Non stationary wave relaxation methods for general linear systems of Volterra equations: convergence and parallel GPU implementation

Conte Dajana[1], Cuesta Eduardo[2*] and Valentino Carmine[3]

[1]DIPMAT, University of Salerno, Via Giovanni Paolo II, Fisciano, 84084, Italy.

[2*]Dept. Matematica Aplicada, University of Valladolid, Campus Miguel Delibes, Valladolid, 47011, Spain.

[3]DIIN, University of Salerno, Via Giovanni Paolo II, Fisciano, 84084, Italy.

*Corresponding author(s). E-mail(s): eduardo.cuesta@uva.es; Contributing authors: dajconte@unisa.it; cvalentino@unisa.it;

Abstract

In the present paper a parallel–in–time discretization of linear systems of Volterra equations of type

$$\bar{\boldsymbol{u}}(\boldsymbol{t}) = \bar{\boldsymbol{u}}_{\mathbf{0}} + \int_0^t \mathbf{K}(\boldsymbol{t}-\boldsymbol{s})\bar{\boldsymbol{u}}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s} + \bar{\boldsymbol{f}}(\boldsymbol{t}), \qquad \mathbf{0} < \boldsymbol{t} \leq \boldsymbol{T},$$

is addressed. Related to the analytical solution a general enough functional setting is firstly stated. Related to the numerical solution a parallel numerical scheme based on the Non Stationary Wave Relaxation (NSWR) method for the time discretization is proposed, and its convergence is studied as well. A CUDA parallel implementation of the method is carried out in order to exploit Graphics Processing Units (GPUs), which are nowadays widely employed for reducing the computational time of several general purpose applications. The performance of these methods is compared to some sequential implementation. It is revealed throughout several experiments of special interest in practical applications the good performance of the parallel approach.

Keywords: Parallel-in-time, Volterra equations, GPU, Wave relaxation

1 Introduction

Time discretization of convolution equations typically gives rise to discrete convolutions whose practical computation turns out typically to be very costly, in particular if compared to time discretizations for ordinary differential equations. Moreover, if one considers a system of convolution equations instead, the problem gets even worse, this is why in this context, more than ever, efficient and fast algorithms are critical.

This paper is concerned with parallel implementations of time discretization for $D \times D$ linear systems of Volterra equations, $D \in \mathbb{Z}^+$, of the form

$$\bar{u}(t) = \bar{u}_0 + \int_0^t \mathbf{K}(t-s)\bar{u}(s)\,\mathrm{d}s + \bar{f}(t), \qquad 0 < t \leq T, \tag{1}$$

where $\bar{u}_0 \in X^D$ stands for the initial data, $\bar{f} : [0,T] \mapsto X^D$ is a given vector–valued function, X typically stands for $\mathbb{R}$, and $\{\mathbf{K}(t)\}_{t\geq 0}$, represents a time–dependent family of $D \times D$ matrix–valued operators.

Besides the computational cost time discretization itself carries out, rather frequently the system (1) turns out to be very large, that is $D >> 0$ which makes the practical evaluation even harder. We might mention as a typical example within this framework those linear systems of equations (1) arising from the spatial discretization of linear Volterra equations

$$u(t,x) = u_0(x) + \int_0^t K(t-s)u(s,x)\,\mathrm{d}s + f(t,x), \qquad t > 0, \tag{2}$$

where $x \in \mathbb{R}^d$, $d = 1, 2$, or 3, and $\{K(t)\}_{t\geq 0}$ stands for a time–dependent family of linear operators defined in a suitable functional setting. Certainly, related to the spatial discretization of those equations, the finer the spatial mesh, the larger the system is, that is the larger the computational cost is.

Consider as a prototype equation arising in the context of (2), might be the simplest one one may consider, the linear equation whose convolution kernel adopts the form

$$K(t) = k(t)A, \qquad t > 0, \tag{3}$$

where $k(t)$ stands for a scalar function, and A is a linear operator. Very often in practical instances the operator A in (3) turns out to be the Laplacian operator $A = \Delta$ in $\mathbb{R}^n$, $n = 1, 2$ or 3, or a fractional Laplacian $(-\Delta)^\beta$, $0 < \beta < 1$ (see [1]), both of them joint with suitable boundary conditions, or merely $A \in \mathcal{M}_{D\times D}(\mathbb{R})$ (find more examples in [2]). Related to the scalar functions $k(t)$ let us mention those defining time fractional integrals/derivatives. Recall for instance and among many others definitions of fractional integrals /derivatives, the Riemann–Liouville time fractional integrals of order $\alpha > 0$ whose associated kernel reads

$$k(t) := \begin{cases} \dfrac{t^{\alpha-1}}{\Gamma(\alpha)}, & \text{if } t > 0, \\ 0, & \text{if } t \leq 0. \end{cases} \tag{4}$$

This kind of equations are nowadays attracting the interest of many researchers (see [2–4] and references therein, among many others).

To go more in–depth in that definition, its associated definition of fractional derivative, and other definitions of fractional integrals/derivatives, there is a vaste literature among which we refer the reader to [4–7] and references therein. In this regard let also mention those definitions extending the non–integer fractional integrals of constant order $\alpha > 0$ to the ones whose fractional order depends on time, i.e. $\alpha(t)$ instead of a constant α [8, 9]. This kind of integrals/derivatives are discussed more precisely in Section 2.

Let us highlight that prototype equations we are considering show a singular behavior as t tends to 0^+ which gives rise to singularities in the solution of the corresponding equations. Notice that most times singular Volterra equations reflect much more accurately the memory effect in many practical cases, instead of the regular ones. Unfortunately their solutions require a more careful study and this fact has to be taken into account in the numerical analysis.

On the other hand the numerical solutions of (1) has been investigated by a large number authors. Of particular interest are the discretizations based on convolution quadratures due mainly to the good stability properties [10–13, 15–17]; the collocation methods provide also a good balance between stability and accuracy [18–22]; or the ones based on the numerical inversion Laplace transform [23], among many others.

However the computational cost these implementations carry out when (1) stands for a large system of equations represents even today a great difficulty in practical instances, therefore raising faster and more efficient implementations is nowadays a great challenge. This issue is usually addressed, at least, in two different ways. On the one hand faster and more efficient numerical methods have been proposed in the literature, see e.g. [22, 24–27] in the framework of convolution quadratures. The second one consists of performing faster and more efficient implementations in the context of parallel computing. In the present paper we focus on the last one.

Parallel computing has emerged in last decades as a very helpful tool to speed up computational processes in particular the ones involved in numerical analysis, all of them once adapted to this architecture. However parallel architectures are very expensive even at present, instead modern hardware technologies allow parallel computing at much lower cost, in fact we are speaking of Graphics Processing Units (GPUs). The GPUs based hardware was initially designed for graphical purposes, however over the last few years it turned into a very useful tool in the framework of scientific computing due to their low cost, and their computational capabilities [28]. Parallel computing within GPUs context has allowed to speed up algorithms for systems of Ordinary Differential Equations (ODEs) [29–31]; Partial Differential Equations (PDEs) [32–34]; and the ones we are interested in the present paper, that is the systems of Volterra Integral Equations (VIEs) [35, 36, 36–40]; among others. In this point we have to highlight most of cites above related to VIEs are concerned with convolution integrals of fractional type.

