

Teaching High Performance Computing to scientists and engineers: A model-based approach

Dr. Georg Hager¹, Prof. Dr. Gerhard Wellein^{1,2}, Dr.-Ing. Jan Treibig¹

¹Erlangen Regional Computing Center (RRZE)
Friedrich-Alexander University of Erlangen-Nuremberg
Martensstraße 1
91058 Erlangen, Germany

²Department of Computer Science
Friedrich-Alexander University of Erlangen-Nuremberg
Martensstraße 3
91058 Erlangen, Germany

`{georg.hager,gerhard.wellein,jan.treibig}@rrze.uni-erlangen.de`

The authors submit on behalf of the HPC Services group at Erlangen Regional Computing Center (RRZE)

Abstract

Over the past ten years we have developed a course called “Programming Techniques for Supercomputers,” whose aim is to make students of computational sciences and engineering familiar with the concepts and dominant programming paradigms of high performance computing (HPC). We believe that working in a scientific computing center for more than a decade and helping scientists to optimize and parallelize their application codes has endowed us with a unique view and deep understanding of the relevant issues in this field.

For a long time, parallelism on the application level was all but absent from computer science and related curricula. Students had no solid understanding about the relevant performance-limiting aspects of modern computer systems, and how to write code that makes good use of the architecture. Hence, optimization efforts were often “shots in the dark” without proper guidance and no concept of how fast an application could potentially run.

Our teaching concept begins with a coverage of modern computer architecture, as far as it is relevant for parallel applications in HPC. From the start we put a strong emphasis on architectural features that may have an impact on performance - qualitatively and also quantitatively. Pipeline stalls, latency and bandwidth bottlenecks, interconnect topologies in shared and distributed memory parallel computers, and the particular properties of modern multicore processors are treated in depth. Bandwidth-based performance models are used extensively to predict the performance of loop kernels and identify sensible optimization efforts. Simple performance tools that provide a manageable level of details and thus are useful to application programmers are presented. We then turn to parallelism on a theoretical level: Basic scalability models and appropriate refinements are used to develop a quantitative understanding of the limits of parallelism. Finally we give introductions into the dominant programming paradigms in scientific computing, OpenMP and MPI, and their specific optimization potentials. We close with a coverage of hybrid MPI+OpenMP programming and its possible performance implications.

Starting from the lecture, for which materials (slides, problems, solutions) can be found in our online learning management system, we have developed four follow-up branches that took up the concept and developed it further for a wider audience:

- A series of tutorials, which have been successful and well-attended at various conferences (SC10, PPOPP11, ISC10). Some of the material has also been part of tutorials organized by our scientific collaborators.

- A textbook, "Introduction to High Performance Computing for Scientists and Engineers" (CRC Press, ISBN13 978-1439811924), which has been adopted by various lectures and courses, and is generally well received in the HPC community. Apart from the main text it also contains many problems with solutions, and an extensive bibliography for further reading. An annotated version of the bibliography (containing, e.g., all abstracts) is also available on the web.
- Industry collaborations, which have emerged naturally out of the increased visibility due to our other activities. This has also led to an interesting extension of scope into fields not directly connected to classic numerical simulation, like medical applications and reactor physics. We have also noticed that there is an urgent need in industry for HPC training, which often cannot be satisfied by internal company resources.
- A spin-off company, "Likwid High Performance Programming," which offers commercial HPC consulting, training, and services to companies outside the academic sector.

In conclusion, our concept of teaching parallel computing with a strong focus on performance limitations and useful models has made a measurable difference within and beyond our institution, and is – to our knowledge – unique in the HPC community.

1 Description of the achievements

1.1 Lecture “Programming Techniques for Supercomputers” (PTfS)

The aim of the PTfS lecture is to make students of computational sciences and engineering familiar with the concepts and dominant programming paradigms of high performance computing (HPC). The format of the one term course is two 90-minute lectures plus one 90-minute tutorial per week. To get the full 7.5 ECTS credits, students are required to score at least 50% of the tutorial credits, and they have to pass an oral exam.

