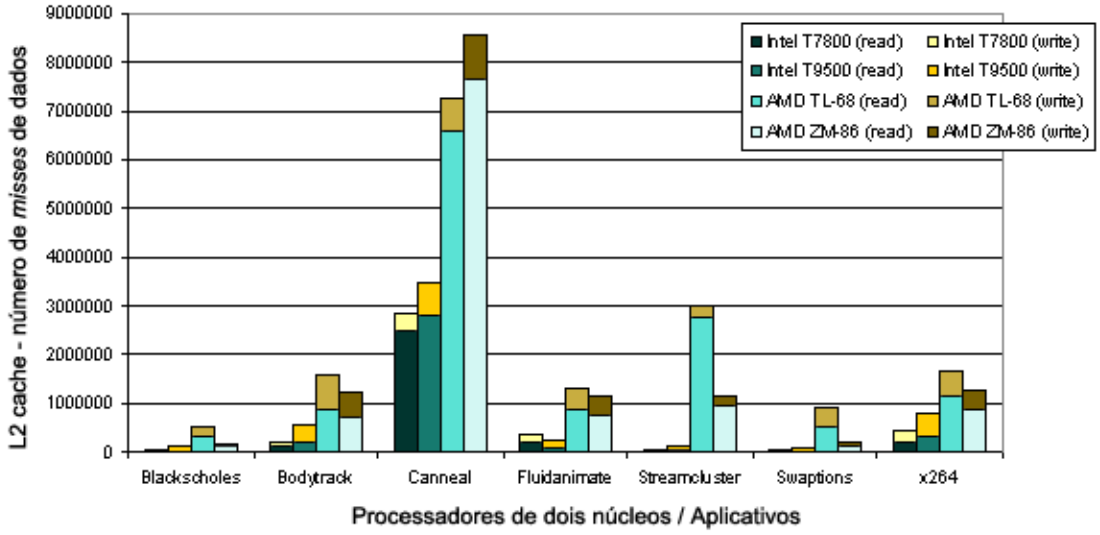

Figura 6.29: Número médio de *misses* de dados das *caches* L2 para os processadores de dois núcleos

Nota-se que nos *benchmarks* que requerem maior número de transações, naturalmente, são obtidos os maiores valores de erros. Percebe-se, nos modelos de dois núcleos, na Figura 6.29, que a família AMD Turion possui quantidades muito maiores de *misses* de dados, de forma análoga ao que já foi verificado nos *misses* de instruções. É possível verificar, em certos aplicativos, que a quantidade de erros das arquiteturas Intel é ínfima comparada ao número de *misses* dos modelos AMD. Nos piores casos, os *misses* dos processadores Intel chegam à metade dos valores obtidos pela AMD.

Novamente, a relação do número de erros de leitura e escrita de dados das *caches* L2 da AMD é muito insatisfatória.

Nas arquiteturas *quad-core*, com exceção do *benchmark* Canneal, os erros nas transações de dados foram mais equilibrados, não existindo diferenças tão gritantes entre os modelos da AMD e Intel. Já neste *benchmark* em específico, a Intel teve um desempenho bastante superior, em ambos os modelos. Mesmo com o equilíbrio verificado entre os resultados, há uma tendência de melhor desempenho dos processadores Intel nas cargas de trabalho.

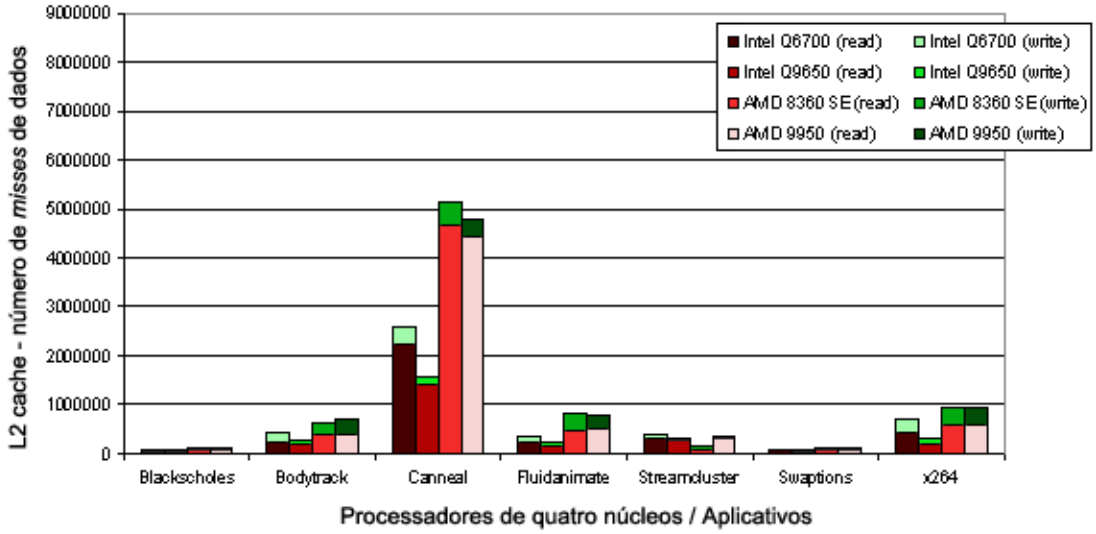

Figura 6.30: Número médio de *misses* de dados das *caches* L2 para os processadores de quatro núcleos

Obviamente, neste panorama de maior paridade nos resultados de número de erros de transações de dados, as latências dos níveis superiores da hierarquia de *cache*, se elas existirem, farão grande diferença no desempenho total. Desta forma, arquiteturas que utilizam um terceiro nível na hierarquia de *cache*, sobre um segundo nível que efetua considerável número de *misses*, acabam tendo certa vantagem, como poderá ser verificado adiante.

Todos os modelos de dois núcleos, tanto Intel quanto AMD, ao ocorrer *misses* na *cache* L2, acessam a memória principal. Isto faz com que o gráfico de ciclos de latência de *misses* de dados seja muito parecido com o gráfico de número dos mesmos *misses*. Isto é apresentado na Figura 6.31, onde percebe-se que os modelos Intel levam vantagem no tamanho da latência devido à menor quantidade de erros de leitura e escrita efetuados. Neste caso, a latência total é diretamente proporcional ao número de *misses*, já que o gráfico apresentado é exatamente o mesmo da Figura 6.29, apenas multiplicado pela latência da RAM.

Na Figura 6.32 aparece o efeito esperado pela utilização do terceiro nível de memória *cache*. Os modelos de quatro núcleos da AMD, que tiveram em geral maior número de *misses*, conseguiram diminuir praticamente pela metade o número de ciclos de latência, comparados aos modelos da Intel, através da utilização da *cache* L3. Este aumento no desempenho torna-se mais visível nas aplicações que realizaram maior número de erros, como Canneal e x264.

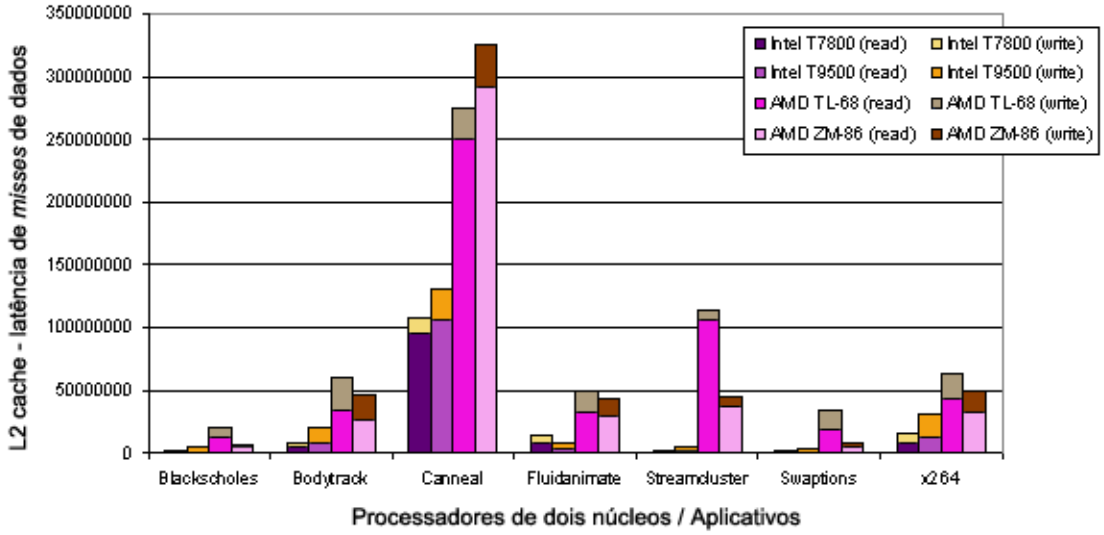

Figura 6.31: Número médio de ciclos de latência de *misses* de dados das *caches* L2 para os processadores de dois núcleos

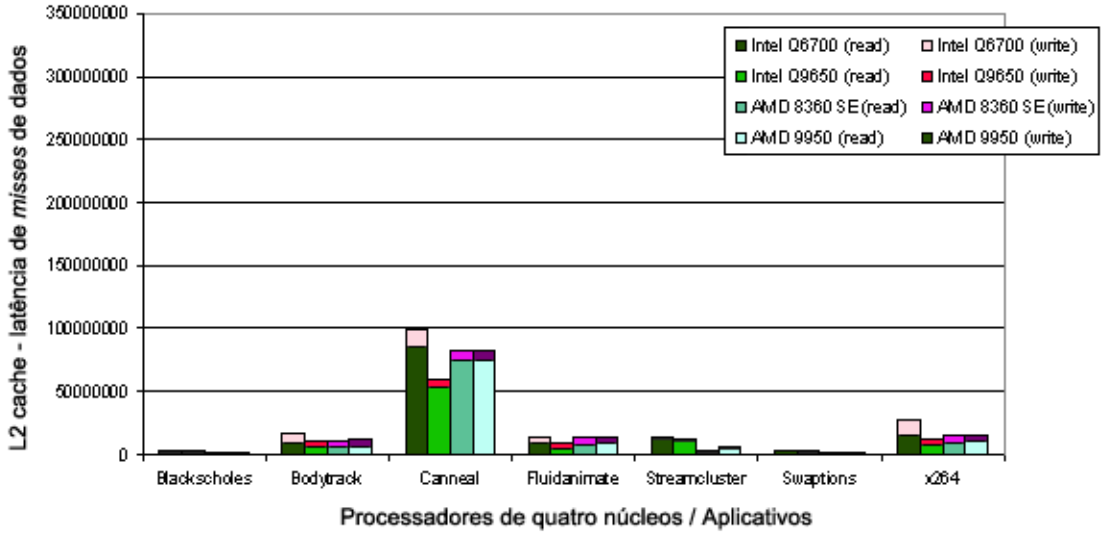

Figura 6.32: Número médio de ciclos de latência de *misses* de dados das *caches* L2 para os processadores de quatro núcleos

Pela utilização de *caches* nível 2 privadas nos modelos AMD, a tendência é de que os dados armazenados nas *caches* tornem-se esparsos e repetidos dentro de diferentes módulos. Além disso, a modificação de um dado em uma *cache* implicará, em algum momento, no tratamento de coerência deste dado em memória principal. Neste ambiente, a *cache* L3 surge com o intuito de centralizar os dados utilizados pelas *caches* L2 e de facilitar o tratamento de coerência, onde ao invés de uma atualização em memória principal, realiza-se a modificação na *cache* L3. Obviamente, a *cache* de nível 3 possui um custo de latência e número de transações associado, os quais serão vistos adiante.

6.2.3 Transações MESI

As transações MESI representam as alterações nos rótulos de cada linha de dados da *cache*, quando este dado sofre modificação sob o ponto de vista de coerência de *cache*. O Simics trata basicamente de dois tipos de transações MESI: *Modified to Shared* (quando um processador solicita um dado que está modificado em outra *cache*) e *Invalidates* (quando um processador modifica um dado em sua *cache* que também está localizado em outras *caches*).

A Figura 6.33 apresenta os resultados do número de transações MESI de cada um dos tipos. É possível notar uma paridade entre a quantidade de transações realizadas de cada tipo.

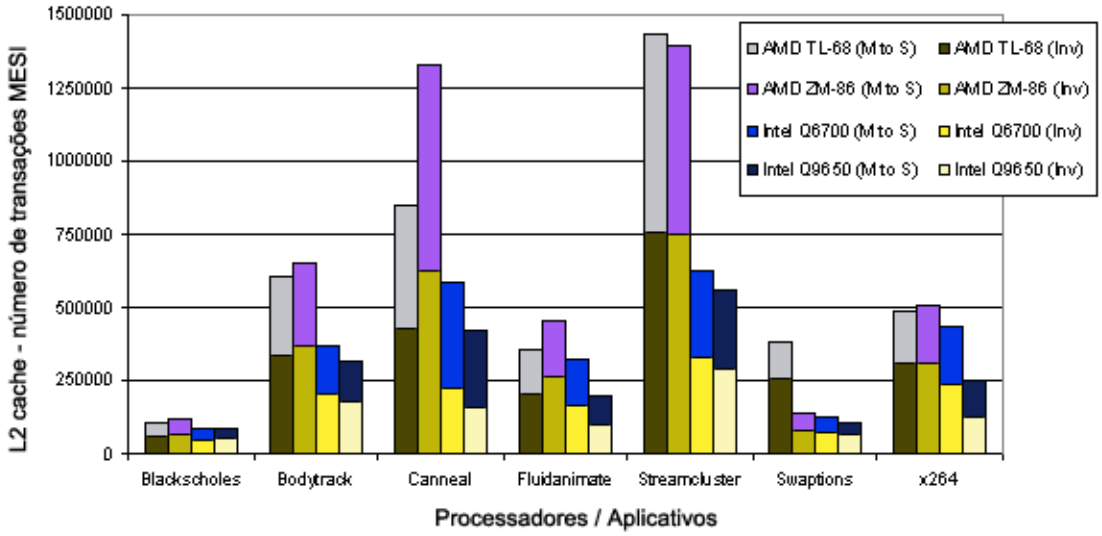

Figura 6.33: Número médio de transações MESI das *caches* L2 para os processadores

No gráfico estão listados dois modelos *dual-core* e dois modelos *quad-core*, que são os modelos utilizados neste trabalho que empregam protocolo MESI de coerência de *cache*. Nota-se que não há um padrão bem definido nos desempenhos das *caches* L2 destas arquiteturas com relação às transações MESI executadas. Entre os processadores de dois núcleos, ambos alternam-se nos resultados, ora um apresentando mais transações, ora outro. Já entre os modelos de quatro núcleos, em todos os *benchmarks* o Q6700 obteve maior número de transações MESI comparado ao Q9650.

Interessante analisar que o número de transações MESI acompanha o número de transações totais de cada arquitetura. Os modelos AMD, de dois núcleos, realizam número médio de transações de coerência significativamente maior que os *quad-cores* da Intel. Além disso, embora os modelos Intel possuam maior número de núcleos, o compartilhamento das *caches* L2 a cada par de processadores intrinsicamente diminui a freqüência de atualizações dos dados.

6.2.4 Transações de *Copy Back*

As *caches* de nível 2 (assim como as de nível 3 que serão analisadas adiante) empregadas nas arquiteturas do estudo deste trabalho utilizam a política de escrita do tipo *write-back*. Nesta técnica, quando algum dado da *cache* é modificado, não há a escrita deste novo valor na memória principal (ou em algum nível superior de *cache*) de forma imediata. A atualização do dado é realizada somente quando este dado é

requerido por outra unidade de *cache*, ou se este dado contido nesta *cache* é escolhido como “vítima” para dar lugar a um novo dado buscado. Para que este mecanismo funcione, em geral utiliza-se o chamado *dirty bit*, que indica que este dado foi modificado e ainda não foi atualizado na memória principal.

A política de *write-back* é muito utilizada pois reduz os acessos de escrita na memória, aumentando o desempenho do sistema. Além disso, diversas escritas sobre um mesmo dado na *cache* requerem apenas uma atualização na origem do dado, diminuindo a latência final do sistema.

As transações *copy back* são justamente as atualizações na memória principal dos dados modificados nas *caches*. As figuras 6.34 e 6.35 apresentam os números de transações deste tipo nas arquiteturas trabalhadas.

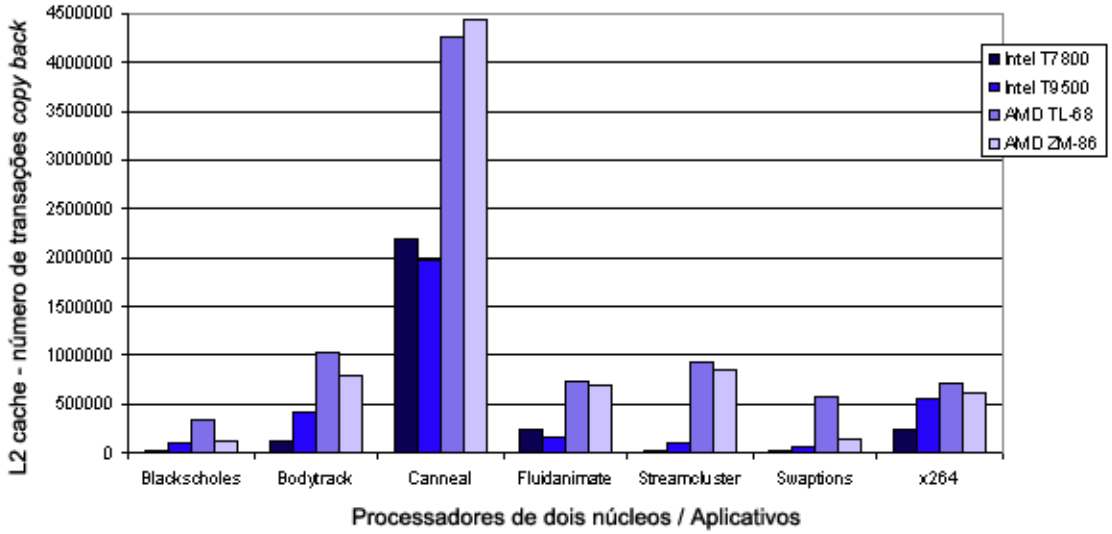

Figura 6.34: Número médio de transações *copy back* das *caches* L2 para os processadores de dois núcleos

Como pode ser visto na Figura 6.34, em todas as cargas de trabalho, os modelos da AMD de dois núcleos realizam maior número de transações *copy back*. Isto pode ser explicado pela menor capacidade de armazenamento das *caches* L2 das arquiteturas AMD, que com maior freqüência encontram-se cheias e necessitam retirar algum dado já existente na *cache* para que um novo dado requisitado seja acumulado.

Nota-se uma certa propensão, em grande parte dos *benchmarks*, de haver uma diferença proporcional entre os dois modelos da Intel e entre os dois modelos da AMD. Outro dado interessante é a diferença do número de transações de atualização realizadas pelo *benchmark* Canneal comparado aos outros aplicativos.

Na Figura 6.35, que representa os resultados das arquiteturas de quatro processadores, nota-se um maior equilíbrio nos resultados, com uma maior participação das transações *copy back* nos modelos Q6700 da Intel e Opteron 8360SE da AMD. Além disso, assim como já ocorreu em outras análises, as duas famílias AMD obtiveram valores muito próximos, provando que duas arquiteturas idênticas – inclusive com os mesmos tamanhos de *cache* – devem apresentar comportamento semelhante. Outro ponto a ser comentado é o satisfatório desempenho do Q9650, principalmente nas aplicações de maior porte, obtendo uma diferença interessante com relação aos demais modelos.

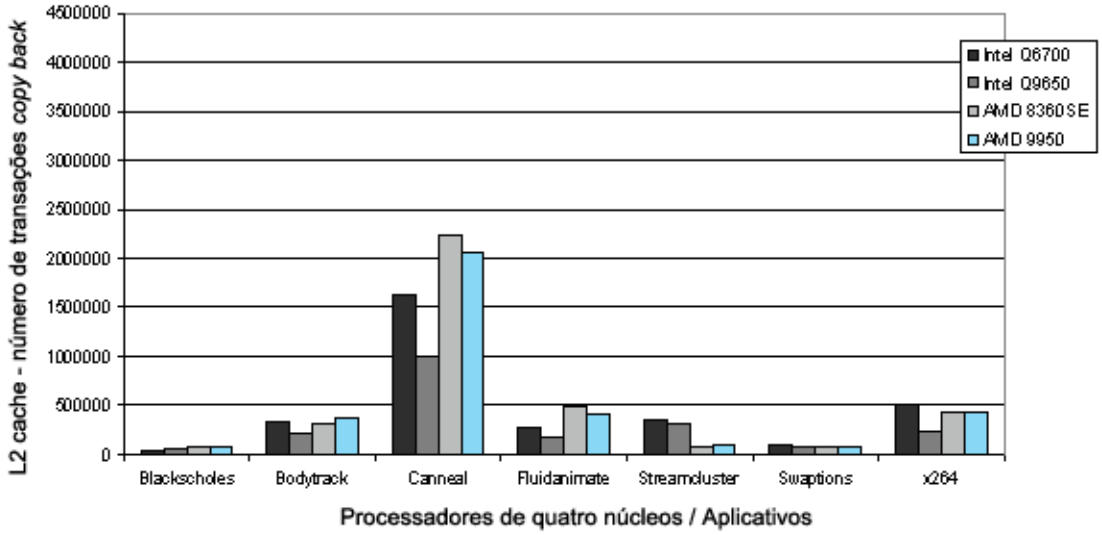

Figura 6.35: Número médio de transações *copy back* das *caches* L2 para os processadores de quatro núcleos

Importante destacar que cada transação MESI do tipo *Modified to Shared*, vistas na seção 6.2.3, origina uma transação *copy back*, devido à necessidade da *cache* de atualizar o dado em questão na memória principal (no caso dos *quad-cores* da AMD, na *cache* de nível 3) para que outro processador possa lê-lo.

