Comput. Methods Appl. Mech. Engrg. 200 (2011) 1490–1508

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

A new era in scientific computing: Domain decomposition methods in hybrid CPU–GPU architectures

M. Papadrakakis, G. Stavroulakis, A. Karatarakis *

Institute of Structural Analysis and Antiseismic Research, National Technical University of Athens, Zografou Campus, Athens 15780, Greece

ARTICLE INFO

Article history:
Received 14 December 2010
Received in revised form 12 January 2011
Accepted 14 January 2011
Available online 19 January 2011

Keywords:
Hybrid computing
Multi-core processing
Many-core processing
Graphics processing units
Domain decomposition methods
FETI method

ABSTRACT

Recent advances in graphics processing units (GPUs) technology open a new era in high performance computing. Applications of GPUs to scientific computations are attracting a lot of attention due to their low cost in conjunction with their inherently remarkable performance features and the recently enhanced computational precision and improved programming tools. Domain decomposition methods (DDM) constitute today an important category of methods for the solution of highly demanding problems in simulation-based applied science and engineering. Among them, dual domain decomposition methods have been successfully applied in a variety of problems in both sequential as well as in parallel/distributed processing systems. In this work, we demonstrate the implementation of the FETI method to a hybrid CPU–GPU computing environment. Parametric tests on implicit finite element structural mechanics benchmark problems revealed the tremendous potential of this type of hybrid computing environment as a result of the full exploitation of multi-core CPU hardware resources and the intrinsic software and hardware features of the GPUs as well as the numerical properties of the solution method.

1. Introduction

In scientific computing, there is a constant need for solving new and highly computationally demanding problems with increased accuracy and enhanced numerical performance. In simulation-based applied science and engineering, there have been considerable improvements in sparse matrix solution algorithms as well as in domain decomposition methods to mitigate execution bottlenecks, thus leading to faster calculation times and reduced memory requirements for the solution of increasingly larger problems. To further increase the speed of their applications, scientists have also relied on advances in hardware and utilization of expensive specialized computing systems with parallel and/or distributed processing capabilities, as well as clusters of interconnected workstations. Moreover, since power density issues limit the increase of the clock frequency, manufacturers have turned to adding more cores to their processors. However, these advancements pose a challenge to software developers since sequential codes run on one of the cores and do not take advantage of the full processing capabilities. Parallel codes do not have this limitation, so incentive for their further development has increased, especially since they have the potential for exploiting the processing power of the graphics processing units (GPUs).

Driven by the demands of the gaming industry, graphics hardware has substantially evolved over the years with remarkable floating point arithmetic performance. These processing capabilities motivated the utilization of graphics hardware for general purpose applications, eventually leading to their initial use for non-graphic operations in 1999. In the early years, these operations had to be programmed indirectly, by mapping them to graphic manipulations and using graphic libraries such as openGL and DirectX. This approach of solving general purpose problems is known as general purpose computing on GPUs (GPGPU). Despite the cumbersome programming, it was soon apparent that the GPUs' potential and capabilities could be utilized for accelerating arithmetic operations, especially since they have considerably lower cost than current supercomputers or workstation clusters.

GPU programming was greatly facilitated with the initial release of the CUDA-SDK [1–3] in 2007, which resulted in a rapid development of GPU computing and the appearance of GPU-powered clusters on the Top500 supercomputers [4]. CUDA, which stands for "compute unified device architecture", is a parallel computing architecture developed by NVIDIA. CUDA gives developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest NVIDIA GPUs become easily accessible for general-purpose applications by eliminating the need for special casting. Recently,

* Corresponding author.
E-mail addresses: mpapadra@central.ntua.gr (M. Papadrakakis), stavroulakis@nessos.gr (G. Stavroulakis), alex@karatarakis.com (A. Karatarakis).

0045-7825/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cma.2011.01.013

openCL has been released as an open industry standard to facilitate portability and vendor-independence, targeting heterogeneous platforms consisting of CPUs, GPUs as well as other types of processors [5]. Unlike CPUs, GPUs have an inherent parallel throughput architecture that focuses on executing many concurrent threads slowly, rather than executing a single thread very fast. Massive hardware multithreading aims to overcome latencies that inevitably derive from device communication. A comparison of the current GPU architecture as well as potential future GPU and modern multi-core processor architecture is provided in [6].

Work pertaining to GPUs has extended to a large variety of applications even before CUDA made their use easier. Non-linear finite element implementations for surgical simulation can be found in [7] with GPGPU and in [8] with CUDA. Engineering applications in the field of fluid mechanics [9–12], molecular dynamics [13,14], topology optimization [15], wave propagation [16], Helmholtz problems with the boundary element method [17], have been recently reported on a variety of GPU platforms using explicit computational algorithms. Linear algebra applications have also been a topic of scientific interest for GPU implementations. A thorough analysis of algorithmic performance of basic linear algebra operations can be found in [18]. Performance of iterative solvers is analyzed in [19], while a parametric study of the PCG solver is performed on multi-GPU CUDA clusters in [20,21]. A hybrid CPU–GPU implementation of dense linear algebra algorithms is reported in [22].

