# Topic 10
# Parallel Programming: Models, Methods and Programming Languages

Scott B. Baden, Paul H. J. Kelly, Sergei Gorlatch, and Calvin Lin

Topic Chairpersons

## The Field

This topic provides a forum for the presentation of the latest research results and practical experience in parallel programming. Advances in programming models, design methodologies, languages, interfaces, run-time libraries, implementation techniques, and performance models are needed for construction of correct, parallel software with portable performance on different parallel and distributed architectures.

The topic emphasises results which improve the process of developing high-performance programs. Of particular interest are novel techniques for assembling applications from reusable parallel components without compromising efficiency on heterogeneous hardware, and applications that employ such techniques. Related is the need for parallel software to adapt, both to available resources and to the problem being solved.

## The Common Agenda

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

- Programming languages, their compilers and run-time systems
- Performance models and their integration into the design of efficient parallel algorithms and programs
- Architectural issues – both influencing parallel programming, and influenced by ideas from the area – including cost/performance modeling
- Software engineering for parallel and distributed systems

This research area has benefited particularly strongly from experience with applications. 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.

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 491–493, 2001.

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

## The Selection Process

We would like to extend our thanks to the authors of the 19 submitted papers, and to the 50 external referees who kindly and diligently participated in the selection process.

Six papers are presented in full-length form. 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 two submissions in this category. Brevity is a virtue, and the short papers propose interesting new approaches which we hope to see developed further in time for next year's conference.

## The Papers

The 8 accepted papers have been assigned to three sessions based on their subject area.

*Session 1: Thread-based models and concurrency*

- "Accordion Clocks: Logical Clocks for Data Race Detection"
  Christiaens and De Bosschere introduce Accordion Clocks, a technique for managing the space occupied by vector clocks needed for race detection in multithreaded applications. Their results dramatically improve the practical usability of race detection in multithreaded Java applications.
- "Partial Evaluation of Concurrent Programs"
  Martel and Gengler present a prototype implementation of their theoretical work specializing a concurrent program to a particular context by partial evaluation. They show how messages can be automatically eliminated by propagating compile-time constants across process boundaries.
- "A Transparent Operating System Infrastructure for Embedding Adaptability to Thread-Based Programming Models"
  Venetis, Nikolopoulos, and Papatheodorou describe transparent, non-intrusive services, e.g. OS-level, to permit multiple threaded jobs to run efficiently together on a single machine. Individual parallel jobs are adaptable – they are able to exploit an additional idle CPU and to maintain efficiency when a CPU reallocated to another job.

*Session 2: Parallel functional programming*

- "Nepal – Nested Data Parallelism in Haskell"
  Chakravarty, Keller, Lechtchinsky, and Pfannenstiel present an extension to the Haskell language for nested parallel arrays, and demonstrate its usefulness with two case studies.

– "Introduction of Static Load Balancing in Incremental Parallel Programming"
  Goodman and O'Donnell describe an incremental development methodology with formal reasoning techniques that supports the introduction of static load-balancing in a functional model of a SPMD system. The potential benefits of the approach are a safe development framework and possible performance improvements.

*Session 3: Cost models and their application in parallel programming*

– "A Component Framework for HPC Applications"
  Furmento, Mayer, McGough, Newhouse, and Darlington present a component framework intended to provide optimal performance for the assembled component application. The framework offers an XML itemize schema, a run-time representation in Java, and a strategy for selecting component implementations.

– "Towards Formally Refining BSP *Barrier*s into Explicit *Two-Sided* Communications"
  Stewart, Clint, Gabarró, and Serna describe a formal transformation scheme which translates BSP-style programs for execution on loosely coupled distributed systems employing asynchronous point-to-point communication.

– "Solving Bi-knapsack Problem Using Tiling Approach for Dynamic Programming"
  Sidi Boulenouar studies how to find the optimum tile size in solving the bi-knapsack problem, which interestingly has problem-dependent dependence distance vectors.

The common ground shared by the 8 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!