Towards an Object Oriented Distributed Matrix Computation Library above C and PVM

Pierre Manneback
Guibo Peng
Laboratoire P.I.P.
Faculté Polytechnique de Mons, Belgium
email : pm,peng@pip.fpms.ac.be

October 7, 1994

1 Introduction

The focus of this contribution is not to present results but to ask some questions to the parallel computing user community. Among others: is PVM a good tool for developing parallel application? How to integrate object oriented design into parallel applications? We strongly believe that answers to these questions are important before designing complex parallel packages, such as ScaLAPACK[3] or LAPACK++ [5]. Our opinion is that PVM is good at the system but not at the application level: a user should not have to struggle with PVM calls in its program! We believe also that truly Distributed Object Oriented (DOO) packages have to be designed from scratch and not reuse act.psortran subroutines, as it is proposed in [5].

It is now widely admitted that traditional imperative programming languages such as Fortran or C are not the best suited for developing complex packages. They present major lacks for extensibility, maintenability or reusability, which are crucial objectives in designing libraries. Curiously enough, in the world of parallel computing which is intrinsically complex, these languages tend to dominate. HPF or Fortran/PVM, C/PVM are now de facto standards.

Object oriented (OO) design provides excellent opportunities to overcome limitations of traditional languages. Their major drawing power is to offer the opportunity to encapsulate complex coding and provide user-friendly interface. Some attempts to exploit object oriented technology in parallel computing have recently been conducted (see for instance EPEE[6], pC++[1] or LAPACK++ [5]). One drawback for their propagation is the lack of standards (which OO language, which communication libraries, which data distribution). In this work, we question the benefits one could obtain in developing numerical libraries using a neat and widespread DOO language. We consider CC++, developed at CalTech [2]. This public domain compiler generates code in C and PVM, ensuring portability. It has a very elegant syntax, freeing the programmer to write cumbersome calls to PVM.

In next section, we present some prospects of a truly DOO matrix computation package. Note that we have not currently written this package, so a lot of questions subsist: what about performance, reusability, extensibility ? Our aim is to argue about some important points. Should a DOO package be based upon existing Fortran package or not (like LAPACK++ based on LAPACK) ? How to express communications in the application program ? Directly in PVM or by extension languages which generate calls to PVM ? We believe that answers to these questions have to be expressed before considering to raise standards. We believe also that CC++ can provide a very interesting platform for developing DOO packages.

2 Case for truly DOO matrix computation package

CC++ is a parallel language that extends C++. Its current compiler is a source to source transformer, replacing extensions to C++ by calls to a runtime library (Nexus) which supports threads and communication [2]. Currently, the runtime library is built upon PVM. Architectures which are now supported are Sun workstations (under SunOS 4.1.2 and Solaris 2.3) and IBM RS6000. Although brand new and still to be improved (specially for performance consideration), we believe that CC++ offers a very interesting alternative to cumbersome programs in C or Fortran with calls to communication libraries, e.g. PVM.

Indeed, CC++ just extends C++ by eigth constructs: *par*, *parfor*, *spawn*, *atomic*, *global*, *processor objects* for defining data distribution, *data transfers* for describing communication protocols. Combined with C++, these extensions provide a powerful parallel programming environment.

An important project in the OO numerics is LAPACK++ [5], which is an OO extension to LAPACK library. Its objective is to maintain speed and efficiency of corresponding native Fortran code while providing all the advantages of OO design. A distributed version is under work. Some critics about LAPACK++ are that genericity is not supported and that it does not exploit all the power of OO language (dynamic binding, multiple inheritance). It looks as a nice user-interface to LAPACK, but not a truly DOO package.

Our aim is to exploit all the power of CC++ to build an intrinsic DOO matrix computation package, keeping in mind genericity, reusability, extensibility and efficiency. Distributed vector and matrix classes of LAPACK++ are good frameworks for basic objects of this package. They have however to be generic and to be extended with standard methods (reduction operations e.g.). BLAS-like operations have to be integrated in the OO code, rather than be linked to external Fortran functions. It is the same with the communication functions, which have to be integrated in OO framework. In this way, we are strongly supporting languages which hide complexity of communication calls to the user, without however hiding completely communications.

In our opinion, the place of communication libraries are at the system or compiler level and not at the application level. We develop this idea in the next section.

3 What is the place of PVM in DOO ?

A standard communication library is a must at a low level of parallel and distributed programming, to ensure portability while keeping efficiency. However, we strongly feel that at the application level, direct calls to this communication library are not the most practical. A real alternative is new parallel languages or extension to sequential languages (as CC++ for C++) which encapsulate these calls. A crucial point is of course the compiler capability to generate efficient code. Some results [6] allow to conjecture that efficiency can be maintained, even in hiding direct communication calls. PVM has its merit, at the system level, but perhaps not at the application level. One good measure of some problems with PVM on the application side is to do the weekly count of all reported questions in the newsgroup comp.parallel.pvm: the analysis of the questions show that a significant number of users have problems with PVM coding! Isn't it because PVM is not for every programmer? Only system programmers or compiler writers should explicitely use it. Let the others exploit elegant languages and objects!

References

[1] F. Bodin, P. Beckman, D. Gannon, S. Narayana, S. Yang, *Distributed pC++: basic ideas for an object parallel language.* In *Proceedings of the First Annual Object-Oriented Numerics Conference (OONSKI'93)*, Rogue Wave Software Publish., Sunriver, Oregon, 1993, pp. 1-24.

[2] P. Carlin, J. Patton, *CC++ V. 0.2. documentation.* CalTech Laboratory, ftp site cs.caltech.edu, 1994.

[3] J. Choi, J.J. Dongarra, R. Pozo, D. W. Walker, *ScaLAPACK: a scalable linear algebra library for distributed memeory concurrent computers*, Frontiers of massively parallel computing, Mc Lean Ed., Virginia, 1992.

[4] J.J. Dongarra, R. Pozo, D. W. Walker, *A design overview of object-oriented extensions for high performance linear algebra*, Computer Science Technical Report, University of Tennesse, 1993.

[5] J.J. Dongarra, R. Pozo, D. W. Walker, *LAPACK++ v. 1.0 User Guide* Computer Science Technical Report, University of Tennesse, 1994.

[6] F. Guidec and J.-M. Jézéquel, *Numeric parallel programming with sequential object oriented languages.* In *Proceedings of the First Annual Object-Oriented Numerics Conference (OONSKI'93)*, Rogue Wave Software Publish., Sunriver, Oregon, 1993, pp. 55-69.