Design of Interactive Environment for Numerically Intensive Parallel Linear Algebra Calculations*

Piotr Luszczek[1] and Jack Dongarra[1,2]

[1] Innovative Computing Laboratory, Computer Science Department, University of Tennessee Knoxville
[2] Computational Science and Mathematics Division, Oak Ridge National Laboratory

Abstract. We focus our attention in this article on how to provide parallel numerical linear algebra capabilities to Problem Solving Environments. Instead of describing a particular implementation, we present an exploration of the design space and consequences of particular design choices. We also show tests of a prototype implementation of our ideas with emphasis on the performance perceived by the end user.

1 Introduction

Numerical linear algebra may well be regarded as the most basic and thus essential component of problem solving environments (PSE) for numerical calculations. In this article, we intend not to focus on the user tool for accessing the parallel numerical capabilities we propose, but rather, on exploration of the design space available for such PSEs. To the user tool we refer as a *host environment*. The challenge is, we believe, in seamlessly integrating parallel computing capabilities with these environments.

The applicability of our arguments exceeds by far the scope of pure numerical linear algebra on dense matrices. Appropriate design of basic objects and their manipulations invites easy introduction of additional features such as sparse and eigenvalue solvers.

2 Related Work

Exhaustive survey of interactive environments for scientific computing deserves an article on its own. Therefore, we give only references to what we believe are the most relevant efforts that are related to numerical linear algebra. Python is an object-oriented programming language but it invites very much interactive

* This work is partially supported by the DOE LACSI – Subcontract #R71700J-29200099 from Rice University and by the NSF NPACI – P.O. 10181408-002 from University of California Board of Regents via Prime Contract #ASC-96-19020.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 270–277, 2004.

style of development and experimentation [1]. Consequently, there exist numerous libraries that extend Python's numerical capabilities, the most popular include Numeric[2], Numarray[1], SciPy[2], MatPy[3], and ScientificPython[4]. Just for completeness' sake, we should also mention a similar framework for Perl called *The Perl Data Language*[5] with its shell for interactive work called `perldl`. Commonly known environments for interactive numerical calculations are Matlab [3], Octave[6], Scilab [4], Interactive Data Language [5], and Rlab[7]. Also, there exist environments that focus on symbolic manipulations with numerical capabilities, they are surveyed elsewhere [6], here we only mention a few: Mathematica [7], Maple [8], Macsyma [9], and Maxima [10]. Finally, there exist relatively many parallel extensions to Matlab[8] despite some scepticism dating back to 1995 [11]. Out of these extensions, Matlab*P [12,13,14] seems to be the most intensively developed, reaching its third major release version at the time of this writing.

LAPACK for Clusters (LFC) [15] is one of the projects of the Self-Adapting Numerical Software (SANS) framework [16]. It is intended to meet the challenge of developing next generation software by automated management of complex computing environments while delivering to the end user the full power of flexible compositions of the available algorithmic alternatives. LFC, in particular, automates the process of resource discovery and selection, data distribution, and execution of parallel numerical kernels for linear algebra calculations. As such, we believe, it is suitable for the interactive environment we describe in this article.

3 Network Model

We consider primarily a typical two-tier client-server architecture without overloading the server with extra functionality that is left for the client. In such scenario there exists clear separation of capabilities – the server only needs to provide high performance computing capabilities.

Similar reasoning is behind placing the object logic on the client rather than the server (which only holds, presumably large, object data). It simplifies the design of the server and makes it possible to use it on a wider variety of platforms. The client, on the other hand, may leverage existing software technologies for remote management of computational objects.

4 Object-Oriented Features

While designing our system, the first decision to make of is to choose either 0-based (first matrix entry is row 0 and column 0) or 1-based indexing scheme.

