ACM Applied Computing Review, **4**(1), Spring 1996, pp. 3-4

Object Oriented Programming in Parallel Scientific Computing An Overview of the Special Issue

Boleslaw K. Szymanski and Charles D. Norton
Guest Editors
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, 12180-3590, USA

1 The State of Object Oriented Scientific Computing

There is little argument that the programming of scientific applications, particularly on distributed memory parallel machines, remains difficult. Scientists are still productive, but this productivity comes at the high cost of designing and developing applications that are hard to maintain, modify, share with collaborators, explain and scale to larger problems. Many of these challenges stem from applying languages and techniques that do not promote abstraction in programming. The human cognitive ability to unify and generalize detailed concepts into abstract representations is the most powerful tool available in scientific and engineering disciplines. Unfortunately, we have not yet been successful in transferring this technique into our general application development process. Programming, via abstraction, is not common practice.

One of the most promising areas of opportunity to address abstraction in scientific programming involves object oriented technology. The advanced techniques and methodologies of this approach can bring coherency to the design and programming process. In March of 1996 in Kanazawa (Ishikawa Prefecture), Japan, the International Workshop on Parallel C++ was held to discuss the state of the art in parallel and distributed C++ technologies and future direction to high performance computing. This meeting, as part of the International Symposium on Object Technologies for Advanced Software held at the Japan Advanced Institute for Science and Technology, was sponsored by the Real World Computing Partnership and the Japan Society for Software Science and Technology. The workshop brought together a small number of leading researchers from around the world including Europe, United States, Japan and others, to investigate and critique the current status of object oriented methods in scientific computing. Topics included application development, software libraries, language and runtime systems, as well as interoperability. As our community gradually makes the paradigm shift toward abstraction in application development for applied computing, events such as the Kanazawa meetings increase the awareness of the current status of object technology in scientific programming. In this special issue of the ACM SIGAPP Applied Computing Review, we have invited workshop contributors to present position statements on their work. The views and experiences presented provide indications of where this methodology is currently applied and its directions for the future, which will influence emerging application development.

The editors would like to acknowledge support received during preparation of this issue from the National Science Foundation under grant CCR-9527151, and from the National Aeronautics and Space Administration under grant NGT-70334. The views expressed in this overview need not represent the position of the U.S. government.

2 Overview of the Special Issue

Finding the best way to support parallelism in scientific applications using object oriented parallel programming is still a topic of on-going research. Two basic approaches have emerged. One approach is to rely on libraries by building highly optimized, extensible class libraries that encapsulate parallelism. Users could use these class libraries without knowing anything about parallelism or about what goes on inside the class library. The rationale for this approach is that object oriented languages in general (and C++ in particular) provide a powerful mechanism for language extension via classes, inheritance, and templates. Additional extensions would only clutter the language. Furthermore, with no consensus on language features, compiler vendors are unlikely to support any language extensions, and users will not want to risk embracing the "wrong" feature.

Two examples of this approach are presented in this issue. Matthew Austern, Ross Towle and Alexander Stepanov from Silicon Graphics, Inc. in *Range Partition Adaptors: A Mechanism for Parallelizing STL* describe a new type of adaptor for the C++ Standard Template Library that can be used to express parallelism. In addition to iterators, algorithms, containers and allocators, the STL also includes adaptors for transforming one interface into another. The paper describes a *range conversion adaptor* that converts a one-dimensional structure (a range) into a two-dimensional structure (a collection of subranges). This operation is essential for expressing such basic parallel strategies as the parallel `for_each` iterator.

Scott Baden from University of California, San Diego in *Software Infrastructure for Non-Uniform Scientific Computations on Parallel Processors* presents two C++ class libraries, LPARX and KeLP, that offer portable performance across a range of parallel machines. The goal is to support non-uniform computations arising in such applications as adaptive solvers of partial differential equations, particle simulations or spatially extended dynamical systems. The key notion is that of *structural abstraction* that separates dynamic non-uniform data layouts from communication patterns incurred by parallelization. The libraries provide an infrastructure consisting of a layered set of such abstractions.

The second direction of research on supporting parallelism in object oriented programs focuses on object oriented language extensions. The main argument in favor of this approach is that parallel composition is as important a concept as sequential composition. With concurrency being a part of the language, compiler technology can more readily develop parallel code optimizations.

