Aceleração de Computações Científicas com Processadores Heterogéneos

Luís Picciochi Oliveira and João M. Lourenço*

CITI — Departamento de Informática,
Universidade Nova de Lisboa, Portugal
p110390@fct.unl.pt Joao.Lourenco@di.fct.unl.pt

Resumo Actualmente o mercado residencial de computadores inclui não só multiprocessadores com diversos núcleos (CPUs) como também placas gráficas (GPUs) cuja capacidade de processamento tem evoluído a um ritmo exponencial. Este poder computacional pode ser utilizado para outros fins para além do processamento gráfico, tal como o processamento de algoritmos comuns em computação científica. Neste artigo é apresentada, discutida e avaliada a *framework Cheetah*, uma *framework* que distribui programas computacionalmente exigentes sobre uma rede de CPUs e GPUs. Um programador que utilize a *Cheetah* apenas necessita de especificar o seu programa como um conjunto de *kernels* OpenCL, relegando para a *framework* a distribuição destes pelas unidades de processamento disponíveis. O programa pode assim escalar à medida que são adicionados novos recursos computacionais, sem quaisquer esforços adicionais de adaptação ou recompilação. Os testes realizados demonstraram a capacidade de a *framework* providenciar acelerações até duas ordens de grandeza com um esforço de desenvolvimento reduzido, mesmo quando na presença de recursos computacionais limitados.

Keywords: Middleware; Computação Heterogénea; Computação Científica; GPGPU; OpenCL; Job Scheduling

1 Introdução

O crescimento contínuo das capacidades computacionais das GPUs, actualmente com performances ao nível do TeraFlop (TF), tem atraído o interesse de áreas da computação não directamente relacionadas com processamento gráfico [21]. No primeiro semestre de 2011 já era possível adquirir uma GPU com uma capacidade de processamento de 1,35 TF/s por cerca de € 200. Com 33 destas placas (cerca de € 6600) é possível atingir um poder computacional equivalente à 207^a posição da lista de supercomputadores do TOP 500 [1].

* Este trabalho foi patrocinado parcialmente pela Sun Microsystems sob a *Sun Worldwide Marketing Loaner Agreement #11497* do Centro de Informática e Tecnologias da Informação (CITI) e pela Fundação para a Ciência e Tecnologia (FCT/MCTES) no contexto dos programas de investigação PTDC/EIA/74325/2006, PTDC/EIA-EIA/108963/2008 e PTDC/EIA-EIA/113613/2009.

Diversas aplicações científicas tanto de nível académico como industrial têm sido adaptadas para utilizar GPUs ou clusters de GPUs, sendo comum a obtenção de melhorias de desempenho muito significativas [15]. Nestes casos é frequente que os esforços de desenvolvimento se concentrem na implementação de optimizações de baixo nível para modelos específicos de GPUs. Este nível de detalhe é um dos entraves à adaptação de novos algoritmos para estas arquitecturas. Recorrendo a uma linguagem de programação independente do hardware e a uma *framework de middleware* capaz de distribuir tarefas sobre um conjunto heterogéneo de Unidades de Processamento (PUs), podem obter-se melhorias de desempenho significativas com custos de desenvolvimento e adaptação muito limitados. Evita-se assim a selecção prévia do hardware durante a fase de concepção dos programas, bem como o desenvolvimento de complexas optimizações de código, mantendo-se a obtenção de melhorias de desempenho significativas com a agregação de mais PUs à infraestrutura computacional.

Neste artigo apresentamos a Cheetah, uma *framework* genérica desenhada para clusters de computadores heterogéneos. A Cheetah considera as GPUs como unidades de processamento de primeira classe a par dos CPUs, sendo que os trabalhos (*jobs*) submetidos e processados nesta *framework* são independentes da infraestrutura hardware existente. Cada job submetido inclui os dados a processar, um *kernel* OpenCL [18] com a especificação do algoritmo e um conjunto de atributos, utilizados pela *framework* para optimizar a atribuição de jobs às PUs.

Na próxima Secção é apresentada uma breve introdução ao OpenCL, seguindo-se uma descrição do desenho e arquitectura da Cheetah na Secção 3; na Secção 4 é apresentado o trabalho de validação e avaliação de desempenho realizado; na Secção 5 discute-se o trabalho relacionado e finalmente, na Secção 6, são apresentadas algumas conclusões.

