Introduction to High Performance Computing for Scientists and Engineers

Introduction to High Performance Computing for Scientists and Engineers

Georg Hager

Gerhard Wellein

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number: 978-1-4398-1192-4 (Paperback)

Library of Congress Cataloging-in-Publication Data

Hager, Georg.
Introduction to high performance computing for scientists and engineers / Georg Hager and Gerhard Wellein.
p. cm. -- (Chapman & Hall/CRC computational science series ; 7)
Includes bibliographical references and index.
ISBN 978-1-4398-1192-4 (alk. paper)
1. High performance computing. I. Wellein, Gerhard. II. Title.

QA76.88.H34 2011
004'.35--dc22 2010009624

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Dedicated to Konrad Zuse (1910–1995)

He developed and built the world’s first fully automated, freely programmable computer with binary floating-point arithmetic in 1941.

Contents

Foreword

Georg Hager and Gerhard Wellein have developed a very approachable introduction to high performance computing for scientists and engineers. Their style and descriptions are easy to read and follow.

The idea that computational modeling and simulation represent a new branch of scientific methodology, alongside theory and experimentation, was introduced about two decades ago. It has since come to symbolize the enthusiasm and sense of importance that people in our community feel for the work they are doing. Many of us today want to hasten that growth and believe that the most progressive steps in that direction require much more understanding of the vital core of computational science: *software and the mathematical models and algorithms it encodes*. Of course, the general and widespread obsession with hardware is understandable, especially given exponential increases in processor performance, the constant evolution of processor architectures and supercomputer designs, and the natural fascination that people have for big, fast machines. But when it comes to advancing the cause of computational modeling and simulation as a new part of the scientific method there is no doubt that the complex software "ecosystem" it requires must take its place on the center stage.

At the application level science has to be captured in mathematical models, which in turn are expressed algorithmically and ultimately encoded as software. Accordingly, on typical projects the majority of the funding goes to support this translation process that starts with scientific ideas and ends with executable software, and which over its course requires intimate collaboration among domain scientists, computer scientists, and applied mathematicians. This process also relies on a large infrastructure of mathematical libraries, protocols, and system software that has taken years to build up and that must be maintained, ported, and enhanced for many years to come if the value of the application codes that depend on it are to be preserved and extended. The software that encapsulates all this time, energy, and thought routinely outlasts (usually by years, sometimes by decades) the hardware it was originally designed to run on, as well as the individuals who designed and developed it.

This book covers the basics of modern processor architecture and serial optimization techniques that can effectively exploit the architectural features for scientific computing. The authors provide a discussion of the critical issues in data movement and illustrate this with examples. A number of central issues in high performance computing are discussed at a level that is easily understandable. The use of parallel processing in shared, nonuniform access, and distributed memories is discussed. In addition the popular programming styles of OpenMP, MPI and mixed programming are highlighted.

We live in an exciting time in the use of high performance computing and a period that promises unmatched performance for those who can effectively utilize the systems for high performance computing. This book presents a balanced treatment of the theory, technology, architecture, and software for modern high performance computers and the use of high performance computing systems. The focus on scientific and engineering problems makes it both educational and unique. I highly recommend this timely book for scientists and engineers, and I believe it will benefit many readers and provide a fine reference.

Jack Dongarra

University of Tennessee
Knoxville, Tennessee
USA

Preface

When Konrad Zuse constructed the world's first fully automated, freely programmable computer with binary floating-point arithmetic in 1941 [H129], he had great visions regarding the possible use of his revolutionary device, not only in science and engineering but in all sectors of life [H130]. Today, his dream is reality: Computing in all its facets has radically changed the way we live and perform research since Zuse's days. Computers have become essential due to their ability to perform calculations, visualizations, and general data processing at an incredible, ever-increasing speed. They allow us to offload daunting routine tasks and communicate without delay.

Science and engineering have profited in a special way from this development. It was recognized very early that computers can help tackle problems that were formerly too computationally challenging, or perform *virtual* experiments that would be too complex, expensive, or outright dangerous to carry out in reality. *Computational fluid dynamics*, or CFD, is a typical example: The simulation of fluid flow in arbitrary geometries is a standard task. No airplane, no car, no high-speed train, no turbine bucket enters manufacturing without prior CFD analysis. This does not mean that the days of wind tunnels and wooden mock-ups are numbered, but that computer simulation supports research and engineering as a third pillar beside theory and experiment, not only on fluid dynamics but nearly all other fields of science. In recent years, *pharmaceutical drug design* has emerged as a thrilling new application area for fast computers. Software enables chemists to discover reaction mechanisms literally at the click of their mouse, simulating the complex dynamics of the large molecules that govern the inner mechanics of life. On even smaller scales, *theoretical solid state physics* explores the structure of solids by modeling the interactions of their constituents, nuclei and electrons, on the quantum level [A79], where the sheer number of degrees of freedom rules out any analytical treatment in certain limits and requires vast computational resources. The list goes on and on: Quantum chromodynamics, materials science, structural mechanics, and medical image processing are just a few further application areas.

