Desenho e Implementação de uma Biblioteca de Padrões Algorítmicos para GPGPU*

Ricardo Marques, Hervé Paulino e Pedro Medeiros

CITI / Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Resumo As unidades de processamento gráfico (GPUs) apresentam-se mais eficientes do que os CPUs em certos domínios de aplicação. No entanto, extrair este poder computacional requer que o programador esteja familiarizado com modelo de execução subjacente. O trabalho apresentado neste artigo foca o problema da orquestração da computação, aplicando-se o conceito de padrão algorítmico (*skeleton*) à computação GPU. A nossa aproximação expande este conjunto dos *skeletons* suportados, bem como permite nidificação entre os mesmos. Uma avaliação inicial revela que a biblioteca desenvolvida oferece ganhos de desempenho substanciais relativamente a aplicações que não utilizam *skeletons*.

1 Introdução

As capacidades de computação da unidade de processamento gráfico (GPU), particularmente ao nível do paralelismo, atribuíram-lhe uma crescente popularidade como meio computacional auxiliar ao processador central (CPU). O poder computacional dos GPUs torna-os mais eficientes do que os CPUs em certos domínios de aplicação. Porém, explorar este poder computacional tem um custo associado, nomeadamente ao nível da complexidade do modelo de programação.

Por conseguinte foram propostas APIs GPGPU (General Purpose Computing on Graphics Processing Units) de baixo [1, 2] e alto [3, 4] nível. Estas exportam um modelo de execução divergente do da computação gráfica, o que facilita a criação de aplicações dirigidas a este tipo de hardware. Contudo, estas APIs possuem limitações que dificultam o seu uso por parte dos programadores ou não permitem extrair todo o potencial da arquitetura.

Recentemente surgiu a ideia de aplicar padrões algorítmicos [5] (*skeletons*) ao contexto do desenvolvimento GPGPU, oferecendo ao programador bibliotecas de estruturas e comportamentos recorrentes no âmbito da computação paralela. Os *skeletons* abstraem o programador dos pormenores de baixo nível inerentes à programação GPU, bem como do seu modelo de execução. Do nosso conhecimento,

* Este trabalho foi parcialmente financiado pela FCT-MCTES através do PEst-OE/EEI/UI0527/2011 - Centro de Investigação em Informática e Tecnologias e no âmbito do Projecto PTDC/EIA-EIA/102579/2008 - Ambiente de Resolução de Problemas para Caracterização Estrutural de Materiais por Tomografia.

atualmente apenas duas plataformas aplicam a noção de *skeleton* ao contexto GPU, designadamente as bibliotecas SkePU [6] e SkelCL [7]. No entanto, esta área de investigação está ainda num estágio muito preliminar.

A nossa proposta foca o desenho e implementação de uma biblioteca C++ de *skeletons* que expanda o conjunto atual de *skeletons* executáveis em GPUs (através de OpenCL). Pretende-se também oferecer transparentemente ao programador a capacidade de aumentar o desempenho das suas aplicações através da sobreposição entre comunicação e computação (c/c). Esta sobreposição aproveita o facto de um GPU permitir efetuar transferências bidirecionais simultâneas para, e da, sua memória, ao mesmo tempo que executa *kernel(s)*.

Por fim, pretende-se atribuir aos *skeletons* a flexibilidade de poderem ser combinados arbitrariamente entre si, permitindo a construção de aplicações paralelas complexas bastando para isso juntar logicamente módulos computacionais distintos. A este mecanismo dá-se o nome de nidificação (*nesting*), sendo que a aplicação final corresponde a uma árvore de composição entre *skeletons*.

O protótipo implementado inclui os skeletons: *Stream*, *Pipeline*, *Loop* e *For*, um caso particular do anterior. A sua semântica é descrita na secção 2 e a sua avaliação na secção 3.

