Workshop 6+16+18 Languages

Henk Sips, Antonio Corradi and Murray Cole

Co-chairmen

Three Workshops(6, 16, and 18) have programming languages and models as central theme. Workshop 6 focuses on the use of object oriented paradigms in parallel programming; Workshop 16 has the design of parallel languages as primary focus, and Workshop 18 deals with programming models and methods. Together they present a nice overview of current research in these areas.

Object Oriented Programming.

The Object-Oriented (OO) technology has received a renovated stimulus by the ever-increasing usage of the Web and Internet technology. The globalisation has enlarged the number of the potentially involved users to such an extent to suggest a reconsideration of the available environments and tools. This has also motivated the attempt to clarify all debatable and ambiguous points, not all of which of practical and immediate application. On the one hand, several issues connected to program correctness and semantics are still unclear. In particular, the introduction of concurrency and parallelism within an object framework is still subject to discussion, in its verification and modelling. The same is for the aggregation of different objects and behaviour in predetermined patterns. How to accommodate several execution capacities and resources into the same object or pattern requires still work and new proposals. In any case, there is much to be done, both in the abstract area and in the applied field. On the other hand, the distributed framework has not only introduced examples in need of practical solutions and environments, but also forced to reflect on all applicable models, starting from traditional ones, such as the client-server one and RPCs, to less traditional ones, such as the agent models. The growth of Java, CORBA, and their capacity of attracting implementors and resources produce unifying perspectives, with the possibility of offering a new integrated framework in which to solve most common problems. This starts to produce the possibility of creating generally available components to be really employed, by reducing the convenience of the redesign from scratch.

This conference session is an occasion to expose to an enlarged audience working into parallelism some of the hot topics and researches going on in the OO area. And, even if the OO community has many occasions of meeting and many forums to exchange opinions, EUROPAR seems a particular opportunity of both presenting experiences and receiving contributions with possibilities of cross-fertilisation. The five papers presented in the conference explore several of the most strategic directions of evolution of the OO area.

The first paper, “Dynamic Type Information in Process Types” by Puntigam, uses the process as an example of objects with dynamic type. The goal is to make possible all the checks typical of static types even in the dynamic case: the process is modelled as an active object that, depending on its state, is capable of accepting different messages from different clients. The presented model is a refinement of a previous work of the same author and is based on a calculus of objects that communicate with asynchronous message passing. The third paper of the session, by Gehrke, “An Algebraic Semantics for an Abstract Language with Intra-Object-Concurrency,” addresses the problems of intra-object concurrency, working with a process algebra method. The goal is to introduce the formal semantics for intra-object concurrency in OO frameworks where active processes can be distinguished from passive object. Let us recall that this is the Java assumption. The topics of the other papers are all connected to Java. The paper by Launay and Pazat, “Generation of distributed parallel Java programs” and the fifth paper, by Giavitto, De Vito and Sansonnett, “A Data Parallel Java Client-Server Architecture for Data Field Computations” addresses the point of enlarging the usage of the Java Framework. Launay and Pazat propose a framework capable of transparently distributing Java components of an application onto the available target architecture. Giavitto, De Vito and Sansonnett apply their effort to make Java usable in the data- parallel paradigm, for client-server applications. The fourth paper, by Lorcy and Plouzeau, “An object-oriented framework for managing the quality of service of distributed applications”, addresses the quality of service problem for interactive applications. The authors elaborate on the known concept of ‘contract’, with new considerations and insight.

Programming Languages.

As up to now data parallel languages have been the most succesful attempt to bring parallel programming closer to the application programmer. The most seriuos attempt has been the definition of High Performance Fortran (HPF) as an extension of Fortran 90. Several commercial compilers are available today. However, user experience indicates that for irregularly structured problems, the current definition is often inadequate. The first two papers in the HPF session deal with this problem. The paper by Brandes and Germain, called “A tracing protocol for optimizing data parallel irregular computations” describes a dynamic approach by allowing the user to specify which data has to be traced for modifications. The second paper by Brandes, Bregier, Counilh, and Roman proposes a programming style for irregular problems close to regular problems. In this way compile-time and run-time techniques can be readily combined.

A recent language extension for shared-memory programming that has caught a lot of attention is OpenMP. OpenMP is a kind of reincarnation of the old PCF programming model. The paper by Chapman and Mehrotra describes various ways how HPF and OpenMP can be combined to form a combined powerful programming system. Also the Java language can be fruitfully used as a basis for parallel programming. The paper by Carpenter, Zhang, Fox, Li, Li, and Wen outlines a conservative set of language extensions to Java to support SPMD style of programming.

General parallel programming languages have a hard time in obtaining optimal performance for specific cases. If the application domain is restricted, better performance can be obtained by using a domain specific parallel programming language. The paper by Spezzano and Talia describes the langauge CARPET intended for programming cellular automata systems.

Finally, the paper by Hofstadt presents the integration of task parallel extensions into a functional programming language. This approach is illustrated by a branch and bound problem example.

Programming Models and Languages.

Producing correct software is already a difficult task in the sequential context. The challenge is compounded by the conceptual complexity of parallelism and the requirement for high performance. Building on the foundations laid in Workshop 7 in the preceding instantiation of Euro-Par, Workshop 18 focuses on programming and design models that abstract from low-level programming techniques, present software developers with interfaces that reduce the complexity of the parallel software construction task, and support correctness issues. It is also concerned with methodological aspects of developing parallel programs, particularly transformational and calculational approaches, and associated ways of integrating cost information into them.

The majority of papers this year work from a "skeletal" programming perspective, in which syntactic restrictions are used both to raise the conceptual level at which parallelism is invoked and to constrain the resulting implementation challenge. Mallet's work links the themes of program transformation (here viewed as a compilation strategy) and cost analysis, using symbolic methods to choose between distribution strategies. His source language is the by now conventional brew of nested vectors and the map, fold, scan skeleton family, while the cost analysis borrows from the polytope volume techniques of the Fortran parallelization world, an interesting and encouraging hybrid. Skillicorn and colleagues work with the P3L language as source and demonstrate that the use of BSP as an implementation mechanism enables a significant simplification of the underlying optimisation problem. The link is continued in the work of Osoba and Rabhi, in which a skeleton abstracting the essence of the multigrid approach benefits from the portability and costability of BSP. In contrast, Keller and Chakravarty work with the well know data-parallel language NESL, introducing techniques which allow the existing concept of "flattening transformations" (which allow efficient implementation of the nested parallel structures expressible in the source language) to be extended to handle user-defined recursive types, and in particular parallel tree structures. Finally, Vlassov and Thorelli apply the ubiquitous principle of simplification through abstraction to the design of a shared memory programming model.

In summary, we expect from these session a fruitful discussion of the hot topics in concurrency and parallelism in several areas. This enlarged exchange of ideas can impact on advances of the discipline in the whole distributed and concurrency field.