Computação em Fluxos de Dados para Ambientes Paralelos e Heterogêneos

George Teodoro Renato Ferreira

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
{george,renato}@dcc.ufmg.br

Abstract. *As recentes mudanças no projeto de computadores tranformaram as plataformas de computação de alto desempenho em ambientes complexos equipados com diversos processadores paralelos e heterogêneos – como CPUs e GPUs. A utilização eficaz desses recursos continua sendo um grande desafio, pois essas plataformas adicionam novos compromissos e detalhes no projeto de aplicações. Nesse trabalho, apresentamos um conjunto de técnicas para maximizar a utilização de ambientes heterogêneos, que são desenvolvidas em nível de sistemas de execução e facilitam a exploração eficiente desses ambientes. Os resultados apresentam ganhos de aproximadamente 2× sobre as técnicas de escalonamento existentes na maioria dos experimentos.*

1. Introdução

Os avanços em arquitetura de computadores transformaram as plataformas distribuídas tradicionais de computação de alto desempenho em ambientes hierárquicos e heterogêneos, onde cada nó pode conter vários processadores heterogêneos. Essas mudanças resultaram em enormes benefícios devido aos ganhos na capacidade de computação, mas por outro lado também adicionaram complexidade no desenvolvimento de aplicações. A evolução nas soluções para desenvolvimento de software que execute em ambientes heterogêneos continua assumindo, na maioria dos casos, que o uso apenas de aceleradores é suficiente. Poucos trabalhos dedicaram-se a utilização colaborativa desses processadores [Luk et al. 2009], sendo que os mesmos assumem que as tarefas internas que compõem a execução de uma aplicação têm o mesmo desempenho relativo (*speedup*) entre os dispositivos.

Entretanto, argumentamos que o desempenho relativo na execução das tarefas que compõem a aplicação pode variar conforme as características de cada uma. Ou seja, uma mesma operação executada sobre dados distintos, utilizando CPU e GPU, pode apresentar desempenhos relativos diferentes. Essa observação reabre o problema de escalonamento eficiente em ambientes modernos heterogêneos, pois particionar tarefas sem considerar a adequação de cada uma delas aos processadores disponíveis como feito até então [Luk et al. 2009] pode resultar em subutilização dos recursos. Logo, nesse trabalho, propomos um conjunto de técnicas para execução eficiente em ambientes heterogêneos que consideram essas variações para maximizar a utilização desses ambientes.

As otimizações aqui propostas são validadas no contexto do modelo de programação filtro-fluxo e obtêm ganhos de aproximadamente 2× sobre as estratégias existentes. Foram implementadas com parte integrante de um sistema de suporte à execução, permitindo que diversas

aplicações se beneficiem. É importante ressaltar, no entanto, que as soluções propostas são genéricas e podem ser utilizadas em outros modelos de programação.

A seguir listamos algumas das contribuições desse trabalho: (i) extensão do modelo filtro-fluxo para plataformas paralelas e heterogêneas [Teodoro et al. 2008a, Teodoro et al. 2009a]; (ii) escalonamento de tarefas sensível a variação de desempenho [Catalyurek et al. 2010, Teodoro et al. 2009b]; (iii) política para controle de fluxos de dados em ambientes heterogêneos [Catalyurek et al. 2010, Teodoro et al. 2011, Teodoro et al. 2010]; (iv) metodologia para estimar desempenho relativo de tarefas em processadores heterogêneos [Teodoro et al. 2011, Teodoro et al. 2010]. Dentre as publicações destacamos o artigo completo [Teodoro et al. 2010] que recebeu o prêmio de **Best Student Paper** patrocinado pela Google Inc. na conferência HPDC 2010.