The current trends in computer architecture no longer allow us to separate code optimization from parallelization. Doing both together and in a structured way will be of vital importance for many software areas in the years to come; hence, the lecture treats efficient serial code and parallel programming techniques as complementary concepts, and on an equal footing. It starts with a short introduction to current supercomputer systems and the basics of correct benchmarking. Then we provide a survey of modern computer architecture, as far as it is relevant for parallel applications in HPC. Traditional courses on computer architecture in computer science curricula are often unsuitable for the computational scientist, since the relevant features that determine application performance are obscured by theoretical ballast. In contrast to that, we put a strong emphasis on features that may have an impact on performance – qualitatively and also quantitatively: Pipeline stalls, latency and bandwidth bottlenecks, the properties of modern multicore processors, and interconnect topologies in shared and distributed memory parallel computers are treated in depth. At every stage we try to pinpoint the performance-limiting aspects of each particular architectural feature. Bandwidth-based performance models are used extensively to predict the performance of loop kernels and identify sensible optimization efforts. We always try to make clear that “shot-in-the-dark” optimizations, i.e., code changes that are not guided by a clear understanding of what speedup could potentially be achieved, are second-rate optimizations. Our experience with programmers who put large efforts into code restructuring only to find out that their application was already limited by the architecture brought us to the conclusion that this way of thinking about optimization is desperately needed and must be taught to students from the start.

We then turn to parallelism on a theoretical level: Basic scalability models and appropriate refinements are used to develop a quantitative understanding of the limits of parallelism. In particular, we extend the standard discussion of Amdahl’s and Gustafson’s Laws by incorporating the influence of communication overhead, and describe what mathematical form the laws take for different problem classes and/or network properties. We also derive mathematically why and how “slow computing,” i.e., the use of relatively slow and low-power processor and system technology in large-scale parallel systems, can lead to scalability and also performance advantages compared to standard clusters. Typical mistakes and misconceptions with preparing and presenting parallel performance data are addressed in due detail.

Finally we give introductions into the dominant programming paradigms in scientific computing, OpenMP and MPI, and optimization potentials that are specific to either approach. In both cases, the performance discussion is again closely connected with computer architecture. E.g., students are made aware of the influence on multicore topology (the actual arrangement of cores with respect to the resources they share, like caches and memory

interfaces) on parallel performance, and of which actions must be taken to ensure that threads and processes are bound to hardware resources to best avoid bottlenecks. We dedicate ample time to explaining the impact of ccNUMA architecture on the performance of memory-bound code, and which programming measures can avoid the typical nonlocality and contention problems. Although we still consider GPGPU computing to be in a state of flux with regard to long-term availability of hardware and software solutions, we also give an introduction into the architecture of GPUs, as far as it is relevant for HPC, and the CUDA framework.

We close with a discussion of hybrid MPI+OpenMP programming and its possible performance implications. Numerous misconceptions exist about hybrid programming even on this ostensibly simple level, and many publications still approach the subject in an incoherent and even dilettantish way. After explaining the basic features of using MPI with threaded processes we introduce the concepts of “vector mode” vs. “task mode” hybrid programming and how the latter may be employed to reliably implement truly asynchronous communication. Finally we provide a survey of possible pros and cons of MPI+OpenMP programming in different application scenarios, and put a strong emphasis on the concept that a well-scalable MPI-only code that behaves in accordance with a suitable performance model is not a candidate for hybrid parallelization.

Owing to the complexity and depth of the material, some of the more peripheral topics are treated in the tutorials. Students are often required to pinpoint or solve a specific performance problem, but we also cover correctness issues in OpenMP and MPI code. A strong emphasis is put on correct benchmarking, and we try to convey all necessary knowledge needed for writing the most efficient low-level code possible.

1.2 Other teaching activities

1.2.1 Multicore programming tutorial

We have developed a tutorial lecture with the title “Ingredients for good parallel performance on multicore-based systems.” It has a typical length of 3-3.5 hours, and fits well into the fringe activities of conferences with HPC related scope. The material mostly stays at the node level and is not just a subset of PTfS, but also contains “best practices” and case studies we do not typically teach in the lecture. It is primarily targeted at practitioners who want to get quickly “up to scratch” with performance issues and simple tools on current multicore hardware. Hence, it does not usually contain hands-on sessions, but these could easily be added if required.

