Towards Automatic Code Generation for GPU architectures

Javier Setoain[1], Christian Tenllado[1], Jose Ignacio Gómez[1], Manuel Arenaz[2], Manuel Prieto[1], and Juan Touriño[2]

[1] ArTeCS Group
Computer Architecture Department
Complutense University of Madrid, Spain
jsetoain@pdi.ucm.es,{tenllado,jigomez,mpmatias}@dacya.ucm.es
[2] Computer Architecture Group
Department of Electronics and Systems
University of A Coruña, Spain
{arenaz,juan}@udc.es

1 Introduction

Driven by the ever-growing demands of game industry, Graphics Processing Units (GPUs) have evolved from application-specific units for 3D scene rendering into highly parallel and programmable multipipelined processors, that can satisfy extremely high computational requirements at low cost. Their numbers are impressive. Today's fastest GPUs can deliver a peak performance in the order of 500 Gflops [11], more than four times the performance of the fastest x86 quad-core processor [7].

This astonishing performance has recently captured the attention of many developers and researchers in different areas, who are using GPUs as commodity data-parallel coprocessors to speed up their own applications [8]. This activity has been christened as General Purpose Computing on GPUs (GPGPU) [6]. Manufacturers in turn, driven by the increasing activity of this community, have developed new software interfaces that facilitate GPU programmability as general purpose parallel coprocessors. The most representative examples are NVIDIA's CUDA [5] and AMD's Brook+ [1].

This new scenario should redirect the efforts in GPGPU research from ad-hoc porting of applications, to the development of new compilation strategies that enable automatic mapping of sequential code. In this context, state-of-the-art tools for automatic recognition of program constructs, such as the XARK compiler framework [2, 4], provide a high level hierarchical representation of the program with valuable semantic information for the mapping process. However, we still need to define some performance metrics and heuristics in order to steer this mapping, and extend the compiler framework accordingly.

In this paper we perform several experiments aimed at analyzing the main factors behind GPU's performance in an attempt to define those heuristics. As a driven example we have used a real world algorithm [9] that exhibits some of

the computing patterns present in many scientific and image processing applications [3].

In the final contribution we will conclude with some hints about the extension of the XARK compiler framework for *automatic GPGPU*.

2 Compiler Support for Mapping Sequential Code onto Modern GPU Hardware

Today's optimizing compilers represent program behavior by means of several graphs that capture information at the statement and/or at the basic block levels. Well-known examples are the data dependence graph and the dominance tree. Our approach hinges on the construction of graphs at the kernel level. In order to recognize these kernels, we use the XARK compiler framework [2–4] as it provides a general solution to the problem of automatic recognition of program constructs. Thus, XARK builds a hierarchical representation that decomposes a program into a set of mutually dependent kernels that capture the behavior of a code fragment and provide the compiler with information about the computations carried out at runtime on scalar and non-scalar variables. Well-known examples of kernels are inductions, reductions and array recurrences. We propose a kernel-level intermediate representation of the program that does not take into account the characteristics of the target GPU hardware. Thus, in order to steer the mapping process, some performance metrics and heuristics of the target GPU hardware need to be defined. In addition, the kernel-level intermediate representation must be extended to meet the information requirements of these metrics and heuristics.

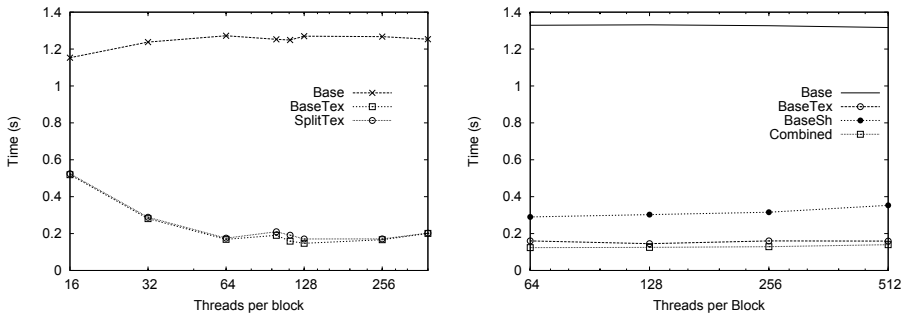
3 Performance Limiting Factors of Modern GPU Hardware

GPU performance is influenced by the architectural organization of the hardware platform. NVIDIA suggests that achieving the highest GPU occupancy and optimizing the use of the memory hierarchy are the two main factors behind GPU performance. In fact, both of them are related since maximizing the occupancy can help to cover latency during global memory loads. We present several experiments aimed at analyzing their relative importance. Our results indicate that code transformations that target efficient memory usage are the major determinant of actual performance. Overall, they ensure the best performance even if some resources remain underutilized. Therefore, maximizing occupancy should be examined at a later stage in the compilation process, once data related issues have been properly addressed.

