Impacto do *Prefetcher* na Precisão de Simulações de Arquiteturas Paralelas*

Valéria S. Girelli[1], Francis B. Moreira[1], Matheus S. Serpa[1] e Philippe O. A. Navaux[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970, Porto Alegre – RS – Brasil

{vsgirelli, fbmoreira, msserpa, navaux}@inf.ufrgs.br

Abstract. *In computer architecture research, the use of simulators is predominant, with a wide variety of approaches and implementations. However, validation studies lack a detailed analysis of parallel architecture simulators that support High Performance Computing (HPC) workloads. This study aims to analyze the impact of the prefetcher in the parallel simulation accuracy of ZSim, a parallel architecture simulator. We could observe that, due to the lack of a prefetcher model, the memory hierarchy statistics show inaccurate behavior, with error rates reaching 2,600%.*

Resumo. *Em arquitetura de computadores, o uso de simuladores é predominante em todos os grupos de pesquisa, com uma ampla variedade de abordagens e implementações.No entanto, falta na literatura uma análise detalhada de simuladores de arquiteturas paralelas que suportem workloads de Computação de Alto Desempenho (High Performance Computing - HPC). Este trabalho busca analisar o impacto do prefetcher na precisão da simulação paralela realizada pelo ZSim, um simulador de arquiteturas paralelas. Observamos que, devido à falta de modelagem de prefetcher, as estatísticas da hierarquia de memória apresentam comportamentos imprecisos, com erros de até 2.600%.*

1. Introdução

Na pesquisa de arquiteturas de computadores, a complexidade e alto custo de manufatura inviabilizam a realização de experimentos e análises com implementações físicas. Simuladores de arquitetura de processadores são amplamente utilizados no desenvolvimento e análise de novas ideias arquiteturais. Deste modo, simulação é considerada a principal forma de se implementar e avaliar uma nova ideia neste campo de pesquisa [Demme 2014].

Em um sistema de computação de alto desempenho (*High Performance Computing - HPC*), encontram-se inúmeros outros problemas além dos já intrínsecos à arquitetura. Desenvolver e analisar novas implementações arquiteturais que minimizem o impacto de problemas decorrentes do paralelismo requer simuladores de arquiteturas *multicore* e *manycore* que trabalhem com *workloads* paralelos e suas particularidades. Por exemplo, sistemas com dezenas de *cores* evidenciam problemas como a interferência entre diferentes *threads* e os custos de comunicação entre elas. A interação entre *threads* se dá principalmente por meio de memória compartilhada, com várias *threads* acessando

*Este trabalho foi parcialmente financiado pelo projeto Petrobras 2016/00133-9, pela CAPES, CNPq e FAPERGS.

os mesmos endereços de memória. Com isso, é necessário manter a coerência dos dados na hierarquia de memória por meio de protocolos de coerência de *cache*, que garantem a consistência dos dados por meio de um conjunto de requisições e mensagens.

A inserção de *prefetching* em uma arquitetura traz uma nova complexidade ao comportamento da hierarquia de memória no processador. *Prefetchers* detectam padrões de acesso de cada *core* e geram acessos à memória de forma especulativa, a fim de mitigar o efeito da latência dos acessos aos dados em memória. Com isso, simulações devem considerar informações muitas vezes descartadas por protocolos de coerência de *cache*, como conteúdos de linha de *cache* [Huberty et al. 2018], valores de registradores [Nesbit and Smith 2004], ou endereços físicos [Bakhshalipour et al. 2019b].

Neste trabalho, utilizamos o ZSim [Sanchez and Kozyrakis 2013], um simulador de arquiteturas paralelas que implementa uma metodologia de simulação também paralela, diminuindo o tempo de simulação. Devido à forma como o ZSim mantém a precisão na **ordem** da simulação de acessos à *cache*, isso impossibilita a simulação correta de *prefetchers*. Deste modo, neste trabalho buscamos analisar como a falta de um modelo de *prefetcher* impacta a precisão da simulação do ZSim conforme varia-se o número de threads e quais as consequências desse impacto.

As próximas seções estão organizadas da seguinte forma. Na Seção 2 detalhamos os trabalhos relacionados. Na seção 3 apresentamos o conceito de *prefetching* e a implementação do simulador utilizado. A Seção 4 apresenta as configurações dos experimentos e simulações. A Seção 5 exibe e discute os experimentos realizados, e por fim a Seção 6 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Devido à propriedade intelectual e à falta de informação explícita que empresas de *hardware* empregam para evitar competição, é difícil obter uma simulação ideal que apresente de forma correta todas as características do processador e de sua arquitetura. Portanto, procura-se em um simulador um erro relativo baixo e sensibilidade em relação ao que se deseja testar [Eeckhout 2010]. Por exemplo, um simulador que não sofre perda de desempenho ao reduzirmos a quantidade de *cache* para programas que exigem muita memória não modela a *cache* corretamente ou sofre de gargalos em outras partes do modelo, impedindo que a sensibilidade da aplicação aos acessos à memória transpareça. *Microbenchmarks* podem ser usados para diminuir o erro em estruturas básicas da arquitetura, e como forma de engenharia reversa de características arquiteturais [Fog 2012].

