Fractional decomposition of matrices and parallel computing

Frédéric Hecht, Sidi-Mahmoud Kaber

Fractional decomposition of matrices and parallel computing

Frédéric Hecht, Sidi-Mahmoud Kaber
Sorbonne Université,
Université Paris-Diderot, CNRS,
Laboratoire Jacques-Louis Lions,
LJLL, F-75005 Paris, France

September 21, 2018

Abstract

We are interested in the design of parallel numerical schemes for linear systems. We give an effective solution to this problem in the following case: the matrix A of the linear system is the product of p nonsingular matrices A_i^m with specific shape: $A_i = I - h_i X$ for a fixed matrix X and real numbers h_i. Although having the special form, these matrices A_i arise frequently in the discretization of evolutionary Partial Differential Equations. The idea is to express A^{-1} as a linear combination of elementary matrices A_i^{-k}. Hence the solution of the linear system with matrix A is a linear combination of the solutions of linear systems with matrices A_i^k. These systems are solved simultaneously on different processors.

1 Introduction

Let X be a real $n \times n$ square real (or complex) matrix and $(h_i)_{i=1}^p$ a collection of pairwise distinct real numbers. Matrices of the shape

$$A_i = I - h_i X \tag{1}$$

with I the $n \times n$ identity matrix, appear in many numerical schemes for differential equations. For small h_i (discretization parameters), such matrices are nonsingular. So is matrix

$$A = \prod_{i=1}^{p} A_i^m \tag{2}$$

for any $m \in \mathbb{N}$. Note that matrices A_i commute and A is a polynomial of the single variable X. The problem of interest here is the following: given $y \in \mathbb{R}^n$, compute the unique solution $x \in \mathbb{R}^n$ of the linear system

$$Ax = b \tag{3}$$

using several processors at our disposal. The aim is to use parallel computing to save computational time. The key idea consists in expressing matrix A^{-1} as a linear combination of matrices $((A_i^{-k})_{i=1}^p)_{k=1}^m$. Since matrices A_i we consider come from the discretization of differential equations by time implicit scheme, our method falls into the family of "parallelization in time" algorithms. We refrer to [3] for a comprehensive overview of parallel time integration methods.
The paper is organised as follows.

- In section 2, we present the algebraic problem to be solved.
- In section 3, we explain how the method can be used to solve homgeneous evolution equations $u_t = \mathcal{L}u$.
- In section 4, we consider nonhomgeneous evolution equations $u_t = \mathcal{L}u + f$ and derive a new parallel algorithm to solve such problems.
- The last section contains remarks on the reliability of the method and its limitations.

All computations have been done using FreeFem++, a Finite Elements software for the discretization of Partial Differential Equations [1].

2 The algebraic problem

We consider in this section the homogeneous linear time-evolution problem

$$\mathbf{u}_t = \mathcal{L}\mathbf{u}. \tag{4}$$

Section 4 is devoted the nonhomogenoeus case which requiers a different numerical treatment. Starting from an initial data $\mathbf{u}^0$, the time discretization of this equation by implicit Euler scheme with time step h_1 reads

$$\frac{\mathbf{u}^1 - \mathbf{u}^0}{h_1} = \mathcal{L}\mathbf{u}^1. \tag{5}$$

Solving (5) with a Finite Difference scheme gives

$$(I - h_1 X)u^1 = u^0. \tag{6}$$

with X a Finite Difference approximation of the spatial operator $\mathcal{L}$. Iterating p steps of the implicit scheme, we obtain an equation for u^p

$$(I - h_p X)\cdots(I - h_2 X)(I - h_1 X)u^p = u^0.$$

We will always assume the time steps h_i small enough insuring that all matrices A_i and A

$$A = \prod_{i=1}^{p} A_i \tag{7}$$

are nonsingular. Hence u^p is given by

$$u^p = A^{-1}u^0.$$

Here comes into play the fractional decompositions of matrices. Indeed, if the h_i are piecwise distinct, there exist p real numbers $(\alpha_i)_{i=1}^p$ such that

$$A^{-1} = \sum_{k=1}^{p} \alpha_i A_i^{-1}. \tag{8}$$

Consequently, the vector u^p could be split into

$$u^p = \sum_{i=1}^{p} \alpha_i x_i,$$

each x_i being the unique solution of the system

$$A_i x_i = u^0. \tag{9}$$

These p linear systems are independent each other allowing the computation of u^p by using p processors to compute simultaneously the x_i. This idea has been applied in [2] to solve the two-dimensional heat equation discretized by a Finite Difference scheme. Note that the α_i involved in (8) have a simple expression

$$\alpha_i = \prod_{k \neq i} (1 - h_k/h_i)^{-1}. \tag{10}$$

We introduce now the general case where the matrix A to be inverted is definied by (2) with A_i defined again as in (1). We will always assume that the h_i are piecewise distinct.

