East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.210715.250316a

Vol. **6**, No. 3, pp. 278-289
August 2016

# Parallel Solution of Linear Systems

Sidi-Mahmoud Kaber[1,*], Amine Loumi[2] and Philippe Parnaudeau[1]

[1] *Sorbonne Universités, UPMC Univ Paris 06, CNRS, UMR 7598, Laboratoire Jacques-Louis Lions, 4, place Jussieu 75005, Paris, France.*
[2] *Laboratoire de Mathématiques et Applications, Université Hassiba Benbouali, Chlef, Algeria and Laboratoire d'Équations aux Dérivées Partielles non linéaires et d'Histoire des Mathématiques, ENS-Kouba, Algers, Algeria.*

*Received 21 July 2015; Accepted (in revised version) 25 March 2016.*

**Abstract.** Computational scientists generally seek more accurate results in shorter times, and to achieve this a knowledge of evolving programming paradigms and hardware is important. In particular, optimising solvers for linear systems is a major challenge in scientific computation, and numerical algorithms must be modified or new ones created to fully use the parallel architecture of new computers. Parallel space discretisation solvers for Partial Differential Equations (PDE) such as Domain Decomposition Methods (DDM) are efficient and well documented. At first glance, parallelisation seems to be inconsistent with inherently sequential time evolution, but parallelisation is not limited to space directions. In this article, we present a new and simple method for time parallelisation, based on partial fraction decomposition of the inverse of some special matrices. We discuss its application to the heat equation and some limitations, in associated numerical experiments.

**AMS subject classifications**: 65Y05, 65F05, 65F10

**Key words**: Parallel computation, numerical linear algebra.

## 1. Introduction

A major challenge in scientific computing is to decrease the time required in numerical simulations. To achieve this, algorithms and scientific software must be adapted to fit new computer architectures, often involving parallel programming paradigms. Parallelism is not a new topic in the High Performance Computing (HPC) community, but it has become important for all programmers from the beginning of the century when there were maybe 2 cores per processor, for there are typically 12 cores already and some $60-80$ cores are expected in the near future.

The Domain Decomposition Method (DDM) is an efficient approach to parallelisation in spatial directions, where the whole domain is subdivided to produce efficient iterative

*Corresponding author. *Email address:* kaber@ljll.math.upmc.fr (S.-M. Kaber)

 ©2016 Global-Science Press

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

and independent solutions of smaller problems on the resulting subdomains. Correction steps are necessary to propagate information from one subdomain to others. However, the time direction is also a candidate for parallelisation. In 1964, Nievergelt [1] introduced a parallel algorithm based on time decomposition, and a few years later Miranker *et al.* [2] defined a parallel solver based on a predictor-corrector scheme. Ref. [7] describes the state of the art on time parallel integration.

In Section 2, we present our Partial Differential Equation (PDE) solver involving time parallelisation, where the original linear system is split into uncoupled linear systems to be solved separately. Our method falls in the category of "Direct Parallel Time Integration" — cf. Refs. [3,5]. We eventually discuss its application to the two-dimensional heat equation in Section 3 and some limitations of our method in Section 4, and make brief concluding remarks in Section 5.

## 2. Parallel Computation for Linear Systems

Let $(A_i)_{i=1}^m$ be a collection of $m$ nonsingular real matrices of size $n \times n$ and set

$$X = A_1 \cdots A_m \, .$$

We are interested in the following problem: how to compute quickly the solution $x \in \mathbb{R}^n$ of the linear system

$$Xx = y \, , \tag{2.1}$$

where $y \in \mathbb{R}^n$ is any given vector?

**Sequential approach.** One should automatically eliminate the computation of the product of all $m$ matrices before solving the linear system, since this is very expensive (requires a large computational time). Let us recall that solving a generic $n \times n$ linear algebraic system by Gauss elimination ($n$ large) requires $n^3/3 + \mathcal{O}(n^2)$ operations. (For simplicity, we only take into account multiplications and divisions, neglecting additions and subtractions.) The product of two matrices of size $n \times n$ involves $n^3$ operations, so computation of the product $X$ above requires $mn^3$ multiplications. The following algorithm is a sensible sequential computation of the solution:

- compute the vector $x_1 \in \mathbb{R}^n$ such that $A_1 x_1 = y$ ;
- then compute $x_2$, the solution of $A_2 x_2 = x_1$ ;
- ...
- and ultimately compute $x_m = x$, the solution of $A_m x_m = x_{m-1}$ .

