Mecanismo para reduzir o desperdício energético na pós-execução de aplicações em GPU

Emmanuell D. Carreño[1], **Adiel S. Sarates Jr.**[1], **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 – Brazil

{edcarreno,assaratesj,navaux}@inf.ufrgs.br

Abstract. *With the increase demand of GPU accelerators for general purpose processing in HPC the impact of power consumption of these resources cannot be overlooked. To reduce the power consumption some strategies have been applied, but their approaches have been mostly focused in energy savings during the application execution. This work is focused in the post-execution energy saving. When the post-execution behavior of the applications is analyzed in newer GPU cards, it is observed that the power draw do not return to idle state in an efficient way, creating an unexpected power waste. For this inefficient process to return to the idle power draw and the power waste in the post-execution, it was developed a strategy to reduce this waste with a minimal impact in global performance. With this strategy, energy savings up to 15% in the post-execution stage in sequential application runs were obtained. In the case of a single execution our approach allows savings up to 59% in the post-execution power consumption.*

Resumo. *Com o aumento da demanda por aceleradores como GPU para processamento de propósito geral em HPC, o impacto do consumo de energia destes recursos não pode ser negligenciado. Para reduzir o consumo de energia, algumas estratégias foram aplicadas, mas suas abordagens têm sido quase sempre focadas na economia de energia durante a execução da aplicação. Este trabalho é focado na economia de energia na pós-execução de aplicações. Quando o comportamento da pós-execução das aplicações é analisado em novas placas, observa-se que o consumo de energia não retorna rapidamente para o estado de repouso (idle) de maneira eficiente, gerando um desperdício de energia inesperado. De encontro a este processo ineficiente visando uma redução no desperdício de energia no período de pós-execução, foi desenvolvida uma estratégia para reduzir este desperdício com um impacto mínimo no desempenho global. Com essa estratégia, foi obtida uma economia de energia de até 15% nesta fase de pós-execução de uma aplicação executada sucessivas vezes. No caso de uma única execução, nossa abordagem permite uma poupança até 59% no consumo de energia na pós-execução.*

1. Introdução

Nos últimos anos, o uso de Unidades de Processamento Gráfico (GPU) como aceleradores de computação tornou-se uma tecnologia importante para computação de alto desempenho (*HPC*). Uma pesquisa recente [Top500.org 2013] dos 500 maiores supercomputadores mostra que 8% deles são sistemas heterogêneos baseados em placas GPU utilizadas

Esta pesquisa foi parcialmente financiada pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processo 132123/2013-4 e pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), processo ME 14/2012.

para propósito geral (*GPGPUs*). Além de microprocessadores de alto desempenho, alguns destes supercomputadores incluem um grande número de *GPGPUs*. Cada *GPU* consiste em milhares de núcleos de processamento altamente especializadas e durante a computação, as *CPUs* descarregam alguns segmentos de código paralelizáveis para essas *GPUs* afim de acelerar a execução. As primeiras *GPUs* foram desenvolvidas para computação gráfica, que exige alta capacidade de processamento computacional, mas com sua evolução passaram a ser usadas na execução de aplicações de uso geral. Para suportar tais aplicações em *GPUs*, foi desenvolvido pela NVIDIA o *CUDA framework* (*Compute Unified Device Architecture*) [Lee et al. 2011]. Com o uso deste framework, programar em *GPUs* tornou-se mais simples, deixando de ser necessárias diversas bibliotecas, como OpenGL.

No entanto, os custos operacionais e energéticos dos sistemas *HPC* em centros de dados aumentaram 36% no período de 2005 a 2010 (indo de 7 *gigawatts* para quase 10 *gigawatts* nos Estados Unidos), de acordo com relatório de [Koomey 2011] sobre o crescimento recente do consumo energético dentro de centros de dados. Este crescimento foi, na verdade, atenuada pela recessão global, que tinha sido prevista pela Agência de Proteção Ambiental para dobrar até 2016, com o Departamento de Energia entrando na era *exascale* da supercomputação [Tiwari et al. 2012]. Este emergente *pensamento verde* que reivindica para o sistema e os programas de *HPC* mais eficiência energética é mais presente em laboratórios de *HPC*. Infelizmente, mesmo com esses esforços visando a *computação verde*, a principal área de pesquisa sobre o consumo energético de aplicações é focada principalmente no desempenho da aplicação.

