Programming Models for Parallel Heterogeneous Computing

Frank Feinbube
Frank.Feinbube@hpi.uni-potsdam.de

This paper describes my work with the Operating Systems and Middleware group for the HPI Research School on "Service-Oriented Systems Engineering".

1 Motivation

Many of the computational problems we are facing today are complex and need huge computational power to be solved. It is well-known that processors will not continue to get faster, but will get more cores instead. More cores are not only harder to utilize by an application programmer, but also challenge hardware designers. Thus various new hardware architectures are designed and evaluated in order to find the ones that will fulfill the needs of future computer systems. Prototypic configuration boards like Intels Single Chip Cloud Computer (SCC) are an attempt to deal with the ever-increasing number of cores by removing essential features of current processors like hardware cache coherency. Another approach is to accompany common general purpose CPUs with sophisticated special purpose processing units. These so called Accelerators are easier to build and very fast for specific application purposes. They are the foundation for the new trend of hybrid computer systems.

2 Domains of Hybrid Systems

2.1 High Performance Computing

The High Performance Computing (HPC) community has a steady need for an increase of floating point operations per second (FLOPS) in order to simulate natural processes with a higher accuracy. Therefore enormous computer clusters are used. These supercomputers are not only very expensive to build, but do also need a huge amount of energy, both for processing and for cooling.

A popular approach to gain more FLOPS while reducing power consumption is to use GPU computing devices like NVIDIAs Tesla modules. Some of the best-performing 500 HPC clusters already contain a great amount of these devices. They have good cost per FLOPS ratio and like Cell processors, field-programmable gate arrays (FPGA) and other special compute accelerators are very energy efficient. They are quite adequate for this application domain and it can be expected that many of the next generation supercomputers will be heterogeneous systems facilitating GPU compute devices.

2.2 Business Servers

Due to ever-increasing application of RFID-technology and clever networks of software systems, business warehouses collect more and more data every day. These data sinks are gold mines for business intelligence. In order to exploit useful knowledge about customers and processes, a big machinery for data mining and information extraction is needed. The performance needs for such analysis applications are far beyond the ones of day-to-day online analytical processing (OLAP) query processors. On the other hand these systems do not have to be as highly available as the business critical online transaction processing (OLTP) servers. Thus more cost efficient components can be used. In order to accompany their mainframes with fast and cost efficient OLAP processing, IBM presented their IBM Accelerator solution at the IBM System z University event. The idea is to build a heterogeneous computing systems consisting of a standard mainframe and an additional server of several blades for very fast, but less reliable analytic operations on cloned databases.

In addition IBM introduced several accelerators to speed up various performance critical and regularly needed algorithms like (de)compression, XML parsing, encryption and decryption, and regular expression matching. These accelerators are special hardware devices designed to run sophisticated calculations at high speed. GPU compute devices can be regarded as another special kind of accelerators. There are also successful applications of GPU computing in the domain of business intelligence. [4, 6, 9] Hybrid systems consisting of reliable CPUs supported by various accelerators are expected to be the future configuration of IBM servers.

2.3 Desktop Computers

Even in the domain of Desktop computers heterogeneity has become common. The success of netbooks has confirmed that many end users are interested in cheap and mobile computers. Since these systems are so energy efficient, they also support the always-on mentality of these days. The trade-off for low power consumptions is a less powerful CPU. In contrast to this, many computers are used as entertainment devices and thus require a good performance, especially to decode and display music and video streams. To speed up these activities Atom processors are accompanied by on-die-GPUs like the Intel GMA 3150 GPU or NVIDIAs ION GPU. While these accelerators are applied for the specific application of video processing, they also can be used to speed up other classes of algorithms.

That medium-class and high-end computers can not only be used for computer games and high definition entertainment purposes, but also for realistic simulations and elaborate calculations has been proven by various BOINC projects (e.g. Folding@HOME). The parallel algorithms used in these projects have to exploit distributed parallel heterogeneous systems and thus present a great challenge to the programmer. On the other hand they are very elegant because they make use of the idle time of the participants computers that would otherwise be wasted.

2.4 Mobile and Embedded Systems

Mobile devices like cell phones have severe restrictions on power consumption because they are only powered by batteries. On the other hand as the success of iPhones demonstrates, users like fancy user interfaces, smooth visualization and entertaining applications. To support this functionality many current phones have a powerful processor and some co-processors. The upcoming smart phones conforming to the Windows 7 Phone specification will even contain a DirectX 9 compatible graphics processor. These processors may also be used for tasks that are not related to computer graphics. Some applications have shown how GPU compute devices can be used to support embedded systems liker routers [9] and home entertainment devices [10].