**Remark 2.1.** The computing cost of this algorithm is $m$ times the computing cost of solving one linear system — i.e. $mn^3/3$ operations.

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

**Parallel approach.** Suppose now that the nonsingular matrices $A_i$ have the specific form

$$A_i = I + h_i A\,, \tag{2.2}$$

where $(h_i)_{i=1}^m$ are $m$ distinct real numbers. Such matrices appear in the discretisation of differential equations, as in Section 3.2. In order to solve the linear system (2.1), we write the matrix $X^{-1}$ as a linear combination of matrices $A_i^{-1}$. In fact, we prove the following crucial result in the Appendix.

**Proposition 2.1.** *The inverse matrix of $X$ (for nonsingular matrices $A_i$) is*

$$X^{-1} = \sum_{i=1}^{m} \alpha_i A_i^{-1}\,, \tag{2.3}$$

*where*

$$\alpha_i = \prod_{k\neq i} (1 - h_k/h_i)^{-1}\ . \tag{2.4}$$

Accordingly, the unique solution $x$ of the linear system (2.1) is broken down into

$$x = \sum_{i=1}^{m} \alpha_i x_i\,, \tag{2.5}$$

where each vector $x_i$ is the unique solution of the linear system

$$A_i x_i = y\ .$$

There are thus *m independent* linear systems to solve. The main objective is to carry out the computation of each $x_i$ on a different processor (we make no distinction here between "process" and "processor", since we associate to each processor only one process).

**Remark 2.2.** Computing $x$ this way is as expensive as solving a single linear system (neglecting the $\alpha_i x_i$ summation in (2.5)). However, we note that in practice the interconnections between processors are far from negligible — cf. Section 3.2.

## 3. Applications

### 3.1. Linear system

For the above linear system where $n = 10\,000$, in Fig. 1 we compare the elapsed times for each of the two methods (sequential and parallel) as a function of $m$, the number of matrices. In the parallel computations, $m$ is also the number of processors used for the computation. The matrix $A$ used in the test is the celebrated one-dimensional finite differences discretisation of the Laplace operator with Dirichlet boundary conditions. The real numbers $h_i$ are $0.16 + 8k$, $k = 0, 1, 2, \cdots, 8$. The computations were performed with `Matlab`, using the `parfor` command. Fig. 1 displays the computation times for each of the

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

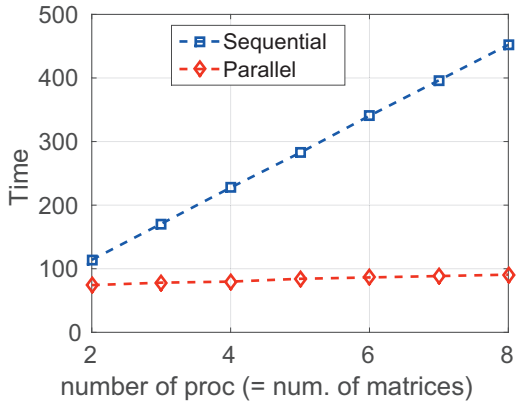



Figure 1: Computation (elapsed) time for the sequential method and the parallel method as a function of the number of processors ($n = 10\,000$).

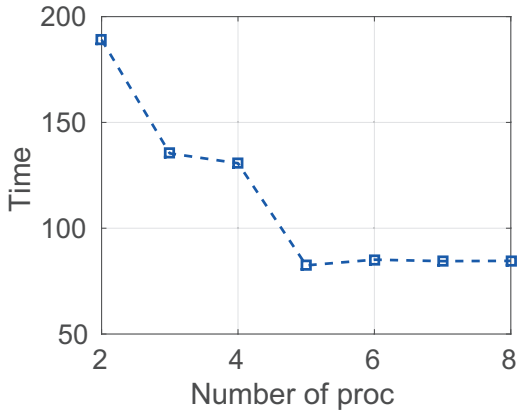



Figure 2: Computation time versus the number of processors ($m = 5$, $n = 10\,000$).

two methods as a function of the number of processors. It is clear that the parallel method is much more efficient than the sequential, with the difference between the elapsed times becoming larger as $m$ increases — e.g. for $m = 5$, the parallel computation becomes 3.3 times faster than the sequential one (about 60% more efficient).