Proposition 1 *There exist mp real numbers $\alpha_{i,k}$ such that the inverse of A defined in (2) is*

$$A^{-1} = \sum_{i=1}^{p} \sum_{k=1}^{m} \alpha_{i,k} A_i^{-k}. \tag{11}$$

This is just the partial fraction decomposition of rational functions written for polynomials of matrices. According to (11), the unique solution x of the linear system (3), with A defined in (2), is decomposed as

$$x = \sum_{i=1}^{p} x_i. \tag{12}$$

with

$$x_i = \sum_{k=1}^{m} \alpha_{i,k} x_{i,k}. \tag{13}$$

and $x_{i,k}$ being the unique solution of the linear system

$$A_i^k x_{i,k} = b.$$

The computation of each x_i is done on one processor allowing the parallelization of the algorithm. Here is an efficient way to do so: On processor number i,

$$\begin{array}{l} \star \text{ compute } x_{i,1} \text{ solution of } A_i x_{i,1} = b, \\ \star \text{ compute } x_{i,2} \text{ solution of } A_i x_{i,2} = x_{i,1}, \\ \qquad \vdots \\ \star \text{ compute } x_{i,m} \text{ solution of } A_i x_{i,m} = x_{i,m-1}, \\ \star \text{ lastly, compute } x_i \text{ by (13).} \end{array} \tag{14}$$

Computation of the solution x by (12) requiers communications between processors. This step maybe very time consuming, especially if the amount of data to transfer is large.

We end this section with numerical issues. The coefficients $\alpha_{i,k}$ in (11) are given by

$$\alpha_{i,j} = \frac{1}{(m-j)!} \lim_{t \to 1/h_i} [\frac{(t - h_i t)^m}{\varphi(t)}]^{(m-j)}. \tag{15}$$

with

$$\varphi(t) = \prod_{i=1}^{p} (1 - h_i t)^m.$$

For $m = 1$, formulas (15) reduces to (10). For $m = 2$, there exist also simple explicit formulas :

$$A^{-1} = \sum_{i=1}^{p} \alpha_{i,1} A_i^{-1} + \alpha_{i,2} A_i^{-2} \tag{16}$$

with

$$\alpha_{i,2} = \prod_{j \neq i} \frac{1}{(1 - \frac{h_j}{h_i})^2}, \quad \alpha_{i,1} = -2\alpha_{i,2} \sum_{j \neq i} \frac{h_j}{h_i - h_j}. \tag{17}$$

For $m > 2$ there is an alternate method to the computation of the mp coefficents $\alpha_{i,k}$ by (15): solve the following linear system for well choosen mp real numbers t_ℓ

$$\sum_{i=1}^{p} \sum_{k=1}^{m} \frac{1}{(1 - h_i t_\ell)^k} \alpha_{i,k} = \frac{1}{\varphi(t_\ell)}, \qquad 1 \leq \ell \leq mp. \tag{18}$$

In other words, compute the coefficients $\alpha_{i,k}$ by interpolation of the function $1/\varphi$ at the distincts nodes t_ℓ (to be well choosen). With $\alpha_i \in \mathbb{R}^m$ and $\alpha \in \mathbb{R}^{mp}$ defined by

$$\alpha_i = \begin{pmatrix} \alpha_{i,1} \\ \vdots \\ \alpha_{i,m} \end{pmatrix}, \quad \alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_p \end{pmatrix},$$

the linear system to solve is

$$\begin{pmatrix} \psi^1(t_1) & \cdots & \psi^p(t_1) \\ \vdots & & \vdots \\ \psi^1(t_{mp}) & \cdots & \psi^p(t_{mp}) \end{pmatrix} \alpha = \begin{pmatrix} 1/\varphi(t_1) \\ \vdots \\ 1/\varphi(t_{mp}) \end{pmatrix} \tag{19}$$

with

$$\psi^i(t_\ell) = (\frac{1}{1 - h_i t_\ell}, \cdots, \frac{1}{(1 - h_i t_\ell)^m})$$

This system, a Vandermonde-like one, is ill-conditionned. Since only the case $m = 1$ was used in our numerical tests, we do not discus further the linear system (19). Of course, preconditionning of the system is necessary for larger values of m.

