POOMA 94 : Issues and Views*

Aswini K. Chowdappa Anthony Skjellum

Department of Computer Science &
NSF Engineering Research Center for Computational Field Simulation
Mississippi State University
PO Box 6176
Mississippi State, MS 39762

TR-CS-941219

1 Abstract

This paper summarizes and organizes the views expressed and the issues discussed by a broad cross section of the participants at the workshop on Parallel Object-Oriented Methods and Applications (POOMA) held in Santa Fe, New Mexico between Dec 5-7, 1994 [11]. About 6 years after the first C++ conference, object-oriented programming (OOP) and its relevance to large-scale applications was discussed at this workshop. The special focus was on the effectiveness of OOP methodology in terms of performance when coupled with the parallel programming paradigm.

2 Object-oriented methods: who needs them ?

The focus of late has been on developing computing hardware that can deliver giga flops performance and not much thought has been given to the development of software that can make the job of programming parallel hardware easier [6]. The object-oriented programming (OOP) revolution has swept desktop computers. An effort should be made to provide desktop-type user interfaces to higher-end machines. We need a industry standard interface to support machine independent Object-Oriented parallel programming with performance, portability and reusability as the key goals. In that sense everybody needs object-oriented methods.
Traditionally OOP has been looked at with a lot of skepticism because of poor performance. Some of the presentations gave performance figures for

- C++ layered on top of optimized Fortran linear algebra kernels that matched performance figures for C routines [12]. The C++ shell is mainly to provide extensibility (via polymorphism) and usability (simplified interface).

*Work supported in part by the NSF Engineering Research Center for Computational Field Simulation, Mississippi State University and DOE Cooperative Agreement DE-FC04-94AL 98921.

- There were also some figures for polymorphism overheads (3 $\times$ C++ function calls) on RS/6000 and Sun workstations [12]
- Other figures quoted estimate 20-300%, 50-200% C++ overheads as compared to C and Fortran codes respectively [4].

All these point to one major handicap, the lack of an efficient optimizing C++ compiler. Some of the optimizations can come from inlining, procedure cloning, and other speculative optimizations [4]. All these would be stop-gap arrangements. Also, in some cases, C++ overhead was attributed to compiler-generated temporaries. But, one of the speakers claimed his library's efficiency resulted from reuse of temporaries [13].

3 Object-oriented numerics: What is it?

Object-oriented programming should not be confused with

- programming with C++
- programming with abstract datatypes

There is a misconception among the programming community that C++ is an object-oriented language. On the contrary, unlike smalltalk, C++ is not an object-oriented language. C++ just supports object-oriented programming. One should be careful during the process of decomposing a problem domain into a hierarchy (to cope with complexity), as a deep hierarchy tends to reduce efficiency [3]. Also, polymorphic classes can also bring down efficiency. For high-level classes one should use *polymorphic classes*, and one should use *value classes* for low-level computations. Container classes should be used to manage complexity and templates are ideal to implement container classes.

3.1 Issues in large-scale parallel object-oriented applications & libraries

The main issue in large-scale parallel object-oriented applications and libraries is complexity. It is evident that "large" applications that are "parallelized" breed "complexity." There was unanimous agreement that object-oriented methods and techniques are the most appropriate to tackle complexity [1]. This leads us to the issue of distributed objects interacting with each other. These objects would have communication primitives embedded within each. Such embedding of communication primitives leads to problems like "inheritance anomaly" [15]. *Inheritance Anomaly* is caused by embedding synchronization with computation in derived classes. If the behavior of the derived class changes then the interaction of the derived class with the base class changes.

3.2 Frameworks: Meta-templates for code reuse

Frameworks are an extension of the idea of templates for programming large-scale applications. The idea is to replace methods and "stuff" objects into a pre-existing environment to particularize it into a new application. This frame-based programming promotes large-scale code reuse [2]

3.3 Language extensions

As the high-performance scientific computing community migrates to object-oriented methods in order to gain additional advantages such as extensibility, reusability, they are using two approaches:

- Convert the existing C and Fortran application to C++ and then parallelize the C++ application.
- Develop the parallel C++ application from scratch.