2. Anthill

Anthill [Ferreira et al. 2005] é um ambiente de execução que implementa o modelo de programação filtro-fluxo. Aplicações nesse modelo são desenvolvidas como um conjunto de unidades de processamento, chamadas de filtros, que comunicam-se através de fluxos de dados unidirecionais. Em tempo de execução, diversas instâncias de cada filtro podem ser criadas através do mecanismo de cópias transparentes. O desenvolvimento de aplicações nesse modelo conduz naturalmente a dois tipos de paralelismo: dados e tarefas. O paralelismo de dados se dá pela criação de cópias transparentes de um filtro e divisão dos dados de entrada entre elas. O paralelismo de tarefas, por sua vez, ocorre a partir da execução concorrente dos diversos filtros que compõem o fluxo da aplicação. Essa exploração de paralelismo simultaneamente em múltiplas dimensões favorece a criação de um grande número de tarefas potencialmente independentes, o que permite utilização concorrente de diversas unidades de processamento.

3. Anthill orientado a eventos

Na interface de programação original do Anthill os filtros que compõem aplicação são implementados por meio de três funções: *initFilter*, *processFilter* e *finalizeFilter*, sendo cada uma delas invocada uma vez durante a execução. Assim, o programador tradicionalmente escreve a função *processFilter* como um laço sobre os fluxos de entrada, recebendo e processando dados, até que não exista mais trabalho a ser feito. A estrutura da função de processamento deixa a cargo do programador toda a responsabilidade de lidar com: (i) sincronização no recebimento e envio de dados; e (ii) dependências de dados entre fluxos.

Anthill orientado a eventos [Teodoro et al. 2008a] foi proposto como alternativa à interface de programação tradicional, com intuito de prover uma nova abstração para desenvolvimento de aplicações que solucione os problemas destacados. Nessa interface, o controle de fluxos de dados é movido para o ambiente de execução. Assim, o programador passa a prover apenas as funções de processamento relativas ao domínio da aplicação – tratador de eventos, invocadas quando existirem dados disponíveis para processamento. O sistema, por sua vez, controla recebimento/envio de dados e escalona eventos para execução quando as dependências de dados são satisfeitas. Esse esquema favorece a exploração de ambientes *multi-core* através da execução concorrente de eventos internos dos filtros nos dispositivos disponíveis.

4. Anthill orientado a eventos em ambientes heterogêneos

A capacidade de explorar ambientes heterogêneos se deu com a extensão da interface orientada a eventos [Teodoro et al. 2009a]. Essa nova interface permite ao programador escrever a função que processa eventos para múltiplos dispositivos, como CPU e GPU. O ambiente de execução, por sua vez, verifica quais dispositivos estão disponíveis e dispara a execução de eventos nos processadores para os quais existem tratadores implementados. Isso permite que filtros utilizem diversos dispositivos através da execução concorrente de eventos nesses processadores.

Em Anthill orientado a eventos para ambientes heterogêneos o espaço de decisões sobre escalonamento de eventos é mais abrangente. O assinalamento de eventos em cada filtro é feito sob demanda entre os processadores e eles compartilham uma fila de eventos prontos para executar. A política de escalonamento utilizada é DDFCFS (*demand-driven, first-come, first-served*), que seleciona o evento há mais tempo enfileirado quando um processador fica disponível.

5. Escalonamento de eventos sensível a variação de desempenho

O escalonamento de tarefas em ambientes heterogêneos, como discutido, deve classificar as tarefas conforme o desempenho para cada processador e utilizar essa informação para fazer um assinalamento eficiente das mesmas. A materialização dessas ideias em nosso trabalho é dada através de uma política para escalonamento de eventos chamada *demand-driven dynamic weighted round-robin* (DDWRR) [Teodoro et al. 2009b]. Essa política escalona eventos prontos para executar em um filtro nos processadores utilizados sob demanda durante a execução. Assim, quando um processador torna-se disponível, DDWRR seleciona o evento considerando o desempenho relativo (speedup) estimado dos eventos para cada processador. A estimativa desse speedup acontece durante a execução para cada um dos processadores utilizados, baseando-se em um dispositivo base e através da metodologia discutida na Seção 6.

Finalmente, para cada processador utilizado por um filtro, DDWRR cria uma fila de eventos prontos para executar, ordenada decrescentemente em termos do desempenho relativo previsto em cada dispositivo. Eventos com o mesmo desempenho relativo são enfileirado em ordem crescente de acordo com o melhor desempenho do evento nos outros processadores do sistema. Quando da escolha do evento a ser executado, DDWRR seleciona o primeiro na fila do processador que executará o evento. Também é importante salientar que DDWRR não necessita de um valor exato do desempenho relativo, mas somente de uma estimativa suficiente para ordenar corretamente os eventos.