Em [Desikan et al. 2001], estudou-se a dificuldade de se obter informações precisas e a relevância disto ao validar o simulador SimpleScalar [Austin et al. 2002]. Ao obter mais informações sobre o modelo do processador Alpha 21264, os autores conseguiram reduzir o erro experimental de *microbenchmarks* de 19,5% para 2%. Já com a carga de trabalho SPEC-CPU 2000, o erro experimental médio foi de 36,7% para 18,2%. Os autores então mostraram como os recursos achados geram gargalos em partes diferentes do sistema do que previamente modelado em vários artigos, invalidando ideias que apresentavam ganhos de performance onde não havia gargalo.

Desde então, vários simuladores foram criados com a intenção de propor juntamente uma validação. Dentre eles, o ZSim foi selecionado para um estudo aprofundado devido à sua velocidade e precisão, características apresentadas tanto em sua validação [Sanchez 2016] quanto no estudo de [Akram and Sawalha 2019]. Na pesquisa de Akram, os simuladores gem5 [Binkert et al. 2011], Multi2Sim [Ubal et al. 2012],

MARSSx86 [Patel et al. 2011], PTLsim [Yourst 2007], Sniper [Carlson et al. 2011], e ZSim [Sanchez and Kozyrakis 2013] são avaliados. Após uma caracterização minuciosa dos simuladores, foram executadas cargas de trabalho de um *core* e de múltiplos *cores* em cada simulador. Os resultados foram comparados com a arquitetura Haswell da Intel [Hammarlund et al. 2014]. Os autores então destacaram as fontes de erro dos simuladores, sua sensitividade a diferentes parâmetros da arquitetura, e seu erro relativo. Assim, concluíram que a falta de validação de certos simuladores, por mais populares que sejam (e.g. gem5), leva a uma precisão baixa e pode tornar experimentos inválidos devido às conclusões errôneas. No entanto, o trabalho de Akram não utiliza cargas de trabalho *multi-thread* para verificar o erro relativo ao número de *threads*.

3. Motivação

Nesta Seção, apresentamos o conceito de *prefetching* e a abordagem empregada pelo ZSim em sua simulação paralela.

3.1. Prefetcher

A memória disponível para um processador está organizada em diferentes níveis. Níveis privados e mais próximos do processador possuem menor capacidade de armazenamento e acesso de dados mais eficiente. Quando um processador faz uma requisição à memória, esta requisição é feita ao primeiro nível da hierarquia de memória interna do processador, a memória *cache* de dados de primeiro nível (L1). Esta memória é relativamente pequena (32 *kilobytes*) e de baixa latência (4 ciclos do processador). Caso uma cópia do dado requisitado não se encontre na *cache* de dados L1, a mesma encaminha a requisição ao próximo nível de memória *cache*, que repete o mesmo procedimento. O último nível de memória *cache* (*Last Level Cache - LLC*), por sua vez, encaminha a requisição para a memória principal. Caso esta não tenha o dado, ocorre uma falta de página e a página precisa ser buscada em um dispositivo armazenamento secundário. Desse modo, deseja-se que os níveis de *cache* mais próximos do processador possuam os dados necessários para o momento da execução de cada requisição à memória.

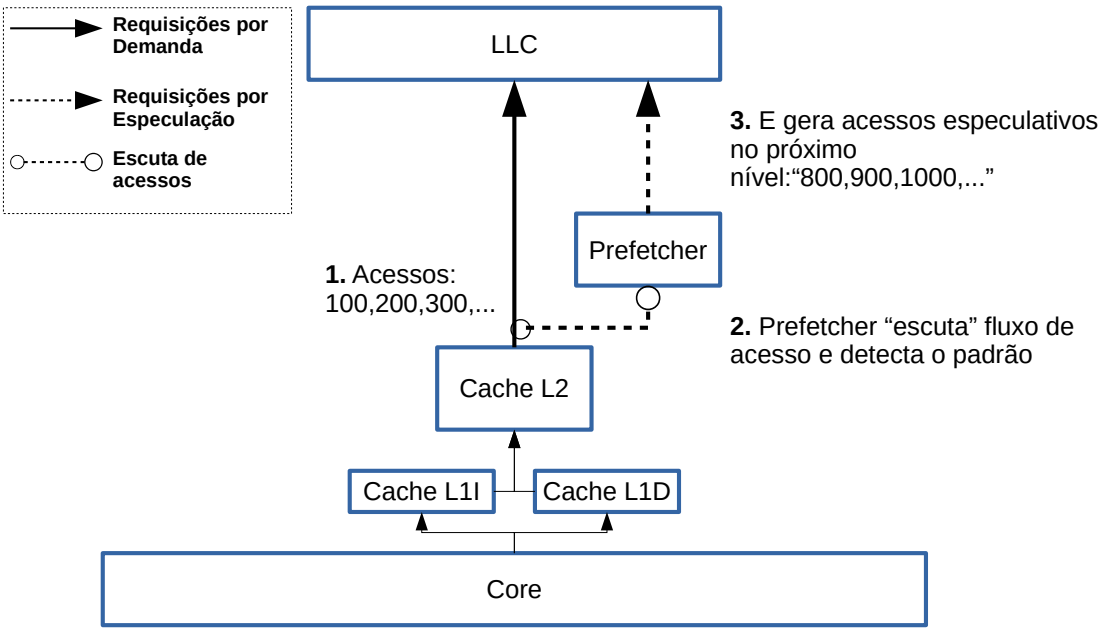

Figura 1. Abstração do funcionamento de um *prefetcher*.