6.3 Memória *Cache* de Nível 3

Componente ainda pouco utilizado nas arquiteturas de computadores, presente apenas nos modelos mais recentes de *multicore*. Em geral, emprega-se *caches* de nível 3 em sistemas com quatro ou mais núcleos, onde existem muitos módulos de *cache* de nível 2, o que aumenta o número de acessos à memória principal e conseqüentemente a latência do sistema.

Das arquiteturas estudadas, apenas as duas famílias de quatro núcleos da AMD possuem *cache* L3, compartilhada entre os quatro núcleos e suas *caches* L2 privadas. Este terceiro nível de *cache* tem função semelhante às *caches* de nível 2: armazenar dados e acelerar leituras.

Neste subcapítulo, como ambas arquiteturas possuem o mesmo número de núcleos e módulos de *cache*, os valores apresentados como resultados são simplesmente os números totais obtidos pelos *caches* L3, não sendo estes valores divididos pelo número de núcleos de cada sistema.

6.3.1 Busca de Instruções

Assim como nos resultados de busca de instruções das *caches* L2, novamente nota-se na Figura 6.36 uma grande quantidade de transações deste tipo realizadas na execução do aplicativo x264. Na média, os valores obtidos por ambos os modelos são próximos, sendo possível verificar esta paridade nas aplicações Blackscholes, Bodytrack e Swaptions. O único resultado mais expressivo é no *benchmark* Fluidanimate, onde as transações mais que dobram de um modelo para o outro.

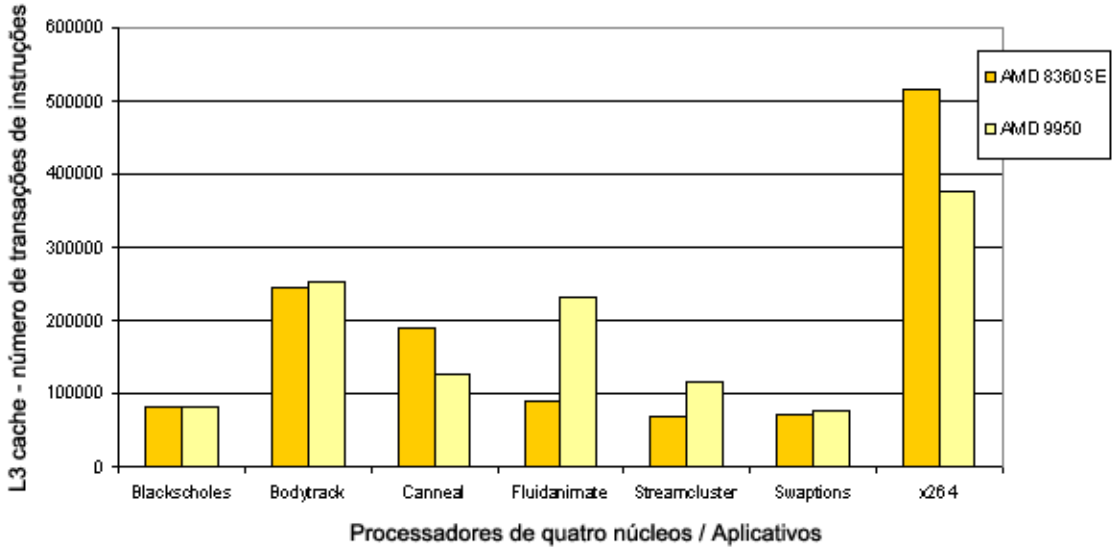

Figura 6.36: Número médio de transações de instruções da *cache* L3 para os processadores de quatro núcleos da AMD

Verificando o gráfico das latências das transações efetuadas sobre a *cache* L3, nota-se que o modelo Opteron 8360SE, que possui latência levemente menor, consegue aumentar a diferença com relação ao Phenom 9950 nas aplicações em que realizou menor número de transações, e diminuir a diferença nos *benchmarks* em que efetuou mais transações. Isso prova que, mesmo no terceiro nível de *cache*, onde o número de transações é bastante reduzido, comparado aos níveis anteriores, e o número de erros de busca é bastante elevado, como veremos a seguir, a utilização de latências de menor valor continua sendo de suma importância.

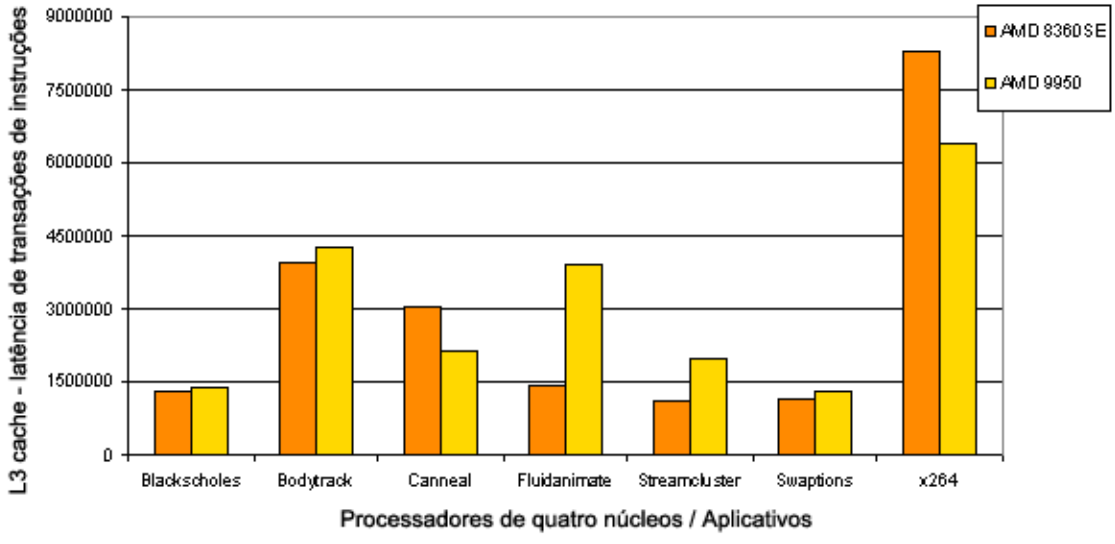

Figura 6.37: Número médio de ciclos de latência de instruções da *cache* L3 para os processadores de quatro núcleos da AMD

A Figura 6.38, muito parecida com a Figura 6.36, apresenta a quantidade de erros na busca de instruções de cada processador. A semelhança nos resultados de *misses* com o número de transações é muito grande, o que permite afirmar que a quantidade de erros é praticamente proporcional à quantidade de transações. Por mais

que esta relação seja facilmente feita, os gráficos das duas figuras exemplificam com propriedade isto.

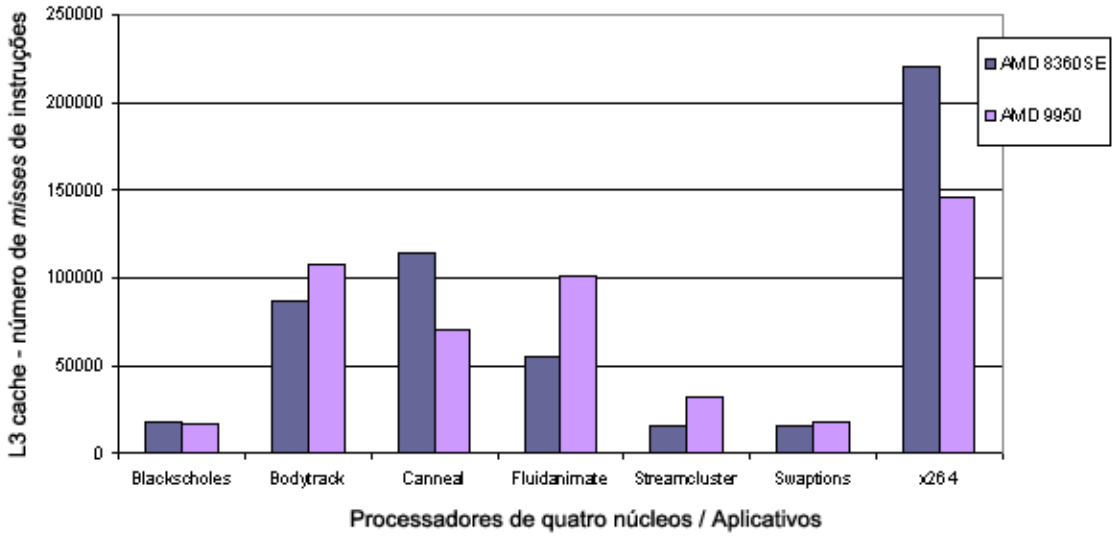

Figura 6.38: Número médio de *misses* de instruções da *cache* L3 para os processadores de quatro núcleos da AMD

Um ponto importante a se destacar é o elevado número de *misses* ocorridos nas *caches* de nível 3, comparado à quantidade total de transações realizadas pelas mesmas. É possível verificar nas tabelas de resultados, encontradas no apêndice deste trabalho, que as taxas de acertos de busca de instruções destes módulos de *cache* L3 são relativamente baixas ou de valor intermediário, com índices médios inferiores a sessenta por cento de acerto. Por tratar-se de um nível de *cache* de certa forma distante dos processadores, é presumível que as taxas deste não sejam muito elevadas, e que realmente ocorra grande número de *misses*.

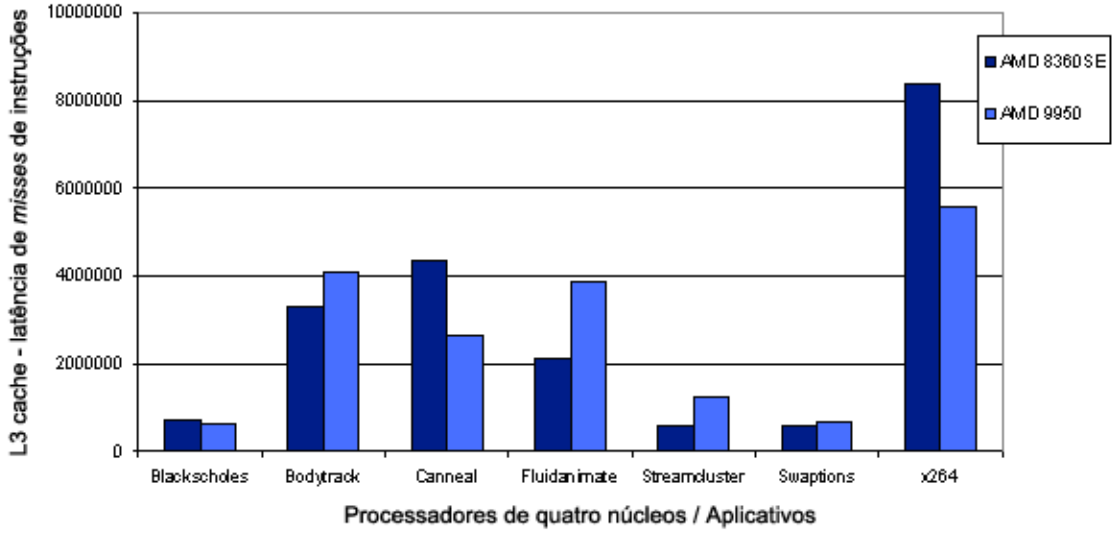

Figura 6.39: Número médio de ciclos de latência de *misses* de instruções da *cache* L3 para os processadores de quatro núcleos da AMD

Mesmo assim, criando uma relação entre número de transações e de *misses*, verifica-se que alguns aplicativos conseguiram manter um baixo número de erros

proporcionais, como o Blackscholes e o Streamcluster. Interessante notar que no *benchmark* Fuidanimate, enquanto o modelo Opteron teve um aumento na relação de erros por transações realizadas, o Phenom conseguiu diminuir esta relação.

Já na questão da latência de *misses*, o número de ciclos de ambos os modelos será diretamente ligado ao número de erros, dado que nos dois casos haverá acesso à memória principal. Na Figura 6.39 verifica-se esta proporcionalidade, a partir da semelhança que o gráfico contido possui com o apresentado anteriormente de número de *misses*.

6.3.2 Leitura e Escrita de Dados

De forma contrária ao comportamento das *caches* L2, dentre as transações sobre dados, as que mais de destacaram nas *caches* L3 foram as de leitura. Isto pode ser explicado devido ao tamanho razoável das *caches* de nível 2, o que diminui a freqüência de escritas em um nível superior de memória.

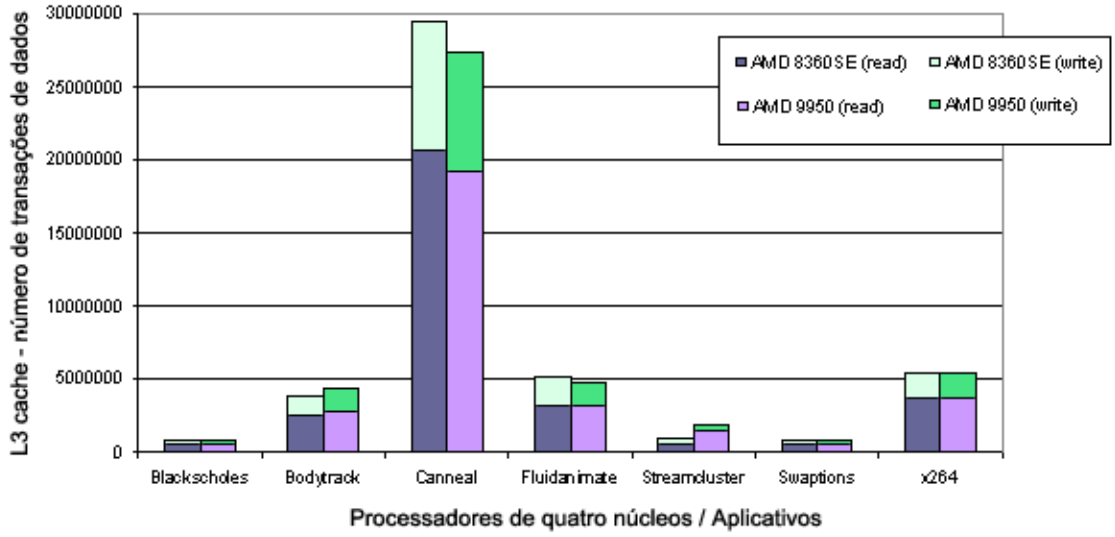

Figura 6.40: Número médio de transações de dados da *cache* L3 para os processadores de quatro núcleos da AMD

É possível notar que os números de escritas de dados foram bem próximos, sendo as maiores diferenças entre os processadores nas leituras de dados. Como em todos os gráficos de comparação entre dados, o *benchmark* Canneal teve destaque no número de transações. Em compensação, outros aplicativos como Blackscholes e Swaptions pouco requereram as *caches* de nível 3 destes processadores. Isto pode ser observado na Figura 6.41, que apresenta a latência das transações de dados das *caches* L3. Estes dois últimos aplicativos tiveram números pouco expressivos de ciclos nestes módulos, comparados ao número total de transações realizadas pelos *benchmarks*.

Nota-se também como a menor latência do Opteron 8360SE novamente ofereceu certa vantagem a este modelo comparado ao Phenom 9950. Apesar disso, tratando-se de um gráfico de latências de pequena escala, pode-se dizer que em geral os dois processadores são muito próximos quanto aos tempos gastos com transações em suas *caches* de nível 3.

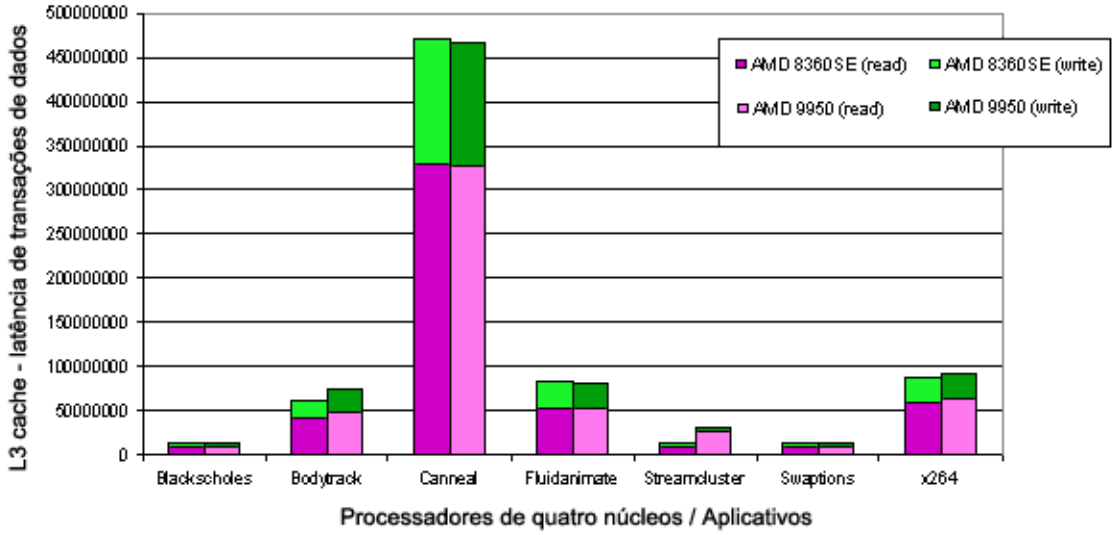

Figura 6.41: Número médio de ciclos de latência de transações de dados da *cache* L3 para os processadores de quatro núcleos da AMD

Verifica-se, a partir da análise da Figura 6.42, a mesma tendência da quantidade de erros ser relacionado com a quantidade de transações, verificada anteriormente na busca de instruções. Também percebe-se que o número de *misses* de leitura são muito maiores que os de escrita, principalmente pelo fato de possuir grande capacidade de armazenamento e de ser basicamente um módulo de busca de dados. Novamente nota-se a pequena utilização da *cache* de nível 3 por parte dos *benchmarks* de menor porte, efetuando poucos erros de transações.

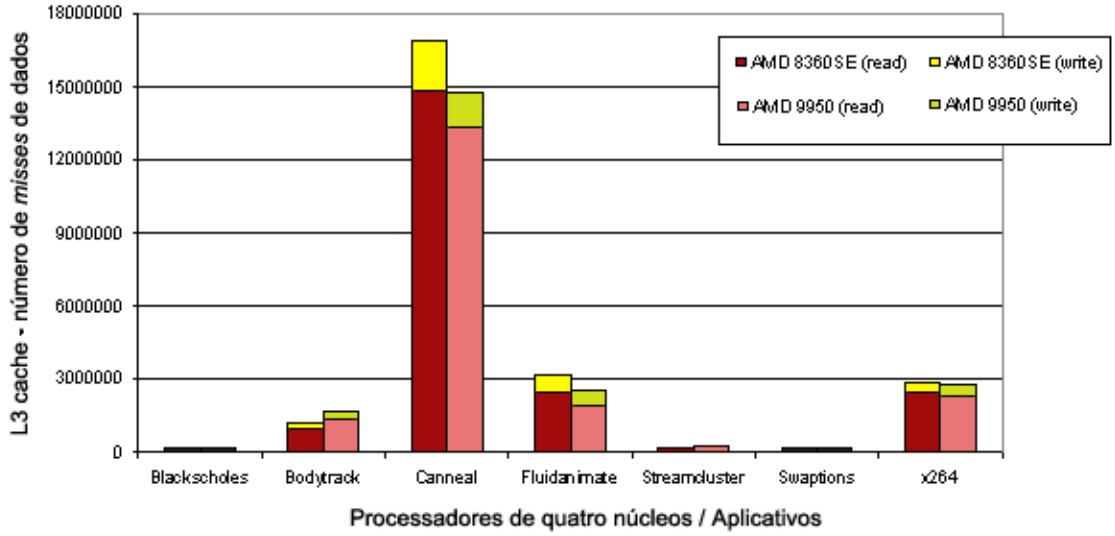

Figura 6.42: Número médio de *misses* de dados da *cache* L3 para os processadores de quatro núcleos da AMD

Com os valores apresentados anteriormente de quantidades de *misses*, tem-se os números de ciclos de latência apresentados na Figura 6.43. Pelo fato de a latência de acesso à memória de ambos modelos ser a mesma, os resultados da latência de *misses* acompanhou o desempenho apresentado pelo número de erros de transações.