Para alcançar a era do *exascale*, reduzindo o consumo de energia, economizando dinheiro, ou, principalmente, economizando mais energia, também precisamos olhar para o comportamento da ***pós-execução*** das aplicações. Quando uma aplicação chega ao fim, o que acontece nesse momento também é importante, ela pode estar perdendo joules valiosos. Com todos os esforços para economizar energia durante o tempo de execução da aplicação, algumas técnicas são muito úteis e essas técnicas podem ser adaptadas para reduzir tal consumo. Melhorar a eficiência energética é um desafio contínuo em *HPC*, devido à necessidade constante de desempenho, apesar das restrições do orçamento de energia e os custos econômicos. Como resultado, é necessário investir em tecnologias de baixo consumo e em sistemas heterogêneos, também baseados em *GPUs*.

2. Trabalhos Relacionados

O consumo de energia em *GPUs* foi estudado usando métodos experimentais em que o consumo é medido diretamente em todo o sistema(*CPU*, *GPU* e periféricos), por meio de modelos de consumo ou desenvolvendo técnicas de otimização de energia [Chen et al. 2011] [Kasichayanula et al. 2012] [Ukidave and Kaeli 2013] [Ukidave et al. 2013].

O Escalonamento Dinâmico da Freqüência de Tensão (sigla em inglês DVFS) é uma técnica usada para reduzir dinamicamente a tensão do dispositivo permitindo um menor consumo de energia. Em [Ge et al. 2013] é desenvolvido um mecanismo usando DVFS para *GPUs*, no entanto, encontraram um efeito colateral indesejável desta técnica, que provoca uma perda potencial no desempenho. Quando a freqüência de clock opera em um nível mais baixo, causa um atraso na execução da aplicação. Eles concluem que

deve haver então um equilíbrio delicado entre o consumo de energia reduzido e perda de rendimento a fim de atingir o consumo de energia e desempenho aceitáveis.

O conceito de *"Fila Verde"* também é baseado em uma série de técnicas de aplicação de DVFS para uso dentro de aplicações *MPI* [Tiwari et al. 2012]. A *"Fila Verde"* é um processo escalável implantado no Centro de Supercomputação de San Diego, na qual os usuários e os operadores das instalações, conseguiram uma economia de 31,7% nos custos operacionais relacionados.

Outro trabalho, embora também relacionado com *CPU* [Lee et al. 2011], mostra como melhorar a taxa de transferência de *GPUs* com restrições de energia, otimizando o número de núcleos operacionais, as tensões e freqüências de ambos os núcleos e interconectores/*caches* intrachips, dependendo das características de cada aplicação, analisando o impacto do número de núcleos de operação, sua tensão e frequência em uma *GPU* operando sob uma restrição de energia.

Nos últimos anos, as *GPUs* de alta performance que suportam o escalonamento de freqüência, como a K20 da Nvidia entraram no mercado. Os benefícios desse escalonamento aparece como uma abordagem eficaz para economizar energia. Em geral, ao reduzir a tensão do núcleo da *GPU*, pode ser gerada uma economia significativa, no entanto [Mei et al. 2013] mostrou que não é fácil encontrar configuração ideal para tal escalonamento.

Para atingir a *era do exascale*, são utilizadas técnicas como o escalonamento de freqüência em núcleos de *GPU* e da memória bem como a divisão da carga de trabalho entre *CPU* e *GPU*. Estruturas para controlar esse escalonamento foram desenvolvidas e *GPGPUs* são agora usadas para fins de *HPC*. *Benchmarks*, bibliotecas de álgebra ou aplicações reais tem sido usados para validar as novas idéias e conceitos de *Computação Verde* [Padoin et al. 2012] [Ma et al. 2012] [Huang et al. 2009].

3. Recursos

As novas placas *GPUs* possuem vários contadores de *hardware* integrados em seus circuitos. Os dados de métricas podem ser recuperados usando bibliotecas e métodos adequados, sem gerar uma sobrecarga significativa. Nesta seção, serão apresentadas as aplicações analisadas, além da placa Nvidia Tesla K20c e o aplicativo desenvolvido para recuperar as métricas necessárias.

3.1. Nvidia Tesla K20

A Nvidia Tesla K20, é uma placa da arquitetura Kepler que compreende mais de 7 bilhões de transistores. É equipada com novas funcionalidades focadas no desempenho de computação e foi projetada para ser uma potência de processamento paralelo para o mercado de *HPC*. Ela possui 2.496 núcleos de processadores que trabalham a uma frequência de 706MHz com 5GB de memória a 2.6GHz.

A arquitetura Kepler foi projetada para melhorar a eficiência de energia e ao mesmo tempo oferecer um melhor desempenho por *watt* em relação a arquitetura anterior(Fermi). As unidades de execução operam em uma alta taxa de *clock*, permitindo que o chip alcance maior produtividade. A contrapartida da placa *GPU* é um consumo energético maior do que uma *CPU*. O consumo máximo de uma K20 é 225W, enquanto a sua energia em idle é 14W [NVIDIA 2012].