2 *Open Computing Language* (OpenCL)

A OpenCL [18] é uma *framework* de computação desenhada para possibilitar o desenvolvimento de software para plataformas e ambientes heterogéneos. Com uma única linguagem é possível criar um programa capaz de executar em diferentes tipos de multiprocessadores como CPUs, GPUs, o IBM CellBE e FPGAs (*Field Programmable Gate Arrays*). Um programa OpenCL é composto pelo código *host* e pelo código de *runtime*. O código *host* executa no CPU anfitrião e prepara a execução do código *runtime*, seleccionando o dispositivo e transmitindo o *kernel* OpenCL e os dados a serem processados para a memória desse dispositivo. Posteriormente, o código de *runtime* (doravante designado *kernel*) executa no dispositivo seleccionado.

O código *host* deve especificar o número de vezes a executar cada kernel através da definição do número de *work-items*. Para o programador, todos os *work-items* são processados em paralelo. Dependendo das propriedades e capacidades do hardware, os itens poderão ser ou não efectivamente processados em paralelo ou sequencializados pelo suporte OpenCL. Os *work-items* são configurados ao longo de um espaço de índices uni-, bi- ou tri-dimensional (*NDRange*)

que indica a afinidade entre *work-items*. A configuração da NDRange pode influenciar significativamente o desempenho em diferentes dispositivos de hardware.

3 Cheetah — Uma *Framework* para Computação em Ambientes Heterogéneos

A Cheetah [20] é uma *framework* de computação genérica que permite utilizar um conjunto de recursos heterogéneos através de uma aproximação do tipo *Single System Image*. A *framework* considera tanto CPUs como GPUs como processadores de primeira classe, cujas especificidades são geridas pelo sistema e ocultadas ao programador/utilizador.

3.1 Desenho e Arquitectura

A *framework* de computação Cheetah é composta por um componente específico do domínio e alguns componentes independentes do domínio, ilustrados na Fig. 1.

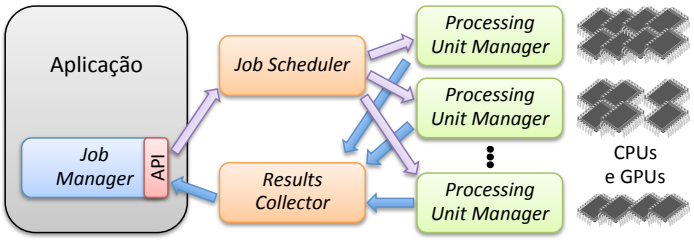

Figura 1: Principais componentes da *framework* e fluxo de tratamento dos jobs.

O *Job Manager* (JM) é o único componente dependente do domínio, estando associado à aplicação que utiliza a *framework*, e é responsável pela definição, criação e submissão de jobs para execução na *framework*. O JM deve ser adaptado ou desenvolvido de raiz para cada aplicação, potencialmente com a intervenção de um especialista do domínio. As submissões de jobs são atendidas pelo *Job Scheduler* (JS), que os atribui às PUs disponíveis. Todas as PUs disponíveis num nó computacional são geridas por um único *Processing Unit Manager* (PUM). Depois de processado, o resultado de cada job é enviado para um *Results Collector* (RC), de onde será enviado de volta à aplicação. A aplicação pode criar novos jobs e submetê-los para processamento sem ter de recolher os dados de jobs anteriores nem esperar pelo seu término.

3.2 Configurações Suportadas

A *framework* Cheetah pode ser utilizada como um sistema centralizado ou distribuído, com todos os seus componentes concentrados numa única máquina

ou repartidos pelos nós de um cluster. Para utilizar a Cheetah num cluster, executa-se uma instância do PU-M em cada nó para permitir distribuir a carga de processamento pelas PUs desse nó.