The main drawback of the method is that the coefficients do not have a constant sign and, in addition, may be very large, consult [2]. This affects the accuracy. Typically, only an approximation $\tilde{x}_i$ of x_i is available: $\|\tilde{x}_i - x_i\| \leq \varepsilon$, from which we deduce the following estimate of the error on the solution x:

$$\|\tilde{x} - x\| \leq \left[\sum_{i=1}^{p} |\alpha_i| \right] \varepsilon. \tag{20}$$

Since the first term, that amplifies the error, may highly increases with the parameter p, it is crucial to limit our investigation for small values of this parameter, see [2]. As an illustration, Table 1 displays the evolution of the amplification term as a function of p.

3 The homogeneous case

As was stated earlier, Proposition 1 was used in [2] (in the case $m = 1$) to solve homogeneous linear time-evolution equation (4) discretized by a Finite Difference method. In this section, we reproduce the

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$
$\sum_{i=1}^{p} \lvert\alpha_i\rvert$	10.00	201.00	530.00	7068.33	10895.83

Table 1: Evolution of the amplification term in (20) as a function of the number of processors p. The h_i are choosen symmetrically : $c = 1/10$ and in the case $p = 2$, $h_i = (1 \pm c)h$, in the case $p = 3$, $h_i \in \{h, (1 \pm c)h\}$, in the case $p = 4$, $h_i \in \{(1 \pm c)h, (1 \pm 2c)h\}$, ...

tests in [2] using a Finite Element scheme. One iteration of a Finite Elelement Method to solve the same problem reads

$$M\frac{u^1 - u^0}{h_1} = Bu^1$$

with M the mass matrix and B the stiffness one. We obtain the same discrete evolution equation as in the Finite Difference case (6) with

$$X = M^{-1}B.$$

As we will see in next section this is not the always the case for nonhomogeneous equations. Consider firstly the case $m = 1$ that will be mostly used in practice (*i.e.* A is defined by (7). Let T_0, T_f be the initial and final times. We define p time steps h_1, h_1, $\cdots$, h_p et intermediate times $t_n = T_0 + n\times$ (time steps) used in a cyclical order: firstly h_1, then h_2, $\cdots$, h_p, $\cdots$

$$t_n = T_0 + \underbrace{h_1 + h_2 + \cdots + h_p + h_1 + h_2 + \cdots}_{n \text{ terms}}$$

It is important to precise that our algorithm computes the solution only at times t_{kp}, $k = 1, \cdots, K$ by going directly from approximation at time t_{kp} to approximation at time $t_{(k+1)p}$. Indeed, suppose u_{kp} known. Instead of computing sequentially

$$\begin{array}{l} \star\ u_{kp+1} \text{ solution of } A_1 u_{kp+1} = u_{kp}, \\ \star\ u_{kp+2} \text{ solution of } A_2 u_{kp+2} = u_{kp+1}, \\ \quad \vdots \\ \star\ u_{kp+p} \text{ solution of } A_p u_{kp+p} = u_{kp+p-1}, \end{array} \tag{21}$$

we write $u_{(k+1)p}$ as the solution of the linear system

$$Au_{(k+1)p} = u_{kp} \tag{22}$$

with $A = \prod_{i=1}^{p} A_i$ and solve this system using the decomposition (8)-(10).
Let us compare sequential versus parallel computations.

1. *Sequential solution.* The cost of the sequential solution obtained by procedure (21) is the cost of solving p linear sytems

2. *Parallel solution.* The cost of the parallel procedure (8)-(10) is that of one linear system sinch each x_i is computed in full parallel. However, we have to take into account the cost of summing up the x_i to get the solution by (12). This imposes a global communication requirement between all of the processors in order to compute and share the updated solutions. This harms the efficiency of the method.

In the general case $m \geq 2$, the comparison is also favourable for the parallel algorithm.

1. *Sequential solution.* Solve mp linear sytems to compute x the solution of (21) with A_i replaced by A_i^m. The cost is that of solving mp linear sytems.

