DOI: http://dx.doi.org/10.29069/forscience.2017v5n3.e261

ALTERANDO NÚMERO MÁXIMO DE ACESSOS *IN-FLIGHT* NA MEMÓRIA DA GPU, AVALIANDO DESEMPENHO E CONSUMO ENERGÉTICO EM AMBIENTE SIMULADO[1]

Ariel Gustavo Zuquello[2]
Romulo de Aguair Beninca[3]
Yoji Massago[4]

RESUMO

Este artigo descreve um experimento de desempenho e consumo energético de Graphics Processing Unit (GPU), Central Processing Unit (CPU) Memória Principal e Cache em ambiente simulado. Utilizou-se um conjunto de *benchmarks* para esta tarefa. Os resultados são baseados na alteração da configuração do parâmetro *MaxInFlightMen* da GPU. Esse parâmetro é responsável pelo número máximo de acesso simultâneo ao vetor de memória da GPU. Essas modificações refletiram diretamente na desempenho e consumo energético dos dispositivos, portanto é claro que houve uma redução no consumo de energia da CPU próximo de 1%, em contrapartida, perda de 2% no desempenho, na GPU obteve-se uma redução de 6% no consumo energético e 12% de redução no desempenho porque as memórias caches e principais não tiveram reduções significativas. Este experimento é relevante para provar como as arquiteturas atuais são otimizadas e como qualquer simples mudança paramétrica reflete no desequilíbrio dos dispositivos arquitetônicos.

Palavras-chave: Simulação. Consumo energético. Análise de desempenho. GPU. CPU. Memória. Cache. Arquiteturas.

1 INTRODUÇÃO

Há alguns anos, foi descoberto que a tecnologia utilizada para a criação dos processadores estava próxima do limite, no que se refere à velocidade do processador. Atualmente, o crescimento do poder computacional não é mais obtido pelo aumento de velocidade dos processadores, mas sim pelo da quantidade de núcleos. CPU suporta

[1] **Como citar este artigo:**
ZUQUELLO, A. G.; BENINCA, R. A.; MASSAGO, Y. Alterando número máximo de acessos *in-flight* na memória da GPU, avaliando desempenho e consumo energético em ambiente simulado. **ForScience:** revista científica do IFMG, Formiga, v. 5, n. 3, e00261, jul./dez. 2017.

[2] Mestre em Ciência da Computação pela Universidade Estadual de Maringá (UEM). Professor colaborador da Universidade do Estado de Santa Catarina (UDESC), Campus Chapecó. Currículo Lattes: http://lattes.cnpq.br/7843368686233904. E-mail: arielzuquello@gmail.com.
[3] Mestre em Ciência da Computação pela UEM. Professor do Instituto Federal de Santa Catarina, Campus Canoinhas. Currículo Lattes: http://lattes.cnpq.br/7486046766117014. E-mail: rbeninca@gmail.com.
[4] Mestre em Ciência da Computação pela UEM. Currículo Lattes: http://lattes.cnpq.br/9137805402174351. E-mail: yojimassago@gmail.com.

multinúcleos, acelerando consequentemente o conjunto de instruções e enriquecendo o conjunto arquitetural. Em paralelo, encontramos um crescimento no desempenho dos dispositivos gráficos, que trilha o mesmo caminho devido à tecnologia multinúcleos e alta quantidade de memória nas GPUs (*GraphicsProcessingUnits*) (GUPTA; BABU, 2011).

Para o desenvolvimento de novas arquiteturas computacionais em nível de hardware, seja ela de um CPU, GPU ou de qualquer outro, bem como para a modificação de características das arquiteturas existentes, seria inviável a criação/produção destes no mundo real para executar a verificação do seu desempenho. Assim, os desenvolvedores fazem uso de simuladores para conseguir verificar como seria o desempenho de alguma arquitetura, bem como, o que cada mudança irá acarretar no desempenho da mesma. Para a execução de uma simulação sobre uma arquitetura, além do simulador é necessário um programa ou um conjunto de programas (*benchmarks)* para analisar o comportamento característico do hardware simulado (GUPTA; BABU, 2011).