3.2. Nvidia NVNL

Para acessar o hardware da placa GPU, a Nvidia oferece acesso aos contadores de *hardware* através do *NVIDIA Management Library* (NVML). Esta biblioteca é uma API baseada em C que permite o monitoramento e gerenciamento dos estados dos dispositivos mais recentes da NVIDIA.

Esta biblioteca permite o acesso de baixo nível para o acesso das características da placa, dados de sensores e informações dos contadores. Além disso, permite que os administradores consultem o estado do dispositivo e com o conjunto adequado de privilégios, conceder-lhes acesso para modificar o estado do dispositivo, como a frequência de operação da mesma. Esta biblioteca é compatível com alguns dispositivos de Fermi e Tesla, mas com suporte limitado a placas mais antigas [NVIDIA 2013].

3.3. GPUtool

Para recuperar as métricas necessárias para analisar o consumo de energia durante a *pós-execução* foi desenvolvida uma ferramenta executada em segundo plano, baseada em C usando as bibliotecas NVIDIA NVML. Esta aplicação funciona recebendo dados dos contadores da placa, sem aumentar de forma significativa o consumo de energia durante a execução em paralelo com a aplicação que está sendo analisada.

A *GPUtool* é executada somente na *CPU*, que coleta dados considerados relevantes para avaliar o comportamento da aplicação pelos contadores da *GPU*. Algumas das métricas recolhidas são o consumo de energia, temperatura, freqüência da memória da *GPU*, freqüência dos multiprocessadores (*SMX*) e da velocidade do cooler durante a execução da aplicação e após a conclusão, até o retorno do consumo de energia para o estado de idle. Para evitar a obtenção de dados incorretos sobre a aplicação e seu comportamento posterior, durante os testes foi assegurado que apenas uma aplicação estava sendo executada na *GPU* durante o processamento e nenhuma aplicação estava sendo executada durante o idle.

Para minimizar a sobrecarga da *CPU*, o que poderia causar o consumo de energia também na *GPU*, o tempo de acesso para recuperar as métricas foi definida para ser maior do que 15ms com base nas medições realizadas. A *GPUtool* pode definir esse intervalo de tempo de acesso por *flags* antes da sua execução. Outro recurso implementado na *GPUtool* é poder definir a freqüência do processador e memória da GPU por valores pré definidos, caso sejam um par válido de valores da placa.

A principal razão para se desenvolver uma aplicação de perfis em vez de utilizar uma já disponível, foi o fato da ausência de uma ferramenta que permita a coleta de dados, mesmo após o final da execução, uma vez que a maioria das ferramentas de análise disponíveis não funcionam sem um contexto válido na *GPU*, o que significa que enquanto não há nenhuma aplicação rodando na GPU, não é possível obter algumas das métricas da placa.

3.4. Benchmarks

Para obter os dados na pós-execução, como o consumo de energia das aplicações na *GPU*, foram utilizados um conjunto de aplicações de dois *benchmarks*: Magma e Rodinia.

3.4.1. Magma

MAGMA é um *framework* para executar cálculos de álgebra linear para arquiteturas heterogêneas. Muitos sistemas heterogêneos permitem que os aplicativos explorem plenamente o poder que cada um de seus componentes [Haidar et al. 2008].

MAGMA tem um conjunto de testes que inclui testes de precisão simples e dupla para operações com matrizes reais e complexas. Tamanhos de entrada diferente para cada instância permitem uma ampla gama de cargas de processamento e, consequentemente, um comportamento de consumo de energia diferente para cada teste [Agullo et al. 2009] [UTK 2008].

Para o perfil usando MAGMA utilizou-se uma rotina que calcula a fatoração QR de uma matriz A geral onde

$$A = QR.$$

As instâncias do *benchmark* usadas para obter o consumo energético na pós-execução são apresentados na Tabela 1.

Subrotina	Tipo de dado
SGEQRF	Valores reais - Precisão simples
DGEQRF	Valores reais - Precisão double
CGEQRF	Valores complexos - Precisão simples
ZGEQRF	Valores complexos - Precisão double

Tabela 1. Sub-rotinas do Magma usadas para analisar o comportamento do consumo de energia pós-execução.

