

Parallelization of direct algorithms using multisplitting methods in grid environments

Jacques M. Bahi and Raphaël Couturier

Laboratoire d'Informatique de l'université de Franche-Comté (LIFC), FRE CNRS 2661,
IUT de Belfort-Montbéliard, BP 527, 90016 Belfort, France
{jacques.bahi, raphael.couturier}@iut-bm.univ-fcomte.fr

Abstract

The goal of this paper is to introduce a new approach to the building of efficient distributed linear system solvers. The starting point of the results of this paper lies in the fact that the parallelization of direct algorithms requires frequent synchronizations in order to obtain the solution for a linear problem. In a grid computing environment, communication times are significant and the bandwidth is variable, therefore frequent synchronizations slow down performances. Thus it is desirable to reduce the number of synchronizations in a parallel direct algorithm. Inspired from multisplitting techniques, the method we present consists in solving several linear problems obtained by splitting the original one. Each linear system is solved independently on a cluster by using the direct method. This paper uses the theoretical results of [6] in order to build coarse grained algorithms designed for solving linear systems in the grid computing context.

1. Introduction

Linear systems have to be solved in most scientific applications and a myriad of algorithms have been designed to solve them. These algorithms are classified in two major classes : the class of direct algorithms and the class of iterative ones. The first class consists in finding the solution in one step whereas the latter proceeds by successive approximations. Although these algorithms have been the object of extensive studies for several years, the emergence of new environments such as clusters and grids has motivated new developments. Indeed, new difficulties appear such as poor efficiencies due to the impact of the heterogeneity of networks and clusters and the unpredictability of the dynamic platforms. These new problems appear particularly when we have to solve very large linear systems. The use of direct algorithms in such an environment is a hard chal-

lenge. This is, in part, due to the importance of communications in such algorithms.

In this paper we propose a new approach in order to parallelize direct algorithms. This approach is based on the so-called multisplitting algorithms [6, 13, 15, 19]. To be more precise we use results published by J. Bahi et al. in [6] in order to build coarse grained algorithms designed for solving linear systems in the grid computing context. Even if the convergence results are derived from [6], it should be noticed that multisplitting algorithms were not designed in the aim of parallelizing direct algorithms.

Our method allows us to deal with coarse grained parallelism, and so, to reduce the frequency of synchronizations. In our approach the original linear system is split into several subproblems, then each subproblem is solved independently using the considered direct algorithm. Naturally, due to the dependencies between the subproblems, the solution is not achieved in one step but in successive steps. So the new method consists in building an iterative algorithm over the network of clusters, each one executing the considered direct algorithm. Thus the communications are those associated to a coarse grained iterative algorithm. Indeed, communications are performed only at the end of each iteration. Another important specificity is the possibility to use any sequential direct solver whether it is dense, band or sparse. This method unifies known algorithms such as the classical block Jacobi, the O'Leary and White multisplitting algorithms and the discrete analogous of Schwarz overlapping algorithms.

Experiments on local and distant clusters show that this approach is very efficient and adapted to grid computing environments. Several discussions such as the impact on the efficiencies of factorization times, overlapping sizes, heterogeneity of the processors and the communications in the network are reported.

It should be noted that, although our approach is perfectly designed for the synchronous context, it is also possible via some hypotheses to execute it in asynchronous mode [8, 17]. The communications and iterations are then

not synchronized. In the context of wide area networks made of geographically distant clusters, this gives the programmer an easy way to parallelize any direct algorithm and to avoid the problems due to communication delays between distant clusters.

This work describes multisplitting methods for linear systems (the theoretical model and practical experiments). In [5], we present an application of multisplitting methods for nonlinear systems. This application models the problem of 3D transport of pollutants. The resolution is also computed in a grid environment.

The rest of the paper is organized as follows. Section 2 is dedicated to the description of the algorithm. Section 3 describes the algorithmic model. In Section 4 we give some particular known algorithms derived from our method. Section 5 gives the reader large classes of linear problems for which the results of the paper are valid. In section 6, we report the experimentations and we analyze the information they give, then we conclude with the future applications of this work and its possible developments.