Para reduzir a latência de acesso aos dados, criou-se o *prefetcher*. Com base no padrão dos acessos gerados pelo processador, especula-se qual seria o próximo endereço a ser requisitado e busca-se o bloco de dados antecipadamente. Deste modo, quando o bloco de dados for requisitado, o mesmo já estará em *caches* mais próximas ao processador. Alguns padrões observados são o *stride* [Chen and Baer 1995] e o *stream* [Le et al. 2007].

A Figura 1 exibe um exemplo do funcionamento do *prefetcher* identificando um padrão de acesso *strided*. O segundo nível de *cache* (L2) encaminha requisições para a LLC (indicado na Figura 1 como o evento **1**). O *prefetcher*, por sua vez, intercepta essas requisições (**2**) e identifica o padrão de acesso sendo gerado. Com base no padrão identificado, acessos especulativos à LLC são realizados (**3**). Estes acessos são vistos como requisições normais realizadas pela L2, então a LLC encaminha esses dados à L2. Assim, quando os dados requisitados por *prefetch* forem necessários, eles já estarão em níveis de *cache* mais próximos do processador. *Prefetchers* são predominantes em arquiteturas atuais como uma das formas de mitigar o gargalo de acesso à memória principal [Bakhshalipour et al. 2019a].

3.2. ZSim

O ZSim implementa a simulação em um método de duas fases chamadas *Bound* e *Weave*. Na fase *Bound* são simulados alguns milhares de ciclos, ignorando-se contenção e usando latências mínimas para todos acessos à memória. Registra-se um traço dos acessos à memória, incluindo quais *caches* foram acessadas, invalidações, etc. Na fase *Weave* é feita a simulação paralela orientada aos eventos do intervalo registrado para determinar as latências reais dos acessos. Como as interações entre os acessos à memória já foram identificadas na primeira fase, esta simulação de *timing* dos acessos pode ser feita de forma eficiente, mantendo-se uma alta precisão.

Os autores observaram que, em um intervalo de alguns milhares de ciclos, a maior parte dos acessos concorrentes entre diferentes *cores* acontece a linhas de *cache* não relacionadas. Portanto, simular estes acessos não relacionados fora de ordem ignorando contenções e, posteriormente, simulá-los em ordem respeitando restrições temporais, é equivalente a simulá-los completamente em ordem.

No entanto, quando acessos estão relacionados, *i.e.*, acessam a mesma linha de *cache*, é necessário manter a coerência das diferentes cópias do dado nos diferentes *cores*. Um exemplo disso é quando um *core* demanda acesso exclusivo a uma linha de *cache* compartilhada para realizar uma escrita na mesma, fazendo com que o protocolo de coerência de *cache* invalide as demais cópias da linha na hierarquia de *cache* do sistema. No entanto, por serem consideradas raras, a **ordem** dessas interações não é modelada pelo ZSim, o que pode resultar na alteração do caminho desses dados na hierarquia de *cache*. A alteração do caminho dos dados na *cache* pode impactar no número de ciclos, de *misses*, e de mensagens de coerência observados na simulação. Além disso, a geração e os caminhos das requisições feitas por *prefetches* também podem sofrer alterações, o que impede a modelagem de *prefetcher* neste modelo.

Em sua validação, o ZSim utiliza o *benchmark* PARSEC [Bienia et al. 2008] e mostra apenas que o *speed-up* é próximo ao obtido com execuções reais ao variar o número de *threads*. Deste modo, uma vez que o *prefetcher* não é simulado, buscamos avaliar a precisão do método de simulação *Bound and Weave* utilizado para simular paralelamente múltiplas estruturas e *threads*.

4. Ambiente de Experimentos

Nesta Seção são especificadas as configurações dos ambientes usados para a realização de nossos experimentos.

4.1. Configurações de Processador

A ferramenta *perf* [De Melo 2010] foi usada para obter os valores dos contadores de *hardware* para todas as execuções reais. Em cada execução real avalia-se um contador, evitando-se assim agregações ou aproximações da ferramenta que ocorrem ao avaliar múltiplas estatísticas na mesma execução. O desempenho e estatísticas da execução real foram avaliados através de 10 execuções de cada estatística, para cada *benchmark*, em máquinas *draco*, todas idênticas, pertencentes ao *cluster* do Parque Computacional de Alto Desempenho (PCAD)[1], pertencente ao GPPD - Grupo de Processamento Paralelo e Distribuído. Para as simulações foram utilizadas configurações que se aproximam da máquina real, uma Intel(R) Xeon(R) CPU E5-2640 v2, de arquitetura Ivy Bridge [James 2012], respeitando-se as limitações do simulador. Cada simulação foi executada apenas uma vez, e todas as estatísticas são extraídas desta mesma simulação.

