Studies on nVidia GPUs in Parallel Computing for Lattice Quantum Chromodynamics and Computational Fluid Dynamics Applications

Simone Coscetti
Ph.D. School of Engineering, Unversità di Pisa
National Institute for Nuclear Physics (INFN) - Pisa, Italy
e-mail: simone.coscetti@pi.infn.it

***Abstract*—During the last few years, we have had the opportunity to watch a fast evolution in the devices for scientific computing. Looking at the Top500 Supercomputer Sites rank, we can see that the most important sites are building new infrastructures with accelerators devices beside traditional Central Processing Units (CPUs). One of the main reasons of this behavior is the investigation of a way to reduce power consumption. For this purpose nowadays the trend is to reduce the amount of memory over the single core in each computing device. The most promising approach to reach this goal is the utilization of Graphics Processing Units (GPUs) alongside traditional CPUs. In this paper, we show a study on nVidia GPUs in parallel computing applied to two demanding fields, such as the Lattice Quantum ChromoDynamics and the Computational Fluid Dynamics.**

Keywords-Parallel Computing; Graphics Processing Units; Lattice Quantum Chromodynamics; Computational Fluid Dynamics.

I. INTRODUCTION

Over the last 20 years, the computing revolution has created many social benefits. The computing energy and environmental footprint have grown, and as a consequence the energy efficiency is becoming increasingly important. The evolution toward an always-on connectivity is adding demands for efficient computing performances. The result is a strong market that pulls for technologies that improve processor performance while reducing energy use.

The improvements in energy performance have largely come as a side effect of the Moore's law [1] - the number of transistors on a chip doubles about every two years, thanks to an ever-smaller circuitry. An improvement in better performance and in energy efficiency is due to more transistors on a single computer, with less physical distance between them.

In the last few years, however, the energy-related benefits resulting from the Moore's law are slowing down [2][3][4], threatening future advances in computing. This is caused by the reaching of a physical limit in the miniaturization of transistors.

The industry's answers to this problem for now are new processor architectures and power efficient technologies.

For decades, the CPU of a computer has been the one designated to run general programming tasks, excelling at running computing instructions serially, and using a variety of complex techniques and algorithms in order to improve speed.

GPUs are specialized accelerators originally designed for painting millions of pixels simultaneously across a screen, doing this by performing parallel calculation using simpler architecture. In recent years the video game market developments compelled GPUs manufacturers to increase the floating-point calculation performance of their products, by far exceeding the performance of standard CPUs in floating point calculations, as illustrated in Figure 1. The architecture evolved toward programmable many-core chips that are designed to process in parallel massive amounts of data. These developments suggested the possibility of using GPUs in the field of High-Performance Computing (HPC) as low-cost substitutes of more traditional CPU-based architectures: nowadays such possibility is being fully exploited and GPUs represent an ongoing breakthrough for many computationally demanding scientific fields, providing consistent computing resources at relatively low cost, also in terms of power consumption (watts/flops). Due to their many-core architectures, with fast access to the on-board memory, GPUs are ideally suited for numerical tasks allowing for data parallelism, i.e., for Single Instruction Multiple Data

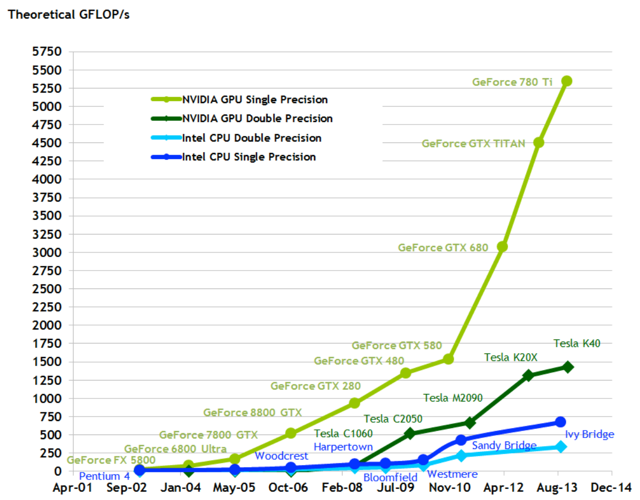


Figure 1. Floating-point operations per seconds for the CPUs and the GPUs [5]

(SIMD) parallelization.