[1] `http://www.stsci.edu/resources/software_hardware/numarray/`
[2] `http://www.scipy.org/`
[3] `http://matpy.sourceforge.net/`
[4] `http://starship.python.net/~hinsen/ScientificPython/`
[5] `http://pdl.perl.org/`
[6] `http://www.octave.org/`
[7] `http://rlab.sourceforge.net/`
[8] `http://supertech.lcs.mit.edu/~cly/survey.html`

There exist large amount of code in production use that requires us to implement both. The problem cannot be easily solved by following the convention of the host environment. Such a solution does not allow for code migration between two host environments that use conflicting indexing schemes, therefore, we allow for both.

A related problem is how the end of a range is specified. This may be illustrated with an array declaration (of size `N`) in Fortran: "`REAL A(N)`" and in C: "`float A[N];`". While both declarations use `N` as the upper bound specifier, Fortran uses it inclusively (the allowed indices are `1`, `2`, ..., `N`) and C uses it exclusively (allowed indices are `0`, `1`, ..., `N-1`). Similarly, Matlab uses inclusive convention and Python uses exclusive one. Since there is no single scheme used across different host environments we need to provide for both.

An important decision to make is to decide whether matrix objects should operate with copy or view semantics. The most common situation when this decision has to be made is during submatrix operations. Consider an m by n matrix A partitioned as follows: $A = [A_1\ A_2]$, where A_1 and A_2 are m by n_1 and m by n_2 matrices, respectively, with $n_1 + n_2 = n$. A common way to refer to A_1 is `A[:,:n1]`. The question is whether such a reference should create a copy of the appropriate portion of A or, instead, only produce an alias (a view). There exist situations where either the former or the latter solution is preferable. Different systems solve this problem differently: Matlab and Python use the copy semantics while Fortran 90 uses the view semantics. Most likely, end users will opt for copy semantics, while developers will prefer the view semantics. Therefore, we choose to allow both in our system.

The flexibility of multiple data types comes at the price of resolving issues with mixed-type operations. Automatic variable casting is a feature of almost any programming language in wide spread use. An expression like `x + y` is handled correctly even if `x` and `y` are variables of different numerical type. The most common behavior in such a case is to *promote* (type-cast to the larger type) one of the values and then perform calculations. The promotion rule works well for statically typed languages but most PSEs use some form of dynamic typing and therefore it is harder to ensure correct type for the result. The two major issues to consider are the memory allocation (promotion could potentially require a few times more space to be used) and tensor-rank change (an outer product of two vectors produces a matrix: $A = xx^T$ – a different data type all together). Various solutions may be more appropriate in different some situations. Hence we opt for providing means for ensuring appropriate kind of automatic casting.

The type of PSE that we are describing, deals with tensors of different ranks: 0 – numerical values, 1 – vectors, and 2 – matrices. Such environments add a unique aspect to the type-casting problem described above: reduction of tensor rank. Consider a general case of matrix-matrix multiply: $C = AB$, where: A is m by k, B is k by n, and C is m by n. If either m or n is 1 then the multiply reduces the tensor rank by 1. If k is 1 then the reduction is by 2. However, the type of the result cannot be changed even if potential tensor rank reduction occurs: if a matrix algorithm (such as an iterative method or a dense linear solver) is formulated in terms of submatrices (so called *block algorithm*) then it is expected to work even if the submatrices degenerate to single values (*block*

size is 1). There is no general way of detecting when type change should follow a tensor rank reduction. Therefore, we choose not to perform the type change by default (with type change being optional) since this facilitates interactive work.

5 Host Environment Integration

Networking capabilities are the most essential for our system. Out of the host environments that we initially target, networking is fully supported in Python. Maple, Mathematica, Matlab, and Octave require an extension written in a native language – this creates a portability problem. Luckily, most of the aforementioned environments support Java so this is the way to write just one code and use it in many environments. Finally, since Octave does not support Java as of this writing, an extension can be written using system calls such as `system()`.