Neste artigo exibimos uma seleção de estatísticas, onde são ilustrados problemas na modelagem de hierarquia de memória do ZSim. Escolhemos as estatísticas de ciclos de execução, acessos à memória *cache* de dados L1, e acessos ao último nível de memória *cache*. As estatísticas de ciclos ilustram o maior número de ciclos levado pelas *threads* paralelas, efetivamente representando o tempo que a aplicação levou para executar. As estatísticas de acessos à memória *cache* L1 representam uma média entre as *caches* de todas as *threads*, onde podemos notar as estruturas de dados sendo divididas entre as *threads* conforme aumentamos o paralelismo da aplicação. O último nível de *cache* é dividido em 8 bancos (um por *core*) conectados por um anel bidirecional. Um *core* acessa todos os bancos, pois os endereços de memória são divididos entre os bancos por meio de uma função de *hash*. Assim, as estatísticas da LLC contam os acessos a todos os bancos. O desvio padrão dos dados relativos às 10 execuções reais das aplicações não é apresentado nos gráficos por ser menor que 1%.

4.2. Configurações de *benchmark*

Para a avaliação de escalabilidade do erro relativo ao número de *threads*, foi usado o *benchmark Numerical Aerodynamic Simulation Parallel Benchmarks* (NPB) [Jin et al. 1999]. Este *benchmark* é composto por dez aplicações paralelas, das quais nove foram usadas. As aplicações utilizadas neste trabalho são:

- **CG**: *Conjugate Gradient* utiliza um método de gradiente conjugado para aproximar o valor do menor autovalor de uma matriz simétrica positiva definida, grande e esparsa. Este *kernel* testa comunicação de longa distância irregular, empregando multiplicação de matriz por vetor sem estrutura.
- **EP**: *Embarrassingly Parallel* resolve o problema de gerar pares de gaussianas e tabular seus valores em quadrados *annuli* sucessivos. Ao contrário do resto da lista, este *kernel* praticamente não possui comunicação, estressando apenas o desempenho de operações de ponto flutuante.
- **FT**: *Fourier Transform* resolve o problema da transformada de Fourier tridimensional de forma paralela usando o algoritmo de transformada rápida de

[1] http://gppd-hpc.inf.ufrgs.br/

Tabela 1. Configurações do processador real e do processador simulado.

	Real	**ZSim**
Frequência do Processador	2 - 2.5Ghz	2Ghz
Número de *Cores*	8	8
Número de Estágios	18	14
Tamanho da Linha de *cache*	64 B	64 B
Cache de Dados L1	8- *way* 32KB	8- *way* 32KB
Latência	4	4
Cache de Instruções L1	8- *way* 32KB	8- *way* 32KB
Latência	4	3
Cache Unificada L2	8- *way* 256KB	8- *way* 256KB
Latência	ca. 11	12
Cache de Último Nível L3	20- *way* 20MB	16- *way* 16MB
Latência	ca. 28	30
Prefetchers	*L1 IP Stride* *Adjacent Line prefetcher* *L2 DCU Stream prefetcher*	

Fourier (FFT). Testa rigorosamente a comunicação de alta distância, ocorrendo comunicação de todos *cores* com todos *cores*.

- **IS**: *Integer Sort* resolve o problema de ordenar inteiros usando *bucket sort*. Possui acessos randômicos à memória, e testa tanto o desempenho de operações com inteiros quanto a comunicação.
- **MG**: *Multigrid* resolve um problema simplificado de cálculo *multigrid*, testando comunicações de curta e longa distâncias. Requer comunicação de longa distância altamente estruturada.
- **UA**: *Unstructured Adaptive Mesh* resolve um problema estilizado de transferência de calor em um domínio cúbico, discretizado em uma malha sem estrutura refinada adaptativamente.
- **BT**: *Block Tridiagonal* resolve um problema sintético de CFD com múltiplos sistemas de equações de blocos tridiagonais 5x5 com diagonais não-dominantes. Tem paralelismo mais limitado que os outros CFD (LU e SP).
- **LU**: *Lower Upper Gauss Seidel Solver* resolve um problema sintético de CFD ao resolver um sistema de equações lineares triangulares esparsas, com blocos de 5x5. Envolve dependências de dados globais, com várias comunicações curtas.
- **SP**: *Scalar Pentadiagonal* resolve sistemas de equações escalares pentadiagonais com diagonais não-dominantes. Assim como o LU, envolve dependências de dados globais, mas com comunicações mais infrequentes e mais longas.

A aplicação **DC** ("Data Cube") não foi usado devido à grande utilização de I/O, o qual não é modelado pelos simuladores.

5. Análise de Comportamento da Hierarquia de Memória

Uma vez que o ZSim não simula o sistema de *prefetching*, poderiam ocorrer alterações no número de ciclos, *misses*, e acessos à LLC. Analisamos, portanto, o número de ciclos e de acessos à *cache* de dados e à LLC observados na simulação e na execução real com e sem a atuação do *prefetch*.