The aim of this paper consists of extending some of previous approaches within parallel computing framework from systems of linear integral equations of fractional type to a more general class of systems of linear convolution equations. In other words beyond the systems of fractional integral equations, in the present paper we consider systems of nonlocal in time equations whose convolution kernels merely satisfy the existence of its Laplace transform in certain half–complex plane. That includes a large list of kernels and equations, some of them more precisely described in Section 2. The parallel approach considered in this paper in based on the Non–Stationary Wave Relaxation Methods (NSWR) introduced in [41]. In this regard since several truncation errors are involved in the iterative method the study goes accompanied by an accurate analysis of the convergence which allows one to ensure the well behavior of such algorithms in practical instances. The performance of these methods is illustrated with several numerical experiments applied to some of the systems of equations proposed in Section 2.

The paper is organized as follows. Section 2 is devoted to state the framework of the present work, both from the continuous and the discrete point of view. In Section 3 we describe precisely the notation and the raised method, and in Section 4 we present our main contributions of this paper, that is the theoretical analysis related to the convergence of the proposed algorithms. Section 5 is devoted to explain the technical resources involved in the implementation of algorithms, and in Section 6 several numerical experiments show the good performance of these. Section 7 includes the most relevant conclusions of the present and future works.

2 Framework statement

2.1 Time continuous setting

First of all we state the notation we will use throughout the following sections. Let us consider $X = \mathbb{R}$, and the system of linear Volterra equations (1) where $\bar{u}, \bar{f} : [0,T] \mapsto \mathbb{R}^D$, $D \in \mathbb{Z}^+$ (which is expected to be large), $\mathbb{R}^D$ is normed by $\|\cdot\|$ (we set below what $\|\cdot\|$ stands for: L_2–norm, sup norm,...), $\{\mathbf{K}(t)\}_{t\geq 0} \subset \mathcal{M}_{D\times D}(\mathbb{R})$, that is a time–dependent family of matrices, and $\bar{u}_0 \in \mathbb{R}^D$ represents the initial data. In order to keep the initial data $\bar{u}_0$ from apart of the source term $\bar{f}$, one may assume without loss of generality that $\bar{f}(0) = \bar{0}$.

Before going further in the paper we state conditions for the well–posedness of (1). In this regard consider the wide class of time dependent operators

$\{\mathbf{K}(t)\}_{t\geq 0}$ element–wise of exponential growth, that is those for which there exist $b, \beta > 0$ such that if $\mathbf{K}(t) = (k_{i,j}(t))_{1\leq i,j\leq D}$, then

$$|k_{i,j}(t)| \leq b\,\mathrm{e}^{-\beta t}, \qquad t > 0. \tag{5}$$

Notice that for the sake of the simplicity of the notation, and without loss of generality, we assume the same constants b and $\beta > 0$ for each entry of $\mathbf{K}(t)$. This means that $\mathbf{K}(t)$, for $t > 0$, admits absolutely convergent element–wise Laplace transform in $\mathcal{D}_{\mathbf{K}} := \{z \in \mathbb{C} : \mathrm{Re}(z) \geq \beta\}$, denoted now and hereafter by $\widetilde{\mathbf{K}}(z) = (\widetilde{k}_{i,j}(z))_{1\leq i,j\leq D}$, $z \in \mathcal{D}_{\mathbf{K}}$, being $\tilde{k}_{ij} = \mathcal{L}(k_{ij}(z))$. This fits the time dependent operators of the form $\mathbf{K}(t) = k(t)A$ where $k(t)$ is a function of exponential growth, and A stands for a matrix, e.g. the one corresponding to some classical spatial discretization of the Laplacian. If in addition $\bar{f} : (0, +\infty) \to \mathbb{R}$ is continuous, then (1) is well–posed [2]. Some assumptions might be lightened, however this framework is general enough for our purposes.

Apart from the model commented in Section 1 based on the Riemann–Liouville fractional integral, let us consider other examples fitting this context and whose implementations will serve to illustrate the performance of our methods.

1. Denote $\alpha : [0,T] \to \mathbb{R}^+$ a positive function, $\alpha(t) > 0$, whose Laplace transform exists and is denoted by $\widetilde{\alpha}(z)$, for $z \in \mathcal{D} \supseteq \mathcal{D}_{\mathbf{K}}$. Define $k : [0,T] \mapsto \mathbb{R}$, the kernel associated to the fractional integral with order varying in time $\alpha(t) > 0$, according the following definition

$$\partial^{-\alpha(t)} f(t) := (k * f)(t) = \int_0^t k(t-s) f(s)\,\mathrm{d}s, \quad t > 0, \tag{6}$$

where

$$k(t) := \mathcal{L}^{-1}(K)(t) \quad \text{with} \quad K(z) := \frac{1}{z^{z\tilde{\alpha}(z)}}, \qquad z \in D. \tag{7}$$

Notice that if $\alpha(t)$ is constant, then definitions (4) and (6)–(7) completely agree. Consider now the equation (2) where $K(t) = k(t)\Delta$, and Δ stands for the 1–dimensional Laplacian operator in $[0, L]$, with homogenous Newmann boundary conditions. Given $D > 0$, and the mesh grid $x_0 = 0 < x_1 < x_2 < \ldots < x_D = L$, $x_j = jh$, $0 \leq j \leq D$ and $h = L/D$, the spatial discretization of (2) by means of a classical second order finite differences scheme gives rise to a finite dimensional operator A which is nothing but a $(D+1) \times (D+1)$ three–diagonal matrix. The Volterra equation (1) arises now with $\mathbf{K}(t) = k(t)A$, and $\bar{f}$ standing for the function $f : [0,T] \to \mathbb{R}^D$ sampled in the spatial grid. Moreover $\bar{u}$ stands for the approximation to the time continuous solution over the spatial grid, that is $\bar{u}(t) = (u_j(t))_{0\leq j\leq D}$ where $u_j(t) \approx u(x_j, t)$, $0 \leq j \leq D$.

Other definitions of fractional derivatives/integrals with order depending on time have been proposed in the literature. Some of these definitions have been discussed in [9] concluding that (6)–(7) might be considered as the most convenient in many practical instances. This is why we adopt such a definition in this paper.

2. A kind of generalization of (3)–(4) comes out in the framework image processing [3]. In fact consider the two dimensional Laplacian in $[0, L] \times [0, L]$, denoted again (if not confusing) by Δ, with homogeneous Newmann boundary conditions again. In [3] the Laplacian applies to $u(x, y, t)$ in the variables x, and y, and $u : [0, L] \times [0, L] \times [0, T] \mapsto \mathbb{R}$ represents an original gray scale image $u_0(x, y)$ evolved up to the time level $t > 0$ to be $u(x, y, t)$. Once one discretizes in space by a classical second order finite difference scheme with $D + 1$ nodes along each coordinate axis, we have the mesh grid $\{(x_i, y_j)\}_{0\le i,j\le D} = \{(ih, jh)\}_{0\le i,j\le D}$, $h = L/D$, and that $u(x, y, t)$, $0 \le x, y \le L$, becomes into a $(D + 1) \times (D + 1)$ matrix $(u_{i,j}(t))_{0\le i,j\le D}$ where the entries represent the approximations to the exact solution, that is $u_{i,j}(t) \approx u(x_i, x_j, t)$, for $t > 0$, $0 \le i, j \le D$. Therefore if one reshapes such a matrix as a vector by stacking columns, then the discrete Laplacian, denoted again by A, reads as a $(D + 1)^2 \times (D + 1)^2$ five–diagonal matrix, and $(u_{i,j}(t))_{0\le i,j\le D}$ as a $(D + 1)^2 \times 1$ vector.

