Numerical library reuse in parallel and distributed platforms*

Nahid Emad[1], Olivier Delannoy[1], Makarem Dandouna[1]

PR*i*SM Laboratory, University of Versailles
45, avenue des États-unis, 78035 Versailles cedex, France

Keywords: large scale distributed systems, numerical library, code reusability, design model

Abstract. In the context of parallel and distributed computation, the currently existing numerical libraries do not allow code reuse. Besides, they are not able to exploit the multi-level parallelism offered by many numerical methods. A few linear algebra numerical libraries make use of object oriented approach allowing modularity and extensibility. Nevertheless, those which offer modularity together with sequential and parallel code reuse are almost non-existent. We analyze the lacks in existing libraries and propose a design model based on a component approach and the strict separation between computation operations, data definition and communication control of applications. We present then an implementation of this design using YML scientific workflow environment jointly with the object oriented LAKe (Linear Algebra Kernel) library. Some numerical experiments on GRID5000 platform validate our approach and show its efficiency.

1 Introduction

To solve linear algebra problems on large scale distributed systems an application can rely on existing libraries such as LAPACK[1] which provides a set of routines that can be used to create solvers. Parallel solvers for distributed memory architectures can be built on top of the services provided by sequential libraries. The approach consists in building the parallel version by using distributed versions of the basic operations used in the library. For example in LAPACK the parallelization is done by the parallelization of BLAS. Nevertheless, these libraries allow neither data type abstraction nor code reuse between the parallel and sequential versions of the applications. That means the subroutines of the solvers are not able to adapt their behaviors depending on the data types. Those subroutines must be defined once for use in sequential and once again in parallel. The component approach used in libraries such as PETSc (Portable, Extensible Toolkit for Scientific Computation)[2] or Trilinos [6] increased drastically the modularity, interoperability and reusability of high level components within the libraries as well as in the user applications. It increases the ease of use, code reuse, and maintainability of libraries. Nevertheless, it doesn't allow sequential/parallel

* Candidate to the Best Student Paper Award

code reusability. The Linear Algebra Kernel (LAKe)[9] is an object oriented library in C++ which makes use of MPI. It introduces code reuse between the sequential and the parallel versions of an application. However LAKe doesn't allow to use concurrently the parallel and sequential versions of a code inside the same application. This feature is required to build hybrid numerical methods in the context of distributed computing. These methods are defined by a set of collaborating classical iterative methods called co-methods. Each co-method aims at decreasing the number of iterations required by the method to compute its results. An extended version of LAKe proposed in [4] allows it to support the hybrid methods. Nevertheless, the scalability of the reusability offered by this extension is limited.

In this paper, we propose a design model for numerical libraries allowing their reuse on parallel and distributed systems. Our approach is based on three levels of abstraction concerning computation aspect, data definition and communication control of an application. The simultaneous reusability between the sequential and the parallel codes is possible thanks to this abstraction. We show that our design can be mapped on some scientific workflow environments. We present then the implementation of our approach using YML scientific workflow environment (http://yml.prism.uvsq.fr/) jointly with LAKe library. We will see that the approach makes possible to exploit the hybrid methods in the context of large scale distributed systems. Finally, we give the results of some experiments in order to validate our solution.

2 Linear Algebra Libraries

2.1 Imperative numerical libaries

In order to implement numerical solvers, one can use libraries such as LAPACK [1] and ARPACK[7] written in FORTRAN using a traditional imperative programming style. They consist in a set of routines which provides the individual steps of the iterative methods. Parallel solvers for distributed memory architectures can be built on top of the services provided by the aforementioned libraries. This approach has been used to build libraries such as ScaLAPACK[3] and P_ARPACK[8]. The parallel solvers exploit intra co-method parallelism. Nevertheless, these libraries allow neither data type abstraction nor code reuse between the parallel and sequential versions of the applications. That means the subroutines of the solvers are not able to adapt their behaviors depending on the data types. Consequently, those subroutines must be defined once for use in sequential and once again in parallel and then the application code is different if using the parallel or sequential library.

2.2 Object oriented numerical libraries

The object oriented approach used in libraries such as PETSc [2] or Trilinos [6] enforced drastically the modularity, interoperability and reusability of high

level components within the libraries as well as in the user applications. Using PETCs or Trilinos, the application specifies the building blocks of the solver. However the solver is provided by the library. The application code no more contains the logic of the method. It provides parallel and sequential solvers and allows to make use of one and/or the other in the same application. However these parallel and sequential solvers still use different application codes. This is also true for the implementation of the library.

