Scilab//: User interactive applications and high performances

Frédéric Desprez
INRIA Rhône-Alpes – ReMaP
C102, ZIRST, 655 av. de l'Europe
38330 Montbonnot, France

and

Eric Fleury and Laura Grigori
LORIA, INRIA Lorraine – Résédas
Campus Scientifique - BP 239
54506 Vandœuvre-Lès-Nancy Cedex, France

Abstract

Scilab, developed at INRIA in the Méta2 project is a software similar to Matlab that allows scientific applications to be developed using an interactive environment on a workstation running Unix or Windows. In this paper, we present the current version of Scilab//, a extension of this software that supports the execution of parallel jobs within Scilab. Using simple commands and the same interface as Scilab, Scilab// allows users to start Scilab on a parallel machine or a network of workstations (NOWs), therefore giving access to the computation power and huge aggregate memory sizes. From the Scilab interactive environment, users can spawn several other Scilab processes, executing remote Scilab scripts, which can then communicate between each other. Finally, processes can print/plot their results. There are many research directions around this project and we give an overview of the current and future developments.

Keywords: Scilab, Parallel Machines, NOWs, Meta-computing, PVM, Parallelization Tools, Numerical Applications.

1 Introduction

Scilab [12, 16] 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. Scilab is available on several platforms and runs under different types of operating systems (Unix, Linux, Windows). There are seven main reasons for its success: (1) the language syntax is simple and easy to learn (Matlab-like syntax); (2) Scilab includes hundreds of built-in math functions and provides a large choice of built-in libraries: numerical algorithms, automatic, 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 modules developed by users; (6) Scilab can easily be interfaced with other languages like C, Fortran or even Maple or Mupad [14]; (7) Scilab can generate Fortran programs.

One possible drawback of using a sophisticated interpreter is that such a language can not give performances as good as classical compiled languages. However the performance loss (between 1 and 10 times) should be opposed to the easiness of development. All the advantages of tools like Matlab can be found in Scilab. It is fairly easy to modify the code, changes the size of the data, print variables, or to modify the problem formulation interactively. The prototyping of code is enhanced by this important feature. Moreover, for large grain applications, the interactive aspect of Scilab is not a limitation.

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, network, cluster of workstations even SMP machines and pile 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 necessary 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 a researcher or scientist to use the supercomputer facilities in the traditional way is notoriously big, the user has generally to choose between two alternatives: performances (in terms of computational and memory resources) or the ease of use. In this paper, we describe the current version of Scilab$_{//}$ and give our view of the future of this project which has been specifically designed and conceived to efficiently use such a diverse computational environment and to tackle the problems raised by this new approach to scientific problem solving on a "pool" of computational resources.

The remainder of this paper is organized as follows. Section 2 presents the Scilab$_{//}$ project. Section 3 reviews the current version of Scilab$_{//}$. Section 4 addresses actual and future developments of the Scilab$_{//}$ project. Finally, we present our conclusion in section 5 where we discuss possible future areas of investigation.

2 Aim of the Scilab$_{//}$ project

Obviously, the benefit of the interactivity of Scilab is lost if the problem takes several days to run. With the coming of less expensive parallel machines [3, 15] and the proliferation of parallel computers in general [9], it is natural to wonder about combining the user-friendliness and interactivity of mathematical software packages such as Scilab with the computing power of the parallel machines.

The first aim of the Scilab$_{//}$ project is to offer high performance to Scilab users. Our first target program paradigm is message passing and we choose to first include a PVM interface in the last version of Scilab. We fix ours choice on PVM since we wanted to be able to manage an heterogeneous network of machines and to dynamically spawn processes. More precisely, our first goals are to

1. keep the Scilab environment as-it-is (including its interactive behavior),
2. provide to the user a high level language for the development of parallel applications and to guarantee their portability across platforms, from heterogeneous networks of workstations to parallel machines,
3. offer interesting performances and to allow coarse grain applications to be started from this environment,
4. provide *transparent* interfaces for several numerical libraries (both sequential and parallel).

Using Scilab$_{//}$, the user will keep its Scilab environment but he/she will have access, more or less transparently, to the performances and huge memory sizes of parallel processors and NOWs.