Contents of tutorial “Ingredients for good parallel performance on multicore-based systems:”

1 Introduction

- Architecture of multsocket multicore systems
- Nomenclature
- Current developments
- Programming models

2 Multicore performance tools

- Finding out about system topology
- Affinity enforcement
- Performance counter measurements

3 Impact of processor/node topology on program performance

- Bandwidth saturation effects
- Programming for ccNUMA
- OpenMP performance
- Simultaneous multithreading (SMT)
- Intranode vs. internode MPI

4 New chances with multicore hardware

- Wavefront parallelization of stencil codes
- Explicit overlap of computation and communication in sparse matrix-vector multiply

Summary

Appendix

1.2.2 Compact courses and workshops

The PTfS lecture and the multicore tutorial have served as starting points for numerous other teaching activities, especially in the form of compact courses or workshop participation. Materials are specially tailored to meet the needs of the respective audience. The following is a list of such activities in recent years:

- *Efficient multicore programming*. Annual compact lecture (3-4 days) with tutorials at the Ohm University of Applied Sciences, Nuremberg. While the scope of this course had been general parallel programming and optimization for shared and distributed-memory systems, we have changed the focus to multicore processors starting in 2009. (2004-2011)
- *Parallel programming of high performance systems*. Annual one-week course with tutorials in collaboration with Leibniz Supercomputing Center in Garching (LRZ). The course is announced all over Germany, and the location traditionally alternates between RRZE and LRZ. We provide access via video conferencing for other locations, given sufficient demand. In recent years the course was complemented by a 3-day “advanced” class covering performance tools and parallel I/O in depth. (2000-2011)
- *Efficient multithreaded programming on modern CPUs and GPUs*. One-week compact course at KTH Stockholm, March 14-18, 2011.

1.3 Textbook “Introduction to High Performance Computing for Scientists and Engineers”

Georg Hager and Gerhard Wellein

CRC Press, ISBN 978-1439811924, 356 pages, July 2010

The textbook grew out of the PTfS lecture and two contributions to a Springer volume in the “Lecture Notes in Physics” series [1][2], but also from our experience with scientific users who try to run and optimize their simulation codes on the machines of RRZE and other centers. It encompasses almost the complete PTfS lecture, but also provides some additional information, e.g., on C++ optimizations and efficient use of MPI. Apart from the main text it contains many problems with solutions (derived and extended from the problems in earlier PTfS tutorials), and an extensive bibliography for further reading. An annotated and updated version of the bibliography (containing, e.g., all abstracts) and additional information about the book is also available on the book’s Web site [3]. The book is the main accompanying text for PTfS students.

Table of contents for the book “Introduction to High Performance Computing for Scientists and Engineers:”

1 Modern Processors

Stored-program computer architecture
General-purpose cache-based microprocessor architecture
Memory hierarchies
Multicore processors
Multithreaded processors
Vector processors

2 Basic Optimization Techniques for Serial Code

Scalar profiling
Common sense optimizations
Simple measures, large impact
The role of compilers
C++ optimizations

3 Data Access Optimization

Balance analysis and lightspeed estimates
Storage order
Case study: The Jacobi algorithm
Case study: Dense matrix transpose

Algorithm classification and access optimizations Case study: Sparse matrix-vector multiply

4 Parallel Computers

Taxonomy of parallel computing paradigms
Shared-memory computers
Distributed-memory computers
Hierarchical (hybrid) systems
Networks

5 Basics of Parallelization

Why parallelize?
Parallelism
Parallel scalability

6 Shared-Memory Parallel Programming with OpenMP

Short introduction to OpenMP
Case study: OpenMP-parallel Jacobi algorithm
Advanced OpenMP: Wavefront parallelization

7 Efficient OpenMP Programming

Profiling OpenMP programs

Performance pitfalls
Case study: Parallel sparse matrix-vector multiply

8 Locality Optimizations on ccNUMA

Architectures

Locality of access on ccNUMA
Case study: ccNUMA optimization of sparse MVM
Placement pitfalls
ccNUMA issues with C++

9 Distributed-Memory Parallel Programming with MPI

Message passing
A short introduction to MPI
Example: MPI parallelization of a Jacobi solver

10 Efficient MPI Programming

MPI performance tools

Communication parameters
Synchronization, serialization, contention
Reducing communication overhead
Understanding intranode point-to-point communication

11 Hybrid Parallelization with MPI and OpenMP

Basic MPI/OpenMP programming models MPI taxonomy of thread interoperability
Hybrid decomposition and mapping
Potential benefits and drawbacks of hybrid programming

Appendix A: Topology and Affinity in Multicore Environments

Appendix B: Solutions to the Problems

1.4 Collaboration with industry

The disruptive multi- and manycore technologies driven by Moore's Law and the unacceptable power consumption of chips with fast clock rates makes optimization and parallelization of code mandatory in almost all sectors of software development. However, code efficiency and parallel programming have not been a part of regular curricula in computer science, let alone in physics, chemistry, or even engineering. As a consequence, software developers in the industry are facing high obstacles when trying to achieve good performance on modern hardware. Thus it is not surprising that several companies have recently approached the HPC services group at RRZE to discuss options for receiving support on various levels: Scientific collaborations, help with optimization and parallelization of code, and, in particular, training.

