

## Partial Fraction Decomposition of Matrices and Parallel Computing

Frédéric Hecht\* and Sidi-Mahmoud Kaber

*Laboratoire Jacques-Louis Lions (LJLL), Sorbonne-Université, CNRS, Université de Paris, Inria, F-75005 Paris, France.*

Received January 11, 2019; Accepted March 28, 2019

---

**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 a special form, these matrices  $A_i$  arise frequently in the discretization of evolutionary Partial Differential Equations. For example, one step of the implicit Euler scheme for the evolution equation  $u' = Xu$  reads  $(I - hX)u^{n+1} = u^n$ . Iterating  $m$  times such a scheme leads to a linear system  $Au^{n+m} = u^n$ . The idea is to express  $A^{-1}$  as a linear combination of elementary matrices  $A_i^{-1}$  (or more generally in term of 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  $A_i$  (or  $A_i^k$ ). These systems are then solved simultaneously on different processors.

**AMS subject classifications:** 65M60, 65Y05, 35K45, 74S05, 74S20

**Key words:** Partial differential equations, parabolic equation, finite element methods, finite difference methods, parallel computing.

---

### 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

$$A_i = I - h_i X \tag{1.1}$$

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

$$A = \prod_{i=1}^p A_i^m \tag{1.2}$$

---

\*Corresponding author. *Email addresses:* hecht@ljl11.math.upmc.fr (F. Hecht), kaber@ljl11.math.upmc.fr (S.-M. Kaber)

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

$$Ax = b \tag{1.3}$$

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. More precisely, it is a Direct Parallel Time Integration like the one developed by Y. Maday and E. Rønquist in [5], based on tensor-product space-time solvers. We refer to the survey article by M. Gander [1] for a comprehensive overview of parallel time integration methods.

Our method is also related to exponential integrators. Consider the linear constant-coefficients system  $u_t = Lu$  whose solution is  $u(t) = e^{tL}u_0$ . Applying  $p$  steps of an implicit scheme, we obtain a numerical evolution  $u_p$  defined by  $Au_p = u_0$  with  $A$  the product of  $p$  matrices  $A_i$  (see Section 2). Writing  $A^{-1}$  as a linear combination of  $A_i^{-1}$ 's is equivalent to seek for a rational approximation of the solution operator  $e^{tL}$ . Rational approximation of the exponential is an important problem in numerical analysis. Consult the survey paper by M. Hochbruck and A. Ostermann ([3]) on exponential integrators.

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 homogeneous evolution equations  $u_t = \mathcal{L}u$ .
- In Section 4, we consider nonhomogeneous 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 [2].

## 2 The algebraic problem

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

$$\mathbf{u}_t = \mathcal{L}\mathbf{u}. \tag{2.1}$$

Section 4 is devoted the nonhomogeneous case which requires 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

$$\frac{\mathbf{u}^1 - \mathbf{u}^0}{h_1} = \mathcal{L}\mathbf{u}^1. \quad (2.2)$$

Solving (2.2) with a Finite Difference scheme gives

$$(I - h_1 X)u^1 = u^0, \quad (2.3)$$

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  $A$

$$A = \prod_{i=1}^p A_i \quad (2.4)$$

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 piecewise distinct, there exist  $p$  real numbers  $(\alpha_i)_{i=1}^p$  such that

$$A^{-1} = \sum_{k=1}^p \alpha_k A_k^{-1}. \quad (2.5)$$

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

$$A_i x_i = u^0. \quad (2.6)$$

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 [4] to solve the two-dimensional heat equation discretized by a Finite Difference scheme. Note that the  $\alpha_i$  involved in (2.5) have a simple expression

$$\alpha_i = \prod_{k \neq i} (1 - h_k / h_i)^{-1}. \quad (2.7)$$

We introduce now the general case where the matrix  $A$  to be inverted is defined by (1.2) with  $A_i$  defined again as in (1.1). We will always assume that the  $h_i$  are piecewise distinct.