O objetivo desse trabalho é analisar o desempenho e consumo energético das memórias principais da GPU e CPU, caches, GPU AMD Radeon 7970 de codinome *Southern Island*, em conjunto do CPU IntelRCoreTM i7-3820 da família *Sandy Bridge*, em um ambiente simulado. Para isso, utilizou-se o simulador heterogêneo *Multi2sim*, que simula computação CPU-GPU contando com modelos superescalares, *multithreads* e CPU *multicore*, bem como arquiteturas de GPU. Os *benchmarks* utilizados no experimento são do pacote *Polybench*, que possuem boas aplicações para teste de desempenho e consumo energético.

Finalmente, alterou-se a característica padrão do número máximo de acessos à memória *in-flight (MaxInflightMem)* do bloco de configuração da unidade de memória vetorial *(VectorMemUnit)* na configuração da GPU, objetivando uma melhoria no conjunto arquitetural padrão da GPU, CPU, Memórias e Caches.

A unidade de memória vetorial é responsável por tratar as operações globais da mesma. Dessa forma, trata os dados nas seguintes fases: 1) decodificação, 2) leitura, 3) envio para memória global, 4) escrita e 5) operação concluída.

Após realizar as alterações no parâmetro *MaxInflightMem* foram executadas as simulações, das quais, como resultado obteve-se um ganho de desempenho quase proporcional ao aumento do consumo energético, ou seja, inviável para os padrões atuais de GPU, como será detalhado no transcorrer do experimento.

O texto está dividido em três Seções, além da “Introdução”, as quais estão distribuídas da seguinte maneira: a Seção II aborda sobre o “Ambiente Experimental” para a execução das

simulações, sobre o qual foram realizadas as análises dos resultados. Os "Resultados Obtidos" por meio das simulações são mostrados na Seção III. E, finalmente, na Seção IV está descrita a "Conclusão".

2 AMBIENTE EXPERIMENTAL

Para executar as simulações utilizou-se a arquitetura ISA x86, com as seguintes configurações de CPU (INTEL, 2013) e GPU (AMD, 2013), mostradas na Quadro 1:

CPU - IntelRCoreTM i7-3820	**GPU - AMD Radeon 7970**
CPU - i7-3820	GPU - AMD Radeon 7970
4 Núcleos	32 *Compute Units*
8 Threads	Frequência da Memória 1500MHz
Velocidade de clock 3,6 Ghz	Frequência 925MHz *EngineClock*
Cache inteligente IntelR 10MB	Memória 3GB DDR5
Microarquitetura: *Sandy Bridge*	Microarquitetura: *Southern Island*

Quadro 1 - Configurações de CPU e GPU utilizados
Fonte: INTEL (2013) e AMD (2013).

2.1 CPU - Microarquitetura Sandy Bridge

A Microarquitetura da Intel de codinome *Sandy Bridge* baseia-se nos sucessos da microarquitetura anterior, de codinome *Nehalem*. Ela oferece as seguintes características inovadoras (INTEL, 2013):

- Avançado Vetor de Extensões;
- Maior *front-end* e mecanismo de execução;
- Melhorias na hierarquia de cache com caminho mais amplo;
- Suporte *system-on-a-chip*; e
- Próxima geração da tecnologia *Intel Turbo Boost*.

A Figura 1, exibida a seguir, mostra o fluxo de uma instrução no *pipeline* e os principais componentes do núcleo do processador que é baseado na microarquitetura *Sandy Bridge*:

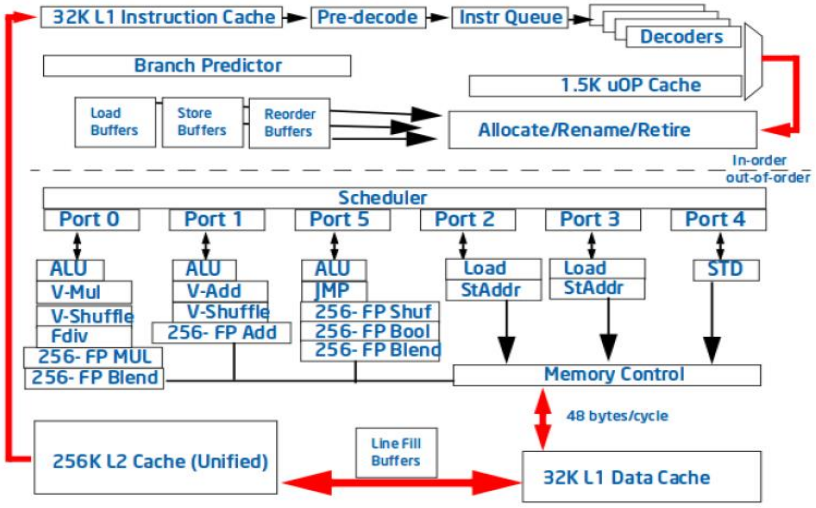

Figura 1 - *Sandy Bridge* Funcionamento do *Pipeline* (CPU)
Fonte: INTEL (2013).