1.5 Likwid High Performance Programming

"Likwid High Performance Programming" [4] is a spin-off company recently founded by Jan Treibig. It offers commercial HPC consulting, training, and services to companies outside the academic sector.

It is not always possible for a university to act as a business partner in the commercial sector. Questions of accountability, billing, and certain contract conditions that are not compatible with an academic institution leave only a narrow scope for business activities. Likwid High Performance Programming makes it easy for the group to offer such services. Since it is the stated goal of the university to foster the transfer of knowledge to the commercial world, there is no problem with using training material that was prepared for academic teaching in the context of commercial training.

2 Availability of the material to the teaching community

2.1 Teaching material

All teaching material from the PTfS lecture, i.e., lecture slides, problem assignments, and sample code, is available via our "Moodle" learning management system (LMS) even for non-enrolled guests [5]. Solution slides to the problems are provided on request.

The LMS is also used to conduct our other courses and workshops, notably those described in Section 1.2.2. The conference tutorials (see Section 1.2.1) are not included here, since they do not contain a hands-on component

and are usually not longer than a day. We provide general information about the tutorials and links to the slides on the Web [6].

We have set up a Web page for our HPC textbook [3], from which some source code from the examples in the book can be downloaded. It also includes the list of all references from the book, including abstracts and updates, where applicable.

2.2 LIKWID toolkit

The LIKWID toolkit [7] is a collection of lightweight tools, which have been developed by Jan Treibig. They support programmers of multicore systems with running multithreaded code in the most efficient way and provide feedback on performance issues using hardware counter measurements. LIKWID has recently been incorporated into the PTfS lecture, since there has been a painful lack of performance tools that students (and computational scientists) can actually use right away without much prior training.

3 Impact

3.1 PTfS lecture

The majority of PTfS students come traditionally from the computational engineering (CE) program at the University of Erlangen-Nuremberg, since it is a “compulsory optional subject” in the CE syllabus. However, there are also frequent guest students – undergraduate as well as postgraduate – from other departments like physics, chemistry, applied mathematics, and chemical engineering.

3.2 Textbook

Our textbook has been well received in the community, and had very positive reviews from IEEE [8] and ACM [9]. After only ten months of availability, several lectures on parallel or scientific computing list it as recommended literature [10][11][12]. It has also been included in the list of books about supercomputing on the Top500 Web site [13].

3.3 Tutorials and workshops

Our multicore tutorial has been presented at two international conferences so far: Supercomputing 2010 (SC10) in New Orleans, LA, and Principles and Practice of Parallel Programming 2011 (PPoPP11) in San Antonio, TX. Feedback forms for attendees were only available at SC10. The tutorial was well attended (about 70 participants), and 39 feedback sheets were returned. Average ratings were 4.5 overall and 4.6 for technical content (scale from 1=poor to 5=excellent). This compares with an average overall rating of 4.2 across all SC10 tutorials.

Parts of the tutorial material have been a component of the popular “Hybrid MPI and OpenMP Parallel Programming” tutorial by R. Rabenseifner, G. Hager, and G. Jost, which has been presented at all Supercomputing conferences since 2007, and has received very positive feedback each year. At the International Supercomputing Conference 2011 (ISC11) in Hamburg (June 19-23), a full-day tutorial called “Performance oriented programming on multicore-based clusters with MPI, OpenMP, and hybrid MPI/OpenMP” has been accepted and will be presented by G. Hager, G. Jost, J. Treibig, and G. Wellein [14]. It is in essence a fusion of the two tutorials mentioned above.

The long-standing parallel programming course developed in collaboration with LRZ (see Sect. 1.2.2) has continuously received strong participation from universities all over Germany, and recently also from industry. On average, between 20 and 30 attendees visit the course.

3.4 Industry collaborations

Two relevant collaborations with industry have emerged so far out of the increased visibility of our group due to the numerous teaching activities:

- The University of Erlangen-Nuremberg has signed a contract with the Schaeffler Group [15], a well known supplier of the automotive industry and one of the leading manufacturers of rolling bearings. Within the contract, the HPC Services group at RRZE supports Schaeffler in provisioning and configuring a compute cluster environment for company-wide use, and also provides know-how for program optimization and parallelization.
- Areva [16], a global leader in the nuclear power industry, has recently agreed to order several optimization and OpenMP tutorials at company locations worldwide. Those will be conducted by our “LIKWID High Performance Programming” spin-off.