In this work, the parallel computing in Lattice Quantum Chromodynamics (Lattice QCD or LQCD) and in Computing Fluid Dynamics (CFD) using multi-GPUs systems is presented, highlighting the approach of the software in each case, and trying to understand how to build and how to exploit the next generation clusters for scientific computing. In Section 2, the Pisa National Institute for Nuclear Physics (Istituto Nazionale di Fisica Nucleare, INFN) data center is described. The application fields and the computing tools used are introduced in Section 3. In Section 4, the approach to the study and the obtained results are described. Finally, in section 5 the results are discussed.

II. BACKGROUND

The numerical simulation of the path integral formulation of the Quantum Field Theory discretized on a Euclidean space-time lattice (Lattice QCD), and the numerical resolution of Navier-Stokes partial differential equations using discretization methods in CFD are two typical demanding fields, where parallel computing is applied.

This work has been developed at the Pisa INFN computing center [6][7]. Two of the main communities that work in the computing center are a theoretical physics one, specialized on field theory like QCD, and a mechanical engineers one, working on automotive engineering, that exploits the computing center for CFD computing since 2002. The other main intended use of the computing center is the Tier2 of Compact Muon Solenoid (CMS), one of the two main experiments working at the Large Hadron Collider (LHC) in Geneva.

The computing center is made up of 1,5 PB of storage (up to 90% CMS data) and consists so far of four clusters (over 3300 cores) dedicated to theoretical physics and engineering activities.

III. APPLICATION FIELDS

Lattice QCD and Computational Fluid Dynamics are two of the most promising and demanding fields for GPUs computing. But they are not the only ones. GPUs are used for the computing in several application fields such as: weather and ocean modeling, computational chemistry, biology, bioinformatics, metagenomic data analysis, earth and space science, computational finance.

However, these particular application fields have been chosen because they are extremely interesting as typical examples of different application contests.

Lattice QCD simulations enable us to investigate aspects of the QCD physics that would be impossible to systematically investigate in perturbation theory. The computation time for Lattice QCD simulations is a strong limiting factor, bounding for example the usable lattice size. Fortunately enough, the most time consuming kernels of the Lattice QCD algorithms are embarrassingly parallel, so today, in a quest to tackle larger and larger lattices, computations are commonly performed using large computer clusters. Moreover, the use of accelerators, such as GPUs, has been successfully explored for several years [8]. More generally, massively parallel machines based on heterogeneous nodes combining traditional powerful multicore CPUs with energy-efficient and fast accelerators are ideal targets for Lattice QCD simulations and are indeed commonly used. Programming these heterogeneous systems can be cumbersome, mainly because of the lack of standard programming frameworks for accelerator-based machines. In most of the cases, reasonable efficiency requires that the code is re-written targeting a specific accelerator, using proprietary programming languages, such as Compute Unified Device Architecture (CUDA) for nVidia GPUs.

OpenACC offers a different approach based on directives, allowing to port applications onto hybrid architectures by annotating existing codes with specific "pragma" directives. A perspective OpenACC implementation of a Lattice QCD simulation code would grant its portability across different heterogeneous machines without the need of producing multiple versions using different languages. However, the price to pay for code portability may be in terms of code efficiency.

A. CUDA

CUDA [9] is a parallel computing platform and application programming interface (API) model created by nVidia. It allows software developers to use CUDA-enabled GPUs for general purpose processing. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements.

The CUDA platform is designed to work with programming languages, such as C, C++ and Fortran. This accessibility makes it easier for a specialist in parallel programming to utilize GPU resources, as opposed to previous Application Program Interface (API) solutions like Direct3D and OpenGL, which required advanced skills in graphics programming. Also, CUDA supports programming frameworks such as OpenCL and OpenACC.

B. OpenACC

OpenACC [10] is a programming framework for parallel computing aimed to facilitate code development on heterogeneous computing systems, and in particular to simplify porting of existing codes. Its support for different architectures relies on compilers; although at this stage the few available ones target mainly GPU devices, thanks to the OpenACC generality the same code can be compiled for different architectures when the corresponding compilers and run-time supports become available.

OpenACC, like OpenCL, provides a widely applicable abstraction of actual hardware, making it possible to run the same code across different architectures. Contrary to OpenCL, where specific functions (called kernels) have to be explicitly programmed to run in a parallel fashion (e.g., as GPU threads), OpenACC is based on pragma directives that help the compiler identify those parts of the source code that can be implemented as parallel functions. Following pragma instructions the compiler generates one or more kernel functions – in the OpenCL sense – that run in parallel as a set of threads.