**Proposition 2.1.** There exist  $mp$  real numbers  $\alpha_{i,k}$  such that the inverse of  $A$  defined in (1.2) is

$$A^{-1} = \sum_{i=1}^p \sum_{k=1}^m \alpha_{i,k} A_i^{-k}. \tag{2.8}$$

This is just the partial fraction decomposition of rational functions written for polynomials of matrices. According to (2.8), the unique solution  $x$  of the linear system (1.3), with  $A$  defined in (1.2), is decomposed as

$$x = \sum_{i=1}^p x_i \tag{2.9}$$

with

$$x_i = \sum_{k=1}^m \alpha_{i,k} x_{i,k} \tag{2.10}$$

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  $i$ ,

- ★ compute  $x_{i,1}$  solution of  $A_i x_{i,1} = b$ ,
  - ★ compute  $x_{i,2}$  solution of  $A_i x_{i,2} = x_{i,1}$ ,
  - ⋮
  - ★ compute  $x_{i,m}$  solution of  $A_i x_{i,m} = x_{i,m-1}$ ,
  - ★ lastly, compute  $x_i$  by (2.10).
- (2.11)

Computation of the solution  $x$  by (2.9) requires 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 (2.8) are given by

$$\alpha_{i,j} = \frac{1}{(m-j)!} \lim_{t \rightarrow 1/h_i} \left[ \frac{(t-h_i t)^m}{\varphi(t)} \right]^{(m-j)} \tag{2.12}$$

with

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

For  $m = 1$ , formulas (2.12) reduces to (2.7). For  $m = 2$ , there exist also simple explicit formulas :

$$A^{-1} = \sum_{i=1}^p \alpha_{i,1} A_i^{-1} + \alpha_{i,2} A_i^{-2} \tag{2.13}$$

with

$$\alpha_{i,2} = \prod_{j \neq i} \frac{1}{(1 - \frac{h_j}{h_i})^2}, \quad \alpha_{i,1} = -2\alpha_{i,2} \sum_{j \neq i} \frac{h_j}{h_i - h_j}. \quad (2.14)$$

For  $m > 2$  there is an alternate method to the computation of the  $mp$  coefficients  $\alpha_{i,k}$  by (2.12): solve the following linear system for well chosen  $mp$  real numbers  $t_\ell$

$$\sum_{i=1}^p \sum_{k=1}^m \frac{1}{(1 - h_i t_\ell)^k} \alpha_{i,k} = \frac{1}{\varphi(t_\ell)}, \quad 1 \leq \ell \leq mp. \quad (2.15)$$

In other words, compute the coefficients  $\alpha_{i,k}$  by interpolation of the function  $1/\varphi$  at the distinct nodes  $t_\ell$  (to be well chosen). 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

$$\begin{pmatrix} \psi^1(t_1) & \cdots & \psi^p(t_1) \\ \vdots & & \vdots \\ \psi^1(t_{mp}) & \cdots & \psi^p(t_{mp}) \end{pmatrix} \alpha = \begin{pmatrix} 1/\varphi(t_1) \\ \vdots \\ 1/\varphi(t_{mp}) \end{pmatrix}, \quad (2.16)$$

with

$$\psi^i(t_\ell) = \left( \frac{1}{1 - h_i t_\ell}, \dots, \frac{1}{(1 - h_i t_\ell)^m} \right).$$

This system, a Vandermonde-like one, is ill-conditioned. Since only the case  $m = 1$  was used in our numerical tests, we do not discuss further the linear system (2.16). Of course, preconditioning 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 [4]. 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  $x$ :

$$\|\tilde{x} - x\| \leq \left[ \sum_{i=1}^p |\alpha_i| \right] \varepsilon. \quad (2.17)$$

Since the first term, that amplifies the error, may highly increase with the parameter  $p$ , it is crucial to limit our investigation for small values of this parameter, see [4]. As an illustration, Table 1 displays the evolution of the amplification term as a function of  $p$ .

Table 1: Evolution of the amplification term in (2.17) as a function of the number of processors  $p$ . The  $h_i$  are chosen 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  $p=4$ ,  $h_i\in\{(1\pm c)h,(1\pm 2c)h\}, \dots$

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

### 3 The homogeneous case

As was stated earlier, Proposition 2.1 was used in [4] (in the case  $m=1$ ) to solve homogeneous linear time-evolution equation (2.1) discretized by a Finite Difference method. In this section, we reproduce the tests in [4] using a Finite Element scheme. One iteration of a Finite Element 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 (2.3) 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 (2.4)). Let  $T_0, T_f$  be the initial and final times. We define  $p$  time steps  $h_1, h_1, \dots, h_p$  et intermediate times  $t_n = T_0 + n \times$  (time steps) used in a cyclical order: firstly  $h_1$ , then  $h_2, \dots, h_p, \dots$