Support of name spaces is an important but not essential feature that we would like to use. Python offers more sophisticated way of dealing with this problem – it has a hierarchical module system comparable to that of ISO C++ and Java. Matlab comes close to it by implementing functions only relevant in the context of one particular class of objects (they are commonly referred to as object methods but in Matlab have invocation syntax just like regular functions). Mathematica implements contexts and packages to cope with name resolution. For all other environments we need to use the prefixing technique.

Object-orientation is an important feature as it allows, among others, for a simple statement like `a+b` to be interpreted differently depending on what `a` and `b` are. Most of the host environments that we know are only object-based. Matlab is somewhat more advanced as it allows for creation of new objects and operator overloading but does not have object destructors. This is an important capability in the presence of overloaded operators since they tend to produce anonymous temporary objects which cannot be reclaimed even manually. This problem can be somewhat alleviated by using Java from within Matlab. Python is an object-oriented language which makes it suitable for our system. In other environments we need to resort to function syntax – it takes a lot from expressiveness but still allows to use the functionality that we offer.

6 Parallel Execution

The first issue to resolve in parallel processing is the fact that vectors and matrices most often have different requirements for data layout: vector computations are likely to benefit from 1D (one dimensional) layout, while for matrices, 2D distribution is preferable. One way to automate the decision process for novice users is to be distributing vectors in 1D fashion and matrices in 2D. In a case when a matrix and vector are to be used together, the vector needs to be made conformant to the matrix' layout to perform the operation efficiently. Such a solution involves relatively small communication penalty. For more advanced users, full control of data distribution is the preferable way.

Another aspect is execution synchronization between the client and the server. The term *lazy evaluation* is used to refer to one of the possible scenarios [17].

Simply put, it means that only every other remote request is blocking the client until the server's completion. Generalization of this way of communication is referred to as asynchronous mode. Such a mode, in our opinion, is not good for an interactive environment since it splits the call process into two phases: submission and completion requests. It is not the way existing sequential environments operate – their behavior is equivalent to a synchronous mode (each request is blocked on the client side until the server fulfills the request). A midway solution is transactional processing: the user starts a transaction, then all the computational requests are submitted, and then the call finalizing the transaction is made which blocks until all the computational requests are served. It allows the server to order the computational steps for better performance.

7 Miscellaneous Issues

An important aspect of any numerical system is compliance with the IEEE 794 standard [18]. While the standard is commonly accepted by many hardware vendors, it is still rare to find fully compliant product. We are bound here by what is the typical behavior of the host environment and what is available on the server. Some environments have a way of dealing with non-conformant hardware or system libraries, e.g. in Python, floating-point exceptions are caught by a Unix signal handler.

There exist a few options for data storage and transfer that we consider useful. Certainly, users will have some data sets stored locally on their client machines. These local data need to be transferred to the server for manipulation. During calculation, the best place for data would be the server while at the end, the results need to be transfered back to the client (in case the server does not provide reliable storage capabilities). In the meantime, the data is prone to be lost due to hardware or software crashes so at some point fault-tolerance should be considered. Another scenario is downloading data from an external source. A very helpful extension is support for scientific data formats.

Security is an important asset of a software piece that provides server-like capabilities. In this area, we only intend to leverage existing solutions with initial focus on the port-forwarding feature of `ssh(1)`. It seems relevant in the presence of firewalls and NATs (Network Address Translation) that prevent connections to all but few selected ports.

When it comes to changing the behavior of a computational environment; two main configuration styles need to be considered: global and local. The global type includes: configuration files (*dot-files* in Unix), environment variables, command line options, and global program variables. In a sense, all of them provide a similar functionality with different timing and scoping. However, since a PSE may be regarded as a language, it is important to maintain its semantic consistency. Therefore, global configuration is a valid solution when there is only one default setting mandated as standard and other choices are only optional. Relevant local configuration types include: object attributes, shadow objects or explicit syntax. The first two are somewhat similar as shadow objects are just aliases of their originals with some of the attributes changed. For example if `A` is a square matrix, `A.I` (a shadow object of `A`) could indicate inverse of `A` but

using `A.I` would not immediately produce a numerical inverse of `A` but rather, LU decomposition would be used instead. Compared to object attributes, shadow objects are more explicit. From clarity standpoint, object attributes are not as good as explicit syntax (e.g. function call) but are far more succinct and more suitable for interactive environments.