Regular C/C++ or Fortran code, already developed and tested on traditional CPU architectures, can be annotated with OpenACC pragma directives (e.g., parallel or kernels clauses) to instruct the compiler to transform loop iterations into distinct threads, belonging to one or more functions to run on an accelerator. Various directives are available, allowing fine-tuning of the application.

IV. Approach, methods and Results

In the following, the GPU computing approach in both Lattice QCD and in CFD fields are presented, and the results concerning performance comparisons are shown.

A. Lattice QCD

From the Lattice QCD side the existing code [11] has been ported and recoded, writing two versions of the code: one in CUDA and one using OpenACC directives. We used a single portion of the original code used for the entirely simulation. In particular, we coded (using plain C) a discretized version of the Dirac matrix, which included two functions D_{eo} and D_{oe}. OpenACC directives instruct the compiler to generate one GPU kernel function for each of them. The function bodies of the OpenACC version strongly resemble the corresponding CUDA kernel bodies, trying to ensure a fair comparison between codes, which perform the same operations. For both the CUDA and OpenACC versions, each GPU thread is associated to a single lattice site. Data structures are allocated in memory following the Structure of Arrays (SoA) layout to obtain better memory coalescing for both vectors and matrices. The basic data element of both the structures is the standard C99 double complex.

We prepared a benchmark code able to repeatedly call the D_{eo} and the D_{oe} functions, one after the other, using the OpenACC implementation or the CUDA one. The two implementations were compiled respectively with the Portland Group (PGI) compiler, version 14.6, and the nVidia nvcc CUDA compiler, version 6.0.

The benchmark code [12] was run on a 32^4 lattice, using an nVidia K20m GPU; results are shown in Table 1, where we list the sum of the execution times of the D_{eo} and D_{oe} operations in nanoseconds per lattice site, for different choices of thread block sizes. All the computations were performed using double precision floating point values.

TABLE 1. EXECUTION TIME PER LATTICE SIZE FOR THE CUDA AND THE OPENACC IMPLEMENTATIONS.

	D_{eo} + D_{oe} functions	
Block size	***CUDA (ns)***	***OpenACC (ns)***
8,8,8	7.58	9.29
16,1,1	8.43	16.16
16,2,1	7.68	9.92
16,4,1	7.76	9.96
16,8,1	7.75	10.11
16,16,1	7.64	10.46

Execution times have a very mild dependence on the block size and for the OpenACC implementation are in general slightly higher; if one considers the best thread block sizes both for CUDA and OpenACC, the latter is ≃ 23% slower.

A slight performance loss with respect to CUDA is expected, given the higher level of the OpenACC language. In this respect, our results are very satisfactory, given the lower programming efforts needed to use OpenACC and the increased code maintainability given by the possibility to run the same code on CPUs or GPUs, by simply disabling or enabling pragma directives. Moreover, OpenACC code performance is expected to slightly improve in the future also due to the rapid development of OpenACC compilers, which at the moment are yet in their early days.

The development of a complete Lattice QCD simulation code fully based on OpenACC is now in progress.

B. CFD

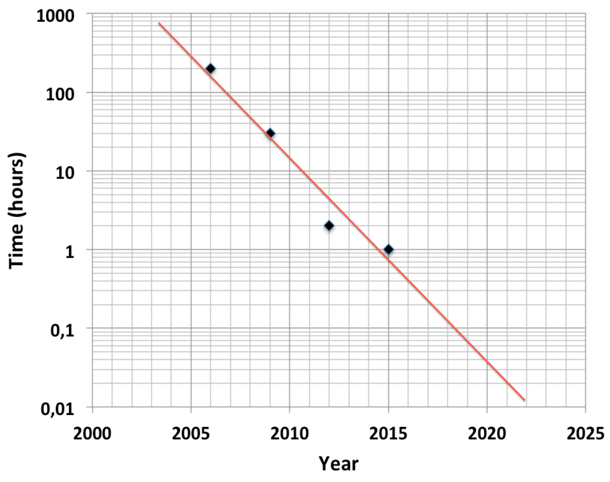
The CFD community that works at the Pisa INFN data center has experienced (from 2002) a reduction of an order of magnitude in the calculation time every 4 years [13]. This huge performance improvement is due basically to the software evolution, and only in small part to the hardware improvement.

During the same period, the number of manageable cells is improved too, from 5 millions in 2002, to 40 millions today. In this case, the hardware is mainly responsible.