3.3 Algoritmos de Escalonamento

O *Job Scheduler* pode recorrer a diversos algoritmos de escalonamento para melhor atribuir os jobs recebidos aos recursos disponíveis. Foi utilizada sempre uma estratégia de *dispatching* simples, seguindo um algoritmo do tipo *First-Come First-Served*, sem qualquer alteração de prioridades nem reordenação dos jobs a processar. A nível de políticas de alocação, foram desenvolvidas duas políticas principais: a primeira, *Round-Robin*, e a segunda, uma política configurável e baseada em pontuações (*Score-Based*). Com a política de alocação *Round-Robin*, as PUs são seleccionadas iterativamente, uma após outra, sem tomar em conta as suas propriedades ou capacidades de processamento. A política de alocação *Score-Based* tem como objectivo melhorar a utilização global da infraestrutura computacional. Para cada job é calculada uma lista de pontuações, uma por PU. Seguidamente é seleccionada a PU com a melhor pontuação para o job em questão. Foram implementadas duas variantes desta política:

- A *configuração fixa* atribui a mesma pontuação a cada PU, independentemente do job. Esta pontuação reflecte uma estimativa do desempenho computacional (FLOPS) dessa PU, pesada com a latência introduzida pela transferência dos dados pelo canal de comunicação entre o JS e a PU.
- A *configuração adaptativa* considera o histórico de tempos de execução para jobs da mesma natureza. Para cada job de natureza distinta, este algoritmo apresenta três fases com comportamentos distintos: primeiro, os jobs são atribuídos às PUs segundo uma política *round-robin* para que pelo menos um job de cada tipo seja executado em cada PU. Seguidamente, comporta-se de forma semelhante à da configuração fixa, seleccionando as PUs com uma frequência proporcional ao seu desempenho, medida durante o arranque do sistema. À medida que o JS recebe informações acerca dos tempos de processamento observados pelas PU-Ms para jobs da mesma natureza, essa informação é guardada como uma média pesada ao longo do tempo e considerada nas futuras selecções de PUs.

4 Avaliação

Foram utilizados dois algoritmos para testar e validar a *framework*: um gerador de fractais de Mandelbrot [16], com um algoritmo simples, altamente paralelizável e apropriado para execução em GPUs; e um sistema de identificação de genes potencialmente alvos de selecção natural [11], com um algoritmo complexo, do tipo MIMD, mais adequado ao processamento em CPUs.

O gerador de fractais computa um detalhe de uma zona do fractal de Mandelbrot. A quantidade de dados a processar é directamente proporcional ao tamanho

da imagem final pretendida. O *peso* da computação, influenciado directamente pelo número de iterações necessárias, é parametrizável pelo número de cores pretendidas para a imagem final.

O algoritmo de identificação de genes detecta genes associados a características de indivíduos de uma população tais como cor de pele e pêlo, genes susceptíveis a vírus como o VIH, genes que influenciam a quantidade de carne em gado para alimentação humana, bem como outros factores diferenciadores. Este algoritmo é relativamente complexo e exigente em termos de memória quando são simulados elevados números de genes.

4.1 Resultados Experimentais

A *framework* Cheetah foi implementada em C, recorrendo a uma camada de comunicação com MPI [17]. O algoritmo a processar, definido nos jobs, consiste num kernel OpenCL, a ser submetido a um dispositivo OpenCL.

Ambos os algoritmos (tanto o gerador de fractais de Mandelbrot como o identificador de genes) foram testados sobre o mesmo conjunto de recursos computacionais. Na Tabela 1 podem observar-se as características das 10 PUs usadas, agregadas em 5 nós. Na coluna da performance relativa mostra-se quantas vezes mais rápido é cada dispositivo OpenCL relativamente ao mais lento (Intel Core 2 6420). As barras horizontais em segundo plano ilustram esses valores através de uma representação visual.

Tabela 1: Detalhes do hardware (barras horizontais separam PUs em nós diferentes).

Designação	Perf. Rel.	Unid. Comp.	Clock	RAM
SunFire X4600 M2	2,91	16	1.00GHz	32GB
Intel Core 2 6420	1,00	2	2.13GHz	2GB
NVIDIA Quadro FX 3800	12,25	24	1.20GHz	1GB
Intel Xeon E5506	2,11	4	2.13GHz	12GB
NVIDIA Quadro FX 3800	11,98	24	1.20GHz	1GB
NVIDIA Tesla C1060	13,53	30	1.30GHz	4GB
NVIDIA Tesla C1060	13,53	30	1.30GHz	4GB
Intel Core i5 650	2,62	4*	3.20GHz	4GB
NVIDIA GeForce GTX 480	88,21	15	1.40GHz	1.5GB
Intel Xeon 5150	2,03	4*	2.66GHz	2GB

*** Processadores com tecnologia HyperThreading**

Foram efectuados testes preliminares, locais, em cada máquina, por forma a determinar a configuração mais adequada para os jobs a lançar sobre a *framework*. Seguidamente foram realizadas avaliações de desempenho de execuções

distribuídas, utilizando-se todas as PUs disponíveis. Nos testes locais foi utilizada uma única instância de cada componente no computador que aloja a PU respectiva. Nos testes distribuídos foi lançado um PU-M em cada nó e foram lançados um Job Manager, um Job Scheduler e um Results Collector num nó adicional com um processador Intel Core 2 Duo T6400 a 2.0 GHz e 4 GB de RAM. As máquinas comunicaram através de uma VPN sobre uma LAN de 100 Mbit.