   One of the novelties of this model is that it relates image processing and fractional calculus and allows to apply a different diffusion rate over every single pixel (x_i, y_j). Such a rates are handled by a particular fractional kernel $k_{i,j}(t)$ of type (4) with order $\alpha_{i,j} > 0$, $0 \le i, j \le D$. In [3] the authors also consider the case of fractional orders depending on time, that is orders $\alpha_{i,j}(t)$, $0 \le i, j \le D$, $t > 0$, according to the definition (6)–(7).

   Anyhow that approach leads to a system of linear equations of type (1) where $\mathbf{K}(t) = I(t)A$, $I(t) =\text{diag}(k_j(t))_{0\le j\le D^2}$, $t \ge 0$, $\alpha_j(t)$ are defined according to (6)–(7), A stands for the $(D + 1)^2 \times (D + 1)^2$ five–diagonal matrix mentioned above, and $\bar{u}_0$ stands for the initial data $u(x, y, 0)$ sampled in the spatial mesh, that is the original image to be restored sampled on the spatial mesh.

2.2 Time discrete setting

In Section 1 we said that a large variety of numerical schemes for the time discretization of Volterra equations (1) exists in the literature. However in this paper we focus on the convolution quadratures based methods introduced by Ch. Lubich in [15].

In particular a first order numerical scheme is considered here and the reason is twofold. On the one hand the lack of time regularity observed in the solutions of most of singular Volterra equations has proved as a barrier to consider higher order time discretization. In particular, for fractional equations of type (2)–(4)), it is very well known that the second order of convergence is reachable if $1 < \alpha < 2$, but it is not obvious at all if the equation involves some non–linear term. Even worse if $0 < \alpha < 1$ for which the second order is hardly reached [13]. The same applies to the solutions of the equations with fractional order depending on time (6)–(7), whose regularity is precisely studied in [9]. Apart from that the choice of a first order numerical scheme does not yield a loss of generality in our study, on the contrary the study might be straightforwardly extended to higher order numerical schemes if the regularity of the corresponding continuous solutions is high enough.

Therefore we adopt the backward Euler based convolution quadrature method as the time discretization for (1). Since the stability, convergence, as well as some other properties, has been already studied [14–27], we here merely recall its formulation. In fact let $\tau > 0$ be the time step of the discretization, $\tau = T/N$, and $t_n = n\tau$, for $0 \le n \le N$. Moreover denote by $q(z)$ the quotient of the generating polynomials of the associated multistep linear method, in the case of the backward Euler method simply $q(z) = 1 - z$. According to the above the time discretization for (1) reads

$$\bar{u}_n = \bar{u}_0 + \sum_{j=0}^{n} \mathbf{K}_{n-j}\bar{u}_j + \bar{f}_j, \qquad 0 \le n \le N, \tag{8}$$

where $\bar{u}_n \in \mathbb{R}^D$ stands for the approximation to $\bar{u}(t_n)$, $\bar{f}_n := \bar{f}(t_n) \in \mathbb{R}^D$, $0 \le n \le N$, and the convolution weights $\mathbf{K}_j \in \mathcal{M}_{D\times D}(\mathbb{R})$ are element–wise given by the expansion

$$\widetilde{\mathbf{K}}\left(\frac{q(z)}{\tau}\right) = \sum_{n=0}^{+\infty} \mathbf{K}_n z^n. \tag{9}$$

For example, if $\mathbf{K}(t) = k(t)A$ with $k(t) = t^{\alpha-1}/\Gamma(\alpha)$, $\alpha > 0$, and $A \in \mathcal{M}_{D\times D}(\mathbb{R})$, then the convolution weights take the form

$$\mathbf{K}_n = k_n A, \tag{10}$$

where

$$k_n = \tau^\alpha \binom{\alpha - n + 1}{n} = \tau^\alpha \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!}, \quad n \ge 1 \quad (k_0 = 1). \tag{11}$$

Let us highlight that in spite of the computation of convolution weights (9) may be directly done by means of the theoretical expression (10) this task requires a very high computational cost. Instead in practical instances that computation is performed by means of the Fast Fourier Transform (FFT), not only for the kernels of fractional type but also for the kernels which Laplace transform is reachable as the ones we consider in this paper.

For the convenience of readers we recall a result related to the convolution weights (9) which will be very useful in the proofs of our results (see Corollary 3.2 in [16]). In fact there exists $C > 0$ depending on $\mathbf{K}$ and T but independent on τ and n, such that the convolution weights in (9) hold the bound

$$\|\mathbf{K}_n\| \le C\tau, \qquad 1 \le n \le N. \tag{12}$$

Finally notice that, without loss of generality, now and hereafter we consider $\mathcal{M}_{D\times D}(\mathbb{R})$ normed by the matrix sup norm denoted if not confusing merely by $\|\cdot\|$.

3 Discrete NSWR methods for linear Volterra equations

The Non–Stationary Wave Relaxation (NSWR) methods for integral equations of type

$$y(t) = y_0 + \int_0^t k(t,s,y(s))\,\mathrm{d}s + f(t), \qquad t>0, \tag{13}$$

have been introduced in [41] and consist of iterative methods that provide a time–dependent sequence $\{y^\eta(t)\}_{\eta\ge 0}$, throughout the difference equation

$$y^{(\eta)}(t) = y_0 + \int_0^t \mathcal{F}^{(\eta)}(t,s,y^{(\eta-1)}(s),y^{(\eta)}(s))\,\mathrm{d}s + f(t), \qquad t>0, \quad \eta=1,2,\ldots, \tag{14}$$

where the set of functions $\mathcal{F}^{(\eta)}:\mathbb{R}^+\times\mathbb{R}^+\times\mathbb{R}\times\mathbb{R}\to\mathbb{R}$ satisfy the equality

$$\mathcal{F}^{(\eta)}(t,s,y,y) = k(t,s,y), \qquad \eta=1,2,\ldots. \tag{15}$$

The following facts have been already noticed in [36]. Firstly it has been observed that if one applies the iterative process over the whole time interval, then the convergence of the method slows noticeably down as the number of time nodes grows up. To overcome this problem the solution adopted in [36], and the one we adopt here, consists of splitting the whole interval $[0,T]$ in sub–intervals/windows of length $b\tau$, $b\in\mathbb{Z}^+$, so that

$$[0,T] = \bigcup_{j=0}^{R-1}[t_{jb}, t_{(j+1)b}] = \bigcup_{j=0}^{R-1}[jb\tau, jb\tau+b\tau], \quad \text{with} \quad Rb\tau = T. \tag{16}$$

