Ensino de Arquiteturas de Processadores Multi-Core Através de um Sistema de Simulação Completo e da Experiência de um Projeto de Pesquisa*

Henrique C. Freitas†, Marco A. Z. Alves, Nicolas B. Maillard, Philippe O. A. Navaux
Grupo de Processamento Paralelo e Distribuído
Instituto de Informática, Programa de Pós-Graduação em Computação
Universidade Federal do Rio Grande do Sul, Brasil
{hcfreitas, marco.zanata, nicolas, navaux}@inf.ufrgs.br

Resumo

Ensinar arquitetura de processadores é um grande desafio em se tratando da rápida evolução da área. Com o surgimento dos processadores multi-core e da possibilidade de processadores de propósito geral com dezenas e centenas de núcleos de processamento (many-core), esse desafio aumenta. Neste contexto surgem também várias alternativas de projeto com novos conceitos e técnicas que aumentam o conteúdo que deve ser passado nas disciplinas relacionadas à arquitetura de computadores. O objetivo deste artigo é descrever o uso de um ambiente de simulação de sistema completo, capaz de auxiliar no ensino, projeto e avaliação de um processador multi-core. O uso deste ambiente de simulação é resultado da experiência adquirida em um projeto de pesquisa, que tem mostrado potencialidades para uso em sala de aula. Este artigo apresenta a metodologia utilizada e uma análise dos trabalhos apresentados pelos alunos.

1. Introdução

Através dos novos processadores *multi-core* [1][2][3] o cenário de processamento de alto desempenho [4][5] tem se mostrado mais atraente tanto para a indústria quanto para a academia. Isto resulta em um aumento do nível de interesse em problemas já conhecidos, tal como a utilização de memória compartilhada [6]. Além disso, novos problemas (Seção 2) aparecem dentro do *chip* e com isto aumentam os desafios na utilização dos processadores. Considerando que a tendência é o aumento significativo dos núcleos de processamento, é necessário que a academia utilize ambientes que suportem a simulação de um sistema completo composto por arquiteturas de processadores *multi-core* para o ensino de arquiteturas.

O surgimento das arquiteturas de processadores *multi-core* não é tão recente. Ao longo da última década viu-se surgir uma série de processadores dedicados com uma quantidade de núcleos superior ao que normalmente se observa atualmente em processadores de propósito geral. Processadores de Rede [7][8] usavam cerca de dezenas de núcleos organizados de uma forma específica e com objetivos de processamento específicos. Nestes casos os núcleos não eram tão robustos, mas usavam das mesmas técnicas que hoje são utilizadas na maioria dos processadores *multi-core* de propósito geral. A Figura 1 ilustra exemplos de Processadores de Rede ressaltando a quantidade de núcleos.

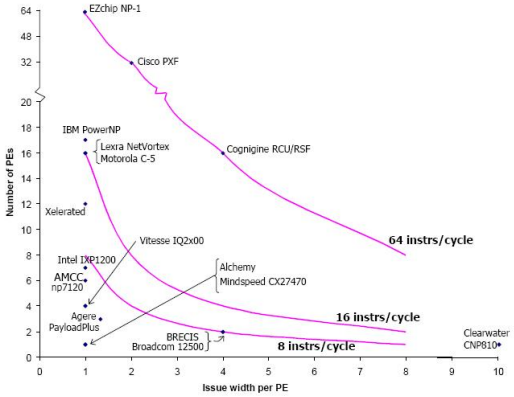

Figura 1. Alguns Processadores de Rede [8]

Espera-se de um núcleo robusto a capacidade de suportar paralelismo no nível de instrução e no nível de *thread* (fluxo de instruções) [9][10]. Um bom exemplo seria um processador com superescalaridade e com suporte a múltiplas *threads* simultâneas

† Licenciado da Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, Brasil, cota@pucminas.br.

* Este trabalho recebe um suporte parcial do CNPq e da Microsoft.

(*Simultaneous Multithreading* -SMT). No entanto, não são todas as aplicações que demandam este tipo de suporte do processador. Por exemplo, servidores *web* e de banco de dados recebem um número muito grande de requisições independentes, que geram transações independentes. Para este tipo de sistema, a vazão [11] de respostas às requisições é muito importante, já que assim o usuário não espera uma resposta por muito tempo, evitando duas conseqüências básicas:

- O usuário desiste de usar o sistema *web*.
- O usuário envia uma nova requisição aumentando o tráfego do sistema.

Esta carga de trabalho, por exemplo, possui um alto paralelismo no nível de *threads*, mas um baixo paralelismo no nível de instrução. Como conseqüência, um processador com núcleos escalares e com chaveamento entre *threads* possui melhor desempenho.