Nos gráficos seguintes as barras referem-se sempre ao tempo que medeia entre a submissão do primeiro job pelo Job Manager até à recepção dos resultados de todos os jobs. As barras azuis/escuras referem-se a execuções em que apenas foram utilizados os CPUs; as barras verdes/claros referem-se a tempos observados quando foram utilizados tanto os CPUs como as GPUs disponíveis.

Gerador de Fractais de Mandelbrot Recorreu-se à *framework* para distribuir este algoritmo sobre as diversas PUs e mediu-se o tempo despendido no cálculo de uma imagem de 100 Megapixel (10240×10240 pixels). Esta imagem foi particionada em faixas horizontais de tamanho igual, sendo que cada job processou uma dessas faixas. Os testes locais, preliminares, conduziram à configuração de cada job com uma NDRange bidimensional de 512×512 itens e 8×8 itens por grupo. Para a configuração adaptativa da política de alocação de trabalho baseada em pontuações, os testes foram precedidos por uma fase de *profiling* para cada PU com um job que gerou uma imagem 100× menor. Na Fig. 2 apresentam-se os tempos de execução observados na geração da imagem, ao utilizar as configurações fixa e adaptativa.

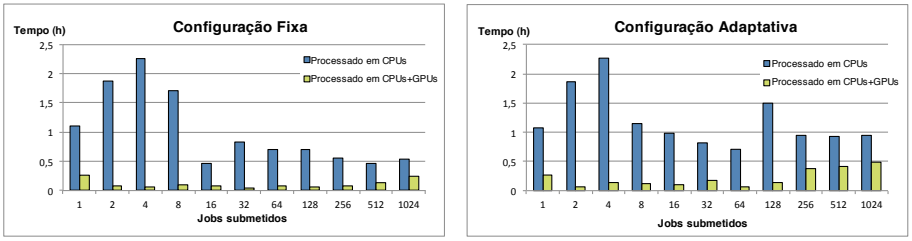

Figura 2: Tempos observados em testes com o fractal de Mandelbrot.

Os resultados obtidos mostram que a utilização de mais jobs (e mais pequenos) levou a um melhor desempenho do sistema. Isto deve-se ao facto de, com mais jobs, as regiões computacionalmente mais exigentes do fractal serem divididas em faixas mais pequenas, originando um grão mais fino e resultando num melhor balanceamento de carga. A introdução de GPUs melhorou sempre o desempenho global do sistema. A aproximação adaptativa apresentou tempos de execução médios ligeiramente superiores (piores). Isto sugere que o job inicial, de *profiling*, apesar de ser representativo da computação *média* global para toda a região, não considera as diferenças reais entre os vários jobs e acabou

por induzir o algoritmo adaptativo em erro. Adicionalmente, é visível que, para a aproximação fixa, há uma melhoria de desempenho significativa dos 8 para os 16 jobs e com a aproximação adaptativa, esta diferença é visível dos 4 para os 8 jobs. É também visível uma variação de desempenho com a utilização de mais jobs, com picos de tempo aos 32 e 128 jobs com a aproximação fixa e adaptativa, respectivamente.

Depois de uma análise cuidada aos relatórios gerados pela *framework* nas execuções deste programa, foi possível concluir que os primeiros jobs submetidos foram enviados para as PUs mais rápidas, enquanto que os subsequentes foram enviados para as PUs mais lentas. Sabe-se que a geração das áreas representadas a preto é computacionalmente muito mais intensiva que as demais. Os primeiros jobs, que são mais leves que os últimos, são portanto atribuídos às PUs mais rápidas. Isto tem o efeito de sobrecarregar as máquinas mais lentas enquanto que as mais rápidas foram sub-utilizadas.