For $n = 10\,000$ and $m = 5$, Fig. 2 shows the elapsed time for the parallel computation as a function of the number of processors, and we observe the computational time initially decreases as the number of processors increases. In particular, it is notable that the computational time with 5 processors is 43% of the computational time using 2 processors.

## 3.2. Ordinary differential equations

Let us consider the system of linear ordinary differential equation

$$\mathbf{u}'(t) = A\mathbf{u}(t) \tag{3.1}$$

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

subject to the initial condition $\mathbf{u}(0) = \mathbf{u}_0$, where the matrix $A \in \mathscr{M}_n(\mathbb{R})$ and the prime denotes differentiation with respect to the time $t$. We discretise Eq. (3.1) with an implicit method (backward Euler method)

$$(I - \delta t_n A)\,\mathbf{u}^{n+1} = \mathbf{u}^n\,.$$

The matrices of these systems are of the form (2.2) and are nonsingular (at least, for small $\delta t_n$). We choose a time-step $\delta t$ and define $m$ distinct time-steps $\delta t_i$ (fixing $m = 4$):

$$\delta t_1 = h_1 \delta t, \quad \delta t_2 = h_2 \delta t, \quad \delta t_3 = h_3 \delta t, \quad \delta t_4 = h_4 \delta t.$$

**Remark 3.1.** Note that the assumption "all the time steps are different" was also necessary in Ref. [6] to design a direct time parallel solver based on a tensor product, to diagonalise a matrix where the eigenvalues are the time steps.

Our goal is the following: from the knowledge of $\mathbf{u}^{4n}$, compute $\mathbf{u}^{4(n+1)}$ in one time step $m$ times faster than using a sequential approach. Let us recall what a sequential procedure would be:

**Sequential procedure**

- Compute successively the solution $\mathbf{u}^{4n+j}$ (for $j = 1, \cdots, 4$) of

$$A_j \mathbf{u}^{4n+j} = \mathbf{u}^{4n+j-1}\,,$$

with $A_j = (I + \delta t_j A)$.

Noting that

$$X\,\mathbf{u}^{4(n+1)} = \mathbf{u}^{4n}$$

with $X = A_1 A_2 A_3 A_4$, from Proposition 2.1 we have

$$\mathbf{u}^{4(n+1)} = X^{-1}\mathbf{u}^{4n} = \sum_{i=1}^{m} \alpha_i x_i\,,$$

where each $x_i$ is the solution of a linear system

$$A_i x_i = \mathbf{u}^{4n}\,,$$

leading to the following parallel procedure we propose.

**Parallel procedure**

- Given $u^{4n}$, for $i = 1, \cdots, m$ compute the solution $x_i$ of

$$A_i x_i = u^{4n}\,.$$

- Compute $u^{4(n+1)}$ from

$$u^{4(n+1)} = \sum_{i=1}^{m} \alpha_i x_i\,. \tag{3.2}$$

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

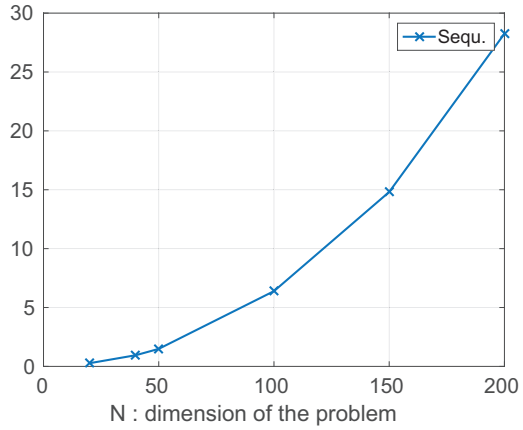



Figure 3: Elapsed time for the sequential case vs the problem size $N$.

**Application to the heat equation.** Let $\Omega = [0,1] \times [0,1]$ be the unit square and $T > 0$ the final time of the simulation. The heat equation in primitive variables is:

$$\begin{aligned} &\partial_t u - \Delta u = f \text{ in } (0,T) \times \Omega \,, \\ &u(x,t) = 0 \text{ on } (0,T) \times \Gamma \,, \\ &u(x,0) = u_0(x,y) \text{ in } \Omega \,, \end{aligned} \tag{3.3}$$

where $\Gamma$ is the boundary of $\Omega$ and $u_0(x,y)$ the initial data. Dirichlet boundary conditions are imposed on $\Gamma$. Suppose the Laplacian is discretised by a second-order central finite difference scheme on an isotropic uniform grid with mesh size

$$h = \delta x = \delta y = \frac{1}{N} \,,$$

where the discretisation parameter $N$ is the number of points used in each direction, and a backward Euler method is adopted for time integration. The initial condition assumed in our numerical experiments is $u_0(x,y) = \sin(\pi x)\sin(\pi y)$.

