Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

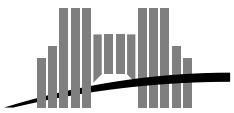
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

SCILAB *to* SCILAB// – *The* OURAGAN *Project*

F. Desprez, M. Quinson, F. Suter
E. Fleury, E. Jeannot
C. Gomez, M. Goursat, S. Steer
S. Chaumette, P. Ramet, J. Roman,
F. Rubi
S. Contassot-Vivier, F. Lombard,
J.M. Nicod, L. Philippe
E. Caron, D. Lazure, G. Utard

Juin 2001

Research Report N° 2001-24



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

INRIA

SCILAB to SCILAB$_{//}$ – The OURAGAN Project

F. Desprez, M. Quinson, F. Suter
E. Fleury, E. Jeannot
C. Gomez, M. Goursat, S. Steer
S. Chaumette, P. Ramet, J. Roman, F. Rubi
S. Contassot-Vivier, F. Lombard, J.M. Nicod, L. Philippe
E. Caron, D. Lazure, G. Utard

Juin 2001

Abstract

In this paper, we present the developments realized in the OURAGAN project around the parallelization of a MATLAB-like tool called SCILAB. These developments use high performance numerical libraries and different approaches based either on the duplication of SCILAB processes or on computational servers. This tool, SCILAB$_{//}$, allows users to perform high level operations on distributed matrices in a metacomputing environment. We also present performance results on different architectures.

Keywords: SCILAB$_{//}$, Parallel libraries, Computational servers, CORBA, Data Redistribution.

Résumé

Dans cet article, nous présentons la parallélisation d'un outil de type MATLAB appelé SCILAB qui utilise des bibliothèques à hautes performances et différentes approches basées soit sur la duplication de processus SCILAB ou l'utilisation de serveurs de calcul. Cet outil permet d'effectuer des opérations de haut niveau sur des matrices distribuées dans un environnement de métacomputing. Nous présentons également des résultats de performances sur différentes architectures.

Mots-clés: SCILAB$_{//}$, bibliothèques parallèles, serveurs de calcul, CORBA, redistribution de données.

SCILAB to SCILAB$_{//}$ – The OURAGAN Project*

E. Caron[†], S. Chaumette[‡], S. Contassot-Vivier[§], F. Desprez[¶], E. Fleury[‖], C. Gomez[**], M. Goursat[**], E. Jeannot[‖†††], D. Lazure[†], F. Lombard[§], J.-M. Nicod[§], L. Philippe[§], M. Quinson[¶], P. Ramet[‡], J. Roman[‡], F. Rubi[‡], S. Steer[**], F. Suter[¶], G. Utard[†]

14th June 2001

1 Introduction

Interactive parallel tools have gained a large interest since the early nineties. Many parallel versions of MATLAB are now available, both in public domain and in the commercial world. SCILAB [21], developed at INRIA in the Métalau project, is a scientific software package for numerical computations in a user-friendly environment. SCILAB is well spread in the scientific community and its popularity has been growing. It is available on several platforms and runs under different types of operating systems (Unix and Unix-like OS, Windows). There are several reasons for its success: (1) the language syntax is simple and easy to learn (MATLAB-like syntax); (2) SCILAB includes hundreds of built-in mathematical functions and provides a large choice of built-in libraries: numerical algorithms, control, linear algebra, signal processing, network analysis and optimization, linear system optimization; (3) it offers a graphical interface; (4) it includes a high level language with a syntax similar to Fortran 90 for matrix notations. Basic matrix manipulations such as concatenation, extraction or transpose are immediately performed as well as basic operations such as addition or multiplication. SCILAB also allows manipulations of high level data structures such as polynomials, rational numbers, sparse matrices, multi-variables systems, lists, ... In one or two lines of code, this language can express a computation that requires dozens of lines of C or Fortran; (5) SCILAB can easily be extended with user-developed modules; (6) SCILAB can easily be interfaced with other languages like C, Fortran or even Maple and Mupad; (7) SCILAB can generate Fortran programs; (8) and last but not least, SCILAB is a public domain software.

One possible drawback of using a sophisticated interpreter is that such a language can not give performance as good as classical compiled languages. However, the performance loss (between 1 and 10 times) should be opposed to the ease of development. All the advantages of tools like MATLAB can be found in SCILAB. It is fairly easy to modify the code, change the size of data,

*This work has been supported in part by the ARC INRIA OURAGAN. URL: `http://www.ens-lyon.fr/~desprez/OURAGAN/`

[†]Paladin, LaRIA. 5, rue du Moulin Neuf, 80 000 Amiens, France

[‡]Alienor, LaBRI. Domaine Universitaire, 351, cours de la Libération, 33405 Talence cedex, France.

[§]SDRP, LIFC. 16, route de Gray, 25030 Besançon cedex, France.

[¶]ReMaP, LIP-ENS Lyon. INRIA Rhône-Alpes, 46, allée d'Italie, 69364 Lyon cedex 07, France.

[‖]Résédas, LORIA. 615, rue du Jardin Botanique, BP 101, 54602 Villers-Les-Nancy, cedex, France.

[**]Métalau, INRIA Rocquencourt. Domaine de Voluceau, BP 105, 78153 Le Chesnay cedex, France.

[††]Part of this work has been done while the author was in postdoc at LaBRI.

print variables, or modify the problem formulation interactively. The prototyping of code is then enhanced by this important feature. Moreover, for coarse grain applications, the interactive aspect of SCILAB is not a limitation and the interpretation overhead remains negligible.

SCILAB should be considered as a "real" language allowing the development of applications. Problems developed by scientists using SCILAB have long execution times and a medium or coarse grain computation. Nowadays, many scientists tend to use a great variety of distributed computing resources such as massively parallel machines, clusters of workstations, SMP machines, and piles of PCs. A SCILAB user who would like to scale his/her application by going to a parallel machine or a network of workstations will not be able to use the SCILAB language and he/she will have to re-program the whole application in C or Fortran. Today's supercomputers still lack of simple user interfaces and access procedures. Parallel computing can then become tremendously tough to use and debug. Moreover, further developments on applications will have to be coded in C or Fortran. Since the investment for researchers or scientists to use the supercomputer facilities in the traditional way is notoriously big, the user has generally to choose between two alternatives: performance (in terms of computational and memory resources) or ease of use.

All these interactive projects use either Matlab duplication or servers. In the OURAGAN project, we aim to offer both approaches. OURAGAN is a join project between several laboratories in France[1] which objective is to bring high performance and memory capacity to SCILAB users. This is a real challenge because we would like to hide as much as possible the use of parallelism to the user.

In fact, we target three different kinds of users. The first one is a parallel computing guru. He/she knows how to write parallel programs using message passing or parallel libraries. He/she wants to keep track of the way data and computations are distributed. For this user, we just provide interfaces to the communication and computation libraries. The second kind of user is a scientist. "*Parallel comput-what? No way! I just need a 45 Gflops workstation with 30 GBytes of memory. Could you provide me with such PC?*". For this kind, everything is hidden in SCILAB. The tool decides itself whether or not it should (re)distribute the data, start new processes, and so on. This is done in SCILAB by operator overloading. Finally, an intermediate level is provided. This type of user wants to have a transparent access to the libraries as much as possible but is also concerned by performance. He/she has a good knowledge of parallel computing and would like to program his/her applications using message passing and computation libraries, but in a more transparent way.