Este é o exemplo que resultou no projeto do processador Niagara (UltraSparc T1) [12], que possui oito núcleos escalares e com suporte a quatro *threads* cada um. Desta forma, o processador executa oito *threads* simultâneas e suporta 32 contextos de *threads* ativos. Este é um bom exemplo de processador para a carga de trabalho descrita anteriormente, mas para uso em um computador de propósito geral teria baixo desempenho, já que sua arquitetura não é superescalar (não suporta paralelismo no nível de instrução).

Considerando que núcleos mais simples ocupam menos espaço no chip, a probabilidade de uma geração de processadores com núcleos simples aumenta. É por isto que as GPUs (*Graphic Processing Units*) [13] atualmente possuem centenas de núcleos de processamento. São núcleos simples, mas para o propósito de processamento gráfico são extremamente úteis e com bom desempenho.

As alternativas existentes para projetar um processador *multi-core* e *many-core* são enormes. Associadas a cada alternativa existem diversos problemas. Alguns destes problemas pouco apareciam no projeto de um processador, mas atualmente são partes obrigatórias do projeto. Neste contexto, os alunos dos cursos de computação, em específico, arquiteturas de computadores devem ser estimulados a pensar nas possíveis técnicas de projeto e suas conseqüências.

Portanto, o objetivo deste artigo é apresentar o uso de um sistema de simulação completo e da experiência de um projeto de pesquisa [14] em sala de aula para o ensino de arquiteturas de processadores *multi-core*. As atividades realizadas com este ambiente de simulação servirão como base para futuros trabalhos envolvendo arquiteturas *many-core*.

O restante do artigo está organizado da seguinte forma: Desafios no Ensino de Arquitetura de Processadores *Multi-Core* e *Many-Core*, Projeto de Pesquisa Base, Características do Ambiente de Simulação Simics, Metodologia Utilizada, Análises e Discussão, Abordagens de Ensino Relacionadas e Conclusões.

2. Desafios no Ensino de Arquitetura de Processadores Multi-Core e Many-Core

Arquitetura de computadores possui um amplo conjunto de tópicos que devem ser abordados em cursos típicos da graduação ou da pós-graduação. No entanto, existem várias abordagens de projeto que podem ser vistas como tópicos avançados ou especiais, que estão presentes nos processadores comerciais de diversas formas.

Recentemente, foi discutido no *Workshop on Computer Architecture Education* (em conjunto com o ISCA 2008), como inserir e qual o melhor momento para ensinar o suporte de virtualização em hardware [15] para turmas de arquitetura de computadores, em específico, na graduação. Além desse tópico, foram discutidos em um painel diversos outros, que com o surgimento das arquiteturas de processadores *multi-core* aparecem como desafios no ensino.

Portanto, o suporte de virtualização em hardware já é um desafio, independente da arquitetura ser *multi-core*. Além do aluno ter que entender o conjunto básico de instruções de um processador, é necessário saber como suportar virtualização, quais as instruções necessárias e como elas se interagem com o sistema de memória e sistema operacional. Considerando, que em um curso de computação a disciplina Sistemas Operacionais é oferecida após Arquitetura de Computadores, mostrar a relação entre instruções e máquinas virtuais pode não ser tão trivial. Se o processador for *multi-core* e principalmente *many-core*, a virtualização passa a ter um ponto positivo muito forte, que é o suporte nativo do processador com dezenas ou centenas de núcleos de processamento. Isto significa executar *threads* simultaneamente sem aumentar a complexidade dos núcleos. Entretanto, como ficaria a comunicação interna e o mapeamento dos processos de máquinas virtuais?

Esta pergunta nos leva a outro desafio, que se refere à rede interna de comunicação do processador. Com o aumento do número de núcleos, soluções típicas como barramentos e chaves *crossbar* passam a oferecer problemas de escalabilidade, já que aumentar o tamanho de uma destas soluções proporcionalmente à quantidade de núcleos resulta em uma alta latência de comunicação, resistência e roteamento do fio, entre outras. Por este motivo, as *Networks-on-Chip* (NoCs)

[16][17] têm sido apontadas como boas alternativas para aumentar o desempenho de processadores com dezenas ou centenas de núcleos. A Figura 2 ilustra uma NoC topologia mesh 3x3 composta por roteadores (*routers*) e *links* interconectando núcleos de processamento (*cores*). Estas redes possuem os mesmos princípios de redes de computadores, com roteadores, algoritmos de roteamento, controle de fluxo, etc. Como a disciplina de Redes de Computadores também costuma ser oferecida depois de Arquitetura de Computadores, inserir NoCs no ensino de arquitetura de processadores pode também ser um desafio que deve ser discutido.