3.4.2. Rodinia

Rodinia é um conjunto de *benchmarks* livres onde suas aplicações são projetadas para infra-estruturas de computação heterogêneas com *GPUs* e *CPUs multicore* [Che et al. 2010]. Os *benchmarks* abrangem uma gama de padrões de paralelismo e computação, fornecendo aplicações de problemas de Álgebra Linear e processamento de imagens médicas. Cada aplicação ou *kernel* representam um tipo diferente de comportamento [Che et al. 2009]. Com o Rodinia, foram utilizados os testes de CFD, que consistem em uma matriz não estruturada de equações de Euler, baseadas na dinâmica de fluidos. As instâncias do *benchmark* usadas para obter o consumo energético na pós-execução são apresentados na Tabela 2.

Entrada de Dados	Sigla
fvcorr.domn.097K	097K
fvcorr.domn.193K	193K
missile.domn.0.2M	0,2

Tabela 2. Sub-rotinas de CFD usadas para analisar o comportamento do consumo de energia na pós-execução.

4. Metodologia para reduzir o desperdício de energia na Pós-Execução

Quando uma aplicação na *GPU* termina, espera-se que o consumo de energia na placa retorne para o estado de *idle*. Isso acontece se nenhuma outra aplicação começar a executar antes da chegada a esse estado, para evitar o consumo desnecessário de energia. O estado de *idle* é caracterizado por um consumo mínimo de energia, tanto pela memória, como pelos processadores, que permanecem em seus valores mais baixos de consumo. O mecanismo para reduzir este tempo ocioso e o desperdício de energia subsequente é aplicar o método de escalonamento de frequência quando se inicia a pós-execução, desta forma ambas as frequências são definidas para os estados mais baixos, resultando em uma diminuição rápida do consumo de energia, assim reduzindo o desperdício de energia para este estado não-computacional.

As novas placas GPU permitem o acesso de seus contadores de *hardware*, incluindo os contadores de consumo de energia, que são responsáveis por armazenar dados como de consumo de energia instantânea entre outros valores. Para ler estes contadores é possível a utilização de alguns analisadores como a *Nvidia Visual Profiler*, no entanto esta ferramenta não consegue acesso aos dados do comportamento da pós-execução. Para isso, foi desenvolvida uma ferramenta chamada *GPUtool*, onde é possível coletar e analisar os dados fornecidos pelos contadores da *GPU* após o término de qualquer aplicação na *GPU*. Com o perfil resultante, foi analisada a quantidade de energia desperdiçada pela *GPU*, indo para estado de *idle* de maneira convencional e com o uso de nossa técnica.

As aplicações executadas nas *GPUs* foram executadas sem restrições de suas freqüências, permitindo que o gerenciamento de energia da *GPU* defina a freqüência adequada. Os maiores valores de cada frequência são 2600MHz para a memória da *GPU* e 766MHz para os núcleos da *GPU*. A combinação mínima válida de memória e valores de freqüências dos núcleos são 324MHz para cada uma. Nossa ferramenta pode realizar mudanças no comportamento da pós-execução antes que esta fase se inicie. Quando a *GPUtool* é usada apenas para recolher os dados de comportamento, não são feitas alterações nas freqüências e a redução de consumo de energia é controlada pelo gerenciador de energia da placa. Por outro lado, quando a *GPUtool* é responsável de reduzir a freqüência, a aplicação assim que termina tem ambas as freqüências definidas para os estados mais baixos.

Esse comportamento na pós-execução pode ser analisada em dois cenários. O primeiro, quando não existem mais aplicações começando antes do consumo de energia atingir o estado de *idle*, chamado de *Cenário Simples* e outro quando uma nova aplicação é iniciada após o término da primeira, chamado *Cenário Múltiplo*. Nas próximas subseções, esses cenários serão descritos e detalhados.

4.1. Cenário Simples

Este cenário é composto por uma fase de pós-execução que vai para o estado de idle, sem qualquer outro aplicativo em execução ou por iniciar. A GPUtool não define quaisquer freqüências neste cenário, apenas coleta os dados. Este cenário captura o comportamento do consumo de energia de pós-execução, sem os efeitos de execuções contínuas de aplicações GPU.

A figura 1 exemplifica com uma linha de tempo cada etapa do processo de execução e pós-execução de uma aplicação partindo do estado de *idle* até o seu retorno.

