High-Performance Computing with Desktop Workstations

Eric J. Kelmelis, John R. Humphrey, James P. Durbano, Fernando E. Ortiz
Accelerated Computing Division
EM Photonics, Inc.
51 E. Main St. Suite 203, Newark, DE, 19711
USA

Abstract: - The performance of modeling and simulation tools is inherently tied to the platform on which they are implemented. In most cases, this platform is a microprocessor, either in a desktop PC, PC cluster, or supercomputer. Microprocessors are used because of their familiarity to developers, not necessarily their performance on the problems of interest. We have developed the underlying techniques and technologies to produce supercomputer performance from a standard desktop workstation for a variety of applications. This is accomplished through the combined use of graphics processing units (GPUs), field-programmable gate arrays (FPGAs), Cell processors, and standard microprocessors. Each of these platforms has unique strengths and weaknesses but can be used in concert to rival the computational power of a high-performance computer (HPC). In this paper, we discuss the relative advantages and disadvantages of each platform and how they can be combined in order to achieve high performance on a variety of applications.

Key-Words: - Accelerator, Cluster, Desktop Supercomputer, FPGA, GPU, Cell Processor, HPC, Simulation, Real-Time Processing

1 Introduction

The general-purpose nature of microprocessors has led to their incorporation into a wide array of platforms, from embedded systems to desktop computers, and used for applications ranging from encryption to word processing. This flexibility, combined with their low price and simple programming model, has led to their popularity. However, this versatility comes at a cost; to maintain their generality, microprocessors sacrifice computational performance. Instead of being optimized for numerical processing, microprocessors focus on integrating features that will benefit the widest variety of applications, thus appealing to the broadest audience. However, many scientific computing applications, such as physics-based simulations and image processing, require far more computational power (and memory) than is available from a standard desktop PC. This has led to the advent of clustered computing, a technique that involves distributing a single problem to a group of commodity PCs and allowing them to perform computations in parallel. This "more is better" approach is epitomized in modern supercomputer design, which typically consists of many standard microprocessors linked by high-speed interconnects.

Although aggregating commodity processors into larger systems provides more computational power than a standard PC, there are drawbacks to this approach. First, the algorithm must be partitioned to run on multiple processing nodes. This is often a nontrivial task and typically results in wasted computational power. This problem is further compounded by the need to transfer information between microprocessors. For a vast collection of processors to work together effectively, they must communicate and synchronize data. This often forces processing elements to stall while waiting for other nodes to complete their tasks. The second major drawback to traditional high-performance computers (HPC) is their size. Clusters and supercomputers are large systems taking up an excessive amount of space and power, while generating tremendous heat. Finally, microprocessor-based HPCs are costly. While a single microprocessor is relatively cheap, building an HPC based on them can be quite expensive. To build a cluster, it is necessary to acquire many standard computers and the infrastructure to connect them. The hardware costs alone for a moderately sized cluster (35-50 nodes) can easily exceed $100,000. Additionally, parallel computational software must be purchased, either through an independent software vendor (ISV) or by hiring programmers.

While traditional microprocessor-based HPCs are frequently used, their performance, physical restrictions, and cost clearly make them far from an ideal solution. To better meet the needs of the

scientific computing community, we have developed the underlying techniques and technologies to produce supercomputer performance from a standard desktop workstation (a "desktop supercomputer"). This is accomplished through the combined use of graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and standard microprocessors. Each of these platforms has unique strengths that, when used in concert, can rival the computational power of traditional HPCs. Microprocessors, as discussed, are general purpose and can be easily programmed to target a variety of applications. FPGAs are reconfigurable chips that can be modified at the gate level by mapping a given architecture directly into the hardware. Because they are completely reconfigurable, FPGAs can be applied to many applications, but require lower-level programming than microprocessors and GPUs. GPUs are built for rendering graphics to a computer screen by performing the significant computations necessary to model a three-dimensional world. While far less flexible than microprocessors and FPGAs, GPUs are extremely efficient at performing single-precision arithmetic in a parallel fashion. For algorithms capable of utilizing such an architecture, the GPU offers high performance from a commodity hardware device.

The recent release of the Cell processor by IBM has provided another computational alternative to standard microprocessors. We have explored the integration of this new commodity device into our desktop supercomputing platform and have begun developing applications for this platform.

In this paper, we discuss the trade-offs of GPU, FPGA, Cell, and microprocessor technology, and how we efficiently partition algorithms to take advantage of the strengths of each while masking their weaknesses.