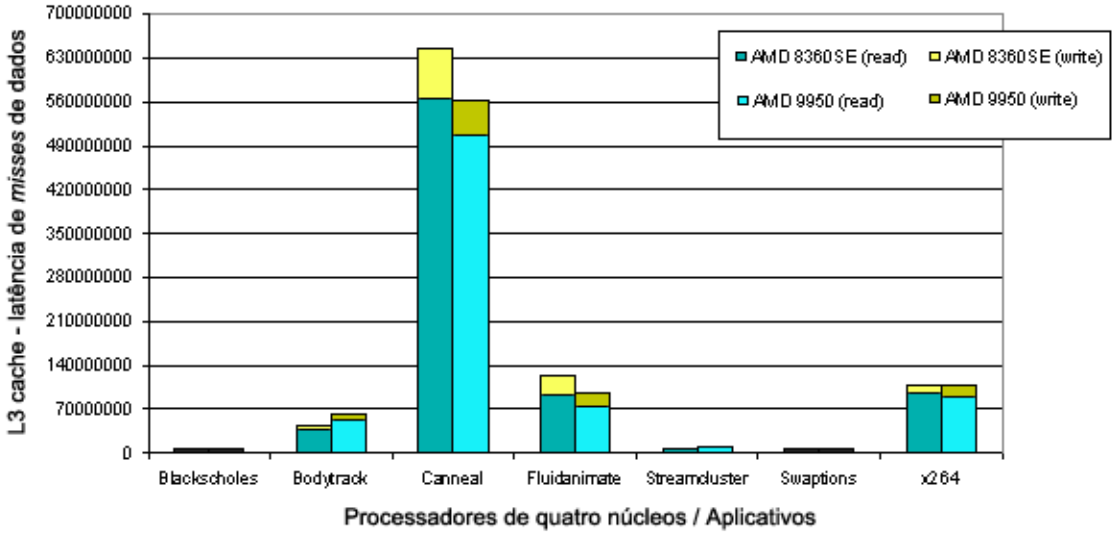

Figura 6.43: Número médio de ciclos de latência de *misses* de dados da *cache* L3 para os processadores de quatro núcleos da AMD

6.3.3 Transações de *Copy Back*

De forma análoga aos erros nas leituras e escritas de dados na *cache*, as transações de *copy back* também são relacionadas à quantidade de transações efetuadas. O gráfico da Figura 6.44 mostra como os resultados dos dois processadores foram semelhantes, com valores ligeiramente maiores no modelo Opteron 8360SE, principalmente nas aplicações de maior execução.

É possível verificar que o número total de transações *copy back* das *caches* de nível 3 foi bastante inferior que nas *caches* de nível 2 destes mesmos modelos, apresentados na Figura 6.35.

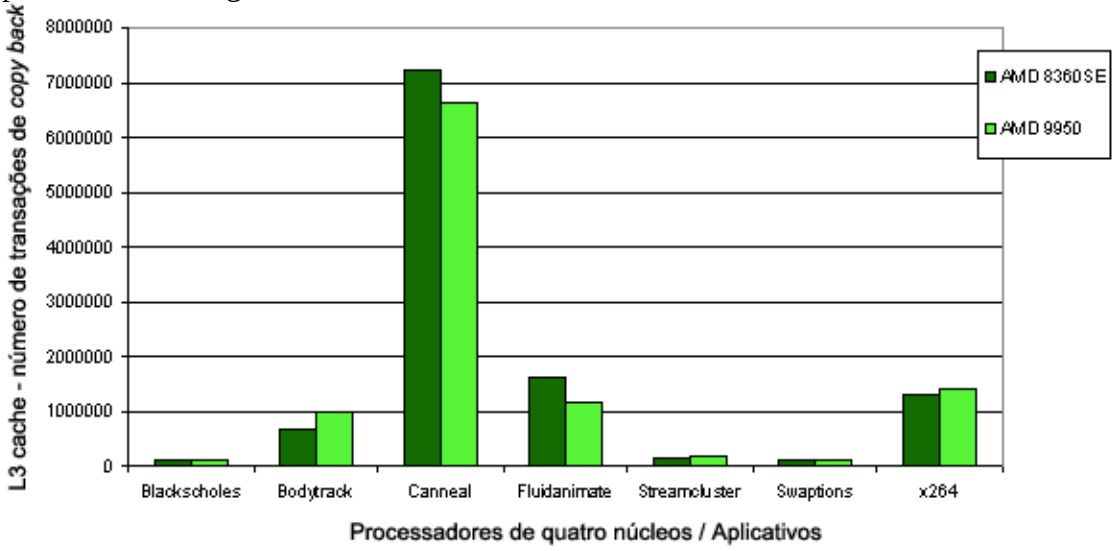

Figura 6.44: Número médio de *misses* de dados da *cache* L3 para os processadores de quatro núcleos da AMD

Não se pode esquecer que os números de transações *copy back* nas *caches* L2, apresentados anteriormente, representam a média por número de núcleos, no caso, quatro *cores*. Desta forma, verifica-se uma quantidade bastantes maior deste tipo de

transação nas *caches* de hierarquia inferior. Outro motivo para este menor número é a ausência nestes módulos de protocolos de coerência de *cache*, que acarretam grande número de transações de atualização.

6.4 Ciclos de Execução

Os ciclos de execução de cada aplicativo dependem das transações que este executa, dos estados das *caches* e das latências ocorridas, tanto nas transações de dados e instruções quanto nos *misses*.

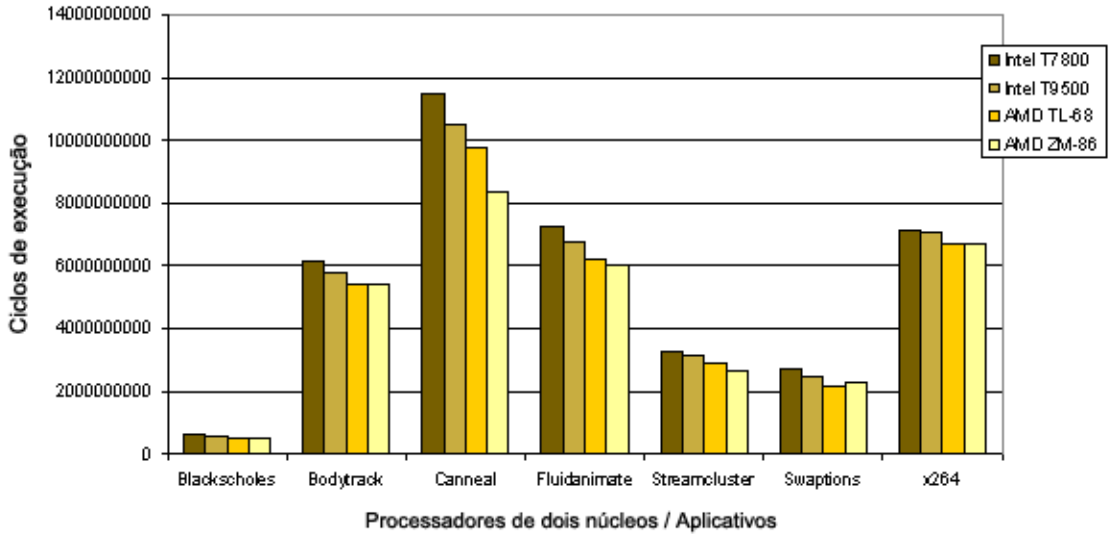

Figura 6.45: Número de ciclos para os processadores de dois núcleos

Analisando as figuras 6.45 e 6.46 é possível notar que os modelos da AMD executaram os *benchmarks* utilizando menor número de ciclos. Também é possível verificar, assim como era esperado, menores números de ciclos nos modelos de quatro núcleos, comparados aos de dois núcleos. Mas verifica-se exceções a esta tendência nos aplicativos Canneal e x264.

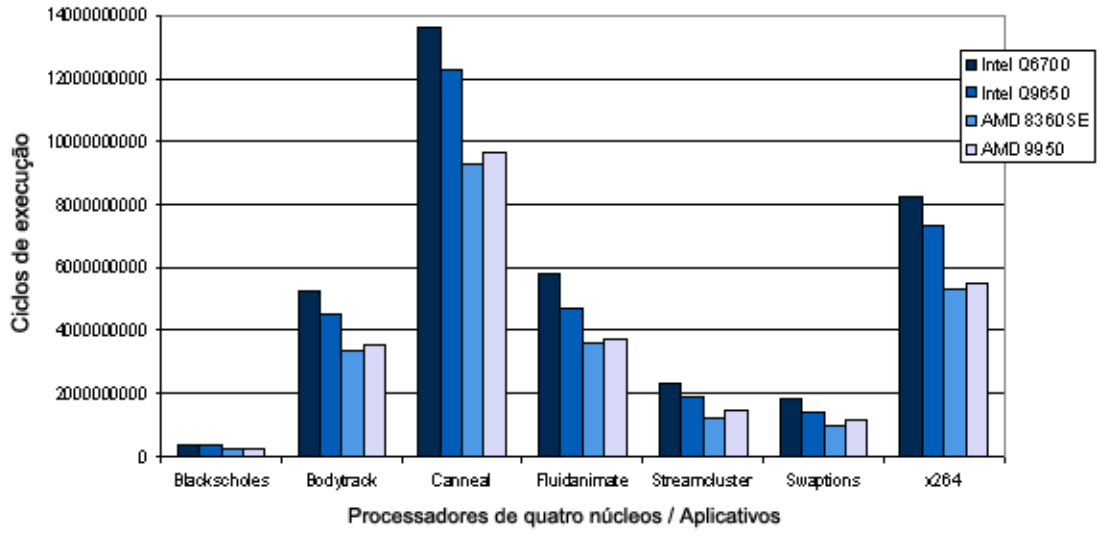

Figura 6.46: Número de ciclos para os processadores de quatro núcleos

Dentre os modelos de dois *cores*, o ZM-86 da AMD foi o que obteve a melhor média de resultados, e nitidamente o T7800 da Intel foi o de pior desempenho. Nos modelos *quad-core*, o Opteron 8360SE da AMD teve os menores números de ciclos de execução, e o Q6700 da Intel obteve os maiores valores.

Os números de ciclos apresentados refletem unicamente os resultados destes modelos sob o ponto de vista de suas arquiteturas de *cache*, sem levar em consideração características de cada processador que possam aumentar ou diminuir este desempenho. Pode-se dizer que, no aspecto das arquiteturas de *cache*, os modelos AMD saíram-se melhor que os da Intel quanto ao número de ciclos executados.

6.5 Tempo de Execução

O tempo de execução de uma aplicação em uma arquitetura depende não somente do número de ciclos que ela executa, mas também da freqüência em que os processadores são capazes de executar estes ciclos em determinado período.

Como já foi dito anteriormente, os tempos de execução foram calculados a partir do número de ciclos de execução das cargas de trabalho em cada arquitetura, obtido através do simulador, e da freqüência dos processadores de cada modelo. Ao dividir o número total de ciclos pela quantidade de ciclos que os processadores executam por segundo, é possível determinar um valor de tempo de execução bastante próximo ao obtido pelas arquiteturas reais.

Pelo fato dos processadores estudados possuírem freqüências bastante próximas, os gráficos dos tempos de execução tornaram-se muito semelhantes aos de número de ciclos, mas com pequenas melhorias principalmente nas famílias Intel.

É possível verificar na Figura 6.47 que os modelos da Intel tiveram resultados de tempo mais próximos dos obtidos pelos modelos da AMD, inclusive com alguns resultados menores, como nos *benchmarks* Bodytrack e x264.

Na comparação entre os modelos da Intel, o modelo T9500 teve melhores resultados que o T7800 em todos os aplicativos. Já entre os processadores da AMD, em apenas uma aplicação o ZM-86 não foi melhor que o TL-68.

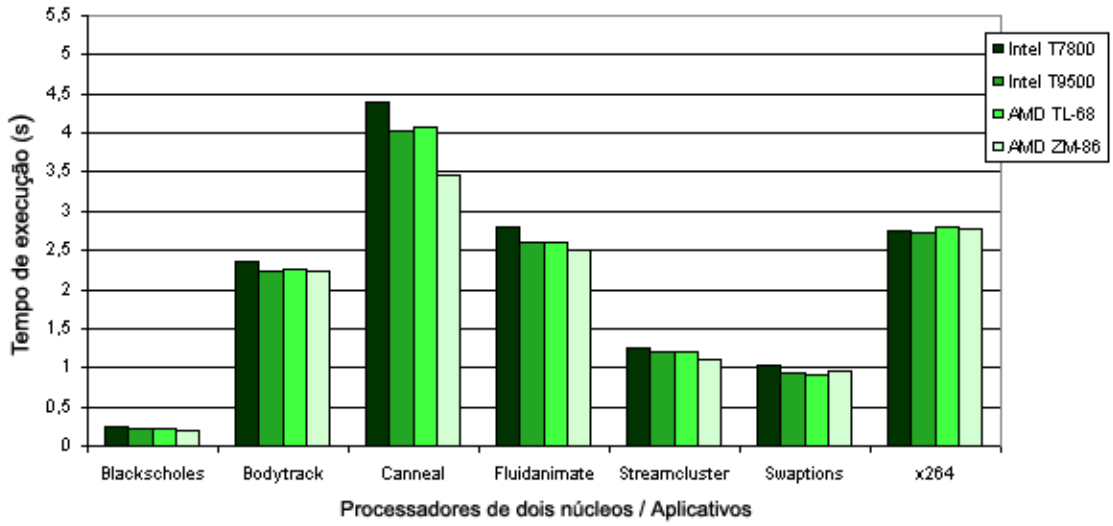

Figura 6.47: Tempo de execução para os processadores de dois núcleos

Entre os modelos de quatro núcleos, os dois da AMD tiveram tempos expressivamente melhores que os da Intel, com uma leve vantagem do Opteron sobre o Phenom. Observa-se que não só o número de ciclos mas também o tempo de execução dos processadores de quatro núcleos da Intel nos *benchmarks* Canneal e x264 foram bastante insatisfatórios.

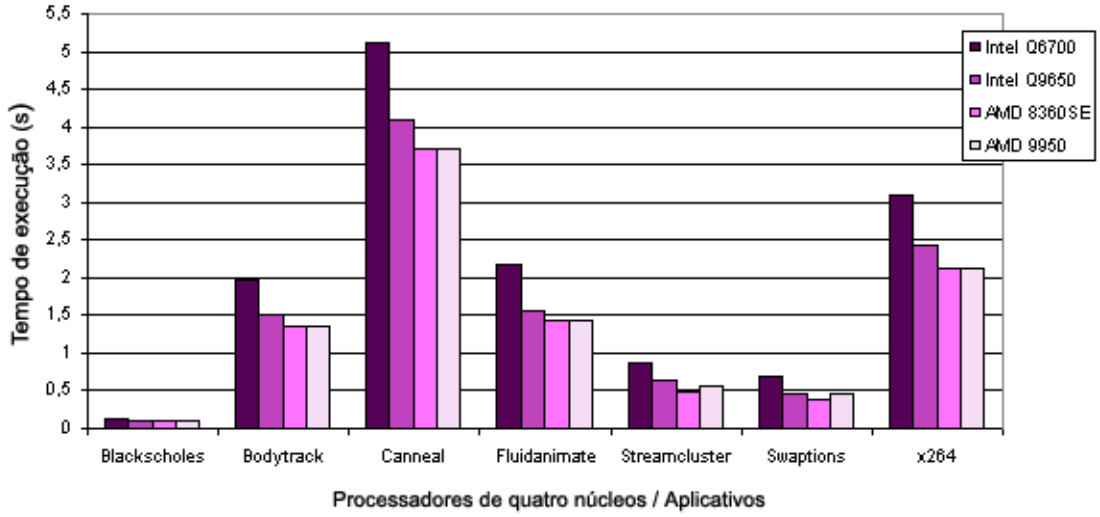

Figura 6.48: Tempo de execução para os processadores de quatro núcleos

É importante novamente ressaltar que estes resultados refletem basicamente as atividades das memórias *cache* dos modelos estudados, sem considerar otimizações implementadas nos processadores ou em outros componentes que não foram modelados neste trabalho. Estes tempos de execução reproduzem as análises realizadas sobre as *caches* de diversos níveis implementadas em cada arquitetura e suas funcionalidades. Mesmo com características bastante distintas e abordagens próprias pra cada família, notou-se um maior número de vantagens dos modelos da AMD com relação aos modelos da Intel, comprovadas, por fim, pelos seus tempos de execução.

7 TRABALHOS RELACIONADOS

Com a popularização dos microcomputadores e a necessidade cada vez maior de respostas rápidas e soluções eficientes, a análise do desempenho das máquinas paralelas tornou-se algo imprescindível na computação. Motivadas também pela concorrência cada vez mais acirrada entre fabricantes, diversas técnicas de *benchmarking* foram desenvolvidas nas últimas décadas.

Apesar disso, poucos trabalhos relacionados à verificação do desempenho de arquiteturas x86 foram realizados de forma científica até hoje. A grande maioria destas pesquisas ou são de cunho totalmente comercial, produzidas por determinado fabricante para demonstrar superioridade de seus produtos com relação aos concorrentes – o que pode gerar dúvidas sobre a imparcialidade da pesquisa – , ou são realizadas por revistas ou sítios na Internet sobre computadores, mas que geralmente comparam apenas determinados componentes, como processadores ou placas gráficas. Estudos mais aprofundados em desempenho, realizados por universidades ou grupos de pesquisa, ainda são bastante esparsos, principalmente quando se trata de arquiteturas x86.

Uma das grandes dificuldades para a avaliação de performance em máquinas paralelas, segundo Kambhatla, Inouye e Walpole (1990), é a diversidade das mesmas, onde uma arquitetura é completamente diferente de outra, o que em geral determina diversos estilos de programação distintos. Como comparar, mesmo que em apenas um determinado ponto da arquitetura, uma máquina de memória compartilhada com outra de memória distribuída? Eles tratam o problema da portabilidade do *benchmark* como ponto fundamental para a avaliação de máquinas paralelas. Para tentar diminuir este problema, eles utilizam uma arquitetura de *software* como uma camada acima do *hardware*, de forma a mascarar a diversidade entre arquiteturas.

Outro estudo foi realizado por Roberts e colaboradores (1988) sobre máquinas vetoriais. Sua comparação foi bastante focada na utilização da tecnologia VLSI em arquiteturas SIMD, verificando o consumo de cada elemento de processamento levando em conta o ganho de desempenho do sistema quanto ao incremento do número de processadores.

Estudos mais recentes utilizam técnicas de programação mais refinadas para avaliar o desempenho de multiprocessadores. Makai (2006) utiliza a heurística de algoritmos genéticos sobre arquiteturas paralelas. Esta abordagem é bastante interessante, pois como os algoritmos genéticos trabalham sobre populações e avaliação de seu *fitness* (aptidão da população para ser a melhor), dependendo do tamanho do conjunto de soluções, a busca pela melhor solução torna-se um problema exponencial. Desta forma, a utilização de arquiteturas paralelas, a fim de dividir o problema, torna-se uma opção bastante eficiente.

Talvez um dos trabalhos mais próximos a este apresentado seja um trabalho de Pandian e outros pesquisadores (1994), que simulou e avaliou o desempenho de arquiteturas paralelas baseadas no modelo i860 da Intel Corporation (ATKINS, 1991). Neste estudo, eles utilizam uma ferramenta chamada SapePar-i860 para simular e avaliar o desempenho de comunicação das *caches* com a memória e dos processadores uns com os outros. Além disso, calcula o *speedup* do sistema em termos dos ciclos de execução, utilizando quantidades variadas de processadores.

8 CONCLUSÕES

Neste trabalho foi apresentado um estudo sobre a análise de desempenho de arquiteturas paralelas x86, fabricadas pela Intel e AMD, sob o ponto de vista de suas memórias *cache*. Para isto, utilizou-se a ferramenta Simics para modelar e simular estas arquiteturas trabalhadas. Após a realização da modelagem e simulação destas, e os devidos testes de desempenho das mesmas, foi possível verificar o comportamento das máquinas em questão, sobre o âmbito de suas hierarquias de *cache*, componente com maior destaque neste trabalho.