Figure 1 shows the overall architecture of SCILAB$_{//}$. Section 2 presents our first approach which consists in duplicating SCILAB processes on different processors. Then, these processes can exchange messages using either PVM or MPI *(Fig 1-A)*. As we target linear algebra operations, we provide interfaces to parallel libraries like ScaLAPACK [5] and its out-of-core prototype *(Fig 1-B)*. Then, in Section 3, we detail our second approach, which uses computational servers. After presenting the interface between SCILAB and NETSOLVE [10], we deal with our developments to enhance NETSOLVE concerning data persistence, resource location, performance evaluation and communication layers *(Fig 1-C)*. Then, we present two target servers we interfaced with this system: PASTIX *(Fig 1-D)*, which is a parallel direct solver for sparse systems and VISIT *(Fig 1-E)*, a visualization tool for distributed data. Finally, we give a conclusion and present our future work in Section 4.

The idea of providing an access to parallel computing to MATLAB is not new. The first approach consists in compiling MATLAB scripts to an other language (like Fortran [27, 36] or C [19]) and then to apply classical optimizations for its parallelization and use high performance li-

[1] Supported by INRIA.

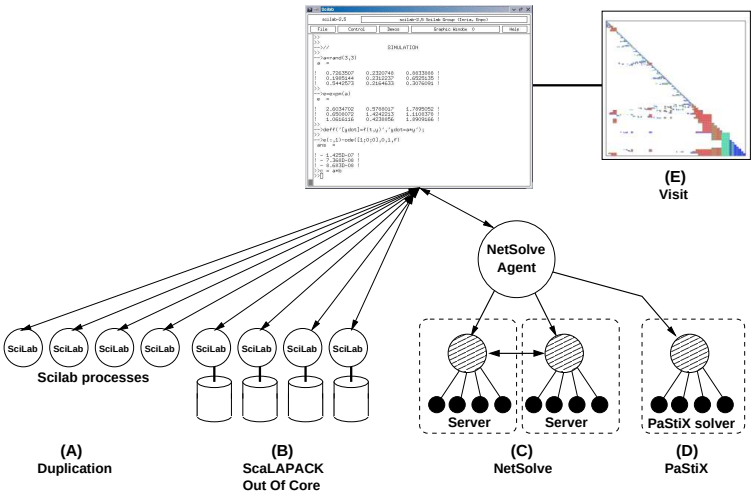


Figure 1: The different approaches and developments around SCILAB$_{//}$ in the OURAGAN project.

braries [13, 19, 35]. The advantages of this approach are its high performance and the use of sophisticated compilation methods. However, interactivity is lost and type inference is a tough problem. The second approach keeps MATLAB interactive and provides parallel extensions. There are also two main approaches for the interactive version of high-performance MATLAB tools. The first idea is to duplicate the tool itself (or a part of it) on every node of the target machine [34, 38]. This approach has one advantage: the "master" process just sends regular commands to the "workers" which in turn interpret commands and execute them before sending the result back. Its main drawbacks are of course performance loss during command interpretations, heavy weight processes, and the need of interfacing every library that need to be added to the tool. The second approach is to rely on a parallel library server that waits for commands [11, 25, 31]. Another approach uses mixed compilation and run-time techniques. MATLAB scripts are compiled into an intermediate language which is executed on a virtual machine. This Matlab Virtual Machine from the Match project provides a high performance runtime environment [4].

2 SCILAB Processes Duplication

The first approach of our Scilab parallelization allows the user to start other remote SCILAB sessions from the SCILAB window, make them communicate and use parallel numerical libraries.

2.1 Message Passing Interfaces

To be able to use SCILAB as a tool for parallel computing, the first step was to provide a "regular" message passing interface for the user. This was done by including the standard PVM [20] interface within SCILAB. This interface allows users to develop parallel programs and benefit from all the main features of SCILAB that simplify numerical computing (as they were listed in introduction). We choose PVM to implement the first message passing interface since it allows to dynamically spawn new processes which is not the case with MPI [37]. Nevertheless, we also added a message passing interface based on MPI. Using MPI implies that the user has to decide at the beginning of its SCILAB session the maximum number of processes he/she will use.

These "regular" message passing interfaces provide low level functions to get the best performance, but are reserved to "expert users". A SCILAB$_{//}$ instance is able to communicate and interact with other SCILAB$_{//}$ instances and the user can send data of any types (including matrices, lists, functions, ...) using PVM (or MPI) commands. These first low-level interfaces provide a tool to easily run parallel algorithms without loosing the power and ease of SCILAB. Indeed, one SCILAB$_{//}$ instance may send any kind of sub-matrix of a matrix A (not only consecutive blocks) with the following instruction `pvm_send(dest,A(1:2:N,:),tag)` or may define a function f and send it to another instance that will be able to execute it on its own data: `deff('[x]=f(y)','x = 1/y')`, `pvm_send(dest,f,tag)`. Among other performance results, [14] shows that (1) when the user sends a full matrix, performances obtained by SCILAB$_{//}$ are as good as a program written in C, and the interpretation of the call does not deteriorate the performance; (2) an overhead is introduced by sending sub-matrices, due to memory copies that take place in both sender and receiver SCILAB$_{//}$ processes. Indeed, an expression like `send(A(1:2:100,2:2:100),...)` send a 50×50 matrix which is not contiguous in memory, so the `send()` routine must copy all elements in a contiguous buffer before sending the data.

2.2 Parallel Libraries Interfaces

2.2.1 Parallel Linear Algebra Package

In order to keep a good portability, interoperability and efficiency, SCILAB$_{//}$ also integrates interfaces to parallel linear algebra libraries like PBLAS, SCALAPACK and the BLACS communication library. Thus, the user may distribute his/her matrices and run parallel routines in order to achieve good performance. As we said above, this level remains dedicated to "expert" users that have good parallel computing skills and that are familiar with the design of the SCALAPACK interface. SCILAB$_{//}$ simplifies calls by enabling default arguments, calling automatically the corresponding complex or double functions by checking the type of parameters, ... The following SCILAB$_{//}$ script illustrates the use of the BLACS routines and gives an example of the function `pblas_gemm` that compute $C = \alpha A * B + \beta C$. This simple example is executed by all SCILAB$_{//}$ processes in a SPMD manner. The following script begins by initializing a 2×2 grid of processors. Once the initialization of each local part of the distributed matrices A, B and C is done, the call to the parallel routine `pblas_gemm` can be executed. The SCILAB$_{//}$ API is very similar to the Fortran one. Note that some parameters are omitted and are set to default values (transposition of matrices, row and column index of the sub-matrix to operate, ...).

```
[mypnum,nprocs] = blacs_pinfo();
blacs_setup(4);
icontxt = blacs_get()
ictxt = blacs_gridinit(icontxt,'R',2,2);
M=1000;K=1000;N=1000;MB = M/2; NB=K/2;
desc_A = sca_descinit(ictxt,M,K,MB,NB,0,0,M);
desc_B = sca_descinit(ictxt,K,N,MB,NB,0,0,K);
desc_C = sca_descinit(ictxt,M,N,MB,NB,0,0,K);
A = rand(M/2,K/2)
B = rand(K/2,N/2)
C = zeros(M/2,N/2)
pblas_gemm(M,N,K,1,"A",desc_A,"B",desc_B,0,"C",desc_C)
```

Nevertheless, some numerical applications are limited by the physical memory size. To break this limit, out-of-core techniques may be employed like using disks as an extension of the main

memory. So we add an interface to the SCALAPACK out-of-core prototype.

