Desempenho do Subsistema de Memória de Arquiteturas Multicore e GPU*

Matheus S. Serpa[1], **Eduardo H. M. Cruz**[2], **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

{msserpa, navaux}@inf.ufrgs.br

[2]Instituto Federal do Paraná (IFPR) Paranavaí – PR – Brasil

eduardo.cruz@ifpr.edu.br

Resumo. *Entre os requisitos para atingir alto desempenho em arquiteturas atuais, o uso eficiente do subsistema de memória aparece no estado da arte como um dos mais importantes. Para otimizar uma aplicação científica, os desenvolvedores devem compreender profundamente o comportamento de seus acessos à memória. Nessa linha, nossa pesquisa tem como objetivo entender quais são os limitadores de desempenho relacionados ao subsistema de memória.*

1. Introdução

Arquiteturas *multicore* possuem dezenas de núcleos, além de uma hierarquia de memória complexa para aliviar a latência dos acessos à memória de todas as *threads*. Essas arquiteturas dependem tanto do paralelismo a nível de instrução (ILP) quanto do paralelismo a nível de *thread* (TLP) para obter alto desempenho. Diferente de arquiteturas *multicore*, as *GPUs* possuem milhares de núcleos simples, que sozinhos são menos poderosos que os *multicore* mas em conjunto podem realizar milhares de tarefas por unidade de tempo.

O uso dessas arquiteturas apresenta vários desafios para a Computação de Alto Desempenho [Mittal and Vetter 2015]. As aplicações precisam ser codificadas considerando as particularidades e restrições de cada ambiente, bem como as características distintas de cada arquitetura [Gropp and Snir 2013]. Por exemplo, na hierarquia de memória, a presença de vários níveis de memória cache, alguns compartilhados e outros privados, introduz tempos de acesso não uniformes, que afetam o desempenho das aplicações [Cruz et al. 2016]. Isso é ainda mais crítico em arquiteturas heterogêneas, pois cada acelerador pode ter sua própria hierarquia de memória distinta [Serpa et al. 2017]. Neste contexto, é importante analisar o desempenho e o comportamento de diferentes arquiteturas, a fim de fornecer um melhor suporte aos desenvolvedores, para que eles possam otimizar suas aplicações para o sistema de destino.

Este trabalho em desenvolvimento visa realizar uma análise do impacto do subsistema de memória utilizado em diferentes arquiteturas. Contadores de *hardware* serão utilizados para medir o impacto de diferentes fatores que influenciam o desempenho dos acessos à memória. Serão analisados contadores que medem o uso de memória *cache* e memória principal. Ao fazer isso, buscamos mostrar como diferentes aspectos da hierarquia de memória afetam o desempenho das aplicações. Esse estudo pode servir como base para desenvolvedores de aplicações paralelas otimizarem suas aplicações.

*Este trabalho foi parcialmente financiado por recursos do projeto Petrobras 2016/00133-9.

2. Trabalhos Relacionados

Um modelo de memória para analisar algoritmos para sistemas *many-core* é apresentado em [Ma et al. 2014]. O modelo considera vários parâmetros da arquitetura, como a latência para acessar a memória, o número de núcleos, a largura de banda da memória, o tamanho da memória *cache* e o número de *threads* por núcleo. Ela também considera o número total de operações que a aplicação em execução deve executar, bem como o número total de operações de memória, o uso da memória cache e o número de *threads*. Os autores concluem que aplicações com características semelhantes podem ter desempenho diferente usando diferentes parâmetros da arquitetura. Analisando seu estudo, observamos que é essencial ter uma compreensão completa do comportamento das aplicações, pois o desempenho por si só não nos permite compreender os gargalos de uma aplicação ou arquitetura.

Mei et al. [Mei and Chu 2015] analisaram as características do subsistema de memória em três arquiteturas de GPU: Fermi, Kepler e Maxwell. Eles usaram um *benchmark* de perseguição de ponteiro para latências de acesso à memória, definindo assim as características de todas as memórias dentro de cada GPU. Os autores concluem que o planejamento da arquitetura Kepler foi agressivo em sua largura de banda de memória, muitas vezes subutilizada, e que, na arquitetura Maxwell, mais recursos foram investidos em memória compartilhada, gerando um sistema mais eficiente e equilibrado. Em nosso artigo, nos concentramos em observar os efeitos dessas características em *benchmarks* reais e comparar seu desempenho em diferentes arquiteturas.