6. Estimador de desempenho

O assinalamento de tarefas aos processadores é realizado com base em uma estimativa dos desempenhos relativos de cada uma delas. Assim, propomos uma metodologia integrada ao ambiente de execução que automatiza essa tarefa. Esse processo é baseado em duas fases: (i) treinamento, onde a aplicação é executada contra uma carga representativa nos dispositivos existentes (eg. CPU e GPU); (ii) previsão, que estima o desempenho relativo das tarefas nos dispositivos utilizados (feita em tempo de execução). Nessa fase, os k registros mais próximos na base de conhecimento gerada durante a primeira fase são retornados (baseando-se em distância entre os

parâmetros) e o desempenho relativo da nova tarefa é estimada como a média do desempenho relativo dos registros retornados.

A metodologia foi avaliada utilizando seis aplicações com implementação para CPU e GPU: Black-Scholes; N-body; Heart Simulation; k-NN; Eclat; Neuroblastoma - NBIA. Detalhes sobre as aplicações e a avaliação podem ser obtidos no texto completo dessa tese. A avaliação da metodologia proposta obteve precisão surpreendentemente alta, com erro menor que 14% e média de 8,52% para as 6 aplicações. A mesma metodologia foi empregada para estimar tempo de execução na CPU das mesmas tarefas da carga de trabalho anterior, mas os erros nesse caso são muito maiores: máximo de 102,6% e média de 62,6%. Essa avaliação empírica é interessante por pelo menos duas razões: (i) mostra que políticas de assinalamento que baseiam-se em estimativa de desempenho relativo estão em vantagem devido a melhor precisão; (ii) o desempenho relativo pode ser utilizado para estimar tempo de execução de uma aplicação executando em dispositivos diferentes, contudo com melhor precisão.

7. Controle de fluxos de dados em ambientes heterogêneos

A utilização eficiente de ambientes distribuídos heterogêneos é ainda mais desafiador quando as máquinas utilizadas são diferentes - a exemplo de quando algumas são equipadas com CPU e GPU e outras possuem apenas CPU. Nesse contexto, as seguintes questões ganham destaque e devem ser consideradas: (i) desbalanceamento de carga entre as cópias de um filtro que são replicadas em máquinas com processadores diferentes; (ii) necessidade de selecionar os dados que maximizam o desempenho dos filtros no lado do envio, de modo que o receptor utilize os processadores adequadamente.

Esses problemas são abordados através da criação de uma política de comunicação entre filtros que é chamada de *on-demand dynamic selective stream (ODDS)* [Teodoro et al. 2010]. Essa política é implementada baseando-se em dois algoritmos: Algoritmo dinâmico de adaptação de filas (*Dynamic Queue Adaptation Algorithm (DQAA)*) e Algoritmo de seleção de *buffers* de dados (*Data Buffer Selection Algorithm (DBSA)*). Nesse tipo de política sob demanda, sempre que um *buffer* de dados é enviado por um filtro ele é enfileirado no lado do envio, sendo enviado na prática quando uma instância do filtro receptor requisitar dados.

O DQAA é executado no lado do receptor de um fluxo e verifica a taxa na qual cada instância de filtro consome dados (processa eventos), modificando a quantidade de requisições ($streamRequestsSize$) por dados em tempo de execução. O objetivo é manter o menor volume de dados no receptor que seja suficiente para manter a ocupação integral dos processadores da máquina, evitando assim desbalanceamento de carga e subutilização dos recursos. O DBSA, por sua vez, é executado nas cópias do filtro no lado do envio do fluxo e seleciona os dados que maximizarão o desempenho dos processadores no lado da recepção. O algoritmo implementado na seleção é o mesmo empregado para o escalonamento DDWRR.