Computer-based simulations have become ubiquitous standard tools, and are indispensable for most research areas both in academia and industry. Although the power of the PC has brought many of those computational chores to the researcher's desktop, there was, still is and probably will ever be this special group of people whose requirements on storage, main memory, or raw computational speed cannot be met by a single desktop machine. High performance parallel computers come to their rescue.

Employing high performance computing (HPC) as a research tool demands at least a basic understanding of the hardware concepts and software issues involved. This is already true when only using turnkey application software, but it becomes essential if code development is required. However, in all our years of teaching and working with scientists and engineers we have learned that such knowledge is *volatile* — in the sense that it is hard to establish and maintain an adequate competence level within the different research groups. The new PhD student is all too often left alone with the steep learning curve of HPC, but who is to blame? After all, the goal of research and development is to make *scientific progress*, for which HPC is just a tool. It is essential, sometimes unwieldy, and always expensive, but it is still a tool. Nevertheless, writing efficient and parallel code is the admission ticket to high performance computing, which was for a long time an exquisite and small world. Technological changes have brought parallel computing first to the departmental level and recently even to the desktop. In times of stagnating single processor capabilities and increasing parallelism, a growing audience of scientists and engineers must be concerned with performance and scalability. These are the topics we are aiming at with this book, and the reason we wrote it was to make the knowledge about them less volatile.

Actually, a lot of good literature exists on all aspects of computer architecture, optimization, and HPC [S1, R34, S2, S3, S4]. Although the basic principles haven't changed much, a lot of it is outdated at the time of writing: We have seen the decline of vector computers (and also of one or the other highly promising microprocessor design), ubiquitous SIMD capabilities, the advent of multicore processors, the growing presence of ccNUMA, and the introduction of cost-effective high-performance interconnects. Perhaps the most striking development is the absolute dominance of x86-based commodity clusters running the Linux OS on Intel or AMD processors. Recent publications are often focused on very specific aspects, and are unsuitable for the student or the scientist who wants to get a fast overview and maybe later dive into the details. Our goal is to provide a solid introduction to the architecture and programming of high performance computers, with an emphasis on performance issues. In our experience, users all too often have no idea what factors limit time to solution, and whether it makes sense to think about optimization at all. Readers of this book will get an intuitive understanding of performance limitations without much computer science ballast, to a level of knowledge that enables them to understand more specialized sources. To this end we have compiled an extensive bibliography, which is also available online in a hyperlinked and commented version at the book's Web site: http://www.hpc.rrze.uni-erlangen.de/HPC4SE/.

Who this book is for

We believe that working in a scientific computing center gave us a unique view of the requirements and attitudes of users as well as manufacturers of parallel computers. Therefore, everybody who has to deal with high performance computing may profit from this book: Students and teachers of computer science, computational engineering, or any field even marginally concerned with simulation may use it as an accompanying textbook. For scientists and engineers who must get a quick grasp of HPC basics it can be a starting point to prepare for more advanced literature. And finally, professional cluster builders can definitely use the knowledge we convey to provide a better service to their customers. The reader should have some familiarity with programming and high-level computer architecture. Even so, we must emphasize that it is an introduction rather than an exhaustive reference; the *Encyclopedia of High Performance Computing* has yet to be written.

What's in this book, and what's not

High performance computing as we understand it deals with the *implementations* of given algorithms (also commonly referred to as "code"), and the *hardware* they run on. We assume that someone who wants to use HPC resources is already aware of the different algorithms that can be used to tackle their problem, and we make no attempt to provide alternatives. Of course we have to pick certain examples in order to get the point across, but it is always understood that there may be other, and probably more adequate algorithms. The reader is then expected to use the strategies learned from our examples.

Although we tried to keep the book concise, the temptation to cover everything is overwhelming. However, we deliberately (almost) ignore very recent developments like modern accelerator technologies (GPGPU, FPGA, Cell processor), mostly because they are so much in a state of flux that coverage with any claim of depth would be almost instantly outdated. One may also argue that high performance input/output should belong in an HPC book, but we think that efficient parallel I/O is an advanced and highly system-dependent topic, which is best treated elsewhere. On the software side we concentrate on basic sequential optimization strategies and the dominating parallelization paradigms: shared-memory parallelization with OpenMP and distributed-memory parallel programming with MPI. Alternatives like Unified Parallel C (UPC), Co-Array Fortran (CAF), or other, more modern approaches still have to prove their potential for getting at least as efficient, and thus accepted, as MPI and OpenMP.