Para confirmar esta conjectura foi realizada uma nova série de testes com o gerador de fractais. Nesta série mantiveram-se as condições de teste da série inicial, mas a ordem de processamento da imagem foi invertida. Assim, os primeiros jobs submetidos corresponderam à zona inferior da imagem, computacionalmente mais exigentes, enquanto que os últimos corresponderam à zona superior, computacionalmente mais leves. Os tempos observados são mostrados na Fig. 3.

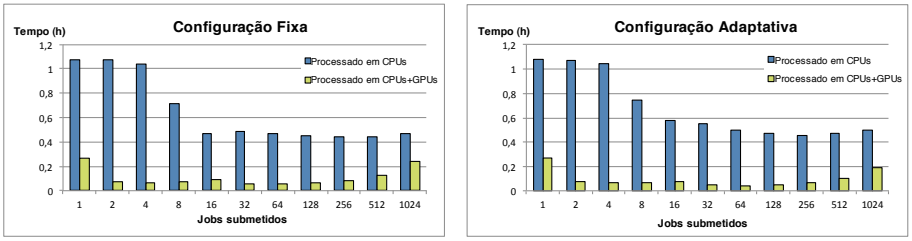

Figura 3: Tempos observados em testes com o fractal de Mandelbrot *invertido*.

Os testes realizados com o fractal de Mandelbrot *invertido* mostram uma melhoria de cerca de 50% do tempo médio de execução. Como esperado, o desempenho observado melhora quando são submetidos mais jobs. Os primeiros jobs, mais intensivos, foram agora atribuídos às PUs mais rápidas e os seguintes, mais leves, às mais lentas. Quando são usados apenas CPUs, os tempos observados são semelhantes tanto com a configuração fixa como com a adaptativa. Ao utilizar CPUs e GPUs os tempos observados são em média melhores com a configuração adaptativa do que com a configuração fixa.

Identificação de Genes Para este algoritmo foram simulados 5,12 milhões de genes. A NDRange para cada job foi configurada como um espaço de índices unidimensional com 512 itens e 1 item por grupo. Durante a fase de *profiling*

da configuração adaptativa foi gerado e submetido um job com 5120 genes para cada uma das PUs. Os tempos de execução obtidos com os testes de desempenho tanto para a configuração fixa como para a adaptativa da política de alocação de trabalhos *Score-Based* do Job Scheduler são apresentados na Fig. 4 .

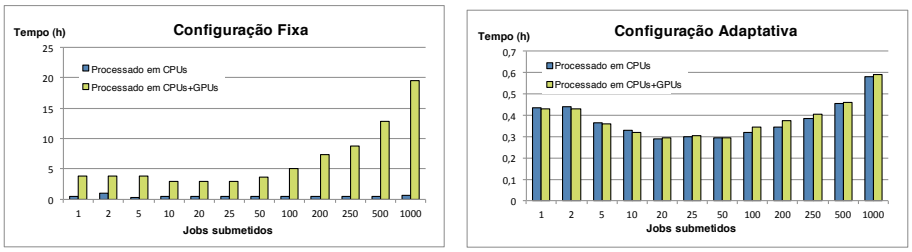

Figura 4: Tempos observados em testes com algoritmo de identificação de genes.

A configuração fixa apresentou tempos de execução razoáveis (na ordem das 0,1 horas) quando foram usados apenas CPUs. No entanto, registou-se uma perda de performance muito significativa (de duas ordens de grandeza) quando foram introduzidas GPUs. Com a configuração adaptativa, a introdução de GPUs não prejudicou o desempenho global do sistema.

Com a configuração fixa, a submissão de menos jobs apresentou tempos de execução inferiores. A submissão de mais jobs levou à atribuição de uma maior quantidade de trabalho às GPUs que, apesar de apresentarem valores de FLOPS elevados, têm um desempenho fraco neste problema. Como consequência da assunção de que a utilização preferencial de GPUs, que apresentam FLOPS mais elevados, resultaria em melhores performances para este problema, a configuração fixa atribui mais trabalho às GPUs, sobrecarregando-as com jobs que demoram mais tempo a executar, enquanto que os CPUs ficam em espera durante a maior parte do tempo. Com a configuração adaptativa, os tempos observados quando foram usados apenas CPUs são inferiores aos registados com a configuração fixa. Também se registou uma melhoria nos tempos observados aquando da introdução de GPUs. Isto deve-se ao facto de as GPUs terem sido seleccionados para processar jobs com uma frequência muito reduzida. Só quando o problema foi particionado em 20 jobs é que um deles foi seleccionado para executar numa GPU; só a partir dos 200 jobs foram usadas todas as GPUs. Este facto levou à redução de alguma da carga computacional dos CPUs ao mesmo tempo que se evitou que as GPUs processassem demasiados jobs, não prejudicando assim o desempenho global do sistema.