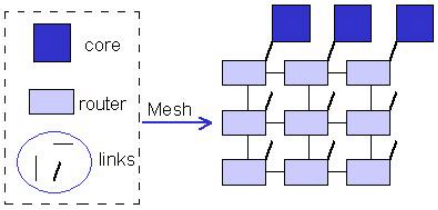

Figura 2. NoC topologia Mesh 3x3

Memória compartilhada é vista como uma boa alternativa, mas também pode ser um gargalo de comunicação. Deve ser levado em consideração *cache misses* além da própria interconexão de acesso. Considerando os limites das soluções de interconexão, algumas propostas apontam para NUCA (*Non Uniform Cache Architecture*) [18], similar ao conceito NUMA (*Non Uniform Memory Architecture*) [19]. Normalmente, este conceito de acesso não uniforme é apresentado em disciplinas de Processamento Paralelo, que também são ofertadas após Arquitetura de Computadores.

Seguindo no caminho do paralelismo, chegamos à Programação Paralela e como se deve programar um processador com vários núcleos (*many-core*). A discussão aponta para o ensino de programação paralela desde o início de um curso de graduação e sendo assim, facilitaria muito o ensino de arquiteturas paralelas e conseqüentemente tudo o que depende deste tipo de arquitetura. Mas talvez a principal dúvida se deve à definição do modelo [20][21]. Se o processador possui uma NoC e se existem *caches* compartilhadas e distribuídas, o modelo de programação paralelo deve ser híbrido? Isto nos leva a outro desafio associado ao ensino de arquiteturas que está associado à carga de trabalho.

O projeto de um processador depende do conhecimento da carga de trabalho típica que será executada (propósito geral ou específico). Na introdução deste artigo foi descrito sobre o suporte a múltiplas *threads*. O termo *Chip Multithreading* [3] tem sido usado principalmente pela SUN e define um processador que suporta várias *threads* através de vários núcleos. Portanto, qual o suporte em hardware (Figura 3) para múltiplas *threads* de aplicações diferentes? *Interleaved* (IMT), *Blocked* (BMT) ou *Simultaneous Multithreading*? Os núcleos são escalares ou superescalares? Estas respostas dependem da carga de trabalho.

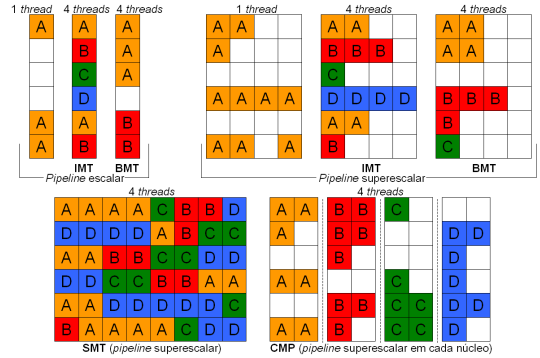

Figura 3. Suporte a múltiplas *threads*

Além dos temas apresentados, existem vários outros tais como: interrupção com NoC, concorrência entre processos, mapeamento de processos, migração de processos, memórias transacionais [22] e computação reconfigurável [23]. O certo é que uma única disciplina de Arquitetura de Computadores não é suficiente para comportar todos os problemas e desafios associados aos processadores *multi-core* e *many-core*. Além disso, os processadores *many-core* reforçarão a importância das disciplinas de arquiteturas de computadores em diferentes períodos dos cursos de Engenharia e Ciência da Computação, tanto na graduação quanto na pós-graduação.

A próxima seção apresentará o projeto de pesquisa usado como base para o auxílio na disciplina Introdução ao Processamento Paralelo e Distribuído da Pós-Graduação em Computação da Universidade Federal do Rio Grande do Sul. Esta disciplina é dividida em dois módulos básicos: Arquiteturas Paralelas e Programação Paralela.

3. Projeto de Pesquisa Base

Nesta seção é feita uma breve descrição do contexto onde se insere o projeto de pesquisa base deste artigo.

Vários projetos de arquiteturas de processadores ao longo de décadas vêm adotando técnicas tradicionais como *pipeline*, superescalaridade e *multithreading* para explorar o paralelismo de execução das aplicações e assim melhorar o tempo de resposta em computadores pessoais ou servidores.

Em um *pipeline* superescalar, além do processamento de instruções dividido em estágios é

feita uma completa sobreposição das instruções, utilizando, para isso, o aumento do número de unidades funcionais e técnicas para solucionar falsas dependências entre as instruções, dentre outras. Desta forma, os processadores superescalares são capazes de aumentar consideravelmente o desempenho na execução de cargas de trabalho com alto grau de paralelismo no nível de instruções.

