Journal of Visual Languages and Computing (2001) **12,** 217–220
doi:10.1006/jvlc.2000.0201, available online at http://www.idealibrary.com on IDEAL®

Issues in Visual Parallel and Distributed Program Development

(Panel Discussion at the VL'2000 Workshop on Parallel and Distributed Programming)

Participants

Philip Cox (Dalhousie University, Canada), Holger Giese (University of Muenster, Germany), Bertrand Ibrahim (University of Geneva, Switzerland), Roland Huegl (University of Linz, Austria), SuTe Lei (Macquarie University, Australia), Katharina Mehner (University of Paderborn, Germany), Stephan Philippi (University of Koblenz, Germany), James Webber (University of Newcastle upon-Tyne, UK), Guido Wirtz (University of Meunster, Germany), Yijun Yu (University of Ghent, Belgium), Kang Zhang (University of Texas at Dallas, U.S.A.)

1. Introduction

AT THE END OF THE ONE-DAY WORKSHOP on Visual Methods for Parallel and Distributed Programming, the workshop participants discussed the state-of-the-art and future of visual languages and technology in the context of parallel and distributed programming. The discussion started with two questions.

- Besides the topics covered in the presented papers—what are the areas that visual techniques are especially well-suited for parallel and distributed programming?
- Which are the most promising applications to bring visual techniques to wider practical use?

The panel discussion evolved as summarized below.

2. Usefulness of Visual Approaches

In general, visual approaches can be more intuitive, especially in the high-level program design. People, however, may have claimed too much in visual solutions. If a visual method works just for the design stage, then one should just claim that visual approach helps in the design stage and general parallel modeling (Giese). We should focus on proper application domains, where performance is critically important and visual design of parallel/distributed programs may have the greatest impact on the performance (Yu).

1045-926X/01/040217+04 $35.00/0

As reported by an empirical paper (by Stankovic, Kranzlmueller and Zhang in this issue), it is interesting that even though people acknowledge the usefulness of visual approaches in parallel programming and the results show that visual languages do help in understanding and design, people are still used to programming using text. Visual methods should at least be useful for understanding, teaching and learning of parallel/distributed programming (Lei). Though it is possible for a visual language to support completely visual specifications of a parallel program, it is not cost-effective to write lower levels details, such as statements, graphically (Cox, Lei, Zhang).

In programming generally, textual languages with a complex syntax are typically used to encode multidimensional structures (algorithms, data arrays, etc.), and that appropriate visualization can make these structures more concrete. This principle should apply to languages for parallel/distributed programming (Cox).

2.1. How to Write Big Programs that can be Distributed Nicely to a Distributed Processing Environment? Can Visual Languages Help? (Wirtz)

There have not been many uses of visual languages in distributed programming and systems. Applications of visual languages in the areas such as Web design will certainly help end-users (Lei). Extending visual programming approaches to distributed computing is a matter of how to load computational nodes to remote processors in a distributed processing environment (Ibrahim). One thing that visual languages could help is to specify process-processor mapping, for example, how to submit computing jobs in a grid-based allocation representation to achieve load-balancing (Webber).

A visual language that allows the user to specify the configuration of a distributed system (e.g. whether on a tightly coupled multiprocessor system, a local area network or wide area network) will be particularly desirable (Cox). Some existing hardware description languages are very useful (Wirtz). But the main challenge is how to specify dynamic load balancing at run-time (Ibrahim).

Visual languages could also be used to specify software patterns, like what an architect does his/her job when designing a new building architecture. Familiar software patterns include those that perform operations on some composite objects such as linked lists, binary trees, etc. There are many other typical patterns in parallel and distributed programs, like loop dependencies, data distribution and communications (Yu).

2.2. How can Parallelization be Automatically Done? (Cox)

Best parallelization should be done automatically, and completely invisible to the user (Ibrahim). This is, however, difficult at the present.