2. *Parallel solution.* On each of the p processors, compute in full parallel one x_i this way:

   (a) compute the $x_{i,k}$ iteratively:

   $$A_i x_{i,1} = b, \quad A_i x_{i,2} = x_{i,1}, \quad \dots \quad A_i x_{i,m} = x_{i,m-1}$$

   by solving m linear systems,

   (b) add the $x_{i,k}$ up to get x_i using (13).

   The cost is that of solving m linear sytems. Of course, there is an additional cost to take into account: the summing up the x_i to get x by (12).

We use a P1-Finite Element method for discretization in space and implicit Euler scheme for time discretization to solve the $2D$ homogeneous heat equation on a domain $\Omega \subset \mathbb{R}^2$ with homogeneous Neumann boundary conditions. The domain Ω is the unit square, the initial condition is is $u_0(x, y) = \cos(n\pi x)\cos(m\pi y)$ so that the exact solution is known. The final time of all numerical experiments presented is $T = 1$. The computations were done on supercomputer SGI-UV2000 with 32 CPUs (Intel Xeon 64 bits EvyBridge E4650 with 10 core) using MPI parallel implementation in Freefem.

There are p processors at our disposal and we use each of them to advance in time (only once) with a time-step h_i ($1 \leq i \leq p$). The linear system to solve is (2) with $m = 1$.

Figure 1 represents the error. The error increases with p the number of processors. Up to 7 processors, the error is constant, almost equal the error made by using one processor. Starting from $p = 8$, the error increases. The growth of the error is due to bad behaviour of the coefficents α_i, see Table 1. As already mentioned, we limit the use of the methos to moderate values of p. The method is therefore far from massively parallel computing!

Figure 2 represents the computational time as a function of p. We observe a decreasing of computational time as the number of processors increases. For example, using 6 processors instead of one, divide the computational time by a factor 5.8.

Two quantities are important in parallel computing: the Speedup defined as the ratio of sequential time over parallel time and the parallel Efficiency defined as $\frac{1}{p} \times$Speedup.

Both are displayed on figure 3. Ideally, the speedup should be equal to p if the communications between processors have no cost. But this is, of course, never the case. We obtain the following speedups: with 2 processors, we have a speedup of 1.95, a speedup of 3.8 using 4 processors, and a speedup of 8.2 with 9 processors. Idealy, the parallel efficiency should be equal to 1. We obtain the following results: with 2 processors, we have an efficiency of 0.98, an efficiency of 0.96 using 4 processors, and an efficiency of 0.9 with 9 processors.

The algorithm gives good results for the values of p we tested. We emphasize again that we do not plan to use our method for large values of p since the amplification term in (20) growth rapidly with p as shown in Table 1. We return to this issue in the next section.

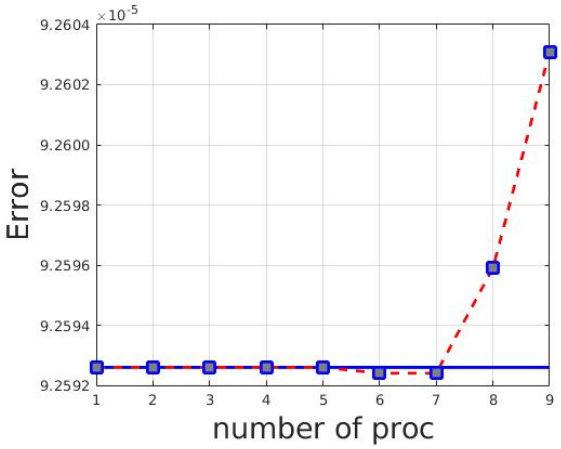

Figure 1: The $2D$ homogeneous heat equation. L^2 error (numerical solution versus exact solution). Numerical parallel solution as a function of p that is also the number of processors. Solid line represents the error of the sequential solution.

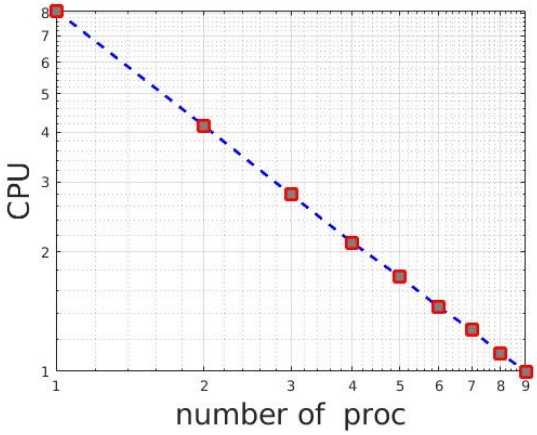

Figure 2: The $2D$ homogeneous heat equation. CPU time (log scale) as a function of the number of processors.