8 Implementation

At the moment, the basic infrastructure of our design has been implemented and successfully applied to a dense matrix factorization and iterative solution method in Matlab and Python environments. Our preliminary tests show that the overhead of remote execution can be offset when problem sizes become prohibitive for a sequential environment and it is indeed possible to reap the benefits of parallel computation.

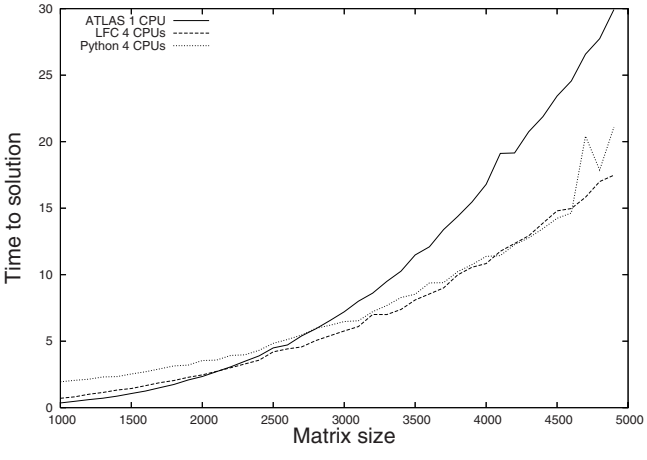

Fig. 1. Comparison of time to solution of a system of linear equations of varying size with different methods

In our tests, we used two dual Xeon 2.4 GHz computers connected with switched Gigabit Ethernet. MPICH 1.2.4 was used as the MPI implementation. Figure 1 shows the timing results for our tests that were performed on a non-dedicated system. The objective was to solve in double precision floating-point arithmetic a system of linear equations by means of LU factorization. Three scenarios were used to obtain a solution: sequential computation, parallel computation, remotely controlled parallel computation.

For the first scenario, ATLAS [19,20] library was used on a single CPU. In particular, the functional equivalent of LAPACK's [21] `DGESV()` routine was used that performs LU decomposition *in-situ*. The second scenario utilized 4 nodes

that performed computations with the LFC's equivalent of ScaLAPACK's [22] `PDGESV()` routine. Again, no data copying was involved. The third scenario used the same hardware and software as the second one but the execution initiation and timing was done on a remote computer running Python interpreter. The round-trip time between the client and one of the nodes of the computational server grid (as measured by the `ping` program) was about 82 milliseconds – a value representing a 16-hop connection (as measured by the `tracepath` program) through wireless access point and an ADSL line. In this scenario, a copy was made of the system matrix to store its LU factors computed by `PDGESV()`: $x = A^{-1}b$ was written as `x = A.I * b` but the inverse of A was not calculated explicitly but rather the LU decomposition of a copy of A was used. It's a trade-off between convenience and optimality (the optimal notation being for example "`pgesv(A, x, b)`") and we intended for our tests to reveal how much this convenience costs.

Figure 1 reveals two important matrix sizes: the size for which parallel execution is faster than sequential execution (3000 in our case) and the size for which the matrix copy overhead is negligible (4000 in our case). The graph shows counter-intuitive effect of copy-free solve being slower than the solve with copy overhead – this is to be expected on a non-dedicated system and is more likely to occur the longer the time to solution is. Worth noting for matrices larger than 4500 is the unexpected increase of time to solution for the remote execution. Very likely explanation is a sudden surge in the load of the network that connects the client and server.