Sobre as simulações, o Simics se mostrou uma ferramenta muito eficaz e apropriada para a realização do trabalho. Com ele foi possível modelar virtualmente os *hardwares* das máquinas estudadas e simulá-las com fidelidade nas execuções para a realização dos testes de desempenho. A partir dele foram obtidas as estatísticas referentes às *caches*, além do número de ciclos utilizados na execução dos aplicativos em cada arquitetura, sendo assim possível estimar os tempos de execução das arquiteturas reais. No que se refere à modelagem das arquiteturas de *cache*, o Simics o faz com grande detalhamento, analisando as transações realizadas por cada módulo da *cache*.

Quanto ao PARSEC, conjunto de cargas de trabalho escolhido, por ser um dos mais recentes e conceituados conjuntos de *benchmarks* na atualidade, seus resultados podem comprovar com grande fidúcia o comportamento de cada arquitetura. A partir da diversidade dos seus programas e áreas de aplicação, o PARSEC consegue ser abrangente na sua análise e funciona como excelente carga de trabalho inclusive para as avaliações de transações realizadas pelo simulador.

No estudo das arquiteturas, mostrou-se bem clara a diferença de abordagens utilizada por cada fabricante, tendo a Intel a utilização de uma hierarquia mais simples, com memórias *cache* L2 compartilhadas e módulos de *cache* com grande capacidade de armazenamento, enquanto a AMD trabalha com hierarquias mais complexas e isoladas, com *caches* privadas por núcleo e de menor tamanho.

Estas abordagens apresentaram resultados bastante distintos, com pontos relevantes em cada uma. As arquiteturas de *cache* de nível 2 da Intel, por possuírem memórias de maior tamanho, obtiveram em geral menor número de *misses* nas transações que as representantes da AMD. Com isso, nota-se um maior ganho nas buscas de instruções e leituras de dados, já que as escritas de dados tiveram bons resultados em todos os modelos. Também ficou claro que o aumento do tamanho da memória auxilia numa maior taxa de acertos, mas em proporções bem diferentes – o aumento das taxas é relativamente pequeno comparado ao incremento da memória.

As arquiteturas projetadas pela AMD tiveram desempenhos de execução (medido através do tempo de execução dos *benchmarks*) melhores que os modelos da Intel, executando os aplicativos com menor número de transações e ciclos. As *caches* de nível 3, dos modelos de quatro núcleos da AMD, conseguiram oferecer mais uma camada de dados e instruções ao processador, mas sendo realmente determinantes no desempenho das aplicações de maior porte. Neste quesito, podemos concluir que uma maior individualização das transações de cada processador em módulos privados de *cache* provê um melhor desempenho total na execução dos programas. Além disso, como já era esperado, módulos de *cache* com maior tamanho tendem a ter melhores resultados estatísticos de acertos, tanto em busca de instruções quanto em leitura e escrita de dados.

Outro ponto importante a se destacar é a extrema importância das *caches* de nível 1, que obtiveram número de *misses* muito pequenos comparado ao total de transações efetuadas por estes módulos. Isso prova mais uma vez que quase a totalidade das transações de instruções e a grande maioria das transações sobre dados efetuadas pelos processadores conseguem ser atendidas tão logo pelas *caches* de nível 1, de acesso muito rápido, o que torna viável a execução de aplicações extremamente grandes.

Quanto à utilização de múltiplos processadores, tema central deste trabalho, foi notório o maior desempenho obtido (salvo exceções verificadas no capítulo 6) pelos processadores de quatro núcleos em relação aos de dois núcleos, havendo uma diminuição no número de ciclos e no tempo de execução dos modelos *dual-core* para os *quad-core*. Verificou-se, com o incremento de núcleos, um aumento escalar no desempenho das aplicações, onde se obteve menores números na análise das transações e *misses* de instruções e dados.

Assim, foi possível simular e realizar os devidos testes de desempenho nas arquiteturas trabalhadas. Com os resultados dos testes e a avaliação do comportamento dos diferentes processadores e de suas abordagens quanto às arquiteturas de memória *cache*, é possível afirmar que o incremento de núcleos no sistema faz com que o desempenho das aplicações executadas seja maior.

No contexto das memórias *cache*, é importante novamente deixar claro que os resultados apresentados acerca dos modelos Intel e AMD dizem respeito às suas hierarquias e características. Estes resultados não levam em conta otimizações implementadas nos próprios processadores, barramentos ou em outros componentes, que possam vir a influenciar no desempenho. Com o estudo realizado neste trabalho é possível concluir que, sob o ponto de vista das arquiteturas de *cache* da maneira como foram modeladas, a abordagem de *caches* privadas mostrou-se mais eficiente. Além disso, a utilização de módulos de *cache* com baixa latência é fundamental para o aumento do desempenho do sistema.

Existem alguns pontos identificados como possíveis trabalhos futuros, os quais podem enriquecer este estudo e torná-lo mais abrangente. Eles são os seguintes:

- Inserir novas arquiteturas ao conjunto estudado, preferencialmente com números diversos de processadores, como oito núcleos ou mais. Há famílias da AMD que possuem arquitetura de três núcleos, que podem fornecer resultados interessantes de comparação, principalmente pelo fato de terem número de núcleos diferente de uma potência de dois.

- Utilizar outros conjuntos de *benchmarks*, a fim de aumentar a gama de aplicativos de teste. Alguns conjuntos relevantes que podem ser utilizados são o SPLASH-2, o NAS e o SPEC OMP.
- Na utilização do PARSEC, a realização de testes com outros conjuntos de entradas para simulação (*simmedium* e *simlarge*). As entradas para simulação de maior tamanho possuem fluxos de instruções com maior paralelização, o que torna suas avaliações de grande relevância.
- Ainda com relação ao PARSEC, o aumento da diversidade no número de *threads* a serem executadas por aplicação pode demonstrar comportamentos diferentes devido à maior paralelização e ao modo que estas *threads* são escalonadas entre os núcleos do sistema avaliado.

REFERÊNCIAS

ACIIÇMEZ, O.; KOÇ, Ç. K.; SEIFERT, J. P. On the Power of Simple Branch Prediction Analysis. In: Proceedings of the 2nd ACM SYMPOSIUM ON INFORMATION, COMPUTER AND COMMUNICATIONS SECURITY, 2., 2007, Singapore. **Proceedings…** New York, USA: ACM, 2007. p. 312-320.

ALAMELDEEN, A. R.; WOOD, D. A. Variability in Architectural Simulations of Multi-threaded Workloads. In: Proceedings of the 9th ANNUAL INTERNATIONAL SYMPOSIUM ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE, 2003, Anaheim, USA. **Proceedings…** Washington, USA: IEEE Computer Society, 2003.

ALMASI, G. S.; GOTTLIEB, A. **Highly Parallel Computing**. 2nd ed. Redwook City, USA: Benjamin Cummings, 1994.

AMD INC. **AMD Hyper Transport Technology-Based System Architecture**. White paper. Sunnyvale, USA: Advanced Device Micros Inc., 2002.

AMD INC. **The AMD Opteron pProcessor for Servers and Workstations**. Data sheet. Sunnyvale, USA: Advanced Device Micros Inc., 2005.

AMD INC. **AMD Turion 64x2 Mobile Technology Dual-Core Processor Product Data Sheet**. Data sheet. Sunnyvale, USA: Advanced Device Micros Inc., 2006.

AMD INC. **ACP – The Truth About Power Consumption Starts Here**. White paper. Sunnyvale, USA: Advanced Device Micros Inc., 2007a.

AMD INC. **AMD64 Architecture Programmer's Manual.** Sunnyvale, USA: Advanced Device Micros Inc., 2007b. v.1.

AMD INC. **Quad-Core AMD Opteron Processor with Direct Connect Architecture**. Data sheet. Sunnyvale, USA: Advanced Device Micros Inc., 2007c.

AMD INC. **AMD Phenom Processor Product Data Sheet**. Data sheet. Sunnyvale, USA: Advanced Device Micros Inc., 2007d.

AMD INC. **AMD Turion 64 X2 Dual-Core Mobile Technology Competitive Comparison**. [S.l.]: Advanced Device Micros Inc., 2008a. Disponível em: <http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_13909_13911,00.html>. Acesso em: out 2008.

AMD INC. **Compreendendo os números dos modelos: Processador móvel AMD Turion X2 Ultra de Dois Núcleos**. [S.l.]: Advanced Device Micros Inc., 2008b. Disponível em: <http://www.amd.com/br-pt/Processors/ProductInformation/0,,30_118_12651_15667%5E15674,00.html>. Acesso em: out 2008.

AMD INC. **Principais recursos arquitetônicos do processador AMD Opteron da terceira geração**. [S.l.]: Advanced Device Micros Inc., 2008c. Disponível em: <http://www.amd.com/br-pt/Processors/ProductInformation/0,,30_118_8796_15224,00.html>. Acesso em: out 2008.

AMD INC. **Compreendendo os números dos modelos do processador AMD Opteron da Terceira Geração**. [S.l.]: Advanced Device Micros Inc., 2008d. Disponível em: <http://www.amd.com/br-pt/Processors/ProductInformation/0,,30_118_8796_15226,00.html>. Acesso em: out 2008.

AMD INC. **Principais recursos de arquitetura dos processadores AMD Phenom X4 de Quatro Núcleos**. [S.l.]: Advanced Device Micros Inc., 2008e. Disponível em: <http://www.amd.com/br-pt/Processors/ProductInformation/0,,30_118_15331_15332%5E15334,00.html>. Acesso em: out 2008.

AMD INC. **Comparação competitiva de processadores – AMD Phenom**. [S.l.]: Advanced Device Micros Inc., 2008f. Disponível em: <http://www.amd.com/br-pt/Processors/ProductInformation/0,,30_118_15331_15332%5E15346,00.html>. Acesso em: out 2008.

AMDAHL, G. M. Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities. In: AFIPS CONFERENCE, 30., 1967, Atlantic City, USA. Proceedings… [S.l.]: AFIPS, 1967.

ANDERSON, D. W.; SPARACIO, F. J.; TOMASULO, R. M. The System/360 Model 91: Machine Philosophy and Instruction Handling. **IBM Journal**, [S.l.], v.11, p. 8-24, 1967.

APPLE INC. **Macintosh 128K:** Technical Specifications. [S.l.], 2008. Disponível em: <http://support.apple.com/kb/SP186>. Acesso em: out 2008.

ASPINALL, D. Structures for Parallel Processing. **Computing & Control Engineering Journal**, [S.l.], v.1, n.1, p. 15-22, 1990.

ATKINS, M. Performance and the i860 Microprocessor. **IEEE Micro**, Santa Clara, USA, v.11, n.5, p. 24-27, 72-78, 1991.

BARNES, G. H. et al. The Illiac IV Computer. **IEEE Transactions on Computers**, [S.l.], v.C-17, n.8, p. 746-757, 1968.

BELL, C. G. Multis: A New Class of Multiprocessor Computers. **Science**, [S.l.], v.228, p. 462-467, 1985.

BIENIA, C. et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In: INTERNATIONAL CONFERENCE ON PARALLEL CONFERENCE ON PARALLEL ARCHITECTURES AND COMPILATION

TECHNIQUES, 17., 2008, Toronto, Canadá. **Proceedings…** [S.l]: IEEE Computer Society, 2008.

BIENIA, C.; KUMAR, S.; LI, K. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. In: IEEE INTERNATIONAL SYMPOSIUM ON WORKLOAD CHARACTERIZATION, 2008, Seattle, USA. **Proceedings…** [S.l.]: IEEE Computer Society, 2008. p. 47-56.

BLOCK, E. The Engineering Design of the STRETCH Computer. In: EASTERN JOINT COMPUTER CONFERENCE, 1959. **Proceedings…** [S.l.:s.n.], 1959.

CANONICAL LTD. **Ubuntu 6.06.2 LTS (Dapper Drake)**. [S.l.], 2008. Disponível em: <http://releases.ubuntu.com/6.06>.Acesso em: out 2008.

CONTESSA, A. An Approach to Fault Tolerance and Error Recovery in a Parallel Graph Reduction Machine: MaRS - A Case Study. **SIGARCH Computer Architure News**, New York, v.16, n.3, p. 25-32, 1988.

DAVIS, R. L. The Illiac IV Processing Element. **IEEE Transactions on Computers**, Washington, USA, v.C-18, n.9, p. 800-816, 1969.

DE ROSE, C. A. F.; NAVAUX, P. A. O. **Arquiteturas Paralelas**. Porto Alegre, Brasil: Sagra Luzzatto, 2003.

ECKERT, J. P. et al. Design of UNIVAC-LARC System 1. In: EASTERN JOINT COMPUTER CONFERENCE, 1959. **Proceedings…** [S.l.: s.n.], 1959. p. 60-65.

ECKERT-MAUCHLY COMPUTER CORP. **Preliminary Description of the UNIVAC**. [S.l.], 1956.

EL-REWINI, H.; ABD-EL-BARR, M. **Advanced Computer Architecture and Parallel Processing**. Hoboken, USA: John Wiley & Sons, 2005.

ENGBLOM, J.; EKBLOM, D. **Simics:** a Commercially Proven Full-Fystem Simulation Framework. European Space Programmes. Noordwijk, Netherlands: [s.n.], 2006.

FLYNN, M. J. Very High-Speed Computing Systems. **Proceedings of the IEEE**, [S.l.], v.54, n.12, p. 1901-1909, 1966.

FOX, A. et al. Cluster-Based Scalable Network Services. **SIGOPS Operational Systems Reviews**, New York, v.31, n.5, p. 78-91, 1997.

GOLDSTINE, H. H.; GOLDSTINE, A. The Electronic Numerical Integrator and Computer (ENIAC). **Mathematical Tables and Other Aids to Computation**. [S.l.: s.n.], 1946.

GREGORY, J.; MCREYNOLDS, R. The Solomon Computer. **IEEE Transactions on Eletronic Computers**, [S.l.], v.EC-12, n.6, p. 774-781, 1963.

HALAAS, A. et al. A Recursive MISD Architecture for Pattern Matching. **IEEE Transactions on very Large Scale Integration Systems**, Piscataway, v.12, n. 7, p. 727-734, 2004.

HELIN, J. Performance Analysis of the CM-2, a Massively Parallel SIMD Computer. In: INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, 6., 1992. **Proceedings** … New York, USA: ACM, 1992. p. 45-52.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture**: A Quantitative Approach. 4th ed. San Francisco, USA: Morgan Kaufmann, 2007.

HEWLETT-PACKARD COMPANY. **CACTI 5.3**. [S.l.], 2008. Disponível em: <http://quid.hpl.hp.com:9081/cacti>. Acesso em: out 2008.

HILLIS, W. D.; TUCKER, L. W. The CM-5 Connection Machine: A Scalable Supercomputer. **Communications of the ACM**, New York, v.36, n.11, p. 31-40, 1993.

HINTON, G. et al. The Microarchitecture of the Pentium 4 Processor. **Intel Technology Journal**, [S.l.], v.1, p. 1-12, 2001.

HWANG, K. **Advanced Computer Architecture:** Paralelism, Scalability, Programmability. New York, USA: McGraw-Hill, 1993.

IMAGE **Amdahl's Law**. [S.l.]: Wikipedia, 2008. Disponível em: <http://en.wikipedia.org/wiki/Image:AmdahlsLaw.svg>. Acesso em: out 2008.

INTEL CORP. **Intel Core 2 Quad Processor**. Product Brief. [S.l.], 2007a.

INTEL CORP. **Intel Core 2 Extreme Quad-Core Processor QX6000 Sequence and Intel Core 2 Quad Processor Q6000 Sequence**. Data sheet. [S.l.], 2007b.

INTEL CORP. **Intel Core 2 Duo Processors and Intel Core 2 Extreme Processors for Platforms Based on Mobile Intel 965 Express Chipset Family**. Data sheet. [S.l.], 2008a.

INTEL CORP. **Intel Core 2 Duo Processor and Intel Core 2 Extreme Processor on 45-nm Process for Platforms Base don Mobile Intel 965 Express Chipset Family**. Data sheet. [S.l.], 2008b.

INTEL CORP. **Intel Core 2 Extreme Processor QX9000 Series and Intel Core 2 Quad Processor Q9000 and Q8000 Series**. Data sheet. [S.l.], 2008c.

IVANOV, L.; NUNNA, R. Modeling and Verification of Cache Coherence Protocols. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2001, Sydney. **Proceedings…** [S.l.]: IEEE Computer Society, 2001. v.5, p. 129-132.

KAMBHATLA, S.; INOUYE, J.; WALPOLE, J. **Benchmarking Parallel Machines Via a Software Architecture**. [S.l.: s.n.], 1990.

LENOVO GROUP LTD. **Personal Systems Reference Intel PC Processors**. Data sheet. Morrisville, USA, 2008.

MAGNUSSON, P. S. et al. Simics: A Full System Simulation Platform. **Computer**, Washington, USA, v.35, n.2, p. 50-58, 2002.

MAGNUSSON, J. P.; WERNER, B. Efficient Memory Simulation in SimICS. In: ANNUAL SIMULATION SYMPOSIUM, 28., 1995. **Proceedings…** Washington, USA: IEEE Computer Society, 1995. p. 62-73.

MAK, P. et al. Shared-Cache Clusters in a System with a Fully Shared Memory. **IBM Journal of Research and Development**, Riverton, USA, v.41, n.4-5, p. 429-448, 1997.

MAKAI, M. C. **A Genetic Algorithm Framework for Benchmarking Parallel Computer Architectures**. [S.l.: s.n.], 2006.

MOORE, C. R. The PowerPC 601 Microprocessor. In: Compcon Spring, 1993. **Proceedings…** San Francisco, USA: IEEE Computer Society, 1993. p. 109-116

PANDIAN, A. et al. Simulation and Performance Evaluation of Parallel Architecture Based on i860 Nodes: SapePar-i860. In: IEEE REGIONR 10's ANNUAL INTERNATIONAL CONFERENCE, 9., 1994. **Proceedings…** [S.l.]: IEEE Computer Society, 1994. v.2, p. 682-686.

PRINCETON UNIVERSITY. **PARSEC Overview**: Comparison. [S.l.], 2008. Disponível em: <http://parsec.cs.princeton.edu/overview.htm#Comparison>. Acesso em: out 2008.

ROBERTS, J. B. G. et al. Benchmarking Parallel Architectures. In: INTERNATIONAL SPECIALIST SEMINAR ON THE DESIGN AND APPLICATION OF PARALLEL DIGITAL PROCESSORS, 1998. **Proceedings…** Lisboa, Portugal: [s.n.], 1988. p. 134-138.

SMITH, B. J. A Pipelined, Shared Resource MIMD Computer. In: IEEE INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 1978. **Proceedings…** USA: IEEE Computer Society, 1978. p. 6-8.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 5. ed. São Paulo, Brasil: Prentice Hall, 2002.

THOZIYOOR, S. et al. A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies. In: IEEEINTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURES, 2008, Beijing, China . **Proceedings…** [S.l.]: IEEE Computer Society, 2008. v.1, p. 51-62.

TOMASULO, R. M. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. **IBM Journal**, [S.l.], v.11, p. 25-33, 1967.

VIRTUTECH AB **Simics User Guide for Windows**. Stockholm, Sweden, 2008a.

VIRTUTECH AB **Solutions by Virtual Platforms**. [S.l.], 2008. Disponível em: <http://www.virtutech.com/solutions/virtual_platform>. Acesso em: out 2008.

WECHSLER, O. **Inside Intel Core Microarchitecture: Setting New Standards for Energy-Efficient Performance**. White Paper. [S.l.]: Intel Corporation, 2006.

WOO, S. C. et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 22., 1995. **Proceedings…** New York, USA: ACM, 1995. p. 24-36.