2.2.2 Out-of-Core Extensions

The benefit of this method is to handle huge matrices on a cheap hardware. As we are developing an out-of-core extension of SCILAB$_{//}$, users can work on matrices that do not fit in memory. Depending on the data size, SCILAB$_{//}$ would spawn automatically either the in-core program or the corresponding out-of-core program, without script modification.

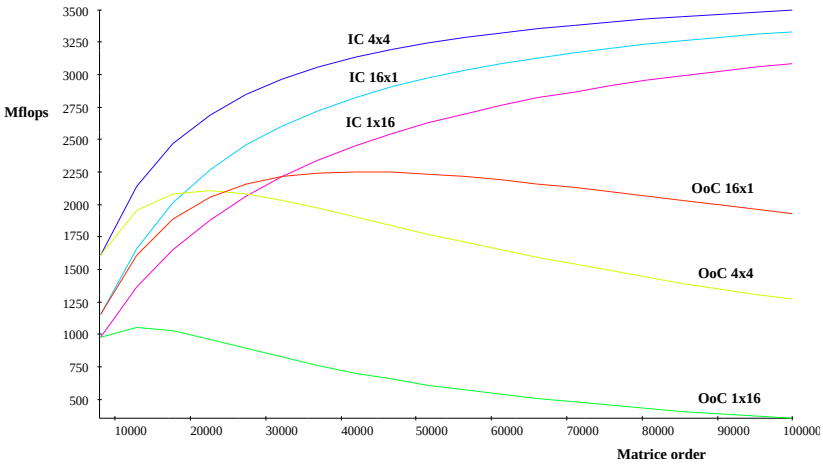


Figure 2: Theoretical performance of the LU factorization.

SCALAPACK provides some out-of-core functions [18] which we interfaced with SCILAB$_{//}$ by adding a new data type to take matrix distribution over disks into account. The performance of the SCALAPACK out-of-core functions were evaluated before their integration into SCILAB$_{//}$. For instance, Figure 2 shows theoretical results of the out-of-core (OoC) left-right looking algorithm for the LU factorization (on a cluster of 16 Celeron PCs with 96 MB/node) and compares it to theoretical performance of the right-looking in-core (IC) algorithm (with no physical memory limit). Performance are shown for 3 kinds of topologies: one row of 16 processors (1×16), a 4×4 grid, and one column of 16 processors (16×1). This figure outlines the impact of distribution on performance. For out-of-core computations, the best distribution is a column of processors where the communication overhead of the algorithm is avoided. This out-of-core function was also modified to allow the overlap of I/O by computation. Then, the performance is very close to the theoretical performance of the in-core algorithm on a (virtual) machine with no physical memory limit (see [9] for details).

Table 1 gives performance obtained for the out-of-core LU factorization in SCILAB$^{ooc}_{//}$ on an Alpha cluster with 6 processors and 768 MB of memory. The overhead of the interface is negligible. In brief, the startup time relative to the time of the call to the out-of-core function by SCILAB$^{ooc}_{//}$ is just some seconds versus several hours or days to execute it.

Unfortunately, only few out-of-core routines are provided in SCALAPACK. We developed some original functions like the out-of-core identity matrix generator, the out-of-core matrix comparison and the out-of-core matrix inversion [8].

Whereas good performance is achieved when computing bound operations like matrix factorization, the I/O overhead can not be hidden for element-wise operations of SCILAB. Consider

Matrix order	Matrix size	Execution time	Performance
12288	1.2GB	23m 15s	886 Mflops
21504	3.7GB	1h 09m	1601 Mflops
27648	6.1GB	2h 47m	1406 Mflops

Table 1: Performance in Mega Flops (Mflops) of SCILAB$^{ooc}_{//}$ LU factorization.

the following expression: `A=sin(A)+cos(B)+sqrt(A)` where `A` and `B` are out-of-core matrices. During the evaluation process, the `A` matrix is read two times, big temporary matrices are generated and re-read, using large space on disk. The I/O cost is greater than computation cost. A solution to reduce I/O cost and avoid generation of big temporary matrices is to split out-of-core matrices `A` and `B` into small blocks in such a way than the whole expression is evaluated in memory block per block.

2.3 Semi-transparent Use of Parallel Computing Using SCILAB$_{//}$

Even if some specialists want to access an expert level, many SCILAB users do not want to spend time learning parallel programming but their main goal still remains to run their programs which become more and more time and memory consuming. In order to provide efficient parallel linear algebra operations inside the SCILAB console but dealing neither with message passing routines nor specific SCALAPACK routines, we decided to add a new distributed type inside SCILAB. This level of transparency is motivated by the need to bring the benefits of interactive environments to supercomputers while maintaining the efficiency and power of highly optimized parallel computational libraries. From the user point of view, the fact that a scalar matrix is distributed or not, will not influence the way of writing SCILAB programs. The only additional commands are:

`scip_init`: initializes the grid, that is, the number of processors that the user will use for this session. Note that a configuration file can be used to specify the default configuration (number of hosts and name of the computers);

`scip_init_dist`: initiates a specific distribution if the user does not want to use the default one;

`scip_distribute`: is the main routine that will distribute a scalar matrix, defined into the SCILAB console or stored on a file system, on the other SCILAB$_{//}$ processes that were previously started by the `scip_init` function.

We overloaded common SCILAB functions and operations so that they work with the distributed type. Thus, operations on distributed matrices will be executed in parallel. At the moment, all operations that have their counterpart in the two parallel linear algebra libraries, PBLAS and SCALAPACK, are overloaded. The main point is that the user is still able to use the SCILAB classical matrix notations and operations to write parallel programs.

Of course, all SCILAB functions working on scalar types are not overloaded. When an operation is intended on a distributed type whereas there is no parallel function corresponding to it, the user may choose between several modes: the first (and simplest) one, is to generate an error; the second one, is to systematically gather the distributed matrix inside the SCILAB console (if it fits in SCILAB memory) and execute the operation in it. The other case is when an operation involves both scalar and distributed data. Once again, the user may choose between several modes: to

generate an error; to gather the distributed matrix or to propagate the distribution. The last choice enables to only distribute the main matrix once and then, the interpretor will automatically propagate the distribution used to the other data involved in further parallel operations. The following example shows how to perform a very simple matrix multiplication on a $P \times Q$ grid of SCILAB$_{//}$ processes. In this example, the distribution used is a bidimensional block-cyclic one with a block size fixed to $MB \times NB$ but the user may have used default values. Then, scalar matrices A and B are distributed and the matrix multiplication is done by using the regular $*$ symbol inside the SCILAB console.

```
CTXT = scip_init(P,Q);
DIST = scip_init_dist("CC",0,0,CTXT(3),MB,NB);
A = rand(M,N); B = rand(M,N);
MatA = scip_distribute("A", DIST);
MatB = scip_distribute("B", DIST);
Res = MatA(1:1000,1:500)*MatB(1:500,1:1000);
size(Res)
ans  = !   1000.    1000. !
```

Table 2 lists functions that support overloading for distributed matrices. As we notice on the example above, setting and retrieving array sections (using only consecutive blocs) also work transparently for distributed matrices. The overloading of major element-wise functions like `cos`, `sin` is done and easy to implement.