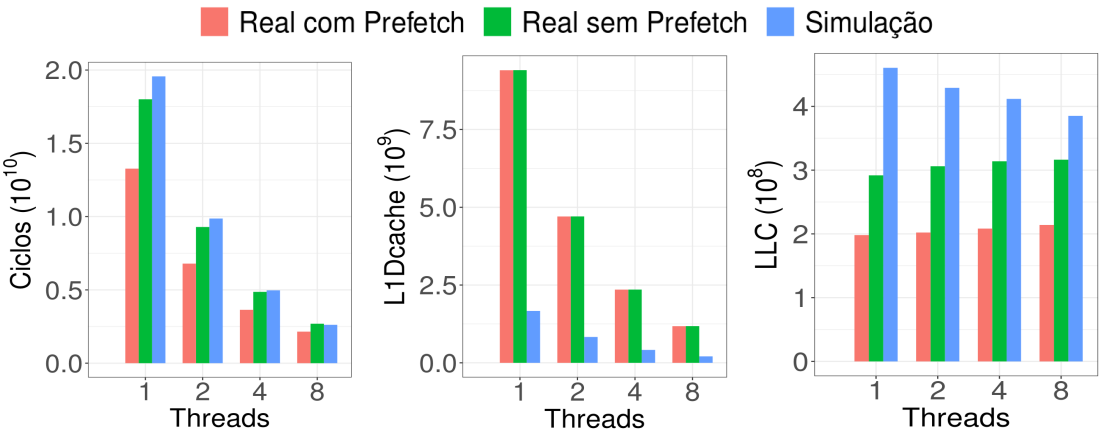

Figura 2. Comparações entre execução real com *prefetch*, execução real sem *prefetch* e simulação da aplicação FT.

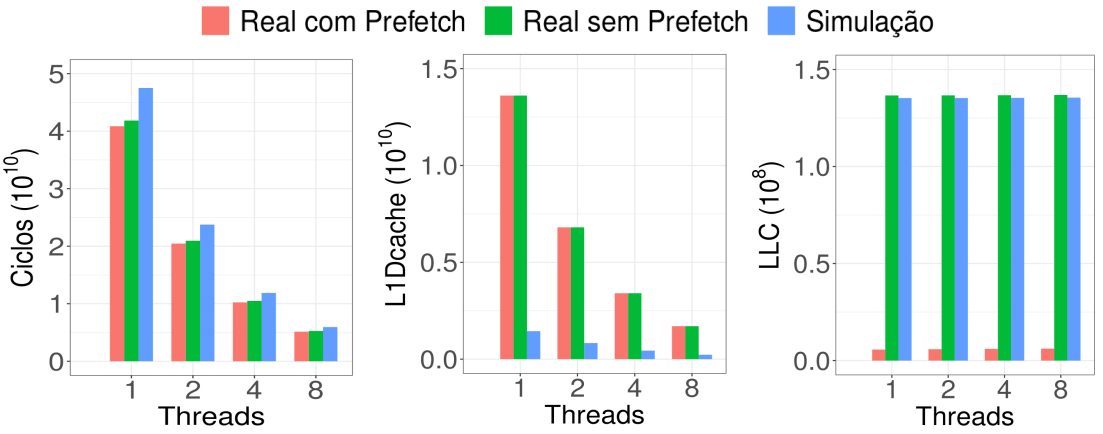

Figura 3. Comparações entre execução real com *prefetch*, execução real sem *prefetch* e simulação da aplicação EP.

A Figura 2 apresenta o comportamento observado na aplicação FT. A Figura 2a mostra o número de ciclos da aplicação. Podemos observar que o erro relativo do ZSim em relação ao *speed-up* da aplicação é bom, pois acompanha a tendência da execução real, comportamento observado em todos os *benchmarks* analisados. A Figura 2b apresenta o número de acessos à memória *cache* L1 de dados da aplicação. Observa-se que, apesar de também seguir a mesma tendência das execuções reais, apresenta uma diferença grande no número de acessos observados. Uma grande diferença também é observada no número de acessos ao último nível de *cache*, como exibido na Figura 2c. No entanto, os acessos à LLC seguem uma tendência contrária à das execuções reais, decrescendo na simulação.

Na Figura 3 observamos os resultados obtidos na aplicação EP. Na Figura 3c, que apresenta os acessos à LLC, pode-se ver claramente a falta de modelagem de *prefetcher*. Comparando-se a execução com *prefetcher* à execução sem *prefetcher*, observa-se que

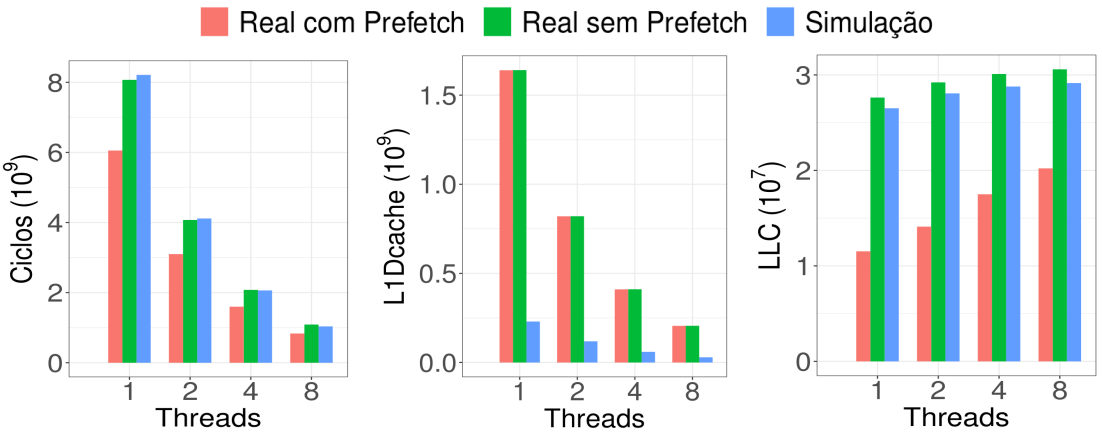

Figura 4. Comparações entre execução real com *prefetch*, execução real sem *prefetch* e simulação da aplicação IS.