In either case, C++ language extensions are necessary to support parallel programming primitives. There are a variety of projects that are aimed at C++ language extensions as well as supporting parallel object-oriented environments are underway. Notable among these are [4, 7, 8, 9, 10]. Compositional C++ is an effort to provide a migration path from sequential C++ applications to parallel C++ applications. CC++ has constructs to represent concurrency explicitly. The developers of CC++ believe that concurrency is a control structure and hence it should not be abstracted from the user [10]. Such explicit support for parallelism within the language supports correctness. The participants were divided over the issue of abstraction of control. It was suggested that we should have functions that contain high-level control structures. The compiler should enforce determinism as determinism makes debugging easier [14]. As regards *language extensions* vs *libraries*, the majority opinion suggested that we need both. Since each library has its own unique features there could be applications that may use more than one library. We would also need multi-lingual interoperability so that these libraries can "talk" to each other [9].

4 Performance, efficiency & other issues

Views were also expressed to the effect that the high-performance computing community has too many goals (expectations) from object-oriented programming in general and C++ in particular. It was suggested that one could leverage better advantages from this paradigm if we reduced the number of issues and focused on the few important ones [5]. The most important ones are as follows [5]:

- Efficiency
- Correctness

The other issues that are not as important as the ones mentioned above are as follows [5]:

- Portability - comes from standardizing hardware
- Code reuse - comes from popular, correct libraries
- Simplicity and ease of use - This is a issue concerning user interface, not a scientific issue.
- Reliability - comes with correctness and is more of a technicality.

The overall philosophy of object-oriented programming should be to produce "sufficiently" efficient programs by choosing objects of right granularity for the application.

5 Conclusions

It was quite evident from the attendance (about 138 participants) at the workshop that the high-performance computing community expects long-term benefits from the object-oriented programming paradigm. There was a general consensus that we could leverage acceptable levels of performance with maturation in compiler technology. But there was a divergence of views on identifying the "key" issues, but this is understandable considering that this is a new research field. A majority of participants also felt that the next workshop/conference should *not* focus on adopting any informal standards. On the other hand,, the discussions should continue so that people can share their experience for on-going projects. One of the authors suggested the formation of a newsgroup devoted to this topic, and the idea was well received.
Finally, Is parallel object-oriented programming here to stay? A definite answer to this question can be given in the coming months. But from the level of excitement seen, one can be optimistic and look forward to a positive answer.

References

[1] John Amborsiano. Issues in large scale object-oriented applicatons & libraries. Presentation at POOMA 94, December 1994.

[2] Robert Armstrong. The use of frameworks for scientific computing. Presentation at POOMA 94, December 1994.

[3] Kent Budge. Just what is object-oriented numerics? Presentation at POOMA 94, December 1994.

[4] Andrew Chien. Efficient concurrent object-oriented programming: Lessons from the Illinois concert system. Presentation at POOMA 94, December 1994. *http://www-csag.cs.uiuc.edu/project/concert.html.*

[5] Eric Van de velde. Parallel object-oriented languages & libraries : performance and & efficiency. Panel Discussion, POOMA 94, December 1994.

[6] David Forslund. Who needs objects ? Presentation at POOMA 94, December 1994.

[7] Dennis Gannon. Converging themes in parallel object-oriented design. Presentation at POOMA 94, December 1994. *http://www/cica.indiana.edu/sage/home-page.html.*

[8] Andrew Grimshaw. Mentat: High performance object-oriented parallel processing. Presentation at POOMA 94, December 1994.

[9] L. V. Kale. Charm: Modularity and reuse in parallel object-oriented software. Presentation at POOMA 94, December 1994. *http://charm.cs.uiuc.edu.*

[10] Carl Kesselman. Compositional c++: A parallel object-oriented programming language. Presentation at POOMA 94, December 1994.

[11] Los Alamos National Laboratory. *First Workshop on Parallel Object-Oriented Methods and Applications POOMA*, Santa Fe, New Mexico, December 1994. *http://www.acl.lanl.gov/Pooma.html.*

[12] Roldan Pozo. High performance linear algebra in C++. Presentation at POOMA 94, December 1994.

[13] Daniel Quinlan. Parallel adaptive mesh refinement. Presentation at POOMA 94, December 1994.

[14] Guy Steele. Parallel language extensions vs parallel libraries & multilevel frameworks. Panel Discussion, POOMA 94, December 1994.

[15] Laurent Thomas. Concurrency in object-oriented languages. Presentation at POOMA 94, December 1994.