Funct.	**Description**	**Funct.**	**Description**
$+, -, *, .*, ./$	Classical matrix binary op.	`size`	Size of an object
`chol`	Cholesky decomposition	`hess`	Hessenberg form
`inv`	Matrix inverse	`linsolve`	Linear eqn. solver
`lu`	LU factor of a Gauss. elim.	`qr`	QR decomp.
`rcond`	Inverse condition number	`schur`	Schur decomp.
`spec`	Eigenvalues	`svd`	Sing. value decomp.

Table 2: Overloaded distributed matrix operations in SCILAB$_{//}$.

Figure 3 plots performance obtained using the standard $*$ operator inside the SCILAB console on distributed matrices. The x-axis represents the size of the matrix and the y-axis the time to execute 2 matrix multiplications since we perform `Res=A*B*C` where A, B and C are square matrices. These tests were performed on a SGI Origin2000. The curve plotted with crosses $(- + -)$ is obtained by doing the computation on a single node, i.e., by using the sequential scalar operator $*$ in SCILAB. The curve plotted with $(- \times -)$ is obtained by using 4 processors and finally, the curve plotted with stars $(- * -)$ is obtained by running the test on 16 processors. The main important point is that it is possible to obtain a very good speedup on such a simple operation that appears many times in SCILAB scripts. Thus, it is worthwhile to use the distributed scalar type when dealing with matrix of size greater than 200×200. The overhead introduced by the distributed type, i.e., sending the instructions to the set of slaves, is not really a problem when computation complexity (and memory capacity) becomes the real burden.

3 Network-Enabled Servers

In the previous section, we have presented our first approach to parallelize SCILAB. But this approach is reserved to expert users. In this section, we present a more transparent way to access parallel resources from SCILAB, using computational servers.

Due to the progress in networking, computing intensive problems in several areas can now be solved using networked scientific computing. In the same way that World Wide Web has changed the way that we think about information, we can easily imagine the types of applications we might construct if we had instantaneous access to a supercomputer from our desktop. The RPC approach [28, 29] is a good candidate to build Network-Enabled Servers (NES) environments on the Grid. Several tools that provide this functionnality exist like NETSOLVE [10], NINF [33], NEOS [32], OVM [7] or RCS [2].

This approach leads us to integrate an interface to NETSOLVE which is a client-agent-servers application that enables users to solve complex scientific problems remotely by accessing hardware and software resources distributed across a network. A load-balancing policy is used by NETSOLVE to ensure good performance by enabling the system to use available computational resources as efficiently as possible. The SCILAB-NETSOLVE interface allows the user to send blocking and non-blocking requests to the NETSOLVE agent which plays the role of a resource broker.

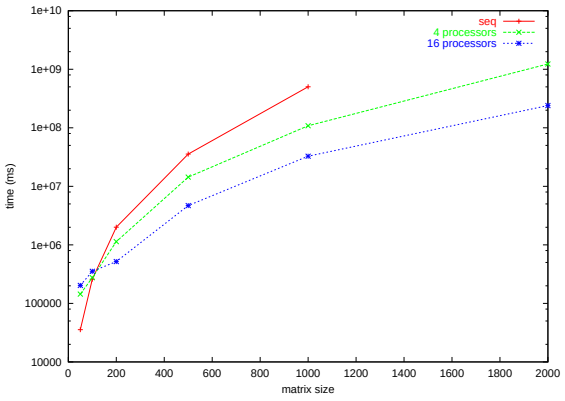


Figure 3: Performance of matrix multiplication using the distributed overloaded operator *.

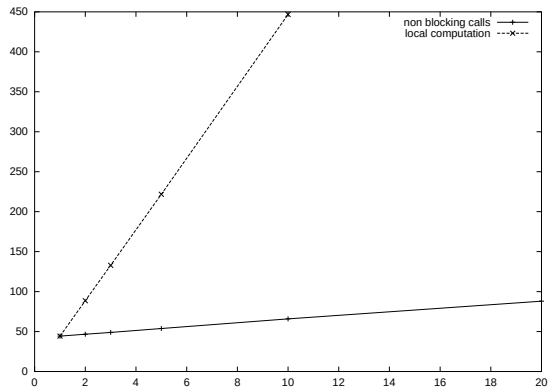


Figure 4: Eigenvalues problems solving using the SCILAB-NETSOLVE interface.

Figure 4 plots performance obtained using non-blocking NETSOLVE routines to solve several eigenvalues problems on a set of matrices. The matrix size was fixed to 500×500. SCILAB was running on a Sparc workstation but one of the server was running on an Origin2000. The x-axis represents the number of calls and the y-axis the time. The curve is linear which in fact corresponds to the time to send and receive the results since the server was able to accept all the calls in parallel. It shows that, from SCILAB, the user is able to obtain high performance almost without having to deal with parallel computing.

3.1 Data Persistence and Data Redistribution

When we interfaced SCILAB and NETSOLVE, we have been confronted by two drawbacks of NETSOLVE concerning *data persistence* and *data redistribution*. When a server has computed a result,

this result may be used again as an input parameter of another request on this server. Hence, it can be useful to use *data persistence*, i.e., cache data on this server. Moreover, this result can also be involved in a computation on another server, in that case it can be useful to *redistribute data* from server to server. However, NETSOLVE does not implement data redistribution between servers: when a server has completed a computation, output objects (results) are retrieved by the client. Therefore, many useless communications could be avoided. This problem as been tackled with the new request sequencing feature [3]. However, the current request sequencing implementation does not allow to handle multiple servers. Moreover, our data persistence implementation allows the client to manage its distributed data and their availability on different servers. We have modified NETSOLVE in order to implement data persistence and redistribution between servers. This has been done in a transparent way, with no change to the API. Existing client programs will work normally after recompiling. Moreover, our implementation is stand-alone: data management works without the help of any other tool. In order to implement this, we modified NETSOLVE servers so that, when a computation completes, data stay locally on the server. The server is waiting for orders from the client. There are five orders a server may receive: *exit* means that the server terminates and all its local data are lost; *send one output object*, the server sends one of its results either to its client or to an other server; *send one input object*, the server sends one of the problem parameter to its client or to an other server; *send all output objects*, all the results are sent to a client or to an other server; *send all input objects*, all the problem parameters are sent to a client or to an other server.

Communications between servers are implemented sockets. When a client wants two servers to exchange data, a socket is established between these servers. We add new functions and data structures to the NETSOLVE client library to allow the use of data persistence and redistribution features. When a client wants to use a remote data, it has only to specify the server session and the object number and type for this session.

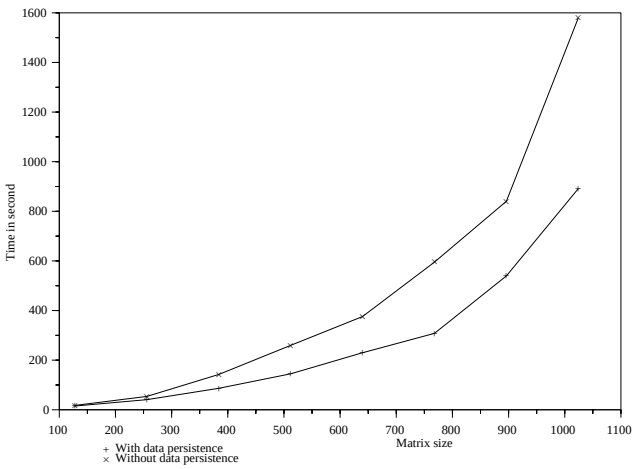


Figure 5: Complex matrix multiplication using NETSOLVE between Nancy and Bordeaux.