2 Uma Biblioteca de Padrões Algorítmicos para GPUs

A biblioteca de *skeletons* proposta tem como principal objetivo proporcionar ao programador uma abstração de alto nível relativamente ao modelo de programação usual dos GPUs. Esta abtração é usualmente feita a dois níveis: orquestração do *host* e computações paralelas (*kernel*), sendo que a nossa proposta foca-se no primeiro nível. O modelo de execução dos *skeletons* propostos é assíncrono e divide-se nas seguintes cinco etapas: 1. Preparação dos dados de entrada, alocação da memória necessária para a escrita dos resultados e composição dos *skeletons* a utilizar. 2. Parametrização do *skeleton* raíz com as referências de memória de entrada e saída. 3. Execução assíncrona do *skeleton* raíz (que retorna um *futuro*). 4. Obtenção dos resultados da execução através da interface disponibilizada pelo *futuro*. 5. Libertação do espaço utilizado pelos *skeletons* e pelos *futuros*.

Atualmente, a biblioteca oferece os seguintes *skeletons*: *Stream*, *Loop* e *Pipeline*. O *Stream* define uma estrutura de computação onde o conjunto dos dados de entrada é submetido de forma incremental ao longo do tempo, induzindo uma ideia de persistência na computação. De modo a explorar o paralelismo providenciado pelos GPUs, particularmente ao nível das transferências entre memórias e as execuções, o *Stream* otimiza o encadeamento entre operações de modo a reduzir o tempo que o GPU se encontra parado, paralelizando entre si escritas, leituras e execuções.

O *Loop* faculta uma estrutura útil para a aplicação de uma computação iterativa a partir de um dado conjunto de dados de entrada e enquanto uma dada condição seja satisfeita. Caso o ciclo dependa dos resultados parciais são sempre efetuadas uma ou mais leituras no final de cada execução. Consequentemente,

existe uma grande fonte de paralelismo a aproveitar, dado que o GPU pode conciliar estas leituras com a execução de outra instância do *kernel*. Torna-se então altamente eficiente parametrizar um *Stream* com um *Loop*, pois aumenta-se a taxa de ocupação do GPU.

O *Pipeline* oferece uma estrutura capaz de interligar eficientemente a execução de diferentes *kernels*, onde os dados de saída do estágio i são passados como dados de entrada para o estágio $i+1$. Considerando que as tranferências entre memórias introduzem uma penalização de desempenho significativa, este esquema computacional é ideal para execução GPU pois os dados intermédios não precisam de ser transferidos de volta para memória principal de modo a ficarem acessíveis ao próximo estágio. Assim, com o *Pipeline* o programador é capaz de combinar eficientemente uma série de tarefas serializáveis interdependentes. Ao parametrizar um *Stream* com um *Pipeline* ganha-se sobreposição ao nível das escritas, leituras e execuções de diferentes estágios.

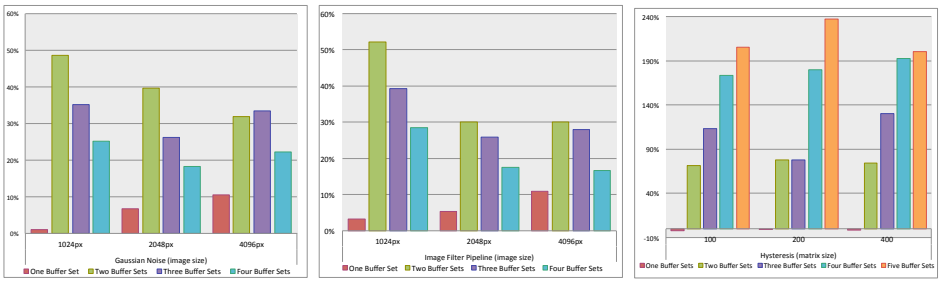
Esta biblioteca permite *nesting* entre todos os *skeletons* mas a diferentes níves, isto é, existem conceptualizações pré-definidas na forma como estes se podem combinar, em particular: 1. O *Stream* não é *nestable*, servindo apenas como raíz da árvore de composição caso se pretenda tirar partido da sobreposição c/c. 2. Os *kernels* são necessáriamente as folhas da árvore.