2 Hardware Platforms

2.1 Microprocessors

Whether in a standalone PC, a high-performance workstation, a PC cluster, or a multiprocessor system, microprocessors have been at the heart of computational systems for decades. Their longevity in this position can be attributed to their ease of programming, generality, and widespread availability. Advances in high-level programming languages, coupled with sophisticated compiler technology, have allowed users, with a variety of scientific and technical backgrounds, to quickly and efficiently program microprocessor platforms. Such broad appeal has allowed them to gain significant momentum as a computational tool, further increasing their use and lowering their cost.

Microprocessors contain a fixed, high clock-rate computational pipeline. This pipeline was designed to address the needs of the widest array of users and undergoes frequent changes to increase performance as newer models are introduced. Although rarely optimized for a given algorithm, microprocessors provide an easy-to-program platform that is capable of solving problems across many applications.

2.2 Field-Programmable Gate Arrays

Field-programmable gate arrays (FPGAs) can be thought of as a "sea" of reprogrammable logic gates, connected via configurable routing resources. That is, the fundamental logic functions (e.g., AND, OR, NOT) are user programmable and can be interconnected to build complex logic designs. FPGAs are programmed via a high-level hardware description language (HDL), such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog. The use of a high-level language abstracts the gate-level details away from the designer, enabling rapid, efficient programming. Once the developer's code is translated into an FPGA configuration file, the actual chip reprogramming can be performed in less than 100 ms.

FPGAs maintain several advantages over microprocessors and digital signal processors (DSPs). First, an FPGA can be ideally configured to perform a given task. For example, programmers of Intel's Pentium 4 processor are restricted to a fixed computational pipeline. Thus, operations that do not map well into this architecture will experience artificial performance degradation. FPGAs, however, can be optimally configured for each application by designing custom arithmetic pipelines. Further, FPGAs are capable of massive computational parallelism within a single chip. For example, consider the Pentium 4, which has only two floating-point arithmetic units. For applications that require multiple floating-point operations, this can represent an incredible performance bottleneck. An FPGA, however, can be reprogrammed with well over 100 floating-point units [1]. Thus, FPGAs allow algorithms to take full advantage of the underlying hardware for optimized performance.

Another advantage of FPGAs is that their processing power is advancing faster than that of microprocessors. FPGAs are a more recent

technology, with the first commercial products being introduced decades after the microprocessor. Moore's Law has held true and this has led to microprocessors operating in excess of 3 GHz. However, the next-generation of microprocessor technology is somewhat unclear. Physical limitations of power consumption and heat dissipation have led Intel to cancel their development of a 4 GHz chip. While it is clear that microprocessor development will continue, this field is maturing and the next generation of advances in computing will result not from simply shrinking feature sizes but from a more clever use of the underlying silicon. FPGAs, however, have not yet come close to reaching their potential. In the coming years, FPGA performance is expected to advance much faster than that of microprocessors because the field is far less mature (Fig. 1). Additionally, as fabrication techniques improve, FPGAs will take advantage of the same technology as next-generation microprocessors, as they are both built on standard CMOS processes.

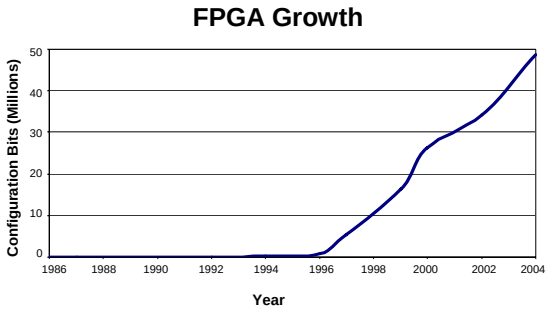

Fig. 1. Growth in the power of FPGAs since 1986. FPGAs have grown significantly in the last decade as shown by the number of configuration bits required to program the chips' gates and built-in functions. This advancement is expected to continue for the foreseeable future.

Because FPGAs maintain several advantages over microprocessors, many researchers [1-5] are looking toward this technology to meet their computing needs (see Fig. 2 for a picture of our custom FPGA platform). Using FPGAs for scientific computations requires a three-step process. First, the given algorithm is analyzed to determine the computational bottleneck. This is typically the algorithm kernel and is where the majority of the computations are performed. Next, this computational kernel is mapped into the FPGA. To do so efficiently requires knowledge of the internal FPGA architecture, a hardware description language, and the physics of the underlying algorithm. It is critical to fully understand the algorithm because code targeted for a microprocessor-based platform will rarely be optimized to run directly in the FPGA. Rather, the algorithm must be modified to take advantage of fully customizable computational pipelines and data caching schemes. This process generally requires manipulating the core equations. To do this in the most efficient manner requires intimate knowledge of the algorithm itself, not simply the equations that describe it.