2.2 GPU – Conjunto de Instruções Arquiteturais das Series *Southern Islands*

Segundo Gupta e Babu (2011), o desempenho e a potência das GPUs são o futuro dos sistemas computacionais, sendo estes adequados para os seguintes casos:

- Quando os requisitos computacionais são grandes: a GPU deve entregar uma grande quantidade de poder computacional para cobrir as necessidades de complexas aplicações de tempo real;
- O paralelismo é significativo: A arquitetura do sistema de *pipeline* para gráficos é adequada ao paralelismo (GUPTA; BABU, 2011).

A série AMD *Southern Islands* de processadores (SI-GPU) implementa uma microarquitetura paralela que fornece uma excelente plataforma para aplicações de computação gráfica e de dados de uso geral. Qualquer aplicação paralela de dados que requer alta largura de banda ou requisitos computacionais significativos é uma candidata para a aceleração em dispositivos SI-GPU. A SI-GPU é composta por um processador de comandos que se comunica com o *host*. O processador de envio com *ultra-thread* aceita comandos do processador de comando e distribui o trabalho em toda a gama de unidades computacionais. Cada unidade computacional contém instruções lógicas (busca, buffer, decodificação, edição), unidades ULA: escalar e vetorial com registros, uma alta largura de banda e baixa latência de memória compartilhada (AMD, 2013).

Na Figura 2 mostrada a seguir, visualiza-se um diagrama das unidades computacionais da GPU:

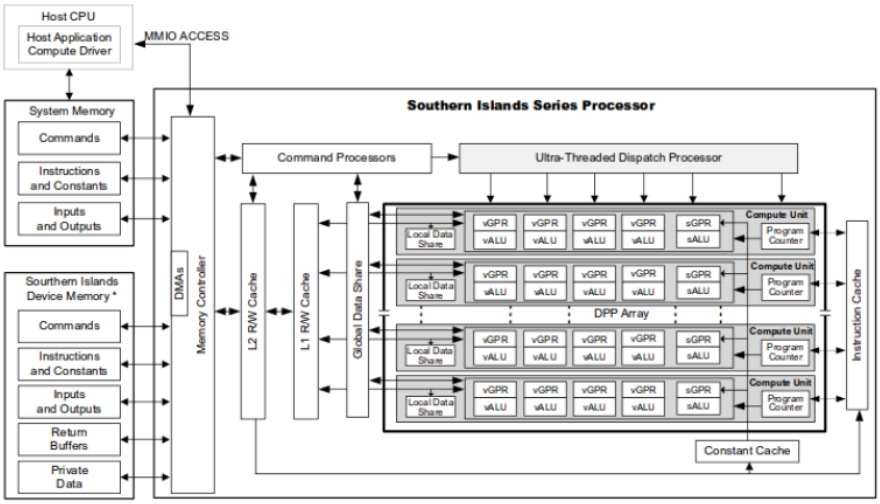

Figura 2 - Diagrama de blocos da Unidade Computacional (GPU).
Fonte: AMD (2013).

2.3 Simulador

2.3.1 Multi2Sim

Multi2Sim é um aplicativo de simulação para a computação heterogênea CPU-GPU escrito em linguagem C. Ele inclui modelos para CPUs superescalares, *multithread*, e *multicore*, bem como arquiteturas de GPU (MULTI2SIM, 2013). Este simulador possui como protocolo de coerência de cache o MOESI, o qual possui cinco estados possíveis para as linhas dos caches: modificado, proprietário, exclusivo, compartilhado e inválido. E, para a simulação das interconexões da rede, o Multi2Sim utiliza a topologia P2P. Outra característica do Multi2Sim é que o mesmo é um simulador do tipo "*Application-Only*", o qual não simula nem o sistema operacional, nem os drivers existentes.

Segundo descrito em Multi2Sim (2013), para conseguir esse efeito, um emulador ISA precisa acompanhar o estado do programa cliente e atualizar dinamicamente instrução por instrução, até a conclusão do programa. O estado de um programa pode ser expresso como a sua imagem da memória virtual e o arquivo de registros. A imagem da memória virtual consiste em um conjunto de valores armazenados em cada localização possível da memória endereçável pelo programa. O estado do arquivo de registro é formado dos valores para cada registro definido em uma arquitetura específica (por exemplo, eax, ebx, ecx, ... em x86).