LAKe is an object oriented library written in C++. It defines a framework to implement iterative solvers. The design approach of this library is based on a strict separation between the computation part, which is composed by numerical algorithms and services, and the data management and communication part of the application code. The latter are used to represent both sequential and parallel data type used by LAKe computation part. Using the object oriented approach and template based generic programming provided by C++, LAKe allows the computation part to be common to both sequential and parallel versions of the application. The computation part of the library is identical in the case of a sequential data set or distributed data set. The parallel version of LAKe makes use of the message passing interface (MPI) standard version 1 [5] for communication between the various involved computation processes. However the use of MPI is completely transparent to the user. He/she can switch from a sequential solver to a parallel one by changing the type of the matrix representing the data. LAKe achieves code reuse between sequential and parallel versions thanks to a strict separation between the computation and the data/comunication management aspects of applications.

The intra-method communication of the parallel version of LAKe increases the time performance when handling huge matrices. However LAKe is not suitable to implement hybrid methods. For hybrid methods we need two levels of parallelism. Using MPI means we need local communication at the co-method level and global communication between the co-methods composing the hybrid method. LAKe provides no access to the MPI communicator to client applications nor to the computation part of the library.

An extension of LAKe has been proposed to support hybrid methods. It makes possible the use of sequential and parallel co-method processes concurrently within the same application. This extension described in [4] discusses a solution that matches the architecture design of LAKe. The user of the library must explicitly define the number of processors allocated to each process representing a co-method. However, its use is not easy due to the configuration of the communicators (with MPI standard version 1). This limits the scalability of the solution proposed to only a few number of co-method processes. Experiments have been done up to three concurrent co-methods.

3 A reusable numerical library design model

Most of previously mentioned libraries suffer from many problems. Imperative numerical libraries lack portability, modularity, interoperability. Despite mod-

ularity and reusability of their high level components, object oriented libraries such as PETSc, Trilinos or LAKe do not allow the simultaneous reusability of components between the sequential and the parallel versions of an application. Extended LAKe allows this kind of reusability but it is not scalable. We notice that all these libraries lack an additional level of abstraction which is necessary to achieve such a kind of reusability.

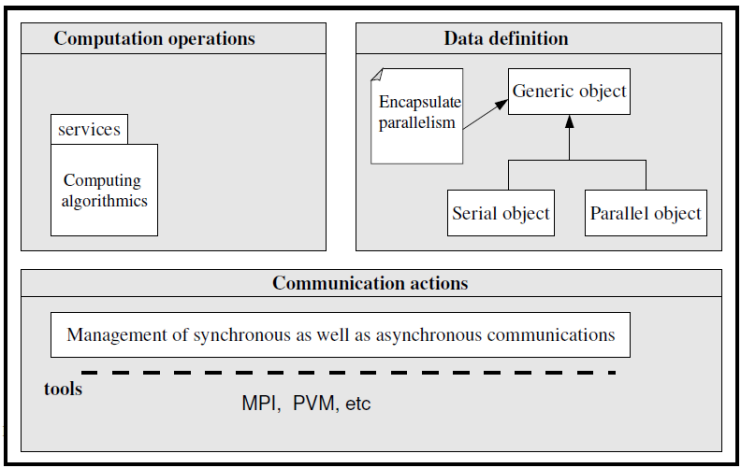

Fig. 1. Reusable numerical libraries design

To remedy to these problems, we propose a library design model based on three levels of abstraction. That means, a model which separates strictly the computation aspect, the data definition and the communication actions of applications (see figure 1). The data definition includes data types abstraction. The computation aspect represents all computation components. These two components communicate through the communication actions. Our main goal is to achieve the simultaneous reusability between sequential and parallel components, so in data definition part we encapsulate the parallelism in a common generic object which has the same interface in parallel and in serial. Then, parallel objects can be used polymorphically. Components of the computation part will be clients of these objects. We want to allow the code to be the same between the sequential and parallel versions of an application. Thereby every function is implemented once and used either in sequential or in parallel. Additionally, the maintainability of the library implemented according to this model would be simplified using this approach.

3.1 Library integration in scientific workflow environment

A scientific workflow environment describes an application along three aspects: a set of services, the control flow and the data flow of the application. Based on these informations, it orchestrates the execution of the application. A service is a public interface associated to an implementation. The public interface describes how a service interacts with clients. Each service defines a set of input and output parameters also known as communication channels. A service can be stateless or not depending on the underlying middleware capabilities in that respect. The control flow consists in describing the order of execution of the services involved in the application. It does not contain the computation code, only the order of computation. It is a coarse grained description of the application where computation is handled by services as defined above. The data flow consists in describing the exchange of data between the services. Some workflow environment mixes the data flow and control flow together. Data migration from data repositories are managed transparently by the workflow environment. To provide a solution independently from the underlying middleware, that services have to be supposed stateless.