We would like also to have several levels of transparency within the same environment. Users of parallel machines have different programming skills and we can not expect a user to be a parallel programming expert or to prevent him/her to access low level functions to get the best performances. To be able to satisfy almost every kind of user, we would like to provide the following levels of functionalities:

Expert level. The user wants to program his/her parallel machine with message passing and libraries by writing a "regular" PVM or ScaLAPACK code. He/She does not want to bother with matrices allocations and wants to use this tool. But he/she wants to manage everything at the lowest level available. For this kind of user, we just provide interfaces to the communications and computation libraries.

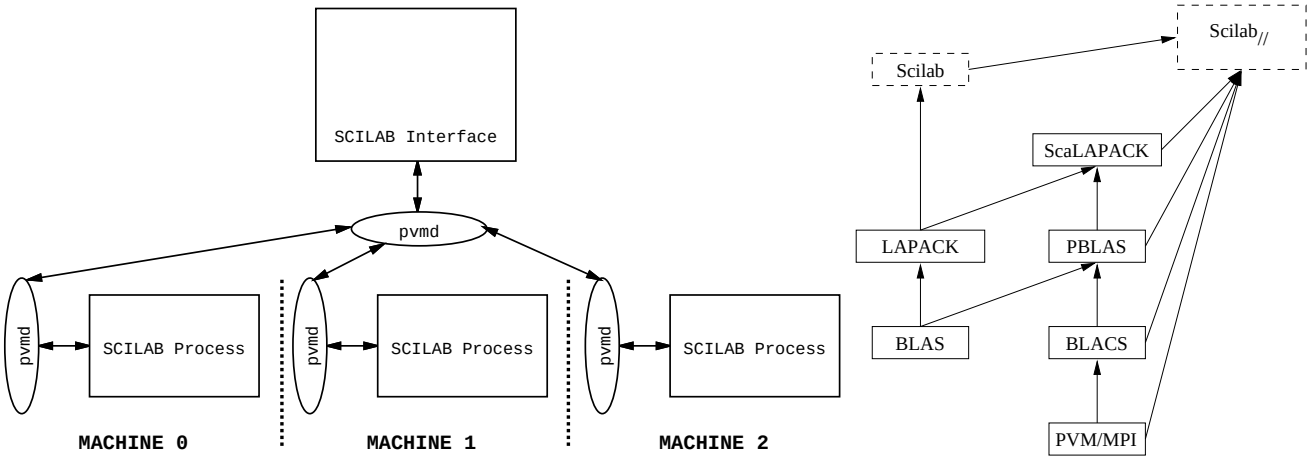
Intermediate level. This kind of user wants to have a transparent access to the libraries as much as possible but he/she is also concerned by the performances of his code. He/she has a good knowledge of parallel computing and would like to program his/her applications using message passing and computation libraries, but in a more transparent way.

Scientist level. *Parallel comput-what? No way! I just need a 45 Gflops workstation with 30 GBytes of memory. Could you provide me with such PC?* Everything is hidden in Scilab. The tool decides itself whether or not it should (re)-distribute the data, start new processes, and so on.

3 Scilab$_{//}$: Today's implementation

3.1 Integration of parallel libraries

In order to keep good portability, interoperability and efficiency, we have chosen to use as much as possible portable libraries like PVM, BLAS, LAPACK and ScaLAPACK. Today's version of Scilab$_{//}$ (Version 1.0) is released in Scilab (Version 2.4) and includes the PVM [11] package (Version 3.3.7). An interface for PVM has been included in Scilab itself. With this interface, a Scilab$_{//}$ instance is able to communicate and interact with other Scilab$_{//}$ instances and the user of Scilab$_{//}$ can send data of any types (including matrices, lists, functions...) using PVM commands. This first low-level interface provides a tool to easily run parallel algorithms without losing the power and ease of Scilab. Figure 1(a) shows a Scilab$_{//}$ session.



(a) Interaction between several `Scilab//` processes.

(b) Hierarchy of the algebra kernel.

Figure 1: Today's `Scilab//` architecture.

The second low-level interface provided by `Scilab//` is an interface to several portable parallel algebra packages, at the moment BLACS, PBLAS and ScaLAPACK. Figure 1(b) illustrates the linear algebra architecture. This allow the user to distribute matrices, unfortunately by hand, and allows him/her to perform calls to parallel functions and to write parallel scripts. Because of the low abstraction degree of all this interfaces, it provides a great flexibility, but in the other hand it gives a relatively poor comfort when using it interactively and for fast prototyping. We will see in section 4 how we will enhance the comfort of `Scilab//`.