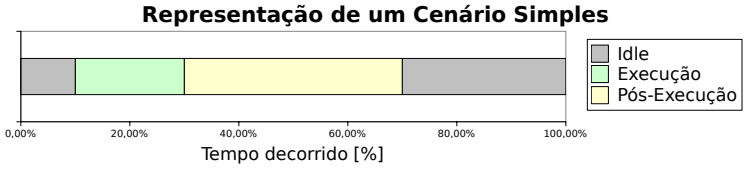

Figura 1. Linha de tempo de um cenário simples

4.2. Cenário Múltiplo

Este cenário consiste em mais de uma execução de uma aplicação em que durante a sua fase de pós-execução uma nova execução do mesmo ou de outra aplicação é iniciada. Neste caso, apenas a última pós-execução é capaz de atingir o estado de *idle*. É um comportamento similar a um escalonador, com tarefas diferentes para calcular na *GPU*. A metodologia de economia de energia é aplicada somente nas fases entre duas execuções seguidas e depois da última (área amarela na timeline). Este cenário captura o comportamento de consumo de energia na pós-execução com os efeitos de execuções contínuas aplicações na *GPU*.

A figura 2 exemplifica com uma linha de tempo cada etapa do processo de execução e pós-execução de uma aplicação partindo do estado de *idle* até o seu retorno, com execuções múltiplas intercaladas neste processo.

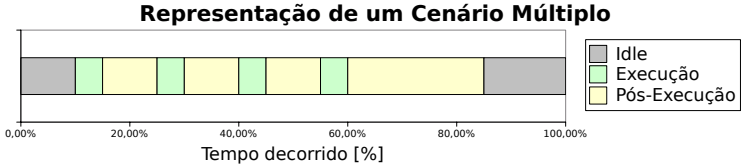

Figura 2. Linha de tempo de um cenário múltiplo

Este cenário também foi considerado, pois em ambos os casos não há perda de desempenho durante a execução. Como o nosso mecanismo não pode prever quantas aplicações serão executadas nem a duração de cada uma delas, o mecamismo precisa ser lançado antes da execução das aplicações para agir após o seu término ou na fase intermediária. Por não produzir impacto na performance (durante a execução), o nosso mecanismo pode economizar energia em todos os momentos que não há aplicações em execução, estando em uma pós-execução ou mantendo a freqüência mais baixa possível na fase de *idle*.

5. Resultados

Para descartar problemas de fabricação, os testes foram executados em 2 placas Nvidia Tesla K20M e 2 Nvidia Tesla K20C. Sem controlar o consumo de energia no final da execução dos processadores da *GPU*, esta permaneceu com altas frequências durante o período de *pós-execução*. É possível ver na figura 3 o comportamento da frequência de clock dos núcleos da *GPU* e o consumo de energia logo após o término da aplicação. Nas versões sem o nosso mecanismo o gasto energético permanece perto do último valor atingido antes do final da aplicação, voltando ao estado de idle só depois de alguns se-

gundos, enquanto que, com o mecanismo apresentado, tanto a freqüência de clock como o consumo de energia são reduzidos quase instantaneamente.

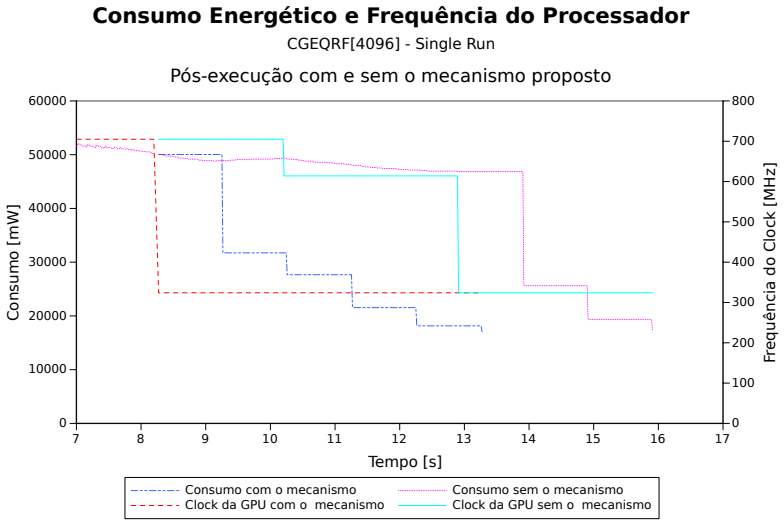

Figura 3. Consumo energético e freqüência do Clock de uma GPU durante a pós-execução sem a atuação do mecanismo proposto.

Nos experimentos, sem definir a freqüência do *clock* para o mínimo, o consumo atinge 31.25J, levando 7.099s para retornar ao estado de *idle*, por outro lado, ao estabelecer tal frequência para o mínimo pelo nosso mecanismo, o consumo de energia atinge 12.61J com tempo de retorno de 3.99s. Isso resulta em uma economia de energia de 59,65% na pós-execução de um teste de CGEQRF usando o *benchmark* MAGMA, com uma matriz de 4096x4096 elementos.