O suporte a múltiplas *threads* é uma alternativa de exploração de paralelismo não mais no nível de instruções, mas no nível de fluxo de instruções (*threads*). Isto significa um aumento na vazão de *threads* (podendo mais de uma *thread* ser executada ao mesmo tempo) ao contrário da superescalaridade onde a vazão é de instruções de uma única *thread*. Diversas são as técnicas para exploração do paralelismo no nível de *threads*, sendo que a mais conhecida é a SMT (*Simultaneous Multithreading*) que é suportada por uma arquitetura superescalar (Figura 3).

Esta complexa abordagem de extração de paralelismo vem dando lugar a uma abordagem diferente. A fim de aumentar ainda mais o desempenho, e ainda algumas vezes diminuir a potência dissipada, o uso de processadores com múltiplos núcleos (*Chip Multiprocessors* - CMPs) vem sendo consolidada como uma boa alternativa para aumento do desempenho de computação.

Com vários núcleos, um processador passa a ter suporte nativo a várias *threads*, sendo os núcleos superescalares ou não. Ou seja, se o CMP possui 8 núcleos escalares, é possível ter 8 *threads* simultâneas. Nada impede que sejam utilizadas superescalaridade e SMT em cada núcleo, mas nestes casos há um aumento considerável na área do CMP. Em função desta nova abordagem de projeto, os processadores com múltiplos núcleos e que suportam múltiplas *threads* também são conhecidos como CMT ou *Chip Multithreading* [3].

Com o surgimento dos CMPs e CMTs algumas pesquisas e processadores comerciais adotaram o compartilhamento da memória *cache* de nível 2 como uma alternativa para aumentar o desempenho de aplicações paralelas utilizando o modelo de programação por memória compartilhada.

Porém, com as atuais inovações tecnológicas, onde cada vez mais indústrias estão passando a utilizar processadores com múltiplos núcleos de processamento, a hierarquia de memória *cache* a ser adotada para esses processadores multiprocessados, ainda é uma dúvida para os projetistas. Sendo que este questionamento é o ponto-chave abordado no projeto de pesquisa base aplicado em sala de aula (Seção 6).

Atualmente, os processadores do estado da arte [24][25][26] utilizam no primeiro nível de memória *cache*, a organização de uma *cache* por processador, sendo a *cache* separada para dados e instruções. Já no segundo nível de *cache*, alguns projetos utilizam o compartilhamento de uma memória *cache* por dois processadores, ou a *cache* isolada para cada processador, e em alguns casos, é utilizado até mesmo um terceiro nível na hierarquia de memória cache.

Desta maneira, nota-se que a hierarquia de memória *cache* a ser adotada em processadores multiprocessados ainda está em discussão, assim como o modelo de compartilhamento dessas memórias entre os núcleos. Isto demonstra a importância de trabalhos que avaliam mais fundo a relação entre processadores multiprocessados e hierarquias de memórias, além de compartilhamento de memórias *cache*.

Um resultado inicial já obtido [14] sobre o impacto das memórias *cache* em *chips multi-core* apresenta a avaliação de desempenho de um sistema modelado com 32 núcleos (Figura 4), onde são variadas as organizações de compartilhamentos de memória *cache*, e medido o desempenho para duas implementações de uma dada aplicação: uma com conjunto de dados contíguos e outra utilizando um conjunto de dados não contíguos. Foi concluído que existe uma forte relação entre tipo de programação adotada e hierarquia de memória do *chip*, para que se possa obter o melhor desempenho.

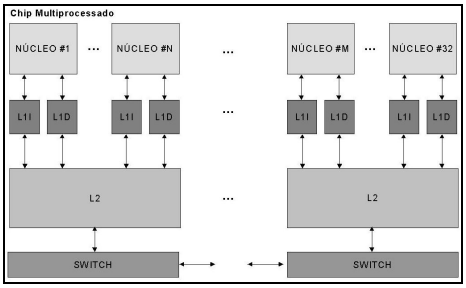

Figura 4. Arquitetura simulada [14]

O ambiente de simulação usado nesta pesquisa é baseado no Simics [27]. A seção seguinte descreve as principais características deste ambiente.

4. Características do Ambiente de Simulação Simics

O ambiente de simulação Simics 3.0 da Virtutech [27][28] é um simulador completo de sistema no nível de conjunto de instruções. Assim, os resultados de tempo de execução são fornecidos em instruções e ciclos. O número de ciclos é dado pelo número de instruções executadas mais os ciclos em espera gerados pelas latências de todos os componentes modelados.

Este simulador fornece possibilidade de modelagem e parametrização de componentes em nível de

arquitetura, podendo ser utilizado os módulos disponíveis no simulador ou então pode ser feito a modelagem de novos módulos. Além disso, o simulador oferece total suporte ao sistema operacional, sendo possível a execução de programas e sistemas operacionais completos sem necessidade de modificações.