Figure 5 shows how a complex matrix multiplication between two distant cities may benefit from data redistribution. In this experiment, we show the time to execute a complex matrix multiplication where computation have been decomposed as follows: (1) $C_{r_1} = A_r \times B_r$; (2) $C_{r_2} = A_i \times B_i$; (3) $C_{i_1} = A_r \times B_i$; (4) $C_{i_2} = A_i \times B_r$; (5) $C_r = C_{r_1} - C_{r_2}$; (6) $C_i = C_{i_1} + C_{i_2}$.

Where A_r (resp. B_r and C_r) is the real part and A_i (resp. B_i and C_i) the imaginary part of complex matrix A (resp. B and C). The four matrix multiplications were computed on one node of the IBM SP2 of the LaBRI in Bordeaux, while the two matrix additions were performed locally in Nancy. One shall remark that steps 1 and 2 can be executed in parallel as well as steps 3 and 4. With data redistribution, objects A_r, B_r, A_i, and B_i are not sent back to the client between steps 1–2 and steps 3–4. We see that in that case computations are 1.77 faster for matrix of size 1024 than the same computation performed without data persistence and redistribution.

3.2 Software Resources Location and Performance Evaluation

To schedule computations over servers, we are facing two problems: first, we have to find which resources are able to satisfy the request, and then, we have to choose the best suited one by evaluating the performance of each proposed solution. To solve the first problem, we are developing a library called SLIM, Scientific Libraries Metaserver. Its goal is to link a problem description to implementations available on servers. In most cases, it is not a one-to-one mapping: a single problem can be solved by many implementations from several libraries, while an other problem may need more than one computational step to be solved. For example, if a user wants to solve a system of linear equations represented by sparse matrix, depending of the data themselves, it can be solved by a direct solver or by a preconditioner followed by a iterative solver. Thus, sequential and parallel implementations may be available. In the first prototype, we decided to use the name of the SCILAB built-in functions as problem description language. Even if this approach is satisfying in this context, it lacks of generality, and we are currently working on a better solution based on the Guide to Available Mathematical Software (GAMS) problem taxonomy [6].

Once SLIM has found which implementations are able to solve the given problem, the system has to evaluate the performance of each one for each machine providing it. The NETSOLVE agent scheduler has some lacks in this domain that we propose to fill. First, it considers that the characteristics of the link (bandwidth and latency) between a client and a server are the same as those of the link between itself and this server. Then, the time complexity of problems must be expressed through a simple expression such as ax^b, where a and b are constants defining the complexity, and x the size of involved data. To improve the knowledge of the metasystem needed by the scheduler to choose the best possible server, we are developing a library called FAST, Fast Agent System Timer [17]. FAST is composed of several layers and relies on low level software, as shown on Figure 6. To address the drawback of the network performance forecast in NETSOLVE, FAST uses the Network Weather Service (NWS) [39]. It is a distributed system that periodically monitors and dynamically forecasts performance of various network and computational resources. Furthermore, the dynamic data acquisition module of FAST enhances NWS. If there is no direct NWS monitoring between two machines, FAST finds the shortest path between them in the graph of monitored links. In this case, the estimated bandwidth is the minimum of those of the path. For the latency, the sum is taken. Concerning the second drawback, FAST includes routines to model the time and space needs of a computation on a given machine as functions of the parameters of the computation. For that, it fits data resulting of benchmarks (realized at installation time with no external load) by linear regression using the least square method. The result is a polynomial function which order is automatically chosen to minimize the error. This allows to take cache and swap effects in account. Furthermore, as the modeled function is more complex, it is more expressive. To store these static data, FAST uses the Lightweight Directory Access Protocol (LDAP) [24]. LDAP was chosen for its optimizations to read and search data. Furthermore, LDAP is widely used in the Grid community. Finally, FAST also includes a user API, which is a set of functions

that combine static and dynamic data acquired from lower level components to produce ready-to-use values. These functions allow the scheduler to get the time to move an amount of data between two computers and the predicted time to solve a problem on a computer taking its actual workload into account.

Figure 6 shows an overview of SLIM, FAST and their interactions with a client application. The scheduling is done in several steps: (1) the client gives the problem description to SLIM. (2) SLIM contacts the database system and searches out the set of implementations which are able to solve the submitted problem. For example, if the problem is a multiplication of dense matrices, the `DGEMM` function of the SCALAPACK library would be a candidate. (3) This set is then sent to FAST to forecast the execution time of each solution for each server. (4) FAST acquires the static data from the database and (5) the dynamic data from NWS. (6) Finally, these data are combined by FAST in a list of couples {implementation i on server s ; estimated time t} and returned to the calling application. Then, the client uses this result to choose which server to contact.

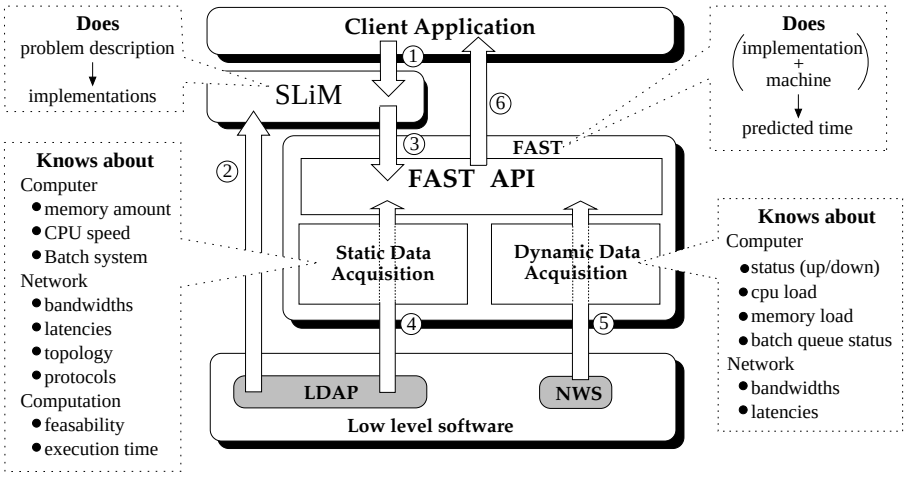


Figure 6: Overview of SLIM and FAST.

3.3 CORBA Interface to Parallel Servers

Communications are a key issue in NES environments. The communication layer should provide both good performance and ease of development. The CORBA norm, defined by the *Object Management Group (OMG)*, aims at providing a standard and transparent interface for the development of object oriented distributed applications over heterogeneous networks. CORBA systems are built around an *Object Request Broker (ORB)*, which is a communication bus between CORBA objects. Communications are initiated by method invocations between objects which can be located on different hosts.

In this section, we give a brief overview of CORBA systems. Then, we discuss the integration and the definition of new services in order to provide metacomputing domain CORBA services. Our current implementation still relies on the NETSOLVE architecture (scheduler, problem description) but it becomes a part of the CORBA services. Thus, we define a new CORBA interface between NETSOLVE components to take advantage of CORBA high level features over the socket interface. We propose a mapping of these two communication layers on a common interface so as to make

them accessible in the same way. The example in section 3.3.3 shows how works a metacomputing session using our platform.