APÊNDICE A :
RESULTADOS DA FAMÍLIA INTEL CORE 2 DUO

A.1 Modelo T7800

Tabela A.1: Resultados da *cache* L2 – modelo Intel T7800

Cache nível 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	269066	5126419	4825090	296491	265248	282334	8466709
Instr. fetch misses	8387	15744	47543	14713	8785	8126	28033
Instr. fetch hit ratio	96.88%	97.83%	99.01%	95.04%	96.69%	97.12%	99.67%
Data read trans.	1746822	11703403	32623163	4552614	21599829	2912342	8515945
Data read misses	24679	207990	4994617	401535	30912	24349	423475
Data read hit ratio	98.59%	98.22%	86.69%	91.18%	99.86%	99.16%	95.03%
Data write trans.	155000246	505774986	887591202	612335618	232179945	252930096	668849706
Data write misses	32915	215906	682849	351010	35344	32006	417032
Data write hit ratio	99.98%	99.96%	99.92%	99.94%	99.98%	99.99%	99.94%
Copy back trans.	43547	234316	4404635	487021	47855	43247	467447

Tabela A.2: Resultados da *cache* L1 de instruções da CPU 0 – modelo Intel T7800

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	390161070	1459424890	2238087089	1591823424	1028838405	780043258	1695206351
Instr. fetch misses	138734	1330478	2576039	161842	149934	149810	4431332
Instr. fetch hit rate	99.96%	99.91%	99.88%	99.99%	99.98%	99.98%	99.74%

Tabela A.3: Resultados da *cache* L1 de dados da CPU 0 – modelo Intel T7800

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	151285485	599353935	945082858	624489902	452838060	331949140	644409251
Data read misses	918871	5898625	17396822	2334133	11893493	1594977	4508142
Data read hit ratio	99.39%	99.02%	98.16%	99.63%	97.37%	99.52%	99.30%
Data write trans.	74907762	253045223	444287677	307522013	112087994	125776845	335741627
Data write misses	3682615	22133790	11751028	7840308	5958129	12183550	21550605
Data write hit ratio	95.08%	91.25%	97.36%	97.45%	94.68%	90.31%	93.58%

Tabela A.4: Resultados da *cache* L1 de instruções da CPU 1 – modelo Intel T7800

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	407666418	1477357117	2282154582	1622700849	1010547445	778251608	1715421968
Instr. fetch misses	126727	1343060	2249051	134649	115414	132924	4035377
Instr. fetch hit ratio	99.97%	99.91%	99.90%	99.99%	99.99%	99.95%	99.76%

Tabela A.5: Resultados da *cache* L1 de dados da CPU 1 – modelo Intel T7800

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	158008888	599208871	956569295	631639735	441042125	329871226	651730533
Data read misses	831135	5803398	15226341	2218481	9706336	1317365	4007803
Data read hit ratio	99.47%	99.03%	98.41%	99.65%	97.80%	99.60%	99.39%
Data write trans.	77682258	252731699	443303525	304813605	120091951	127153251	333108079
Data write misses	3881366	21786452	12168061	7428762	5130292	11001743	20652124
Data write hit ratio	95.00%	91.38%	97.26%	97.56%	95.73%	91.35%	93.80%

Tabela A.6: Ciclos e tempo de execução dos *benchmarks* – modelo Intel T7800

Tempo de Execução – Intel Core 2 Duo T7800							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	633720000	6116760376	11457063885	7252522871	3278166643	2708142068	7145924474
Time exec. (s)	0,244	2,353	4,406	2,789	1,261	1,041	2,748

A.2 Modelo T9500

Tabela A.7: Resultados da *cache* L2 – modelo Intel T9500

Cache nível 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	445066	5386607	4852894	288261	638329	837529	12690542
Instr. fetch misses	11505	18768	39828	10939	5402	11084	32734
Instr. fetch hit ratio	97.41%	99.65%	99.18%	96.21%	99.15%	98.68%	99.74%
Data read trans.	3073474	14832256	39314921	4469562	21720197	5001912	15629322
Data read misses	19848	405163	5624830	196220	78213	37606	657279
Data read hit ratio	99.35%	97.27%	85.69%	95.61%	99.64%	99.25%	95.79%
Data write trans.	158368379	517783117	921083492	609801809	234689983	255976198	680418120
Data write misses	237347	699294	1289798	265503	159375	109694	956058
Data write hit ratio	99.85%	99.86%	99.86%	99.96%	99.93%	99.96%	99.86%
Copy back trans.	210688	818321	3965876	326426	188152	121020	1103819

Tabela A.8: Resultados da *cache* L1 de instruções da CPU 0 – modelo Intel T9500

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	417479147	1360224000	2205823928	1596954870	225456705	759259990	1556844395
Instr. fetch misses	227382	2343804	736835	150931	185350	501659	6000705
Instr. fetch hit ratio	99.95%	99.83%	99.97%	99.99%	99.92%	99.93%	99.61%

Tabela A.9: Resultados da *cache* L1 de dados da CPU 0 – modelo Intel T9500

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	150130090	556944066	915189469	624494128	444840340	316958718	588610639
Data read misses	1774687	7123422	21075942	2322366	11584370	2662757	7455199
Data read hit ratio	98.82%	98.72%	97.70%	99.63%	97.40%	99.16%	98.73%
Data write trans.	73241540	257321176	451714148	305845389	112871664	130662361	338326901
Data write misses	5611208	29227506	18133462	7391534	6569267	15852019	30292816
Data write hit ratio	92.34%	88.64%	95.99%	97.58%	94.18%	87.87%	91.05%

Tabela A.10: Resultados da *cache* L1 de instruções da CPU 1 – modelo Intel T9500

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	405689055	1294332766	1946570374	1618381250	178874851	795025447	1445744480
Instr. fetch misses	217684	3042803	4116059	137330	452979	335870	6689837
Instr. fetch hit ratio	99.95%	99.76%	99.79%	99.99%	99.75%	99.96%	99.54%

Tabela A.11: Resultados da *cache* L1 de dados da CPU 1 – modelo Intel T9500

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	145334663	553143919	879502653	629823410	431199925	329523271	576044767
Data read misses	1298787	7708834	18238979	2147196	10135827	2339155	8174123
Data read hit ratio	99.11%	98.61%	97.93%	99.66%	97.65%	99.29%	98.58%
Data write trans.	75766226	260461941	469369344	303956420	121818319	125313837	342091219
Data write misses	6408038	28290274	31105895	7991354	7627748	14596892	35389637
Data write hit ratio	91.54%	89.14%	93.37%	97.37%	93.74%	88.35%	89.65%

Tabela A.12: Ciclos e tempo de execução dos *benchmarks* – modelo Intel T9500

Tempo de Execução – Intel Core 2 Duo T7800							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	583423365	5787180894	10495981822	6766003086	3153525460	2455522982	7105740493
Time exec. (s)	0,224	2,226	4,037	2,602	1,213	0,944	2,733

APÊNDICE B :
RESULTADOS DA FAMÍLIA AMD TURION 64X2

B.1 Modelo TL-68

Tabela B.1: Resultados da *cache* L2 da CPU 0 – modelo AMD TL-68

Cache nível 2 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	209189	735323	454493	98207	95331	211121	2599618
Instr. fetch misses	72923	89959	89156	52771	50496	124584	165710
Instr. fetch hit ratio	65.14%	87.77%	80.38%	46.27%	47.03%	40.99%	93.63%
Data read trans.	1083384	5655506	14409649	1790149	10827096	2342807	3731054
Data read misses	300848	935021	6590974	834362	2860080	507046	1112513
Data read hit ratio	72.23%	83.47%	46.74%	53.39%	73.58%	78.36%	70.18%
Data write trans.	75164110	257229099	447916123	305555063	116905026	138787697	337186007
Data write misses	205219	641982	671825	418403	245791	408056	484135
Data write hit ratio	99.73%	99.75%	99.85%	99.86%	99.79%	99.71%	99.86%
Copy back trans.	323178	1013902	3976057	740497	969707	598718	726510
MESI M. to S. trans.	52638	278894	459734	173174	695941	136371	196151
MESI Inv trans.	50010	308193	384088	188482	737663	245567	288516

Tabela B.2: Resultados da *cache* L1 de instruções da CPU 0 – modelo AMD TL-68

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	368340297	1373933780	2280764739	1618300289	1033668048	862517260	1703212513
Instr. fetch misses	209189	735323	454493	98207	95331	211121	2599618
Instr. fetch hit ratio	99.94%	99.95%	99.98%	99.99%	99.99%	99.98%	99.85%

Tabela B.3: Resultados da *cache* L1 de dados da CPU 0 – modelo AMD TL-68

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	139117501	560314815	964953414	632273138	453807071	354131510	644738193
Data read misses	1083384	5655506	14409649	1790149	10827096	2342807	3731054
Data read hit ratio	99.22%	98.99%	98.51%	99.72%	97.61%	99.34%	99.42%
Data write trans.	75164110	257229099	447916123	305555063	116905026	138787697	337186007
Data write misses	5563855	30389499	10757828	7472685	5541391	14998122	19833874
Data write hit ratio	92.60%	88.19%	97.60%	97.55%	95.26%	89.19%	94.12%

Tabela B.4: Resultados da *cache* L2 da CPU 1 – modelo AMD TL-68

Cache nível 2 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	215125	1032228	416829	94182	80248	323639	2484277
Instr. fetch misses	70381	62782	66202	52497	47079	113048	164145
Instr. fetch hit ratio	67.28%	93.92%	84.12%	44.26%	41.33%	65.07%	93.39%
Data read trans.	1082719	6610873	15261499	1808132	9970088	2028738	3775521
Data read misses	331781	835914	6571602	885679	2685999	494243	1184803
Data read hit ratio	69.36%	87.36%	44.06%	51.02%	73.06%	75.64%	68.62%
Data write trans.	75209367	265669994	445641841	307718394	119815201	138824994	336549703
Data write misses	236340	710589	680472	440805	215850	380546	512611
Data write hit ratio	99.69%	99.73%	99.85%	99.86%	99.82%	99.73%	99.85%
Copy back trans.	343503	1059956	4567810	728089	899009	544680	711336
MESI M. to S. trans.	42164	262882	389888	130750	654655	119048	163293
MESI Inv. Trans.	64217	360395	467067	221902	784916	264599	335091

Tabela B.5: Resultados da *cache* L1 de instruções da CPU 1 – modelo AMD TL-68

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	360843444	1236708943	2294735694	1617781203	1038472778	875346181	1721627298
Instr. fetch misses	215125	1032228	416829	94182	83248	323639	2484277
Instr. fetch hit ratio	99.94%	99.92%	99.98%	99.99%	99.98%	99.96%	99.86%

Tabela B.6: Resultados da *cache* L1 de dados da CPU 1 – modelo AMD TL-68

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	138315375	544157227	953436815	626149418	451461381	355555480	649109732
Data read misses	1082719	6610873	15261499	1808132	9970088	2028738	3775521
Data read hit ratio	99.22%	98.79%	98.40%	99.71%	97.79%	99.43%	99.42%
Data write trans.	75209367	265669994	445641841	307718394	119815201	138824994	336549703
Data write misses	6369730	24800024	11731381	7920278	4743858	13657002	21549300
Data write hit ratio	91.53%	90.67%	97.37%	97.43%	96.04%	90.16%	93.60%

Tabela B.7: Ciclos e tempo de execução dos *benchmarks* – modelo AMD TL-68

Tempo de Execução – AMD Turion 64x2 TL-68							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	524362769	5432112087	9766402232	6216988623	2873183905	2157327376	6726580882
Time exec. (s)	0,218	2,263	4,069	2,590	1,197	0,899	2,803

B.2 Modelo ZM-86

Tabela B.8: Resultados da *cache* L2 da CPU 0 – modelo AMD ZM-86

Cache nível 2 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	63447	495717	667999	85420	94259	67574	2688700
Instr. fetch misses	22443	101831	82576	37361	43689	27649	88541
Instr. fetch hit ratio	64.63%	79.46%	87.64%	56.26%	53.65%	59.08%	96.71%
Data read trans.	464924	5574826	15408535	1915913	10342459	756056	3883066
Data read misses	102129	702736	6983148	749533	905953	114151	836965
Data read hit ratio	78.03%	87.39%	54.68%	60.88%	91.24%	84.90%	78.45%
Data write trans.	72807720	256828405	457349884	305364639	107913534	117833845	327760811
Data write misses	58699	509444	811691	394781	177597	76721	394173
Data write hit ratio	99.92%	99.80%	99.82%	99.87%	99.84%	99.93%	99.88%
Copy back trans.	118757	806528	4232843	732633	862864	150515	619832
MESI M. to S. trans	56601	292600	731539	201897	668818	68322	223112
MESI Inv. trans	61546	344863	583576	243067	716066	69378	283671

Tabela B.9: Resultados da *cache* L1 de instruções da CPU 0 – modelo AMD ZM-86

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	353116480	1456848555	2188511834	1593110127	939449807	701161121	1661170993
Instr. fetch misses	63447	495717	667999	85420	94259	67574	2688700
Instr. fetch hit ratio	99.98%	99.97%	99.97%	99.99%	99.99%	99.99%	99.84%

Tabela B.10: Resultados da *cache* L1 de dados da CPU 0 – modelo AMD ZM-86

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	130212404	581939786	934275373	626075496	403494393	292740547	626695515
Data read misses	464924	5574826	15408535	1915913	10342459	756056	3883066
Data read hit ratio	99.64%	99.04%	98.35%	99.69%	97.44%	99.74%	99.38%
Data write trans.	72807720	256828405	457349884	305364639	107913534	117833845	327760811
Data write misses	2793359	25468748	17197720	7559985	4863937	8629189	19751083
Data write hit ratio	96.16%	90.08%	96.24%	97.52%	95.49%	92.68%	93.97%

Tabela B.11: Resultados da *cache* L2 da CPU 1 – modelo AMD ZM-86

Cache nível 2 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	73409	741716	1466952	89893	105048	75119	2494597
Instr. fetch misses	26619	98574	61245	36017	46633	24941	76893
Instr. fetch hit ratio	63.74%	86.71%	95.83%	59.93%	55.61%	66.80%	96.92%
Data read trans.	662380	5807074	16530982	1861833	10507142	951806	3697059
Data read misses	130953	697172	8359717	755436	1027194	134894	883484
Data read hit ratio	80.23%	87.99%	49.43%	59.43%	90.22%	85.83%	76.10%
Data write trans.	71625829	260926185	462890597	304181101	107009566	116552662	325211063
Data write misses	65963	514547	954142	383613	211574	73434	448273
Data write hit ratio	99.91%	99.80%	99.79%	99.87%	99.80%	99.94%	99.86%
Copy back trans.	118440	781806	4641745	668329	828478	127723	619083
MESI M. to S. trans.	50121	276758	676703	180066	620939	51557	169499
MESI Inv. Trans.	68366	386130	663393	290997	791086	96165	332856

Tabela B.12: Resultados da *cache* L1 de instruções da CPU 1 – modelo AMD ZM-86

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	371551461	1421184623	2052866606	1623084616	963917209	721253570	1681933260
Instr. fetch misses	73409	741716	1466952	89893	109048	75119	2494597
Instr. fetch hit ratio	99.98%	99.95%	99.93%	99.99%	99.98%	99.99%	99.85%

Tabela B.13: Resultados da *cache* L1 de dados da CPU 1 – modelo AMD ZM-86

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	133740258	569716362	904660552	626328966	406192920	296296565	634851223
Data read misses	662380	5807074	16530982	1861833	10507142	951806	3697059
Data read hit ratio	99.50%	98.98%	98.17%	99.70%	97.41%	99.68%	99.42%
Data write trans.	71625829	260926185	462890597	304181101	107009566	116552662	325211063
Data write misses	3629173	24022530	20151805	7402343	6297848	8973754	20898994
Data write hit ratio	94.93%	90.79%	95.65%	97.57%	94.11%	92.30%	93.57%

Tabela B.14: Ciclos e tempo de execução dos *benchmarks* – modelo AMD ZM-86

Tempo de Execução – AMD Turion 64x2 ZM-86							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	483529230	5378826634	8335420202	6000561194	2660142178	2269025327	6683806933
Time exec. (s)	0,201	2,241	3,473	2,500	1,108	0,945	2,785

APÊNDICE C :
RESULTADOS DA FAMÍLIA INTEL CORE 2 QUAD

C.1 Modelo Q6700

Tabela C.1: Resultados da *cache* L2 das CPUs 0 e 1 – modelo Intel Q6700

Cache nível 2 – CPUs 0 e 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	75692	2375181	2876920	97480	98793	89175	5587323
Instr. fetch misses	6449	24503	29564	11631	11275	8163	38556
Instr. fetch hit ratio	91.48%	98.97%	98.97%	88.07%	88.59%	90.85%	99.31%
Data read trans.	681834	7466705	16967184	2068915	10871202	863330	7464018
Data read misses	82481	504287	4337805	516952	615580	114959	862114
Data read hit ratio	87.90%	93.25%	74.43%	75.01%	94.34%	86.68%	88.45%
Data write trans.	126908072	385979094	890118960	428738157	179811477	170541424	588926237
Data write misses	36539	431315	844545	284129	119514	58731	712180
Data write hit ratio	99.97%	99.89%	99.91%	99.93%	99.93%	99.97%	99.88%
Copy back trans.	136107	745056	3666670	634271	734731	181735	1169091
MESI M. to S. trans.	85337	321727	770918	322527	605310	109290	425467
MESI Inv. Trans.	97840	405461	380111	323116	657317	140707	439054

Tabela C.2: Resultados da *cache* L1 de instruções da CPU 0 – modelo Intel Q6700

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	296635444	877643606	1647076916	1038915512	614556107	460654064	1244363092
Instr. fetch misses	24752	1512537	2776747	54651	54757	54571	3153089
Instr. fetch hit ratio	99.99%	99.83%	99.83%	99.99%	99.99%	99.99%	99.75%

Tabela C.3: Resultados da *cache* L1 de dados da CPU 0 – modelo Intel Q6700

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	113446403	381439486	747427448	404728532	261773224	191832773	510006086
Data read misses	317962	4195141	11699245	908035	5427774	408730	4546511
Data read hit ratio	99.72%	98.90%	98.43%	99.78%	97.93%	99.79%	99.11%
Data write trans.	63614341	196496782	456076160	216072934	90828035	85830125	294674195
Data write misses	1105561	12833889	24049362	2685807	2359806	3998284	19307030
Data write hit ratio	98.26%	93.47%	94.73%	98.76%	97.40%	95.34%	93.45%

Tabela C.4: Resultados da *cache* L1 de instruções da CPU 1 – modelo Intel Q6700

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	303555233	990144230	2039586246	1066637445	635926804	471562701	1269647120
Instr. fetch misses	50940	862644	100173	42829	44036	34604	2434234
Instr. fetch hit ratio	99.98%	99.91%	99.99%	99.99%	99.99%	99.99%	99.81%

Tabela C.5: Resultados da *cache* L1 de dados da CPU 1 – modelo Intel Q6700

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	113931363	391976239	813293222	427384202	265568095	193446175	505677996
Data read misses	363872	3271564	5267939	1160880	5443428	454600	2917507
Data read hit ratio	99.68%	99.17%	99.35%	99.73%	97.95%	99.76%	99.42%
Data write trans.	63293731	189482312	434042800	212665223	88983442	84711299	294252042
Data write misses	2084822	9192508	6259631	6108937	2740778	4521475	15187396
Data write hit ratio	96.71%	95.15%	98.56%	97.13%	96.92%	94.66%	94.84%

Tabela C.6: Resultados da *cache* L2 das CPUs 2 e 3 – modelo Intel Q6700

Cache nível 2 – CPUs 2 e 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	100501	1804870	206678	97998	167344	97529	4558363
Instr. fetch misses	6320	24034	32988	7627	11378	7813	30895
Instr. fetch hit ratio	93.71%	98.67%	84.04%	92.22%	93.20%	91.99%	99.32%
Data read trans.	740599	6961195	17315650	2018394	11300806	1010042	5331831
Data read misses	99882	504282	4598526	466051	646905	123544	806617
Data read hit ratio	86.51%	92.76%	73.44%	76.91%	94.28%	87.77%	84.87%
Data write trans.	128127240	378958895	870547971	422849230	178398463	170241104	592768705
Data write misses	42809	299388	624164	127009	130206	70616	469294
Data write hit ratio	99.97%	99.92%	99.93%	99.97%	99.93%	99.96%	99.92%
Copy back trans.	17960	600667	2894845	426301	706435	173584	840812
MESI M. to S. trans.	75435	326782	684446	300123	574835	99707	380883
MESI Inv. Trans.	101784	422046	506623	351617	674110	148084	506945

Tabela C.7: Resultados da *cache* L1 de instruções da CPU 2 – modelo Intel Q6700

Cache de Instruções nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	304477427	1002827220	1994365931	1080646985	639967184	476551188	1256828390
Instr. fetch misses	64143	909394	91169	56258	109870	62999	2186638
Instr. fetch hit ratio	99.98%	99.91%	99.99%	99.99%	99.98%	99.99%	99.83%

Tabela C.8: Resultados da *cache* L1 de dados da CPU 2 – modelo Intel Q6700

Cache de Dados nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	113974931	396849296	786751498	422625532	265132289	194113929	502893669
Data read misses	425393	3512700	3118427	1057354	5668013	622453	2602264
Data read hit ratio	99.63%	99.11%	99.60%	99.75%	97.86%	99.68%	99.48%
Data write trans.	63516495	188924724	440139967	211235395	89973061	84807614	294131096
Data write misses	1987746	14202364	4258179	3428966	3683093	5145278	11972417
Data write hit ratio	96.87%	92.48%	99.03%	98.38%	95.91%	93.93%	95.93%