Para simulação mais detalhada, alguns modos de operação estão disponíveis: *fast* (rápido); *stall* (lento) ou; mai (microarquitetura). Assim, de acordo com o modo utilizado é permitido observar informações a respeito de tempo, passos de execução, ciclos de execução, registradores em diferentes graus de complexidade, detalhamento e velocidades de simulação.

Para simulações considerando o subsistema de memória *cache*, que é o foco do projeto base, o simulador possui suporte nativo a dois modelos de *cache*: *g-cached* e *g-cached-ooo*. O modelo *g-cached* fornece todas as condições para modelagem de uma *cache* ligada a um processador executando as instruções em ordem e fornecendo relatórios sobre as atividades realizadas. Já o modelo *g-cached-ooo* além das funcionalidades apresentadas pela *g-cache*, provê ainda, a possibilidade de ser utilizada em simulações de processamento fora de ordem.

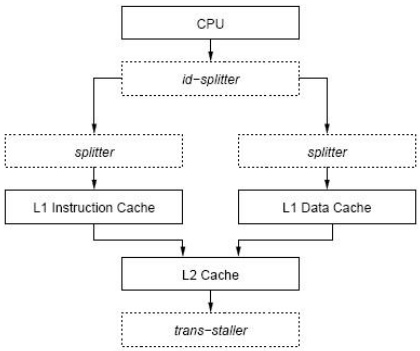

Figura 5. Modelo *g-cache* [28]

Mesmo o modelo *g-cache-ooo* sendo o mais completo para uma simulação de *cache*, este modo não foi utilizado em [14], devido a restrições de tempo, pois torna inviável a execução de aplicações completas. O funcionamento básico do modelo *g-cache* é ilustrado na Figura 5, onde podemos ver os seguintes componentes:

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

Além da modelagem detalhada do subsistema de memórias *cache*, o simulador oferece outras funcionalidades como criação de traços de execução, detalhamento e parametrização do processador, abstrações de *chip multiprocessor* e também permite a execução de máquinas distribuídas, que podem ser utilizadas em conjunto simulando um *cluster*. Dessa maneira esse ambiente de simulação apresenta boas características funcionais e didáticas.

5. Metodologia Utilizada

A metodologia foi baseada em um estudo inicial de quais seriam as principais atividades que poderiam ser desenvolvidas pelos alunos e qual seria o ambiente para estas atividades.

A escolha de um ambiente de simulação completo, que possibilite o projeto de um processador *multi-core*, tal como instalar sistemas operacionais e executar programas reais é o mais adequado para testar um projeto de arquitetura.

Neste momento se fez uma relação entre o projeto de pesquisa base (Seção 3) e a disciplina. Desta forma, foi possível aproveitar o conhecimento gerado, aplicando na disciplina um ambiente de simulação com características que pudessem suportar as necessidades de ensino.

Portanto, o ambiente de projeto e simulação de arquiteturas *multi-core* adotado foi o Simics. Os programas desenvolvidos pelos alunos devem ser executados e avaliados neste ambiente para testar a arquitetura projetada.

Como conseqüência a metodologia foi dividida em três métodos básicos:

- Introdução ao Simics.
- Método de trabalho dos alunos.
- Avaliação da disciplina.

A introdução ao ambiente Simics se fez com base em um método dividido nas seguintes atividades:

1. Uma aula teórica com as principais características e funcionalidades.
2. Uma aula prática para desenvolvimento de um projeto introdutório e básico.
3. Materiais de auxílio disponibilizados através da *web*.
4. Monitoria em função da demanda de dúvidas dos alunos.

O método de trabalho dos alunos está baseado nas seguintes etapas:

5. Estudo e projeto de uma arquitetura de processador *multi-core*.
6. Estudo e desenvolvimento de um programa paralelo.

7. Avaliação de desempenho do programa paralelo executado na arquitetura *multi-core* projetada.

O método de avaliação da disciplina é dividido em duas partes conforme módulos principais da disciplina:

8. Arquiteturas Paralelas: avaliação do projeto da arquitetura do processador.
9. Programação Paralela: avaliação do programa paralelo executado.

As etapas descritas podem ser ilustradas através do cronograma da Tabela 1. No mês de março a disciplina é iniciada e os conceitos principais são introduzidos. Os alunos passam a desenvolver o trabalho em duas etapas, finalizando com uma avaliação de desempenho.

Tabela 1. Cronograma do trabalho

Etapas	Março	Abril	Maio	Junho	Julho
1		**X**			
2		**X**			
3		**X**			
4		**X**	**X**	**X**	**X**
5		**X**	**X**		
6			**X**	**X**	
7				**X**	**X**
8					
9			**X**		**X**

A próxima seção apresenta análises dos trabalhos apresentados pelos alunos.