Satish et al. [Satish et al. 2012] analisam a lacuna de desempenho entre códigos simples e altamente otimizados em arquiteturas *multi-core* e *many-core*. Nos sistemas avaliados, eles concluíram que o código otimizado poderia melhorar o desempenho em até 24$\times$ sobre a implementação *naive*. Para tanto, consideram características críticas da hierarquia de memória e o comportamento dos algoritmos, além de outros aspectos. Com base nessas informações, eles propõem otimizações para o código-fonte que podem fazer melhor uso da hierarquia de memória e das instruções de vetorização. Eles também mostram como os novos recursos de *hardware*, como o suporte à instrução *gather*, podem ajudar a melhorar o desempenho. O objetivo do nosso trabalho é entender o comportamento e os gargalos de aplicações paralelas em diferentes sistemas, o que pode ajudar desenvolvedores e fabricantes de processadores a melhorarem seus produtos.

Nasciutti et al. [Nasciutti et al. 2018] realizaram uma análise de desempenho de estênceis 3D em GPUs com foco no uso adequado da hierarquia de memória. Eles avaliaram diferentes codificações, mostrando que é necessário explorar a memória compartilhada, a *cache read-only*, e o reuso de registradores para atingir alto desempenho em GPUs. Além disso, mostraram que o tamanho da *cache* L2 afeta diretamente o desempenho da aplicação.

Os trabalhos relacionados demonstram que, levando em consideração os diferentes recursos de cada arquitetura, os desenvolvedores podem melhorar o desempenho de uma aplicação. Nosso trabalho vai além de uma análise de escalabilidade e busca uma maior compreensão do impacto de diferentes aplicações em diferentes sistemas, medidos a partir de contadores de *hardware*.

3. Metodologia

Os experimentos vem sendo realizados nos sistemas Broadwell e Pascal. O sistema Broadwell é composto por dois processadores Intel Xeon E5-2699 v4, em que cada processador é composto por 22 núcleos físicos, permitindo a execução de 88 *threads* com Hyper-Threading. O sistema Pascal é uma GPU Tesla P100, sendo que utilizamos uma das GPUs com 3584 núcleos CUDA.

Os experimentos apresentam a média de 30 execuções aleatórias do *benchmark* Rodinia (OpenMP e CUDA), o qual implementa um conjunto de aplicações com características distintas de execução paralela. O desvio padrão apresentado é dado pela distribuição *t-Student* com um intervalo de confiança 95%. Além disso, também estamos investigando outras métricas, como taxa de uso do núcleo, largura de banda e taxa de acertos das *caches*. As ferramentas Intel PCM [Intel 2012] serão utilizadas na Broadwell, enquanto para a Pascal, utilizaremos a ferramenta nvprof [Nvidia 2016].

4. Resultados preliminares

Para entender melhor o impacto do subsistema de memória nas arquiteturas de Broadwell e Pascal, primeiro verificamos como as aplicações paralelas se comportam nessas arquiteturas. Podemos observar, na Figura 1, que os tempos de execução de cada aplicação variam de arquitetura para arquitetura. Na maioria das aplicações, a Pascal executa mais rapidamente. No entanto, para as aplicações *b+tree*, *hotspot3D*, *nw* e *srad*, a Broadwell executa mais rápido.

Após essa análise inicial, temos a hipótese que há algumas características nas aplicações que fazem com que a aplicação se beneficie especificamente de uma arquitetura ao invés de outra. Isso nos motiva a procurar as razões dessas diferenças, analisando pricipalmente o impacto do subsistema de memória das arquiteturas, ponto mostrado ser um dos mais importantes pelo estado da arte. O resultado pode auxiliar os desenvolvedores a entenderem os gargalos em relação ao paralelismo e à comunicação em cada arquitetura.