2. Description of the multisplitting-direct algorithm

Several variants can be derived from our approach. In this section we describe a simple one based on multisplitting algorithms without overlapping.

Consider the n dimensional linear system

$$Ax = b \quad (1)$$

We suppose that (1) has an unique solution. Our approach consists in splitting the matrix into horizontal band matrices. Then each cluster or each processor is responsible for the management of a band matrix. For the sake of simplicity, we only consider in the figure below the case where a band matrix is assigned to a processor (see remark 2). With this distribution, a processor knows the offset of its computation. This offset enables us to define the submatrix, noted $ASub$, which a processor is in charge of managing. The part of the band matrix before the submatrix represents the left dependencies, called $DepLeft$, and the part after the submatrix $ASub$ represents the right dependencies, called $DepRight$. Similarly, $XSub$ represents the unknown part to solve and $BSol$ the right hand side involved in the computation. Figure 1 describes the decomposition and the important parts for the computation.

At each step, a processor solves $XSub$ by using the following subsystem:

$$ASub * Xsub = BSub - DepLeft * XLeft - DepRight * XRight$$

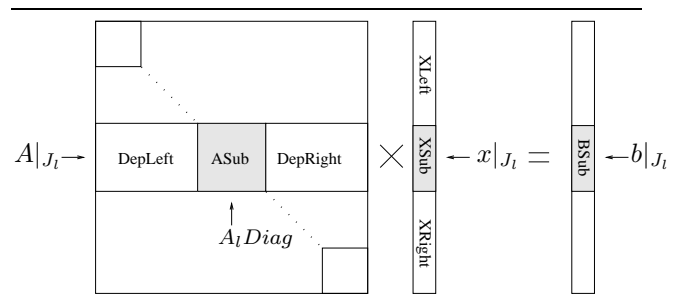


Figure 1. Decomposition of the matrix

Then the solution $Xsub$ must be sent to each processor which depends on it.

2.1. Multisplitting-direct solver algorithms

2.1.1. Description of the algorithms Suppose we have L processors and consider subsets J_l of $\{1, \dots, n\}$ where n is the dimension of the unknown vector, $k \in \{1, \dots, L\}$, such that $\bigcup_{k=1}^L J_l = \{1, \dots, n\}$. The subsets need not to be disjoint. For each $l \in \{1, \dots, L\}$, decompose A as in figure 1. So we have L splittings of A . Then each processor solves a linear subsystem associated to a splitting.

In algorithm 1 we summarize our synchronous and asynchronous multisplitting-direct solvers. The four main steps are described as follows:

1. Initialization

The way the matrix is loaded is free. Either one processor is in charge of it and it distributes the band matrix corresponding to each processor, or each processor itself manages the load of the band matrix (in the algorithm the band matrix corresponds to $DepLeft + ASub + DepRight$). Then until the convergence, each processor iterates on:

2. Computation

At each iteration, each processor computes $BLoc = BSub - DepLeft * XLeft - DepRight * XRight$. Then, it solves $XSub$ using the $DirectSolve(ASub, BLoc)$ function.

3. Data exchange

Each processor sends its dependencies to its neighbors. The receptions are managed directly in the code in the synchronous case and in a separate thread in the asynchronous case. This is why we do not state the reception code in the algorithm [3]. Nonetheless, when a processor receives a part of the solution vector (noted $Xsub$) of one of its neighbors, it should update its part of $XLeft$ or $XRight$ vector according to the rank of the sending processor.

