The OOPS Framework: High Level Abstractions for the Development of Parallel Scientific Applications

Eloiza Sonoda [*]
Instituto de Matemática e Estatística
Universidade de São Paulo
São Paulo, SP, Brazil
elo@ime.usp.br

Gonzalo Travieso
Instituto de Física de São Carlos
Universidade de São Paulo
São Carlos, SP, Brazil
gonzalo@ifsc.usp.br

Abstract

OOPS (Object-Oriented Parallel System) is a framework designed to support programming of concurrent scientific applications for parallel execution. The high level abstractions provided by OOPS free the programmer from dealing with many parallel implementation details, such as the ones found in hand-coded MPI programs. However, for performance reasons, parallelism is not completely hidden. The use of the classes supplied by OOPS simplifies the implementation of parallel applications, without introducing significant overhead.

Categories and Subject Descriptors D.1.5 [*Software*]: Programming Techniques—Object-oriented programming; D.1.3 [*Software*]: Programming Techniques—Concurrent programming

General Terms Design, Performance

Keywords Object-oriented frameworks, parallel scientific programming

1. Introduction

Parallel processing is an appropriate way to satisfy the high demand for computational power required by scientific applications [6]. But in the state of the art of parallel systems programming, the responsibility for developing a program that efficiently explores the available parallel resources lies with the application programmer.

The development of a parallel program is a complex task. In parallel programming, the developer must deal with considerations such as distribution of tasks across the different processors; distribution of data; communication and synchronization among the tasks; load balancing among the processors; and communication latencies. To enable a widespread use of parallel computing, it is thus of fundamental importance that the developer be freed from involvement in too much implementation details, working in contact with a sophisticated interface that simplifies the development of scalable, efficient, and portable programs [2].

In this context, the development of tools to assist parallel systems programming is a necessity. Various approaches have been followed to enable and support the development of parallel applications. It is well established that object-orientation can facilitate reuse and extension (through inheritance and polymorphism) in parallel and concurrent programming [1]. The OOPS framework (Object-Oriented Parallel System) is here proposed as an aid for the programming of scientific applications for parallel systems. Its aim is to foster the implementation of such applications by reducing the semantic gap between the organization of the parallel algorithm and the resulting source code. Using object-orientation, OOPS also helps encapsulate expert knowledge about efficient parallel programming techniques in easy to use interfaces that can be reused in many applications.

[*] Financial support by CAPES.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

2. The OOPS Framework

An OOPS program is an explicit parallel program, where the programmer decides data distribution, the elements to execute in parallel, and the communication operations needed. Parallel programming is assisted by providing object-oriented interfaces for these operations in a higher level of abstraction than common in parallelization libraries such as MPI [7], allowing the programmer to keep the focus on the problem itself. OOPS is tailored to scientific applications that require high performance and deal with regular structures and containers, such as arrays and matrices. It models various forms of data distribution and communication structures among the processors. OOPS is implemented in C++ over the MPI standard. The use of these standard language and tools enable portability to different parallel architectures. Although the framework can be implemented on all parallel systems where a communication library is available, its design decisions are tailored for the use in clusters or other intra-business solutions, as the communication costs will be too high for grid computing [4] environments.

Some features provided by High Performance Fortran (HPF) [5], such as distribution modes of matrix elements are useful to scientific programmers and therefore included in the OOPS design. The flexibility introduced by object-orientation allows OOPS to support other important parallel constructions, e.g., task parallelism with parallel components and communication of user-defined types through serializable objects.

2.1 OOPS Features

The main OOPS abstractions are *virtual processors* to represent the concurrency of the program; *groups* to organize virtual processors; *topologies* to specify a communication structure for a group; *distributed containers* to hold distributed data and take care of its access; and *parallel components* to specify the code to be executed on the processors. These abstractions are illustrated in Fig 1, where a group of nine virtual processors is arranged in a grid topology.

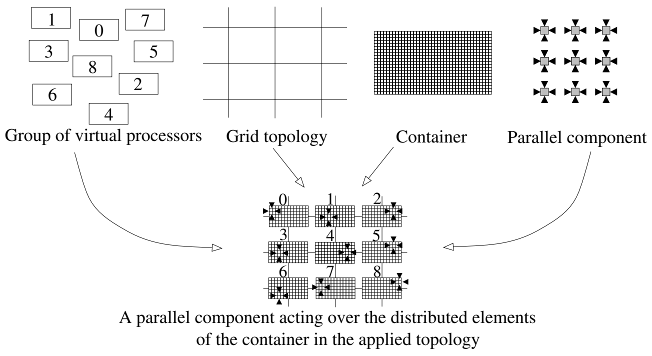
1
0
7
3
8
5
6
2
4
Group of virtual processors
Grid topology
Container
Parallel component
0
1
2
3
4
5
6
7
8
A parallel component acting over the distributed elements
of the container in the applied topology

Figure 1. OOPS abstractions

A matrix is distributed across the virtual processors and a parallel component executes over its elements.