Fig. 3 shows the evolution of the elapsed time as a function of the parameter $N$ in the sequential computation. The computation cost increases dramatically with $N$. Let us now turn to a parallel computation where we use the Message Passing Interface (MPI) [4] for the parallel communications, although other parallel programming paradigms could also be used (OpenMP, Threads, OpenCL, OpenACC). Fig. 4 presents the parallel elapsed time as a function of the number of MPI processors, and for several values of the discretisation parameter $N$. The horizontal solid line represents the sequential computational time. For small values of $N$ (viz. 20, 40 and 50) the computational time decreases as the number of processors grows. However, starting from $N = 100$ we observe that using two processors instead of one increases the CPU time — there is an extra cost is due to communication between the processors. Indeed, Eq. (3.2) imposes a global communication requirement between all of the processors in order to compute and share the updated solutions $u^{4(n+1)}$, and this cost increases with the size of the problem.

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

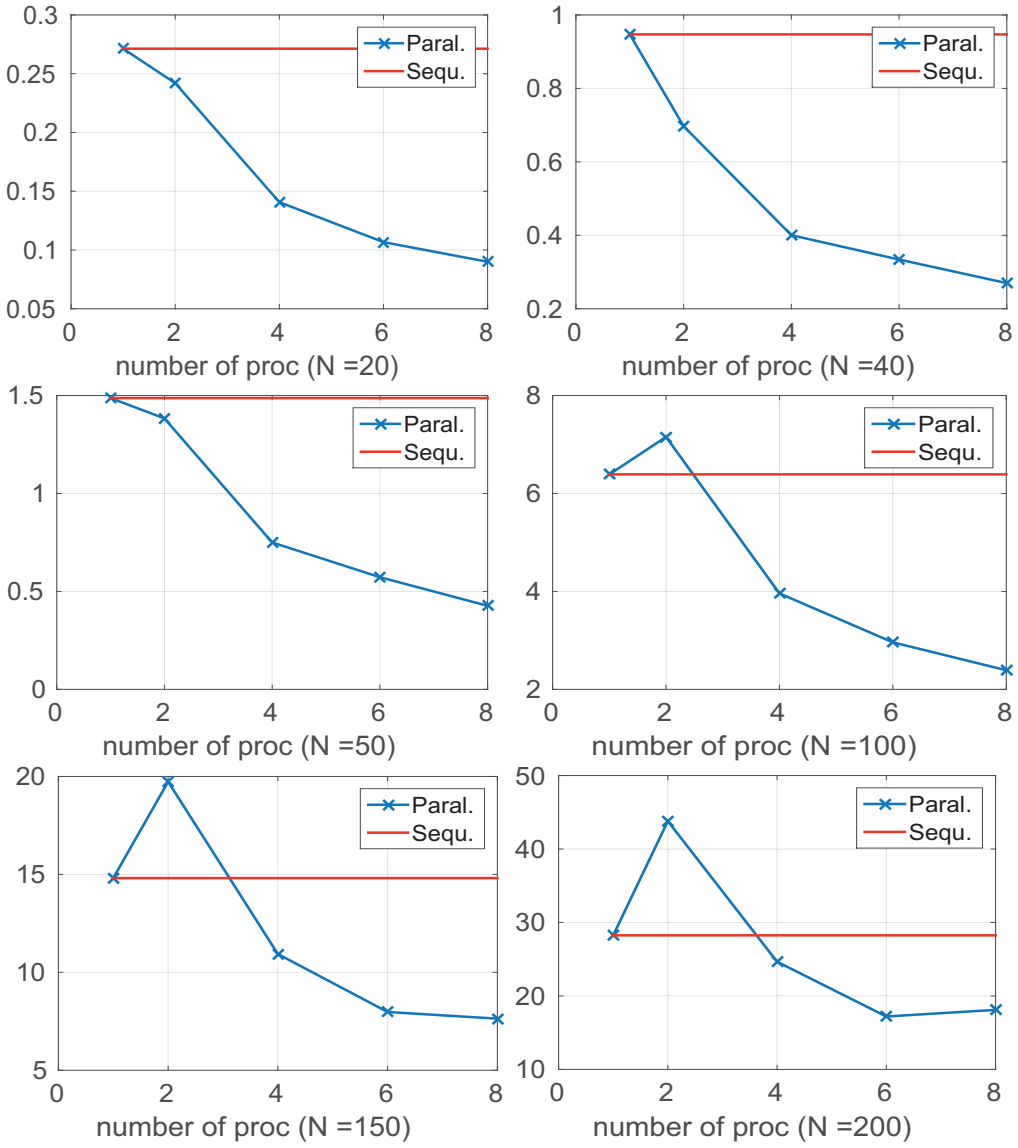



Figure 4: Evolution of elapsed time as a function of the number of processors, compared with the sequential case for $N \in \{20, 40, 50, 100, 150, 200\}$.

Let us introduce some concepts related to the computation times:

- the sequential time to compute the solution at $T = mK\delta t$ for a problem of size $N$ is

$$f(1, T, N) = f(1, mK\delta t, N) = mKf(1, \delta t, N)\,;$$

- the total time to solve a problem of size $N$ using $m$ processors is

$$f(m, T, N) = f(1, K\delta t, N) + KC(m, N)\,,$$

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

where $C(m,N)$ is the cost of communications between $m$ processors sharing a data of size $N$.

Two quantities of interest in parallel computing are the speedup (the ratio of the sequential time to the parallel one) and the efficiency (speedup divided by the number of processors):

- Speedup