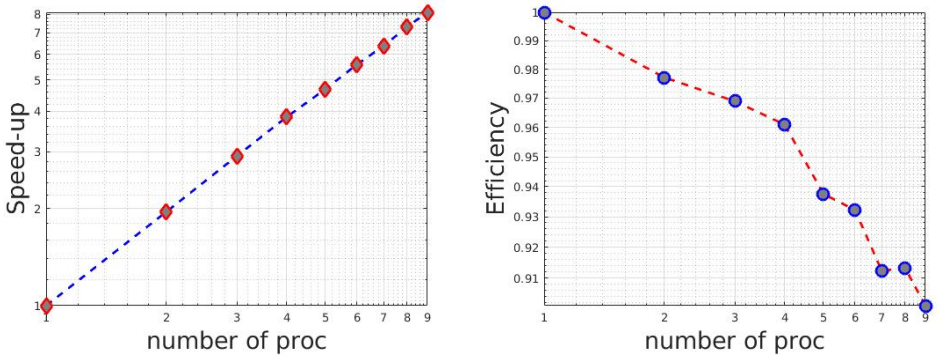

Figure 3: The $2D$ homogeneous heat equation. Speedup (left) and Efficiency (right) as a function of the number of processors.

4 The nonhomogeneous case

Consider the nonhomogeneous Differential Equation

$$\mathbf{u}_t = \mathcal{L}\mathbf{u} + \mathbf{f}. \tag{23}$$

One step in the Finite Element Method to solve (23) reads

$$M\frac{u^{n+1}-u^n}{h_i} = Bu^{n+1} + \mathbf{f}^{n+1}$$

with M the mass matrix, B the stiffness one, and

$$\mathbf{f}^{n+1} = (\langle f(.,t_{n+1}), \varphi_i\rangle)_i,$$

with $(\varphi_i)_i$ the Finite Element Basis. If f belongs to $\mathcal{V}$, the Finite Element space, we have

$$\mathbf{f}^{n+1} = MF^{n+1}, \quad F^{n+1} = (f(S_i, t_{n+1}))_i,$$

with S_i the nodes of the Finite Element triangulation. In that case, we obtain the same expression as in the Finite Difference setting with

$$A_i = I - h_i X, \quad X = M^{-1}B.$$

We introduce now some notations to simplify the presentation:

- $\star$ $\Delta T = m\sum_{k=1}^{p} h_k$,
- $\star$ $T_n = T_0 + n\Delta T$ and u_n the approximation of $u(T_n)$,
- $\star$ $T_{n,j} = T_n + m\sum_{k=1}^{j} h_k$ and $u_{n,j}$ the approximation of $u(T_{n,j})$.
  Note that $T_{n,0} = T_n$ and $T_{n,p} = T_{n+1}$.

Suppose u_n known, using m times the time-step h_1, we get $u_{n,1}$ solution of

$$A_1^m u_{n,1} = u_n + h_1 \sum_{j=1}^{m} A_1^{j-1} f(T_n + jh_1).$$

Using then m times the time-step h_2, we get $u_{n,2}$ solution of

$$A_2^m u_{n,2} = u_{n,1} + h_2 \sum_{j=1}^{m} A_2^{j-1} f(T_{n,1} + jh_2).$$

Finally

$$A_p^m u_{n,p} = u_{n,p-1} + h_p \sum_{j=1}^{m} A_p^{j-1} f(T_{n,p-1} + jh_p).$$

So that, $u_{n+1} = u_{n,p-1}$ is the solution of the linear system

$$Au_{n+1} = u_n + g_n, \tag{24}$$

with

$$g_n = \sum_{k=1}^{p} h_k B_k \sum_{j=1}^{m} A_k^{j-1} f(T_{n,k-1} + jh_k) \tag{25}$$

and

$$B_k = \prod_{\ell=1}^{k-1} A_\ell^m.$$

The idea is to compute u^{n+1} using Proposition 1 to solve (24).