As for the other application domains mobile and embedded systems take advantage from hybrid approaches because they provide high special purpose performance while consuming only a small amount of power.

3 Programming Models

"This is the year when applications developed on GPU computing go into production." said NVIDIA CEO Jen-Hsun Huang. While most accelerators for hybrid systems are still at a prototypic stage, GPU computing has already achieved a variety of success stories. Especially calculation-intensive scientific applications seem to fit onto GPU computing devices very well. They were used to speed-up seismic exploration, weather modeling, computer vision, and medical imaging. The military applies GPU computing for advanced image processing and electromagnetic simulations. It is also used for business intelligence, complex event processing, speech recognition, engineering modeling, and analysis solutions.

In order to get a deep understanding of the programming model for GPU computing, we build a prototype that solved the NQueens puzzle for large board sizes on GPU compute devices. We learned that the CUDA programming model relies on SPMD kernels that work on a complex memory hierarchy in parallel. While some fundamental concepts like blocks, threads, and shared memory are made explicit, others like coalescing, occupancy, and local memory are hidden to the programmer. Understanding the explicit and implicit characteristics of a CUDA kernel execution is essential to predict the performance of a CUDA application. We presented our experiences at the ISPDC'2010 (section 5.1).

At the Summer School of the Universal Parallel Computing Research Center (UPCRC Illinois) (section 5.2) an overview about the various programming models for parallel shared memory systems was presented. The topics covered task-based approaches like Intels Threading Building Blocks (TBB), parallel for-loops like OpenMP, lambda expressions, vectorization principles, as well as GPU Computing with OpenCL.

OpenCL [1] is a standard introduced by the Khronos Group - a consortium of CPU and GPU hardware vendors. OpenCL is a programming model that allows to write programs for CPUs, GPUs and Cell Broadband Engine Architecture (CBEA) processors. It shares the concept of kernels and memory hierarchy of CUDA. In addition OpenCL

introduces the concept of streams. The OpenCL API is more low level than CUDA.

Powerful programming models and APIs like CUDA and OpenCL allow time efficient reformulation of multi-threaded code for CPUs for GPU computing. While this means little effort to execute parallel general purpose algorithms on GPUs, these will not utilize the GPU hardware well. Kirk et al. describe the situation this way: "If the application includes what we call data parallelism, it is often a simple task to achieve a 10x speedup with just a few hours of work. For anything beyond that, we invite you to keep reading!" [3]

We created a survey on best practices for optimizing GPU computing applications in order to really benefit from the acceleration GPU hardware can offer. This survey will be published in a special issue of IEEE:Software Journal in 2011 (section 5.3).

Although GPU computing is a mature accelerator category, it is still hard to write GPU computing code, even harder to utilize the GPU appropriately. Even the highly experienced developers of the National Supercomputer Center in Shenzhen were only able to reach 43 % of the theoretical peak performance for their hybrid supercomputer Nebulea. Nebulea is number two in the top 500 list of supercomputers and consists of NVIDIA C2050 GPUs. The utilization of 43% was reached for a dedicated implementation of the embarrassingly parallel LINPACK algorithm. [5]

Hybrid computing and heterogeneous computing need good programming models and tool support to overcome these difficulties and enable developers to benefit from these new system architectures. The industry is also interested in recommendations for hardware changes. Coming up with good programming models and tools for hybrid systems is hard. This is highlighted by the fact that the well-established sector of parallel and multi-core computing is still looking for appropriate programming models and empowering developer tools.

4 Research Plan

The area of my research is on parallel hybrid systems and accelerator technologies. I aim to help developers to handle the complexity of such system w.r.t. to coding experience and resulting application performance. Consequently, my focus is on parallel languages, parallel libraries, and parallel toolkits for hybrid systems.

As high-lighted again at the UPCRC (section 5.2) appropriate tools and programming models for parallel and multi-core computing are still ongoing research topics. In the field of hybrid systems, programming models and tools have not only to cope with parallelism, but also with differing execution characteristics of the processors and accelerators in a given system configuration. State-of-the-art programming is done with CUDA and OpenCL, which extend the C++ language. Especially OpenCL is a very low-level interface and thus laborious to work with. There are first approaches to reduce the burden for the programmer. Lee et al. [11] show that it is possible to use the OpenMP-API for GPU computing. Most of the other approaches are simple wrappers for higher-level languages. The problem with the current approaches is that they do not map onto the accelerators hardware very well, are only suitable for very special subset of problems, or lead to severe code bloat. This is where I want to work at.

