Computational Science, Engineering and Technology Series: 34

Patterns for Parallel Programming on GPUs

Computational Science, Engineering and Technology Series

Substructuring Techniques and Domain Decomposition Methods
Edited by: F. Magoulès

Soft Computing in Civil and Structural Engineering
Edited by: B.H.V. Topping and Y. Tsompanakis

Trends in Civil and Structural Engineering Computing
Edited by: B.H.V. Topping, L.F. Costa Neves and R.C. Barros

Parallel, Distributed and Grid Computing for Engineering
Edited by: B.H.V. Topping and P. Iványi

Trends in Engineering Computational Technology
Edited by: M. Papadrakakis and B.H.V. Topping

Trends in Computational Structures Technology
Edited by: B.H.V. Topping and M. Papadrakakis

Computational Methods for Acoustics Problems
Edited by: F. Magoulès

Mesh Partitioning Techniques and Domain Decomposition Methods
Edited by: F. Magoulès

Computational Mechanics using High Performance Computing
Edited by: B.H.V. Topping

High Performance Computing for Computational Mechanics
Edited by: B.H.V. Topping, L. Lämmer

Parallel and Distributed Processing for Computational Mechanics: Systems and Tools
Edited by: B.H.V. Topping

Saxe-Coburg Publications:

Programming Distributed Finite Element Analysis: An Object Oriented Approach
R.I. Mackie

Object Oriented Methods and Finite Element Analysis
R.I. Mackie

Domain Decomposition Methods for Distributed Computing
J. Kruis

Computer Aided Design of Cable-Membrane Structures
B.H.V. Topping and P. Iványi

Patterns for Parallel Programming on GPUs

Edited by
F. Magoulès

published 2014 by
Saxe-Coburg Publications
Civil-Comp Ltd, Dun Eaglais, Station Brae
Kippen, Stirlingshire, FK8 3DY, Scotland

Saxe-Coburg Publications is an imprint of Civil-Comp Ltd

Computational Science, Engineering and Technology Series: 34
ISSN 1759-3158
ISBN 978-1-874672-57-9

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Publisher's production & editorial team: Steven Miller, Jane Tabor, Rosemary Brodie
Printed in Great Britain by Bell & Bain Ltd, Glasgow

Contents

Foreword

Our modern society is increasingly relying on computers. They are ubiquitous in our daily life even if they are hidden in most cases (in appliances, cars, telephones, USB keys, *etc.*). Digital television (TV) would not be the unique broadcasting system today without important compute resources at the recording end and in the display. TV sets nowadays use advanced graphics processing units (GPU) to upscale digital Versatile Disc (DVD) images (not to mention BluRays®) or pan and zoom your favorite show. The anti-lock braking system (ABS) of your car is controlled by a computer. The latest three-dimensional scan of a baby would not be possible without computers and powerful GPUs. This list would be too tedious to make exhaustive.

In the scientific and technical world, computers also play an ever growing role, frequently replacing experiments. Numerical simulation has an increasing economical weight in various domains ranging from car crashes to the evaluation of the biological effects of a new pharmaceutical molecule. Simulations have replaced experiments in many cases either because they reduce costs by allowing many variations of experimental conditions (in destructive experiments such as car crashes) or simply because experiments are not feasible (*e.g.* seeing a protein unfolding in which case *in vitro* experimentation has been replaced by *"in silico"* analysis).

In all cases, using simulations help the scientists to understand complex phenomena. Yet this understanding is closely related to the size of the simulation: the finer the simulation, the greater the insight can be. A finer simulation is for example to predict the weather for areas of $10km^2$ instead of $100km^2$. As a consequence the larger a simulation is, the more compute capability is required. For example, dividing the size of a cubic simulation by two in each dimension of space increases the number of elements to process by 8 (which means that the processing time also has increased by a factor of 8 or more, depending on the type of simulation). Users (or customers) are not always ready to pay for such an increase in simulation time and they dream of speedier computers solving larger problems in a constant time. Since computers are powered by electricity, deploying more compute power leads to more electrical consumption. This cannot be an endless process for the price of electricity will eventually limit the growth of the supercomputers. Current petaflop machines are in the range of 5MW. With a simple extrapolation, an exaflop machine could be in the order of 100MW or more without a technology change.

To overcome this practical (infrastructure size) and economic limit (*i.e.* the budget to run the installation in the long run), a true disruptive technology is required and is in fact already emerging. The evolution of supercomputers processing elements is parallel to the one of commodity hardware (*e.g.* smartphones): the number of process-

ing units per chip is increasing to deliver greater performances within a constrained energy budget (limit the number of megawatts for the former, longer battery life for the latter). This is especially true of the usage of graphical processing units for general purpose computations (GPGPU).