Um aumento no consumo na pós-execução não depende diretamente do tamanho da aplicação e sim de quantidade de energia que está sendo consumida no final da mesma. Aplicações maiores podem chegar ao final com um baixo consumo e terão um consumo baixo na fase de pós-execução. O perfil de energia deste comportamento no teste DGEQRF, também presente no *benchmark* MAGMA, pode ser visto na figura 4.

O alto consumo na pós-execução é devido ao tempo gasto enquanto o consumo de energia volta ao estado de *idle*. A figura 5 mostra alguns casos testados para o MAGMA, variando de precisão e tamanho da entrada de dados. Em todos os casos, usando o mecanismo proposto, o consumo de energia atinge o estado de idle mais rapidamente. Sem o uso do mecanismo é observada uma grande variação de tempo de regresso ao *idle*, enquanto usando o mecanismo, surge uma tendência de voltar em 4 segundos, independentemente da aplicação e do consumo de energia final. Em alguns casos, o tempo para atingir o estado de idle diminuiu em 20%, embora haja casos em que a este tempo foi reduzido em 50%, com uma redução considerável do consumo de energia na pós-execução.

O tamanho do exemplo analisado, não afeta diretamente a economia, uma vez que o que realmente importa para reduzir o consumo de energia é o consumo de energia ao final da aplicação. A figura 6 mostra a proporção de economia de energia entre o consumo na pós-execução com e sem o nosso mecanismo. Ao reduzir o tempo de retorno ao idle

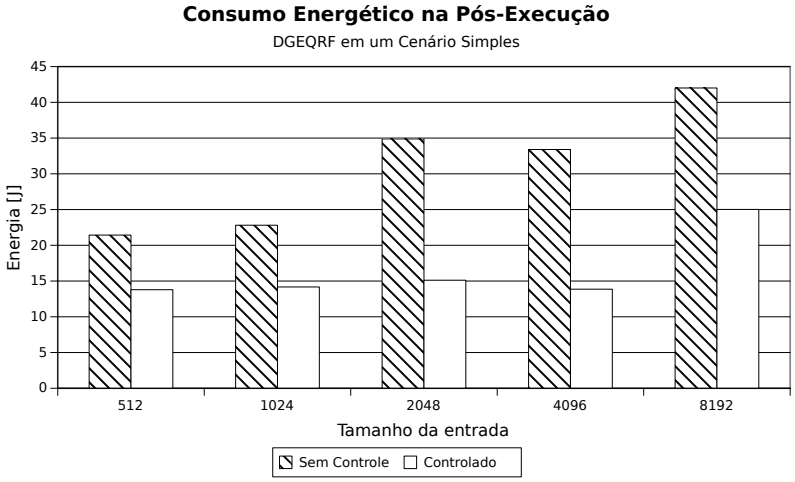

Figura 4. Consumo energético em um cenário simples, utilizando diversos tamanhos de dados.

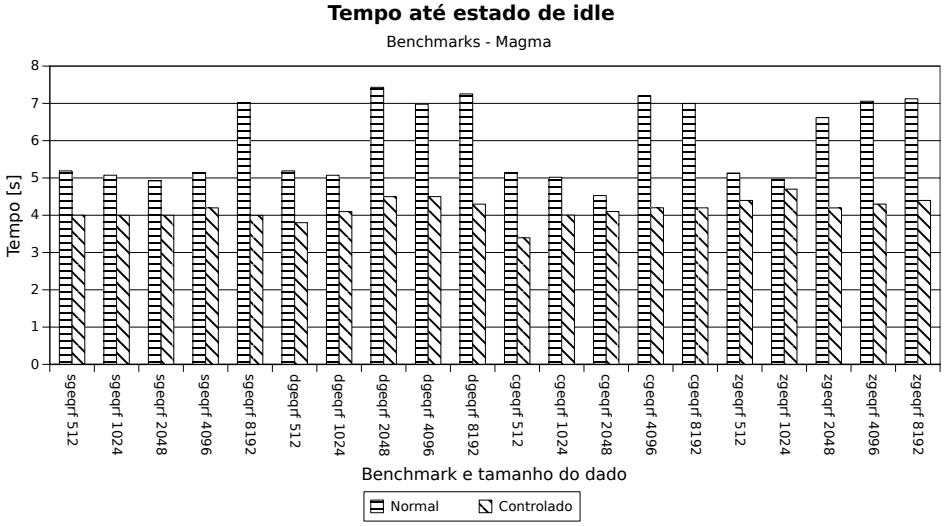

Figura 5. Comparação entre os tempo até o estado de idle com e sem o mecanismo na pós-execução.

definindo a frequência de *clock* para o mínimo, foram obtidos resultados com economias de cerca de 25%.