Dado um estado do programa associado a um ponto específico na sua execução, o emulador é capaz de atualizá-lo para o próximo estado após consumir uma única instrução ISA. Esse processo é realizado em quatro etapas: i) a nova instrução é lida a partir da imagem de memória que contém o código do programa no local apontado pelo ponteiro de instrução; ii) a instrução é decodificada, tirando proveito da interface fornecida pelo módulo de software "desmontador"; *iii)* a instrução é emulada, atualizada a imagem da memória e registros de acordo com *o upcode* da instrução e de entrada/saída de operandos; e *iv)* o ponteiro de instrução é movido para a próxima instrução a ser emulada.

Quando usado de forma independente, o simulador funcional executa um programa para a conclusão. Inicialmente, o ponteiro de instrução é simulado na entrada do programa, isto é, o endereço da primeira instrução a ser executada. Em seguida, um *loop* de simulação mantém instruções emulando repetidamente até que o programa funcione com sua rotina de rescisão. Mas o simulador funcional também fornece uma interface para simulação para próximas etapas.

Neste caso, uma entidade independente de software pode requerer emulação para a próxima instrução disponível. Depois de atualizar internamente o estado do programa de convidado, o simulador retorna todas as informações resultantes com a instrução emulada, conforme as informações retornadas pela sua chamada interna para o "desmontador" (MULTI2SIM, 2013).

2.3.2 McPAT

Fornece uma interface XML que pode interagir com um simulador de desempenho adequadamente modificado por meio de um arquivo de texto simples em formato XML, o McPAT recebe os parâmetros arquitetônicos e as características tecnológicas do processador modelado, bem como algumas estatísticas de acesso estruturadas fornecidas por um simulador de desempenho. Com essas informações, calculam-se as estatísticas da área total dos circuitos e da energia consumida pelo sistema. O McPAT utiliza a ferramenta Cacti (SHYAMKUMAR *et al.*, 2008) para obter os modelos das matrizes de dados, tabelas, e estruturas de cache do arquivo de entrada XML (MULTI2SIM, 2013).

Para que o McPAT possa utilizar os relatórios gerados pelo Multi2Sim é necessário fazer uma conversão para o formato XML. Para tal faz-se necessário uma ferramenta para mapeamento. Dentre as aplicações existentes que foram encontradas: *m2s2xml.sh*

(MULTI2SIM, 2013) e *multi2simtomcpat.java* (MULTI2SIM, 2013). Para essa simulação utilizou-se o *multi2simtomcpat.java.*

2.3.3 Benchmark – Polybench

É uma coleção de benchmarks cujo objetivo é uniformizar a execução e monitoramento de *kernel*. As características do PolyBench incluem (GRAUER-GRAY; POUCHET, 2012):

- Um único arquivo ajustável em tempo de compilação, usado para a preparação do *kernel*. Ele executa operações extras, como limpeza de cache antes da execução do *kernel*, e pode definir a programação em tempo real para evitar a interferência do sistema operacional;
- Inicialização de dados não nulos;
- Construções sintáticas para evitar qualquer eliminação do código no *kernel*;
- Limites paramétricos para *loop*, e para implementação de uso geral;
- Limpeza de marcação do *kernel*, usando delimitadores *# pragmascop e # pragmaendscop.*

O Polybench para GPU possui uma coleção de benchmarks para as linguagens CUDA (NVIDIA, 2013) (*Compute UnifiedDeviceArchitecture*), OpenCL(*Open ComputingLanguage*) e HMPP(*HybridManycoreParallelProgramming*). Destes, utilizar-se-á o conjunto de benchmark para GPU, na linguagem OpenCL.

No OpenCl, um programa é executado num dispositivo computacional, que pode ser um CPU, GPU, ou outro dispositivo acelerador. O dispositivo contém um ou mais *"compute units"* (núcleos de processamento), sendo que essas unidades são compostas por um ou mais SIMD (*Single-InstructionMultiple-Data*) (STONE; GOHARA; SHI, 2010).