Several contacts have also been established with different parts of Siemens AG. Negotiations have just started.

5 References

- [1] G. Hager and G. Wellein: *Architectures and Performance Characteristics of Modern High Performance Computers*. In Fehske et al., Lect. Notes Phys. 739, 681-730 (2008), ISBN: 978-3-540-74685-0 [2] G. Hager and G. Wellein: *Optimization Techniques for Modern High Performance Computers*. In Fehske et al., Lect. Notes Phys. 739, 731-767 (2008), ISBN: 978-3-540-74685-0
- [3] <http://www.hpc.rze.uni-erlangen.de/HPC4SE>
- [4] <http://www.likwid-software.com>
- [5] PTfS 2011 (ongoing):
<http://moodle.rze.uni-erlangen.de/moodle/course/view.php?id=145&username=guest&password=guest>
PTfS 2010: <http://moodle.rze.uni-erlangen.de/moodle/course/view.php?id=112>
PTfS 2009: <http://moodle.rze.uni-erlangen.de/moodle/course/view.php?id=73>
PTfS 2008: <http://moodle.rze.uni-erlangen.de/moodle/course/view.php?id=40>
PTfS 2007: <http://moodle.rze.uni-erlangen.de/moodle/course/view.php?id=18>
Older lectures are not available online any more.
- [6] <http://blogs.fau.de/hager/tutorials/>
- [7] <http://code.google.com/p/likwid>
- [8] Computing in Science & Engineering, Vol. 13 Issue 1, pp. 5-8, Jan./Feb. 2011 (access is restricted; see attached hardcopy).
<http://dx.doi.org/10.1109/MCSE.2011.3>
- [9] ACM Computing Reviews, 04/28/11, Review # CR139012 (access is restricted; see attached hardcopy).
http://www.reviews.com/review/review_reviewprint.cfm?review_id=139012
- [10] J. Demmel, K. Yelick: *Applications of Parallel Computers*. U.C. Berkeley CS267/EngC233, spring 2011. http://www.cs.berkeley.edu/~demmel/cs267_Spr11/
- [11] D. Göttsche, S. Turek: *High Performance Computing und parallele Numerik*. University of Dortmund, summer 2011.
<http://www.mathematik.tu-dortmund.de/de/studiumlehre/vorlesungsverzeichnis/vorlesung/semester=SS11/lec=011200.htm>
- [12] Z. Bai: Large Scale Scientific Computing. U. C. Davis ECS 231, spring 2011.
<http://www.cs.ucdavis.edu/~bai/ECS231/>
- [13] <http://top500.org/resources/books>
- [14] http://www.supercomp.de/isc11_ap/index.php?mod=2¶m=::Tutorials,Workshops:::0
- [15] <http://www.schaeffler.com>
- [16] <http://www.aveva.com>

Attachments

- Book reviews “Introduction to High Performance Computing for Scientists and Engineers”

and Repast pages to see how they can turn the outstanding theory in this book into actual code.

In sum, *Networks, Crowds, and Markets* is an exceptional book. In a time where people are talking about e-books heralding the eventual demise of books in general, this is a book you're going to want to own—in print. The authors seem committed to keeping the prepublication draft available on their

site in PDF, which is a great service to humanity and cash-strapped students, so you can even read it for free before purchasing your copy. I managed to learn a great deal about game theory and markets (topics I never studied as a student) and am planning on incorporating these ideas in my teaching and in future research projects.

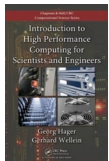
George K. Thiruvathukal is a computer scientist and interdisciplinary researcher

who codirects Loyola University Chicago's new Center for Textual Studies and Digital Humanities. He has written books for Prentice Hall PTR and Sun Microsystems Press and knows how difficult it is to make money writing books, even when you write a good one. Thiruvathukal is associate editor in chief for *CiSE* and for *Computing Now* (an online initiative of the IEEE Computer Society). For more information, please visit <http://home.thiruvathukal.com>.

A HIGHLY USEFUL HPC REFERENCE FOR SCIENTISTS AND ENGINEERS

By Hang Liu

G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, Chapman & Hall/CRC Press, Computational Science Series, 2011, ISBN: 978-1-4398-1192-4, 330 pp.