The iterative method (14) reads now, for $t\in(rb\tau, rb\tau+b\tau]$,

$$y^{(\eta)}(t) = y_0 + \int_0^{rb\tau} k(t,s,y(s))\,\mathrm{d}s + \int_{rb\tau}^t \mathcal{F}^{(\eta)}(t,s,y^{(\eta-1)}(s),y^{(\eta)}(s))\,\mathrm{d}s + f(t), \tag{17}$$
$$\eta = 1,2,\ldots$$

In the case of the system (1), the method (17) reads

$$\bar{u}^{(\eta)}(t) = \bar{u}_0 + \int_0^{rb\tau} \mathbf{K}(t-s)\bar{u}(s)\,\mathrm{d}s + \int_{rb\tau}^{t} \mathcal{F}^{(\eta)}(t-s, \bar{u}^{(\eta-1)}(s), \bar{u}^{(\eta)}(s))\,\mathrm{d}s + \bar{f}(t), \tag{18}$$

for $\eta = 1, 2, \ldots$, where, by the simplicity of the notation, and having in mind that the equations we are considering in the present paper are of convolution type, we have replaced $\mathcal{F}^{(\eta)}(t, s, \bar{u}, \bar{v})$ by $\mathcal{F}^{(\eta)}(t-s, \bar{u}, \bar{v})$, that is $\mathcal{F}^{(\eta)}$ is now a function $\mathcal{F}^{(\eta)} : \mathbb{R}^+ \times \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^D$. The condition (15) then reads

$$\mathcal{F}^{(\eta)}(t-s, \bar{u}, \bar{u}) = \mathbf{K}(t-s)\bar{u}, \qquad 0 < s < t, \quad \eta = 1, 2, \ldots \tag{19}$$

Therefore we discretize (17) according to the numerical scheme (8)–(9) in the time mesh $\{t_n\}_{1\le n\le N}$, giving rise to the discrete NSWR method

$$\bar{u}_n^{(\eta)} = \bar{u}_0 + \sum_{j=0}^{rb} \mathbf{K}_{n-j}\bar{u}_j + \sum_{j=rb+1}^{n} \mathcal{F}_{n-j}^{(\eta)}(\bar{u}_j^{(\eta-1)}, \bar{u}_j^{(\eta)}) + \bar{f}_n, \qquad \eta \ge 1, \tag{20}$$

for $t_n \in (rb\tau, rb\tau + b\tau]$. Once again for the sake of the simplicity of the notation we have replaced $\mathcal{F}^{(\eta)}(t_n - t_j, \bar{u}_j^{(\eta-1)}, \bar{u}_j^{(\eta)})$ by $\mathcal{F}_{n-j}^{(\eta)}(\bar{u}_j^{(\eta-1)}, \bar{u}_j^{(\eta)})$.

The iterative method (20) provides at every single time step $t_n = n\tau$ a sequence of vectors $\{\bar{u}_n^{(\eta)}\}_{\eta\ge 0}$, for which the number of iterations required to achieve the stated accuracy at time level t_n will depend at every single time level solely on the last windows but not on the time step t_n itself. That is, for the r–th window, $0 \le r \le R-1$, there will be a threshold $m_r > 0$ so that the iterative method reaches the required accuracy with at most m_r iterations. Therefore we adopt $\bar{u}_n^{(m_r)}$ as the numerical approximation to $\bar{u}(t_n)$, that is $\bar{u}_n^{(m_r)} \approx \bar{u}(t_n)$, for every $t_n \in (t_{(r-1)b}, t_{rb}]$, $1 \le n \le N$, and $1 \le r \le R$.

A convenient choice of the operators $\mathcal{F}_j^{(\eta)} : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}^D$, for $0 \le j \le N$, and $\eta \ge 0$, will allow the parallelization of the method by decoupling the system at every single time level t_n into two independent systems. These operators will be required to comply two hypotheses:

[H1] All operators are Lipchitz in both variables, that is there exists $L > 0$ such that

$$\|\mathcal{F}_j^{(\eta)}(\bar{u}, \cdot) - \mathcal{F}_j^{(\eta)}(\bar{v}, \cdot)\| \le L\|\bar{u} - \bar{v}\|, \quad \|\mathcal{F}_j^{(\eta)}(\cdot, \bar{u}) - \mathcal{F}_j^{(\eta)}(\cdot, \bar{v})\| \le L\|\bar{u} - \bar{v}\|, \tag{21}$$
$$\forall \bar{u}, \bar{v} \in \mathbb{R}^D,$$

for $0 \le j \le N$ and $\eta \ge 0$. Without lost of generality, we assume a common Lipchitz constant $L > 0$ for both variables.

[H2] For $0 \le j \le N$, and $\eta \ge 0$, there holds the equality

$$\mathcal{F}_j^{(\eta)}(\bar{u}, \bar{u}) = \mathbf{K}_j\bar{u}, \qquad \forall \bar{u} \in \mathbb{R}^D, \quad 0 \le j \le N, \quad \eta \ge 0, \tag{22}$$

where $\{\mathbf{K}_j\}_{j\geq 0}$ stand for the quadrature weights defined in (9).

For instance, the family of linear operators $\{\mathcal{F}_j^{(\eta)}\}_{1\leq j\leq N,\eta\geq 0}$, defined by

$$\mathcal{F}_j^{(\eta)}(\bar{u},\bar{v}) = N_j^{(\eta)}\bar{u} + M_j^{(\eta)}\bar{v}. \tag{23}$$

is probably the simplest one satisfying [H1], and for a convenient choice of $N_j^{(\eta)}$ and $M_j^{(\eta)}$ also [H2]. In particular, we focus our attention on the discrete NSWR (20) where

$$\mathcal{F}_j^{(\eta)}(\bar{u},\bar{v}) = N_j^{(\eta)}\bar{u} + M_j^{(\eta)}\bar{v}, \qquad \eta\geq 0, \tag{24}$$

with

$$M_j^{(\eta)} = \mu_j^{(\eta)} I \quad \text{and} \quad N_j^{(\eta)} = \mathbf{K}_j - M_j^{(\eta)}, \qquad 0\leq j\leq N, \quad \eta\geq 0, \tag{25}$$

for certain values $\{\mu^{(\eta)}\}_{\eta\geq 0}$ to be determined later, and where I is the identity matrix in $\mathbb{R}^D$. In our approach we avoid the dependence on j of $\mu_j^{(\eta)}$ so we have $\mu^{(\eta)}$ instead of $\mu_j^{(\eta)}$, and we might write $M^{(\eta)}$ instead of $M_j^{(\eta)}$.

