Guest-editorial

Software systems for scalable computers

David R. O'Hallaron [a] and
Boleslaw K. Szymanski [b]
[a] *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
E-mail: droh@cs.cmu.edu
[b] *Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180, USA*
E-mail: szymansk@cs.rpi.edu

After a long period of intense research conducted mainly in academia and supported by government agencies, parallel processing has moved to industry and is deriving the bulk of its support from the commercial world. Such a move has brought with it a change of emphasis from record-breaking performance to price-performance ratios and sustained speed of application program execution. The current parallel architectures are fast and economically sound. As a result, there is a strong trend towards widening the application base of parallel processing both in terms of hardware and software. At the same time, quickly developing technology fundamentally changes balances between cost of computing, communication and programming, and new balances often lead to new approaches.

On the hardware side, the prevailing tendency is to use off-the-shelf commercially available components (processors and interconnection switches) that benefit from the rapid pace of technological advancement fueled by Moore's Law. The other tendency is the convergence of different architectural approaches as the successful ones spread to new systems. Workstations interconnected by fast networks approach the performance of special-purpose parallel machines. Shared memory machines with multilevel caches and sophisticated prefetching strategies execute programs with efficiency similar to distributed memory machines. At the hardware level, the essential aspect of a quickly changing landscape is the difference in growth of network bandwidth, processor speed and memory access times, which are listed in the descending order of their speed of improvement.

All-optical networks and interconnects are changing the balance on the networking side. Throughput now tends to be limited by processor speed and software overheads rather than by network bandwidth, as was the case in the past. On the one hand, the latency of networks is fundamentally limited by the speed of light and the distance that the transferred data need to travel. In addition, processor speed is growing faster than memory access time, where the technological advances are used to increase the memory chip capacity rather than its speed. The resulting use of buffering to mask the speed differences has led to the multi-memory hierarchy in which registers, primary cache, secondary cache and main memory are typical layers with progressively lower speed but larger capacity.

One result of these trends is the growing importance of data locality for the performance of computer systems, where the architectural details dictate the structure of the most efficient object code. However, improvements in hardware speed are happening faster than advances in programming efficiency, causing programmer time to become more expensive relative to the cost of hardware. To expect a programmer to find the optimum run-time structure would be contrary to this trend. As a result, the programming trends are towards portability and reuse of software, which require abstraction from the architectural details of the computer. Hence, compilers and run-time systems must be responsible for tuning the portable software for a particular architecture, and we find that research on automatic optimization of data locality both at compile and run time has been growing in importance.

The increasing complexity of interactions between processor, memory hierarchy and network in parallel systems must be encapsulated in a proper abstract model capable of providing a universal representation of parallel algorithms. The widening base of the users relies on standardization of parallel programming tools. Parallel programmers face a daunting challenge, especially with increasingly large and complex appli-

ISSN 1058-9244 / $8.00 © 1999, IOS Press. All rights reserved

cations. They must identify parallelism in an application, extract and translate that parallelism into their codes, design and implement communication and synchronization that preserve the program semantics and foster the efficiency of parallel execution. All these steps must be guided by the currently available architectures which are constantly changing, potentially making some parts of the software design suboptimal or inefficient. Not surprisingly, in such an environment parallel programming has experienced a long and difficult maturation process. By protecting the programmer's investment in software, standardization promotes development of libraries, tools and application kits that in turn attract more end-users to parallel processing. It appears that parallel programming is ending a long period of craft design and is entering a stage of industrial development of parallel software.

One of the main lessons from the last 20 years of research in parallel computing is the importance of addressing problems at all levels of a computer system, including low-level architecture, compilers, run-time systems, languages, and applications. The papers included in this issue address problems at each of these levels. They were selected from 47 submissions to the International Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers (LCR98), which was held in May 1998 at Carnegie Mellon University. A total of 23 papers were presented at the workshop, and based on the reviews, we selected 9 of them for invitation to this special issue. The authors of the invited papers submitted extended and updated versions of their presentations that underwent a new round of reviews. The final versions of the papers, with changes suggested by the new reviews, are presented in this issue.