$$S(m,T,N) = \frac{f(1,T,N)}{f(m,T,N)} = m\frac{f(1,\delta t,N)}{f(1,\delta t,N)+C(m,N)} .$$
- Efficiency
$$E(m,T,N) = \frac{1}{m}S(m,T,N) = \frac{f(1,1,N)}{f(1,1,N)+C(m,N)} .$$

Ideally, the communications would have no cost, leading to the speedup and efficiency

$$S^*(m,T,N) = m , \qquad E^*(m,T,N) = 1 ,$$

but the communication time $C(m,N)$ is not negligible. To have a clear idea of the efficiency, one should know how the key quantity $C(m,N)/f(1,\delta t,N)$ evolves as $m$ and $N$ vary, but this is a hardware dependent problem outside the scope of this study.

Fig. 5 shows the speedup for several runs. For small values of $N$, we obtain a speedup larger than 3 for moderate values of the problem size, illustrating the advantage of our parallelisation strategy, although the speedup is only 1.5 for the largest values of $N$. The efficiency is displayed in Fig. 6. For the first values of $N$, we observe good efficiency (larger than 0.4) — and although the efficiency is lower (0.2) for other values of $N$, it nevertheless remains almost constant as the number of processors increases. In the next section, we discuss some limitations of our method.

## 4. Limitations of the Parallel Computational Method

Our main goal is to compute the solutions faster using parallel rather than sequential processing, but we now identify two notable issues that affect the accuracy of our method.

- The assumption that "the time steps are all different" is a barrier to good accuracy, since constant times steps minimise the discretisation error — cf. [5]). However, this can be overcome somewhat by using a geometric time step $\delta t_k = \varrho^k \delta t$ [6].
- Another issue in practical situations is that the solution $x_i$ of the linear system $A_i x_i = y$ is only known through an approximation $\tilde{x}_i$ with an error $\varepsilon_i$:
$$\|\tilde{x}_i - x_i\| \le \varepsilon_i ,$$
from which we deduce that an upper bound on the error on the solution $\mathbf{x}$ is
$$\|\tilde{\mathbf{x}} - \mathbf{x}\| \le (\max_{1\le i\le m} \varepsilon_i) \sum_{i=1}^{m} |\alpha_i| ,$$