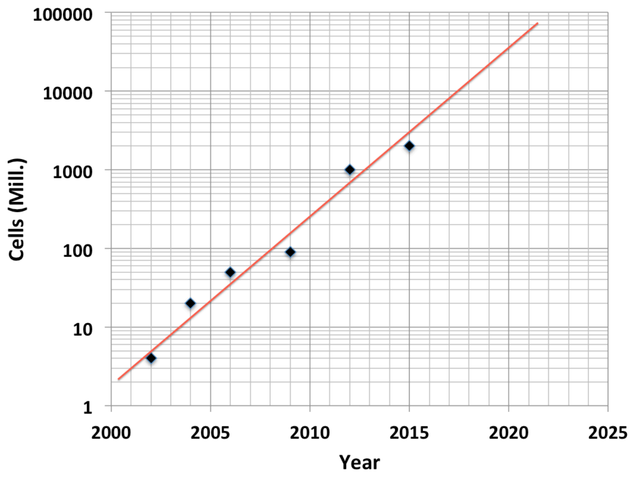
Assuming that the actual trend continues in the next years, the expected scenario is shown in Figure 2.

The ability of simulating 40 millions cells lattice in 1 minute of computation time and managing even 20 billions cells is not trivial. The processors architecture will have to evolve to maintain the trend. The implication will be that both the software and the methodological approach need to evolve and update.

The most promising approach is represented by the employment of GPUs in the simulation chain. However, this technology isn't mature yet, but a version of ANSYS Fluent software enabled to exploit GPUs computing has



(a)



(b)

Figure 2. (a) Computing time and (b) number of cells analyzed during past years with an estimation of the performances in the next years.

been released in late 2013.

One of the main aspects to be considered is that the software behaves differently if GPUs are involved in the computing. In this case, there is an "agglomeration" step on the CPU before the data are sent to the GPUs, and naturally there is a backward step once data have been processed by the GPUs [14][15].

This procedure requires additional time to be performed, and it has two main effects: i) the total execution time will not necessarily be reduced by the exploitation of the GPUs; ii) the performance improvement is difficult to evaluate, taking into account the case and the machine architecture.

The performance improvement depends substantially on the fraction of the whole computation time dedicated to the "Linear Solver". More generally, using only CPUs remains convenient in those cases where CPUs have huge loads (high number of cells per core) and when every single iteration is expensive (double precision, coupled equations, thermal exchange).

The test case is an 8.3 millions cells one, with density base solver, k-epsilon turbulence model [16], with the equation energy activated and a second order discretization accuracy.

The case has been run on a machine set up in the following way:

- CPU: Intel Xeon® E5-2650 v2 @ 2.60 GHz; 2 Socket x 8 cores; 128 GB memory;
- GPU: 2x Tesla K40m, Kepler GK110B, 2880 CUDA cores @ 745 MHz; 12 GB memory.

In Figure 3, one can see how the performance improvement using GPUs decreases increasing the number of CPU cores involved. The improvement has been totally cancelled using all the 16 cores in the machine. We have to underline that due to the limited computing power of the machine, the adopted test case is a lightweight one, both for the cells number and for the model analyzed.

In the case of two CPU cores the benefit of using two GPUs amount to a satisfying 30%.

V. Conclusions

Current high performance computing (HPC) systems in the Top500 list have reached petaflops of computational power. Every decade, the computing power of HPC systems increases by a factor of ten. This leads to exascale systems within the next decade. One of the major challenges will be the power. Accelerators offer an unprecedented computational power per watt. Therefore, they are becoming a vital part in todays and future HPC systems. Few dozens systems in the recent (November 2015) Top500 Supercomputing sites list are using either coprocessor or GPGPU technology, including Tianhe-2, Titan, Pin Daint and Stampede in the Top10. Heterogeneous systems combine accelerators and multicore CPU nodes to achieve a better performance while being more energy efficient. One of the major

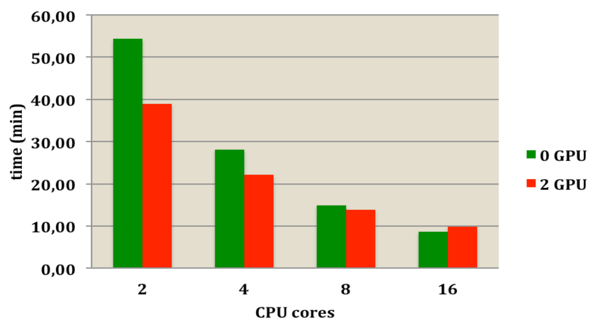


Figure 3. Improvement in AMG with GPUs on a single machine.

challenges of these systems is the scalability between host CPUs and accelerators.