Finally, consider the windows length equal to τ, that is $b = 1$. In this case the discrete NSWR method is so–called discrete time–point (or time–step) NSWR method and reads

$$\bar{u}_n^{(\eta)} = \bar{u}_0 + \sum_{j=0}^{n-1} \mathbf{K}_{n-j}\bar{u}_j + \mathcal{F}^{(\eta)}(\bar{u}_n^{(\eta-1)}, \bar{u}_n^{(\eta)}) + \bar{f}_n, \quad \eta\geq 1. \tag{26}$$

Observe that since $\mathcal{F}_j$ does not depend in fact on the sub–index j and solely there appears $\mathcal{F}_0^{(\eta)}$, we denoted this operator simply as $\mathcal{F}^{(\eta)}$. Moreover, in the same manner $N_j^{(\eta)}$ and $M_j^{(\eta)}$ do not depend on j and are denoted by $N^{(\eta)}$ and $M^{(\eta)}$ respectively. The time–step NSWR we propose reads as

$$\bar{u}_n^{(\eta)} = \bar{u}_0 + \sum_{j=0}^{n-1} \mathbf{K}_{n-j}\bar{u}_j + (K_0 - \mu^{(\eta)}I)\bar{u}_n^{(\eta-1)} + \mu^{(\eta)}I\bar{u}_n^{(\eta)} + \bar{f}_n, \quad \eta\geq 1. \tag{27}$$

The class of iterative methods defined by (23)–(26) is known as Richardson discrete time–point NSWR methods, and they are the subject of our study in this paper.

Before ending this section several comments have to be made. On the one hand, if $\mu^{(\eta)} = 0$, for all η, then the iterative method (23)–(26) reduces to the functional iteration method.

Moreover, in spite of we here consider a time–point NSWR method, the results and the idea of the proofs straightforwardly extends for larger windows, $b > 1$.

Finally observe that the choice (24) allows decoupling the $D \times D$ system (26) into D linear equations which can be independently solved, or in other words this allows the parallelization of the algorithm by solving independently D decoupled equations.

4 Main results

The first issue to be observed is that, as typically occurs with iterative methods, the method (23)–(26) does not allow to reach the exact solution $\bar{u}_n$ at any single step t_n within a finite number of iterations, even in the case of the exact solution could be theoretically achieved. It is mainly due to the round–off errors. But not only the round–off errors are the matter, even if the exact solution can be theoretically achieved in a finite number of iterations, such a number may be so large that it makes the computational effort unfordable, in other words the required number of iterations might be so large as to make the computational advantages of the parallelization approach negligible. Keeping this in mind assume that, for $1 \leq n \leq N$, we merely will have an approximation $\bar{u}_n^{(m_n)}$ instead of the exact value $\bar{u}_n$. The numerical solution $\bar{u}_n^{(m_n)}$ will stand for the true numerical approximation to $\bar{u}(t_n)$, $1 \leq n \leq N$, under a previously stated tolerance.

Observe that in the case of the Richardson time–step NSWR methods the number of iterations in each window obviously coincides with the number of iterations for every single time level, namely our case.

The first result of this section concerns the form the error takes for the numerical scheme (23)–(26) so firstly denote the error

$$\bar{e}_n^{(\eta)} := \bar{u}_n - \bar{u}_n^{(\eta)}, \qquad 1 \leq n \leq N, \quad \eta \geq 0. \tag{28}$$

Denote also

$$P_\eta(x) = \prod_{r=1}^{\eta} (x - \mu^{(r)}), \qquad \eta \geq 0, \tag{29}$$

where $\{\mu^{(r)}\}_{r \geq 0}$ is the set of values involved in (24). These values are discussed below. Admit (with abuse of the notation) that the evaluation of $P_\eta(x)$ may be matrix–wise done. Therefore we have the following lemma.

Lemma 1 The error (28) yielded by the numerical scheme (23)–(26) writes as

$$\bar{e}_n^{(\eta)} = A^{(\eta)} \left\{ B^{(\eta)} \sum_{j=1}^{n-1} \mathbf{K}_{n-j} \bar{e}_j^{(m_j)} + \bar{e}_n^{(0)} \right\}, \qquad \eta \geq 1, \tag{30}$$

where,

$$A^{(\eta)} = \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)}, \qquad B^{(\eta)} = \sum_{j=1}^{\eta} P_{j-1}(1) P_j(\mathbf{K}_0)^{-1}, \quad \eta \geq 1. \tag{31}$$

Proof The proof straightforwardly follows by solving some difference equations. We consider first the case $n=1$, for $\eta \geq 0$. From (28) and (24)–(25) we have

$$\begin{aligned}\bar{e}_1^{(\eta)} &= \{\bar{u}_0 + \mathbf{K}_1\bar{u}_0 + \mathbf{K}_0\bar{u}_1\} - \{\bar{u}_0 + \mathbf{K}_1\bar{u}_0 + \mathcal{F}^{(\eta)}(\bar{u}_1^{(\eta-1)}, \bar{u}_1^{(\eta)})\} \\ &= \mathbf{K}_0\bar{u}_1 - \{N^{(\eta)}\bar{u}_1^{(\eta-1)} + M^{(\eta)}\bar{u}_1^{(\eta)}\}.\end{aligned}$$

Definition (25), and adding and subtracting $\mathbf{K}_0\bar{u}_1^{(\eta)}$, lead to

$$\begin{aligned}\bar{e}_1^{(\eta)} &= \{\mathbf{K}_0\bar{u}_1 - \mu^{(\eta)}\bar{u}_1^{(\eta)}\} - (\mathbf{K}_0 - \mu^{(\eta)}I)\bar{u}_1^{(\eta-1)} \\ &= \mathbf{K}_0\{\bar{u}_1 - \bar{u}_1^{(\eta)}\} + \{\mathbf{K}_0 - \mu^{(\eta)}I\}\bar{u}_1^{(\eta)} - \{\mathbf{K}_0 - \mu^{(\eta)}I\}\bar{u}_1^{(\eta-1)}.\end{aligned}$$

Finally, adding and subtracting $N^{(\eta)}\bar{u}_1$ leads to

$$\begin{aligned}\bar{e}_1^{(\eta)} &= \mathbf{K}_0\bar{e}_1^{(\eta)} - N^{(\eta)}\bar{e}_1^{(\eta)} + N^{(\eta)}\bar{e}_1^{\eta-1} \\ &= M^{(\eta)}\bar{e}_1^{(\eta)} + N^{(\eta)}\bar{e}_1^{(\eta-1)}.\end{aligned}$$

Therefore, recursively and by definitions (25) it follows that

$$\begin{aligned}\bar{e}_1^{(\eta)} &= \left(I - M^{(\eta)}\right)^{-1} N^{(\eta)}\bar{e}_1^{(\eta-1)} \\ &= \frac{1}{1-\mu^{(\eta)}}\left(\mathbf{K}_0 - \mu^{(\eta)}I\right)\bar{e}_1^{(\eta-1)} \\ &\vdots \\ &= \left\{\prod_{j=1}^{\eta}\frac{1}{1-\mu^{(j)}}\left(\mathbf{K}_0 - \mu^{(j)}I\right)\right\}\bar{e}_1^{(0)} \\ &= \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)}\bar{e}_1^{(0)}.\end{aligned}$$