4.1 The algorithm

First version

1. For $k = 0, \cdots, K_1 - 1 \quad (K_1 \Delta T = T_{final})$
   (a) u_k is known (it is an approximation of u at time T_k)
   (b) Compute g_k defined in (25)
   (c) Compute u_{k+1} solution of (24) To do so
      i. compute in parallel x_i by (13) and (14) with $b = u_k + g_k$,
      ii. make a "reduce" step to compute u_{k+1} by (12):

$$u_{k+1} = \sum_{i=1}^{p} x_i.$$

   (d) u_{k+1} is computed (it is an approximation of u at time $T_{k+1} = T_n + \Delta T$)

It is important to point out that the algorithm computes the solution only at times T_k, $k = 1, \cdots, K_1$ by going directly from approximation at time T_k to approximation at time T_{k+1}. In one step, the advance in time is equal to ΔT. Communications between the processors is done K_1 times (during the sommation process).

Remarque 1 *Step 1b of the algorithm may be improved. Instead of computing g_k by one processor while the others sleep, we use all processors to compute in parallel the p next right-hand sides $g_{k'}$.*

This leads to the following algorithm that we used in the computations. Note that, now, the outer loop is performed $K_2 = K_1/p$ times.

Second version

1. For $k = 0, \cdots, K_2 - 1 \quad (K_2 p \Delta T = T_{final})$
   (a) u_{kp} is known (it is an approximation of u at time T_{kp})
   (b) Compute in parallel the p right-hand sides $g_{kp+(i-1)pm}$ for $i = 1, \cdots, p$.
   (c) Compute sequentially u_{kp+ipm} for $i = 1, \cdots, p$ solution of

$$Au_{kp+ipm} = u_{kp+(i-1)pm} + g_{kp+(i-1)pm}$$

   To do so
      i. compute in parallel x_i by (13) and (14) with $b = u_{kp+(i-1)pm} + g_{kp+(i-1)pm}$,
      ii. make a "reduce" step to compute u_{kp+ipm} by (12):

$$u_{kp+ipm} = \sum_{i=1}^{p} x_i.$$

   (d) $u_{(k+1)p}$ is computed (it is an approximation of u at time $T_{(k+1)p} = T_{kp} + p\Delta T$)

The algorithm computes the solution only at times T_{kp}, by going directly from approximation at time T_{kp} to approximation at time $T_{(k+1)p}$. In one step, the advance in time is equal to $p\Delta T$. Communications between all of the processors is done K_2 times. *Thus, we have p times less parallel overhead.* That is very valuable as we will see in the next section.

4.2 Numerical experiments

We use a P1-Finite Element method for discretization in space and implicit Euler scheme for time discretization to solve the the $2D$ nonhomogeneous heat equation

$$\partial_t u - \Delta u = f$$

on a domain $\Omega \subset \mathbb{R}^2$ with Dirichlet boundary conditions. The domain Ω is the unit square. The source term f is choosen so that the exact solution is $u(x, y, t) = \sin(n\pi x)\sin(m\pi y)\cos(mt)$. The final time of all numerical experiments presented is $T = 1$.

We fix $m = 1$ and vary p, the number of processors. Figure 4 represents the error. As already mentioned, the error increases with p the number of processors. For $p = 8$ the error becomes unacceptable. Afterward, we will consider at most $p = 7$ processors.

Figure 5 represents the computational time as a function. We observe a decreasing of computational time as the number of processors increases. For example, using 6 processors instead of one, divide the computational time a factor 4.9.

Speedup and parallel Efficiency are displayed on figure 6. We obtain the following speedups: with 2 processors, we have a speedup of nearly 1.9, a speedup of 3.14 using 4 processors and a speedup of 4.89 with 6 processors. We obtain the following results for efficiency: with 2 processors, we have an efficiency of 0.96, an efficiency of 0.79 using 4 processors and an efficiency of 0.816 with 6 processors.

Although they are less impressive than in thehomogeneous case, both Speedup and Parallel are very good. We mention again that we plan to use our method only for moderate values of p.