4.2 Discussão de Resultados

Os testes com o gerador de fractais num ambiente distribuído mostram melhorias de desempenho consideráveis, tanto com a configuração fixa como com a

adaptativa, tendo a primeira apresentado valores ligeiramente melhores. Os diversos jobs desta aplicação têm requisitos de processamento diferentes: um job correspondente a uma zona mais colorida é computacionalmente mais rápido a processar que um job correspondente ao processamento de uma zona a preto. Esta disparidade introduz um factor de erro sobre os tempos de execução observados ao usar a configuração fixa do Job Scheduler.

Para o algoritmo de identificação de genes, a configuração fixa provou ser totalmente inadequada quando foram usadas GPUs, e a adaptativa mostrou ser neutra a essa introdução. A perda de performance na configuração fixa é compreensível se for tido em conta o facto de que as GPUs são processadores intrinsecamente SIMD. As aplicações que não seguem este modelo de execução observam performances muito inferiores. Uma vez que a configuração fixa atribui jobs baseando-se nos FLOPS das PUs, as GPUs acabaram por receber muito mais jobs para processar do que os CPUs. Com a configuração adaptativa, a má performance das GPUs foi tida em conta pelo que não se observou um impacto significativo sobre os tempos de execução totais. Uma análise mais aprofundada destes e outros resultados pode ser consultada em [20].

Análise de Acelerações (*Speedups*) A versão original, sequencial, em C, do gerador de fractais de Mandelbrot foi processada em 5h 2min 20s no CPU mais rápido, o Core i5, e em 10h 9min 42s no CPU mais lento, o Core 2. Com a *framework* Cheetah obteve-se o melhor desempenho com a submissão de 32 jobs e a configuração fixa, com um tempo de execução de 2min 35s, representando uma aceleração de 117× e 135× relativamente às execuções mais rápida e mais lenta, respectivamente.

A versão original, sequencial, em C, do algoritmo de identificação de genes foi processada em 3h 56min 50s no CPU Core i5 e em 7h 31min 25s no CPU Core 2, os processadores mais rápido e mais lento, respectivamente. A melhor performance distribuída correspondeu a um tempo de execução de 17min 13s com a submissão de 20 jobs. Estes valores correspondem a uma aceleração de uma ordem de grandeza, mais precisamente de 14× e 26× relativamente às execuções sequenciais mais rápida e mais lenta, respectivamente.

5 Trabalho Relacionado

Existem actualmente diversos sistemas para gestão de clusters de computadores. Vários destes sistemas englobam componentes de escalonamento de trabalhos para optimizar a utilização dos recursos. Apesar do interesse crescente na utilização de GPUs em aplicações com elevados requisitos, ainda muito poucos sistemas as consideram como processadores de primeira classe.

A *Moab Cluster Suite* [6] foi concebida para simplificar a gestão de clusters. Possui um gestor de carga interno que, ao ser integrado com o *Maui Scheduler* [23], possibilita a utilização de algoritmos de escalonamento mais avançados. O *Oracle Grid Engine* [4] (anteriormente conhecido como *Sun Grid Engine*) é um sistema de gestão de clusters, que pode também ser integrado com o projecto

Hedeby [5] por forma a melhorar a sua operação em ambientes de multi-cluster. O *TORQUE* [7] é um gestor de recursos, o sucessor do histórico *PBS* (*Portable Batch System*) [2]. O seu principal objectivo é a gestão de recursos mas também dispõe de escalonadores de processos, podendo igualmente ser estendido com outros escalonadores mais avançados. O *Condor* [3] é uma *framework* desenhada para ser usada em ambientes de cluster e COW (Clusters de Workstations). A sua característica mais distintiva é a capacidade de utilizar apenas os computadores que estejam desocupados de processos que não sejam geridos pelo próprio *Condor*. Nenhum destes sistemas considera GPUs como processadores de primeira classe, sendo estas comummente consideradas como dispositivos de I/O que podem ser utilizados e devem ser geridos pelas aplicações. Isto significa que quando um programa é lançado numa dada máquina, estes gestores consideram-na como *ocupada*, mesmo que a aplicação utilize uma GPU e o CPU da máquina esteja maioritariamente em espera, potencialmente disponível para processar outros jobs.