3.2 Performances

The goal is not to present here all the speedups of the several linear algebra libraries, but just to compare the overhead introduced by the interface when using message passing between several `Scilab` processes. Figure 2 plots the the performances of the classical "ping-pong" tests. The x-axis represents the message size in bytes and the y-axis the time to execute a round trip. The tests were performed on an "pile of PCs" interconnected by a Myrinet switch. The curve plotted with diamonds ($-\diamond-$) is obtained by sending a sub-matrix (sub-matrix notation, `A[1:n,1:n]`). The curve plotted with squares ($-\square-$) is obtained by using the same PVM subroutines but called from a C-program. Finally, the curve plotted with crosses ($-+-$) is obtained by sending a full matrix, that is when the user gives the name of the variable to send and not a sub-part of it. Two important points: (1) when the user sends a full matrix, performances obtained by `Scilab//` are as good as a C-program, and the interpretation of the call does not deteriorates the performances; (2) the overhead introduced by sending submatrices is due to memory copies that take place in both the sender and receiver `Scilab` processes since an expression like `send(A(1:2:100,2:2:100),...)` will send a 50×50 matrix which is not contiguous in A, so the send routine must copy all the elements in a contiguous buffer before sending the data. The same problem may arise in the receive process since the expression `B(1:6:300,1:6:300) = recv(...)` is a valid one. Note that the second method which consists in sending a matrix by giving its name was retained in the interface between `Scilab//` and the several parallel linear algebra packages for two major reasons: (1) it avoids memory copies and thus it obtains better performances; (2) it keeps the same API as described in [7] and [8] such that the user can refer to several existing technical reports and user guides.

4 Actual developments and future directions

4.1 Network-enabled solvers

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 `Scilab//` project is developing basic software infrastructure for computations that integrate geographically distributed computational and information resources.

This approach leads us to integrate an interface to NetSolve [4] which is a client-server application that enables users to solve complex scientific problems remotely. The system allows users to access both hardware and software computational resources distributed across a network. NetSolve searches for computational resources on a network, chooses the best one available, and using retry for fault-tolerance solves a problem, and returns the answers to the user.

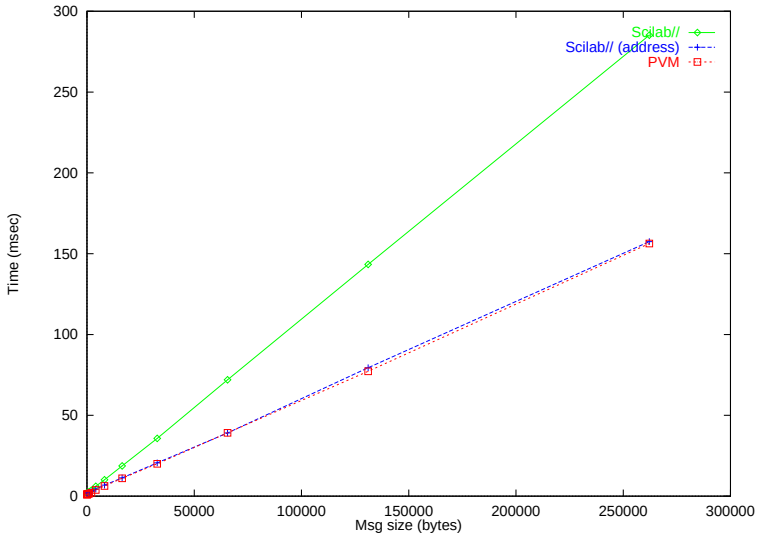


Figure 2: Performances and overhead of the communication in `Scilab`$_{//}$.

A load-balancing policy is used by the NetSolve system to ensure good performance by enabling the system to use the computational resources available 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. When sending a non-blocking request, the user gets back the control immediately and can do some more work in parallel and check for the completion of the request later.

4.2 Full Programming Paradigms

At the moment, `Scilab`$_{//}$ supports only one programming paradigm, called *remote computing paradigm* in [5]. That is, both program (`Scilab` scripts) and data are sent to the nodes of the parallel virtual machine which returns the results after execution of the scripts. We are currently working on a full interactive way to develop parallel application in `Scilab`$_{//}$. To be able to do this, the user choose at startup-up to be in a mode where `Scilab` processes on the machine are waiting for `Scilab` commands (some of them being data-parallel ones, like for example linear algebra operations). Figure 3 presents the overall architecture of the future version of Scilab which handle automatic distribution, load-balancing and fault-tolerance.