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

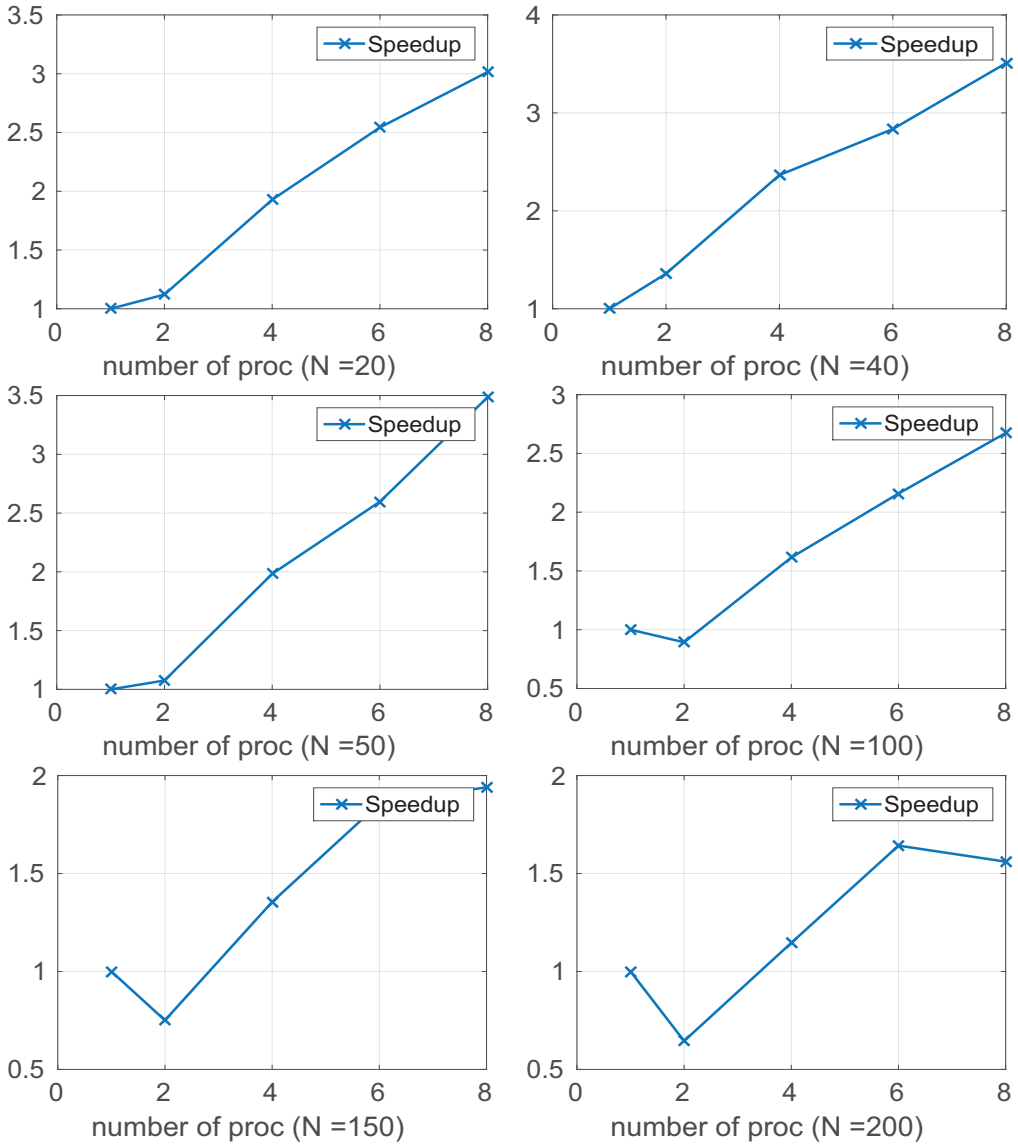



Figure 5: Speedup for $N \in \{20, 40, 50, 100, 150, 200\}$ as a function of the number of processors.

which may be very large — e.g. in the simple case where $h_i = ih$, we have

$$\sum_{i=1}^{m} |\alpha_i| \geq \alpha_m = \frac{m^{m-1}}{(m-1)!} \simeq C^{ste} \frac{e^m}{\sqrt{m}},$$

and some effort is required to compute the vectors $x_i$ very accurately. Moreover, the coefficients $\alpha_i$ may not only be very large but also may not have the same sign, so special emphasis should be placed on computing the summation of the $x_i$ in Eq. (2.5).