Têm surgido diversas propostas que tiram partido das GPUs para melhorar o tempo de processamento de aplicações genéricas. Víctor Jiménez et al. [14] propuseram um *middleware* capaz de lançar jobs sobre o CPU ou a GPU de uma máquina. Esta proposta é restringida por dois aspectos que são resolvidos pela nossa aproximação: primeiro, o middleware aceita jobs de dois tipos (kernels CUDA [19] para execução em GPU e código convencional para execução em CPU), o que duplica o esforço de desenvolvimento exigido aos programadores que pretendam que o middleware seleccione automaticamente o processador a usar; segundo, sendo um sistema para uma única máquina, não cobre ambientes de clustering com diversas máquinas. Everton Hermann et al. propuseram, em [13], um sistema de *middleware* com o objectivo de abstrair o número de GPUs numa máquina. De forma semelhante à proposta de Jiménez et al., este sistema está restringido à utilização de uma única máquina e também requer a submissão de duas implementações para o mesmo algoritmo para que este seja executado em CPU ou GPU. O *Maestro* [22] é outra *framework de middleware* desenhada para gerir múltiplos dispositivos OpenCL e optimizar a utilização dos recursos pelas aplicações. Os jobs submetidos neste middleware são re-executados diversas vezes com diferentes configurações da NDRange por forma a determinar a melhor configuração para cada kernel. A Cheetah cobre não só ambientes com uma só máquina como também ambientes com diversas máquinas, que não são cobertos pelo *Maestro*. O *rCUDA* [12] permite a execução remota de kernels CUDA, que são enviados para execução assim que são submetidos. Este sistema não implementa portanto qualquer estratégia mais complexa de escalonamento dos kernels, ficando estes ao nível das filas locais de cada dispositivo e à responsabilidade do runtime CUDA. Em contraste com o *rCUDA*, a Cheetah usa OpenCL para o processamento de dados tanto em GPU como em CPU, e recorre a algoritmos de escalonamento para optimizar a distribuição de carga entre as PUs. O *MOSIX* [10,9] é um sistema de gestão de clusters capaz de migrar processos de máquinas sobrecarregadas para outras menos utilizadas. Este sistema tem capacidade de suporte para OpenCL (*Virtual OpenCL layer VCL*) [8], permi-

tindo às aplicações ver todas as GPUs e CPUs instaladas em todas as máquinas de um cluster como se estivessem acessíveis directamente na máquina anfitriã do processo em questão. As aplicações podem requisitar a execução de kernels em um ou mais dispositivos. Em contraste com a Cheetah, não são suportadas actualmente quaisquer estratégias de escalonamento ou balanceamento de carga.

6 Conclusões

Tanto quanto nos foi possível apurar, este artigo apresenta a primeira proposta de uma infraestrutura capaz de executar aplicações genéricas sobre um conjunto de recursos computacionais que considera GPUs como processadores de primeira classe, a par dos CPUs. Na *framework* Cheetah, o código fonte para cada job é um kernel OpenCL, sendo portanto universal e independente do tipo de arquitectura do CPU/GPU onde será compilado e executado. Esta aproximação liberta o programador do fardo do desenvolvimento de código afinado para cada tipo de processador/hardware, resultando numa solução sub-óptima a nível de desempenho mas ainda assim extremamente eficiente, tal como comprovado pela avaliação realizada.

Avaliou-se exaustivamente o sistema com duas aplicações distintas, tendo-se concluído que as acelerações obtidas dependem claramente da natureza e da forma como cada problema é implementado. Para alguns problemas, as GPUs comportam-se como apenas mais um processador, enquanto que para outros possibilitam a obtenção de acelerações muito significativas. Em qualquer dos casos, com os algoritmos de escalonamento apropriados, a Cheetah selecciona o processador mais adequado para cada job, tirando o máximo partido da infraestrutura computacional.