Tabela C.9: Resultados da *cache* L1 de instruções da CPU 3 – modelo Intel Q6700

Cache de Instruções nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	294393930	1007661170	2029982465	1084695252	636116398	467881994	1290416163
Instr. fetch misses	36358	895476	115509	41740	57474	34530	2371725
Instr. fetch hit ratio	99.99%	99.91%	99.99%	99.99%	99.99%	99.99%	99.82%

Tabela C.10: Resultados da *cache* L1 de dados da CPU 3 – modelo Intel Q6700

Cache de Dados nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	111159859	395732953	816264080	423027205	267108621	192001350	503024835
Data read misses	315206	3448495	14197223	961040	5632793	387589	2729567
Data read hit ratio	99.72%	99.13%	98.26%	99.77%	97.89%	99.80%	99.46%
Data write trans.	64610745	190034171	430408004	211613835	88425402	85433490	298637609
Data write misses	1558918	18732770	13344964	2702144	3381634	4182972	13521244
Data write hit ratio	97.59%	90.14%	96.90%	98.72%	96.18%	95.10%	95.47%

Tabela C.11: Ciclos e tempo de execução dos *benchmarks* – modelo Intel Q6700

Tempo de Execução – Intel Core 2 Quad Q6700							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	369500958	5229430548	13609942189	5771686642	2317140913	1805483669	825830187
Time exec. (s)	0,139	1,966	5,116518797	2,170	0,871	0,679	3,105

C.2 Modelo Q9650

Tabela C.12: Resultados da *cache* L2 das CPUs 0 e 1 – modelo Intel Q9650

Cache nível 2 – CPUs 0 e 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	87286	337533	144154	80054	46634	63684	3672419
Instr. fetch misses	2120	5582	13426	7730	5735	3300	14817
Instr. fetch hit ratio	97.57%	98.35%	90.69%	90.34%	87.70%	94.82%	99.60%
Data read trans.	659502	5447710	19856851	1649502	10461507	716657	3715340
Data read misses	72270	344094	3239773	238259	541395	86587	406668
Data read hit ratio	89.04%	93.68%	83.68%	85.56%	94.82%	87.92%	89.05%
Data write trans.	127988190	372777930	876225355	389299578	162266739	171074191	570265149
Data write misses	44608	191263	594835	228852	69485	57563	264855
Data write hit ratio	99.97%	99.95%	99.93%	99.94%	99.96%	99.97%	99.95%
Copy back trans.	118527	447321	2496827	403156	616750	151427	520195
MESI M. to S. trans.	78268	280934	584239	214496	547316	93811	229833
MESI Inv. Trans.	94586	388889	282165	189683	567681	126414	237726

Tabela C.13: Resultados da *cache* L1 de instruções da CPU 0 – modelo Intel Q9650

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	296796840	542083322	2005140480	935941027	571700191	456893622	1300521296
Instr. fetch misses	24891	175482	114239	37778	15179	21105	1962229
Instr. fetch hit ratio	99.99%	99.97%	99.99%	99.99%	99.99%	99.99%	99.85%

Tabela C.14: Resultados da *cache* L1 de dados da CPU 0 – modelo Intel Q9650

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	113024170	376694385	823884859	365773921	244348170	190622283	504898537
Data read misses	288915	2745270	17515742	812043	5221395	339298	2230970
Data read hit ratio	99.74%	99.27%	97.87%	99.78%	97.86%	99.82%	99.56%
Data write trans.	63524620	186876833	432299161	195669085	81412530	85425375	285373029
Data write misses	1029699	7574990	14660210	5256630	1407346	3661510	11032394
Data write hit ratio	98.38%	95.95%	96.61%	97.31%	98.27%	95.71%	96.13%

Tabela C.15: Resultados da *cache* L1 de instruções da CPU 1 – modelo Intel Q9650

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	303102328	558001993	1990181341	966837420	588718326	466660343	1285167830
Instr. fetch misses	63382	162051	29915	42276	31455	42579	1710190
Instr. fetch hit ratio	99.98%	99.97%	99.99%	99.99%	99.99%	99.99%	99.87%

Tabela C.16: Resultados da *cache* L1 de dados da CPU 1 – modelo Intel Q9650

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	111919329	375904986	769571588	364391253	245929875	190348991	493085436
Data read misses	370587	2702440	2341109	837459	5240112	377359	1484370
Data read hit ratio	99.67%	99.28%	99.70%	99.77%	97.87%	99.80%	99.70%
Data write trans.	64463570	185901097	443926194	193630493	80854209	85648816	284892120
Data write misses	1928875	6813329	1367282	2847280	2247099	4357287	9034142
Data write hit ratio	97.01%	96.33%	99.69%	98.53%	97.22%	94.91%	96.83%

Tabela C.17: Resultados da *cache* L2 das CPUs 2 e 3 – modelo Intel Q9650

Cache nível 2 – CPUs 2 e 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	122679	399575	95723	105842	124087	106346	3884714
Instr. fetch misses	6276	5838	6530	7720	6802	3207	10784
Instr. fetch hit ratio	94.88%	98.54%	93.18%	92.71%	94.52%	96.98%	99.72%
Data read trans.	836396	5874160	8935612	1917949	10733646	1039973	3438916
Data read misses	93865	356830	2347612	297161	570372	102653	358434
Data read hit ratio	88.78%	93.93%	73.73%	84.51%	94.69%	90.13%	89.58%
Data write trans.	129820008	369209559	884925521	389871338	159711262	168180670	583007551
Data write misses	51413	172842	74108	118767	78829	55737	200349
Data write hit ratio	99.96%	99.95%	99.99%	99.97%	99.95%	99.97%	99.97%
Copy back trans.	125399	427624	1538301	298658	597729	134564	406815
MESI M. to S. trans.	67514	276968	473342	161157	525517	78840	274225
MESI Inv. trans.	112506	324526	353574	219959	596133	135894	260206

Tabela C.18: Resultados da *cache* L1 de instruções da CPU 2 – modelo Intel Q9650

Cache de Instruções nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	306672351	564636047	1993683564	970318462	597154001	481257575	1289439038
Instr. fetch misses	60496	224878	47251	55781	70558	75588	1635681
Instr. fetch hit ratio	99.98%	99.96%	99.99%	99.99%	99.99%	99.98%	99.87%

Tabela C.19: Resultados da *cache* L1 de dados da CPU 2 – modelo Intel Q9650

Cache de Dados nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	114573898	377863387	768096603	364068703	249712517	195216904	493363376
Data read misses	468954	2960167	6529961	997878	5418256	598837	1457335
Data read hit ratio	99.59%	99.22%	99.15%	99.73%	97.83%	99.69%	99.70%
Data write trans.	62920580	186586497	440789635	193909499	78990248	83403337	285037341
Data write misses	2669934	13113127	3129198	3304880	3238626	5135455	8525328
Data write hit ratio	95.76%	92.97%	99.29%	98.30%	95.90%	93.84%	97.01%

Tabela C.20: Resultados da *cache* L1 de instruções da CPU 3 – modelo Intel Q9650

Cache de Instruções nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	296806903	563238607	1993513552	950125325	586973665	468819770	1344930640
Instr. fetch misses	25014	174697	48472	50061	53529	30758	2249033
Instr. fetch hit ratio	99.99%	99.97%	99.99%	99.99%	99.99%	99.99%	99.83%

Tabela C.21: Resultados da *cache* L1 de dados da CPU 3 – modelo Intel Q9650

Cache de Dados nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	112280017	389454717	768966877	363670866	246217519	191828752	499780601
Data read misses	257204	2913993	2405651	920071	5315390	441136	1981581
Data read hit ratio	99.77%	99.25%	99.69%	99.75%	97.84%	99.77%	99.60%
Data write trans.	63756532	182623062	444135886	195961839	80721014	84777333	297970210
Data write misses	1089108	14914478	1826853	2909877	3152705	4435230	8968090
Data write hit ratio	98.29%	91.83%	99.59%	98.52%	96.09%	94.77%	96.99%

Tabela C.22: Ciclos e tempo de execução dos *benchmarks* – modelo Intel Q9650

Tempo de Execução – Intel Core 2 Quad Q9650							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	336863384	4535823856	12268582092	4684587872	1890088606	1419083994	7328221472
Time exec. (s)	0,112	1,512	4,089	1,561	0,630	0,473	2,443

APÊNDICE D :
RESULTADOS DA FAMÍLIA AMD OPTERON X4

D.1 Modelo 8360SE

Tabela D.1: Resultados da *cache* L3 – modelo AMD 8360SE

Cache nível 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	81242	246106	190053	88691	69012	70902	517076
Instr. fetch misses	18170	86750	113973	55540	15396	15800	220517
Instr. fetch hit ratio	77.63%	64.75%	40.03%	37.38%	77.69%	77.72%	57.35%
Data read trans.	508669	2546508	20590800	3225558	582880	487186	3700726
Data read misses	131273	971167	14856496	2415362	147597	133818	2471728
Data read hit ratio	74.19%	61.86%	27.85%	25.12%	74.68%	72.53%	33.21%
Data write trans.	270363	1278248	8931442	1916149	287494	270854	1700804
Data write misses	43818	197094	2069487	783585	50909	41974	397756
Data write hit ratio	83.79%	84.58%	76.83%	59.11%	82.29%	84.50%	76.61%
Copy back trans.	118278	676783	7246420	1622603	135019	117822	1295535

Tabela D.2: Resultados da *cache* L2 da CPU 0 – modelo AMD 8360SE

Cache nível 2 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	17300	461639	2333308	706054	14107	21449	2101286
Instr. fetch misses	7441	80871	528027	77736	8990	12568	171960
Instr. fetch hit ratio	56.99%	82.48%	77.37%	88.99%	36.27%	41.41%	91.82%
Data read trans.	153571	2858320	6587503	2695016	4740584	339337	3262708
Data read misses	35668	422068	2203520	571950	57555	39952	847661
Data read hit ratio	76.77%	85.23%	66.55%	78.78%	98.79%	88.23%	74.02%
Data write trans.	63343552	193327073	472105098	217280799	78556901	84300008	288476891
Data write misses	19977	311580	734244	608020	21810	31012	395286
Data write hit ratio	99.97%	99.84%	99.84%	99.72%	99.97%	99.96%	99.86%
Copy back trans.	29562	379513	1709148	820587	35567	39445	515569

Tabela D.3: Resultados da *cache* L1 de instruções da CPU 0 – modelo AMD 8360SE

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	293973172	954012760	1170925788	342511772	224318937	225347373	1316878281
Instr. fetch misses	17300	461639	2333308	706054	14107	21449	2101286
Instr. fetch hit ratio	99.99%	99.95%	99.80%	99.79%	99.99%	99.99%	99.84%

Tabela D.4: Resultados da *cache* L1 de dados da CPU 0 – modelo AMD 8360SE

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	110268913	382881808	768663954	371129313	240668389	187818269	506637975
Data read misses	153571	2858320	6587503	2695016	4740584	339337	3262708
Data read hit ratio	99.86%	99.25%	99.14%	99.27%	98.03%	99.82%	99.36%
Data write trans.	63343552	193327073	472105098	217280799	78556901	84300008	288476891
Data write misses	592546	11898203	21619187	14588467	1251633	3521275	15033784
Data write hit ratio	99.06%	93.85%	95.42%	93.29%	98.41%	95.82%	94.79%

Tabela D.5: Resultados da *cache* L2 da CPU 1 – modelo AMD 8360SE

Cache nível 2 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	37911	171433	2320468	48759	42927	38093	1870792
Instr. fetch misses	20924	40613	399352	18062	21272	18689	126391
Instr. fetch hit ratio	44.81%	76.31%	82.79%	62.96%	50.45%	50.94%	93.24%
Data read trans.	315396	2448374	10494632	901451	4841081	407390	2027734
Data read misses	83236	316085	6012148	385996	102244	74148	451702
Data read hit ratio	73.61%	87.09%	42.71%	57.18%	97.89%	81.80%	77.72%
Data write trans.	62631209	186897381	438411302	206519553	78050223	83360683	290371351
Data write misses	57555	200520	591595	173747	59087	60184	318751
Data write hit ratio	99.91%	99.89%	99.87%	99.92%	99.92%	99.93%	99.89%
Copy back trans.	73302	242438	2478265	294901	77940	76683	398485

Tabela D.6: Resultados da *cache* L1 de instruções da CPU 1 – modelo AMD 8360SE

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	300556368	1023606186	1441801230	430023352	236083010	233114498	1291101220
Instr. fetch misses	37911	171433	2320468	48759	42927	38093	1870792
Instr. fetch hit ratio	99.99%	99.98%	99.84%	99.99%	99.98%	99.98%	99.86%

Tabela D.7: Resultados da *cache* L1 de dados da CPU 1 – modelo AMD 8360SE

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	111774733	401271143	862593985	382332846	243018293	190030094	492188629
Data read misses	315396	2448374	10494632	901451	4841081	407390	2027734
Data read hit ratio	99.72%	99.39%	98.78%	99.76%	98.01%	99.79%	99.59%
Data write trans.	62631209	186897381	438411302	206519553	78050223	83360683	290371351
Data write misses	1722825	11202393	13657452	3464670	2541702	4374141	12958506
Data write hit ratio	97.25%	94.01%	96.88%	98.32%	96.74%	94.75%	95.54%

Tabela D.8: Resultados da *cache* L2 da CPU 2 – modelo AMD 8360SE

Cache nível 2 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	55263	201937	115566	218914	47674	32171	1595503
Instr. fetch misses	25806	59087	49727	22626	21594	21221	101619
Instr. fetch hit ratio	53.30%	70.74%	56.97%	89.66%	54.70%	34.04%	93.63%
Data read trans.	331693	2618502	7025349	1273081	4864523	427030	2085386
Data read misses	94213	363765	5600899	514803	110726	86360	508914
Data read hit ratio	71.60%	86.11%	20.28%	59.56%	97.72%	79.78%	75.60%
Data write trans.	62548741	188702012	441560223	208705143	78331751	82442233	293639636
Data write misses	73162	240218	160643	244006	83711	60988	295348
Data write hit ratio	99.88%	99.87%	99.96%	99.88%	99.89%	99.93%	99.90%
Copy back trans.	90381	301757	2964755	399412	103705	77795	366433

Tabela D.9: Resultados da *cache* L1 de instruções da CPU 2 – modelo AMD 8360SE

Cache de Instruções nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	303897466	1018072990	1448386770	423138088	230254887	238679792	1329832902
Instr. fetch misses	55263	201937	115566	218914	47674	32171	1595503
Instr. fetch hit ratio	99.98%	99.98%	99.99%	99.95%	99.98%	99.99%	99.88%

Tabela D.10: Resultados da *cache* L1 de dados da CPU 2 – modelo AMD 8360SE

Cache de Dados nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	112628806	391317104	778508440	383121574	242428168	192117938	497553574
Data read misses	331693	2618502	7025349	1273081	4864523	427030	2085386
Data read hit ratio	99.71%	99.33%	99.10%	99.67%	97.99%	99.78%	99.58%
Data write trans.	62548741	188702012	441560223	208705143	78331751	82442233	293639636
Data write misses	2159256	11066250	4093429	5192339	3142313	4380806	13012323
Data write hit ratio	96.55%	94.14%	99.07%	97.51%	95.99%	94.69%	95.57%

Tabela D.11: Resultados da *cache* L2 da CPU 3 – modelo AMD 8360SE

Cache nível 2 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	39823	242599	373415	459322	27992	43050	1507531
Instr. fetch misses	23093	65535	51062	33790	17156	18424	117106
Instr. fetch hit ratio	42.01%	72.99%	86.33%	92.64%	38.71%	57.20%	92.23%
Data read trans.	318202	2771314	8130039	1615648	4779598	377086	2218621
Data read misses	83658	396665	4867872	455829	94323	71702	545195
Data read hit ratio	73.71%	85.69%	40.12%	71.79%	98.03%	80.99%	75.43%
Data write trans.	63070776	189290796	442894460	208725054	77339344	83872808	288324059
Data write misses	61200	295607	419879	271207	53424	62840	337869
Data write hit ratio	99.90%	99.84%	99.91%	99.87%	99.93%	99.93%	99.88%
Copy back trans.	77118	354540	1779274	401249	70282	76931	420317

Tabela D.12: Resultados da *cache* L1 de instruções da CPU 3 – modelo AMD 8360SE

Cache de Instruções nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	303446493	1004756113	1450451335	401020117	234143405	228903811	1299910379
Instr. fetch misses	39823	242599	373415	459322	27992	43050	1507531
Instr. fetch hit ratio	99.99%	99.98%	99.97%	99.89%	99.99%	99.98%	99.88%

Tabela D.13: Resultados da *cache* L1 de dados da CPU 3 – modelo AMD 8360SE

Cache de Dados nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	111773214	390750221	808619946	381059917	243603999	188790709	496792859
Data read misses	318202	2771314	8130039	1615648	4779598	377086	2218621
Data read hit ratio	99.72%	99.29%	98.99%	99.58%	98.04%	99.80%	99.55%
Data write trans.	63070776	189290796	442894460	208725054	77339344	83872808	288324059
Data write misses	1949115	12999771	5815424	6797772	2418127	4428814	12418651
Data write hit ratio	96.91%	93.13%	98.69%	96.74%	96.87%	94.72%	95.69%

Tabela D.14: Ciclos e tempo de execução dos *benchmarks* – modelo AMD 8360SE

Tempo de Execução – AMD Opteron 8360SE							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	240282840	3368707153	9294661349	3603505466	1227860812	952581741	5295828854
Time exec. (s)	0,096	1,347	3,718	1,441	0,491	0,381	2,118

APÊNDICE E :
RESULTADOS DA FAMÍLIA AMD PHENOM X4

E.1 Modelo 9950

Tabela E.1: Resultados da *cache* L3 – modelo AMD 9950

Cache nível 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	81050	251649	126637	230348	115776	77883	376846
Instr. fetch misses	16183	107728	69718	101385	32490	17120	146238
Instr. fetch hit ratio	80.03%	57.19%	44.95%	55.99%	71.94%	78.02%	61.19%
Data read trans.	501697	2799156	19229252	3126261	1458270	485531	3724466
Data read misses	114314	1380241	13339183	1940758	228397	128486	2338058
Data read hit ratio	77.21%	50.69%	30.63%	37.92%	84.34%	73.54%	37.22%
Data write trans.	265689	1510187	8206825	1614301	378370	271524	1708828
Data write misses	33684	263627	1458538	554997	41575	39934	453356
Data write hit ratio	87.32%	82.54%	82.23%	65.62%	89.01%	85.29%	73.47%
Copy back trans.	98258	987227	6637162	1181289	172203	114308	1396067

Tabela E.2: Resultados da *cache* L2 da CPU 0 – modelo AMD 9950

Cache nível 2 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	11764	713664	1180645	322862	30581	50706	1529515
Instr. fetch misses	6683	69246	169422	47947	11403	14186	62563
Instr. fetch hit ratio	43.19%	90.30%	85.65%	85.15%	62.72%	72.02%	95.91%
Data read trans.	115876	4519460	5711688	1356063	4878483	331288	2289952
Data read misses	21967	496140	1765207	491682	229498	47703	437817
Data read hit ratio	81.04%	89.02%	69.09%	63.74%	95.30%	85.60%	80.88%
Data write trans.	63450604	195053031	463307797	213547898	83769972	84489056	299341282
Data write misses	19116	437569	398067	269129	32578	39511	361677
Data write hit ratio	99.97%	99.78%	99.91%	99.87%	99.96%	99.95%	99.88%
Copy back trans.	22737	542283	1214387	401748	52537	50311	459445

Tabela E.3: Resultados da *cache* L1 de instruções da CPU 0 – modelo AMD 9950

Cache de Instruções nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	289670176	868861321	1824713925	991285983	600125692	461305967	1336280626
Instr. fetch misses	7764	713664	1180645	322862	30581	50706	1609758
Instr. fetch hit ratio	99.99%	99.92%	99.94%	99.97%	99.99%	99.99%	99.88%

Tabela E.4: Resultados da *cache* L1 de dados da CPU 0 – modelo AMD 9950

Cache de Dados nível 1 – CPU 0							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	108881401	374531929	766850797	389840310	252543509	188571301	492625898
Data read misses	115876	4519460	5711688	1356063	4878483	331288	2289952
Data read hit ratio	99.89%	98.79%	99.26%	99.65%	98.07%	99.82%	99.54%
Data write trans.	63450604	195053031	463307797	213547898	83769972	84489056	299341282
Data write misses	417314	16462567	13063036	5565863	1611271	3531669	13552286
Data write hit ratio	99.34%	91.56%	97.18%	97.39%	98.08%	95.82%	95.47%