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

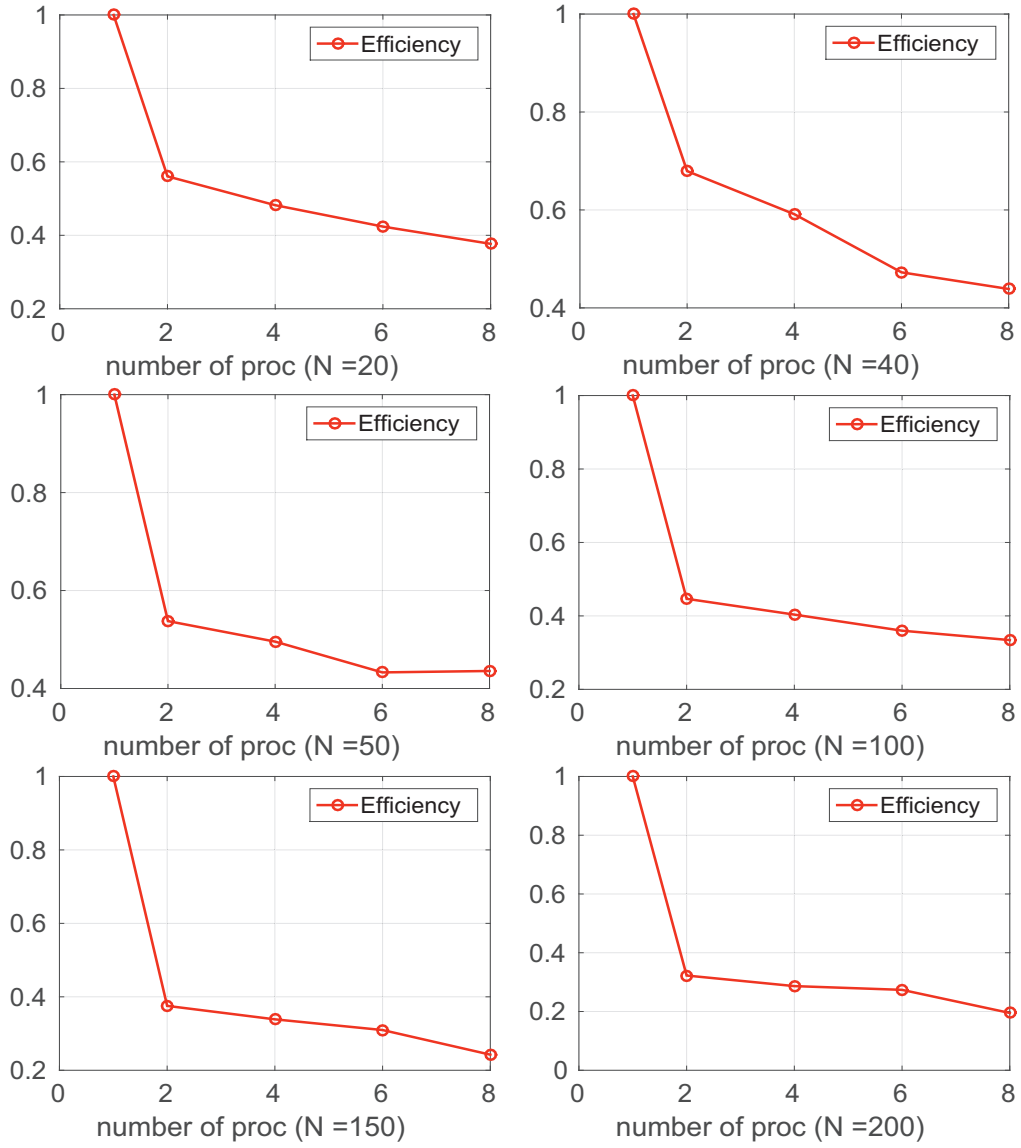



Figure 6: Efficiency for $N \in \{20, 40, 50, 100, 150, 200\}$ as a function of the number of processors.

## 5. Conclusion

We have presented a new method to solve linear time-dependent systems using a fractional decomposition of the inverse of the matrix resulting from the discretisation of the time evolution operator. The method has been used to solve the two-dimensional heat equation, and we obtain quite good results. However, it appears that our method should be limited to a moderate number of processors for acceptable precision and stability. We are currently investigating further developments, including also implementing space domain

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

decomposition in the solution of other partial differential equations.