Dentre os diversos benchmarks existentes no conjunto do Polybench, para a linguagem OpenCl, selecionaram-se alguns para o experimento, os quais são descritos na Quadro 2:

Benchmark	Descrição
2mm	2 Matrix Multiplications (D = A.B, E = C.D)
3mm	3 Matrix Multiplications (E=A.B; F=C.D; G=E.F)
Atax	Matrix Transpose and Vector Multiplication
Bicg	BiCG Sub Kernel of BiCGStab Linear Solver
Correlation	Correlation Computation
Covariance	Covariance Computation
fdtd-2d	2-D Finite Different Time Domain Kernel
Gem	Matrix-multiply C=alpha.A.B+beta.C
Gesummv	Scalar, Vector and Matrix Multiplication
Gramschmidth	Gram-Schmidt decomposition
Mvt	Matrix Vector Product and Transpose
syr2k	Symmetric rank-2k operations
Syrk	Symmetricrank-k operations
2dconv	Não utilizado
3dconv	Não utilizado

Quadro 2 – Benchmarks
Fonte: Adaptada de Grauer-Gray e Pouchet (2012).

2.3.4 Execução

Após a seleção do Simulador (Multi2Sim), da suíte de benchmarks (PolyBench) e do Framework (OpenCL) realizou-se os experimentos com a alteração no parâmetro *MaxInflightMem* do arquivo de configuração da GPU, parâmetro este, responsável pelo número de acessos à memória *In-Flight* do bloco de configuração da memória vetorial (*VectorMemUnit*), analisando o desempenho e o consumo energético. O padrão desse parâmetro é: MaxInflightMem = 32. Realizaram-se algumas alterações nesse parâmetro, tanto para um maior número de acessos (64 e 128) quanto para um menor (16 e 8), sendo feito um comparativo no comportamento do conjunto após as modificações.

Com essas ferramentas e dados obtidos, executou-se a simulação, seguindo os passos exibidos na Figura 3. Nela observa-se que, para a execução dessa simulação, foram necessários vários arquivos (configuração, benchmarks, mapeamento e XML base) e aplicativos/ferramentas (Multi2Sim, McPAT, GNUPlot, conversor e extrator de dados - este último pode ser feito manualmente).

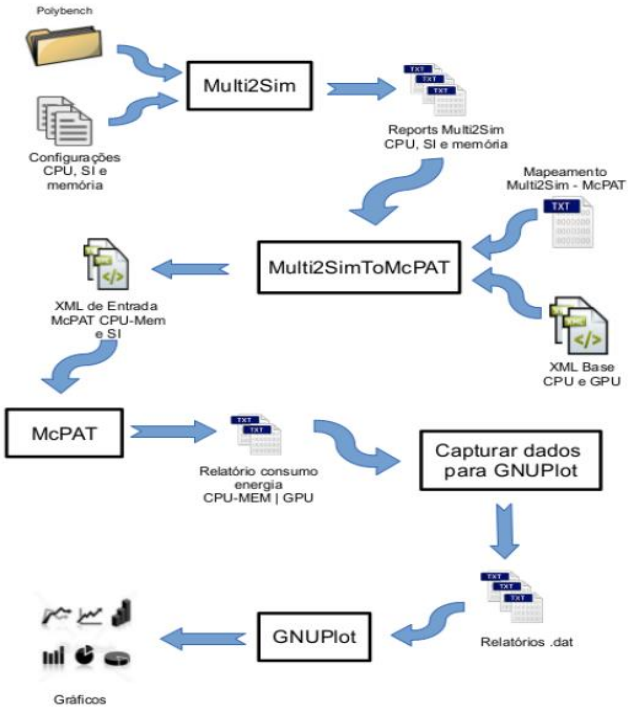

Figura 3 – Passos executados para simulação
Fonte: Autor (2013).

Os dados selecionados para análise foram: desempenho (número de ciclos, instruções por ciclo da CPU e GPU) e consumo energético (Potência da GPU, CPU e Memórias) de todos os resultados obtidos pelas execuções dos benchmarks.

3 RESULTADOS E DISCUSSÕES

Após análise final dos dados realizou-se a normalização dos mesmos, levando em consideração o desempenho (ciclos) e o consumo energético (potência) de CPU e GPU que sofreram alterações mais relevantes, sobre o qual foram obtidos os gráficos comparativos, entre estes dois valores (Figura 4 e Figura 5). Assim observou-se que as alterações feitas no parâmetro *MaxInFlightMen* geraram um desequilíbrio no comportamento funcional dos dispositivos.