One of the papers presented in this issue directly addresses the issue of parallel composition extensions in object oriented language. Three other papers in this issue advocating extensional approaches describe efforts in three geographically dispersed centers: HPC++ group in US, Europa consortium in Europe and MPC++ efforts in Japan.

Dennis Gannon, Shridar Diwan and Elizabeth Johnson from Indiana University, Bloomington in *HPC++ and the Europa Call Reification Model* compare the approaches being developed in US and Europe. HPC++ consists of a modest set of compiler directives (focusing on parallel loops), a parallel extension to the STL and a multidimensional array class. The call reification model relies on representing a remote object by a proxy object which, when called, collects arguments, sends them to the remote location for execution, receives the generated value and returns it to the caller. In the paper, the authors show how call reification can be implemented in HPC++.

Yutaka Ishikawa from Real World Computing Partnership, Japan in *MPC++ Approach to Parallel Computing Environment* presents the approach which is actively being pursued in Japan. MPC++ is an extension of C++ that supports parallel and metalevel programming. The system includes a small set of parallel description primitives and an extendible programming facility that enables the user to extend or modify C++ language features. The core notion is that of a metalevel architecture that defines an abstract compiler for each level of an abstract architecture also supporting user-defined builtin libraries.

Finally, Andrew Chien from University of Illinois at Urbana-Champaign *ICC++ – A C++ Dialect for High Performance Parallel Computing*, describes a new concurrent C++ dialect which supports a single source code for sequential and parallel program versions. ICC++ relies on program annotations to identify potential concurrency. Concurrency control is enforced at the object level and multidimensional arrays are integrated with the object system. Distributed data abstractions are further supported through collections - a compatible extension of arrays. Since annotations identify potential but not actual parallelism, the system can optimize execution granularity to match the target architecture.

Carl Kesselman from California Institute of Technology in *High Performance Parallel and Distributed Computation in Compositional CC++* presents a parallel extension to C++ that supports parallelism in the different programming styles over a broad range of parallel programming paradigms, such as message passing, shared memory and/or active objects concurrency. C++ major feature is a support for object-oriented programming, yet the language enables a range of different, non object-oriented programming styles, such as the generic algorithms that were widely used in the definition of the Standard Template Library (STL). Hence, CC++ attempts to enable parallel execution in all types of C++ programs, not just the ones that use the object-oriented programming style. CC++ consists of six language extensions that augment C++ with a well defined parallel semantics. These extensions, when combined with the rest of the C++ language, create reusable parallel paradigm libraries which implement a specific parallel programming paradigm. The author illustrates use of CC++ in defining a reusable parallel paradigm libraries and describes its use in one parallel application enables the programmer to combine different types of parallelism in a single application.

If parallel processing components conformed to a standard interface description, they could be used transparently by commercial application developers. That would make parallel computing relevant to a broader user base and encourage vendors to develop parallel hardware for the commercial markets. The feasible way of doing this is to encapsulate parallelism within objects, making the parallel component a particularly fast version of an existing sequential code.

From that perspective, the decision of the High Performance Fortran (HPF) designers to base the language on Fortran 90 was very helpful. Fortran 90 includes all the basic constructs required of object oriented programming and there is an increasing interest in object oriented programming using Fortran 90. Viktor Decyk from University of California, Los Angeles and the editors of this special issue describe their experience with Fortran 90 parallel object oriented programming of plasma simulation based on the Particle-in-Cell (PIC) approach in the paper entitled *On Parallel Object Oriented Programming in Fortran 90*. The authors argue that the new constructs of Fortran 90 encourage the creation of abstract data types, encapsulation, information hiding, inheritance, generic programming and many features of the language (beyond array syntax) ensure the safe development of advanced programs. Additionally, the Fortran 90 standard is backward compatible with Fortran 77 allowing new features to be incrementally introduced into existing applications. Many of the new ideas in Fortran 90 fit in well with object oriented programming. The authors demonstrate use of these features of Fortran 90 in examples taken from parallel codes for plasma simulation. They also provide performance results indicating that Fortran 90 parallel programs introduce smaller execution penalty for abstraction and object oriented-ness than comparable programs in C++.

The editors hope that the presented collection of the invited papers will provide the readers with the interesting snapshot of the current state-of-the-art in the area of parallel object oriented programming. The editors would like to thank the authors of the presented papers for their timely contributions. Finally, we would like to thank Barrett Bryant from University of Alabama, Birmingham, the Editor-in-Chief of the *Applied Computing Review* for extending an invitation to us to prepare this issue and his encouragement and help with bringing it to fruition.