Também foi analisado o comportamento no cenário múltiplo, quando a frequência é modificada entre a duas execuções seguidas, agindo apenas na fase sem aplicação e na pós-execução final. O tempo de execução de cada aplicação não muda com o uso ou ausência do mecanismo proposto e a economia se obtem em sua maioria na pós-execução e uma pequena parte entre as execuções. A figura 7 mostra a proporção de economia de energia em cinco diferentes intervalos de tempo onde a aplicação é executada. Os intervalos testados são menores do que o tempo para o retorno ao estado de *idle*, caso contrário, os resultados teriam sido os mesmos que muitas execuções de um cenários simples uma vez que o *ganho real* neste cenário está presente no último retorno. Em

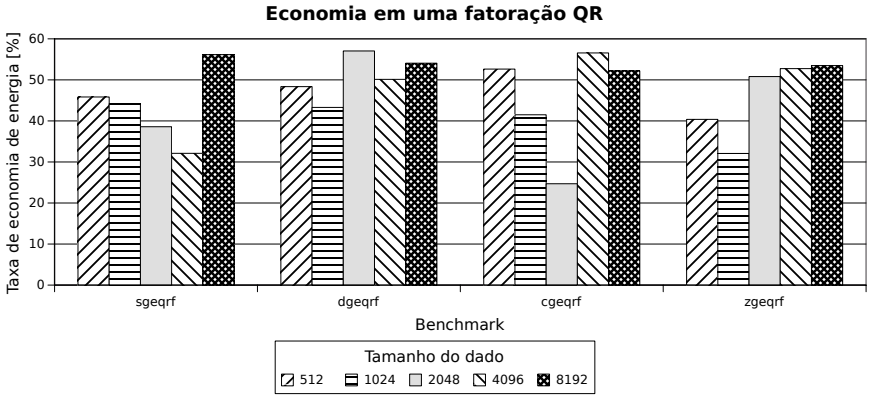

Figura 6. Comparação entre desperdício de energia com e sem o mecanismo na pós-execução de uma fatoração QR.

alguns casos, a energia economizada entre dois lançamentos de *kernels* são praticamente nulos, mas nunca maiores. Adicionando o fato de que o tempo total de execuções não aumenta, o nosso mecanismo pode ser utilizado em todos os tipos de cenários.

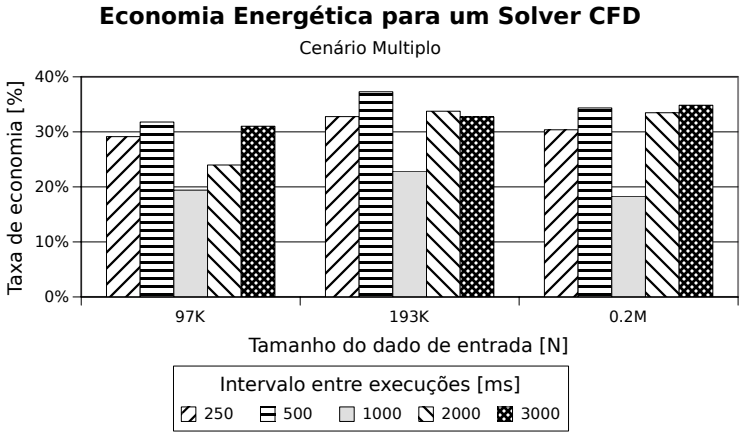

Figura 7. Comparação entre desperdício de energia com e sem o mecanismo na pós-execução de um Solver CFD agrupados pelo tempo entre execuções.

Os dados utilizados na figura 7 foram obtidos a partir de 5 execuções consecutivas. No pior dos casos para *Solver CFD*, foram obtidas economias de energia na pós-execução superiores a 15%.

6. Conclusões

Neste trabalho foi realizado um estudo de medição de consumo e economia de energia. Foram analisados o comportamento em placas *GPU* após a execução de uma aplicação e entre a execução sequêncial de múltiplas instâncias da mesma aplicação. Foram utilizados *benchmarks* e aplicações bem conhecidas com variações nos seus tamanhos de entrada, para gerar diferentes perfis de consumo de energia ao final das suas execuções.