We target a scientific workflow environment which model is defined by three main layers: a layer to interact with end users. A second layer which includes workflow manager and an integrator of services such as databases and computation codes. Finally, there is a layer to interact with middleware. In the environments, based on this model, the user can make use of large scale distributed architectures transparently and independently from the deployed middleware. The computation and data components in the model are represented by some services. The communication between these services would be done by the maddeware. Note that the strict separation between computation aspect, data definition and communication actions required by our library design model matches easily with environments realized according to aforementioned model.

YML is a scientific workflow environment based on the above model [10]. It permits to represent computation and data definitions of our library design model by the corresponding YML components. Besides, it confides the communication actions of our model to the middelware. The activities graph which defines a solver will then be described by YML workflow langage. As a consequence, the solvers are independent from the communication mechanisms used by the middelware (MPI or others). In order to achieve our objective of simultaneous serial and parallel code reusability, we integrate the computation and data components of LAKe library in YML. This solution allows to exploit the multi level parallelism of hybrid methods on parallel and distributed systems.

4 Experiments

For our experiments, we selected two matrices from the MatrixMarket collection. The used matrices are summarized in the table 1. *NNZ* corresponds to the number of non zero elements of the matrix, we also added the Froebenius norm which impacts on the convergence criterion. In order to validate our approach, we

Matrix name	Size	NNZ	Froebenius norm
pde490000	490000	2447200	10^{+3}
pde1000000	1000000	4996000	10^{+3}

Table 1. List of matrices used for experiments

evaluated YML/LAKe on the Grid'5000 platform. Grid'5000 is a French national testbed dedicated to large scale distributed system experiments. We make use of computing resources of the Grid Explorer cluster of Grid'5000. We demonstrate the validity of our approach (a) by presenting the feasibility, (b) by decreasing the number of iterations needed to converge when the number of co-methods increases and (c) by showing the scalability of the solution in regards of the matrix sizes and of the number of co-methods used to solve a large eigenproblem with MERAM (multiply explicitly restarted Arnoldi method).

MERAM is a hybrid method composed of several instances of the same iterative method ERAM. In other words, this method is based on a set of p instances of ERAM. The latter is an iterative method which computes a few eigenvalues and eigenvectors of a large sparse non-Hermitian matrix. The instances of ERAM work on the same problem but they are initialized with different subspace size $m_i, i \in [1, k]$. MERAM defines a set of parameters, the most significant ones are the matrix A, n the size of this matrix, r the number of desired eigenelements, $m_1, .., m_p$ the subspace sizes of the co-methods ERAM composing MERAM also noted MERAM($m_1, ..., m_p$) and *tol* denoting the tolerance expected for the results. An horizontal line denotes the tolerance or the error allowed on the results. The vertical axis represents the estimated error of the solution obtained at each iterations. The horizontal axis represents the number of iterations. In figure 2, we present two executions of MERAM for the matrix pde490000. The two executions differ in the number of involved co-methods. In the first execution, MERAM(10,30,50) requires 98 iterations to converge while MERAM(10, 20, 30, 50) requires 91 iterations. In other words, the increase in the number of co-methods decreases the iteration count of the hybrid method. We notice that by making use of YML/LAKe we are able to overcome the limitation in the number of co-methods composing a hybrid method. One of the motivation of YML/LAKe is the scalability issue in regards of the number of co-methods composing an hybrid one and the size of the problems to be solved. Figure 3 illustrates the progresses made in that regards. Using YML/LAKe we have been able to solve eigenproblems with one million-order matrices. Our approach is based onto the fragmentation in blocks of the matrix of the problem and its distribution during the projection step of the iterative method. The second scalability issue relates to the number of co-methods used to solve an eigenproblem. Extended LAKe allows to test the hybrid methods composing by only a small number of co-methods (up to 3). Using YML/LAKe we have been able to test effortlessly with ten co-methods and it is possible to increase this number.

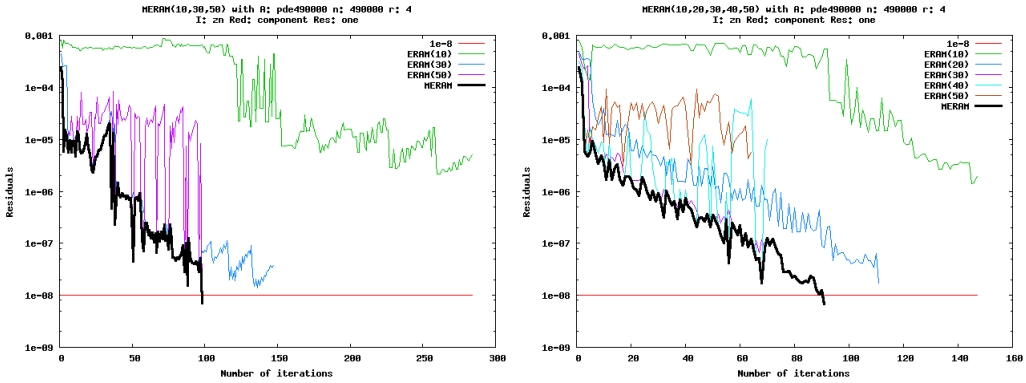

Fig. 2. Convergence of MERAM for matrix 490000 with different number of co-methods