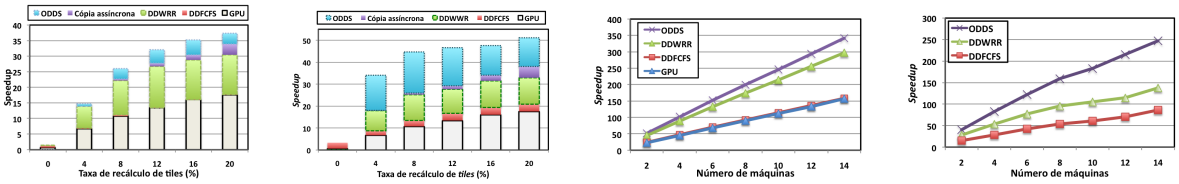
8. Resultados experimentais

Como exemplo utilizamos uma aplicação real (NBIA) usada para a avaliação do prognóstico de neuroblastoma, um tipo de câncer que acomete principalmente crianças. A entrada da aplicação é um conjunto de *slides* de microscópio digitalizadas, que são particionados em pedaços (*tiles*)

menores que podem ser processados independentemente. Os *tiles* são representados em diversas resoluções utilizando uma estratégia piramidal. A análise para cada *tile* começa na resolução mais baixa e é interrompida quando o mesmo for corretamente classificado ou processado na resolução mais alta disponível. NBIA foi executada utilizando 26.742 *tiles* que são representados em duas dimensões: 32×32 e 512×512 $pixels$. Durante a avaliação variamos a taxa de recálculo de $tiles$: percentual de $tiles$ calculados na segunda resolução. É importante ressaltar que outras aplicações foram avaliadas e os resultados são apresentados no texto completo da tese.

Configuração dos experimentos: A avaliação utiliza duas configurações de *clusters*: (i) *cluster* homogêneo com 14 máquinas equipadas com CPU Intel dual-core e GPU NVidia GeForce 8800GT; (ii) *cluster* heterogêneo com as mesmas 14 máquinas, entretanto as GPUs são desligadas em 7 delas. Dessa maneira, criamos heterogeneidade entre as máquinas, pois 7 nós são equipados com CPU e GPU e outros 7 têm apenas a CPU dual-core. ODDS foi comparada a outras duas políticas de comunicação sob demanda, sendo que ambas mantêm um número fixo de requisições de dados entre filtros ligados no fluxo ($streamRequestsSize$): (i) DDFCFS seleciona, no envio, o *buffer* de dados enfileirado há mais tempo e na recepção faz a mesma escolha do próximo evento a processar; (ii) DDWRR seleciona, no envio, o *buffer* de dados enfileirado há mais tempo e na recepção utiliza o escalonamento que observa variações no desempenho relativo. Os *speedups* são calculados com base na versão sequencial.

Resultados: Na Figura 1(a) apresentamos os resultados de desempenho para o caso base do *cluster* homogêneo (uma máquina equipada com CPU e GPU). Primeiramente, nota-se que a utilização de um núcleo CPU adicional em cooperação com a GPU e a política DDFCFS contribui pouco para o desempenho em relação a versão apenas GPU, sendo que DDWRR quase dobra o desempenho da GPU. ODDS superou o desempenho de DDWRR, inclusive o caso base homogêneo, o que deve-se em razão da sua habilidade em selecionar *buffers* de dados que maximizam o desempenho do processador receptor. Isso ocorre mesmo em uma máquina por que os *buffers* são enfileirados no lado do envio para ambas políticas, entretanto, ODDS seleciona os dados que maximizam o desempenho dos processadores receptores no momento do envio.



(a) Base homogêneo. (b) Base heterogêneo. (c) *Cluster* homogêneo. (d) *Cluster* heterogêneo.

Figura 1. Avaliação das políticas de escalonamento - melhor visualizado ao ampliar.

Os resultados no caso base heterogêneo (um computador equipado com CPU dual-core e GPU e o segundo com apenas uma CPU dual-core) são apresentados na Figura 1(b). Primeiramente, quando comparamos os resultados do caso base homogêneo e heterogêneo, vistos nas Figuras 1(a) e 1(b), respectivamente, nota-se que DDFCFS e DDWRR alcançam pequenos ganhos de desempenho com a adição do segundo nó de computação equipado com CPU apenas, enquanto a utilização da segunda máquina e ODDS melhora significativamente o desempenho.