It should be noted that all implementations prior to CUDA 1.3 are performed in single-precision, since support for double-precision floating point operation is added on CUDA 1.3. This has caused some misinterpretations in a number of published comparisons between the GPU and the CPU, usually in favor of the GPU. However, GPUs were (and still are) perfectly suitable for mixed-precision solvers. Performance and accuracy of mixed-precision iterative and multigrid solvers is thoroughly discussed in [23].

Domain decomposition methods (DDM) constitute today an important category of methods for the solution of highly demanding problems in simulation-based applied science and engineering. Among them, dual domain decomposition methods have been successfully applied in a variety of problems in both sequential as well as in parallel/distributed processing systems. In this work, we demonstrate the implementation of the FETI method to a hybrid CPU–GPU computing platforms. DDM are generally considered unsuitable for GPU applications due to their difficulty in exploiting the full capacity of the fine-grained parallelism of the GPUs. However, this weakness of DDM is overcome in the proposed implementation, with customized parallelization routines applied for every part of the solution algorithm. Parametric tests on implicit finite element structural mechanics benchmark problems revealed the tremendous potential of this type of hybrid computing environment as a result of the full exploitation of the intrinsic software and hardware features of the GPUs as well as the numerical properties of the solution method. We believe that, with the exploitation of the processing capabilities of hybrid CPU–GPU systems in conjunction with the implementation of efficient domain decomposition methods, which ensure high hardware utilization and minimize idle time, a new era in scientific computing with great social impact is emerging.

The remainder of the paper is organized as follows: For readers not familiar with CUDA, we briefly explain the programming model in Section 2. In Section 3, the basic steps of the FETI method are described, followed in Section 4 by its algorithmic implementation on a hybrid CPU–GPU workstation. A dynamic load balancing is proposed in this paper between the heterogeneous workstation components and is analyzed in Section 5. Finally, in Section 6, a parametric study is performed on benchmark 3D elasticity problems and the concluding remarks are summarized in Section 7.

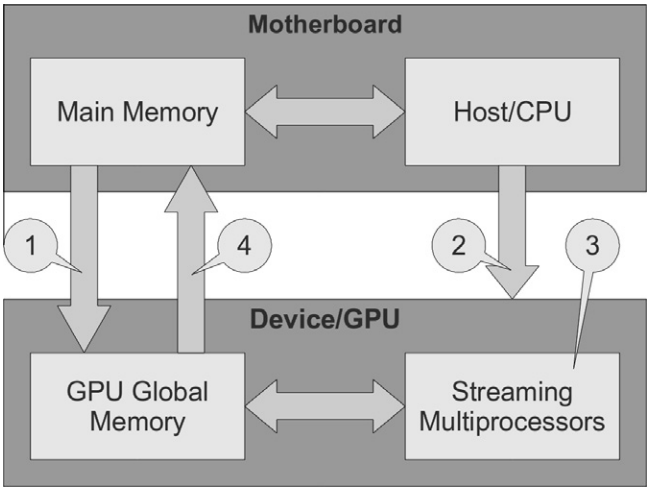

Fig. 1. CUDA processing flow paradigm. (1) Data transfer to GPU memory. (2) CPU instructions to GPU. (3) GPU parallel processing. (4) Result transfer to main memory.

2. Overview of graphics processing units and CUDA environment

GPUs are parallel devices of the SIMD (single instruction, multiple data) classification, which describes devices with multiple processing elements that perform the same operation on multiple data simultaneously and exploit data level parallelism. Programming in CUDA is easier than legacy GPGPU, since it only involves learning a few extensions to C and thus requiring no graphic-specific knowledge. In CUDA programming, the CPU is also referred to as a host and the GPU is also referred to as a device. The general processing flow of CUDA programming is depicted in Fig. 1. GPUs have a large number of streaming processors (SPs) grouped together in streaming multiprocessors (SMPs) (Fig. 2). Each SP has its own arithmetic units. They can collectively offer significantly more gigaflops than current high-end CPUs.

2.1. CUDA threads

The GPU applies the same functions on a large number of data. These data-parallel functions are called kernels. Kernels generate a large number of threads in order to exploit data parallelism, hence the single instruction multiple thread (SIMT) paradigm. A thread is the smallest unit of processing that can be scheduled by an operating system. It generally results from a forking execution into two or more concurrently running tasks. Threads in GPUs take very few clock cycles to generate and schedule due to the GPU's underlying hardware support, unlike CPUs where thousands of clock cycles are required. All threads generated by a kernel define a grid and are organized in blocks. A grid consists of a number of blocks (all equal in size), and each block consists of a number of threads (Fig. 3).

There is another type of thread grouping called warps. Warps are the units of thread scheduling in SMPs. Only one warp can be executed by a SMP at any given time. The number of threads in a warp is specific to the particular hardware implementation – it depends on how many threads the available hardware can process at the same time. The purpose of warps is to ensure high hardware utilization. For example, if a warp initiates a long-latency operation and is waiting for results in order to continue, it is put on hold and another warp is selected for execution in order to avoid having idle processors while waiting for the operation to complete. When the long latency operation completes, the original warp will eventually resume execution. With a sufficient number of warps, the