Some goals of the CORBA norm are: to allow a high transparency level in the communication primitives of an application; to normalize the features of object oriented distributed systems; to allow interoperability between these systems (i.e., transparent communications between different CORBA implementations); to provide a distributed programming environment that is independent from the language (i.e., communications between applications written in different languages are transparent); to normalize most common system processes into CORBA services; and to reduce the development time for distributed applications.

3.3.1 CORBA Services for Metacomputing.

As the CORBA norm provides a transparent way to implement distributed application, its use in the domain of metacomputing should be considered. Existing metacomputing platforms are usually subject to very frequent experimental modifications and features add-ons. CORBA systems allow, with a very low communication time increase, a great ease of development and a greater maintainability of the code. Moreover, when communicating between heterogeneous architecture, CORBA can even be faster than XDR [16]. As a matter of fact, our tests [1] have shown that communication times with an ORB are equal to those with standard sockets plus a constant value. Table 3 shows the time necessary to some free ORBs to send various sizes of characters arrays on a local network. Performance of the sockets library in the same conditions are given too. These results confirm that the overhead induced by CORBA is not significant when the data amount grows.

size in bytes	10	100	1000	10000
Mico	0.65 ms	0.73 ms	1.55 ms	9.83 ms
OmniOrb2	0.56 ms	0.68 ms	1.4 ms	9.81 ms
OrbAcus C++	0.57 ms	0.67 ms	1.54 ms	10.26 ms
OrbAcus Java	1 ms	2 ms	6 ms	47 ms
Jonathan	1.06 ms	1.23 ms	3.37 ms	24.3 ms
Sockets	0.26 ms	0.35 ms	1.24 ms	9.67 ms

Table 3: Communication times of some free ORBs compared with the socket library.

Thus, even if slightly better performances could be obtained by writing optimized socket applications, CORBA seems to be a good choice for the development of a metacomputing platform. It is well suited for resource allocation in distributed systems, which is a critical point in metacomputing. We thus propose that CORBA systems are an interesting alternative for the development of a metacomputing platform.

On such platforms, data can be moved across the network without the client being notified of their new location. No common service is still able to perform this kind of function in a metacomputing context. Moreover, the existing services are too complex to be customized to satisfy our specific needs. A set of specific services should be developed for metacomputing. We have specified a set of metacomputing services and propose to interface them with the SCILAB$_{//}$ software. We describe below each of these services and we give an overview of how they could work together to allow a transparent metacomputing process.

Our metacomputing system is composed of three kinds of entities: SCILAB$_{//}$ clients; computational servers; and metacomputing services. Clients are SCILAB$_{//}$ processes embedded in CORBA

objects. Computational servers are numerical libraries with a CORBA frontal which allows remote invocations. Clients never send their computation requests directly to servers. The metacomputing services are in charge to transmit these requests to the appropriate servers. For each computation, a server is chosen to achieve best performances. We define two main metacomputing services, which are implemented as CORBA objects.

The *trading service*, or *trader*, is responsible for the servers pool management. It keeps up to date a table of all the available servers and their features (e.g., what problems they are able to solve, CPU, memory, ...). A client sends its computation request to the trader which chooses a server and transmits the request to it. In order to provide good performance, the trader takes many parameters into account. In the best case, it tends to choose a server which is idle and which already holds a data involved in the computation. If we are not in the ideal case, the trader chooses a server which minimizes the sum of communication and computation time. The scheduler of our metacomputing platform is thus part of our trading service.

The *location service* is responsible for keeping track of the data migrations across the platform. Every data is associated with a unique *identifier* or *reference*. The *location service* keeps up to date a table which associates each data identifier with the server or client owning it. This service should be warned of every data migration. Then, it is able to be called by a client to retrieve a result or by the trader when data locations are needed to choose a server.

3.3.2 Integration into NETSOLVE.

In order to benefit of the existing developments made over NETSOLVE in the OURAGAN project, we choose to integrate our metacomputing services into NETSOLVE. The resulting version of NETSOLVE should be able to use both sockets and CORBA communication layers. For technical reasons, our two services are included in the same CORBA object which is built upon the NETSOLVE agent. The resulting CORBA service is called metacomputing agent as well.

In order to implement our CORBA layer in parallel with the existing socket layer, a *common interface* has been designed. This interface is a set of functions which can be executed by both layers [15]. This interface acts like a wrapper which allows the developers of the core of the agent to use the two communications layers in the same way. This common interface is designed to allow data persistence and further extensions like data duplication. In the next, we give an overview of the resulting architecture and of the interactions between the components of the platform.

3.3.3 Metacomputing Session Example.

Figure 7 shows the architecture of the metacomputing platform in a general way. The purpose of this section is to show how a metacomputing session that uses CORBAworks.

First of all, servers have to be registered within the agent with details about their features (e.g., problems to solve, CPU, memory available, ...). When a client begins a metacomputing session, it has first to look for the reference of the agent upon the system using the CORBA *naming service*. Now, it is able to export data. Exported data have a reference in the *location service* of the agent. Then, when the client submits a problem to the metacomputing platform, it invokes a method of the agent. Parameters of this submission are the problem to solve and the involved exported data. This way, the agent is able to choose a server to perform the submitted problem using both its *trading* and *location services*. The components that own involved data are also invocated to send them to the chosen server. The computation is launched when all data are received. The return of these different methods is a reference to the result. Then the client is able to retrieve this result,

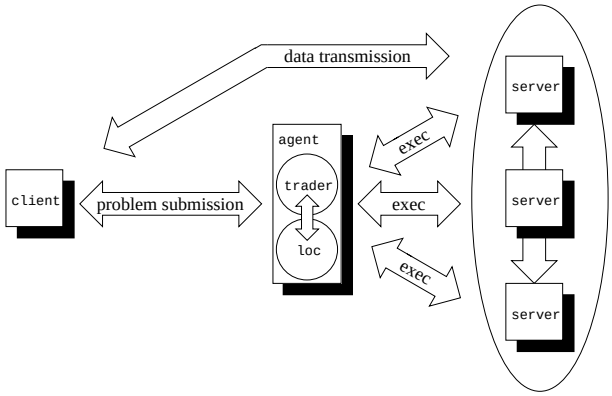


Figure 7: An example of a metacomputing session using the common interface (`c=a*b`).

if necessary, or to submit another problem using it. So, this kind of session allows the system to reduce the transmission data over the network.

This example illustrates the ease of the design of data persistence. In the same way, several extensions can be specified for a more complex data management, e.g., data duplication, lazy copy, ... Moreover, CORBA allows a flexible client/server programming by allowing communications without permanent connection between components of the system.

3.4 Parallel Direct Solver for Sparse Symmetric Positive Definite Systems

As a first target of the computational servers approach in SCILAB$_{//}$, we chose an efficient parallel software processing chain able to solve large linear systems with direct method called PASTIX. This project is developed by the ALiENor team from LaBRI. Solving large sparse symmetric positive definite systems $Ax = b$ of linear equations is a crucial and time-consuming step, arising in many scientific and engineering applications. PASTIX focuses on the block partitioning and scheduling problem for high performance sparse LDL^T factorization without pivoting on parallel/distributed architectures; we consider a parallel supernodal version of sparse LDL^t factorization with total local aggregation. In [22], a first version describing a mapping and scheduling algorithm for 1D distribution of blocks was presented. Then, an original algorithm based on a mixed 1D/2D block distribution has been presented in [23]; it computes an efficient static scheduling that fully drives the block computations of the parallel solver. Parallel experiments were run on an IBM SP2[2], whose nodes are 120 MHz Power2SC thin nodes. These results show that our PASTIX software compares very favorably to PSPASES [26].