Envisioned at CEA as early as 2007, the use of GPU computing is becoming widespread (medicine, avionics, HPC) if not mainstream (Apple is harnessing the GPU power through OpenCL and its OS X, your smartphone relies of its GPU to playback videos and so on). GPUs solve part of the equation for a number of applications: they are of an order of magnitude more efficient than regular processors for the same electrical needs, on some algorithms. An increasing number of success stories demonstrates that GPUs are here to stay for a long time in high performance computing (either embedded or not). In that respect, the Titane and Curie machines hosted at CEA (see `http://www-hpc.cea.fr`) are good examples of advanced architectures which have found a community of users for their ability to produce results for the hardest problems yet in a limited time.

The introduction of new hardware compute technologies forces evolutions in the software stack used by programmers. Operating systems, compiler chains as well as libraries are not immune to the nature of the underlying hardware. But the development of a comprehensive software stack takes a significant amount of time. It is therefore very important to start the developments as soon as possible to make sure that the code developers have all the necessary tools for their daily job, when the given technology will spread to the masses. It means also that those developers will have to change their programming habits and adapt themselves to the new technologies.

GPU computing is still a vast field of research which encompasses various domains such as programming languages, compiler technologies, parallelization techniques or linear algebra methods. Such research is fundamental in every field of computer science. They aim to increase the productivity of the developers, yield a better maintainability of the code and reach the best possible performances. The impact of the results achieved is enhanced if the underlying technologies rely on open specifications and are themselves made available open to the whole community of developers and users. Along this line, OpenCL is emerging as the only viable low level technology to drive many forms of compute units, yet in a portable manner.

OpenGPU, a project of the systematic cluster of competitiveness, is clearly at the crossroads of all these needs (such as high performance or portability) and techniques (use of a GPU, CUDA or OpenCL programming). OpenGPU has proved to be a good platform to go forward in the understanding of the pros and cons of GPUs and, at the same time, to promote GPUs' diffusion through the development of a viable ecosystem. This book, entitled *Patterns for Parallel Programming on GPU* edited by Frédéric Magoulès, summarizes some of the achievements of the OpenGPU project. It is an invitation for the reader to deep dive into the fascinating world of GPU programming and usage.

Guillaume Colin de Verdière
CEA, France

Preface

At present, multi-core and many-core platforms lead the computer industry, forcing software developers to adopt new programming paradigms, in order to fully exploit their computing capabilities. Graphics Processing Units (GPUs) are one representative of many-core architectures, and certainly the most widespread.

GPU-based application development requires a great effort from application programmers. On one hand, they must take advantage of the massively parallel platform in the problem modeling. On the other, the applications have to make an efficient use of the heterogeneous memory system, managing several levels that are software or hardware controlled. Generally the programmers' methodology consists in evaluating several mapping alternatives, guided by their experience and intuition, which becomes inefficient for software development and maintenance.

In order to help the programmers to make the good choices, this book presents in thirteen chapters some methodologies and design patterns for parallel programing on GPUs. Each chapter, written by different authors, presents a state-of-the-art of some innovative methods, techniques or algorithms, useful for GPU computing. A bibliography is included at the end of each chapter for the reader who wishes to go further.

This book starts at Chapter 1 with an evaluation of state-of-the-art parallelizing compilers generating CUDA code for heterogeneous computing. This chapter evaluates and compares tool frameworks that automatically generate code for GPUs, saving time and effort to programmers. These frameworks take advantage of polyhedral model techniques to exploit parallelism and to satisfy the specific GPU constraints. The authors show the key features of some of these source-to-source compilers and analyze the codes generated. Finally, the authors discuss the importance of some key aspects such as data mapping and code quality.

In Chapter 2, the authors present an unusual strategy to perform dynamic code generation for GPUs. Usual compiler relies on assumption that are not always true, which can lead to sub-optimal code due to a lack of information available to the compiler. By using a code generator called deGoal that can produce a code in a pseudo-assembly code representation for GPUs, the authors show how to dynamically generate code usable by the GPU. This chapter illustrates the tool usage by the matrix multiplication on various configurations. This example show that it allows to develop an application and get near optimal results faster than developping a specialized version.

Chapter 3 shows how to achieve on GPUs great performance with applications commonly handled by CPU only. This hybrid system leads to complex programming designs combining multiple paradigms to manage each hardware architecture. The

authors present how parallel skeletons can help tackle this challenge by abstracting some of these programming designs while automating optimizations. Through a simple example using the OCaml programming language to develop and compose skeletons, this chapter demonstrates how simple modifications in using parallel skeletons can ease GPU programming while offering good performance speed-ups.

Chapter 4 shows how data flow applications can be efficiently programmed on GPUs from a unique high level capture. The authors rely on a tooling approach, through the SPEAR design environment, to point out the underlying productivity gain with regard to performance. For efficient code generation purposes, several optimisations at different levels are detailed, followed by some numerical experiments performed on a representative radar application.