In order to accomplish my research goal, the following tasks have to be completed:

1. Identify best practices and patterns for multi-core development and hybrid computing. (section 5.3)

2. Identify hardware capabilities and restrictions of hybrid architectures.

3. Identify common use cases and algorithms for hybrid computing. (section 5.4, section 5.5)

4. Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms. (first steps in section 5.6)

5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

Until now I worked mainly on milestones one to three. In order to realize step four, I will start with a selection of examples (section 5.5) and evaluate how each of them can be applied to a particular architecture. The next step is to apply the optimizations described in section 5.3 manually and learn which optimizations that can be applied automatically. The findings will than be formalized and provided as a .NET library.

5 Recent Activities

5.1 Paper presentation at the 9th International Symposium on Parallel and Distributed Computing (ISPDC)

I presented my experiences with the NQueens problem and CUDA at the 9th International Symposium on Parallel and Distributed Computing in Istanbul, Turkey in July 2010. [8]

The conference serves as a forum for engineers and researches and covers topics from parallel to distributed computing. The keynotes were held by D. Keyes and Wolfgang Gentzsch. They highlighted the current and future challenges for the high performance computing (HPC) community.

The focus of the first session was GPU computing. Our presentation started with an overview of the CUDA programming model. The NQueens problem was introduced and our parallelization approach was described. The main focus was the application of various optimizations onto our solution in order to achieve a better utilization of the card. These optimization led to contrary performance implications on the two available CUDA-enabled card generations of NVIDIA.

The other sessions discussed models and algorithms, multi-cores, Web Services and Multi-Agent Systems, Interconnection Topologies, Networks and Distributed Systems, Grids and P2P Systems, and Scientific Programming.

5.2 Visiting the UPCRC Summer School

I was visiting the Summer School of the Universal Parallel Computing Research Center (UPCRC Illinois). It is a joint research endeavor of the Department of Computer Science, the Department of Electrical and Computer Engineering, and corporate partners Microsoft and Intel. It aims to pioneer and promote parallel computing research and education.

The school started with an introduction on shared memory parallelism and multicore technology by UPCRC Co-Director Marc Snir. UPCRC Principal Investigator Danny Dig presented parallelism techniques for object-oriented languages and how refactoring can be applied to transform sequential applications into concurrent ones. Clay Breshears and Paul Peterson from Intel illustrated how OpenMP and Threading Building Blocks can be used for parallelizations. In addition they introduced cutting-edge developer tools by Intel: the Intel Parallel Inspector, the Intel Parallel Amplifier and the Intel Parallel Advisor. Phil Pennington and James Rapp from Microsoft presented the C++ Concurrency Runtime and the .NET Task Parallel Library (TPL). María Garzarán gave an overview on vectorization and described various techniques to apply them. UPCRC Illinois Co-Director and PI for the world's first NVIDIA CUDA Center of Excellence Wen-mei W. Hwu introduced OpenCL. John E. Stone of the Illinois Beckman Institute illustrated CUDAs utility with a Electrostatic Potential Maps application. Marc Snir concluded the school with his Taxonomy of Parallel Programming Models. As a special final event we were visiting the Petascale Computing Facility at Illinois which will house the Blue Waters sustained-petaflop supercomputer.

Besides getting a lot of practical experiences with various wide-spread parallel programming tools and libraries, I got an overview on application classes and use case for parallel computing. In addition I learned about the challenging problems for the area of parallel computing: programming models that are easy to use and powerful in leveraging parallel platforms. I also learned about the hardware trends that will push the shift from parallel to heterogeneous computing and thus will increase the importance of good programming models and execution platforms.

5.3 Journal Paper for the IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

Modern graphic cards are able to act as additional compute device beside the processor. Parallel computing is therefore no longer a dedicated task for the CPU, the new trend is *heterogeneous computing* of main processor and graphics processing unit (GPU).

Our journal article presents a synthesis of important strategies for utilizing the additional graphic processor power. We explain the primary concepts of GPU hardware and the according programming principles. Based on this foundation, we discuss a collection of commonly agreed critical performance optimization strategies. These optimizations are the key factor for getting true scalability and performance improvements when moving from a multi-threaded to a GPU-enhanced version of your application.

It will be published in a special issue of IEEE:Software Journal in 2011.

5.4 Knowledge Sink for Use Cases, Tools and Libraries

While we were creating our survey on best practices (section 5.3), we collected a lot of literature about GPU computing and related topics. This collection is available at [7]. Besides general topics, there are special collections for use cases of GPU Computing, as well as a collection of tools and libraries.

5.5 Example Implementations of Representative Use Cases

The hands-on labs at the Summer School of the Universal Parallel Computing Research Center (section 5.2) included a variety of algorithms that can be executed on parallel platforms. These included matrix-matrix multiplication, convolution, prefix scan, quicksort, and minimum spanning tree. Starting with these, we created a collection of representative use cases for parallel computing. We used further literature to extend the collection. [3, 12] These use cases will be used to evaluate our new programming models.

