Journal of Molecular Graphics and Modelling 30 (2011) 82–89

Contents lists available at ScienceDirect

Journal of Molecular Graphics and Modelling

journal homepage: www.elsevier.com/locate/JMGM

Topical perspectives

Topical perspective on massive threading and parallelism

Robert M. Farber*

PNNL, P.O. Box 999, Richland, WA 99352, United States

ARTICLE INFO

Article history:
Received 9 May 2011
Received in revised form 15 June 2011
Accepted 17 June 2011
Available online 29 June 2011

Keywords:
CUDA
OpenCL
Parallel computing
GPU
Computer architecture

ABSTRACT

Unquestionably computer architectures have undergone a recent and noteworthy paradigm shift that now delivers multi- and many-core systems with tens to many thousands of concurrent hardware processing elements per workstation or supercomputer node. GPGPU (General Purpose Graphics Processor Unit) technology in particular has attracted significant attention as new software development capabilities, namely CUDA (Compute Unified Device Architecture) and OpenCL™, have made it possible for students as well as small and large research organizations to achieve excellent speedup for many applications over more conventional computing architectures. The current scientific literature reflects this shift with numerous examples of GPGPU applications that have achieved one, two, and in some special cases, three-orders of magnitude increased computational performance through the use of massive threading to exploit parallelism. Multi-core architectures are also evolving quickly to exploit both massive-threading and massive-parallelism such as the 1.3 million threads Blue Waters supercomputer. The challenge confronting scientists in planning future experimental and theoretical research efforts – be they individual efforts with one computer or collaborative efforts proposing to use the largest supercomputers in the world is how to capitalize on these new massively threaded computational architectures – especially as not all computational problems will scale to massive parallelism. In particular, the costs associated with restructuring software (and potentially redesigning algorithms) to exploit the parallelism of these multi- and many-threaded machines must be considered along with application scalability and lifespan. This perspective is an overview of the current state of threading and parallelize with some insight into the future.

Published by Elsevier Inc.

1. Introduction and application

The trend to massive parallelism appears inescapable as current high-end commodity processors, the workhorse for most scientific computing applications, now support eight or more simultaneous threads of execution per CPU socket. Most computational nodes and scientific workstations contain several multi-core sockets, allowing these systems to deliver sixteen or more concurrent threads of execution. Multi-threaded applications that fully utilize this hardware capability can deliver an order of magnitude increase in computational throughput and corresponding decrease in time-to-solution. A multi-threaded application breaks an application into multiple pieces, or threads of execution, that run concurrently. Highly parallel applications can utilize multi-threading to reduce application runtime by the number of threads, which can be significant when using a newer architectures that supports tens of thousands of parallel threads. Applications that use one or very few threads of execution will likely not benefit from newer architectures.[1]

A 10× performance increase is certainly significant, but it does not necessarily represent a fundamental change for computation-dependent science. Machines with this level of performance make the computational workflow more interactive because computational tasks that previously took hours can finish in minutes and extended computational tasks that previously took days can run overnight.

Applications that deliver 100× or faster performance, as has been demonstrated with General purpose graphics processors units (GPGPU) technology across a broad spectrum of scientific problems, are disruptive and have the potential to fundamentally affect scientific research by removing time-to-discovery barriers. Computational tasks that previously would have required a year to

* Corresponding author.
E-mail address: rmfarber@usa.net

[1] Some single-threaded distributed applications that use MPI or other distributed software frameworks can use each computational core as a separate process. As discussed later, this can be wasteful and begs the question if this is the most computationally efficient mapping for many applications. It also precludes using other architectures such as GPUs.

1093-3263/$ – see front matter. Published by Elsevier Inc.
doi:10.1016/j.jmgm.2011.06.007

complete can finish in days. Better scientific insight becomes possible because researchers can work with more data and have the ability to utilize more accurate, albeit more computationally expensive, approximations and numerical methods.

Dennard's scaling laws are at the heart of the industry-wide change to multi-core processors [1,2]. Effectively they say that power density (voltage×(current/area)) will remain constant even as the number of transistors and their switching speed increases. For that relationship to hold, voltages need to be reduced in proportion to the linear dimensions of the transistor. Fabrication techniques have reduced the size of transistors to the point that manufacturers are no longer able to lower operating voltages sufficiently to match the performance gains that can be achieved by simply adding more computational cores to the processor chip. In a competitive market, minor changes in processor performance do not translate into increased sales for CPU manufacturers – hence the proliferation of multi-core processors. In general, the industry appears to believe this trend will continue and the number of cores per processors will increase.