A seguir, apresentamos o desempenho de NBIA à medida em que aumentamos o número de máquinas para cada caso base. No caso base heterogêneo mantivemos a relação de 50% de

máquinas equipadas com CPU e GPU e 50% somente CPU. Na Figura 1(c), apresentamos os *speedups* de NBIA no *cluster* homogêneo. Os resultados mostram que DDFCFS tem quase o mesmo desempenho da versão somente GPU de NBIA, enquanto DDWRR foi capaz de aproximadamente dobrar o desempenho da versão GPU. Adicionalmente, ODDS é 15% mais rápido que DDWRR até mesmo no ambiente homogêneo. Os resultados no caso heterogêneo, Figura 1(d), apresentam ODDS com ganhos ainda maiores — 2× melhor que DDWRR.

9. Conclusões

Esse trabalho aborda o problema de exploração eficiente de ambientes *multi-core* e heterogêneos, utilizando como caso de uso o modelo de programação filtro-fluxo. Os resultados obtidos mostram que a adequada utilização em cooperação de processadores heterogêneos, como CPU e GPU, pode melhorar efetivamente o desempenho de aplicações intensivas em computação. As observações feitas nesse trabalho reabrem a discussão sobre escalonamento em ambientes heterogêneos modernos, pois pouco benefício havia sido alcançado até então com a utilização cooperativa desses dispositivos. As técnicas criadas nesse trabalho também podem ser aplicadas em outros modelos de programação. O software desenvolvido nesse trabalho encontra-se em uso em diversos projetos e pode ser acessado através do seguinte endereço em construção: $http://www.speed.dcc.ufmg.br/anthill/$.

Referências

Catalyurek, U., Ferreira, R., , Sachetto, R., and Teodoro, G. (2010). *Scientific Computing with Multicore and Accelerators*, chapter Dataflow Frameworks for Emerging Heterogeneous Architectures and Their Applications to Biomedicine. Chapman & Hall / CRC Press.

Ferreira, R., Jr., W. M., Guedes, D., Drummond, L., Coutinho, B., Teodoro, G., Tavares, T., Araujo, R., and Ferreira, G. (2005). Anthill:a scalable run-time environment for data mining applications. In *SBAC-PAD*.

Luk, C.-K., Hong, S., and Kim, H. (2009). Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *42nd Int. Symp. on Microarchitecture (MICRO)*.

Teodoro, G. (2006). Suporte a fluxos de trabalho de aplicacões intensivas em dados (**Best Master Thesis**). CTD-SBAC/WSCAD. 2006.

Teodoro, G., Fireman, D., Guedes, D., Jr., W. M., and Ferreira, R. (2008a). Achieving multilevel parallelism in the filter-labeled stream programming model. *Inter. Conf. on Parallel Processing (ICPP)*, 0:287–294.

Teodoro, G., Hartley, T. D. R., Catalyurek, U., and Ferreira, R. (2011). Optimizing dataflow applications on heterogeneous environments — accepted for publication. *Cluster Computing: The Journal of Networks, Software Tools and Applications*.

Teodoro, G., Hartley, T. D. R., Catalyurek, U. V., and Ferreira, R. (2010). Run-time optimizations for replicated dataflows on heterogeneous environments. In *Proc. of the 19th ACM Inter. Symposium on High Performance Distributed Computing (HPDC)* - **Best student paper**.

Teodoro, G., Sachetto, R., Fireman, D., Guedes, D., and Ferreira, R. (2009a). Exploiting computational resources in distributed heterogeneous platforms. In *21st Intel Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*.

Teodoro, G., Sachetto, R., Sertel, O., Gurcan, M., Jr., W. M., Catalyurek, U., and Ferreira, R. (2009b). Coordinating the use of GPU and CPU for improving performance of compute intensive applications. In *IEEE Int. Conf. on Cluster Computing*.

Teodoro, G., Tavares, T., Ferreira, R., Kurc, T., Meira, W., Guedes, D., Pan, T., and Saltz, J. (2008b). A run-time system for efficient execution of scientific workflows on distributed environments. *International Journal of Parallel Programming*, 36.