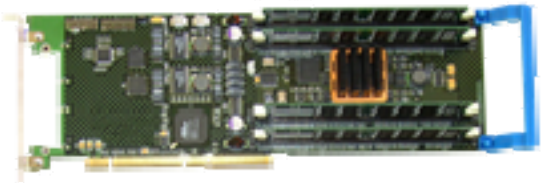

Fig. 2. EM Photonics Celerity™ accelerator board. This card connects to the standard PCI bus of a desktop machine and houses a Xilinx Virtex-II 8000 FPGA and 16 GB of DDR SDRAM.

2.3 Graphics Processing Units

Another recent trend in scientific computing is harnessing the immense power of commodity graphics processing hardware to accelerate numerical algorithms. A current GPU costs roughly the same amount as a high-end CPU, but is capable of significantly higher floating-point performance. The reason for this disparity is that the GPU needs only to perform specialized calculations, such as those required to render graphics to the screen, while the CPU must offer a complete set of functions. The benefit for scientific computing is that GPUs are high-powered computation engines, which provide hardware support for many common linear algebra and trigonometric functions. As this is precisely what many numerical algorithms require, there has been considerable effort focused on achieving large performance improvements by adapting existing software solvers to this platform.

Many applications have been ported to GPUs, including high-level algorithms and low-level fundamental mathematical methods, demonstrating the platform versatility for general-purpose computing. The list of low-level techniques includes the FFT [6] and linear algebra operations on dense [7], banded [8], and sparse [9] matrices. High-level examples encompass complete algorithms and include computational fluid dynamics [10], ray tracers [11], and image tone mapping [12].

The main challenge in general-purpose computing on GPUs is casting the problem into a form amenable to the graphics pipeline (Fig. 3). The first step is to convert all data into 1D or 2D arrays, which are stored in the GPU as textures. Then, a program is written for the fragment processor that is

intended to perform the given computational task. In order to run this program, a quadrilateral polygon is drawn in a manner that produces a 1:1 mapping between pixels and texels (texture elements), thus ensuring the desired function is executed once and only once on each piece of input data. Because some problems are extremely well suited for a GPU implementation and others are not, we will describe the advantages and disadvantages of using the GPU for computations in the next section.

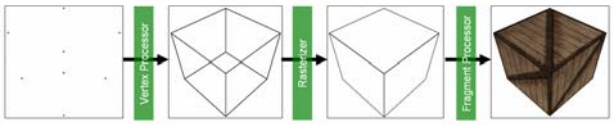

Fig. 3. Showing data as it progresses through the GPU pipeline. This figure shows a simple example of a box being drawn in the GPU pipeline. The vertices are processed in the vertex processor to form polygons, which are then passed to the rasterizer for filling. The rasterizer produces "blank" fragments that are "shaded" in the fragment processor. The images in this figure were created using sample OpenGL code available at http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=07 (as of March 2006).

2.4 Cell Processor

The Cell, or Cell Broadband Engine Architecture (CBEA), is a processor developed through a joint venture between IBM, Sony, and Toshiba. The chip consists of a PowerPC core, known as the Power Processing Element (PPE), coupled with eight streamlined vector processors, known as Synergistic Processing Elements (SPEs) (Fig. 4). The PPE is general-purpose and, as such, is usually responsible for running the operating system and coordinating the work of the SPEs. Each SPE is a 128-bit SIMD RISC processor with 256 KB local store for instructions and data. In a single clock cycle, one SPE can operate on 16 8-bit integers, 8 16-bit integers, 4 32-bit integers, or 4 single-precision floating-point numbers (there is no native double-precision floating-point support).

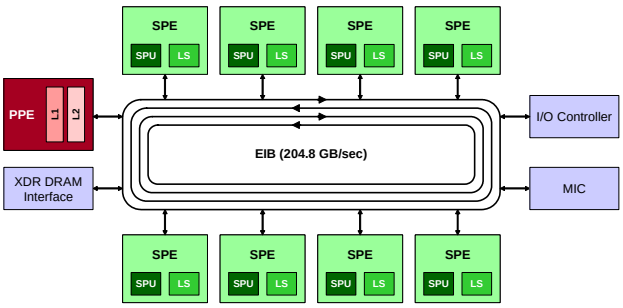

Fig. 4. Cell processor architecture. Shown is the conceptual layout of the Cell processor, including the PPE unit connected to the eight SPE units by a double-ring bus.