For $n>1$ and $\eta \geq 0$, in a similar manner as for $n=1$, we have again from (28) and (24)–(25)

$$\begin{aligned}\bar{e}_n^{(\eta)} &= \{\bar{u}_0 + \mathbf{K}_n\bar{u}_0 + \mathbf{K}_{n-1}\bar{u}_1 + \mathbf{K}_{n-2}\bar{u}_2 + \cdots + \mathbf{K}_1\bar{u}_{n-1} + \mathbf{K}_0\bar{u}_n\} \\ &\quad -\{\bar{u}_0 + \mathbf{K}_n\bar{u}_0 + \mathbf{K}_{n-1}\bar{u}_1^{(m_1)} + \mathbf{K}_{n-2}\bar{u}_2^{(m_2)} + \cdots + \mathbf{K}_1\bar{u}_{n-1}^{(m_{n-1})} + \mathcal{F}^{(\eta)}(\bar{u}_n^{(\eta-1)}, \bar{u}_n^{(\eta)})\} \\ &= \sum_{j=1}^{n-1}\mathbf{K}_j(\bar{u}_{n-j} - \bar{u}_{n-j}^{(m_{n-j})}) + \mathbf{K}_0\bar{u}_n - \{N^{(\eta)}\bar{u}_n^{(\eta-1)} + M^{(\eta)}\bar{u}_n^{(\eta)}\}\end{aligned}$$

Now, adding and subtracting $\mathbf{K}_0\bar{u}_n^{(\eta)}$ first, then adding and subtracting $N^{(\eta)}\bar{u}_n$, joint with Definition (25) lead to

$$\begin{aligned}\bar{e}_n^{(\eta)} &= \sum_{j=1}^{n-1}\mathbf{K}_j\bar{e}_{n-j}^{(m_{n-j})} + \mathbf{K}_0(\bar{u}_n - \bar{u}_n^{(\eta)}) + (\mathbf{K}_0 - \mu^{(\eta)}I)\bar{u}_n^{(\eta)} - (\mathbf{K}_0 - \mu^{(\eta)}I)\bar{u}_n^{(\eta-1)} \\ &= \sum_{j=1}^{n-1}\mathbf{K}_j\bar{e}_{n-j}^{(m_{n-j})} + \mathbf{K}_0\bar{e}_n^{(\eta)} - N^{(\eta)}\bar{e}_n^{(\eta)} + N^{(\eta)}\bar{e}_n^{(\eta-1)} \\ &= \sum_{j=1}^{n-1}\mathbf{K}_j\bar{e}_{n-j}^{(m_{n-j})} + M^{(\eta)}\bar{e}_n^{(\eta)} + N^{(\eta)}\bar{e}_n^{(\eta-1)}, \qquad (32)\end{aligned}$$

where $\bar{e}_j^{(m_j)}$ stands for the error of the iterative process after m_j iterations once reached the threshold, at time step t_j, $1 \le j \le N$. Therefore from (32) we have

$$\begin{aligned}
\bar{e}_n^{(\eta)} &= \left(I - M^{(\eta)}\right)^{-1} \left\{ \sum_{j=1}^{n-1} \mathbf{K}_j \bar{e}_{n-j}^{(m_{n-j})} + N^{(\eta)} \bar{e}_n^{(\eta-1)} \right\} \\
&= \frac{1}{1-\mu^{(\eta)}} \left\{ \sum_{j=1}^{n-1} \mathbf{K}_j \bar{e}_{n-j}^{(m_{n-j})} + \left(\mathbf{K}_0 - \mu^{(\eta)} I\right) \bar{e}_n^{(\eta-1)} \right\}. \qquad (33)
\end{aligned}$$

The solution of the difference equation (33) straightforwardly writes

$$\begin{aligned}
\bar{e}_n^{(\eta)} &= \left\{ \sum_{j=1}^{\eta} \frac{1}{1-\mu^{(j)}} \left(\prod_{k=j+1}^{\eta} \frac{1}{1-\mu^{(k)}} \left(\mathbf{K}_0 - \mu^{(k)} I\right) \right) \right\} \sum_{j=1}^{n-1} \mathbf{K}_j \bar{e}_{n-j}^{(m_{n-j})} \\
&\quad + \left(\prod_{j=1}^{\eta} \frac{1}{1-\mu^{(j)}} \left(\mathbf{K}_0 - \mu^{(j)} I\right) \right) \bar{e}_n^{(0)} \\
&= \left\{ \sum_{j=1}^{\eta} \frac{P_{j-1}(1)}{P_\eta(1)} P_\eta(\mathbf{K}_0) P_j(\mathbf{K}_0)^{-1} \right\} \sum_{j=1}^{n-1} \mathbf{K}_j \bar{e}_{n-j}^{(m_{n-j})} + \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)} \bar{e}_n^{(0)} \\
&= \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)} \left\{ \left\{ \sum_{j=1}^{\eta} P_{j-1}(1) P_j(\mathbf{K}_0)^{-1} \right\} \sum_{j=1}^{n-1} \mathbf{K}_j \bar{e}_{n-j}^{(m_{n-j})} + \bar{e}_n^{(0)} \right\}.
\end{aligned}$$

Therefore the statement of the lemma follows. □

In view of the error expression (30)–(31), it can be observed that operator $\mathbf{K}_0$ plays a key role, and note also that $\mathbf{K}_0$ expresses as

$$\mathbf{K}_0 = \widetilde{\mathbf{K}}\left(\frac{q(0)}{\tau}\right) = \widetilde{\mathbf{K}}\left(\frac{1}{\tau}\right), \qquad (34)$$

in particular if $\{\mathbf{K}(t)\}_{t\ge 0}$ are the operators defined in (10), then $\mathbf{K}_0 = \tilde{k}(1/\tau)A = \tau^\alpha A$.

This suggests that the choice of the parameters $\{\mu^{(r)}\}_{r\ge 0}$ in (29) must be related to the eigenvalues of $\mathbf{K}_0$ as in fact does occur in our approach. In this regard denote the spectrum of $\mathbf{K}_0$ as

$$\sigma(\mathbf{K}_0) := \{\lambda_r \in \mathbb{C} : \lambda_r \text{ is an eigenvalue of } \mathbf{K}_0, 1 \le r \le D\} \subset \mathbb{C}, \qquad (35)$$

and assume that they are sorted in increasing order of the absolute value. For the sake of the simplity of the notation and without loss of generality all of them with multiplicity one, that is $0 \le \|\lambda_1\| < \|\lambda_2\| < \|\lambda_3\| < \ldots < \|\lambda_D\|$. Therefore let us consider in (29)

$$\mu^{(r)} = \lambda_r, \qquad \lambda_r \in \sigma(\mathbf{K}_0), \quad \text{for} \quad 1 \le r \le D. \qquad (36)$$

Notice that in that case the iterative method (26) reaches the exact solution, at least theoretically, in a finite number of iterations since

$$P_j(\mathbf{K}_0) = 0 \in \mathcal{M}_{D\times D}, \quad \text{if} \quad j \geq D. \tag{37}$$

However, as we have discussed above, in spite of the exact solution may theoretically be achieved after D iterations, since we are expecting that D is going to be quite large, that property will be useless from the practical point of view due to the high computational cost carried out by the large number of iterations required. But even though the number of iterations is from the computational point of view acceptable, hardly ever the exact solution will be reached due to the cumulative round–off errors.

