International Journal of Parallel Programming, Vol. 28, No. 5, 2000

Introduction

This is the second of two issues devoted to the Dagstuhl-Seminar on *Instruction-Level Parallelism and Parallelizing Compilation*, held at Schloss Dagstuhl on April 18–23, 1999. See the Introduction of the first issue for a short description of the seminar. This issue contains work on parallelizing compilation, specifically on the following topics.

Dependence Analysis

A central issue of parallelizing compilation is the dependence analysis of programs and their optimization for parallel or ILP processors. The paper by Amme, Braun, Zehendner, and Thomasset of the University of Jena applies techniques of symbolic evaluation for memory reference disambiguation on assembly language code with the aim of increasing ILP.

Loop Parallelization

Programs consisting of many nested loops are ideal for parallelizing compilers. To model the task of executing a nest of n loops in parallel, the compiler views the set of loop iterations as a set of points in an n-dimensional integer grid. The geometric and algebraic structure of this point set can be analyzed and exploited to create an efficient execution strategy. Quilleré and Rajopadhye from IRISA, Rennes, consider a loop nest whose iteration domain is a union of polytopes. The problem is to scan the points of this union efficiently. Instead of testing at runtime whether an iteration point falls in the domain or lies outside—which can be quite expensive—they split the domain into pure polyhedra and scan these without any runtime tests.

Software Pipelining

Darte and Huard of the ENS Lyon propose an improvement to the shift and compact method for software pipelining. In this technique, the

0885-7458/00/1000-0429$18.00/0

loop body is compacted by a list scheduler after first shifting operations between iterations; the length of the compacted schedule becomes the initiation interval of the software pipeline. They find a fast algorithm for shifting operations that minimizes both the critical path and the number of dependence edges that are not loop-carried, and give the results of an experiment that shows this to be an efficient and effective shifting strategy.

CONCLUSIONS

The seminar brought to light several of the exciting developments taking place in parallel computer architecture. It also made clear the heavy burdens that are being placed on compilers by current parallel machines. The efficient use of performance-enhancing hardware such as cache hierarchies, pipelined functional units, and predicated execution calls for highly sophisticated analysis and code generation techniques. It remains to be seen how the portability of parallel software can be maintained in this scenario. Portability is essential. After all, a parallel computer whose main purpose is high performance inevitably becomes obsolete after about five years. Sadly, in the past, the sinking computer has often taken its software down with it.

D. K. Arvind (Edinburgh)
K. Ebcioglu (Yorktown Heights)
C. Lengauer (Passau)
R. Schreiber (Palo Alto)
Guest Editors