[3] The interested reader can find more information about the driven application, as well as a description of an ad-hoc mapping in [9].

4 Preliminary Results

We have run several experiments on a NVIDIA GeForce 8800 GTX to study the influence of different code transformations on both the occupancy and the memory usage, analyzing their impact on overall performance. Figure 1 sketches part of our results. In order to maximize occupancy we have tried loop transformations, such as loop splitting, that reduce the pressure on shared resources and thus maximize occupancy. However, as shown in Figure 1(a), memory bandwidth is often the limiting factor. Essentially, any transformation of the original code has no impact on performance unless key memory issues has been properly addressed.



(a) (b)

Fig. 1. Influence of memory optimizations. (a) Using textures outperforms any occupancy related optimization. (b) Using shared memory alone does not solve alignment problems. A smart combination of textures and shared-memory provides the best performance

NVIDIA's programming guidelines stress the importance of exploiting the on-chip shared memory. The results in Figure 1(b) highlight that in practice, texture caches can provide similar or even higher benefits, despite having a higher access latency than shared memory. This is a consequence of memory alignment problems. Indeed, we have found that this is one of the most crucial aspects of memory optimization. At the expense of some data transfers overheads, accessing data through texture caches always guarantee aligned accesses. Although a similar access pattern may potentially be implemented using shared memory, it will involve (at the programming or compiler level) a complex analysis and substantial code transformations.

Summing up these experiments, (1) alignment should be target early in the compilation process and (2) if it remains an issue combining textures and shared memories in a smart way leads to the best performance.

In the final contribution we will include more elaborated discussion of these results as well as some hints about the extension of kernel-level intermediate

representation in order to provide useful information for the efficient mapping of sequential codes onto GPU architectures.

References

1. AMD Stream Computing White-paper: Software Stack. Available on-line at: http://ati.amd.com/technology/streamcomputing/firestream-sdk-whitepaper.pdf.
2. Manuel Arenaz, Juan Touriño, and Ramón Doallo. XARK: An Extensible Framework for Automatic Recognition of Computational Kernels. *ACM Transactions on Programming Languages and Systems* (accepted for publication)
3. Manuel Arenaz, Juan Touriño, and Ramón Doallo. A GSA-Based Compiler Infrastructure to Extract Parallelism from Complex Loops. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS'03, pages 193–204, New York, USA, 2003.
4. Manuel Arenaz, Juan Touriño, and Ramón Doallo. Program Behavior Characterization through Advanced Kernel Recognition. In *Proceedings of Euro-Par*, *Lecture Notes in Computer Science*, vol. 4641, pages 237–247. Springer-Verlag, 2007.
5. NVIDIA CUDA Programming Guide. Available on-line at: http://www.nvidia.com/object/cuda_develop.html.
6. General Purpose Computations on GPU's. http://www.gpgpu.org.
7. Nebojsa Novakovic. Harpertown benchmarks show a monster in the making core wars barcelona needs a big speed bump to compete. *The Inquirer*. September 18th, 2007. Available on-line at: http://www.theinquirer.net/
8. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. A Survey of General Purpose Computation on Graphics Hardware. In *Eurographics 2005: State of the art reports*, pages 21–51, 2005.
9. Javier Setoain, Manuel Prieto, Christian Tenllado, Antonio Plaza, and Francisco Tirado. Parallel Morphological Endmember Extraction Using Commodity Graphics Hardware. *IEEE Geoscience and Remote Sensing Letters, vol. 4, issue 3, pp. 441-445*, 4:441–445, July 2007.
10. Christian Tenllado, Javier Setoain, Manuel Prieto, Luis Piñuel, and Francisco Tirado. Parallel implementation of the 2d discrete wavelet transform on graphics processing units: Filter bank versus lifting. *IEEE Transactions on Parallel and Distributed Systems*, 19(3):299–310, 2008.
11. NVIDIA Tesla. GPU computing technical brief. May 2007. Available on-line at: http://www.nvidia.com/object/tesla_computing_solutions.html