The fragment below shows the implementation of the parallel program entry point `OOPS::Main::executeOn` of a simple program. This method receives a pointer to an `OOPS::Group` object which holds all available virtual processors.

```
1 void OOPS::Main::executeOn(const OOPS::Group
2                                        *allProcs) {
3   OOPS::TopologyPipe topologyPipe(allProcs);
4   int x = 0;
5   topologyPipe.fromPrevious(x);
6   x++;
7   topologyPipe.toNext(x);
8 }
```

Line 3 shows the declaration of a *communication topology*. Topologies provide communication through message passing between the virtual processors in a group, and define a relation of neighborhood among then. Communications in the topology can be *point-to-point* between two neighboring processors or *collective* among all virtual processors in a (sub)group. The pipe topology used in line 3 defines that virtual processor i can send messages to virtual processor $i + 1$ (if $i + 1$ does not exist, the message is discarded). Lines 5 and 7 show examples of point-to-point communications in this topology. OOPS already implements some frequently used topologies such as pipe-ring, linear, ring, grid, and torus, and others can be implemented as required.

Given a topology, data is distributed across its processors, through a *distributed container*. Examples are `OOPS::Vector<T>` and `OOPS::Matrix<T>` which are parametric types that accept all basic types and types derived from `OOPS::Sendable`, a base class for serializable objects. Distributed containers use a distribution mode (derived from `OOPS::Distribution`) in High Performance Fortran style. This can be seen in the following code fragment. A linear topology (similar to the pipe one, but allowing communication in both directions), is declared at line 1; lines 3–5 declare vectors of `N` elements with blocked distribution (line 2) where blocks of contiguous elements are distributed over the virtual processors; lines 6-7 fetch the data from a file and distribute them as declared. The code proceeds with an element-wise product of vectors `v1` and `v2` (executed in parallel in all processors) in line 8 and a summation (over all processors) in line 9.

```
1 OOPS::TopologyLinear topologyLinear(allProcs);
2 OOPS::DistributionBlocked block;
3 OOPS::Vector<double> v1(N, block, topologyLinear),
4                      v2(N, block, topologyLinear),
5                      v(N, block, topologyLinear);
6 ifstream file1("vec1.dat"), file2("vec2.dat");
7 file1 >> v1; file2 >> v2;
8 v = v1 * v2;
9 double scalarProduct = v.sum();
```

A *parallel component* defines the code to be executed by all virtual processors in a group. It can use the containers distributed in these processors. Different components can be combined through *sequential* or *concurrent* composition [3]. The concurrent composition of parallel components allows task parallelism in OOPS.

2.2 Performance

The performance of a parallel application is a factor of fundamental importance, as it is usually the justification to the use of parallel programming. Many design decisions in the OOPS implementation reflect the need of high performance resulting in a small overhead due to the use of the framework in comparison with a direct use of MPI, as described elsewhere [9, 8]. As an example, for a parallel Mandelbrot fractal computation, the overhead is about 4% for two processors and drops to less than 1% for eight processors.

3. Conclusion

The increasing availability of parallel systems and their importance to achieve high performance make them ideal for scientific applications. The OOPS framework substantially facilitates the development of parallel scientific applications by supplying high level abstractions, such as communications topologies and distributed containers. That alleviate the programmer's task without significant performance overhead.

OOPS is an appropriate tool to support parallel programming. It demonstrates a promising approach to the development of parallel programs that should be further developed to be adequate to a wider range of application types.

References

[1] J.-P. Briot, R. Guerraoui, and K.-P. Löhr. Concurrency and Distribution in Object-Oriented Programming. *ACM Computing Surveys*, 30(3):291–329, 1998.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White. *Sourcebook of Parallel Computing*. Morgan Kaufmann, 2003.

[3] I. Foster. *Designing and Building Parallel Programs*. Addison Wesley, 1995.

[4] I. Foster, C. Kesselman, and S. Tuecke The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[5] High Performance Fortran Forum. *High Performance Fortran Language Specification*, 1993. Version 1.0.

[6] T. G. Mattson, B. A. Sanders, and B. L. Massingill. *Patterns for Parallel Programming*. Addison-Wesley, 2005.

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, 1995.

[8] OOPS Home Page. `http://appl.ifsc.usp.br/oops/`. visited 08/2006.

[9] E. Sonoda. *OOPS — Object-Oriented Parallel System*. PhD thesis, Instituto de Física de São Carlos - USP, 2006. In Portuguese.