## Appendix: Proof of Proposition 2.1

**Lemma A.1.** *Recall that $(h_i)_{i=1}^m$ are m distinct real numbers. The real numbers $\alpha_i$ defined in Eq.* (2.4) *and the $\beta_{i,j}$ defined by*

$$\prod_{k\neq i}(1+h_k t) = \sum_{j=0}^{m-1} \beta_{i,j} t^j$$

*are such that*

$$\sum_{i=1}^{m} \alpha_i = 1 \tag{A.1}$$

*and*

$$\sum_{i=1}^{m} \alpha_i \beta_{i,j} = \begin{cases} 1\,, & \text{for } j=0 \\ 0\,, & \text{for } j=1,\cdots,m\,. \end{cases} \tag{A.2}$$

*Proof.* The $m$ Lagrange polynomials

$$\omega_i(t) = \alpha_i \prod_{k\neq i}(1+h_k t)$$

of degree $m-1$ satisfy $\omega_i(-1/h_j) = 0$ for $j \neq i$ and $\omega_i(-1/h_i) = 1$. The sum of these polynomials

$$p : t \longmapsto \sum_{i=1}^{m} \omega_i(t)$$

is constant, equal to 1. Indeed, the polynomial $q = p-1$ of degree $m-1$ has $m$ distinct zeros $-1/h_j$, hence it is the null polynomial and in particular $q(0) = 0$. This proves the first part of the Lemma. We then use the identity $p \equiv 1$ to establish the second part, which follows from the identity

$$1 = p(t) = \sum_{i=1}^{m} \omega_i(t) = \sum_{j=0}^{m-1} \left\{ \sum_{i=1}^{m} \alpha_i \beta_{i,j} \right\} t^j\,. \quad \square$$

**Proof of Proposition 2.1.**

Recall that the $A_i = I + h_i A$ are nonsingular matrices. Now if matrix $A$ is either diagonal or diagonalisable, the identity in Proposition 2.1 is such that for all real numbers $t \neq -1/h_k$ we have

$$\frac{1}{\prod_{i=1}^{m}(1+h_i t)} = \sum_{i=1}^{m} \alpha_i \frac{1}{1+h_i t}\,,$$

https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press

the partial fraction decomposition of a rational fraction. Using the $\omega_i$ defined in the proof of the Lemma above, we have

$$\frac{1}{\prod_{i=1}^{m}(1+h_i t)} = \sum_{i=1}^{m} \frac{\omega_i(t)}{\prod_{k=1}^{m}(1+h_k t)} = \sum_{i=1}^{m} \frac{\alpha_i}{1+h_i t} \, .$$

Further, even if $A$ is not diagonalisable the result remains true. By analogy with the scalar case, let us define the polynomial of matrices

$$\Omega_i(A) = \alpha_i \prod_{k \neq i} A_k \, ,$$

and the following identity holds:

$$\sum_{i=1}^{m} \Omega_i(A) = I \, . \tag{A.3}$$

Indeed, using (A.2) we have

$$\sum_{i=1}^{m} \Omega_i(A) = \sum_{i=1}^{m} \alpha_i \sum_{j=0}^{m-1} \beta_{i,j} A^j = \sum_{j=0}^{m-1} \left\{ \sum_{i=1}^{m} \alpha_i \beta_{i,j} \right\} A^j = I \, ,$$

and the proof of the Proposition 2.1 is just an application of this identity:

$$X^{-1} = \sum_{i=1}^{m} X^{-1} \Omega_i(A) = \sum_{i=1}^{m} \alpha_i A_i^{-1}. \quad \square$$

## References


[1] J. Nievergelt, *Parallel methods for integrating ordinary differential equations*, Commun. ACMachinery **7**, 731-733 (1964).

[2] W.L. Miranker and W. Liniger, *Parallel methods for the numerical integration of ordinary differential equations*, Math. Comput. **21**, 303-320 (1967).

[3] J.-L. Lions, Y. Maday and G. Turinici, *Résolution d'EDP par un schéma en temps "pararéel"*, C. R. Acad. Sci. Paris Sér. I Math. **7**, 332, 661-668 (2001).

[4] Message Passing Interface Forum, http://www.mpi-forum.org.

[5] M.J. Gander, L. Halpern and T.T.B. Tran, *A direct solver for time parallelization*, Domain Decomposition Methods in Science and Engineering XXII, Lecture Notes in Comp. Sci. and Eng., Springer (2015).

[6] Y. Maday and E.M. Rønquist, *Parallelization in time through tensor-product space-time solvers*, C. R. Math. Acad. Sci. Paris **1-2**, 346, 113-118, (2008).

[7] M.J. Gander, *50 Years of Time Parallel Time Integration*, in *Multiple Shooting and Time Domain Decomposition*, pp. 69-114, T. Carraro, M. Geiger, S. Körkel and R. Rannacher (Eds.), Springer (2015).


https://doi.org/10.4208/eajam.210715.250316a Published online by Cambridge University Press