Inicialmente a pós-execução demonstrou um desperdício de energia. Este desperdício de energia na placa existe tanto em aplicações com baixo ou alto consumo ao

atingir o final de sua execução. Esta questão energética poderia ser contornada usando escalonamento de freqüência, o que já é uma estratégia comum para economizar energia durante a execução de aplicações, mas não é usado para a poupança de energia na pós-execução. Até o momento não existem trabalhos anteriores realizados sobre o uso dessa estratégia para economizar energia na pós-execução. A economia de energia obtida neste trabalho não comparou o consumo total de energia de aplicação, porque o tempo gasto na realização podem variar de alguns segundos a várias horas e a economia de energia durante a execução não era o foco principal do mecanismo proposto. O foco deste trabalho foi economizar energia apenas na pós-execução das aplicações.

Usando o mecanismo proposto, uma economia de até 59,649% foi obtida além de otimizar o tempo de retorno ao idle na pós-execução. Até agora, este mecanismo é executado a partir de uma aplicação externa à placa, mas poderia ser implementada no gerenciador de energia da *GPU* para reduzir o desperdício de energia para qualquer aplicação na pós-execução. A fim de reduzir o consumo de energia de *GPUs* configuráveis, o mecanismo proposto pode ser um ponto de partida para reivindicar a atenção para uma área negligenciada com respeito ao desperdício de energia, o consumo de energia na pós-execução. Estes resultados obtidos pelo presente trabalho podem ser úteis no caminho para a era do *exascale* onde há esforços para economizar cada *watt* possível durante e após a computação.

Referências

Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Ltaief, H., Luszczek, P., and Tomov, S. (2009). Numerical linear algebra on emerging architectures: The plasma and magma projects. In *Journal of Physics: Conference Series*, volume 180, page 012037. IOP Publishing.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. IEEE.

Che, S., Sheaffer, J. W., Boyer, M., Szafaryn, L. G., Wang, L., and Skadron, K. (2010). A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–11. IEEE.

Chen, J., Li, B., Zhang, Y., Peng, L., and Peir, J.-k. (2011). Statistical gpu power analysis using tree-based methods. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–6. IEEE.

Ge, R., Vogt, R., Majumder, J., Alam, A., Burtscher, M., and Zong, Z. (2013). Effects of dynamic voltage and frequency scaling on a k20 gpu. In *2nd International Workshop on Power-aware Algorithms, Systems, and Architectures*.

Haidar, A., Tomov, S., Yamazaki, I., Solca, R., Schulthess, T., Dong, T., and Dongarra, J. (2008). Magma: A breakthrough in solvers for eigenvalue problems. `http://icl.utk.edu/magma/`.

Huang, S., Xiao, S., and Feng, W.-c. (2009). On the energy efficiency of graphics processing units for scientific computing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE.

Kasichayanula, K., Terpstra, D., Luszczek, P., Tomov, S., Moore, S., and Peterson, G. D. (2012). Power aware computing on gpus. In *Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on*, pages 64–73. IEEE.

Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. *Oakland, CA: Analytics Press. August*, 1:2010.

Lee, J., Sathisha, V., Schulte, M., Compton, K., and Kim, N. S. (2011). Improving throughput of power-constrained gpus using dynamic voltage/frequency and core scaling. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 111–120. IEEE.

Ma, K., Li, X., Chen, W., Zhang, C., and Wang, X. (2012). Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 48–57. IEEE.

Mei, X., Yung, L. S., Zhao, K., and Chu, X. (2013). A measurement study of gpu dvfs on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, page 10. ACM.

NVIDIA (2012). Whitepaper nvidia's next generation cuda compute architecture: Kepler gk110. `http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`.

NVIDIA (2013). Nvidia management library (nvml). `https://developer.nvidia.com/nvidia-management-library-nvml`.

Padoin, E. L., Pilla, L. L., Boito, F. Z., Kassick, R. V., Velho, P., and Navaux, P. O. (2012). Evaluating application performance and energy consumption on hybrid cpu+ gpu architecture. *Cluster Computing*, pages 1–15.

Tiwari, A., Laurenzano, M., Peraza, J., Carrington, L., and Snavely, A. (2012). Green queue: Customized large-scale clock frequency scaling. In *Cloud and Green Computing (CGC) , 1-3 November , 2012*, Xiangtan, Hunan, China. IEEE, IEEE.

Top500.org (2013). China's tianhe-2 supercomputer maintains top spot on 42nd top500 list. `http://www.top500.org/blog/lists/2013/11/press-release/`.

Ukidave, Y. and Kaeli, D. R. (2013). Analyzing optimization techniques for power efficiency on heterogeneous platforms. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1040–1049. IEEE.

Ukidave, Y., Ziabari, A., Mistry, P., Schirner, G., and Kaeli, D. (2013). Quantifying the energy efficiency of fft on heterogeneous platforms. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 235–244.

UTK, I. C. L. (2008). Matrix algebra on gpu and multicore architectures. `http://icl.utk.edu/magma/`.