The practical application of parallel processing, fueled by the integration of ever-faster processors into ever-faster parallel systems, is changing the computer industry. Advances in hardware integration and packaging have brought to the market machines classified as workstations (Hewlett-Packard/Apollo’s Prism and Ardent’s Titan), mainframes (Sequent’s Symmetry and Encore’s Multimax), and even specialized, distributed systems (BBN’s Butterfly and Intel’s iPSC).

The commercial availability of these machines poses a challenge to the software industry. What is the best way to harness this power? What tools do we need to program them effectively? Last
year, a special issue of *IEEE Software* examined languages and compilers for parallel-processing systems. This year, we focus on the practical results of such efforts: environments and languages for parallel systems to harness the hardware.

Parallel programmers want to know:
- How to express a parallel algorithm in a programming language.
- How to parallelize a program to work.
- How to tune the application to get the best speedup.

While these areas correspond to serial programs' development, debugging, and performance-tuning phases, a parallel programmer needs to know much more: Parallel algorithms, the languages they are written in, and the systems they run on are inherently more complex.

**Environments.** Visualization tools are the key to simplifying parallel programming. These visual aids let a programmer capture, debug, and tune his application for a particular parallel architecture. The three development environments described in this issue all use some visualization techniques.

- **CODE.** J.C. Browne and his colleagues have developed the Computation-Oriented Display Environment. CODE, which is being used in classes at the University of Texas at Austin, lets you develop modular parallel programs graphically in an environment built around fill-in templates. It also lets you incorporate programs written in any sequential language into parallel programs targeted for any parallel architecture.

- **Faust.** Vincent Guarino and his colleagues describe Faust, an integrated environment for the development of large, scientific applications. Developed at the Center for Supercomputing Research and Development, some of Faust's components will be distributed publicly this year. Faust includes a project-management tool, a context editor that is interfaced to a program database, and performance-evaluation tools.

- **Pat/Start.** Bill Applebe, Kevin Smith, and Charlie McDowell address the question of how to use existing sequential Fortran code on multiprocessors. Their answer is Start/Par, an interactive toolkit that automates the parallelization of sequential Fortran as it teaches the programmer how to exploit and understand parallel structures and architectures. The Start/Par prototype has been installed at several user sites.

**Languages.** The languages we use today were designed for sequential programming. Tomorrow's languages must do a better job of expressing parallelism. The three language articles in this issue reflect the two basic approaches to parallel-programming language development: extending sequential languages and designing new parallel languages.

- **Spur Lisp.** Ben Zorn and his colleagues at the University of California at Berkeley describe their multiprocessing extensions to Common Lisp. They have added a few simple, expressive features on which they could build high-level constructs. When Spur Lisp has been ported to and optimized on the Spur workstation, programmers will be able to use it to make symbolic programs parallel.

- **Qlisp.** Ron Goldman and Richard P. Gabriel describe another version of Common Lisp for multiprocessing. Qlisp supports medium-grained parallelism for artificial-intelligence and symbolic programs. Like Spur Lisp, Qlisp supports futures. It also introduces partially, multiply invoked functions that let you synchronize program components. The implementation of Qlisp they describe, done on an Alliant FX/8, is based on Lucid Common Lisp, a commercial system.

- **Consul.** Doug Baldwin describes a new constraint-based language, Consul, that can exploit implicit parallelism. While many believe logic languages like Prolog hold promise for implicit parallelism, Consul neither adopts a logic-based notation nor extends a logic language with constraint-satisfaction heuristics. In his report on the first phase of his project, Baldwin produces empirical evidence that Consul programs do exhibit considerable parallelism.

**Architectures.** Finally, Evan Tick compares two parallel logic-programming architectures, an On-parallel Prolog system and an And-parallel flat-guarded-Horn-clause system. Tick compares the performance of these two architectures.

**Acknowledgments**

I thank the more than 50 authors who submitted articles for this issue: Peter Harrison, who did a marvelous job attracting articles and reviewers in Europe; the reviewers, who did an excellent job refereeing the articles; and Ted Lewis, who gave me encouragement and support. Finally, I thank Jon Schenk for secretarial help and Sequent for its support.

**Shreeram Thakkar** is a staff systems engineer at Sequent Computer Systems. His research interests include highly parallel systems, computer architecture, parallel programming and algorithms, and VLSI design.

Thakkar received a BS with honors in statistics and computer science from the University of London and an MS and PhD in computer science from the University of Manchester. He is a chartered engineer and a member of the IEEE, ACM, BCS, and IEE.

Address questions about this issue to Thakkar at Sequent Computer Systems, 15450 SW Koll Pkwy., Beaverton, OR 97006; UUCP: net user@sequent.