

Workshop 04

Automatic Parallelization and High-Performance Compilers

Jean-François Collard

Co-chairmen

Presentation

This workshop deals with all topics concerning automatic parallelization techniques and the construction of parallel programs using high performance compilers. Topics of interest include the traditional fields of compiler technology, but also the interplay between compiler technology and communication libraries or run-time support. Of the 16 papers submitted to this workshop, 7 were accepted as regular papers, 2 as short papers, and 7 were rejected.

Organization

The first session focuses on data placement and data access. In “Data Distribution at Run-Time: Re-Using Execution Plans,” Beckmann and Kelly show how data placement optimization techniques can be made efficiently available in run-time systems by a mixed compile- and run-time technique. On the contrary, the approach by Kandemir *et al.* in “Enhancing Spatial Locality using Data Layout Optimizations” to improve cache performance in uni- and multi-processor systems is purely static. They propose an array restructuring framework based on a combination of hyper-plane theory and reuse vectors. However, when data structures are very irregular, such as meshes, the compiler alone can in general extract very little information. In “Parallelization of Unstructured Mesh Computations Using Data Structure Formalization,” Koppler introduces a small description language for mesh structures which allows him to propose a special-purpose parallelizer for the class of applications he tackles. It is worth noticing the wide spectrum of techniques, ranging from completely static methods to purely run-time ones, explored in this field. This definitely illustrates the difficulty of the problem, and the three papers mentioned above make significant contributions asserted by real-life case studies.

The second session starts with “Parallel Constant Propagation,” where Knoop presents an extension to parallel programs of a classical optimization of sequential programs: constant propagation. Another classical sequential optimization, extended to parallel programs, is redundancy elimination [2]. It is well known, however, that redundancies can be an asset in the parallel setting. Eisenbiegler takes benefit of this property in his paper “Optimization of SIMD Programs

with Redundant Computations,” and reports very encouraging execution time improvements. Finally, in “Exploiting Coarse Grain Parallelism from FORTRAN by Mapping it to IF1,” Lachanas and Evripidou describe the parallelization of Fortran programs through conversion to single assignment. This work is also interesting for its smart use of two separately available tools: the front-end of Parafrase 2 and the back-end of the SISAL compiler.

In the third session, Feautrier presents in “A Parallelization Framework for Recursive Tree Programs” a novel framework to analyze dependencies in programs with recursive data. It is most noteworthy that a topically related paper has recently been published elsewhere [3], illustrating that the analysis of programs with recursive structures currently is a matter of great interest. How to extend the static scheduling techniques crafted by the author [1] to this framework is an exciting issue. Like scheduling, tiling is a well-known technique to express the parallelism in programs at compile-time. In their paper “Optimal Orthogonal Tiling,” Andonov, Rajopadhye and Yanev bring a new analytical solution to the problem of determining the tile size that minimizes the total execution time.

A mixed compile- and run-time technique is addressed in the last paper. In “Enhancing the Performance of Autoscheduling in Distributed Shared Memory Multiprocessors,” Nikolopoulos, Polychronopoulos and Papatheodoro present a technique to enhance the performance of autoscheduling, a parallel program compilation and execution model that combines automatic extraction of parallelism, dynamic scheduling of parallel tasks, and dynamic program adaptability to the machine resources.

Conclusion

When trying to gain retrospect on the papers presented in this workshop, one may notice that the borderline between compile-time and run-time is getting blurred in data dependence techniques, data placement, and the exploitation of parallelism. In other words, intensive research is being conducted to benefit from the best of both worlds so as to cope with real-size applications. This orientation is very encouraging and we can hope the end-user will soon benefit from the nice work proposed by the papers of this workshop.

References

1. P. Feautrier. Some efficient solutions to the affine scheduling problem, part I, one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, October 1992.
2. J. Knoop, O. Rüthing, and B. Steffen. Optimal code motion: Theory and practice. *ACM Transactions on Programming Languages and Systems, TOPLAS*, 16:1117–1155, 1994.
3. Y. A. Liu. Dependence analysis for recursive data. In *Int. Conf. on Computer Languages*, pages 206–215, Chicago, Illinois, May 1998. IEEE.