Parallel computation still not ready for the mainstream.

by Domenico Talia

Parallel computation has helped solve problems in applied science and engineering such as high-performance applications. However, it is still not used in all areas of computer science because of a lack of general-purpose parallel computing models. Parallel computing systems include single instruction multiple-data (SIMD) machines and multiple-instruction-multiple data computers.

For the past 20 years, parallel computation has helped solve many significant problems in applied science and engineering, most notably those in high-performance applications not implementable on sequential computers [1]. However, parallel computing is still not used in all areas of computer science nor has it found a significant role in mainstream computing. Even if parallelism meets its technical goals, achieving the status of a mature technology in the next years remains a serious challenge.

A major factor limiting parallel computation in mainstream computer science is the lack of general-purpose parallel computing models. Parallel computing is a class of systems including many different architectures--from single-instruction-multiple-data (SIMD) machines to distributed-memory, multiple-instrucion-multiple-data (MIMD) computers and workstation clusters. Therefore, it is difficult to define a unifying architectural model for parallel computing, in the same way the von Neumann model is the unifying model for sequential computing. Moreover, some specialists who believe finding a unifying model is just not possible have gone in another direction, developing parallel software that lacks portability.

On the software side, the architecture differences in parallel computers correspond to a large set of different parallel models and languages often architecture-dependent and that offer only partial solutions to programming portable parallel applications in sequential computing using standard languages, like C, Pascal, and Fortran. Many parallel-programming languages used today are of the low-level variety that require the programmer to face the architectural issues of the parallel machine on which the application runs.

On the other hand, high-level parallel languages abstract from architectural issues but deliver unpredictable performance on different architectures. Thus, porting the same program to different parallel computers from, say, a message-passing multicomputer to a shared-memory multiprocessor can dramatically alter the machine's performance.

A Realistic Strategy

Finding solutions to these problems and limitations in parallel computation requires two actions:

* Hardware. Make the design and implementation of general-purpose parallel computers [2] capable of supporting a wide range of programming models and providing predictable performance.

* Software. Make the definition of programming models architecture-independent, allowing abstraction and portability across different parallel computers. At the same time, make these models simple and expressive.

In the 1980s and 1990s, several general-purpose MIMD computers were developed using microprocessors connected through a communication network or shared memory space. These systems, including the Intel iPSC, the Sequent Symmetry, and the Transputer-based multicomputers, were used in high-performance applications. Although they do not completely spare programmers from the architectural issues, they showed a practical way to achieve high-level, general-purpose, parallel computation. Today, such systems point the way for development of new high-performance parallel computers consisting of large numbers of general-purpose microprocessors.

An important step to success is the definition of high-level, architecture-independent languages to demonstrate that parallel programming is no more difficult than sequential programming.

Low-level approaches, such as the Parallel Virtual Machine (PVM) and Message Passing Interface (MPI), are driven by heterogeneous parallel computing, which tries to offer, on different computers, library primitives for parallelism and communication. These approaches partly meet the portability goal but are based on tedious low-level library functions and do not free the programmer from the issues of concurrency, communication, and synchronization. In fact, even though PVM and the MPI [3] are de facto standards in parallel programming, their related programming style looks in many respects like assembler-level programming of sequential computers.

However, several proposed high-level approaches--the Bulk Synchronous Parallel (BSP) [4], the LogP [5], and the

Parallel computation still not ready for the mainstream.

Bird-Meertens Formalism [6]--may represent good candidates for architecture-independent programming models on general-purpose computers.

Other promising models are the skeleton-based [7] and the actor-based [8] languages. Although these models suffer from low performance, they represent an interesting starting point toward architecture-independence because they abstract from architectural issues and allow predictable performance. If the parallel computing community convinces itself that it needs a clear strategy based on high-level languages to find a unifying model for parallel computation, these models can be used to drive this process.

Adopting this strategy would unite high-level programming, generality, and high performance, leading parallel computation to the computing mainstream.

Conclusion

The issues addressed here are not so simple. However, the general-purpose approach I've outlined might represent a significant step toward a very large use of parallel computing in many application areas for developing portable and standard parallel software. Enlarging the parallel-computing community to the majority of computer science users in both research and industry and making parallel computers the standard computing platforms of the next century may also be another step in the right direction.

REFERENCES

[1.] Skillicorn, D B. and Talia, D., eds. Programming Languages for Parallel Processing. IEEE Computer Society Press, 1994.

[2.] May, D. Towards general-purpose parallel computers. In Proceedings of CRAI Spring International Seminar on Highly Parallel Processing, (Capri, Italy), May 1990.

[3.] Dongarra, J.J., Otto, S.W. Snir, M. and Walker, D. A message passing standard for MPP and workstations. Commun. ACM 39, 7, (Jul. 1996), 84-90.

[4.] Valiant, L G. A bridging model for parallel computation Commun. ACM 33, 8 (Aug. 1990, 103-111.

[5.] Culler, D. et al. LogP: A practical model of parallel computation. Commun. ACM 39, 11 (Nov. 1996)

[6.] Skillicorn, D. B. Architecture-independent parallel computation. IEEE Comput. 23, 12 (Dec. 1990), 38-51.

[7.] Cole, M. Algorithmic Skeletons: Structured management of parallel computation. In Research Monographs in Parallel and Distributed Computing. Pitman, London, 1989.

[8.] Agha, G. and Callsen, C.J. ActorSpace: An open distributed programming paradigm. In Proceedings of Symposium on POPP, (San Diego, Calif.), ACM Sigplan Not, Jul. 1993, 23-32.

DOMENICO TALIA (talia@si.deis.unical.it) is a senior researcher at the Institute of System Analisys and Information Technology, Rende, Italy.