6. Análises e Discussão

Nesta seção são apresentados exemplos de três trabalhos apresentados pelos alunos na disciplina. Os alunos utilizaram arquiteturas de processadores *dual-core* e *quad-core* conhecidas, sem exercitar grandes mudanças que representassem alterações e tentativas de avaliação do desempenho de uma nova proposta de arquitetura. Nos trabalhos foram utilizados sistemas operacionais Linux e compiladores GNU GCC. As configurações principais são descritas a seguir:

O primeiro trabalho modelou um processador *quad-core* AMD Opteron de 64 bits. As memórias *cache* L1, locais a cada núcleo, possuem tamanho de 128kB e não são compartilhadas. A memória *cache* L2 possui tamanho de 512kB e é compartilhada entre todos os núcleos. Os ciclos de penalidade para cada memória são: 3 ciclos para *cache* L1, 10 ciclos para *cache* L2 e 60 ciclos para a memória principal. Além do modelo da arquitetura foi desenvolvido um programa em OpenMP para paralelizar um Decodificador MP3.

No segundo trabalho foi modelado um processador *dual-core* Pentium 4 de 2GHz. Cada núcleo possui uma *cache* L1 de 64kB e os dois núcleos compartilham uma *cache* L2 de 2MB. Foi utilizado uma latência de 3 ciclos para a *cache* L1 e 14 ciclos para a *cache* L2. Neste trabalho o aluno estudou o Intel Core Duo, núcleo Yonah, para especificar as latências do projeto. A aplicação avaliada nesta arquitetura é um Decodificador MPEG paralelizado através de MPI.

O terceiro trabalho é focado na arquitetura UltraSparc II. O aluno modelou um processador com quatro núcleos, mantendo também a tendência mostrada nos trabalhos anteriores de compartilhamento de *cache* L2 e *caches* L1 locais e privadas. No entanto o aluno desenvolveu dois modelos: no primeiro a *cache* L2 é compartilhada entre todos os quatro núcleos, no segundo a *cache* L2 é dividida em duas, sendo que apenas dois núcleos compartilham cada uma delas. No primeiro projeto a *cache* L1 possui 32kB e a *cache* L2 possui 1024kB. No segundo projeto a *cache* L1 permanece com o mesmo tamanho, mas as *caches* L2 possuem agora 512kB cada uma. Neste trabalho o aluno desenvolveu e avaliou um programa escrito em MPI para convolução de imagens.

As principais métricas, para avaliação de desempenho, utilizadas pelos alunos foram as seguintes:

- *Cache miss*
- Tempo de execução
- *Speedup* (ganho)

Neste artigo não será feita uma avaliação dos resultados obtidos pelos alunos em cada simulação, mas uma análise da metodologia utilizada em face aos trabalhos apresentados pelos alunos. Portanto, seguem algumas análises que fazem relação à aplicação da metodologia, aos tópicos principais da disciplina e aos trabalhos dos alunos.

A introdução ao Simics e o suporte dado aos alunos se mostrou eficiente. No entanto, percebeu-se a necessidade de introduzir o ambiente de simulação logo no início da disciplina, mesmo antes do ensino teórico sobre arquiteturas paralelas. Por isso, foi necessária a apresentação da ferramenta CACTI [29] para uma correta modelagem de latências de memória. Neste caso, as atividades de 1 a 5 seriam iniciadas em março (Tabela 1). A atividade de monitoria deve ser incentivada, sem esperar que os alunos sintam necessidade de tirar dúvidas. Isto aumenta a probabilidade de que os alunos iniciem os trabalhos mais cedo.

O projeto da arquitetura dos processadores pode ser mais bem elaborado pelos alunos nos próximos semestres. Isto é conseqüência do fato de que a proposta deixa de ser uma novidade fazendo com que os próximos alunos tenham uma real noção do grau de dificuldade e do tempo necessário para realização do projeto. Sendo assim, trabalhos com o grau de dificuldade e/ou profundidade já apresentados devem

ser melhorados. Portanto, será solicitado e é de se esperar, que os próximos trabalhos sejam mais complexos.

O mesmo pode ser aplicado à programação paralela. Uma vez que os alunos conseguem elaborar projetos de arquitetura mais avançada, é possível exercitar programas paralelos mais otimizados em função de arquiteturas não triviais e que fogem ao tradicional que é apresentado pelo mercado.

De um modo geral, a metodologia cumpriu com o esperado. Os alunos projetaram e desenvolveram uma arquitetura de processador *multi-core*, simularam esta arquitetura em um ambiente completo com sistema operacional, rede e programas reais. Além disso, desenvolveram e avaliaram um programa paralelo para a arquitetura proposta.

A seção seguinte descreve abordagens de ensino relacionadas, que utilizaram experiências em pesquisa e indústria para auxiliar atividades de sala de aula.