Tabela E.5: Resultados da *cache* L2 da CPU 1 – modelo AMD 9950

Cache nível 2 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	45849	229821	72123	101903	54241	48816	1609758
Instr. fetch misses	15088	56059	27154	49806	31788	24391	107498
Instr. fetch hit ratio	67.09%	75.61%	62.35%	51.12%	41.39%	50.03%	93.32%
Data read trans.	224881	2718873	2379033	1140320	4814177	456611	2169440
Data read misses	53633	289600	1634474	544780	289280	79030	606245
Data read hit ratio	76.15%	89.35%	31.30%	52.23%	93.99%	82.69%	72.06%
Data write trans.	63408906	188895224	449591296	210806930	83504679	82650458	288453522
Data write misses	45345	190955	111029	210390	76220	61463	291348
Data write hit ratio	99.93%	99.90%	99.98%	99.90%	99.91%	99.93%	99.90%
Copy back trans.	56026	231825	823110	366257	100855	77502	367248

Tabela E.6: Resultados da *cache* L1 de instruções da CPU 1 – modelo AMD 9950

Cache de Instruções nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	295425634	1001167513	2070122620	1041301741	609836803	477533289	1336280626
Instr. fetch misses	45849	229821	72123	101903	54241	48816	1609758
Instr. fetch hit ratio	99.98%	99.98%	99.99%	99.99%	99.99%	99.99%	99.88%

Tabela E.7: Resultados da *cache* L1 de dados da CPU 1 – modelo AMD 9950

Cache de Dados nível 1 – CPU 1							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	109507772	388323499	804554876	395141059	252326214	192786786	515641165
Data read misses	224881	2718873	2379033	1140320	4814177	456611	2169440
Data read hit ratio	99.79%	99.30%	99.70%	99.71%	98.09%	99.76%	99.58%
Data write trans.	63408906	188895224	449591296	210806930	83504679	82650458	288453522
Data write misses	1338761	8437336	3015748	4090077	2791529	4407516	11990926
Data write hit ratio	97.89%	95.53%	99.33%	98.06%	96.66%	94.67%	95.84%

Tabela E.8: Resultados da *cache* L2 da CPU 2 – modelo AMD 9950

Cache nível 2 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	109689	174751	82801	210403	109853	75324	1831633
Instr. fetch misses	37980	58781	31004	63113	38402	20818	96330
Instr. fetch hit ratio	65.37%	66.36%	62.56%	70.00%	65.04%	72.36%	94.74%
Data read trans.	486023	2739102	2298354	1293115	5097241	414351	2626425
Data read misses	128463	340372	1468605	535911	358153	77757	688509
Data read hit ratio	73.57%	87.57%	36.10%	58.56%	92.97%	81.23%	73.79%
Data write trans.	61843757	187070512	435277356	209628319	82608223	84219490	291032016
Data write misses	90482	239186	144295	364798	90803	61527	369031
Data write hit ratio	99.85%	99.87%	99.97%	99.83%	99.89%	99.93%	99.87%
Copy back trans.	113836	303121	756064	500048	120030	77407	461462

Tabela E.9: Resultados da *cache* L1 de instruções da CPU 2 – modelo AMD 9950

Cache de Instruções nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	314584753	1016062923	2224217240	1059606122	622589385	468942416	1326098740
Instr. fetch misses	109689	174751	82801	210403	109853	75324	1831633
Instr. fetch hit ratio	99.97%	99.98%	99.99%	99.98%	99.98%	99.98%	99.86%

Tabela E.10: Resultados da *cache* L1 de dados da CPU 2 – modelo AMD 9950

Cache de Dados nível 1 – CPU 2							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	115030733	390108792	929950909	401734807	255980149	189923943	520459129
Data read misses	486023	2739102	2298354	1293115	5097241	414351	2626425
Data read hit ratio	99.58%	99.30%	99.75%	99.68%	98.01%	99.78%	99.50%
Data write trans.	61843757	187070512	435277356	209628319	82608223	84219490	291032016
Data write misses	2966962	10626427	3968001	7699180	3385956	3874664	14537015
Data write hit ratio	95.20%	94.32%	99.09%	96.33%	95.90%	95.40%	95.01%

Tabela E.11: Resultados da *cache* L2 da CPU 3 – modelo AMD 9950

Cache nível 2 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	34014	225148	1094524	125145	64810	35734	1518604
Instr. fetch misses	20442	67563	52556	69482	27153	18488	110455
Instr. fetch hit ratio	39.90%	69.99%	95.20%	44.48%	58.10%	48.26%	92.73%
Data read trans.	297109	2856604	19415180	1131374	4900759	425424	2125158
Data read misses	87789	438433	12856077	487119	301416	66272	643034
Data read hit ratio	70.45%	84.65%	33.78%	56.94%	93.85%	84.42%	69.74%
Data write trans.	62322127	186281292	421065623	209050270	83864545	83797919	287860882
Data write misses	54902	366901	851498	222452	80322	52268	326805
Data write hit ratio	99.91%	99.80%	99.80%	99.89%	99.90%	99.94%	99.89%
Copy back trans.	73090	432958	5413264	346248	104948	66304	420673

Tabela E.12: Resultados da *cache* L1 de instruções da CPU 3 – modelo AMD 9950

Cache de Instruções nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Instr. fetch trans.	298497825	1009167268	2013493145	1052814567	608807624	464475070	1351227439
Instr. fetch misses	34014	225148	1094524	125145	64810	35734	1518604
Instr. fetch hit ratio	99.99%	99.98%	99.95%	99.99%	99.99%	99.99%	99.89%

Tabela E.13: Resultados da *cache* L1 de dados da CPU 3 – modelo AMD 9950

Cache de Dados nível 1 – CPU 3							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Data read trans.	112097157	398956969	807455230	395699132	252531641	189519521	524288663
Data read misses	297109	2856604	19415180	1131374	4900759	425424	2125158
Data read hit ratio	99.73%	99.28%	97.60%	99.71%	98.06%	99.78%	99.59%
Data write trans.	62322127	186281292	421065623	209050270	83864545	83797919	287860882
Data write misses	1704822	17600640	14518998	4811365	2883614	4135768	13233123
Data write hit ratio	97.26%	90.55%	96.55%	97.70%	96.56%	95.06%	95.40%

Tabela E.14: Ciclos e tempo de execução dos *benchmarks* – modelo AMD 9950

Tempo de Execução – AMD Opteron 8360SE							
	Blackscholes	Bodytrack	Canneal	Fluidanimate	Streamcluster	Swaptions	x264
Cycles	248840355	3547222173	9634749636	3730767727	1439740980	1170961236	5492020056
Time exec. (s)	0,096	1,364	3,706	1,435	0,554	0,450	2,112

APÊNDICE F : *SCRIPTS* DE MODELAGEM E LATÊNCIAS DAS ARQUITETURAS

F.1 Família Intel Core 2 Duo

F.1.1 Modelo T7800

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 2}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "pentium-4e"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação Memória Principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = [conf.cpu0, conf.cpu1]
@l20.config_line_number = 65536
@l20.config_line_size = 64
@l20.config_assoc = 8
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = staller

#
#Especificação I-cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 512
@ic0.config_line_size = 64
```

```
@ic0.config_assoc = 4
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 512
@ic1.config_line_size = 64
@ic1.config_assoc = 4
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l20

#
#Especificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 512
@dc0.config_line_size = 64
@dc0.config_assoc = 4
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 512
@dc1.config_line_size = 64
@dc1.config_assoc = 4
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l20

#
#Transaction Splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64
```

```
#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

#
#Definição da hierarquia
#
@l20.higher_level_caches = [ic0, dc0, ic1, dc1]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration([staller, l20, ic0, ic1, dc0, dc1, ts_i0, ts_i1, ts_d0,
ts_d1, id0, id1], None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1

staller->stall_time = 38
l20->penalty_read = 11
l20->penalty_write = 11
ic0->penalty_read = 4
ic0->penalty_write = 4
ic1->penalty_read = 4
ic1->penalty_write = 4
dc0->penalty_read = 4
dc0->penalty_write = 4
dc1->penalty_read = 4
dc1->penalty_write = 4
```

F.1.2 Modelo T9500

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 2}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "pentium-4e"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"
```

```
#
#Especificação memória principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = [conf.cpu0, conf.cpu1]
@l20.config_line_number = 98304
@l20.config_line_size = 64
@l20.config_assoc = 12
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = staller

#
#Especificação I-Cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 512
@ic0.config_line_size = 64
@ic0.config_assoc = 4
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 512
@ic1.config_line_size = 64
@ic1.config_assoc = 4
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l20

#
#Especificação D-Cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 512
@dc0.config_line_size = 64
@dc0.config_assoc = 4
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
```

```
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 512
@dc1.config_line_size = 64
@dc1.config_assoc = 4
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l20

#
#Transaction Splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

#
#Definição da hierarquia
#
@l20.higher_level_caches = [ic0, dc0, ic1, dc1]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration([staller, l20, ic0, ic1, dc0, dc1, ts_i0, ts_i1, ts_d0,
ts_d1, id0, id1], None)
```

```
@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1

staller->stall_time = 38
l20->penalty_read = 9
l20->penalty_write = 9
ic0->penalty_read = 4
ic0->penalty_write = 4
ic1->penalty_read = 4
ic1->penalty_write = 4
dc0->penalty_read = 4
dc0->penalty_write = 4
dc1->penalty_read = 4
dc1->penalty_write = 4
```

F.2 Família AMD Turion 64x2

F.2.1 Modelo TL-68

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 2}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "x86-hammer"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação Memória Principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = conf.cpu0
@l20.config_line_number = 8192
@l20.config_line_size = 64
@l20.config_assoc = 16
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = staller

@l21 = pre_conf_object('l21', 'g-cache')
@l21.cpus = conf.cpu1
@l21.config_line_number = 8192
@l21.config_line_size = 64
@l21.config_assoc = 16
@l21.config_virtual_index = 0
@l21.config_virtual_tag = 0
```

```
@l21.config_write_back = 1
@l21.config_write_allocate = 1
@l21.config_replacement_policy = 'lru'
@l21.penalty_read = 0
@l21.penalty_write = 0
@l21.penalty_read_next = 0
@l21.penalty_write_next = 0
@l21.timing_model = staller

#
#Especificação I-cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 1024
@ic0.config_line_size = 64
@ic0.config_assoc = 2
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 1024
@ic1.config_line_size = 64
@ic1.config_assoc = 2
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l21

#
#Especificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 1024
@dc0.config_line_size = 64
@dc0.config_assoc = 2
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 1024
@dc1.config_line_size = 64
@dc1.config_assoc = 2
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
```

```
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l21

#
#Transaction splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

#
#Transaction splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

#
#Definição da hierarquia e coerência
#
@l20.higher_level_caches = [ic0, dc0]
@l21.higher_level_caches = [ic1, dc1]
@l20.snoopers = [l21]
@l21.snoopers = [l20]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration([staller, l20, l21, ic0, ic1, dc0, dc1, ts_i0, ts_i1,
ts_d0, ts_d1, id0, id1], None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1

staller->stall_time = 38
l20->penalty_read = 9
l20->penalty_write = 9
l21->penalty_read = 9
l21->penalty_write = 9
ic0->penalty_read = 3
ic0->penalty_write = 3
ic1->penalty_read = 3
ic1->penalty_write = 3
```

```
dc0->penalty_read = 3
dc0->penalty_write = 3
dc1->penalty_read = 3
dc1->penalty_write = 3
```

F.2.2 Modelo ZM-86

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 2}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "x86-hammer"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação Memória Principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = conf.cpu0
@l20.config_line_number = 16384
@l20.config_line_size = 64
@l20.config_assoc = 16
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = staller

@l21 = pre_conf_object('l21', 'g-cache')
@l21.cpus = conf.cpu1
@l21.config_line_number = 16384
@l21.config_line_size = 64
@l21.config_assoc = 16
@l21.config_virtual_index = 0
@l21.config_virtual_tag = 0
@l21.config_write_back = 1
@l21.config_write_allocate = 1
@l21.config_replacement_policy = 'lru'
@l21.penalty_read = 0
@l21.penalty_write = 0
@l21.penalty_read_next = 0
@l21.penalty_write_next = 0
@l21.timing_model = staller

#
#Especificação I-cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
```

```
@ic0.config_line_number = 1024
@ic0.config_line_size = 64
@ic0.config_assoc = 2
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 1024
@ic1.config_line_size = 64
@ic1.config_assoc = 2
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l21

#
#Esepcificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 1024
@dc0.config_line_size = 64
@dc0.config_assoc = 2
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 1024
@dc1.config_line_size = 64
@dc1.config_assoc = 2
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l21

#
#Transaction Splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
```

```
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

#
#Definição da hierarquia e coerência
#
@l20.higher_level_caches = [ic0, dc0]
@l21.higher_level_caches = [ic1, dc1]
@l20.snoopers = [l21]
@l21.snoopers = [l20]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration([staller, l20, l21, ic0, ic1, dc0, dc1, ts_i0, ts_i1,
ts_d0, ts_d1, id0, id1], None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1

staller->stall_time = 38
l20->penalty_read = 10
l20->penalty_write = 10
l21->penalty_read = 10
l21->penalty_write = 10
ic0->penalty_read = 3
ic0->penalty_write = 3
ic1->penalty_read = 3
ic1->penalty_write = 3
dc0->penalty_read = 3
dc0->penalty_write = 3
dc1->penalty_read = 3
dc1->penalty_write = 3
```

F.3 Família Intel Core 2 Quad

F.3.1 Modelo Q6700

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
```

```
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 4}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "pentium-4e"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação memória principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = [conf.cpu0, conf.cpu1]
@l20.config_line_number = 65536
@l20.config_line_size = 64
@l20.config_assoc = 16
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = staller

@l21 = pre_conf_object('l21', 'g-cache')
@l21.cpus = [conf.cpu2, conf.cpu3]
@l21.config_line_number = 65536
@l21.config_line_size = 64
@l21.config_assoc = 16
@l21.config_virtual_index = 0
@l21.config_virtual_tag = 0
@l21.config_write_back = 1
@l21.config_write_allocate = 1
@l21.config_replacement_policy = 'lru'
@l21.penalty_read = 0
@l21.penalty_write = 0
@l21.penalty_read_next = 0
@l21.penalty_write_next = 0
@l21.timing_model = staller

#
#Especificação I-cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 512
@ic0.config_line_size = 64
@ic0.config_assoc = 8
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20
```

```
@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 512
@ic1.config_line_size = 64
@ic1.config_assoc = 8
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l20

@ic2 = pre_conf_object('ic2', 'g-cache')
@ic2.cpus = conf.cpu2
@ic2.config_line_number = 512
@ic2.config_line_size = 64
@ic2.config_assoc = 8
@ic2.config_virtual_index = 0
@ic2.config_virtual_tag = 0
@ic2.config_replacement_policy = 'lru'
@ic2.penalty_read = 0
@ic2.penalty_write = 0
@ic2.penalty_read_next = 0
@ic2.penalty_write_next = 0
@ic2.timing_model = l21

@ic3 = pre_conf_object('ic3', 'g-cache')
@ic3.cpus = conf.cpu3
@ic3.config_line_number = 512
@ic3.config_line_size = 64
@ic3.config_assoc = 8
@ic3.config_virtual_index = 0
@ic3.config_virtual_tag = 0
@ic3.config_replacement_policy = 'lru'
@ic3.penalty_read = 0
@ic3.penalty_write = 0
@ic3.penalty_read_next = 0
@ic3.penalty_write_next = 0
@ic3.timing_model = l21

#
#Especificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 512
@dc0.config_line_size = 64
@dc0.config_assoc = 8
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 512
@dc1.config_line_size = 64
@dc1.config_assoc = 8
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
```

```
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l20

@dc2 = pre_conf_object('dc2', 'g-cache')
@dc2.cpus = conf.cpu2
@dc2.config_line_number = 512
@dc2.config_line_size = 64
@dc2.config_assoc = 8
@dc2.config_virtual_index = 0
@dc2.config_virtual_tag = 0
@dc2.config_replacement_policy = 'lru'
@dc2.penalty_read = 0
@dc2.penalty_write = 0
@dc2.penalty_read_next = 0
@dc2.penalty_write_next = 0
@dc2.timing_model = l21

@dc3 = pre_conf_object('dc3', 'g-cache')
@dc3.cpus = conf.cpu3
@dc3.config_line_number = 512
@dc3.config_line_size = 64
@dc3.config_assoc = 8
@dc3.config_virtual_index = 0
@dc3.config_virtual_tag = 0
@dc3.config_replacement_policy = 'lru'
@dc3.penalty_read = 0
@dc3.penalty_write = 0
@dc3.penalty_read_next = 0
@dc3.penalty_write_next = 0
@dc3.timing_model = l21

#
#Transaction Splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

@ts_i2 = pre_conf_object('ts_i2', 'trans-splitter')
@ts_i2.cache = ic2
@ts_i2.timing_model = ic2
@ts_i2.next_cache_line_size = 64

@ts_i3 = pre_conf_object('ts_i3', 'trans-splitter')
@ts_i3.cache = ic3
@ts_i3.timing_model = ic3
@ts_i3.next_cache_line_size = 64

#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
```

```
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

@ts_d2 = pre_conf_object('ts_d2', 'trans-splitter')
@ts_d2.cache = dc2
@ts_d2.timing_model = dc2
@ts_d2.next_cache_line_size = 64

@ts_d3 = pre_conf_object('ts_d3', 'trans-splitter')
@ts_d3.cache = dc3
@ts_d3.timing_model = dc3
@ts_d3.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

@id2 = pre_conf_object('id2', 'id-splitter')
@id2.ibranch = ts_i2
@id2.dbranch = ts_d2

@id3 = pre_conf_object('id3', 'id-splitter')
@id3.ibranch = ts_i3
@id3.dbranch = ts_d3

#
#Definição da hierarquia e coerência
#
@l20.higher_level_caches = [ic0, ic1, dc0, dc1]
@l21.higher_level_caches = [ic2, ic3, dc2, dc3]
@l20.snoopers = [l21]
@l21.snoopers = [l20]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration(
[staller, l20, l21, ic0, ic1, ic2, ic3, dc0, dc1, dc2, dc3, ts_i0, ts_i1,
ts_i2, ts_i3, ts_d0, ts_d1, ts_d2, ts_d3, id0, id1, id2, id3], None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1
@conf.cpu2_mem.timing_model = conf.id2
@conf.cpu3_mem.timing_model = conf.id3

staller->stall_time = 38
l20->penalty_read = 13
l20->penalty_write = 13
l21->penalty_read = 13
l21->penalty_write = 13
ic0->penalty_read = 6
ic0->penalty_write = 6
ic1->penalty_read = 6
ic1->penalty_write = 6
ic2->penalty_read = 6
ic2->penalty_write = 6
ic3->penalty_read = 6
ic3->penalty_write = 6
dc0->penalty_read = 6
```

