Introduction

Recent years have seen a greatly increased 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 feasible, which will give rise to yet more parallel computers.

Parallel computers not only can increase system input but also present opportunities to speed up individual programs. However, it is commonly known that writing parallel programs is much more difficult than writing sequential ones. It is equally challenging, if not more so, to design tools which can automatically transform sequential programs into parallel machine code. Designers of parallel languages and compilers have long worked on meeting the challenge of bringing parallel computing to the main stream. The increased availability of commercial parallel computers only makes the old challenge more imminent and practical. Many researchers have made serious research efforts to meet the challenge. Last August, over seventy people working in this field met at the Tenth Annual International Workshop on Languages and Compilers for Parallel Computing (LCPC'97) to exchange ideas and the state of the art. Based on the program committee's recommendations, we invited several research groups to revise their presentations as formal articles for a special issue of 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 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, concerns 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 deal with analysis and improvement of parallel programs. The article by Lee, Midkiff and Padua, titled "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 correctly deals with interaction between parallel threads. The article by Han, Tseng and Keleher, titled "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-Crummy and Adve, titled "Simplifying Control Flow in Compiler-Generated Parallel Code". presents algorithms implemented in the Rice dHPF compiler which remove redundant conditionals often seen in code generated by 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 new difficulty for tiling due to the multi-level parallelism and multi-level memory in 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 the problem of maximizing the amount of data reuse in the cache while minimizing the number of tiled loops (thus reducing the extra control cost associated with loops). The last article, titled "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 *superthread* which provides 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 initial review and recommendation of the above articles. We also thank the external reviewers for their valuable comments and suggestions.

Zhiyuan Li
Department of Computer Sciences
Purdue University

Pen-Chung Yew
Department of Computer Science and Engineering
University of Minnesota