Algorithm 1 multisplitting-direct solver algorithm

Initialize the communication interface
MyRank = Rank of the processor
NbProcs = Number of processors
Size = Size of the matrix
SizeSub = Size of the submatrix
Offset = Offset of the matrix
ASub[SizeSub][SizeSub] = Submatrix
DepLeft[SizeSub][Offset] = Submatrix with left dependencies
DepRight[SizeSub][Size-Offset-SizeSub] = Submatrix with right dependencies
DependsOnMe[NbProcs] = Array with depend processor
BSub[SizeSub] = Array with right hand side of subsystem
XSub[SizeSub] = Array with solution of the subsystem
XLeft[Offset] = Array with left solution of the system
XRight[Size-Offset-SizeSub] = Array with right solution of the system
BLoc = Array with local computation of right hand side
repeat
 BLoc = BSub
 if MyRank!=0 **then**
 BLoc = BLoc-DepLeft*XLeft
 end if
 if MyRank!=NbProcs-1 **then**
 BLoc = BLoc-DepRight*XRight
 end if
 XSub = DirectSolve(ASub,BLoc)
 for i=0 to NbProcs-1 **do**
 if i!=MyRank and DependsOnMe[i] **then**
 Send(XSub,i)
 end if
 end for
 Convergence detection
until Global convergence is achieved

4. Convergence detection

Two methods are possible to detect the convergence either we can use a centralized algorithm described in [2] or a decentralized version that is more general as described in [4].

3. Algorithmic model of the multisplitting-direct algorithm

In the sequel we will denote by $x^l, l \in \{1, \dots, L\}$, a vector of \mathbb{R}^n . To a matrix $C = (C_{ij})_{1 \leq i, j \leq n}$, let us associate the matrix $|C| = (|C_{ij}|)_{1 \leq i, j \leq n}$ and recall that the spectral radius of C is equal to $\max_{1 \leq i \leq n} \{|\lambda_i|\}$, where λ_i is an eigenvalue of C .

The multisplitting-direct algorithm can be described by an extended fixed point mapping defined from $(\mathbb{R}^n)^L$ into

itself. This fixed point mapping is defined as follows, see [6],

$$\begin{cases} T : (\mathbb{R}^n)^L & \longrightarrow & (\mathbb{R}^n)^L \\ X = (x^1, \dots, x^L) & \longrightarrow & Y = (y^1, \dots, y^L), \end{cases} \quad (2)$$

such that for $l \in \{1, \dots, L\}$

$$\begin{cases} y^l = F_l(z^l) \\ z^l = \sum_{k=1}^L E_{lk} x^k, \end{cases} \quad (3)$$

where E_{lk} are weighting matrices satisfying

$$\begin{cases} E_{lk} \text{ are diagonal matrices} \\ E_{lk} \geq 0 \\ \sum_{k=1}^L E_{lk} = I_n \text{ (identity matrix)} \\ \forall l \in \{1, \dots, L\}. \end{cases} \quad (4)$$

In (3), $F_l(z^l) = M_l^{-1} N_l z^l + M_l^{-1} b$ where

$$A = M_l - N_l, \quad l = 1 \dots L \quad (5)$$

is a splitting of A and M_l is the block diagonal matrix defined in Figure 2.

$$M_l = \begin{array}{|c|c|} \hline \text{shaded} & 0 \\ \hline 0 & A_l \text{Diag} \\ \hline \end{array}$$

Figure 2. The matrix M_l

Then it can be shown that if each splitting is convergent i.e. $\rho(M_l^{-1} N_l) < 1$, the fixed point mapping is also convergent to the extended solution of (1) say (x^*, \dots, x^*) and then the synchronous algorithm converges.

To prove the convergence of asynchronous algorithms we need the additional condition $\rho(|M_l^{-1} N_l|) < 1$ as in proposition 3.2 of [6].

Note that $\rho(|M_l^{-1} N_l|) < 1 \Rightarrow \rho(M_l^{-1} N_l) < 1$ and as well-known, asynchronous convergence implies synchronous one.

Theorem 1 If for all $l \in \{1, \dots, L\}$, $\rho(M_l^{-1}N_l) < 1$, then the synchronous version of the above parallel algorithm converges to the solution of (1). Moreover if for all $l \in \{1, \dots, L\}$, $\rho(|M_l^{-1}N_l|) < 1$, then any asynchronous version of the above parallel algorithm converges to the solution of (1).