In order to prepare a mid-size computing center such as the Pisa INFN one for the advent of new architectures dedicated to scientific computing, this work has been finalized to the study of two of the most promising and demanding fields: Lattice Quantum Chromodynamics and Computational Fluid Dynamics. These two fields are extremely interesting as typical examples of different context.

Lattice QCD scientists have a “home-made” code for their simulations, and the approach to a multi-GPU system has been made by porting and recoding the existent code. Different programming approaches, such as CUDA and OpenACC, have been tested, measuring the performances in each case.

Instead, CFD simulations are performed running commercial software (ANSYS Fluent) and the code able to run in a CPUs-GPUs system is not adjustable. In CFD there are two different performance evaluation contexts: how many cells the system can simulate, to more and more defined simulations, and how much time the system spends to perform a simulation with a fixed number of cells, the ideal context for optimizations activities. CFD approach to GPU technology is in its early years, but both hardware and software are proceeding in the right direction.

In particular, for the CFD field, this work represents an in-depth examination whose final target will be the fulfillment of a heterogeneous cluster made of CPUs and GPUs, fully dedicated to the CFD simulations. The purpose is to reach the computing power of 1 PetaFlop, starting from todays 16 TeraFlops cluster (classical CPU-based). The groups concerned to reach the goal will be the CFD-skilled engineers to better understand the simulating strategies, and the ANSYS Fluent experts to configure, set-up and optimize the simulation runs.

GPUs approach is not the only one. The utilization of Xeon Phi coprocessors alongside traditional CPUs has been tested in various environment and application fields and the opinion is that the new generation advent (Knights Landing model) will extend the range of possible applications in fields such as Lattice QCD and CFD.

REFERENCES

[1] G. E. Moore, "Progress in digital integrated electronics", Electron Devices Meeting, vol. 21, 1975, pp. 11-13.

[2] S. Thompson and S. Parthasaranthy, "Moore's law: the future of Si microelectronics", Materials Today, vol. 9, issue 6, 2006, pp. 20-25.

[3] L. B. Kish, "End of Moore's law: thermal (noise) death of integration in micro and nano electronics", Phys. Letters A, vol. 305, issue 3, 2002, pp. 144-149.

[4] R. A. Robison, "Moore's law: predictor and driver of the silicon era", World Neurosurgery, vol. 78, issue 5, 2012, pp. 399-403.

[5] http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_ Guide.pdf [retrieved: February 2016].

[6] S. Arezzini et al., "Optimization of HEP analysis activity using a Tier2 infrastructure", Jou. of Physics: Conf. series, vol. 396, part. 4, 2012, pp. 2054-2059.

[7] S. Arezzini et al., "INFN-Pisa scientific computation environment (GRID, HPC and Interactive Analysis)", Jou. of Physics: Conf. series, vol. 513, track 6, 2014, 062030.

[8] G. I. Egri et al., "Lattice QCD as a video game", Com. Phys. Comm., vol. 177, Issue 8, 2007, pp. 631-639.

[9] https://developer.nvidia.com/about-cuda [retrieved: February, 2016].

[10] http://www.openacc.org [retrieved: February, 2016].

[11] C. Bonati, G. Cossu, M. D'Elia, and P. Incardona, "QCD simulations with staggered fermions on GPUs", Com. Phys. Comm., vol. 183, Issue 4, 2012, pp 853-863.

[12] C. Bonati et al., "Development of scientific software for HPC architectures using OpenACC: the case of LQCD", 2015 IEEE/ACM 1st International Workshop on Software Engineering for High Performance Computing in Science (SE4HPCS), May 2015, pp. 9-15, doi:10.1109/SE4HPCS.2015.9.

[13] A. Ciampa, S. Coscetti, G. Lombardi, and M. Maganzi, "Impact of the power computing breakthrough on the aerodynamic project: an outlook on GPUs utilization in ANSYS-FLUENT", Ansys Convergence, 2015 Regional Conference, 2015, unpublished.

[14] D. Gaudlitz, B. Landmann, and T. Indinger, "Accelerated CFD simulations using Eulerian and Lagrangian methods on GPUs", Procedia Engineering, vol. 61, 2013, pp. 392-397.

[15] V. Sellappan and B. Desam, "Accelerating Ansys Fluent simulations with Nvidia GPUs", Ansys Advantage, vol. IX, issue 1, 2015, pp. 51-53.

[16] B. Mohammadi and O. Pironneau, "Analysis of the k-epsilon turbolence model", 1993.