Most concepts are presented on a level independent of specific architectures, although we cannot ignore the dominating presence of commodity systems. Thus, when we show case studies and actual performance numbers, those have usually been obtained on x86-based clusters with standard interconnects. Almost all code examples are in Fortran; we switch to C or C++ only if the peculiarities of those languages are relevant in a certain setting. Some of the codes used for producing benchmark results are available for download at the book's Web site: http://www.hpc.rrze.uni-erlangen.de/HPC4SE/.

This book is organized as follows: In Chapter 1 we introduce the architecture of modern cache-based microprocessors and discuss their inherent performance limitations. Recent developments like multicore chips and simultaneous multithreading (SMT) receive due attention. Vector processors are briefly touched, although they have all but vanished from the HPC market. Chapters 2 and 3 describe general optimization strategies for serial code on cache-based architectures. Simple models are used to convey the concept of "best possible" performance of loop kernels, and we show how to raise those limits by code transformations. Actually, we believe that performance modeling of applications on all levels of a system's architecture is of utmost importance, and we regard it as an indispensable guiding principle in HPC.

In Chapter 4 we turn to parallel computer architectures of the shared-memory and the distributed-memory type, and also cover the most relevant network topologies. Chapter 5 then covers parallel computing on a theoretical level: Starting with some important parallel programming patterns, we turn to performance models that explain the limitations on parallel scalability. The questions why and when it can make sense to build massively parallel systems with "slow" processors are answered along the way. Chapter 6 gives a brief introduction to OpenMP, which is still the dominating parallelization paradigm on shared-memory systems for scientific applications. Chapter 7 deals with some typical performance problems connected with OpenMP and shows how to avoid or ameliorate them. Since cache-coherent nonuniform memory access (ccNUMA) systems have proliferated the commodity HPC market (a fact that is still widely ignored even by some HPC "professionals"), we dedicate Chapter 8 to ccNUMA-specific optimization techniques. Chapters 9 and 10 are concerned with distributed-memory parallel programming with the Message Passing Interface (MPI), and writing efficient MPI code. Finally, Chapter 11 gives an introduction to hybrid programming with MPI and OpenMP combined. Every chapter closes with a set of problems, which we highly recommend to all readers. The problems frequently cover "odds and ends" that somehow did not fit somewhere else, or elaborate on special topics. Solutions are provided in Appendix B.

We certainly recommend reading the book cover to cover, because there is not a single topic that we consider "less important." However, readers who are interested in OpenMP and MPI alone can easily start off with Chapters 6 and 9 for the basic information, and then dive into the corresponding optimization chapters (7, 8, and 10). The text is heavily cross-referenced, so it should be easy to collect the missing bits and pieces from other parts of the book.

Acknowledgments

This book originated from a two-chapter contribution to a Springer "Lecture Notes in Physics" volume, which comprised the proceedings of a 2006 summer school on computational many-particle physics [A79]. We thank the organizers of this workshop, notably Holger Fehske, Ralf Schneider, and Alexander Weisse, for making us put down our HPC experience for the first time in coherent form. Although we extended the material considerably, we would most probably never have written a book without this initial seed.

Over a decade of working with users, students, algorithms, codes, and tools went into these pages. Many people have thus contributed, directly or indirectly, and sometimes unknowingly. In particular we have to thank the staff of HPC Services at Erlangen Regional Computing Center (RRZE), especially Thomas Zeiser, Jan Treibig, Michael Meier, Markus Wittmann, Johannes Habich, Gerald Schubert, and Holger Stengel, for countless lively discussions leading to invaluable insights. Over the last decade the group has continuously received financial support by the "Competence Network for Scientific High Performance Computing in Bavaria" (KONWIHR) and the Friedrich-Alexander University of Erlangen-Nuremberg. Both bodies shared our vision of HPC as an indispensable tool for many scientists and engineers.

We are also indebted to Uwe Küster (HLRS Stuttgart), Matthias Müller (ZIH Dresden), Reinhold Bader, and Matthias Brehm (both LRZ München), for a highly efficient cooperation between our centers, which enabled many activities and collaborations. Special thanks goes to Darren Kerbyson (PNNL) for his encouragement and many astute comments on our work. Last, but not least, we want to thank Rolf Rabenseifner (HLRS) and Gabriele Jost (TACC) for their collaboration on the topic of hybrid programming. Our Chapter 11 was inspired by this work.