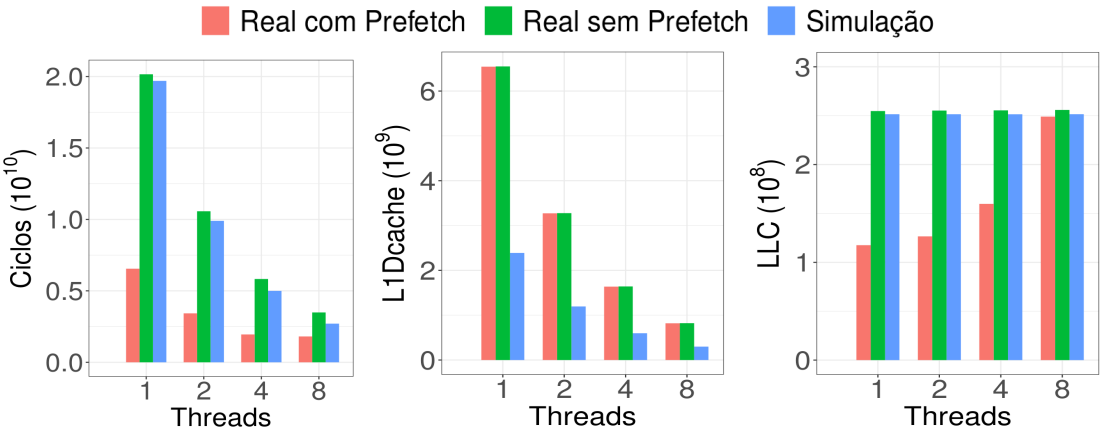

Figura 5. Comparações entre execução real com *prefetch*, execução real sem *prefetch* e simulação da aplicação MG.

um grande número de acessos à LLC é evitado pelos acessos especulativos realizados pelo *prefetcher*. Apesar de o simulador seguir as tendências da execução real fielmente, há uma diferença maior que 2.000% na quantidade de acessos que chegam ao último nível de *cache*. Comportamento semelhante é observado na aplicação IS, ilustrada na Figura 4. Embora nesta aplicação o número absoluto de acessos ao último nível de *cache* seja menor, podemos observar que em todas as execuções que a tendência é de aumento nesta métrica. Esta mesma tendência ocorre nas aplicações LU, UA e MG.

Na aplicação MG, ilustrada na Figura 5, os acessos ao último nível de *cache* da execução real com *prefetch* aumentam tanto que se aproximam de uma execução sem *prefetch*, como visto na Figura 5c. Isto pode ocorrer devido a *prefetchers* imprecisos ou aumento na comunicação. Além disso, com o crescimento do número de *threads*, a diferença entre os tempos da simulação e da execução real sem *prefetch* diminuem se com-

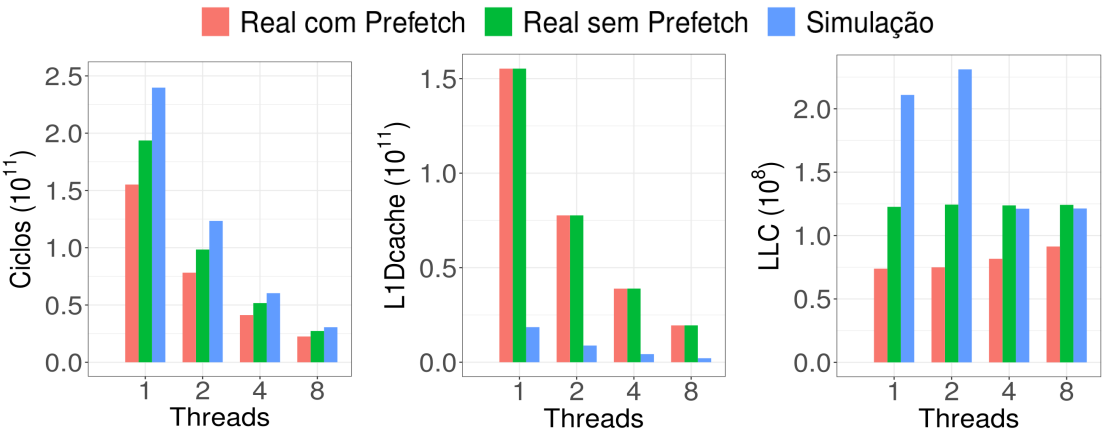

Figura 6. Comparações entre execução real com *prefetch*, execução real sem *prefetch* e simulação da aplicação BT.