The Cell processor is currently being targeted for several platforms, though few commercial products have reached the marketplace. It is scheduled to be deployed in the gaming market through the Sony Playstation 3, in the server market through blade and rack-mountable computers, the workstation market through PCI Express-based co-processing cards, and the home video market through high-definition equipment. While the form factors will be different, the programming models for each will be the same. There are currently four programming models supported by the Cell. The first is function offload where the PPE runs the majority of a program with small portions offloaded to the SPEs for fast computations. The second model is for the PPE to hold a task queue from which SPEs can pull jobs when they become available. The third option is problem partitioning or self multi-tasking of SPEs where a larger computational problem is divided among the SPEs in the system. The final model is stream processing, whereby the results from one SPE fed to another SPE where the further operations are performed on that data. How this technology integrates with our desktop supercomputing architecture is described in the next section.

3 Building a Desktop Supercomputer

Each hardware technology integrated into our desktop supercomputing platform has advantages and disadvantages. It is the ability to maximize the unique strengths of microprocessors, FPGAs, GPUs, and Cell processors, while masking their weaknesses, which allows our desktop supercomputer to be applied to a wide variety of applications and still perform at a high level (Fig. 5).

Fig. 5. Desktop supercomputer. Here we see our desktop supercomputer platform, with a Pentium 4 microprocessor, NVIDIA GeForce 7800 GTX GPU, and Celerity™ FPGA-based accelerator, shown in perspective to a standard monitor, keyboard, and mouse.

High-end microprocessors, while cheaper than high-end FPGAs, can run over $1,000. They are general purpose and can be programmed to perform almost any task. While they are easy to use and very flexible, they are not optimized for most computational tasks. Many of their key arithmetic operations are not pipelined, or must be achieved

through several simpler operations, which results in low computational performance. Microprocessors also have a relatively low memory bandwidth. This slow connection to memory, coupled with no fine-grained control over the caching scheme, makes it costly to read and write data. Despite these limitations, however, microprocessors play a key role in our platform. Their most obvious tasks include running the underlying operating system, managing hardware devices, and executing user applications. Additionally, while the bulk of the repetitive computations performed by our desktop supercomputer are run on the GPU and FPGA, the microprocessor performs many irregular, non-computationally intense tasks related to setup and post-processing.

FPGAs also maintain several disadvantages when compared to the other hardware platforms. First, high-end FPGAs are relatively expensive (an order-of-magnitude more costly than microprocessors and GPUs). However, their performance typically justifies this cost, which is why they are included in our platform. Second, they are the most difficult to program of the three computational platforms discussed. While high-level programming languages are available, they are more complex than languages such as C and FORTAN, which were designed for microprocessors. Also, FPGA synthesis (similar to the microprocessor concept of compiling) and debugging require significantly more time (by an order of magnitude) than their microprocessor language alternatives. Finally, while our FPGA-based platform has demonstrated very high memory bandwidth for up to 16 GB of RAM, it does incur a one-time penalty due to the relatively slow transfer of problem initialization data over the PCI bus. Although this overhead is trivial when running large simulations, it must be considered when determining whether the FPGA is suitable for a given algorithm or task.

It is important to note that GPUs also have several weaknesses, the biggest of which is their limited memory. Top-of-the-line GPUs currently have only 1 GB of onboard memory. This is significantly less than the microprocessor can access and is far less than the 16 GB on our FPGA co-processing card. A second major consideration for GPUs is that, while extremely good at single-precision, floating-point computations, they lack native support for double-precision arithmetic. Both of these limitations are unlikely to change in the near future because neither larger memories nor double-precision computations are required for rendering graphics. Additionally, though not as complex as FPGAs, GPUs are relatively difficult to program. Efficient use of the underlying hardware requires intimate knowledge of the graphics pipeline and the ability to map the algorithm into the computational resources available.

The Cell processor is by far the least mature of these technologies and, consequently, its future remains somewhat unclear. Initial performance benchmarks have shown promise, but it remains to be seen how the final systems will be deployed. Due to its architectural similarities to the GPU, it is unlikely both would be added as coprocessors to the same desktop workstation, however, for certain applications, one could be chosen over the other based on price, power, form factor, programming model, and other considerations. A more likely scenario for using the Cell processor in the same system with an FPGA and GPU would be to replace the standard microprocessor with the Cell. With its PPE, the Cell processor could handle the operating system and control tasks generally targeted for the microprocessor and its SPE units would support high-performance computations.