```
dc0->penalty_write = 6
dc1->penalty_read = 6
dc1->penalty_write = 6
dc2->penalty_read = 6
dc2->penalty_write = 6
dc3->penalty_read = 6
dc3->penalty_write = 6
```

F.3.2 Modelo Q9650

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 4}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "pentium-4e"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação memória principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = [conf.cpu0, conf.cpu1]
@l20.config_line_number = 98304
@l20.config_line_size = 64
@l20.config_assoc = 12
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = staller

@l21 = pre_conf_object('l21', 'g-cache')
@l21.cpus = [conf.cpu2, conf.cpu3]
@l21.config_line_number = 98304
@l21.config_line_size = 64
@l21.config_assoc = 12
@l21.config_virtual_index = 0
@l21.config_virtual_tag = 0
@l21.config_write_back = 1
@l21.config_write_allocate = 1
@l21.config_replacement_policy = 'lru'
@l21.penalty_read = 0
@l21.penalty_write = 0
@l21.penalty_read_next = 0
@l21.penalty_write_next = 0
@l21.timing_model = staller

#
#Especificação I-cache L1
```

```
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 512
@ic0.config_line_size = 64
@ic0.config_assoc = 8
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 512
@ic1.config_line_size = 64
@ic1.config_assoc = 8
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l20

@ic2 = pre_conf_object('ic2', 'g-cache')
@ic2.cpus = conf.cpu2
@ic2.config_line_number = 512
@ic2.config_line_size = 64
@ic2.config_assoc = 8
@ic2.config_virtual_index = 0
@ic2.config_virtual_tag = 0
@ic2.config_replacement_policy = 'lru'
@ic2.penalty_read = 0
@ic2.penalty_write = 0
@ic2.penalty_read_next = 0
@ic2.penalty_write_next = 0
@ic2.timing_model = l21

@ic3 = pre_conf_object('ic3', 'g-cache')
@ic3.cpus = conf.cpu3
@ic3.config_line_number = 512
@ic3.config_line_size = 64
@ic3.config_assoc = 8
@ic3.config_virtual_index = 0
@ic3.config_virtual_tag = 0
@ic3.config_replacement_policy = 'lru'
@ic3.penalty_read = 0
@ic3.penalty_write = 0
@ic3.penalty_read_next = 0
@ic3.penalty_write_next = 0
@ic3.timing_model = l21

#
#Especificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 512
@dc0.config_line_size = 64
@dc0.config_assoc = 8
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
```

```
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 512
@dc1.config_line_size = 64
@dc1.config_assoc = 8
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l20

@dc2 = pre_conf_object('dc2', 'g-cache')
@dc2.cpus = conf.cpu2
@dc2.config_line_number = 512
@dc2.config_line_size = 64
@dc2.config_assoc = 8
@dc2.config_virtual_index = 0
@dc2.config_virtual_tag = 0
@dc2.config_replacement_policy = 'lru'
@dc2.penalty_read = 0
@dc2.penalty_write = 0
@dc2.penalty_read_next = 0
@dc2.penalty_write_next = 0
@dc2.timing_model = l21

@dc3 = pre_conf_object('dc3', 'g-cache')
@dc3.cpus = conf.cpu3
@dc3.config_line_number = 512
@dc3.config_line_size = 64
@dc3.config_assoc = 8
@dc3.config_virtual_index = 0
@dc3.config_virtual_tag = 0
@dc3.config_replacement_policy = 'lru'
@dc3.penalty_read = 0
@dc3.penalty_write = 0
@dc3.penalty_read_next = 0
@dc3.penalty_write_next = 0
@dc3.timing_model = l21

#
#Transaction Splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

@ts_i2 = pre_conf_object('ts_i2', 'trans-splitter')
@ts_i2.cache = ic2
@ts_i2.timing_model = ic2
@ts_i2.next_cache_line_size = 64
```

```
@ts_i3 = pre_conf_object('ts_i3', 'trans-splitter')
@ts_i3.cache = ic3
@ts_i3.timing_model = ic3
@ts_i3.next_cache_line_size = 64

#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

@ts_d2 = pre_conf_object('ts_d2', 'trans-splitter')
@ts_d2.cache = dc2
@ts_d2.timing_model = dc2
@ts_d2.next_cache_line_size = 64

@ts_d3 = pre_conf_object('ts_d3', 'trans-splitter')
@ts_d3.cache = dc3
@ts_d3.timing_model = dc3
@ts_d3.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

@id2 = pre_conf_object('id2', 'id-splitter')
@id2.ibranch = ts_i2
@id2.dbranch = ts_d2

@id3 = pre_conf_object('id3', 'id-splitter')
@id3.ibranch = ts_i3
@id3.dbranch = ts_d3

#
#Definição da hierarquia e coerência
#
@l20.higher_level_caches = [ic0, ic1, dc0, dc1]
@l21.higher_level_caches = [ic2, ic3, dc2, dc3]
@l20.snoopers = [l21]
@l21.snoopers = [l20]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration(
[staller, l20, l21, ic0, ic1, ic2, ic3, dc0, dc1, dc2, dc3, ts_i0, ts_i1,
ts_i2, ts_i3, ts_d0, ts_d1, ts_d2, ts_d3, id0, id1, id2, id3], None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1
@conf.cpu2_mem.timing_model = conf.id2
@conf.cpu3_mem.timing_model = conf.id3
```

```
staller->stall_time = 38
l20->penalty_read = 10
l20->penalty_write = 10
l21->penalty_read = 10
l21->penalty_write = 10
ic0->penalty_read = 6
ic0->penalty_write = 6
ic1->penalty_read = 6
ic1->penalty_write = 6
ic2->penalty_read = 6
ic2->penalty_write = 6
ic3->penalty_read = 6
ic3->penalty_write = 6
dc0->penalty_read = 6
dc0->penalty_write = 6
dc1->penalty_read = 6
dc1->penalty_write = 6
dc2->penalty_read = 6
dc2->penalty_write = 6
dc3->penalty_read = 6
dc3->penalty_write = 6
```

F.4 Família AMD Opteron x4

F.4.1 Modelo 8360SE

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 4}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "x86-hammer"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação memória principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L3
#
@l30 = pre_conf_object('l30', 'g-cache')
@l30.cpus = [conf.cpu0, conf.cpu1, conf.cpu2, conf.cpu3]
@l30.config_line_number = 32768
@l30.config_line_size = 64
@l30.config_assoc = 32
@l30.config_virtual_index = 0
@l30.config_virtual_tag = 0
@l30.config_write_back = 1
@l30.config_write_allocate = 1
@l30.config_replacement_policy = 'lru'
@l30.penalty_read = 0
@l30.penalty_write = 0
@l30.penalty_read_next = 0
@l30.penalty_write_next = 0
@l30.timing_model = staller

#
```

```
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = conf.cpu0
@l20.config_line_number = 8192
@l20.config_line_size = 64
@l20.config_assoc = 16
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = l30

@l21 = pre_conf_object('l21', 'g-cache')
@l21.cpus = conf.cpu1
@l21.config_line_number = 8192
@l21.config_line_size = 64
@l21.config_assoc = 16
@l21.config_virtual_index = 0
@l21.config_virtual_tag = 0
@l21.config_write_back = 1
@l21.config_write_allocate = 1
@l21.config_replacement_policy = 'lru'
@l21.penalty_read = 0
@l21.penalty_write = 0
@l21.penalty_read_next = 0
@l21.penalty_write_next = 0
@l21.timing_model = l30

@l22 = pre_conf_object('l22', 'g-cache')
@l22.cpus = conf.cpu2
@l22.config_line_number = 8192
@l22.config_line_size = 64
@l22.config_assoc = 16
@l22.config_virtual_index = 0
@l22.config_virtual_tag = 0
@l22.config_write_back = 1
@l22.config_write_allocate = 1
@l22.config_replacement_policy = 'lru'
@l22.penalty_read = 0
@l22.penalty_write = 0
@l22.penalty_read_next = 0
@l22.penalty_write_next = 0
@l22.timing_model = l30

@l23 = pre_conf_object('l23', 'g-cache')
@l23.cpus = conf.cpu3
@l23.config_line_number = 8192
@l23.config_line_size = 64
@l23.config_assoc = 16
@l23.config_virtual_index = 0
@l23.config_virtual_tag = 0
@l23.config_write_back = 1
@l23.config_write_allocate = 1
@l23.config_replacement_policy = 'lru'
@l23.penalty_read = 0
@l23.penalty_write = 0
@l23.penalty_read_next = 0
@l23.penalty_write_next = 0
@l23.timing_model = l30

#
```

```
#Especificação I-cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 1024
@ic0.config_line_size = 64
@ic0.config_assoc = 2
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 1024
@ic1.config_line_size = 64
@ic1.config_assoc = 2
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l21

@ic2 = pre_conf_object('ic2', 'g-cache')
@ic2.cpus = conf.cpu2
@ic2.config_line_number = 1024
@ic2.config_line_size = 64
@ic2.config_assoc = 2
@ic2.config_virtual_index = 0
@ic2.config_virtual_tag = 0
@ic2.config_replacement_policy = 'lru'
@ic2.penalty_read = 0
@ic2.penalty_write = 0
@ic2.penalty_read_next = 0
@ic2.penalty_write_next = 0
@ic2.timing_model = l22

@ic3 = pre_conf_object('ic3', 'g-cache')
@ic3.cpus = conf.cpu3
@ic3.config_line_number = 1024
@ic3.config_line_size = 64
@ic3.config_assoc = 2
@ic3.config_virtual_index = 0
@ic3.config_virtual_tag = 0
@ic3.config_replacement_policy = 'lru'
@ic3.penalty_read = 0
@ic3.penalty_write = 0
@ic3.penalty_read_next = 0
@ic3.penalty_write_next = 0
@ic3.timing_model = l23

#
#Especificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 1024
@dc0.config_line_size = 64
@dc0.config_assoc = 2
@dc0.config_virtual_index = 0
```

```
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 1024
@dc1.config_line_size = 64
@dc1.config_assoc = 2
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l21

@dc2 = pre_conf_object('dc2', 'g-cache')
@dc2.cpus = conf.cpu2
@dc2.config_line_number = 1024
@dc2.config_line_size = 64
@dc2.config_assoc = 2
@dc2.config_virtual_index = 0
@dc2.config_virtual_tag = 0
@dc2.config_replacement_policy = 'lru'
@dc2.penalty_read = 0
@dc2.penalty_write = 0
@dc2.penalty_read_next = 0
@dc2.penalty_write_next = 0
@dc2.timing_model = l22

@dc3 = pre_conf_object('dc3', 'g-cache')
@dc3.cpus = conf.cpu3
@dc3.config_line_number = 1024
@dc3.config_line_size = 64
@dc3.config_assoc = 2
@dc3.config_virtual_index = 0
@dc3.config_virtual_tag = 0
@dc3.config_replacement_policy = 'lru'
@dc3.penalty_read = 0
@dc3.penalty_write = 0
@dc3.penalty_read_next = 0
@dc3.penalty_write_next = 0
@dc3.timing_model = l23

#
#Transaction Splitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64

@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

@ts_i2 = pre_conf_object('ts_i2', 'trans-splitter')
@ts_i2.cache = ic2
@ts_i2.timing_model = ic2
@ts_i2.next_cache_line_size = 64
```

```
@ts_i3 = pre_conf_object('ts_i3', 'trans-splitter')
@ts_i3.cache = ic3
@ts_i3.timing_model = ic3
@ts_i3.next_cache_line_size = 64

#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

@ts_d2 = pre_conf_object('ts_d2', 'trans-splitter')
@ts_d2.cache = dc2
@ts_d2.timing_model = dc2
@ts_d2.next_cache_line_size = 64

@ts_d3 = pre_conf_object('ts_d3', 'trans-splitter')
@ts_d3.cache = dc3
@ts_d3.timing_model = dc3
@ts_d3.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

@id2 = pre_conf_object('id2', 'id-splitter')
@id2.ibranch = ts_i2
@id2.dbranch = ts_d2

@id3 = pre_conf_object('id3', 'id-splitter')
@id3.ibranch = ts_i3
@id3.dbranch = ts_d3

#
#Definição da hierarquia
#
@l30.higher_level_caches = [l20, l21, l22, l23]
@l20.higher_level_caches = [ic0, dc0]
@l21.higher_level_caches = [ic1, dc1]
@l22.higher_level_caches = [ic2, dc2]
@l23.higher_level_caches = [ic3, dc3]

#
#Adiciona componentes a configuração
#
@SIM_add_configuration(
[staller, l30, l20, l21, l22, l23, ic0, ic1, ic2, ic3, dc0, dc1, dc2, dc3,
ts_i0, ts_i1, ts_i2, ts_i3, ts_d0, ts_d1, ts_d2, ts_d3, id0, id1, id2, id3],
None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1
```

```
@conf.cpu2_mem.timing_model = conf.id2
@conf.cpu3_mem.timing_model = conf.id3

staller->stall_time = 38
l30->penalty_read = 16
l30->penalty_write = 16
l20->penalty_read = 9
l20->penalty_write = 9
l21->penalty_read = 9
l21->penalty_write = 9
l22->penalty_read = 9
l22->penalty_write = 9
l23->penalty_read = 9
l23->penalty_write = 9
ic0->penalty_read = 3
ic0->penalty_write = 3
ic1->penalty_read = 3
ic1->penalty_write = 3
ic2->penalty_read = 3
ic2->penalty_write = 3
ic3->penalty_read = 3
ic3->penalty_write = 3
dc0->penalty_read = 3
dc0->penalty_write = 3
dc1->penalty_read = 3
dc1->penalty_write = 3
dc2->penalty_read = 3
dc2->penalty_write = 3
dc3->penalty_read = 3
dc3->penalty_write = 3
```

F.5 Família AMD Phenom x4

F.5.1 Modelo 9950

```
if not defined freq_mhz      {$freq_mhz      = 20}
if not defined cpi           {$cpi           = 1}
if not defined disk_size     {$disk_size     = 20496236544}
if not defined disk_image    {$disk_image    = "ubuntu6.craff"}
if not defined rtc_time      {$rtc_time      = "2008-09-30 10:00:00 UTC"}
if not defined num_cpus      {$num_cpus      = 4}
if not defined memory_megs   {$memory_megs   = 512}
if not defined cpu_class     {$cpu_class     = "x86-hammer"}
if not defined text_console  {$text_console  = "yes"}
if not defined use_acpi      {$use_acpi      = TRUE}

add-directory "%script%"
run-command-file "%simics%/targets/x86-440bx/dredd-common.simics"

#
#Especificação memória principal
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
#Especificação Cache L3
#
@l30 = pre_conf_object('l30', 'g-cache')
@l30.cpus = [conf.cpu0, conf.cpu1, conf.cpu2, conf.cpu3]
@l30.config_line_number = 32768
@l30.config_line_size = 64
@l30.config_assoc = 32
@l30.config_virtual_index = 0
@l30.config_virtual_tag = 0
@l30.config_write_back = 1
```

```
@l30.config_write_allocate = 1
@l30.config_replacement_policy = 'lru'
@l30.penalty_read = 0
@l30.penalty_write = 0
@l30.penalty_read_next = 0
@l30.penalty_write_next = 0
@l30.timing_model = staller

#
#Especificação Cache L2
#
@l20 = pre_conf_object('l20', 'g-cache')
@l20.cpus = conf.cpu0
@l20.config_line_number = 8192
@l20.config_line_size = 64
@l20.config_assoc = 16
@l20.config_virtual_index = 0
@l20.config_virtual_tag = 0
@l20.config_write_back = 1
@l20.config_write_allocate = 1
@l20.config_replacement_policy = 'lru'
@l20.penalty_read = 0
@l20.penalty_write = 0
@l20.penalty_read_next = 0
@l20.penalty_write_next = 0
@l20.timing_model = l30

@l21 = pre_conf_object('l21', 'g-cache')
@l21.cpus = conf.cpu1
@l21.config_line_number = 8192
@l21.config_line_size = 64
@l21.config_assoc = 16
@l21.config_virtual_index = 0
@l21.config_virtual_tag = 0
@l21.config_write_back = 1
@l21.config_write_allocate = 1
@l21.config_replacement_policy = 'lru'
@l21.penalty_read = 0
@l21.penalty_write = 0
@l21.penalty_read_next = 0
@l21.penalty_write_next = 0
@l21.timing_model = l30

@l22 = pre_conf_object('l22', 'g-cache')
@l22.cpus = conf.cpu2
@l22.config_line_number = 8192
@l22.config_line_size = 64
@l22.config_assoc = 16
@l22.config_virtual_index = 0
@l22.config_virtual_tag = 0
@l22.config_write_back = 1
@l22.config_write_allocate = 1
@l22.config_replacement_policy = 'lru'
@l22.penalty_read = 0
@l22.penalty_write = 0
@l22.penalty_read_next = 0
@l22.penalty_write_next = 0
@l22.timing_model = l30

@l23 = pre_conf_object('l23', 'g-cache')
@l23.cpus = conf.cpu3
@l23.config_line_number = 8192
@l23.config_line_size = 64
@l23.config_assoc = 16
@l23.config_virtual_index = 0
@l23.config_virtual_tag = 0
@l23.config_write_back = 1
```

```
@l23.config_write_allocate = 1
@l23.config_replacement_policy = 'lru'
@l23.penalty_read = 0
@l23.penalty_write = 0
@l23.penalty_read_next = 0
@l23.penalty_write_next = 0
@l23.timing_model = l30

#
#Especificação I-cache L1
#
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 1024
@ic0.config_line_size = 64
@ic0.config_assoc = 2
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l20

@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 1024
@ic1.config_line_size = 64
@ic1.config_assoc = 2
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l21

@ic2 = pre_conf_object('ic2', 'g-cache')
@ic2.cpus = conf.cpu2
@ic2.config_line_number = 1024
@ic2.config_line_size = 64
@ic2.config_assoc = 2
@ic2.config_virtual_index = 0
@ic2.config_virtual_tag = 0
@ic2.config_replacement_policy = 'lru'
@ic2.penalty_read = 0
@ic2.penalty_write = 0
@ic2.penalty_read_next = 0
@ic2.penalty_write_next = 0
@ic2.timing_model = l22

@ic3 = pre_conf_object('ic3', 'g-cache')
@ic3.cpus = conf.cpu3
@ic3.config_line_number = 1024
@ic3.config_line_size = 64
@ic3.config_assoc = 2
@ic3.config_virtual_index = 0
@ic3.config_virtual_tag = 0
@ic3.config_replacement_policy = 'lru'
@ic3.penalty_read = 0
@ic3.penalty_write = 0
@ic3.penalty_read_next = 0
@ic3.penalty_write_next = 0
@ic3.timing_model = l23
```

```
#
#Especificação D-cache L1
#
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 1024
@dc0.config_line_size = 64
@dc0.config_assoc = 2
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l20

@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 1024
@dc1.config_line_size = 64
@dc1.config_assoc = 2
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l21

@dc2 = pre_conf_object('dc2', 'g-cache')
@dc2.cpus = conf.cpu2
@dc2.config_line_number = 1024
@dc2.config_line_size = 64
@dc2.config_assoc = 2
@dc2.config_virtual_index = 0
@dc2.config_virtual_tag = 0
@dc2.config_replacement_policy = 'lru'
@dc2.penalty_read = 0
@dc2.penalty_write = 0
@dc2.penalty_read_next = 0
@dc2.penalty_write_next = 0
@dc2.timing_model = l22

@dc3 = pre_conf_object('dc3', 'g-cache')
@dc3.cpus = conf.cpu3
@dc3.config_line_number = 1024
@dc3.config_line_size = 64
@dc3.config_assoc = 2
@dc3.config_virtual_index = 0
@dc3.config_virtual_tag = 0
@dc3.config_replacement_policy = 'lru'
@dc3.penalty_read = 0
@dc3.penalty_write = 0
@dc3.penalty_read_next = 0
@dc3.penalty_write_next = 0
@dc3.timing_model = l23

#
#Transaction Slitter I-cache L1
#
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64
```

```
@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64

@ts_i2 = pre_conf_object('ts_i2', 'trans-splitter')
@ts_i2.cache = ic2
@ts_i2.timing_model = ic2
@ts_i2.next_cache_line_size = 64

@ts_i3 = pre_conf_object('ts_i3', 'trans-splitter')
@ts_i3.cache = ic3
@ts_i3.timing_model = ic3
@ts_i3.next_cache_line_size = 64

#
#Transaction Splitter D-cache L1
#
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64

@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64

@ts_d2 = pre_conf_object('ts_d2', 'trans-splitter')
@ts_d2.cache = dc2
@ts_d2.timing_model = dc2
@ts_d2.next_cache_line_size = 64

@ts_d3 = pre_conf_object('ts_d3', 'trans-splitter')
@ts_d3.cache = dc3
@ts_d3.timing_model = dc3
@ts_d3.next_cache_line_size = 64

#
#Instruction-Data Splitter
#
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0

@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1

@id2 = pre_conf_object('id2', 'id-splitter')
@id2.ibranch = ts_i2
@id2.dbranch = ts_d2

@id3 = pre_conf_object('id3', 'id-splitter')
@id3.ibranch = ts_i3
@id3.dbranch = ts_d3

#
#Definição da hierarquia
#
@l30.higher_level_caches = [l20, l21, l22, l23]
@l20.higher_level_caches = [ic0, dc0]
@l21.higher_level_caches = [ic1, dc1]
@l22.higher_level_caches = [ic2, dc2]
@l23.higher_level_caches = [ic3, dc3]

#
```

```
#Adiciona componentes a configuração
#
@SIM_add_configuration(
[staller, l30, l20, l21, l22, l23, ic0, ic1, ic2, ic3, dc0, dc1, dc2, dc3,
ts_i0, ts_i1, ts_i2, ts_i3, ts_d0, ts_d1, ts_d2, ts_d3, id0, id1, id2, id3],
None)

@conf.cpu0_mem.timing_model = conf.id0
@conf.cpu1_mem.timing_model = conf.id1
@conf.cpu2_mem.timing_model = conf.id2
@conf.cpu3_mem.timing_model = conf.id3

staller->stall_time = 38
l30->penalty_read = 17
l30->penalty_write = 17
l20->penalty_read = 10
l20->penalty_write = 10
l21->penalty_read = 10
l21->penalty_write = 10
l22->penalty_read = 10
l22->penalty_write = 10
l23->penalty_read = 10
l23->penalty_write = 10
ic0->penalty_read = 3
ic0->penalty_write = 3
ic1->penalty_read = 3
ic1->penalty_write = 3
ic2->penalty_read = 3
ic2->penalty_write = 3
ic3->penalty_read = 3
ic3->penalty_write = 3
dc0->penalty_read = 3
dc0->penalty_write = 3
dc1->penalty_read = 3
dc1->penalty_write = 3
dc2->penalty_read = 3
dc2->penalty_write = 3
dc3->penalty_read = 3
dc3->penalty_write = 3
```