7. Abordagens de Ensino Relacionadas

Neste artigo foi apresentado o uso de um projeto de pesquisa como forma de auxiliar o ensino em arquiteturas de processadores *multi-core*. Relacionadas a este tipo de abordagem, não necessariamente processadores *multi-core*, é possível descrever brevemente alguns trabalhos que levam para sala de aula a experiência obtida em empresas e também em pesquisas acadêmicas.

Em [30] o autor descreve como utilizar a experiência obtida em empresas como forma de motivar os alunos a desenvolver melhores projetos relacionados aos trabalhos das disciplinas. O foco principal é passar o modelo de equipe e ambiente de trabalho da indústria para desenvolvimento dos trabalhos. Esta abordagem foi aplicada em disciplinas de graduação e pós-graduação em arquitetura de computadores.

Outra abordagem interessante é descrita em [31]. Neste artigo os autores apresentam o simulador desenvolvido para o modelo de arquitetura do processador SPARC, suas características principais e potencialidades. Um dos objetivos é mostrar sua aplicação no ensino de arquitetura de computadores citando exemplos como *multithreading*, avaliação de desempenho e interrupções que podem ser analisados e avaliados pelos alunos.

O uso de um simulador para auxílio ao ensino de computação reconfigurável também é apresentado em [32]. Neste artigo os autores ressaltam a importância do tema, sua inclusão nos currículos tradicionais das disciplinas de arquitetura de computadores e também a utilização de simuladores. O simulador BobSim é apresentado em sua segunda versão e tem sido usado para ensinar computação reconfigurável em memórias *cache*.

Na mesma linha de memórias *cache*, trabalhos de pesquisas [33] se mostram promissores como incentivo aos alunos da disciplina de Arquitetura de Computadores no estudo, projeto e avaliação de memórias *cache*. Nesta linha foi possível constatar em outros eventos WEAC que os alunos têm desenvolvido novos simuladores com novas funcionalidades, o que melhora consideravelmente o aprendizado, aumentando ao grau de profundidade das turmas seguintes.

8. Conclusões

Este artigo fez uma descrição das atividades de ensino relacionadas às arquiteturas *multi-core* na disciplina Introdução ao Processamento Paralelo e Distribuído. Neste contexto, foi utilizada a experiência de um projeto de pesquisa como forma de auxiliar o projeto e avaliação destas arquiteturas de processadores. O ambiente de simulação Simics foi utilizado pelos alunos para projeto da arquitetura *multi-core* e serviu como plataforma de *hardware* para execução de programas paralelos desenvolvidos em sala de aula como atividade prática.

Os trabalhos atingiram o objetivo inicial de ensino e atividade prática relacionada à arquitetura de processadores *multi-core*. Segundo análise da metodologia, as aulas e monitorias relativas ao trabalho serão iniciadas mais cedo nos próximos semestres. Além disso, é esperado um aumento da complexidade e profundidade dos projetos em função da experiência gradativa que será adquirida ao longo da utilização.

Como metas futuras, está a avaliação do uso da metodologia descrita através das seguintes etapas:

- Aprendizagem de novos conceitos e técnicas.
- Aplicação de um questionário sobre o trabalho realizado.
- Uma comparação evolutiva dos trabalhos dos alunos.

Referências

[1] K. Olukotun, and L. Hammond, “The Future of Microprocessors”, ACM Queue, Vol. 3, Issue 7, pp. 26-29, September 2005.

[2] R. Kumar, et al. “Heterogeneous Chip Multiprocessor”, IEEE Computer, Vol. 38, Issue 11, pp. 32-38, 2005.

[3] L. Spracklen, S. G. Abraham, “Chip Multithreading: Opportunities and Challenges”, IEEE International Symposium on High-Performance Computer Architecture, pp. 248-252, February 2005.

[4] Dongarra, J., T. Sterling, H. Simon, and E. Strohmaier, “High-Performance Computing: Clusters, Constellations, MPPs, and Future Directions”, Computing in Science & Engineering, pp. 51-59, March/April 2005.

[5] Buyya, R., High Performance Cluster Computing – Architectures and Systems, Volume 1, Prentice Hall, 1999.

[6] L. Kontothanassis, et al., “Shared Memory Computing on Clusters with Symmetric Multiprocessors and System Area Networks”, ACM Transactions on Computer Systems, Vol. 23, No 3, pp. 301-335, August 2005.

[7] Comer, D. E., “Network Systems Design Using Network Processors”, Prentice Hall, 2003.

[8] Shah, N., “Understanding network processors”, Master’s thesis, University of California, Berkeley, USA, 2001.

[9] M. L. Pilla, B. Childers, F. M. G. França, A. T. Costa, P. O. A. Navaux, “Limits for a feasible speculative trace reuse implementation” International Journal of High Performance Systems Architecture, v. 1, p. 69-76, 2007.