GPGPUs are an alternative commodity massively parallel architecture that is readily available and inexpensive. Since GPGPU architectures evolved in the highly competitive visualization and gaming market, they were designed from the beginning for massive parallelism. These architectures utilize replication of hardware computational units called shaders to perform real-time visualization tasks in a high-performance and scalable fashion. With the advent of programmable shaders, which have been refined into the current generation of general-purpose streaming multiprocessors, GPGPUs quickly evolved into very capable high performance computational devices that can deliver a number of teraflops (10^{12} floating-point operations per second) of computational capability for graphics and scientific problems. By varying the number of the streaming multiprocessors, low-end devices with tens to hundreds of hardware computational units can be offered at current retail price points under a hundred dollars while high-end devices containing many hundreds to a thousand hardware computational units can be sold at current retail price points of a few thousands of dollars. Since GPGPUs are small and can provide an excellent flop per watt ratio, very large supercomputer clusters are currently being built that utilize both multicore and GPGPU technology. As of the November 2010 TOP500 list, the first and third fastest supercomputers in the world (China's Nebulae and Tehane-1 hybrid CPU/GPU supercomputers) provide examples of this developing trend. For the right applications, GPGPU technology can provide orders of magnitude increased performance for applications running on personal computers to leadership class supercomputers.

Programmable in high-level languages such as the C language with the NVIDIA CUDA (Compute Unified Development Architecture) tools, OpenCL™ and other frameworks, GPGPUs have brought massively parallel supercomputing to the worldwide masses. Freely available software development kits (SDKs) have made programming these devices accessible to anyone from teenagers to large research efforts.

Currently, the NVIDIA CUDA SDK is the most popular. First introduced in February 2007, CUDA is now taught at over 362 universities and academic institutions worldwide. NVIDIA estimates there is already an installed base of over 250 million CUDA-enabled GPGPUs [3]. OpenCL is a standard based alternative to CUDA that provides multi-platform support for products from most hardware vendors such as AMD, NVIDIA, IBM, Intel, Apple, and others. The OpenCL standard is administered by the Khornos group [4]. Source translators such as SWAN [5] can convert CUDA programs to OpenCL. Other software such as MCUDA (CUDA to Multi-core) [6] and Ocelot [7] allow CUDA applications to run on multi-core processors [8], while OpenCL can directly compile to run on multi-core processors. New data-parallel C++ extensions such as Thrust [9] can dramatically simplify GPGPU code development.

Performance, low cost, and power efficiency are three reasons to consider GPGPU technology as legacy application software must be modified or redesigned, which requires a software investment. The benefit is potentially orders of magnitude increased performance for both experimental and simulation based research. As the future is clearly parallel, legacy projects that do not consider investing in software and algorithm development to use one or more of these hardware architectures risk stagnation and loss of competitiveness [10].

The recent scientific and technical literature published during the previous three years demonstrates a proliferation of GPGPU-enabled applications and algorithms that deliver one to two orders of magnitude speedup (10–100×) in performance over conventional processors across a broad spectrum of algorithmic and scientific application areas and additional speedups through the use of multiple GPGPUs either within a workstation or joined together within a computational cluster. This represents a remarkable rate of adoption as CUDA in particular has only been around for 4 years.

Summarizing the breadth and proliferation of applications that map well to GPGPU technology is a challenge. Those applications that achieve high performance are massively threaded so they can fully utilize the many hundreds to thousands of simultaneous hardware threads of execution that become available when one or several GPGPU boards are plugged into a conventional processor motherboard or are joined within a computational cluster.

The NVIDIA Community Showcase [3] provides a central repository that can be used to examine the broad applicability of GPGPU technology to a wide-variety of computational problems as reported in the peer-reviewed and technical literature. While admittedly showcasing applications that perform well, the following graph shows the top 100 fastest reported speedups (maximum 2600×, median 1350×, and minimum 100×) as of May 9, 2011 on NVIDIA CUDA GPUs. Other peer-review surveys reporting both techniques utilized and speedups are appearing [11] (Fig. 1).

While successful GPU applications can deliver impressive performance, it is important to note that GPGPU technology is good for some but not all computational problems. Due to limitations in the SIMD (Single Instruction Multiple Data) execution model and per-thread resources, GPGPUs tend to have a "knife-edge" performance envelope, which means it can be challenging to get high-performance with these devices. Also, it is important to verify that the reported speedups represent realistic performance of a multicore system as opposed to single-core performance. As has been stressed by the Intel Throughput Computing Lab and Architectures Group, further scrutiny of reported results is needed to ensure that equal effort was made to optimize both CPU and GPU implementations [12].

Many authors report GPGPU performance based on single-precision floating-point performance. While double-precision (64-bit) performance is increasing, single-precision (32-bit) performance is still faster on the current generation of GPGPU products. Double-precision performance can be dramatically slower on older generations of GPGPUs. Using mixed-precision arithmetic judiciously, say for reduction operations [13,14] can preserve accuracy along with other techniques for long-running simulations [15].

Many statistical modeling applications map efficiently to GPGPUs. Since they are massively parallel, Monte Carlo methods map well as seen in the 30–100× speedups [16,17] for a single GPU as reported by Salazay and Rohonczy for simulations of dynamic NMR spectra [18]. Other authors report performance increases up to 120× for Monte Carlo simulations using multiple GPUs in an OpenMP framework [19]. In performance surveys of GPU-based