Topic 04
Compilers for High Performance Systems

Barbara Chapman

Global Chair

1 Introduction

This workshop is devoted to research on technologies for compiling programs to execute on high performance systems. Contributions were sought for all related areas, including parallelization of imperative languages, as well as translation of object-oriented languages and several other paradigms for parallel systems. Compiler analysis and transformation increasingly makes use of run-time information; the interaction of compilers with development and execution environments is thus explicitly included in the scope of this workshop.

We received a very healthy crop of submissions to this area, and their topics reflected the variety of current compiler research in this field. Both shared memory and distributed memory parallelism was targeted. The accepted papers have been organised into three sessions whose contents are outlined in the following.

2 The Papers

Array privatisation is one of the traditional techniques employed by parallelizing compilers in order to reduce the number of references to shared data. When applicable, it creates a local copy of the data structure in question for each processor, and if necessary, inserts additional code to resolve control flow. Several researchers have sought to lower the memory overhead implied by this technique. The first paper in this workshop, by Cohen and Lefebvre, presents just such an approach, which generalises the technique of partial array expansion to handle arbitrary loop nests. Some compilers convert programs into Static Single Assignment form, in order to eliminate all but true dependences. This technique was recently extended to handle arrays on a per-element basis, enabling its use to perform constant propagation for array elements, and more. Collard extends this technology in his contribution to the workshop, in which he develops an array SSA framework which can be applied to explicitly parallel programs containing parallel sections, and a range of synchronisation mechanisms. Another existing analysis framework is extended for use in conjunction with explicitly parallel programs, in the last paper of this session. Here, Knoop introduces demand-driven data flow analysis, in which one searches for the weakest preconditions which must hold if a specific data flow factum is to be guaranteed. He shows how this may be performed for sequential programs, and then proceeds to develop an approach to deal with shared memory parallel programs.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 373–374, 1999.

The second session begins with two publications related to systolic computing, and to the design of systolic arrays. The first of these, by Fimmel and Merker, considers the problem of minimising the cost of implementing communication channels in silicon, where bandwidth and communication delay may be varied. Their work also applies to the problem of obtaining locality for data dependences on a given interconnection topology. Crop and Wilde also contribute to techniques for the design of systolic arrays in their paper on the scheduling of affine recurrence equations. They use scheduling to add temporal information to the system, which helps determine the placement of pipeline registers, and more. Dependence analysis must accordingly be extended to derive constraints on the timing functions. Laure and colleagues describe the compilation of Opus, a language which permits the expression of coarse grained HPF tasks and their coordination. The role of HPF compilers and the run time system in this approach are discussed. Next, Leair and colleagues describe their implementation of a generalised block distribution in a commercial HPF compiler. The impact of such distributions, where the owner of a specific datum cannot be computed, on compiler optimisation is discussed. Performance comparisons of program versions using standard and general block distributions, respectively, indicate the usefulness of the latter.

There are many applications whose parallelised version requires communication mainly to perform one or more global reduction operations. In the last session of this workshop, Gutierrez et al. present a new method for parallelising irregular reduction operations to run on a shared memory multiprocessor. As with the inspector-executor approach employed for distributed memory systems, their strategy distributes data and assigns the computation in a manner which enforces the locality of writes to the reduction array. Yet it achieves superior performance by reordering accesses to the subscripted arrays. Kandemir and colleagues discuss the impact of I/O on techniques for loop tiling applied to out of core computations. They show that traditional methods which optimise the usage of cache may not be suitable when I/O is performed within the loop, and propose an algorithm which considers both the I/O and the computation within loop nests to obtain an overall tiling strategy. Shafer and Ghose examine the impact of message reordering and task duplication optimisations on a scheduled set of tasks, and propose a strategy to determine whether to merge messages between pairs of tasks. Although it appears that the average payoff is not large, the authors argue that the techniques are worthwhile because they lead to some improvement in a large fraction of cases tested. In the final paper of this workshop, Pfannenstiel proposes an implementation technique for nested parallel programs which combines piecewise execution based on cost-effective multithreading with fusion optimisation. The strategy supports execution of programs with large data sets. Experiments show the importance of the optimisations.