3 Avaliação

A avaliação elaborada foca o aumento de desempenho proporcionado pela sobreposição c/c. Por conseguinte, foram testadas diversas aplicações, cada uma das quais com duas versões (uma puramente OpenCL sem sobreposição c/c e outra com *skeletons*). Cada versão foi executada com diferentes dados de entrada, com um grão continuamente crescente. Foram calculados os tempos médios de execução de cada versão com diferentes números de conjuntos de *buffers*, comparando-os posteriormente com versões equivalentes e de igual grão. Os intervalos de tempo referentes à inicialização/finalização foram excluídos. A plataforma de testes possui as seguintes especificações: CPU Intel Xeon E5506 @2.13 GHz (4-core); 12 GB RAM DDR-3; GPU Tesla C2050 e sistema operativo Linux Ubuntu 10.04.4 LTS kernel 2.6.32-41 com o driver gráfico versão 295.41

A primeira aplicação aplica um filtro Gaussian Noise a uma imagem. Esta é dividida em fatias, o que sugere a utilização do *Stream* de modo a obter sobreposição c/c. A segunda aplica três filtros consecutivos a uma imagem, sendo estes: Gaussian Noise, Solarize e Mirror. Neste exemplo utiliza-se um *Stream* parametrizado com um *Pipeline*. A última aplicação usa um método de análise de imagens tomográficas, denominado por Histerese. O processo é constituído por três ciclos. Logo, utilizam-se três *Streams* parametrizados com um *Loop*.

Como se pode constatar, os ganhos de desempenho são significativos em todas as aplicações desde que se use pelo menos dois conjuntos de buffers. Isso significa que a sobreposição c/c se mostra vantajosa em todos os exemplos, chegando a melhorar o desempenho em cerca de 240% no caso da última aplicação.



(a) Gaussian Noise (b) Pipeline (c) Histerese

Figura 1: Ganhos de desempenho

4 Conclusão

Neste artigo apresentou-se uma biblioteca de *skeletons* para GPUs, onde se introduzem novos *skeletons* ao contexto GPGPU. Pretende-se com os *skeletons* não só aumentar o grau de abstracção relativamente ao modelo de computação subjacente, mas também aumentar o desempenho aplicacional transparentemente para o programador através da sobreposição c/c. Adicionalmente, pretende-se flexibilizar o uso dos *skeletons*, permitindo uma combinação rica entre os demais.

A avaliação efetuada permite concluir que a penalização de desempenho introduzida pelos *skeletons* é diminuta, reduzindo-se com o aumento do grão computacional. Em contrapartida, o aproveitamento da sobreposição c/c aumentou o desempenho de todas as aplicações de teste, mostrado-se vantajoso dado ser completamente transparente para o programador.

No que diz respeito ao trabalho futuro, pretende-se expandir o conjunto de *skeletons* oferecidos pela biblioteca proposta, bem como a exploração de múltiplos GPUs.

Referências

1. NVIDIA Corporation: NVIDIA CUDA. `http://www.nvidia.com/object/cuda_home_new.html`
2. Munshi, A., et al.: The OpenCL Specification. Khronos OpenCL Working Group. (2009)
3. AMD Corporation: Aparapi API for data parallel Java. `http://developer.amd.com/zones/java/aparapi/Pages/default.aspx` (2011)
4. Tarditi, D., Puri, S., Oglesby, J.: Accelerator: using data parallelism to program GPUs for general-purpose uses. In: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, ACM (2006) 325–335
5. Cole, M.I.: Algorithmic skeletons: structured management of parallel computation. MIT Press, Cambridge, MA, USA (1991)
6. Enmyren, J., Kessler, C.W.: SkePU: a multi-backend skeleton programming library for multi-GPU systems. In: Proceedings of the fourth international workshop on High-level parallel programming and applications. HLPP '10, ACM (2010) 5–14
7. Steuwer, M., Kegel, P., Gorlatch, S.: SkelCL - a portable skeleton library for high-level GPU programming. In: 25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011 - Workshops, IEEE (2011) 1176–1182