That is why a compromise solution is typically adopted: To give up the exact solution of the iterative method, instead we opt for a strategy to obtain an approximate solution within an acceptable computational effort. In this regard we fix a threshold expected to be reached at every time step t_n within a number of iterations let say m_n expected to be much smaller than D, and giving rise to the numerical approximation we are looking for.

Let us highlight that some authors made use of Chebyshev polynomials and the *Minimax* property they enjoy [35, 36, 40, 41]. This approach consists of taking the set $\{\mu^{(r)}\}_{r\geq 0}$ as the roots associated the Chebyshev polynomial re–scaled to the interval bounded by the eigenvalues of $\mathbf{K}_0$ with minimum and maximum absolute value, let say λ_1 and λ_D, that is

$$\mu^{(r)} := \frac{\lambda_D - \lambda_1}{2} \cos\left(\frac{(2r-1)\pi}{2D}\right) + \frac{\lambda_D + \lambda_1}{2}, \qquad 1 \leq r \leq D. \tag{38}$$

Our approach does not allow the use of the *Minimax* property if this choice is made. Moreover it has a disadvantage: The choice (36) requires the computation of the whole spectrum of $\mathbf{K}_0$ which for a large D may be highly costly, in spite of this such a computation is certainly done once for all, or in the particular case of the one dimensional Laplacian the spectrum is explicitly known. On the contrary the choice (38) only requires the eigenvalues λ_1 and λ_D which is much less expensive.

Since the proof of the main result follows the same ideas for both choices, for the sake of the simplicity of presentation we opted in this paper for (36). Anyhow the presence of a memory term in (30) makes in both cases new assumptions necessary in order to reach the error stability. In fact, if the choice (36) is made, then the new assumption is related to $\sigma(\mathbf{K}_0)$ and reads

[H3] The spectrum of $\mathbf{K}_0$ is located in $\mathbb{C}^- := \{z \in \mathbb{C} : \mathrm{Re}(z) < 0\}$ excepts might be the eigenvalue 0, and there exists $0 < \rho < 1$ such that

$$\frac{\|\lambda_j - \lambda_r\|}{\|1 - \lambda_r\|} \leq \rho, \qquad \lambda_j, \lambda_r \in \sigma(\mathbf{K}_0). \tag{39}$$

Two comments must be made on the Hypothesis [H3]. First of all the Hypothesis [H3] is in general not too demanding in view of the form $\mathbf{K}_0$ takes in many cases, for instance in (10) where $\mathbf{K}_0$ writes as $\mathbf{K}_0 = \tau^{\alpha} A$, $\alpha > 0$. In that case [H3] reduces to take the time step τ small enough.

Moreover, in case of $\sigma(\mathbf{K}_0) \not\subset \mathbb{C}^-$, a new family of operators whose spectrum locates in $\mathbb{C}^-$ may be achieved by shifting conveniently the original one. Notice that in spite of this hypothesis may look strange, it does not since the spectrum of $\mathbf{K}_0$ is closely related to the spectrum of $\mathbf{K}$ and consequently to the conditions for the well–posedness of (1).

On the other hand if the choice (38) is made, one simply has to re–formulate the Hypothesis [H3] in the following form

[H3'] The spectrum of $\mathbf{K}_0$ is located in $\mathbb{C}^-$ excepts might be the eigenvalue 0, and there exists $0 < \rho < 1$ such that

$$\frac{\|\lambda_j - \mu^{(r)}\|}{\|1 - \mu^{(r)}\|} \leq \rho, \qquad \lambda_j \in \sigma(\mathbf{K}_0), \quad 1 \leq r \leq D, \tag{40}$$

where $\mu^{(r)}$ are defined according to (38).

In that case the proof of the convergence follows the same steps as in Theorem 1 below.

Before stating the following result recall that m_n denotes the maximum number of iterations for the time step t_n, which according to the above discussion are expected to be much less than D.

Theorem 1 Consider the iterative method (24)–(27), joint with (36) under the Hypothesis [H3]. Assume also that bound (12) satisfies

$$C\tau < 1/D, \tag{41}$$

for τ small enough.

Therefore, given a tolerance $TOL > 0$, there exist $m_n > 0$, $1 < n < N$, such that

$$\|\bar{e}_n^{(m_n)}\| \leq TOL, \quad \text{for} \quad 1 \leq n \leq N. \tag{42}$$

Notice that (41) is straighforwardly satisfied merely taking a time step small enough, however it can be certainly relaxed if a maximum number of iterations $m << D$ is previously fixed for each time step t_n, that is $m_n \leq m$, for $1 \leq n \leq N$. If so (41) may be replaced by the less demanding one $C\tau < 1/m$.

Theorem 1 states sufficient conditions to keep the error of the iterative method proposed under a given tolerance. This results reflects in the numerical experiments where the results show that reaching accurate numerical solutions is feasible with much lower computational effort than with classical/sequential approaches.

Proof The proof makes use of the error expression (30), where $A^{(\eta)}$ and $B^{(\eta)}$ are given by (31). According to this we have to prove that

$$\|A^{(\eta)}\| \to 0, \quad \text{and} \quad \left\| A^{(\eta)} B^{(\eta)} \sum_{j=1}^{n-1} \mathbf{K}_{n-j} \bar{e}_j^{(m_j)} \right\| \to 0, \quad \text{as} \quad \eta \to +\infty,$$

at every time level t_n.

To prove that $\|A^{(\eta)}\|$ tends to zero as $\eta \to +\infty$, observe that

$$A^{(\eta)} = \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)} = \prod_{r=1}^{\eta} \frac{(\mathbf{K}_0 - \lambda_r I)}{1-\lambda_r}, \tag{43}$$

whose spectrum consists of

$$\sigma\left(\frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)}\right) = \{0\} \cup \left\{ \prod_{r=1}^{\eta} \frac{\lambda_j - \lambda_r}{1-\lambda_r} \right\}_{j=\eta+1}^{D}, \qquad 1 \le \eta \le m. \tag{44}$$

Note that 0 turns out to be an eigenvalue of $P_\eta(\mathbf{K}_0)/P_\eta(1)$ of multiplicity η, and by Hypothesis [H3], there exists $0 < \rho < 1$ such that

$$\frac{\lambda_j - \lambda_r}{1-\lambda_r} \le \rho, \qquad 1 \le r \le \eta, \quad 1 \le j \le D. \tag{45}$$

Therefore, from (44) we straightforwardly have that

$$\|A^{(\eta)}\| = \left\| \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)} \right\| \le \rho^\eta \to 0, \quad \text{as} \quad \eta \to +\infty. \tag{46}$$

In particular once reached the maximum number of iterations $\eta = m_n$,

$$\|A^{(m_n)}\| = \left\| \frac{P_{m_n}(\mathbf{K}_0)}{P_{m_n}(1)} \right\| \le \rho^{m_n}, \qquad 1 \le n \le N. \tag{47}$$

The first part the proof then follows.