Because each hardware technology maintains unique advantages and disadvantages, every algorithm can be mapped onto the platform for which it is best suited. For instance, an algorithm that requires single-precision calculations on two-dimensional solution spaces, such as many explicit time-domain methods, maps well onto a GPU. However, an algorithm that requires double-precision, complex arithmetic will likely map better into the FPGA, where the programmer can build custom, ideally pipelined arithmetic units.

Effectively mapping computationally intense algorithms into the desktop supercomputer requires maximizing the strengths of each hardware technology while minimizing their weaknesses. As shown, no single technology is ideal for all applications. By intelligently mapping a given algorithm, however, developers are able to achieve HPC performance from a single desktop workstation.

Additionally, instead of simply implementing an entire algorithm on a single hardware device, it can be partitioned to utilize the strengths of all four. By decomposing an algorithm into its functional components and then mapping these smaller units to the most suited device, we can achieve greater performance than any individual hardware platform

could offer. By combining all of the technologies discussed, one desktop machine is able to support an ideal mapping of each formulation in a single environment.

4 Conclusion

Performing comprehensive simulations of physical phenomena requires significant computational power. The majority of modern solvers are built around the most well known computational hardware: the microprocessor. While easy to program and familiar to most engineers, microprocessors are far from the most powerful computational devices available. In this paper, we presented three alternative platforms: FPGAs, GPUs, and Cell processors. These devices are very powerful when applied to scientific computing tasks and each is well suited for large classes of problems. While others have used the technologies discussed in this paper individually, we have found that they are most effective when used together. Consequently, we have created a desktop supercomputer that combines a high-end microprocessor, a state-of-the-art GPU, and the largest FPGA available. We are currently integrating the Cell processor into this platform, both through a co-processing board and by building a workstation that uses the Cell as the system's microprocessor. By tailoring a given problem to this combined platform and intelligently utilizing the underlying hardware, we are able to achieve cluster-like performance from a single desktop machine. Effectively mapping a problem to this platform involves reformulating an algorithm and partitioning it to map each piece into the most appropriate hardware solver. By maximizing the strengths of each device while hiding its weaknesses, an ideal problem mapping can be obtained.

As the need for computational power grows, so does the need for novel approaches to address this demand. It is clear that building faster microprocessors and larger clustered systems is only part of the solution. By utilizing newer hardware technologies, we are able to achieve equivalent computational performance in a much smaller form factor. We see this as a trend that will continue, driven by the need for more detailed simulations and complex analyses.

References:

[1] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, and D. W. Prather, "FPGA-Based Acceleration of the 3D Finite-Difference Time-Domain Method," presented at 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2004.

[2] Z. K. Baker and V. K. Prasana, "Time and area efficient pattern matching on FPGAs," presented at ACM/SIGDA Twelfth ACM International Symposium on Field-Programmable Gate Arrays, 2004.

[3] J. F. Keane, C. Bradley, and C. Ebeling, "A compiled accelerator for biological cell signaling simulations," presented at ACM/SIGDA Twelfth ACM International Symposium on Field-Programmable Gate Arrays, 2004.

[4] H. Quinn, L. A. S. King, M. Leeser, and W. Meleis, "Runtime assignment of reconfigurable hardware components for image processing pipelines," presented at 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2003.

[5] J. Frigo, D. Palmer, M. Gokhale, and M. Popkin-Paine, "Gamma-ray pulsar detection using reconfigurable computing hardware," presented at 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2003.

[6] T. Sumanaweera and D. Liu, "Medical Image Reconstruction with the FFT," in *GPU Gems 2*, M. Pharr, Ed. Boston: Addison-Wesley, 2005.

[7] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication," presented at Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2004.

[8] J. Kruger and R. Westermann, "Linear Algebra Operators for GPU Implementation of Numerical Algorithms," *ACM Transactions on Graphics (TOG)*, vol. 22, pp. 908--916, 2003.

[9] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder, "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," *ACM Transactions on Graphics (TOG)*, vol. 22, pp. 917-924, 2003.

[10] M. J. Harris, "Fast Fluid Dynamics Simulation on the GPU," in *GPU Gems*, R. Fernando, Ed. Boston: Addison-Wesley, 2004, pp. 637-665.

[11] T. J. Purcell, "Ray Tracing on a Stream Processor," Stanford University, March 2004.

[12] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys, "A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware," presented at Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2003.