5.6 Prototype to Run OpenCL-Code from .NET

The current tools and techniques to use the OpenCL API restrict programmers to write C++ like code. In order to provide a greater audience with easy access to GPU computing, Jan-Arne Sobania and I were cooperating to run OpenCL programs using the .NET Framework. We used a simple parallel loop approach that is known to .NET developers because the parallel for loop of Microsofts Task Parallel Library (TPL) has become a part of the .NET 4.0 Framework. Our prototypic implementation demonstrates that we can provide OpenCL access similar to the TPL via a .NET library. This way it is easy and intuitive for a .NET developer to benefit for GPU computing. Recently a similar approach for Java has been made available by ATI. [2]

Based on our work we want to evaluate which of the best practices for optimizations that are described in section 5.3 can be applied by our .NET library. We expect that type and meta data information available in the runtime will be useful for that approach. Ueng et al. [13] demonstrated that coalescing memory access - a very popular optimization - can be applied automatically. Some other best practices have potential for runtime support as well.

6 Conclusion

This paper introduces the research area of hybrid systems. Section 2 gives an overview of the application domains for hybrid computing systems. The High Performance Computing (HPC) always aims at more flops and smaller power consumptions. Business servers use Accelerators for for OLAP and specific application needs. Home computer need to be high-performing entertainment devices that consume very little energy. Power restrictions are more severe on mobile and embedded devices, but even in this sector applications become more and more resource intensive. Programming

models and appropriate developer tools for parallel and multi-core computer systems are active researched topics. Programming models and tools for the domain of heterogeneous and hybrid systems have not only to cope with parallelism, but also with differing execution characteristics of the processors and accelerators in a given system configuration. My research aims at help developers to handle the complexity of such system w.r.t. to coding experience and resulting application performance. I worked on surveys on best practices and patterns, hardware architectures and uses cases of hybrid computing. Currently I am working on a library for a high-level language to access OpenCL-enabled accelerators. With the help of this library I plan to apply selected best practice optimizations automatically. The usefulness will be demonstrated using the collection of use case examples I created. Section 5 provides an overview on my recent work.

References

[1] The OpenCL Specification - Version 1.1, 6 2010.

[2] Advanced Micro Devices, Inc. Aparapi. `http://developer.amd.com/zones/java/Pages/aparapi.aspx`.

[3] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 1 edition, 2 2010.

[4] P. B. Volk, D. Habich, and W. Lehner. GPU-Based Speculative Query Processing for Database Operations. In *Proceedings of First International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures, ADMS, 10*, 9 2010.

[5] Jack Dongorra. Challenges for Exascale Computing. Technical report.

[6] Wenbin Fang, Bingsheng He, and Qiong Luo. Database Compression on Graphics Processors. In , 2010.

[7] Frank Feinbube. GPU Readings List. `http://www.dcl.hpi.uni-potsdam.de/research/gpureadings/`.

[8] Frank Feinbube, Bernhard Rabe, Martin von Löwis, and Andreas Polze. NQueens on CUDA: Optimization Issues. In *Proceedings of 9th International Symposium on Parallel and Distributed Computing, Istanbul, Turkey (ISPDC), 2010*, 2010. Istanbul, Turkey (to appear).

[9] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: a GPU-accelerated software router. In *Proceedings of SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, New Delhi, India*, pages 195–206, New York, NY, USA, 2010. ACM. New Delhi, India.

[10] Taskin Kocak and Nicholas Hinitt. Exploiting the Power of GPUs for Multi-gigabit Wireless Baseband Processing. In *Proceedings of ISPDC '10: Proceedings of the 2010 Ninth International Symposium on Parallel and Distributed Computing*, pages 56–62, Washington, DC, USA, 2010. IEEE Computer Society.

[11] Seyong Lee, Seung-Jai Min, and Rudolf Eigenmann. OpenMP to GPGPU: a compiler framework for automatic translation and optimization. In *Proceedings of PPoPP '09: Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming, Raleigh, NC, USA*, pages 101–110, New York, NY, USA, 2009. ACM. Raleigh, NC, USA.

[12] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, 7 2010.

[13] Sain-Zee Ueng, Melvin Lathara, Sara S. Baghsorkhi, and Wen-Mei W. Hwu. CUDA-Lite: Reducing GPU Programming Complexity. In *Proceedings of 21th International Workshop on Languages and Compilers for Parallel Computing, Edmonton, Canada (LCPC), 2008*, pages 1–15, Berlin, Heidelberg, 8 2008. Springer-Verlag. Edmonton, Canada.