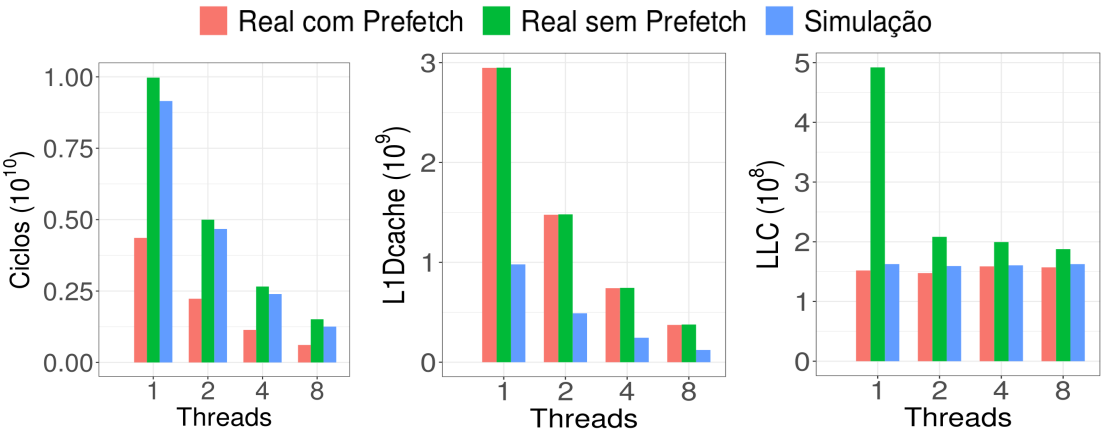

Figura 7. Comparações entre execução real com *prefetch*, execução real sem *prefetch* e simulação da aplicação CG.

parados à execução com *prefetch*. Dessa forma, o paralelismo explicita a comunicação como gargalo ao aumentar o número de requisições no último nível de *cache*, inutilizando *prefetchers* e limitando o comportamento em todos os casos analisados. A única aplicação onde isto não ocorre é a EP, na qual a necessidade de memória e comunicação é tão pequena que cabe no último nível de *cache*. Com isso, observamos pouca diferença entre os ciclos da execução real com *prefetch* ou sem *prefetch* para o EP, como visto na Figura 3c.

Nas aplicações restantes podemos observar comportamentos anômalos. Na aplicação BT, ilustrada na Figura 6, observamos um comportamento similar ao notado até agora para ciclos e acessos à *cache* L1. Porém, a simulação do ZSim apresenta comportamento errático com relação aos acessos à LLC com uma e duas *threads*. Observa-se na Figura 6c que o número de acessos ao último nível de *cache* é muito maior que em ambas as execuções reais. Com quatro e oito *threads* o comportamento volta ao normal. A

Tabela 2. Erro relativo aos acessos à LLC simulados pelo ZSim, resultado da não de modelagem de *prefetcher*.

Threads:	1	2	4	8
CG	7%	8%	1%	3%
EP	2.600%	2.600%	2.150%	2.150%
FT	132%	112%	97%	79%
IS	130%	98%	64%	44%
MG	114%	99%	57%	0%
UA	86%	83%	79%	72%
BT	187%	212%	49%	34%
LU	471%	333%	305%	310%
SP	184%	164%	118%	83%

reexecução da simulação foi feita, e este comportamento se mantém, o que indica algum problema específico na interação do simulador com este *benchmark*.

Na aplicação CG, ilustrada na Figura 7, a execução real sem *prefetch* com uma *thread* apresenta um número muito elevado de acessos ao último nível de *cache*. O desvio padrão observado é menor do que 1%, o que inviabiliza uma possível variância nos experimentos como explicação para tal comportamento. Curiosamente, a simulação do ZSim não segue o comportamento da máquina real sem *prefetch* nestes acessos, mas se aproxima no valor de ciclos.

Portanto, é possível constatar que, para todos os *benchmarks* analisados, o número de ciclos simulados é sempre maior do que nas execuções reais com *prefetcher*. No que diz respeito aos acessos à *cache* de dados, o número observado na simulação é sempre menor. Já as estatísticas de acesso à LLC têm comportamento errático, resultado de imprecisão na modelagem da hierarquia de *cache*.

A Tabela 2 ilustra um resumo da diferença existente entre a simulação e a execução real com *prefetch* no que diz respeito ao número de acessos à LLC . Verifica-se claramente o impacto do *prefetcher* no desempenho das aplicações. Simulações de memória *cache* têm seus efeitos alterados pela falta de *prefetcher*, o que poderia invalidar resultados de vários artigos, inclusive o do próprio ZSim. Com isso, o efeito apresentado em artigos sobre política de substituição de *cache* seria reduzido, pois o *prefetcher* já mitiga a latência de acessos à memória principal [Jain and Lin 2018]. Além disso, o efeito da comunicação e a própria ação do *prefetch* aumentam a contenção nos bancos da *cache* de último nível. Dessa forma, conforme o número de *threads* aumenta, o gargalo encontrado na memória fica cada vez mais explícito, diminuindo o impacto que o *prefetcher* tem sobre o desempenho da aplicação.