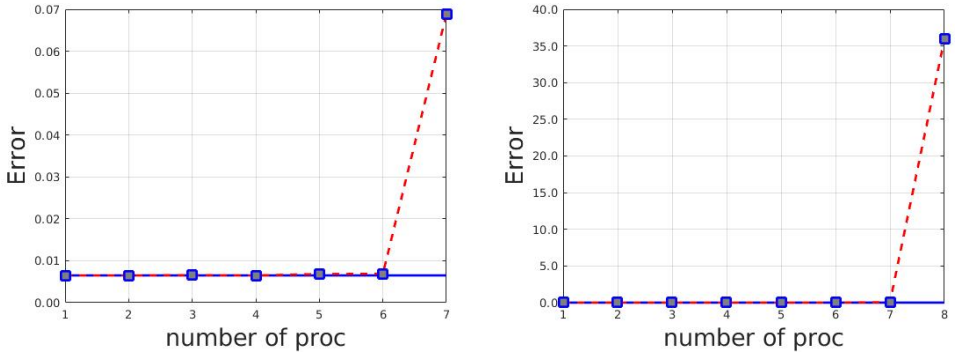

Figure 4: The $2D$ nonhomogeneous heat equation. L^2 error (numerical solution versus exact solution). Numerical parallel solution as a function of p that is also the number of processors. Solid line represents the error of the sequential solution.

5 Conclusion

We have extended the work initiated in [2] on the fractional decomposition of the inverse of some specific matrices resulting from the discretization of evolutionary differential equations. We apply this decomposition to solve nonhomgeneous Partial Differential Equations using parallel computing. Although, both speedup and parallel efficiency are good, the method is effective, for accuracy purposes, only for use on a moderate number of processors. Several developments of the method are currently investigated: analyse of the case $m \geq 2$ (see discussion in Section 2), combine first order methods computed in parallel to get a high order method, and eventually combine the method with space-parallelization (domain decomposition methods).

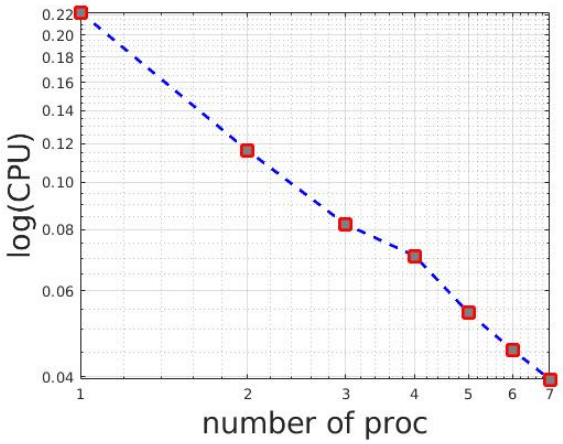

Figure 5: The $2D$ nonhomogeneous heat equation. CPU time (log scale) as a function of the number of processors.

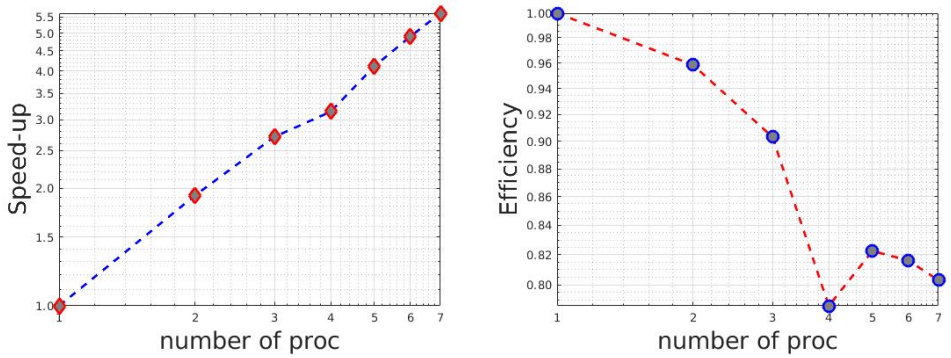

Figure 6: The $2D$ nonhomogeneous heat equation. Speedup (left) and Efficiency (right) as a function of the number of processors.

References

[1] F. Hecht, New development in FreeFem++. J. Numer. Math. 20, no. 3-4, 2012.

[2] S.-M. Kaber, A. Loumi and P. Parnaudeau, Parallel Solution of Linear Systems. East Asian Journal on Applied Mathematics, Vol. 6, No. 3, 2016.

[3] M.J. Gander, 50 Years of Time Parallel Time Integration, to appear in 'Multiple Shooting and Time Domain Decomposition', T. Carraro, M. Geiger, S. Körkel, R. Rannacher, editors, Springer Verlag, 2015.