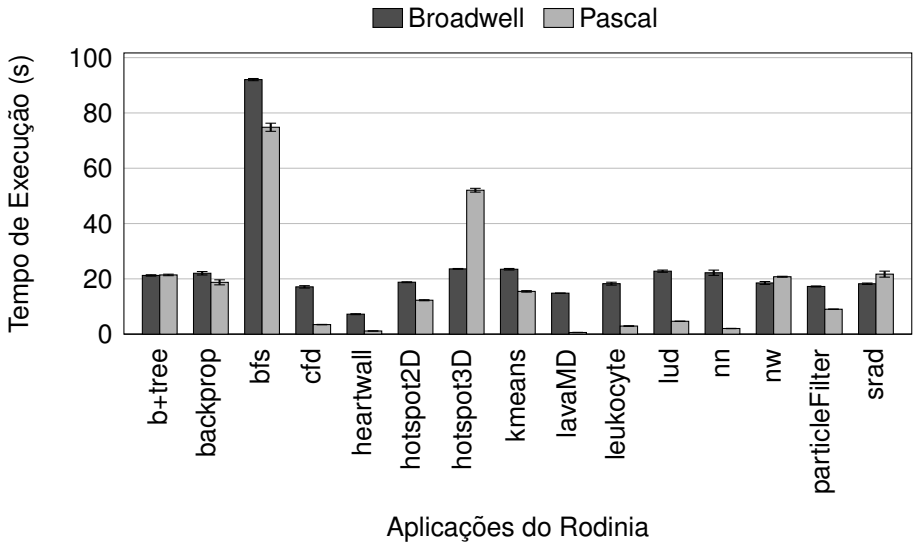

Figura 1. Tempo de Execução nas Arquiteturas Broadwell e Pascal.

5. Considerações Finais

A ampla gama de arquiteturas existente apresenta um desafio para os desenvolvedores ao otimizarem suas aplicações devido às suas diferentes características. Este trabalho em desenvolvimento tem como objetivo analisar o impacto de uma das características mais importantes das arquiteturas atuais: o subsistema de memória.

Trabalhos relacionados nesta área se concentram apenas no desempenho ou consumo de energia, enquanto o nosso trabalho realizará uma análise mais profunda do comportamento da hierarquia de memória. Para isso, contadores de *hardware* serão empregados para medir estatísticas, como faltas de dados nas *caches*, transações de memória principal, entre outros.

Referências

Cruz, E. H., Diener, M., Alves, M. A., Pilla, L. L., and Navaux, P. O. (2016). Lapt: A locality-aware page table for thread and data mapping. *Parallel Computing (PARCO)*, 54:59 – 71.

Gropp, W. and Snir, M. (2013). Programming for exascale computers. *Computing in Science Engineering*, 15(6):27–35.

Intel (2012). Intel Performance Counter Monitor - A better way to measure CPU utilization.

Ma, L., Agrawal, K., and Chamberlain, R. D. (2014). A memory access model for highly-threaded many-core architectures. *Future Generation Computer Systems*, 30:202 – 215. Special Issue on Extreme Scale Parallel Architectures and Systems, Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems, {ICPADS} 2012 Selected Papers.

Mei, X. and Chu, X. (2015). Dissecting GPU memory hierarchy through microbenchmarking. *CoRR*, abs/1509.02308.

Mittal, S. and Vetter, J. S. (2015). A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):69:1–69:35.

Nasciutti, T. C., Panetta, J., and Lopes, P. P. (2018). Evaluating optimizations that reduce global memory accesses of stencil computations in gpgpus. *Concurrency and Computation: Practice and Experience*, page e4929.

Nvidia (2016). Developer Zone - CUDA Toolkit Documentation.

Satish, N., Kim, C., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., Girkar, M., and Dubey, P. (2012). Can traditional programming bridge the ninja performance gap for parallel computing applications? In *International Symposium on Computer Architecture (ISCA)*, pages 440–451, Washington, DC, USA. IEEE Computer Society.

Serpa, M. S., Cruz, E. H., Diener, M., Krause, A. M., Farres, A., Rosas, C., Panetta, J., Hanzich, M., and Navaux, P. O. (2017). Strategies to improve the performance of a geophysics model for different manycore systems. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 49–54. IEEE.