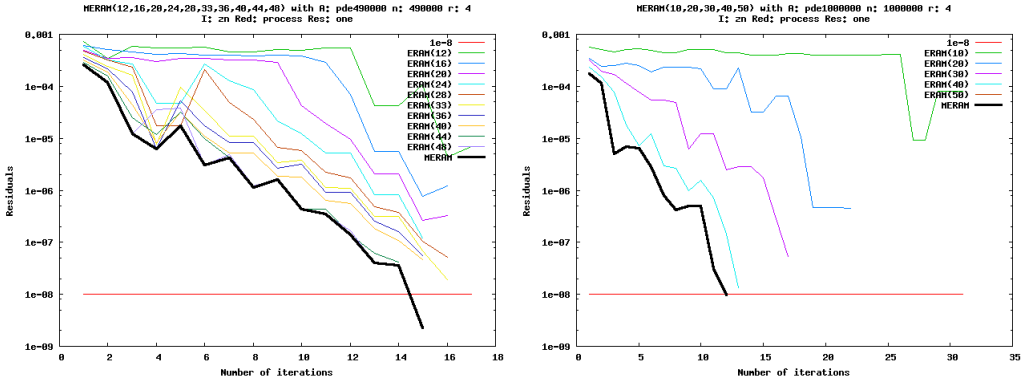

Fig. 3. Scalability of the solution: number of co-methods/size of A

5 Conclusion

Hybrid methods and some of linear algebra applications are well adapted to parallel and distributed systems as well as large scale distributed memory architectures such as GRID and peer to peer systems. Such methods require several levels of parallelism in the same application organized in a tree. Existing numerical libraries are not able to exploit all these levels of parallelism. Their use on distributed systems is still difficult and complex. Their design doesn't allow the simultaneous reusability between the sequential and the parallel versions of an application. Moreover, they do not manage effectively communications in these complex environments; they combine communication with the definition of data and computations. We have presented a model to design reusable numerical libraries for parallel and distributed systems. Our approach is based on three levels of abstraction consisting of the computation, the data definition and the communication actions of applications. We involve in our solution the use of scientific workflow environments. These environments provide tools to orchestrate the execution on a set of distributed services and they allow the use of middleware transparently to end users. To solve simultaneous serial and parallel reuse, we proposed to map our library design model on such a scientific workflow environment. We realized this solution by the integration of LAKe library in YML framework.

We validated our approach with experiments using YML/LAKe. Future works will include the application of our approach to some existing libraries such as PETSc or certain components of Trilinos. This is possible by including some interfaces in the library source code and its integration on a scientific workflow environment such as YML. According to this approach, we can obtain a set of numerical libraries which can cooperate together for the resolution of a problem. Applications developers can use these libraries more easily in the context of large scale distributed systems through the workflow environment.

6 Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see `https://www.grid5000.fr/`)

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
2. Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries.

In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
3. J. J. Dongarra, S. Hammarling, and A. Petitet. Case studies on the development of ScaLAPACK and the NAG numerical PVM library. pages 236–248, 1997.
4. Nahid Emad and Ani Sedrakian. Toward the reusability for iterative linear algebra software in distributed environment. *Parallel Comput.*, 32(3):251–266, 2006.
5. Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.
6. Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.
7. R. Lehoucq, D. Sorensen, and C. Yang. Arpack users' guide: Solution of large scale eigenvalue problems with implicitly restarted arnoldi methods, 1997.
8. K. J. Maschhoff and D. C. Sorensen. P_ARPACK: An efficient portable large scale eigenvalue package for distributed memory parallel architectures. In *PARA*, pages 478–486, 1996.
9. E. Noulard and N. Emad. A key for reusable parallel linear algebra software. *Parallel Computing Journal, Elsevier Science*, 27(10):1299–1319, 2001.
10. N. Emad O. Delannoy and S. Petiton. Workflow global computing with YML. In *The 7th IEEE/ACM International Conference on Grid Computing*, Barcelona, Spain, September 28th-29th 2006.