Remark 1 In Section 2, we have considered a simple variant of our approach. In this simple case, our algorithm is similar to block-Jacobi kind methods. Nevertheless, it should be noticed that even in this simple case the block-Jacobi method is generated by a single decomposition of A whereas multisplitting methods are generated by the combination of several decompositions of A .

In the case where the subset J_l are not disjoint, we obtain the discrete analogue of Schwarz alternating algorithms, the convergence condition of theorem 1 is then different from the Jacobi convergence one. Indeed, the Jacobi convergence condition is related to a single iteration matrix, whereas our convergence condition must be satisfied by each iteration matrix $M_l^{-1}N_l$ obtained from a decomposition of A as in (5).

3.1. Important remarks

Remark 2 Notice that it can be assigned several non adjacent band matrices to a given processor, so this processor computes non adjacent components of the unknown vector. By the way of permutation matrices we can obtain the case of figure 1.

Remark 3 It must also be noticed that the subsets J_l are not necessarily disjoint, so two processors may compute shared unknown components. In the above definition of the extended fixed point mapping especially about the introduction of the family of matrices E_{lk} , it is worthwhile to note that our formulation allows:

- to give a presentation of either the Schwarz alternating method or general Schwarz multisplitting methods [15, 14] thanks to dependences of E_{lk} on the index l .
- to take $E_{lk} = E_k$ in order to obtain O'Leary and White multisplitting algorithms [13, 19].

See Section 4 for more details.

Remark 4 Direct sparse solvers first start with the factorization of the matrix and this step may be time-consuming. In our case, the factorization is achieved only once on smaller matrices, at the first iteration. The execution times of Section 6 show the impact of this point on performances.

Remark 5 If the linear system is ill conditioned then we can apply our method after having used a good preconditioner. Preconditioning methods have not been used in this paper. This will probably be the subject of future work.

4. Some derived algorithms

4.1. O'Leary and White multisplitting algorithms

Take the diagonal positive matrices E_{lk} depending only on k

$$E_{lk} = E_k,$$

and satisfying

$$\begin{cases} \sum_{k=1}^L E_k = I_n \\ (E_k)_{ii} = 0, \quad \forall i \notin I_k, \end{cases}$$

then we obtain the O'Leary and White multisplitting algorithms defined by the fixed point mapping which are

$$T(x^1, \dots, x^L) = (y^1, \dots, y^L) \text{ such that}$$

$$\begin{cases} y^l = F_l(z) \\ z = \sum_{k=1}^L E_k x^k. \end{cases}$$

4.2. Discrete analogue of the Schwarz alternating method

Let us first consider the case $l = 2$. Suppose $I_1 \cap I_2 \neq \emptyset$, so we have an overlap between the 1st and the 2nd subdomains. Consider the matrices E_{lk} such that

$$\begin{aligned} (E_{11})_{ii} &= \begin{cases} 1 & \forall i \in I_1 \\ 0 & \forall i \notin I_1 \end{cases}, & (E_{12})_{ii} &= \begin{cases} 0 & \forall i \in I_1 \\ 1 & \forall i \notin I_1 \end{cases} \\ (E_{21})_{ii} &= \begin{cases} 1 & \forall i \notin I_2 \\ 0 & \forall i \in I_2 \end{cases}, & (E_{22})_{ii} &= \begin{cases} 0 & \forall i \notin I_2 \\ 1 & \forall i \in I_2. \end{cases} \end{aligned}$$

Then equations (2)-(3) become

$$T(x^1, x^2) = (y^1, y^2) \text{ such that for } l = 1, 2$$

$$\begin{cases} y^l = F_l(z^l) \\ z^l = \sum_{k=1}^2 E_{lk} x^k. \end{cases}$$

The mapping T describes the additive discrete analogue of the Schwarz alternating method.