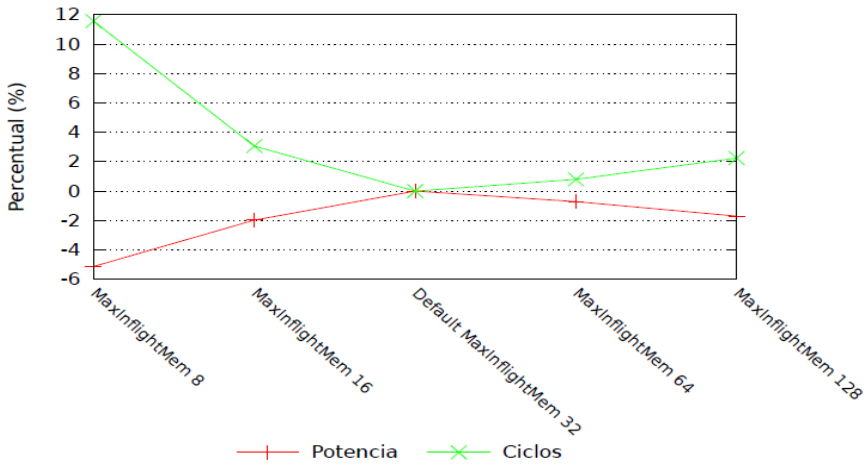

Figura 4 – Potência X Ciclos – GPU
Fonte: Dos autores (2013).

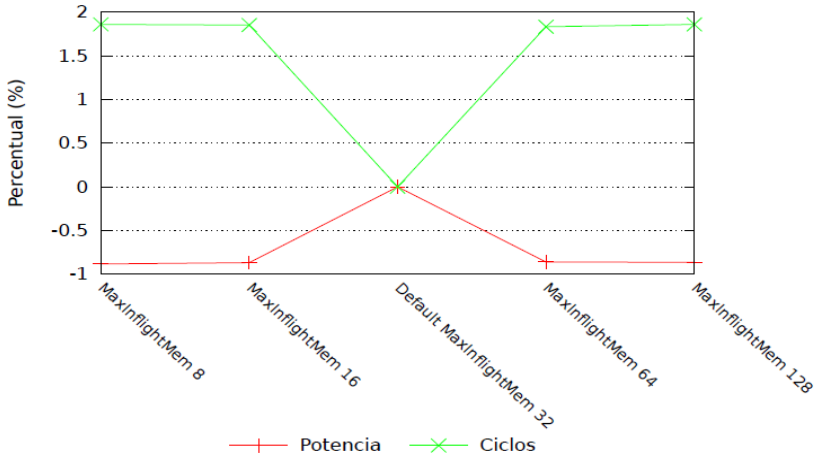

Figura 5 – Potência X Ciclos – CPU
Fonte: Dos autores (2013).

Por meio das alterações realizadas nesses parâmetros, não houve uma melhora relevante no custo/benefício da GPUe CPU, em relação ao parâmetro original. Quando se obtém uma pequena redução no consumo energético, o desempenho cai quase que proporcionalmente ou até mais, sendo que o mesmo acontece no sentido inverso, como observado nas Figuras 4 e 5.

4 CONCLUSÃO

No intuito de encontrar uma solução que melhorasse o desempenho e/ou reduzisse o consumo energético dos dispositivos CPU, GPU, Memórias Principais e Caches, executou-se uma simulação com uma modificação paramétrica na arquitetura; o parâmetro modificado foi *MaxInFlightMem* do *VectorMemUnit*, o qual é o número máximo de acessos à memória vetorial da GPU em tempo de execução, o mesmo possui como padrão o valor 32. Essas alterações surtiram efeito em alguns programas da coleção de benchmarks, sendo que em outros não se observou mudanças significativas no desempenho (ciclos) nem no consumo energético (Potência) da GPU, CPU, memórias e caches.

Assim, as alterações encontradas foram normalizadas e analisadas, logo constatou-se que ao fazer uma modificação apenas neste parâmetro (*MaxInFlightMen*), não resulta numa melhora significativa de desempenho em relação ao desempenho e/ou consumo energético. Entretanto, os valores com maior relevância encontrados ao final do experimento foram, na CPU, redução de aproximadamente 1% no consumo energético e perda de 2% no desempenho. Já na GPU, obteve-se uma redução próxima a 6% no consumo energético e redução de 12% no desempenho. Os demais dispositivos analisados, memórias principais e caches, também não apresentaram melhorias significativas.