Books on high-performance computing typically fall under two headings. One type is concerned with developing high-performance parallel algorithms for specific tasks, such as linear algebra operations and fast Fourier transforms. The other type discusses how to implement a given algorithm and execute it on different types of modern parallel computer systems. This book by Georg Hager and Gerhard Wellein is a complete and well-organized example of the latter, with a clear main theme: how to achieve and improve scientific computing performance through multiple concurrency and data locality.

Content Overview

In Chapter 1, Hager and Wellein introduce modern processor designs,

including chip transistor counts and clock speeds. They start by emphasizing concepts for improving application performance using multiple concurrency and data locality at hardware and instruction level. Examples they discuss include pipelined functional units, superscalar architectures, single instruction multiple data (SIMD), memory cache hierarchies, and multithreaded processors.

Chapters 2 and 3 deal with performance optimization techniques for serial code. In high-performance computing, data access is the most important factor that limits performance. Hence, the focus here is on data access efficiency, and the authors offer many valuable tips. They also introduce balance analysis and bandwidth-based performance modeling, then demonstrate them using the Stream benchmarks to show the hardware's practical capabilities in real applications.

Chapter 4 introduces parallel computing at the application level. It discusses parallelization's benefit and power, as well as the factors that limit parallel performance. These factors include communication/synchronization costs, load balancing, and parallelism's inherent overhead. The authors quantify the details of the parallel performance model, efficiency, speed-up, scaling, and scalability metrics in Chapter 5. Readers must understand these essential concepts if they want to

- analyze the parallel performance they can expect,
- interpret the reported parallel performance analysis,
- measure an application's parallel performance, and
- identify opportunities for improving parallel performance in an application.

Having developed all the major concepts of high-performance parallel computing, the authors focus on techniques and tools for developing parallel applications in the rest of the book. For example, Chapters 6 and 9 discuss how to develop parallel applications using the most accepted APIs—that is, OpenMP on shared memory


systems and MPI on distributed memory systems—while Chapter 11 focuses on MPI/OpenMP hybrid programming. These three chapters are brief. Although definitely too short for serious developers, the chapters provide a simple yet clear introduction for beginners. This truncation is a wise choice; OpenMP and MPI have been the de facto standards for shared and distributed memory programming for a long time, and rich references for them are available elsewhere at a various sophistication levels.

Highs and Lows

Much of this book is commendable: there are many figures in the chapters and a rich bibliography at the end. The figures make it easy to understand the book's points, especially when presenting performance results. The bibliography is organized by topic, making it easy for readers to check for further references on subjects of interest.

As with all computer science books that involve developing technologies,


there will be disappointments when the newest hardware isn't discussed. The authors mention in the preface that they deliberately ignore parallel I/O systems whose efficiency is heavily dependent on hardware specifics. While it's true that parallel I/O is a complex subject, it still deserves an examination. With the increased parallel scale of scientific and engineering computing, I/O is easily a bottleneck to application and system performance. Some relevant I/O performance tips would be a helpful supplement to the book's focus on data access performance. Also, the authors ignore the recent developments of modern accelerator technologies and the partitioned global address space (PGAS) programming models. Currently, many accelerator-enabled clusters are available to research communities. One would hope for a brief outlook section or chapter reviewing the opportunities—and limitations—that these tools offer in terms of performance, programmability, and productivity. Perhaps in the next edition.

Overall, Hager and Wellein have written a great book; it's self-consistent and clear, full of practical and invaluable experiences, and is especially suitable as an in-hand reference for scientists and engineers interested in applying high-performance computing to their research. The case studies and problems at the end of the chapters resonate with practical performance tips and suggestions useful to anyone interested in high-performance computing—including me, a scientist at a supercomputer center who often helps and trains fellow researchers in these techniques. 

Hang Liu is a research associate in the Texas Advanced Computing Center at the University of Texas, Austin. His research interests include algorithm development, implementation, validation, and performance optimization to suite large-scale physics calculations on modern high-performance computing architectures. Liu has a PhD in physics from Ohio University. Contact him at hliu@tacc.utexas.edu.

Why

YOU BELONG



as a Member of IEEE Computer Society

Need to keep up with the newest developments in computing and IT?

Looking to enhance your knowledge and skills?

Want to shape the future of your profession?

If you answered “yes” to any of these questions, IEEE Computer Society membership is definitely for you!

With benefits that include:

- Access to 600 online books from top publishers, such as O'Reilly Media.
- Access to 3,500 technical and business courses.
- Access to conferences, publications, and certification credentials at exclusive member-only savings.

Discover even more benefits and become an IEEE Computer Society Member today at

www.computer.org