Many people have asked about whether Prograph can be parallelized, for example, with the capability of automatic analysis of parallelism. It is, therefore, apparent that there is a strong need in the community for automatic parallelization in visual parallel programming languages or environments. B. Lanaspre at the University of Southampton, UK, has investigated automatic methods for dispatching parallel tasks according to data types used by these tasks (in 'Static Analysis for Distributed Prograph', Ph.D. thesis, 1998) (Cox). This work, as an example, could be applied to visual parallelizing tools.

2.3. At Which Stages (Design, Construction, Debugging and Performance Tuning) Visual Languages or Visualization are Most Useful? (Zhang)

Visual approaches should be useful in all the mentioned stages (Webber, Cox, Ibrahim). It is certainly true that visual methods are very useful in almost any science and engineering domains, even social science areas. Using graphs to visualize abstract concepts or to provide analog representations of real work phenomenon has been a common practice when people approach the design, modeling and analysis of any systems, whether on a paper or on drawing machine. People have largely acknowledged the usefulness of visual approaches using graphs (Wirtz).

The question then is at what level in high-performance computing it is desirable to use visual methods for specifications, given that we would like almost everything to be done automatically. If parallelization can be done nicely and parallelizing compilers are powerful enough, do we really need visual specifications (Ibrahim)? It is of course doubtful that parallelizing compilers can be so powerful. One obstacle, for example, that could prevent automatic parallelization is the consistency issue. It is desirable for a parallel programmer to be able to specify at least how much consistency should be maintained before parallelization. Of course, end-users are not expected to be able to provide such specifications (Webber).

3. Current Problems

The consistency issue is not simple. When the coefficient matrix of linear equations obeys a strict condition on its spectral radius, parallel execution of the asynchronous iterative solver can neglect the data dependencies yet can still converge to sequential results in the sense of epsilon range. To understand such a phenomenon, visualizing consistency semantics at the algorithmic level rather than programming level is highly desirable and needs also be supported (Yu).

End-users should not be expected to be able to conduct explicit parallel programming (Wirtz). Automatic parallelization tools are extremely limited. A way forward may be to separate parallelism in specific domains, such as string search.

How to parallelize a computation that involves a large number of objects? (Cox). Perhaps a good approach is to treat them as abstract data types and use them to specify what to do in the computation, rather than how to do (Ibrahim).

People in the parallel and distributed programming community have used various types of graph formalisms to visualize parallel software design, construction, debugging, and performance tuning. Different graphs are used in different development stages. A major challenge is to find a graph formalism that is not only generally intuitive and appeals to the scientific and engineering community, but also supportive across all the development stages (Zhang).

4. Future

4.1. Can Interface Heavy Software for End-users be Automated Using Visual Languages? (Cox)

Creating complex software, including the functionality of rotating and rendering images, can be challenging. Companies like Adobe surely needs some kind of visual tools to help

writing such complex software. It will be very desirable to have a language that these companies can easily extend or customize to suit their needs. Tools aiming at producing such languages are very important (Cox).

We may look for what visual languages and visualization can help or represent intuitively while text cannot, such as data types, pointers, and object patterns (Webber). It is generally not a good idea to convert from visual programs to the text and then parallelize the text for fast processing (Webber and Cox).

4.2. Would it be Possible to Have a Commonly Accepted Visual Formalism to Represent and Support Parallel Programming (as MPI for Message Passing)? (Zhang)

Dataflow may be a good candidate. Dataflow graphs have been used in parallel program design, analysis and even debugging. But dataflow alone is not good as a visual tool since the traditional dataflow languages have to use things like multiplexors to select data streams, which make dataflow graphs difficult to read (Cox). Certain control flow features may have to be used to convey such information as execution order of parallel programming components.

There is a need for parallel visual programming tools that allow users to visually specifiy both the levels at which parallelization can apply, and features that are parallelizable. So the users are relieved from having to learn parallel programming at lower levels (Ibrahim).

The success of visual approaches to parallel and distributed programming in the future will be largely dependent on how well the visual formalisms used could represent intuitively key issues in parallel/distributed programs, such as parallelism, communication and load-balancing. There is clearly a trend in high-performance computing research and industry that high productivity of parallel/distributed software increasingly relies on visual development tools that have user-friendly interface (Zhang).

KANG ZHANG AND GUIDO WIRTZ