4.3. Discrete analogue of the multisubdomains Schwarz method

Let's define matrices E_{lk} in (4) as follows

$$\begin{aligned} (E_{ll})_{ii} &= \begin{cases} 1 & \text{if } i \in I_l \\ 0 & \text{if } i \notin I_l \end{cases} \\ (E_{lk})_{ii} &= \begin{cases} 0 & \text{if } i \in I_l \\ (E_k)_{ii} & \text{if } i \notin I_l, \end{cases} \end{aligned}$$

then (2) and (3) give rise to the discrete analogue of the multisubdomains Schwarz method which is defined by

$$T(x^1, \dots, x^L) = (y^1, \dots, y^L) \text{ such that}$$

$$\begin{cases} y^l = F_l(z^l) \\ z^l = \sum_{k=1}^m E_{lk} x^k. \end{cases}$$

5. An important class of linear systems that can be solved using multisplitting-direct algorithms

5.1. Convergence under diagonal dominance hypotheses

Proposition 1 *If matrix A is either strictly or irreducibly diagonally dominant then the linear problem can be solved on L processors using algorithm 1.*

Proof *If A is strictly or irreducibly diagonally dominant then the associated point Jacobi matrix J satisfies $\rho(|J|) < 1$, ([18] theorem 3.4) so A can be split into L convergent Jacobi like splittings.*

5.2. Convergence in the context of Z matrices

An important class of linear systems is the class of Z matrices, i.e. square matrices for which the off-diagonal entries are non positive. Linear systems are very often involved in physical, biological and social sciences [16, 7]. For example they have to be solved when we deal with scientific applications modeled by PDEs and discretized by the finite difference method.

Proposition 2 *If A is a Z matrix and if there exists a lower and an upper triangular matrices L and U such that $PAP^t = LU$ where P is a permutation matrix, then the parallel algorithm converges to the solution of (1) whatever its asynchronous execution.*

Proposition 3 *If A is a Z matrix and if every real eigenvalue of A is positive then the parallel algorithm converges to the solution of (1) whatever its asynchronous execution.*

Proof *If A is a Z matrix and if it satisfies one of the conditions of the above propositions then it is a M matrix (see [7], theorem 2.3). Now if A is a M matrix then it has a convergent weak regular multisplitting. i.e. a nonnegative convergent splitting the spectral radius of which is strictly less than 1.*

6. Experimentations

We have chosen the SuperLU library [10, 12], this is a general purpose library for the direct solution of large,

sparse, non-symmetric linear systems on high performance machines. Our choice to build our algorithms with this version rather than another one is just motivated by the fact that this library is well known [1, 11] and considered as a good one although not necessarily the best one. This library performs an LU decomposition with partial pivoting and triangular system solving through forward and back substitution. Three different versions have been developed, one for sequential machines, an other one for shared memory architectures and a third for distributed architectures using MPI library for communications.

Following our approach, two multisplitting-direct solvers have been developed. One for synchronous parallel algorithms with MPI and a second for asynchronous parallel algorithms with Corba. Both our solvers are based on the sequential version of SuperLU (version 3.0) whereas the distributed version of SuperLU used for comparison is the version 2.0.

Our experiments have been conducted in order to study different properties of our algorithms. First of all, we have studied the scalability of the three algorithms in a local homogeneous cluster. Then, we have experimented a few matrices with different kinds of properties in different contexts (local and distant). Due to security restrictions, we did not have the possibility to use more than two distant sites. In fact, it is frequent that clusters are behind firewalls, in this case, we could not perform our experimentations since our versions of MPI and Corba are not designed for this feature. So we have introduced network perturbations to simulate far clusters and to test the robustness of the algorithms. Finally, we have measured the impact of the overlapping on the performances.