We have implemented a PASTIX server for NETSOLVE, and defined an interface to perform globally or separately the whole steps of our parallel software chain. The integration of PASTIX into SCILAB$_{//}$ is now effective and we are currently working to improve data persistence into that server. First experiments on irregular industrial problems are promising. However, they show that the initialization of the coefficients of the matrix is a time consuming step. Indeed, the storage format used by the SCILAB$_{//}$ platform must be more compatible with the PASTIX data structures; we currently use the RSA sparse matrix format as input for the server.

[2]The IBM SP2 of the CINES, located in Montpellier, France.

3.5 Visualization of Distributed Data

Industrial applications mainly use standard data structures such as matrices, but most of the time provide a specific problem-oriented implementation, e.g., Compressed Sparse Column (CSC). Specific implementations are used especially often when dealing with large sparse and irregular data structures, such as matrices coming from the domain of finite elements. The gap between the implementation and the abstract data structure it implements is even bigger when considering parallel applications. Hence, there is a need for tools that make it possible for developers to visualize both their data, their structure, and the operation that are applied to it, whatever their effective implementation and distribution are. These tools must carry the semantics of the application and provide synthesis or filtering mechanisms that make it possible to focus on specific aspects of the problem. Our project was first to define a framework to support the development of such tools, and then to implement a set of software components using this framework. VISIT is one of these tools which is developed at LaBRI. In the OURAGAN project, our goal was to integrate VISIT as a visualization server for SCILAB$_{//}$. In the following, for illustrative purpose, we focus on its integration with the PASTIX computational server.

Our approach consists in providing support for using sparse and irregular data-structures inside applications: we want to provide tools dealing with such data structure at a high level of abstraction. To achieve this goal, we use a model with four levels: (1) the *Implementation Graph* describes the implementation of the data structure in terms of data items and access functions, i.e., the way they are accessed within the application, for instance CSC storage for a matrix; (2) the *Abstraction Graph* describes the abstract data structure, e.g., a matrix; (3) the *Mapping Graph* describes the relationship between the implementation graph and the abstraction graph. In this graph, a node is defined as a pair containing both a node of the abstraction graph and a set of nodes of the implementation graph (an empty set matches a hole in a sparse data structure); (4) the *View Graph*, this level is used for synthesis and filtering of information, e.g., it makes it possible to focus on a column of a matrix.

Based on this model, we have developed libraries to abstract from implementation and distribution of data structures. These libraries allow the manipulation of the data structures at any level contained within our framework, while keeping the same efficiency when working at the mapping graph level or at the implementation level. Using our libraries, high level tools work only with graphs, which they can for instance go through to achieve some operations. It is possible to attach information to any node of any of the graphs. This feature is used by VISIT (see [12] for details). For the visualization of large matrices, VISIT uses the MatView software [30].

The PASTIX code is basically made up of a set of tasks, one for each matrix block. The distribution of these data blocks infers the distribution of the tasks on the processors of the parallel computer. It seems interesting to observe the execution of PASTIX in terms of distributed data blocks. It should be noted that the distribution is irregular and is computed in an initial step. We have described the structure of the data storage used in PASTIX as an implementation graph and a mapping graph. Figure 8 shows the distribution of data blocks, the abstraction graph being the full matrix; each processor has its own grayscale. Figure 9 is a MatView zoom of the lower right corner of the matrix. The data that are used come from the Oilpan of the Harwell-Boeing Sparse Matrix Collection and is of size 75000×75000. The target parallel computer is an IBM SP2 with 16 processors.

We are currently instrumenting PASTIX in order to generate traces in terms of access to data blocks. This will make it possible to show the dependencies between data tasks/blocks in terms of remote read, remote write, local read an local write operations.

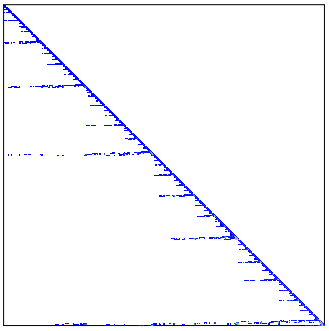


Figure 8: Matrix blocks distribution.

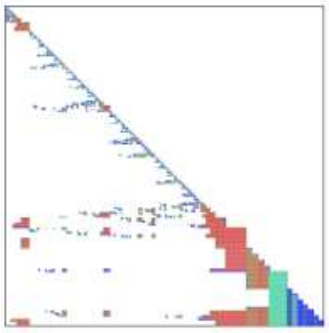


Figure 9: A MatView zoom.

4 Conclusion and future work

Matlab is an interesting approach to Problem Solving Environments. In this paper, we have presented a parallel version of SCILAB, a Matlab-like tool developed at INRIA. Two approaches have been presented. The first one duplicates SCILAB processes on different processors and then uses either message passing with PVM or MPI or high performance numerical routines with ScaLAPACK (in-core and out-of-core and with or without operator overloading). The second one uses an improved version of a high performance Network Enabled Server, NETSOLVE. We added an accurate evaluation of the performance of the metacomputing platform and data persistence on the servers which avoids too many exchanges between the client and the servers and allows redistribution of data between servers. We also presented how we manage two interfaces for the communications between the different components of the tool, i.e., sockets and CORBA.

Concerning the SCILAB processes duplication, the SCILAB interpreter should be able to cast between different data-types when needed. For example, this is the case when computing the product of one (in-core) distributed matrix with one out-of-core matrix. Similarly, the product of two in-core (resp. out-of-core) distributed matrices may lead to one out-of-core (resp. in-core) matrix. Cast can be implemented in two ways. The first way consists in writing computation routines for the different combinations of data-types and call them appropriately. The second way consists in defining new cast operators to promote the lower data-types (e.g., distributed in-core) to bigger ones (e.g., out-of-core) and then do the computation. This solution is easier to implement because there are few cast operators to develop. To achieve a fully functional out-of-core extension of SCILAB, the main job is to develop a lot of out-of-core routines and interface them. Another way consists in defining new operators which allow selection of in-core (distributed or not) blocks of out-of-core matrices. Then, with these new operators, new out-of-core functionalities can be directly written using the built-in programming language of SCILAB. The prototyping and development of new out-of-core functions should be made easier.

Another future work consists in designing a scalable, portable and hierarchical set of agents to improve the Network-Enabled Servers part of our developments. To achieve this goal, we are currently designing DIET, a Distributed Interactive Engeeniering Toolbox [16]. We also would like to add parallel libraries (like ScaLAPACK or PETSc) as computational servers and to handle the redistribution of distributed data between servers on a heterogeneous platform. This is mandatory if we want to be independent of SCILAB, and port our tools on a real grid platform involving different clusters and different networks, as the one connecting several research centers (and their

clusters) from INRIA with a 2.5 Gb/s network[3].

Acknowledgements

We would like to thank all the people that helped us in the OURAGAN project in different ways like D. Goudin, P. Hénon, J.-M. Lepine, F. Pellegrini and N. Viollet.