Several companies, through their first-class technical support and willingness to cooperate even on a nonprofit basis, deserve our gratitude: Intel (represented by Andrey Semin and Herbert Cornelius), SGI (Reiner Vogelsang and Rüdiger Wolff), NEC (Thomas Schönemeyer), Sun Microsystems (Rick Hetherington, Ram Kunda, and Constantin Gonzalez), IBM (Klaus Gottschalk), and Cray (Wilfried Oed).

We would furthermore like to acknowledge the competent support of the CRC staff in the production of the book and the promotional material, notably by Ari Silver, Karen Simon, Katy Smith, and Kevin Craig. Finally, this book would not have been possible without the encouragement we received from Horst Simon (LBNL/NERSC) and Randi Cohen (Taylor & Francis), who convinced us to embark on the project in the first place.

Georg Hager & Gerhard Wellein

Erlangen Regional Computing Center
University of Erlangen-Nuremberg
Germany

About the authors

Georg Hager is a theoretical physicist and holds a PhD in computational physics from the University of Greifswald. He has been working with high performance systems since 1995, and is now a senior research scientist in the HPC group at Erlangen Regional Computing Center (RRZE). Recent research includes architecture-specific optimization for current microprocessors, performance modeling on processor and system levels, and the efficient use of hybrid parallel systems. His daily work encompasses all aspects of user support in high performance computing such as lectures, tutorials, training, code parallelization, profiling and optimization, and the assessment of novel computer architectures and tools.

Gerhard Wellein holds a PhD in solid state physics from the University of Bayreuth and is a professor at the Department for Computer Science at the University of Erlangen. He leads the HPC group at Erlangen Regional Computing Center (RRZE) and has more than ten years of experience in teaching HPC techniques to students and scientists from computational science and engineering programs. His research interests include solving large sparse eigenvalue problems, novel parallelization approaches, performance modeling, and architecture-specific optimization.

List of acronyms and abbreviations

ASCII	American standard code for information interchange
ASIC	Application-specific integrated circuit
BIOS	Basic input/output system
BLAS	Basic linear algebra subroutines
CAF	Co-array Fortran
ccNUMA	Cache-coherent nonuniform memory access
CFD	Computational fluid dynamics
CISC	Complex instruction set computer
CL	Cache line
CPI	Cycles per instruction
CPU	Central processing unit
CRS	Compressed row storage
DDR	Double data rate
DMA	Direct memory access
DP	Double precision
DRAM	Dynamic random access memory
ED	Exact diagonalization
EPIC	Explicitly parallel instruction computing
Flop	Floating-point operation
FMA	Fused multiply-add
FP	Floating point
FPGA	Field-programmable gate array
FS	File system
FSB	Frontside bus
GCC	GNU compiler collection
GE	Gigabit Ethernet
GigE	Gigabit Ethernet
GNU	GNU is not UNIX
GPU	Graphics processing unit
GUI	Graphical user interface

HPC	High performance computing
HPF	High performance Fortran
HT	HyperTransport
IB	InfiniBand
ILP	Instruction-level parallelism
IMB	Intel MPI benchmarks
I/O	Input/output
IP	Internet protocol
JDS	Jagged diagonals storage
L1D	Level 1 data cache
L1I	Level 1 instruction cache
L2	Level 2 cache
L3	Level 3 cache
LD	Locality domain
LD	Load
LIKWID	Like I knew what I'm doing
LRU	Least recently used
LUP	Lattice site update
MC	Monte Carlo
MESI	Modified/Exclusive/Shared/Invalid
MI	Memory interface
MIMD	Multiple instruction multiple data
MIPS	Million instructions per second
MMM	Matrix–matrix multiplication
MPI	Message passing interface
MPMD	Multiple program multiple data
MPP	Massively parallel processing
MVM	Matrix–vector multiplication
NORMA	No remote memory access
NRU	Not recently used
NUMA	Nonuniform memory access
OLC	Outer-level cache
OS	Operating system
PAPI	Performance application programming interface
PC	Personal computer
PCI	Peripheral component interconnect
PDE	Partial differential equation
PGAS	Partitioned global address space

PLPA	Portable Linux processor affinity
POSIX	Portable operating system interface for Unix
PPP	Pipeline parallel processing
PVM	Parallel virtual machine
QDR	Quad data rate
QPI	QuickPath interconnect
RAM	Random access memory
RISC	Reduced instruction set computer
RHS	Right hand side
RFO	Read for ownership
SDR	Single data rate
SIMD	Single instruction multiple data
SISD	Single instruction single data
SMP	Symmetric multiprocessing
SMT	Simultaneous multithreading
SP	Single precision
SPMD	Single program multiple data
SSE	Streaming SIMD extensions
ST	Store
STL	Standard template library
SYSV	Unix System V
TBB	Threading building blocks
TCP	Transmission control protocol
TLB	Translation lookaside buffer
UMA	Uniform memory access
UPC	Unified parallel C