Five matrices and three cluster configurations have been considered. First, three matrices, called *cage10.rua*, *cage11.rua* and *cage12.rua* have been chosen, they can be found in the University of Florida Sparse Matrix Collection [9]. Those matrices are models of DNA electrophoresis. The degree of the first is 11397 (ie. 11397×11397 elements), the degree of the second is 39082 and the degree of the third is 130228. In order to have another kind of matrices we have developed a generator that builds diagonal dominant matrices. The degree of the first matrix is 500000 and the second is 100000. It should be noticed that this second matrix has especially been chosen to measure the influence of the overlapping, that is why its spectral radius is close to 1. Three cluster configurations have been experimented:

- a local homogeneous cluster of 20 machines, called cluster1, with Pentium IV 2.6Ghz with 256Mo memory. The network is a standard 100Mb.
- a local heterogeneous cluster of 8 machines, called cluster2. The machine configuration ranges from Pen-

tium IV 1.7Mhz to Pentium IV 2.6Mhz with 512Mo memory. The network is a standard 100Mb.

- a distant heterogeneous cluster of 10 machines scattered on two distinct sites (7 machines in one site and 3 in the other one), called cluster3. Sites are standard 100Mb and are connected by 20Mb Internet links. The machine configuration ranges from Pentium IV 1.7Mhz to Pentium IV 2.6Mhz with 512Mo memory.

For all experimentations, a series of tests have been conducted and the times (expressed in seconds) reported correspond to the average of each series. The accuracy for each experiment is fixed to $1e - 8$. When the times for the factorization steps are reported they are equivalent for both the synchronous and asynchronous versions. For experiments over the distant heterogeneous cluster, no grid-middleware are used, we only used standard versions of MPI and Corba according to the synchronous or asynchronous case.

6.1. First experiments: the local homogeneous cluster case

number of processors	distributed SuperLU	synchronous multisplitting-LU	asynchronous multisplitting-LU	factorization time
1	157.63	-	-	-
2	89.27	34.15	33.38	32.61
3	69.24	19.14	19.90	18.26
4	50.32	8.43	8.05	7.82
6	39.77	2.14	2.16	1.84
8	34.34	1.05	1.04	0.84
9	30.77	0.60	0.60	0.45
12	33.36	0.29	0.36	0.19
16	33.71	0.20	1.05	0.11
20	45.99	0.14	1.84	0.06

Table 1. Experiments with cluster1 to measure the scalability of distributed SuperLU and our multisplitting-LU algorithms with the cage10.rua matrix

In Tables 1 and 2, we can notice that the speed up of distributed SuperLU version is quite good up to 10 processors in the first table and up to 20 in the second one but, in spite of this, both versions of the multisplitting-direct algorithms outperform the distributed SuperLU version. Moreover, the

number of processors	distributed SuperLU	synchronous multisplitting-LU	asynchronous multisplitting-LU	factorization time
4	1496.28	131.69	131.45	126.78
6	949.20	44.29	44.17	41.73
8	762.76	12.44	12.25	11.09
9	679.17	11	11	9.91
12	540.49	3.77	3.78	3.16
16	456.54	1.24	2.34	0.71
20	471.70	1.01	2.03	0.30

Table 2. Experiments with cluster1 to measure the scalability of distributed SuperLU and our multisplitting-LU algorithms with the cage11.rua matrix

most time-consuming step in those multisplitting-direct algorithms is the factorization step. In the experiments related to the second table, the considered matrix requires too much memory to be solved with less than 4 processors. Those experiments also highlight that performances of both synchronous and asynchronous multisplitting algorithms are similar on a local homogeneous cluster with a slight advantage for the synchronous one when the computation/communication ratio decreases. Nonetheless, we believe that this advantage would have been in favor of asynchronous algorithms if we could have done experimentations with a very large number of processors. The bad performances of the asynchronous case with 16 and 20 processors are due to the convergence detection that takes more time with more processors and to the number of iterations that increases when the computation parts are very short.

6.2. Second experiments: the distant heterogeneous clusters case

Table 3 reports the execution times of the distributed version of SuperLU and the multisplitting-LU solvers. In all cases, those experiments clearly illustrate the very good speed-up between the distributed SuperLU and our algorithms. It can be seen that the factorization times are quite significant. Moreover, the asynchronous algorithm is slightly faster than the synchronous one.