[10] T. Ungerer, B. Robic and J. Silc, “A Survey of Processors with Explicit Multithreading”, ACM Computing Surveys, Volume 35, Issue 1, pp. 29-63, March 2003.

[11] S. Chaudhry, et al., “High-performance throughput computing”, IEEE MICRO, Vol. 25, Issue 3, pp.32-45, 2005.

[12] P. Kongetira, “Niagara: a 32-way multithreaded Sparc processor”, IEEE MICRO, Volume 25, Issue 2, pp.21-29, March-April 2005.

[13] J. D. Owens, et al., “GPU Computing”, Proceedings of the IEEE, pp.879-899, 2008.

[14] M. A. Z. Alves, et al., “Influência do compartilhamento de cache l2 em um chip multiprocessado sob cargas de trabalho com conjuntos de dados contíguos e não contíguos”, VIII Workshop em Sistemas Computacionais de Alto Desempenho, pp.27-34, 2007.

[15] M. K. Ferreira, H. C. Freitas, P. O. A. Navaux, “From Intel VT-x to MIPS: An ArchC-based Model to Understanding the Hardware Virtualization Support”, Workshop on Computer Architecture Education, em conjunto com o ISCA 2008, Beijing, pp.9-15, 2008.

[16] R. Kumar, V. Zyuban and D. M. Tullsen, “Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling”, ACM/IEEE International Symposium on Computer Architecture, pp. 408-419, 2005.

[17] T. Bjerregaard and S. Mahadevan, “A Survey of Research and Practices of Network-on-Chip”, ACM Computing Surveys, Vol. 38, No 1, pp. 1-51, March 2006.

[18] C. Kim, D. Burger, S. W. Keckler, “An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches”, 10th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.211-222, 2002.

[19] A. S. Carissimi, et al., “Aspectos de Programação Paralela em Máquinas NUMA”, Minicurso do Workshop em Sistemas Computacionais de Alto Desempenho, 2007.

[20] Dongarra, J., I. Foster, G. C. Fox, W. Gropp, K. Kennedy, L. Torczon and A. White, “The Sourcebook of Parallel Computing”, Morgan Kaufmann, 2003.

[21] Akhter, S., Roberts, J., “Multi-Core Programming, Increasing Performance through Software Multi-threading”, Intel Press, 336p., April 2006.

[22] M. Herlihy, J. E. B. Moss, “Transactional Memory: Architectural Support for Lock-Free Data Structures”, IEEE International Symposium on Computer Architecture, pp.289-300, 1993.

[23] K. Compton and S. Hauck, “Reconfigurable Computing: A Survey of Systems and Software”, ACM Computing Surveys, v. 34, n. 2, p. 171-210, June 2002.

[24] T. Takayanagi, et al., “A Dual-Core 64-bit UltraSPARC Microprocessor for Dense Server Applications”, IEEE Journal of Solid-State Circuits, Vol. 40, No 1, January 2005.

[25] AMD, Inc., “The AMD Opteron Processor for Servers and Workstations”, AMD Opteron Processor Overview, 2005.

[26] C. Mcnairy, R. Bhatia, “Montecito: a dual-core, dual-thread itanium processor”, IEEE Micro, v.16, 2005.

[27] P. S. Magnusson, et al., “Simics: a full system simulation platform”, IEEE Computer Micro, 2002.

[28] Virtutech, Simics 3.0: User Guide for Unix. Virtutech - Simics, 2007, URL: http://www.simics.net, Acessado em 02/10/2008.

[29] S. Thoziyoor, et al., “A comprehensive memory m odeling tool and its application to the design and analysis of future memory hierarchies”, IEEE International Symposium on Computer Architecture, pp.51–62, 2008.

[30] G. Paez-Monzon, “Teaching and Learning by Applying Industry Environments in the Classroom”, Workshop on Computer Architecture Education, em conjunto com o ISCA 2008, Beijing, pp.24-29, 2008.

[31] S. Chen, J. Xiong, “Explore OpenSPARC by Full System Simulator SAM - SPARC Architecture Model”, Workshop on Computer Architecture Education, em conjunto com o ISCA 2008, Beijing, pp.30-34, 2008.

[32] R. B. Kerr Jr., E. T. Midorikawa, “Introdução da Computação Reconfigurável e o Uso de Ferramentas no Ensino de Arquitetura de Computadores”, Workshop sobre Educação em Arquitetura de Computadores, em conjunto com o SBAC-PAD 2007, Gramado, pp.15-22, 2007.

[33] L. M. N. Coutinho, J. L. D. Mendes, C. A. P. S. Martins, “Web memory hierarchy learning and research environment”, Workshop on Computer Architecture Education, em conjunto com o ISCA 2006, Boston, Article No 5, 2006.