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

It is important to precise that our algorithm computes the solution only at times  $t_{kp}$ ,  $k=1, \dots, K$  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

$$\begin{aligned} & \star u_{kp+1} \text{ solution of } A_1 u_{kp+1} = u_{kp}, \\ & \star u_{kp+2} \text{ solution of } A_2 u_{kp+2} = u_{kp+1}, \\ & \quad \vdots \\ & \star u_{kp+p} \text{ solution of } A_p u_{kp+p} = u_{kp+p-1}, \end{aligned} \tag{3.1}$$

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

$$Au_{(k+1)p} = u_{kp} \tag{3.2}$$

with  $A = \prod_{i=1}^p A_i$  and solve this system using the decomposition (2.5)-(2.7).

Let us compare sequential versus parallel computations.

1. *Sequential solution.* The cost of the sequential solution obtained by procedure (3.1) is the cost of solving  $p$  linear systems
2. *Parallel solution.* The cost of the parallel procedure (2.5)-(2.7) is that of one linear system since 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 (2.9). 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 systems to compute  $x$  the solution of (3.1) with  $A_i$  replaced by  $A_i^m$ . The cost is that of solving  $mp$  linear systems.
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 (2.10).

The cost is that of solving  $m$  linear systems. Of course, there is an additional cost to take into account: the summing up the  $x_i$  to get  $x$  by (2.9).

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  $\Omega$  is the unit square, the initial condition 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 (1.2) 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 coefficients  $\alpha_i$ , see Table 1. As already mentioned, we limit the use of the method 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  $\frac{1}{p} \times \text{Speedup}$ .

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. Ideally, 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 (2.17) growth rapidly with  $p$  as shown in Table 1. We return to this issue in the next section.

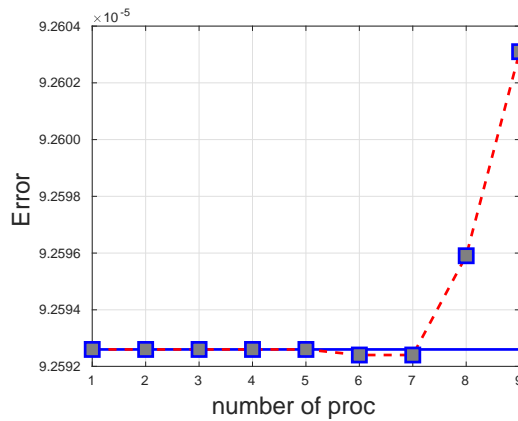


Figure 1: The 2D homogeneous heat equation.  $L^2$  error (numerical solution versus exact solution). Numerical parallel solution as a function of  $p$  that is also the number of processors. Solid line represents the error of the sequential solution.

## 4 The nonhomogeneous case

Consider the nonhomogeneous Differential Equation

$$\mathbf{u}_t = \mathcal{L}\mathbf{u} + \mathbf{f}. \tag{4.1}$$

One step in the Finite Element Method to solve (4.1) 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

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



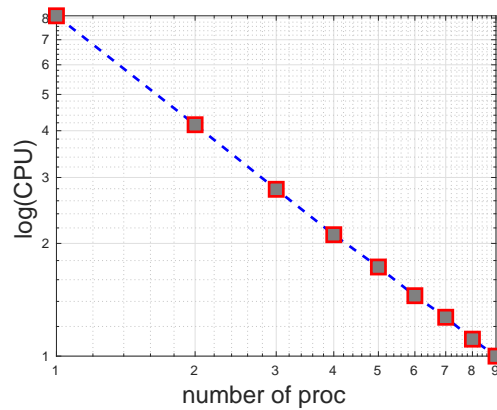


Figure 2: The 2D homogeneous heat equation. CPU time (log scale) as a function of the number of processors.