The first two papers address low-level system issues. The first paper, entitled "Impulse: memory system support for scientific applications", by John Carter, Wilson Hsieh, Leigh Stoller, Mark Swanson, Lixin Zhang, and Sally McKee, describes a radical new memory controller for improving the performance of programs with irregular memory access patterns that are known only at run-time. Such programs are the core of the large scientific simulations that motivate much of parallel computing. The Impulse controller is an exciting new idea because it could be incorporated easily into existing system designs. The second paper, entitled "The statistical properties of host load", by Peter Dinda, investigates statistical properties of host load through traces collected at different machines. These properties are of utmost importance for building scalable parallel distributed computations on clusters of workstations. Dinda's results are surprising because they identify unexpected structure in load traces, including the property of self-similarity.

Compilers have long been recognized as essential tools for programming parallel computers. However, the problem of parallelizing sparse and irregular codes has proven to be difficult. The paper entitled "On the automatic parallelization of sparse and irregular Fortran programs", by Yuan Lin and David Padua, deals with the tough problem of parallelizing sparse and irregular Fortran codes at compile-time. The authors identify and classify relevant kernel codes, and introduce a promising new approach.

An exciting new research direction in parallel computing is the discovery of ways that compilers and run-time systems can cooperate to produce more efficient parallel programs. In "Combining compile-time and run-time parallelization", by Sungdo Moon, Byoungro So, and Mary Hall, the authors describe how efficient run-time tests, supported by high-quality compile-time analysis, can identify new opportunities for loop-level parallelism. The results are integrated into a real compiler/run-time system and validated on programs from the SPECFP95 and NAS benchmark suites.

Other researchers are investigating this compile-time and run-time integration in the context of distributed workstation clusters. Traditional distributed computing research focuses on resource availability, result correctness, code portability and transparency of access to the resources, rather than the issues of speed, efficiency, and scalability that are central to parallel computing. The ever-decreasing cost of hardware encourages configuring computer systems for a peak demand that is much higher than the average demand. So generally, computers are underutilized. Hence, there are large computational resources available at any moment over the LAN (Local Area Network), WAN (Wide Area Network) and the Internet. Distributed and parallel systems that are capable of exploring such resources are of growing interest. However, challenges to build them for truly universal use are formidable, among them security of the accessed machines, fault tolerance, transparency, and the system's ability to adapt to the changing availability of computers,

The paper entitled "CRAUL: Compiler and run-time integration for adaptation under load", by Sotiris Ioannidis, Umit Rencuzogullari, Robert Stets, and Sandhya Dwarkadas, shows how the compiler and run-time system can work together to load-balance applications running on distributed shared memory workstation clusters. Another paper, entitled "Flexible IDL compilation for complex communication patterns", by Eric Eide, James Simister, Tim Stack, and Jay Lepreau describes how to build a flexible optimizing compiler that produces specialized and efficient communication code for the run-time systems of complex distributed applications.

The last three papers address issues at the application and language levels. The idea of using both data and task parallelism in a single application is not new. However, dynamic integration of these two types of parallelism is very useful for real-time interactive applications where the response time is at premium. The paper entitled "Integrated task and data parallel support for dynamic applications", by James Rehg, Kathleen Knobe, Umakishore Ramachandran, Rishiyur Nikhil, and Arun Chauhan describes a new architecture for parallelizing real-time applications using such an integration, and show how this architecture can be applied to complex and dynamic multimedia applications such as a smart kiosk. In the paper entitled "Menhir: An environment for high performance Matlab", Stéphane Chauveau and François Bodin describe a compiler system for parallelizing programs written in Matlab, a popular high-level language for specifying numerical algorithms. Finally, in "Irregular computations in Fortran – expression and implementation strategies", Jan Prins, Siddhartha Chatterjee, and Martin Simons describe some elegant techniques for representing and implementing sparse irregular codes in Fortran.

In preparation of this special issue, the reviewing of the papers was an important stage both for the selection of the best submissions as well as for improving the published papers. Many thanks are due to the following volunteers for their timely and thoughtful reviews: David Bakken, Sigfried Benkner, Christopher Carothers, Peter Dinda, Ian Foster, Thomas Gross, Chen Ding, Chuanyi Ji, Guoha Jin, Charles Koelbel, Erwin Laure, William Maniatty, Pantona Mario, Eduard Mehofer, Piyush Mehrotra, Charles Norton, Joel Saltz, Jaspal Subhlok, Bernd Wender, Bwolen Yang, Katherine Yelick, Mohammed Zaki, and Hans Zima.