Another important fact concerns the memory required for the computation. Standard direct solvers require much more memory than both our algorithms. This issue is clearly illustrated with the *cage12* matrix which cannot be executed

matrix	cluster configuration	distributed SuperLU	synchronous multisplitting-LU	asynchronous multisplitting-LU	factorization time
case11	cluster2	1212	12.7	12.1	11
case12	cluster3	nem	441.5	441.2	430.3
500000 matrix	cluster3	15145	17.44	15.76	4.05

Table 3. Comparison of the three solvers with cluster2 and cluster3 (nem stands for not enough memory)

with the given cluster configuration whereas our algorithms run perfectly. In addition, the execution of the sequential version of SuperLU with *case11* has been tested on a single processor with 1Gb memory. The computation has failed because the program required more memory. That is why we did not report any speed-up as compared with the sequential version of SuperLU.

6.3. Third experiments: the impact of communications

number of perturbing communications	distributed SuperLU	synchronous multisplitting-LU	asynchronous multisplitting-LU
0	15145s	17.44s	15.76s
1	18321s	33.50s	18.60s
5	20296s	63.4s	29.33s
10	22600s	99.35s	44.13s

Table 4. Impacts of load of networks with cluster3 with the 500000 generated matrix

Table 4 shows the impact of the load of the network and so the impact of communications on the algorithms. We perturbed the network by artificially adding perturbing communications between the two distant sites. So the bandwidth is considerably reduced. It may be pointed that the number

of perturbing tasks does not linearly influence the execution times. Indeed, computations and perturbing tasks interact and slow down each other. Moreover, it is strongly probable that other tasks were also running simultaneously (ftp, machine update, mail, ...).

This table obviously highlights the robustness of the algorithms with respect to the heterogeneity of the communications, because even with a lot of bandwidth traffic, performances are hardly perturbed. As a consequence, we can conclude that our algorithms do not require a lot of bandwidth, furthermore, in the case of bandwidth perturbations, the asynchronous version is more efficient.

6.4. Fourth experiments: the impact of overlapping

In the numerical analysis framework it is known that overlapping techniques can enhance the convergence by reducing the number of iterations. Computer scientists are interested in reducing the total time of execution of an algorithm, in our case, times of factorization are thus very important factors to study.

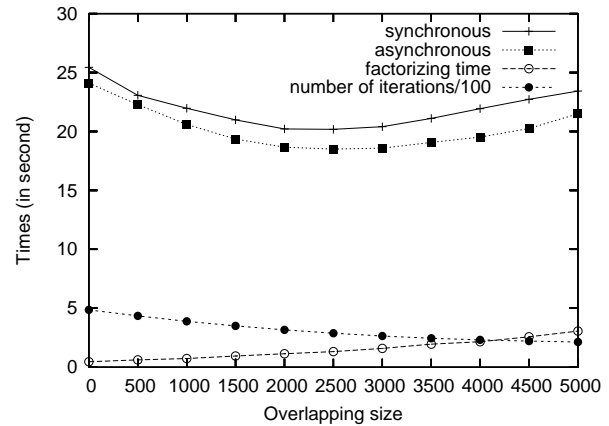


Figure 3. Impacts of overlapping with cluster3 with the 100000 generated matrix

Figure 3 illustrates the impact of overlapping on the proposed algorithms. The larger the overlapping is, the more the factorization step is time-consuming. Consequently, this size should be defined carefully and should take into account the size of the matrix and its parameters such as time of factorization, spectral radius, etc. In the figure, the number of iterations for the synchronous algorithm are reported. This number is divided by 100 for display commodity. We see in the figure that, in our case, the best overlapping size is 2500.

Concerning the asynchronous algorithm, we did not mention the number of iterations because in the asynchronous context, each processor freely iterates and this number widely differs from one processor to another. However, this number is systematically greater than the synchronous one.