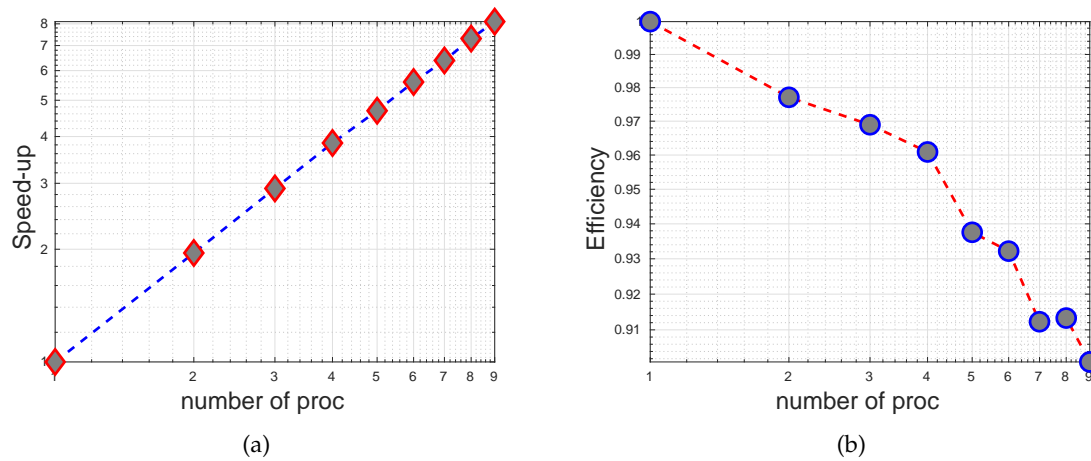


Figure 3: The 2D homogeneous heat equation. Speedup (left) and Efficiency (right) as a function of the number of processors.

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:

$$\star \Delta T = m \sum_{k=1}^p h_k,$$

- ★  $T_n = T_0 + n\Delta T$  and  $u_n$  the approximation of  $u(T_n)$ ,
  - ★  $T_{n,j} = T_n + m\sum_{k=1}^j h_k$  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

$$Au_{n+1} = u_n + g_n, \tag{4.2}$$

with

$$g_n = \sum_{k=1}^p h_k B_k \sum_{j=1}^m A_k^{j-1} f(T_{n,k-1} + jh_k), \tag{4.3a}$$

$$B_k = \prod_{\ell=1}^{k-1} A_\ell^m. \tag{4.3b}$$

The idea is to compute  $u^{n+1}$  using Proposition 2.1 to solve (4.2).

### 4.1 The algorithm

#### First version

1. For  $k=0, \dots, K_1 - 1$  ( $K_1\Delta T = T_{final}$ )
  - (a)  $u_k$  is known (it is an approximation of  $u$  at time  $T_k$ ).
  - (b) Compute  $g_k$  defined in (4.3).
  - (c) Compute  $u_{k+1}$  solution of (4.2).  
 To do so
    - i. compute in parallel  $x_i$  by (2.10) and (2.11) with  $b = u_k + g_k$ ,

- ii. make a “reduce” step to compute  $u_{k+1}$  by (2.9):

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

- (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  $T_k, k=1, \dots, K_1$  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  $\Delta T$ . Communications between the processors is done  $K_1$  times (during the summation process).

**Remark 4.1.** 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  $k=0, \dots, K_2-1$  ( $K_2 p \Delta T = T_{final}$ )

- (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  $i=1, \dots, p$ .  
 (c) Compute sequentially  $u_{kp+ipm}$  for  $i=1, \dots, p$  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 (2.10) and (2.11) with  $b = u_{kp+(i-1)pm} + g_{kp+(i-1)pm}$ ,  
 ii. make a “reduce” step to compute  $u_{kp+ipm}$  by (2.9):

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

- (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  $\Omega$  is the unit square. The source term  $f$  is chosen 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 6 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 procesors 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  $p$ .