6. Conclusão e Trabalhos Futuros

Devido ao modelo *Bound and Weave* implementado no ZSim, torna-se difícil inserir acessos especulativos na hierarquia de memória, impossibilitando a modelagem de *prefetch*. Como consequência disto, observamos diferenças significativas no comportamento das aplicações, com erros de até 2.600%. Dessa forma, diversos estudos têm seus resultados afetados, como artigos sobre políticas substituição de *cache* e simulações que consideram *workloads* limitados pela memória.

Observamos que, para todos os *benchmarks* analisados, o *speed-up* da simulação segue as mesmas tendências observadas nas execuções reais. No entanto, é possível observar algumas inconsistências resultantes da modelagem dos acessos à *cache* implementada pelo ZSim, como nas aplicações BT e CG.

Trabalhos futuros consistirão na investigação de uma possível implementação de *prefetcher* no ZSim, dada a popularidade e velocidade de simulação deste, que resultam em grande praticidade para realização de experimentos. É necessário também investigar-se inconsistências como as observadas nas aplicações BT e CG. Adicionalmente, uma vez que o número de acessos à *cache* de dados observado na simulação é sempre menor que o observado na execução real, é necessário também investigar-se a relação do simulador com acessos à memória em geral. Especificamente, é necessário investigar a decodificação de instruções do simulador, a qual requer atualizações para suportar novas instruções, e.g. AVX (*Advanced Vector Extensions*) [Lomont 2011].

Referências

Akram, A. and Sawalha, L. (2019). A survey of computer architecture simulation techniques and tools. *IEEE Access*.

Austin, T., Larson, E., and Ernst, D. (2002). Simplescalar: An infrastructure for computer system modeling. *Computer*, (2):59–67.

Bakhshalipour, M., Shakerinava, M., Lotfi-Kamran, P., and Sarbazi-Azad, H. (2019a). Bingo spatial data prefetcher. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 399–411. IEEE.

Bakhshalipour, M., Tabaeiaghdaei, S., Lotfi-Kamran, P., and Sarbazi-Azad, H. (2019b). Evaluation of hardware data prefetchers on server processors. *ACM Computing Surveys (CSUR)*, 52(3):52.

Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., et al. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7.

Carlson, T. E., Heirmant, W., and Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE.

Chen, T.-F. and Baer, J.-L. (1995). Effective hardware-based data prefetching for high-performance processors. *IEEE transactions on computers*, 44(5):609–623.

De Melo, A. C. (2010). The new linux'perf'tools. In *Slides from Linux Kongress*, volume 18.

Demme, J. D. (2014). *Overcoming the Intuition Wall: Measurement and Analysis in Computer Architecture*. PhD thesis, Columbia University.

Desikan, R., Burger, D., and Keckler, S. W. (2001). Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th annual international symposium on Computer architecture*, pages 266–277. ACM.

Eeckhout, L. (2010). Computer architecture performance evaluation methods. *Synthesis Lectures on Computer Architecture*, 5(1):1–145.

Fog, A. (2012). The microarchitecture of intel, amd and via cpus: An optimization guide for assembly programmers and compiler makers. *Copenhagen University College of Engineering*, pages 02–29.

Hammarlund, P., Martinez, A. J., Bajwa, A. A., Hill, D. L., Hallnor, E., Jiang, H., Dixon, M., Derr, M., Hunsaker, M., Kumar, R., et al. (2014). Haswell: The fourth-generation intel core processor. *IEEE Micro*, 34(2):6–20.

Huberty, T. J., Meier, S. G., and Agarwal, M. (2018). Content-directed prefetch circuit with quality filtering. US Patent 9,886,385.

Jain, A. and Lin, C. (2018). Rethinking belady's algorithm to accommodate prefetching. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 110–123. IEEE.

James, D. (2012). Intel ivy bridge unveiled—the first commercial tri-gate, high-k, metal-gate cpu. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–4. IEEE.

Jin, H.-Q., Frumkin, M., and Yan, J. (1999). The openmp implementation of nas parallel benchmarks and its performance.

Le, H. Q., Starke, W. J., Fields, J. S., O'Connell, F. P., Nguyen, D. Q., Ronchetti, B. J., Sauer, W. M., Schwarz, E. M., and Vaden, M. T. (2007). Ibm power6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639–662.

Lomont, C. (2011). Introduction to intel advanced vector extensions. *Intel White Paper*, pages 1–21.

Nesbit, K. J. and Smith, J. E. (2004). Data cache prefetching using a global history buffer. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 96–96. IEEE.

Patel, A., Afram, F., and Ghose, K. (2011). Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors. In *1st International Qemu Users' Forum*, pages 29–30.

Sanchez, D. (2016). Zsim tutorial validation. http://zsim.csail.mit.edu/tutorial/slides/validation.pdf.

Sanchez, D. and Kozyrakis, C. (2013). Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer architecture news*, volume 41, pages 475–486. ACM.

Ubal, R., Jang, B., Mistry, P., Schaa, D., and Kaeli, D. (2012). Multi2sim: a simulation framework for cpu-gpu computing. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 335–344. IEEE.

Yourst, M. T. (2007). Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, pages 23–34. IEEE.