4.3 Automatic Distribution and Redistribution

In order to support the previous interactive mode, and to use parallel numerical libraries such as ScaLAPACK, `Scilab`$_{//}$ requires a distribution of the data on the target parallel machine. The distribution has to be carefully chosen in order to get the best performances. The user of `Scilab`$_{//}$ may just want to type $[l, u] = \mathrm{LU}(a)$ in his/her `Scilab` interface instead of dozen of lines to distribute matrices and gather the results. The goal is to hide the data distribution without distributing and gathering matrices before and after each call. To do this, we create a new `Scilab` type `101`, for *distributed real or complex constant matrix*. Thus, in the interactive mode, based on the types of the parameters involve in a expression, we can choose the data distribution on-the-fly, depending of the machines load, the algorithm and the data in place.

Depending of the costs of computation routines and the parameters of the target machine (bandwidth, latency, processor's speed), it is sometimes interesting to redistribute the data between different calls. Again, the use of redistribution points has to be carefully chosen to avoid a decrease of performances. We are working on a project called ALaSca [1]. This project aims at providing heuristics and algorithms for the computation of a set of (re)distributions in case of a graph of ScaLAPACK routines calls. These algorithms will be directly used in `Scilab`$_{//}$.

4.4 More Numerical Libraries

We would like to provide an access to many scientific libraries for the user of `Scilab`$_{//}$. Sparse matrices are widely used in diverse scientific computation fields such as structural and integrated circuit analysis, computational fluid dynamics and, more generally, as the solution of partial differential equations. For example, `Scilab` does not provide many features to handle sparse matrices. Many libraries exist in the public domain such as Aztec, PETSc, SparsKit, P-SparseLIB which implement iterative methods, or SPOOLES, SuperLU, PSPASES which implement direct methods and efforts are done to provide standard interface [10].

Our aim is to provide access to a number of those sparse matrix numerical libraries. Having such an ac-

[1] **URL:** `http://www.ens-lyon.fr/~desprez/FILES/RESEARCH/SOFT/ALAS`

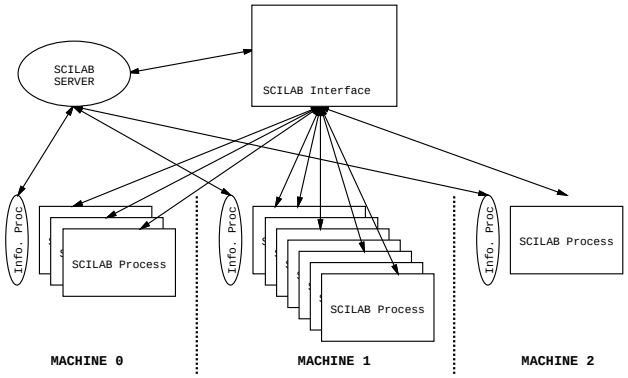


Figure 3: Processes architecture of the LAN version of Scilab//.

cess will help to adapt the full range of Scilab dense matrix operations to the sparse case. At the same time, parallelism will be added to those operations while using the best sparse matrix algorithms implemented recently. An accessibility problem is raised here. We wish to offer access to more libraries at the same time and in this case we will face problems similar to ones encountered in the NetSolve project: the uniformity of access [6].

As a short-term goal, we will concentrate on elementary sparse matrix operations, such as addition and multiplication, and on direct methods for solving sparse linear systems of equations. The direct methods for solving linear systems of equations involving sparse matrices contain four phases: compute fill-reducing ordering, perform symbolic factorization, compute numerical factorization and solve triangular systems of equations.

Disposing of some knowledge of the linear system can be important when trying to obtain the best performance out of a library. For example, there exist different ordering methods (SPOOLES [2] implements them). When the user is knowledgeable of details of his problem, he has the opportunity to choose the best ordering available.

The four phased process previously mentioned can be complicated for a novice user. To help with such a case, we will offer more levels of functionalities. For example, a novice user could employ some so-called "wrapper objects" that we can include in the implementation. Those objects are similar to the ones provided in SPOOLES, which offer a simplified approach of the library. Moreover, we wish to offer a completely transparent interface to the user. This interface consists of normal Scilab operations on matrices. When calling it, the user is hidden from the details of sparse algorithms or parallelism implementation.