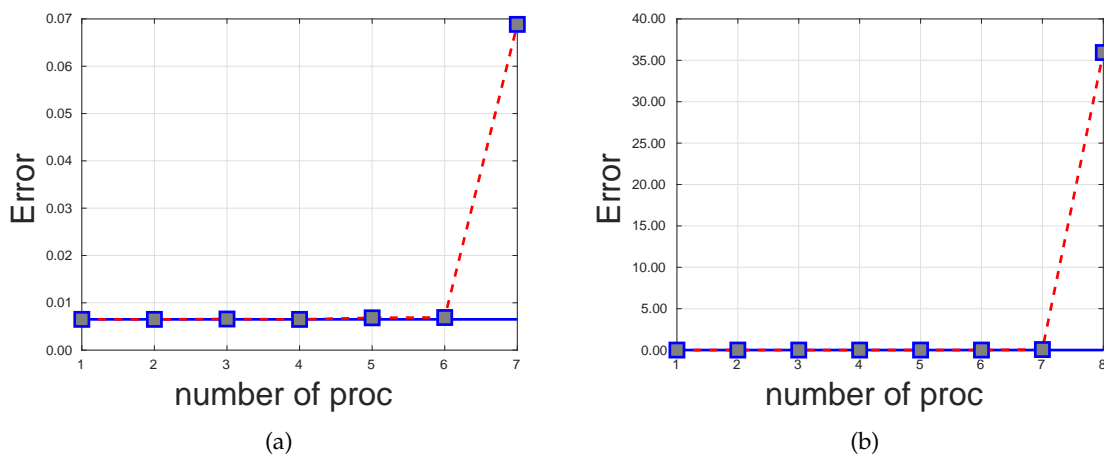


Figure 4: The 2D nonhomogeneous heat equation.  $L^2$  error (numerical solution versus exact solution). Numerical parallel solution as a function of  $p$  that is also the number of processors. Solid line represents the error of the sequential solution.

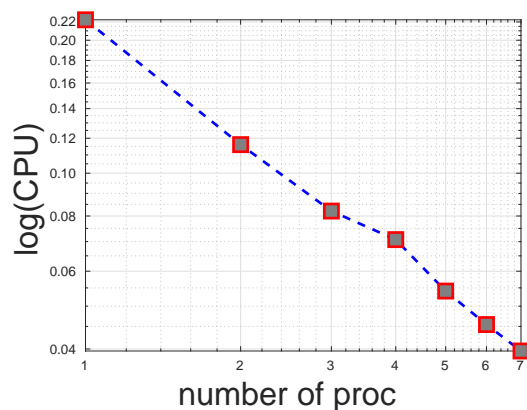


Figure 5: The 2D nonhomogeneous heat equation. CPU time (log scale) as a function of the number of processors.

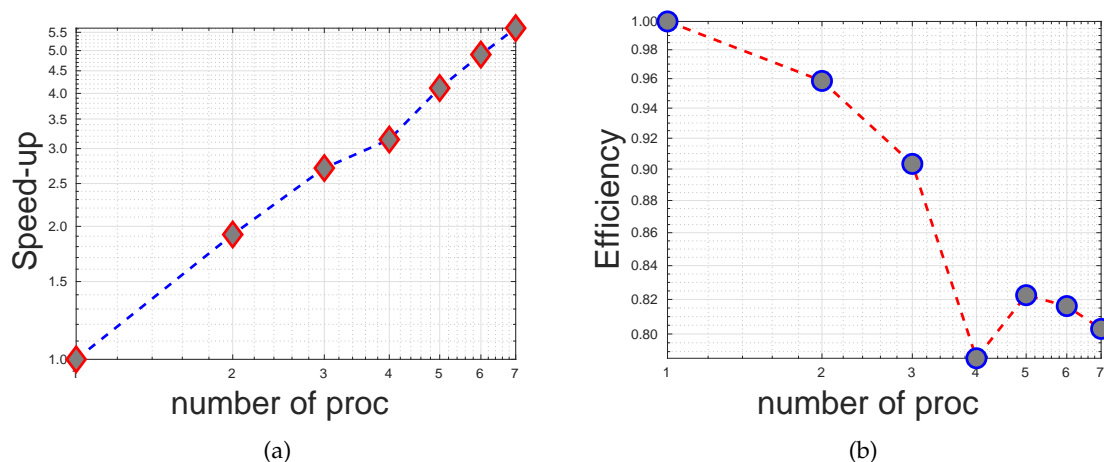


Figure 6: The 2D nonhomogeneous heat equation.  $L^2$  error (numerical solution versus exact solution). Numerical parallel solution as a function of  $p$  that is also the number of processors. Solid line represents the error of the sequential solution.

## 5 Conclusion

We have extended the work initiated in [4] 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 nonhomogeneous 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).

### References

- [1] 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.
- [2] F. Hecht. New development in FreeFem++. *J. Numer. Math.*, 20(3-4), 2012.
- [3] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numerica*, 19:209-286, 2010.
- [4] S.-M. Kaber, A. Loumi and P. Parnaudeau. Parallel solution of linear systems. *East Asian Journal on Applied Mathematics*, 6(3), 2016.
- [5] Y. Maday and E.M. Rønquist. Parallelization in time through tensor-product space-time solvers. *C. R. Math. Acad. Sci. Paris*, 346:1-2, 2008.