Chapter 5 proposes a study of the optimization process of parallel applications run on modern hybrid architectures. Different optimization schemes are proposed for overlapping computations with communications; and for computation kernels. Development methodologies are introduced to obtain different optimization degrees and specific criteria are defined to help developers to find the most suited degree of optimization according to the application and parallel system considered. Both the performance and code complexity increase are analyzed.This last point is an important issue, as it directly impacts on development and maintenance costs. Experiments are performed to evaluate the different variants of a benchmark application that consists in a dense matrix product.

Porting and maintaining multiple versions of a code base require different skills; and the efforts required in the process, as well as the increased complexity in debugging and testing, are time consuming. Some solutions based on compilers emerge. They are based either on directives added to *C* in openhmpp or openacc or on automatic solutions like pocc, Pluto, ppcg, or par4all. However compilers cannot retarget in an efficient way any program written in a low-level language such as unconstrained *C*. Programmers should follow good practices when writing code so that compilers have more room to perform the transformations required for efficient execution on heterogeneous targets.

Chapter 6 explores the impact of different patterns used by programmers, and defines a set of good practices allowing a compiler to generate efficient code.

Chapter 7 introduces the Melt framework and domain-specific language to extend the Gcc compiler. It explains the major internal representations (Gimple, Tree-s, *etc.*) and the overall organization of Gcc. It shows the major features of Melt and illustrates why extending and customizing the Gcc compiler using Melt is useful; for instance, to use GPUs through OpenCL. It gives some concrete advice and guidelines for the development of such extensions with Melt.

In Chapter 8, the authors study the opportunities that OpenCL offers to high performance computing applications to provide a solution to unify new developments. In order to overcome the lack of native OpenCL support for some architectures, the authors survey the third-party research work that propose a source-to-source approach to transform OpenCL into other parallel programming languages. For instance, FPGAs are considered as a case study, because of their dramatic OpenCL support compared

to GPUs. These transformation approaches could also lead to potential works in the model driven engineering (MDE) field as conceptualized in this chapter. Moreover, OpenCL's standard API is quite rough. Thus, the authors introduce several APIs from the simple high-level binder to the source code generator to ease and boost the development process of any OpenCL application.

In Chapter 9, the authors propose a parallel implementation of the preconditioned conjugate gradient algorithm on a GPU platform. The preconditioning matrix is a first order approximate inverse derived from the SSOR preconditioner. Used through sparse matrix-vector multiplication, the preconditioner proposed by the authors is well-suited to massively parallel architecture like GPUs. As compared to CPU implementation of the conjugate gradient algorithm, the GPU preconditioned conjugate gradient implementation is between eight and sixteen times faster.

In Chapter 10, the authors present the implementation on GPUs of two classical methods for solving sparse linear systems. Those methods are the conjugate gradient method which is usually used for solving symmetric linear systems, and the generalized minimal residual method (GMRES), which is commonly used for solving unsymmetric linear systems. For each method, the authors describe how they have adapted the sequential version for the GPU with the CUDA programming environment. The performances of these parallel algorithms are tested and analyzed on GPU and on CPU clusters.

Non-coding ribonucleic acid (RNA) are functional RNAs that are not translated into proteins. Computational studies of non-coding RNAs have recently become an important challenge in bioinformatics, including their identification and structure prediction. In Chapter 11, the authors present several algorithms for these applications. With the development of next-generation sequencing technologies, huge amounts of genomic and RNA sequences data have been produced, and a parallelization of such tools is required to overcome the long execution time. In this chapter, miRNAFold, an algorithm developed by the authors for the search for microRNAs in genomic sequences, is described together with its implementation in CUDA on GPUs.

Chapter 12 aims to provide both a feedback and a methodological approach on how to port a legacy application to GPU clusters for quantitative metagenomic data, *i.e.*, studying many organisms with whole deoxyribonucleic acid (DNA) information, without getting access to the single and pure species information. The proposed approach is based on the clustering principles that rely on a large correlation matrix computation on polled data. To allow fast computation and get the best of the GPU, many optimizations to cover several optimization strategies are investigated by the authors. This chapter explores dynamic and static optimizations and presents benchmarks on supercomputers and small clusters of GPUs.

Testing random numbers relies on a heavy battery of statistical tests that can take hours of computations or small samplings of numbers. In Chapter 13, the authors demystify this slow checking process by showing the parallel nature of two representative tests and how to implement them on GPU with OpenCL. The authors illustrates these concepts with two examples of test codes translated from *C* to OpenCL. In addition, an original algorithm, made in the context of testing the quality of the true

random number generator is illustrated, and provides a totally new and safe way of dealing with random number generator.

Naturally, the present book cannot provide a complete record of the many approaches, applications, features and schemes related to GPUs. However, it does provide a useful synopsis of the recent methodologies used in academic circles and in industry to handle efficiently hybrid architectures. This book will be of interest to engineers, computer scientists and applied mathematicians. The editor wishes to thank the authors for their willingness to contribute to this book dedicated to patterns for parallel programming on GPUs.

Frédéric Magoulès
Ecole Centrale Paris, France