References

[1] J.-L. Anthoine, P. Chatonnay, D. Laiymani, J.-M. Nicod, and L. Philippe. Parallel Numerical Computing Using Corba. In H. R. Arabnia, editor, *Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume III, pages 1221 – 1228, Las Vegas, july 1998.

[2] P. Arbenz, W. Gander, and J. Moré. The Remote Computational System. *Parallel Computing*, 23(10):1421–1428, 1997.

[3] D. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, *Proc. of the Sixth European Conference on Parallel Computing (Euro-Par 2000)*, volume 1900 of *LNCS*, pages 1213–1222, Munich, Germany, August 2000. Springer Verlag.

[4] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and D. Zaretsky. A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems. In *Proceedings of the Int. Symp. on FPGA Custom Computing Machines (FCCM-2000)*, April 2000.

[5] S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

[6] R.F. Boisvert. The Architecture of an Intelligent Virtual Mathematical Software Repository. *Mathematics and Computers in Simulation*, 36:269–279, 1994.

[7] G. Bosilca, G. Fedak, and F. Cappello. "ovm: Out-of-order execution parallel virtual machine". In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID2001)*, May 2001.

[8] E. Caron, D. Lazure, and G. Utard. Inversion of Huge Matrix on Cluster. In *Cluster 2000. IEEE International Conference on Cluster Computing. Chemnitz, Germany.*, 28 nov - 2 dec 2000.

[9] E. Caron, D. Lazure, and G. Utard. Performance Modeling and Analysis of Parallel Out-of-Core Matrix Factorization. In *HiPC'2000. 7th International Conference on High Performance Computing. Bangalore, India*, December 17-20 2000.

[10] H. Casanova and J. Dongarra. A Network-Enabled Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997. and proceedings of Supercomputing'96, Pittsburgh.

[3] RNRT VTHD project (`http://www.telecom.gouv.fr/rnrt/projets/pres_d74_ap99.htm`).

[11] H. Casanova and J. Dongarra. Providing Uniform Access to Numerical Software. In M. Heath, A. Ranade, and R. Schreiber, editors, *Algorithms for Parallel Processing*, volume 105 of *IMA Volumes in Mathematics and its Applications*, pages 345–355. Springer-Verlag, 1998.

[12] S. Chaumette, F. Rubi, and J.M. Lépine. A Graph Based Framework for the Definition of Tools Dealing with Sparse and Irregular Distributed Data Structures. In IEEE, editor, *Proc. of the 3rd Int. Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 62–70, March 1998.

[13] S. Chauveau and F. Bodin. Menhir: An Environment for High Performance Matlab. In David O'Hallaron, editor, *Languages, Compilers and Run-Time Systems for Scalable Compilers (LCR'98*, volume 1511 of *LNCS*, pages 27–40, Pittsburgh, PA, May 1998. Springer Verlag.

[14] F. Desprez, E. Fleury, C. Gomez, S. Steer, and S. Ubéda. Bringing Metacomputing to Scilab. In *Computer Aided Control System Design (CACSD 99)*, Hawai'i, USA, August 1999. IEEE.

[15] F. Desprez, E. Fleury, E. Jeannot, J.M. Nicod, and F. Suter. Interactive Parallel Computing Using Scilab. *Submitted to Parallel Computing, Special issue on Parallel Matrix Algorithms and Applications*, 2001.

[16] F. Desprez, E. Fleury, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network-Enabled Servers. OURAGAN Whitepaper, available at `http://www.ens-lyon.fr/~desprez/OURAGAN/`, 2001.

[17] F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Forecasting for Network Enabled Servers in a Heterogeneous Environment. In *To appear in the proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*, 2001.

[18] J. Dongarra and E. D'Azevedo. The Design and Implementation of the Parallel Out-of-core ScaLAPACK LU, QR, and Cholesky Factorization Routines. Technical Report UT-CS-97-347, Department of Computer Science, University of Tennessee, January 1997.

[19] P. Drakenberg, P. Jacobson, and B. Kågström. A CONLAB Compiler for a Distributed Memory Multicomputer. In *Proc. of the 6th SIAM Conf. on Parallel Processing for Scientific Computing*, pages 814–821. SIAM Press, March 1993.

[20] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.

[21] C. Gomez, editor. *Engineering and Scientific Computing with Scilab*. Birkhaüser, 1999.

[22] P. Hénon, P. Ramet, and J. Roman. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. In *EuroPAR'99, LNCS 1685*, pages 1059–1067. Springer Verlag, 1999.

[23] P. Hénon, P. Ramet, and J. Roman. PaStiX: A Parallel Sparse Direct Solver Based on a Static Scheduling for Mixed 1D/2D Block Distributions. In *Proceedings of Irregular'2000, LNCS 1800*, pages 519–525. Springer Verlag, May 2000.

[24] I. Howes, M. Smith, and G. Good. *Understanding and deploying LDAP directory services*. Macmillian Technical Publishing, 1999.

[25] P. Husbands, C. Isbell, and A. Edelman. Interactive Supercomputing with MITatlab. `http://www-math.mit.edu/~edelman`.

[26] M. Joshi, G. Karypis, V. Kumar, A. Gupta, and Gustavson F. PSPASES : Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems. Technical report, University of Minnesota and IBM Thomas J. Watson Research Center, May 1999.

[27] A. Malishevsky, N. Seelam, and M.J. Quinn. Translating MATLAB Scripts into Parallel Programs Utilizing ScaLAPACK and Other Parallel Numerical Libraries. *Submitted to IEEE Transactions on Sotfware Engineering*, 1999. Available at `http://www.cs.orst.edu/~quinn/matlab.html`.

[28] S. Matsuoka and H. Casanova. Network-Enabled Server Systems and the Computational Grid. `http://www.eece.unm.edu/~dbader/grid/WhitePapers/GF4-WG3-NES-whitepaper%-draft-000705.pdf`, July 2000. Grid Forum, Advanced Programming Models Working Group whitepaper (draft).

[29] S. Matsuoka, H. Nakada, M. Sato, , and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. `http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf`, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[30] MatView. Scalable Sparse Matrix Viewer. `http://www.epm.ornl.gov/~kohl/MatView/`.

[31] G. Morrow and R. van de Geijn. A Parallel Linear Algebra Server for Matlab-like Environments. In *Supercomputing'98*, Orlando, Nov 1998.

[32] NEOS. `http://www-neos.mcs.anl.gov/`.

[33] NINF. `http://ninf.etl.go.jp/`.

[34] S. Pawletta, A. Westphal, T. Pawletta, W. Drewelow, and P. Duenow. *Distributed and Parallel Application Toolbox (DP Toolbox) for use with MATLAB(r)*. Institute of Automatic Control, University of Rostock and Department of Mechanical Engineering, FH Wismar, Germany, version 1.4 edition, March 1999.

[35] S. Ramaswamy, E.W. Hodges IV, and P. Banerjee. Compiling MATLAB Programs to ScaLAPACK: Exploiting Task and Data Parallelism. In *International Parallel Processing Symposium (IPPS'96)*, pages 613–619, April 1996.

[36] L. De Rose and D. Padua. A MATLAB to Fortran 90 Translator and its Effectiveness. In *Proceedings of the 10th ACM International Conference on Supercomputing - ICS'96*, Philadephia, PA, May 1996.

[37] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996.

[38] A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen. MultiMATLAB: MATLAB on multiple processors. Technical Report 96-239, Cornell Theory Center, 1996.

[39] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6):757–768, Oct. 1999.