9 Future Work

Our implementation might exhibit itself as an OGSA-compliant service. Such a service would not be running on the server but rather on a proxy capable of OGSA interaction. The proxy would interact with the actual computational server through a simplified protocol – like NetSolve's three-tier approach [23]. A direction to pursue is creation of compilation system so that it is possible to translate existing scripts to a stand-alone executable. Such capability provides opportunity to have a client-server environment for experimentation and debugging while the compiled executable could be used on systems with only batch queue access where setting up a server is not possible.

References

1. Venners, B.: Programming at Python speed: A conversation with Guido van Rossum (2003) Available at `http://www.artima.com/intv/speed.html`.
2. Dubois, P., Hinsen, K., Hugunin, J.: Numerical Python. Computers in Physics **10** (1996)
3. Mathworks Inc.: MATLAB 6 User's Guide. (2001)
4. Gomez, C., ed.: Engineering and Scientific Computing with Scilab. Birkhäuser, Boston (1999)
5. Gumley, L.: Practical IDL Programming. First edn. Morgan Kaufmann Publishers (2001)

6. Schrüfer, E.: EXCALC – a package for calculations in modern differential geometry. In Shirkov, D., Rostovtsev, V., Gerdt, V., eds.: Proc. IV Int. Conf. Comp. Algebra in Physical Research, Dubna, U.S.S.R., World Scientific, Singapore, 1990, 71–80
7. Wolfram, S.: Mathematica: A System for Doing Mathematics by Computer. Addison-Wesley, Reading, Mass. (1988)
8. Char, B., et al.: Maple V, Language Reference Manual. Springer (1991)
9. Rand, R.: Computer algebra in applied mathematics: an introduction to MACSYMA. Number 94 in Research notes in mathematics. Pitman Publishing Ltd., London, UK (1984)
10. de Souza, P., Fateman, R., Moses, J., Yapp, C.: The Maxima book. 2003
11. Moler, C.: Why there isn't parallel Matlab. Mathworks Newsletter (1995).
12. Choy, L., Edelman, A.: MATLAB*P 2.0: A unified parallel MATLAB. Technical report, Massachusetts Institute of Technology (2003)
13. Choy, L.: MATLAB*P 2.0: Interactive supercomputing made practical. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2002)
14. Husbands, P.: Interactive Supercomputing. PhD thesis, Department of Electrical Engineering and Comp. Science, Massachusetts Institute of Technology (1999)
15. Chen, Z., Dongarra, J., Luszczek, P., Roche, K.: Self-adapting software for numerical linear algebra and LAPACK for clusters. Parallel Computing **29** (2003) 1723–1743
16. Dongarra, J., Eijkhout, V.: Self adapting numerical algorithms for next generation applications. International Journal of High Performance Computing Applications **17** (2003) 125–132 ISSN 1094-3420.
17. Norris, B.: An environment for interactive parallel numerical computing. Technical Report UIUCDCS-R-99-2123, University of Illinois, Urbana, Illinois (1999)
18. IEEE 754: Standard for binary floating point arithmetic. Technical report, Institute of Electrical and Electronics Engineers (1985)
19. Whaley, R., Petitet, A., Dongarra, J.: Automated empirical optimizations of software and the ATLAS project. Parallel Computing **27** (2001) 3–35
20. Dongarra, J., Whaley, C.: Automatically tuned linear algebra software (ATLAS). In: Proceedings of SC'98 Conference, IEEE (1998)
21. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK User's Guide. Third edn. Society for Industrial and Applied Mathematics, Philadelphia (1999)
22. Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.: ScaLAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia (1997)
23. Agrawal, S., Dongarra, J., Seymour, K., Vadhiyar, S.: NetSolve: Past, present, and future – a look at a grid enabled server. In Berman, F., Fox, G., Hey, A., eds.: Grid Computing: Making the Global Infrastructure a Reality. Wiley Publisher (2003)