Apesar dos recursos computacionais limitados, ambas as aplicações de teste — um gerador de fractais de Mandelbrot e um algoritmo de identificação de genes — obtiveram acelerações consideráveis relativamente à sua implementação original. A aplicação de identificação de genes, que segue um modelo de execução do tipo MIMD, apresentou uma aceleração de mais de 20×. O gerador de fractais de Mandelbrot, que segue um modelo de execução SIMD, mais adequado para a execução em GPUs, obteve acelerações de mais de 200×. Estes resultados demonstram que a nossa aproximação é válida e eficiente, permitindo a distribuição transparente de jobs computacionalmente muito intensivos para CPUs e GPUs, numa estratégia quase-óptima, com melhorias de performance consideráveis e com custos de desenvolvimento reduzidos.

Referências

1. TOP500 Supercomputing Sites. `http://www.top500.org/`.
2. The Portable Batch System. `http://www.nas.nasa.gov/Software/PBS/`, 1998.
3. Condor: High Throughput Computing. `http://www.cs.wisc.edu/condor/`, 2010.
4. gridengine: Home. `http://gridengine.sunsource.net/`, 2010.
5. Hedeby: Hedeby Project Overview. `http://hedeby.sunsource.net/`, 2010.

6. Adaptive Computing Enterprises Inc. *Moab Workload Manager — Administrator's Guide*, version 5.4 edition, 2010.
7. Adaptive Computing Enterprises, Inc. *Torque Administrator's Manual*, version 2.4.5 edition, 2010.
8. Amnon Barak, T. Ben-Nun, E. Levy, and Amnon Shiloh. A package for OpenCL based heterogeneous computing on clusters with many GPU devices. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1 –7, 2010.
9. Amnon Barak and Amnon Shiloh. *MOSIX — User's and Administrator's Guides and Manuals*, revised for mosix-2.29.0 edition, 2010.
10. Amnon Barak and Amnon Shiloh. The MOSIX Management System for Linux Clusters, Multi-Clusters, GPU Clusters and Clouds, 2010.
11. Mark A. Beaumont and Richard A. Nichols. Evaluating Loci for Use in the Genetic Analysis of Population Structure. In *Biological Sciences*, volume 263, pages 1619–1626. The Royal Society, December 1996.
12. J. Duato, A. J. Peñ anda, F. Silla, R. Mayo, and E. S. Quintana-Orti. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 224–231, July 2010.
13. Everton Hermann, Bruno Raffin, François Faure, Thierry Gautier, and Jérémie Allard. Multi-GPU and Multi-CPU Parallelization for Interactive Physics Simulations. In *Europar 2010*, Ischia-Naples Italy, 09 2010. I.: Computing Methodologies/I.3: COMPUTER GRAPHICS, I.: Computing Methodologies/I.6: SIMULATION AND MODELING.
14. Víctor Jiménez, Lluís Vilanova, Isaac Gelado, Marisa Gil, Grigori Fursin, and Nacho Navarro. Predictive runtime code scheduling for heterogeneous architectures. In André Seznec, Joel Emer, Michael O'Boyle, Margaret Martonosi, and Theo Ungerer, editors, *High Performance Embedded Architectures and Compilers*, volume 5409 of *Lecture Notes in Computer Science*, pages 19–33. Springer Berlin / Heidelberg, 2009.
15. Volodymyr V. Kindratenko, Jeremy J. Enos, Guochun Shi, Michael T. Showerman, Galen W. Arnold, John E. Stone, James C. Phillips, and Wen-mei Hwu. GPU Clusters for High-Performance Computing. In *Proceedings of the Workshop on Parallel Programming on Accelerator Clusters (PPAC'09)*, August 2009.
16. Benoit B. Mandelbrot. *Fractals and the Geometry of Nature*. New York: Free-man, 1983.
17. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, version 2.2 edition, September 2009.
18. Aaftab Munshi. *The OpenCL Specification*, 1.1.36 edition, 2010.
19. NVIDIA. *NVIDIA CUDA C Programming Guide*, version 4.0 edition, May 2011.
20. Luís M. Picciochi Oliveira. A Framework for Scientific Computing with GPUs. Master's thesis, Universidade Nova de Lisboa, Portugal, 2011.
21. John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.
22. Kyle Spafford, Jeremy Meredith, and Jeffrey Vetter. Maestro: data orchestration and tuning for opencl devices. In *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II*, Euro-Par'10, pages 275–286, Berlin, Heidelberg, 2010. Springer-Verlag.
23. Supercluster Research and Development Group. *Maui Administrator's Guide*, maui 3.2 edition, May 2002.