Introduction

In recent years there has been an increase in the number of parallel computers shipped by computer vendors, which, to a large extent, can be attributed to the popularity of small-scale multiprocessor workstations and servers. Large-scale parallel computers have also gained wider acceptance in the commercial world. With the rapid advance of VLSI technologies, multiple processors on a single chip may soon be commercially available, which will give rise to even wider use of parallel computers.

Parallel computers not only can increase system throughput but also present opportunities to speed up individual programs. However, writing parallel programs is still much more difficult than writing sequential programs. It is equally challenging, if not more so, to develop tools which can automatically restructure sequential programs for parallel machines. Designers of parallel languages and compilers have long worked toward bringing parallel computing to mainstream. The increased availability of commercial parallel computers makes such effort even more urgent and challenging.

In August 7–9, 1997, over seventy researchers working in this field met at the Tenth Annual International Workshop on Languages and Compilers for Parallel Computing (LCPC'97) held at the University of Minnesota, to exchange ideas and present their state-of-the-art results. Based on the program committee's recommendations, we invited several research groups to present their revised papers as formal articles for a special issue in the *International Journal of Parallel Programming*. The articles presented in this special issue on Languages and Compilers for Parallel Computing represent an important part of the current research efforts in the field.

This special issue is published in two parts. The first part contains four articles. The first article, "Parallel Programming and Performance Evaluation with the URSA Tool Family," by Park, Voss, Armstrong, and Eigenmann, deals with compilers which automatically parallelize sequential programs. It addresses the question of how to make such parallelizing compilers more interactive. The other three articles in this first part are on the analysis and performance enhancement of parallel programs. The article by Lee, Midkiff, and Padua, entitled "A Constant Propagation Algorithm for Explicitly Parallel Programs," presents a new intermediate representation (IR) of programs written in parallel forms by programmers. Based on the new IR, they present a constant propagation algorithm which takes interaction between parallel threads into consideration. The article by Han, Tseng, and Keleher, entitled "Reducing Barrier Synchronization for Compiler-Parallelized Codes on Software DSMs," targets a software distributed-shared-memory (DSM) system built on top of an IBM SP2, a message-passing, distributed-memory parallel computer. The article by Mellor-Crummey and Adve, entitled "Simplifying Control Flow in Compiler-Generated Parallel Code," presents algorithms, which have been implemented in the Rice dHPF compiler, to remove redundant conditionals often seen in the code generated by the HPF compilers. Their experiments are performed on an IBM SP2.

The second part of this special issue consists of three articles. The first two articles concern an important compiler transformation called *tiling*. The article by Mitchell, Högstedt, Carter, and Ferrante, titled "Quantifying the Multi-Level Nature of Tiling Interactions," illustrates the difficulty in tiling due to multi-level parallelism and multi-level memory systems in some new computer architectures. They propose to use multi-level cost functions to guide tiling in order to overcome such difficulty. The second article, "Reuse-Driven Tiling for Improving Data Locality," by Xue and Huang, presents a solution to maximize the amount of data reuse in the cache memory while minimizing the number of tiled loops (thus reducing the extra control overhead associated with the tiled loops). The last article, entitled "Compiler Techniques for the Superthreaded Architectures," by Tsai, Jiang, and Yew, looks beyond current parallel architectures and considers a new form of parallel execution called *superthreading* which provides some hardware support for speculative parallel execution at the thread level. They explore compiler techniques for such a processor architecture and present simulation results.

We thank the LCPC'97 committee for their help in the reviewing process and their recommendation of these articles. We also thank the external reviewers for their valuable comments and suggestions.

The last article, "Compiler Techniques for the Superthreaded Architectures," by Tsai, Jiang, and Yew was originally scheduled to appear in *International Journal of Parallel Programming*, Volume 26, Number 6, December 1998, but has been moved to Volume 27, Number 1, February 1999.

Zhiyuan Li[1] and Pen-Chung Yew[2]
Guest Editors

[1] Department of Computer Sciences, Purdue University.
[2] Departement of Computer Science and Engineering, University of Minnesota.