Topic 10

Programming Languages, Models, and Methods

Paul H.J. Kelly, Sergei Gorlatch, Scott Baden, and Vladimir Getov

Topic Chairmen

1 The Field

This topic provides a forum for the presentation of the latest research results and practical experience in parallel programming. Advances in algorithmic and programming models, design methods, languages, and interfaces are needed for construction of correct, portable parallel software with predictable performance on different parallel and distributed architectures.

Our topic emphasises results which improve the process of developing high-performance programs. Of particular interest these days are novel techniques by which parallel software can be assembled from reusable parallel components without compromising efficiency. Related to this is the need for parallel software to adapt, both to available resources and to the problem being solved.

2 The Common Agenda

The discipline of parallel programming is characterised by its breadth — there is a strong tradition of work which combines

- Programming languages, their compilers and run-time systems
- Formal methods for program specification, derivation and verification
- Architectural issues – both influencing parallel programming, and influenced by ideas from the area – including cost/performance modeling
- The wider techniques for parallelism originated software engineering

This research area has benefited particularly strongly from theoretical ideas. There is a very fruitful tension between, on the one hand, a *reductive approach*: develop tools to deal with program structures and behaviours as they arise, and on the other, a *constructive approach*: design software and hardware in such a way that the optimisation problems which arise are structured, and presumably, therefore, more tractable. To find the right balance, we need to develop theories, languages, cost models, and compilers - and we need to learn from practical experience building high-performance software on real computers.

It is interesting to reflect on the papers presented here, and observe that despite their diversity, this agenda really does underly them all.

3 The Selection Process

We would like to extend our thanks to the authors of the 29 submitted papers, and to the 82 referees who kindly and diligently participated in the selection process.

Eleven papers are presented in full-length form, reflecting a remarkably strong field. Our decision was normally supported by four referees; in three cases only two or three of the four or five referees supported the paper, but, after extensive and enjoyable discussion, the programme committee decided to accept them in order to bring attention to interesting new work.

One of the strengths of Euro-Par is the tradition of accepting new and less mature work in the form of short papers. We were very pleased to select 9 submissions in this category. Brevity is a virtue, and several short papers propose interesting new approaches which we hope to see developed further in time for next year's conference.

4 The Papers

The 20 accepted papers have been assigned to five sessions based on their subject area:

- **Compilation and Performance Issues**

This session begins with a distinguished paper by Theobald, Kumar, Agrawal, Heber, Thulasiram and Gao, reporting on their implementation of sparse matrix-vector multiply on a simulation of EARTH, their novel multithreaded architecture. The application is a critical kernel for many important applications, and a key motivation behind the design of multithreaded machines. The theme - understanding actual performance issues and the consequences for programming - is continued with work on the predictive value of the BSP cost model, the performance benefit of programming a scalable shared-memory machine using distributed-memory techniques, and the influence of language on the compiler's ability to optimise array algorithms.

- **Structured Parallel Programming**

This session begins with Zavanella's paper on his optimising compiler for a "skeleton" language which combines data- and task-parallelism. The key idea here is to exploit program structure so that a simple BSP cost model is applicable, and to use this in automatic performance optimisation - thus "tuning" the program for different target machines. The remainder of the session examines this "skeleton" approach from different perspectives - semantics, skeletons and design patterns, skeletons for the computational "grid", and evaluating the performance impact of foreknowledge of the communication pattern - structured parallel programming should lead to "oblivious" BSP programs.

– **Distributed Applications and Java**

Our third session is devoted to programming distributed applications efficiently and to the particular role of the Java language. R. van Nieuwpoort, Kielmann and Bal describe their distributed-memory implementation of divide-and-conquer, using task stealing and serialisation on demand. The session continues with work on structuring concurrency control in multi-threaded programs, and using Java to encapsulate conventional high-performance applications in order to use the language's powerful techniques for coordinating networked computations.

– **Efficient Implementation Techniques**

This session starts with the paper by Nieplocha, Ju and Straatsma, which describe their implementation of a remote memory access library for a distributed-memory machine with SMP nodes. The next paper concerns the trade-off between concurrency (and potential parallelism) and synchronisation overheads, with the salutary conclusion that cooperative multitasking can be better even on a parallel machine. The last paper of the session asks how well performance portability is achieved in a parallel functional language implementation.

– **Novel Parallel Languages and Formalisms**

Our final session exhibits several current trends in designing and implementing new languages for parallel programming. It starts with the presentation by Costa, Rocha and Silva, an extended analysis of a tricky implementation problem in parallel logic programming: how to manage environments which are shared and updated by different processes. The remainder of the session deals with a variety of novel parallel languages based mostly on functional formalisms.

The common ground shared by the 20 papers presented here lies in understanding the goals and problems in parallel programming models and languages. What is also very striking is the diversity of approaches being taken!