In the more distant future, we intend to implement higher-level sparse matrix operations such as iterative methods for linear systems. One interesting feature will be to let the tool choose the library for you. This is a key issue when dealing with sparse matrices. The method chosen for solving a problem depends on the values of the matrix involved. The automatic selection of a method is not trivial. Of course, we can make a choice between several implementations of a given method.

A goal at least as important as the previous ones is to allow a Scilab user to visualise sparse, unstructured, and even extremely large matrices. We intend to do this by integrating a matrix visualization tool, like Emily [13] or showmap in SMMS [1]. These tools offer a several views of a matrix ranging from a single element to the whole matrix. Moreover, they offer access to multiple colormaps in order to extract different features and structures of the same matrix.

4.5 Load Balancing

One critical issue of parallel computing on heterogeneous network of workstations is the load-balancing. A first load-balancing can be used at the start of one execution by taking informations about the state of the parallel virtual machine. At run-time, the load-balancing can be changed depending of the load of the different machines. We can of course make use of load-balancing of software like but a load-balancing strategy tuned for our applications will of course be more efficient.

There are two key issues in this optimization phase. First we must be able to know when load-balancing must occur. Thus we need to know the state of a given machine and network traffic. The second issue is how to balance the load. We see two steps. First, before starting a new computation phase, Scilab// should be able to choose the "perfect" parallel virtual machine and avoiding processors too loaded. Second, during the execution of a phase, we should be able to migrate a task and its data. This is of course not really easy when using a given parallel library.

5 Conclusion

In this paper we have given an overview of the Scilab// project which main goal is to bring high performance to Scilab users and transform a Scilab

session into a meta-computing one, which means, ease of use and access to possibly remote computational resources. Scientists want to spend their time doing science; they are generally not interested fooling with the nuts and bolts of computers and accompanying technologies.

As stated in section 4, several possible extensions of this work are open to investigation. Main open researches concern automatic distribution and redistribution even if a lot of works has been done, especially for the High Performance Fortran language. This problem is well known to be a difficult problem and deserve further study, especially for interactive parallel computing.

References

[1] F. L. Alvarado. Manipulating and Visualization of Sparse Matrices. *ORSA Journal on Computing*, 2:186–207, 1990.

[2] C. Ashcraft and R. Grimes. SPOOLES: An object-oriented sparse matrix library. In *Conference on Parallel Processing for Scientific Computing,*. SIAM, March 1999.

[3] S. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, and C. Packer. BEOWULF: a parallel workstation for scientific computation. In *International Conference on Parallel Processing (ICPP 95)*, 1995.

[4] H. Casanova and J. Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

[5] Henri Casanova and Jack Dongarra. NetSolve: a Network-Enabled Solver; Examples and Users. In John K. Antonio, editor, *Heterogeneous Computing Workshop*, pages 19–28. IEEE Computer Society, March 1998. ISBN 1097-5209.

[6] Henri Casanova and Jack 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.

[7] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and Whaley. R.C. LAPACK Working Note: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performances. Technical Report 95, Computer Science Department, University of Tennessee, 1995.

[8] J. Choi, J. Dongarra, S. Ostrouchov, A.and Walker D. Petitet, and R.C. Whaley. LAPACK Working Note: A Proposal for a Set of Parallel Linear Algebra Subprograms. Technical Report 100, Computer Science Department, University of Tennessee, 1995.

[9] J. Dongarra, H. W. Meuer, H.D. Simon, and E. Strohmaier. Changing technologies of HPC. *Future Generation Computer Systems*, 12(5):461–474, April 1997.

[10] BLAST Forum. *Document for the BLAS Standard (DRAFT)*, chapter 3: Sparse BLAS. 1999.

[11] 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.

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

[13] T. Loos and R. Bramley. Emily: a Visualisation Tool for Large Sparse Matrices. Technical Report TR412a, Computer Science Department Indiana University, 1994.

[14] The MuPAD Group, Benno Fuchssteiner, et al. *MuPAD User's Manual - MuPAD Version 1.2.2*. John Wiley and sons, Chichester, New York, first edition, march 1996. includes a CD for Apple Macintosh and UNIX.

[15] D. Ridge, D. Becker, P. Merkey, and T. Sterling. Beowulf: Harnessing the power of parallelism in a pile-of-PCs. In *Aerospace*. IEEE, 1997.

[16] Scilab Group. Introduction to Scilab: User's guide. Technical report, INRIA - Unité de recherche de Rocquencourt - Projet Méta2, 1997. Also available electronically, the URL is `http://www-rocq.inria.fr/scilab`.