Contudo, não se pode afirmar que essa modificação será implementada na prática, pois trata-se de um experimento em ambiente simulado, onde há algumas ameaças aos experimento como por exemplo: divergência no protocolo de coerência de cache, onde o simulador utiliza o protocolo MOESI, enquanto o CPU I7 utiliza MESI e também pelo fato do simulador ser *Aplication-Only* dentre outras.

No entanto, em futuros experimentos poder-se-á fazer uso dessa base de informações para testar novas arquiteturas com intuito de aperfeiçoar o desempenho e o consumo energético arquitetural nesse tipo de dispositivos.

CHANGING MAXIMUM NUMBER OF IN-FLIGHT ACCESSES IN THE MEMORY OF THE GPU, EVALUATING PERFORMANCE AND ENERGY CONSUMPTION IN A SIMULATED ENVIRONMENT

ABSTRACT

This article describes an experiment in performance and energy consumption of the GPU, CPU, Main Memory and Cache in a simulated environment. It was used a set of benchmarks for this task. The results are based on the changing the GPU MaxInFlightMem parameter setting. This parameter is responsible for the maximum number of simultaneous accesses to the memory of GPU vector. These changes directly reflected the performance and energy

consumption of the devices, thus it is clear that there was a reduction in CPU power consumption close to 1%, in contrast, 2% loss in performance. GPU obtained a reduction of 6% in energy consumption and 12% reduction in performance because the main and caches memories have not had significant reductions. This experience is relevant to prove how current architectures are optimized and as any simple parametric change reflects the imbalance of architectural devices.

Keywords: Simulation. Energy Consumption. Performance Analysis. GPU. CPU. Memory. Cache. Architectures.

REFERÊNCIAS

ADVANCED MICRO DEVICES (AMD). **AMD Radeon™ HD 7970 Graphics**. [©2013] Disponível em: <http://www.amd.com/enus/products/graphics/desktop/7000/7900>. Acesso em: 13 maio 2013.

GRAUER-GRAY, S.; POUCHET, L.-N. **PolyBench/C**: the polyhedral benchmark suite. Version 3.2. [Columbus, OH, 2012]. Disponível em: <http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/>. Acesso em: 17 maio 2013.

GUPTA, S.; BABU, M. R. Performance analysis of GPU compared to singlecore and multicore CPU for natural language applications. **International Journal of Advanced Computer Science andApplications** (IJACSA), v. 2, n. 5, p. 50-53, 2011. Disponível em: <https://scholar.google.com/citations?view_op=view_citation&hl=en&user=pfSSNyMAAAAJ&citation_for_view=pfSSNyMAAAAJ:D_sINldO8mEC>. Acesso em: 17 maio 2013.

INTEL. **Intel®Core™ i7-3820 Processor (10MB cache, UP to 3.80 GHZ)**. [Santa Clara, CA, USA: Intel Corporation, 2013]. Disponível em: <http://ark.intel.com/products/63698/Intel-Core-i7-3820-Processor-10M-Cache-up-to-3_80-GHz>. Acesso em: 29 abr. 2013.

MULTI2SIM. **The Multi2Sim simulation framework**: a CPU-GPU Model for heterogeneous computing (For Multi2Sim v. 4.2). [2013]. 210 p. Disponível em: <http://www.multi2sim.org/downloads/m2s-guide-4.2.pdf>. Acesso em: 10 abr. 2013.

NVIDIA. **Parallel programming and computing platform | CUDA | NVIDIA.** 2013. Disponível em: <http://www.nvidia.com/object/cuda_home_new.html>. Acesso em: 5 de maio de 2013.

SHYAMKUMAR, T. et al. CACTI 5.1. **HP Labs**, Palo Alto, p. 1-74, Abr. 2008. Disponível em: <http://www.hpl.hp.com/techreports/2008/HPL-2008-20.html>. Acesso em: 28 abr. 2013.

STONE, J. E.; GOHARA, D.; SHI, G. OpenCL: a parallelProgramming Standard for heterogeneous computing systems. **Computing in Science & Engineering**, v. 12, n. 3, p. 66–

73, Maio 2010. Disponível em: <http://dx.doi.org/10.1109/MCSE.2010.69>. Acesso em: 28 abr. 2013.

Recebido em: 11/08/2017

Aprovado em: 1º/09/2017

Publicado em: 13/11/2017