7. Conclusion

In this paper a new approach for the parallelization of direct algorithms is introduced, it consists in splitting the original matrix into as many partitions as processors (or clusters) and in assigning a submatrix to each processor (cluster). Then, each processor solves the corresponding subvector at its own rhythm. As each processor computes independently its unknown subvector, the solution is not obtained in one step. This approach is equivalent to building a block iterative algorithm over the network of clusters. It particularizes known algorithms such as "O'Leary and White" and "Schwarz alternating" algorithms, even if those algorithms were not designed in the aim of parallelizing direct algorithms. The convergence to the solution of the original problem in synchronous and asynchronous modes is proven.

The reported experiments in this paper show that these algorithms are strongly efficient in distributed environments, especially in distant ones. The class of applications is wide, it covers several scientific computation areas. The synchronous version is efficient in homogeneous network conditions, whereas the asynchronous one provides robustness to the unpredictable perturbations of the network bandwidth. Several important factors are studied, such as the time of factorization of the matrix, the size of overlapping components, the memory requirement and the impact of the heterogeneity of the communications.

This work is the first in a series of future works. Now we plan to improve our approach on several other direct solvers. We will also consider the case where different direct algorithms on different clusters are used and we will study the impact of coupling such direct algorithms. Finally, we plan to generalize this approach to the case of nonlinear problems.

References

- [1] P. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, 27(4):388–421, 2001.
- [2] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Asynchronism for iterative algorithms in a global computing environment. In *International Conference on High Performance Computing Systems and Applications, HPCS'02*, pages 90–97. IEEE Computer Society Press, 2002.
- [3] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Performance comparison of parallel programming environments for implementing AIAC algorithms. In *International Parallel and Distributed Processing Symposium, IPDPS*, pages 247b, 8 pages. IEEE computer science press, April 2004.
- [4] J. M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 1:4–13, 2005.
- [5] J. M. Bahi, R. Couturier, and M. Salomon. Solving three-dimensional transport models with synchronous and asynchronous iterative algorithms in a grid computing environment. In *19th IEEE and ACM Int. Conf. on Parallel and Distributed Processing Symposium, IPDPS 2005*, 2005. To appear.
- [6] J. M. Bahi, J.-C. Miellou, and K. Rhofi r. Asynchronous multisplitting methods for nonlinear fixed point problems. *Numerical Algorithms*, 15(3,4):315–345, 1997.
- [7] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Reprinted by SIAM, Philadelphia, 1996. Academic Press, New York, third edition, 1984.
- [8] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, Englewood Cliffs NJ, 1989.
- [9] T. Davis. University of florida sparse matrix collection, 1997. NA Digest, see <http://www.cise.ufl.edu/research/sparse/matrices/>.
- [10] L. Grigori and X. S. Li. A new scheduling algorithm for parallel sparse lu factorization with static pivoting. In *Super Computing 2002*. IEEE computer society press and ACM sigarch, 2002. paper 139 on CD.
- [11] A. Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. *ACM Transactions on Mathematical Software*, 28(3):301–324, 2002.
- [12] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, June 2003.
- [13] D. P. O'Leary and R. E. White. Multi-splittings of matrices and parallel solution of linear systems. *Journal on Algebraic and Discrete Mathematic*, 6:630–640, 1985.
- [14] G. Rodrigue, L. Kang, and Y. Liu. Convergence and comparison analysis of some numerical Schwarz methods. *Numerische Mathematik*, 56(2/3):123–138, Oct. 1989.
- [15] G. H. Rodrigue and J. Simon. A generalized numerical schwarz algorithm. In R. Glowinski and J.-L. Lions, editors, *Computer Methods in Applied Sciences and Engineering VI.*, pages 272–281. Elsevier, 1984.
- [16] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, New York, 1996.
- [17] M. E. Tarazi. Some convergence results for asynchronous algorithms. *Numerische Mathematik*, 39:325–340, 1982.
- [18] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, 1962.
- [19] R. E. White. Multisplitting of a symmetric positive definite matrix. *SIAM Journal on Matrix Analysis and Applications*, 11:69–82, 1990.