Topic 10
Parallel Programming: Models, Methods and Programming Languages

Kevin Hammond

Global Chair

1 Introduction

The greatest philosopher amongst us is as confined and hamstrung as the least significant thinker by the very language and notations in which his or her ideas can be expressed. By encapsulating complex concepts and ideas in simple words and phrases that we then reuse, we avoid the need to repeat the small and stumbling steps of our predecessors. So too is Computer Science advanced, allowing practitioners to benefit from the toil and wisdom of the pioneers through reusing models and abstraction.

This EuroPar workshop provides a forum for the presentation of the latest research results and practical experience in parallel programming models, methods and languages. 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.

2 The Research Papers

The 9 papers that have been selected for the workshop target various language paradigms and technologies: functional, object-oriented and skeletal approaches are all represented. A primary theme of the papers in this year's workshop is how technologies can cross over paradigm boundaries to find wider application. A second theme is exploitng abstraction mechanisms to reduce communication costs.

Two papers demonstrate cross-over from the functional community to conventional parallel systems. Firstly, Field, Kelly and Hansen show how the idea of *shared reduction variables* can be used to control synchronisation within SPMD programs. Shared variables can be introduced to eliminate explicit communications, thereby simplifying code structure. Furthermore, a *lazy evaluation* mechanism is used to fuse communications. The result is an improvement in performance over the original version due to the reduction in communication.

Secondly, Liniker, Beckman and Kelly propose to use *delayed evaluation*, to recapture context that has been lost through abstraction or compilation. In the initial stages of execution, evaluation is delayed and the system captures data flow information. When evaluation is subsequently forced through some demand, the data flow information can be used to construct optimised versions of the

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 603–604.

software components as appropriate to the calling context. The approach has been tested experimentally in the context of four simple scientific applications using the BLAS linear algebra library.

Skeleton approaches promise to dramatically increase programming abstraction by packaging common patterns of parallelism in high level routines. There has, however, been a historical lack of support for skeletons in conventional languages such as C or C++. Kuchen's paper introduces a library of basic skeletons for such languages that supports required skeleton functionality including polymorphism, higher-order functions and partial applications at minimal cost in efficiency. The library is built on MPI and therefore portable and efficient.

Having the *right* skeletons available when required is equally important for effective program construction. Bischof and Gorlatch introduce a new skeleton construct, the *double-scan* primitive, a combination of two conventional scan operations: one left scan with one right counterpart. The work is applied to existing software components whose purpose is to solve a system of linear equations. The paper demonstrates both predictability of performance and absolute performance that is comparable to a hand-coded version of the problem.

Recent developments in FPGA technology provide the potential for cheap large-scale hardware parallelism. The paper by Hawkins and Abdallah shows how this potential can be exploited by using a high-level functional language as a behavioural specification that can be systematically transformed into Handel-C and thus to FPGA circuitry. The work is applied to a real-world problem: a JPEG decompression algorithm.

At a more abstract level, Pedicini and Quaglia introduce a new system for distributed execution of λ-terms, PELCR. Their approach used *Directed Virtual Reduction*, a parallel graph-rewriting technique, enhanced with a priority mechanism. Speedup is demonstrated for a standard λ-calculus benchmark, DDA.

Scalability and predictability are key concerns. Work by Sobral and Proença studies scalability issues for object-oriented systems. Their objective is to ensure scalability dynamically by automatically increasing task granularity and reducing communication through runtime coalescing of messages. The work has been evaluated empirically on a number of platforms using a farm-type application.

Finally, exception handling and I/O mechanisms that have been designed for sequential languages and systems can present difficulties for concurrency. One particular problem arises in the context of explicit asynchronous method invocation, where the caller may no longer be in a position to handle remotely induced exceptions at the point they are raised. The paper by Keen and Olsson addresses this issue, introducing new language constructs for forwarding remotely induced exceptions to appropriate handlers. The mechanism has been implemented in JR, an extended Java aimed at tightly coupled concurrent systems. Bougeé, Danjean and Namyst meanwhile consider how to improve responsiveness to I/O events in multithreaded reactive systems, by introducing a synchronous *detection server* to provide a managed service to such events. This approach is demonstrably superior to standard approaches based on polling.