Related to the term $A^{(\eta)} B^{(\eta)} \sum_{j=1}^{n-1} \mathbf{K}_{n-j} \bar{e}_j^{(m_j)}$, let us show first the boundness of $A^{(\eta)} B^{(\eta)}$. To this end we re–write such a term as follows

$$A^{(\eta)} B^{(\eta)} = \frac{P_\eta(\mathbf{K}_0)}{P_\eta(1)} \sum_{j=1}^{\eta} P_{j-1}(1) P_j(\mathbf{K}_0)^{-1} = \sum_{j=1}^{\eta} \left\{ \frac{1}{1-\lambda_j} \prod_{r=j+1}^{\eta} \frac{(\mathbf{K}_0 - \lambda_r I)}{(1-\lambda_r)} \right\}, \tag{48}$$

and in the same manner as in (46) we have

$$\left\| \frac{1}{1-\lambda_j} \prod_{r=j+1}^{\eta} \frac{(\mathbf{K}_0 - \lambda_r I)}{(1-\lambda_r)} \right\| \le \frac{\rho^{\eta-j}}{|1-\lambda_j|} < \rho^{\eta-j+1}. \tag{49}$$

Therefore it easily follows that, for $\eta \ge 1$,

$$\left\| A^{(\eta)} B^{(\eta)} \right\| \le \sum_{j=1}^{\eta} \rho^{\eta-j+1} \le \sum_{j=1}^{\eta} \rho^j \le \rho \frac{1-\rho^\eta}{1-\rho}, \tag{50}$$

in fact,

$$\left\| A^{(m_n)} B^{(m_n)} \right\| \le \rho C_n, \quad 1 \le n \le N, \quad \text{where} \quad C_n := \frac{1-\rho^{m_n}}{1-\rho}. \tag{51}$$

Recursively and according to (12), (47), and (51) we easily leads to

$$\begin{aligned}
\|\bar{e}_n^{(m_n)}\| &= \left\| A^{(m_n)} B^{(m_n)} \sum_{j=1}^{n-1} \mathbf{K}_{n-j} \bar{e}_j^{(m_j)} + A^{(m_n)} \bar{e}_n^{(0)} \right\| \\
&\leq \left\| A^{(m_n)} B^{(m_n)} \right\| \sum_{j=1}^{n-1} \|\mathbf{K}_{n-j}\| \, \|\bar{e}_j^{(m_j)}\| + \|A^{(m_n)}\| \, \|\bar{e}_n^{(0)}\| \\
&\leq C\tau C_n \rho \sum_{j=1}^{n-1} \left\{ \rho^{m_j} \|\bar{e}_j^{(0)}\| \prod_{r=j+1}^{n-1} (1 + C\tau C_r \rho) \right\} + \rho^{m_n} \|\bar{e}_n^{(0)}\|,
\end{aligned}$$

for $1 \leq n \leq N$. For the simplicity of the notation we have denoted $\prod_{r=n}^{n-1} (1 + C\tau C_r \rho) = 1$.

Finally, since $C_n \leq m_n$, for $1 \leq n \leq N$, and applying (41), if at every single time step t_n the initial error of the iterative stage satisfies

$$\rho^{m_j} \|\bar{e}_j^{(0)}\| \leq \frac{TOL}{N(1+\rho)^N}, \quad \text{for} \quad 1 \leq j \leq N, \tag{52}$$

for a given tolerance $TOL > 0$, then

$$\|\bar{e}_n^{(m_n)}\| \leq TOL, \quad \text{for} \quad 1 \leq n \leq N, \tag{53}$$

and the proof ends. □

It is worthy noting the following:

- The condition (52) may be slightly relaxed by replacing $(1 + \rho)^N$ by $(1 + \rho)^{n-j-2}$.
- Moreover, by the convergence of the original numerical solution $\{\bar{u}_n\}_{1 \leq n \leq N}$, $\|\bar{u}_n - \bar{u}_{n-1}\| = O(\tau)$. Moreover the convergence of the iterative method $\{\bar{u}_n^{(m_n)}\}_{1 \leq n \leq N}$, and setting a small enough time step τ, $\bar{u}_n^{(0)} = \bar{u}_{n-1}^{(m_{n-1})}$ seems certainly to be a suitable candidate for the starting value of the iterative method at time level t_n satisfying (52).

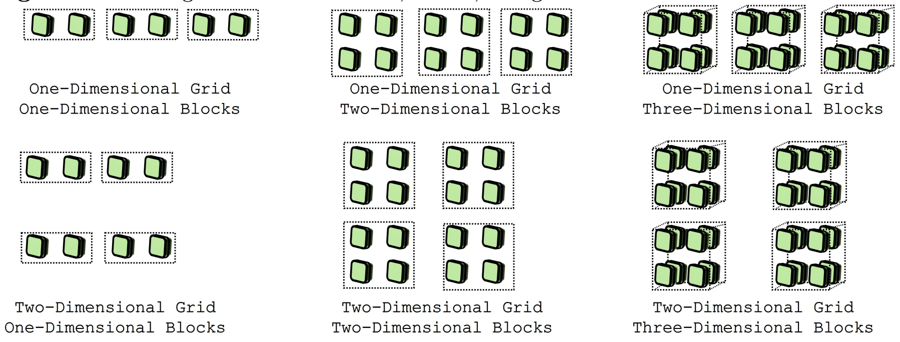
5 CUDA Implementation

The implementation of the parallel numerical scheme proposed in Section 3 for the problem (1), and theoretically analyzed in Section 4, is discussed more in depth in the present section. In particular details related to the algorithm and technical resources involved in the practical implementation of the algorithms are shown below.

Instead of a MIMD (Multiple Instruction - Multiple Data) strategy, based on the cooperation of various Central Processing Units (CPUs), the proposed approach exploits a SIMD (Single Instruction - Multiple Data) methodology according to Flynn taxonomy [42]. This methodology takes advantages of Graphics Processing Units (GPUs) and allows using them for general purpose through CUDA (Compute Unified Device Architecture), introduced by NVIDIA in the 2006. In fact, GPUs were originally designed for graphical purposes such as the computation of image geometrical transformations (translations, rotations, scaling, projections), rendering operations for image finishing, or rastering tasks.

Instead, GPUs for General Purpose (GPGPUs) paradigm aim to exploit Arithmetic Logic Units (ALUs) of GPUs. In this context, CUDA provides a general purpose architecture based on an instruction set so-called Parallel Thread eXecution (PTX), and an extension set for various programming languages. In addition, in the CUDA architecture, the GPU (device) represents a co-processor of the CPU (host) that is able to invoke kernels, functions that are execute by the device through threads.

Fig. 1 Possible organizations of threads, blocks, and grid.



Threads are the fundamental elements for the CUDA parallel design, since a thread executes a set of instructions on a data subset. Moreover, they have not activation/deactivation cost and a large number of them can be used to achieve the full efficiency. Threads are organized in blocks, and blocks are divided in a grid. Grid and blocks can be one-dimensional, two-dimensional or three-dimensional according to the specific problem (see Figure 1). This structure based on grid, blocks, and threads allows to simplify the parallel implementation and takes advantages of a coordinates set accessible through instructions *gridDim*, *blockDim*, *blockIdx*, and *threadIdx*.

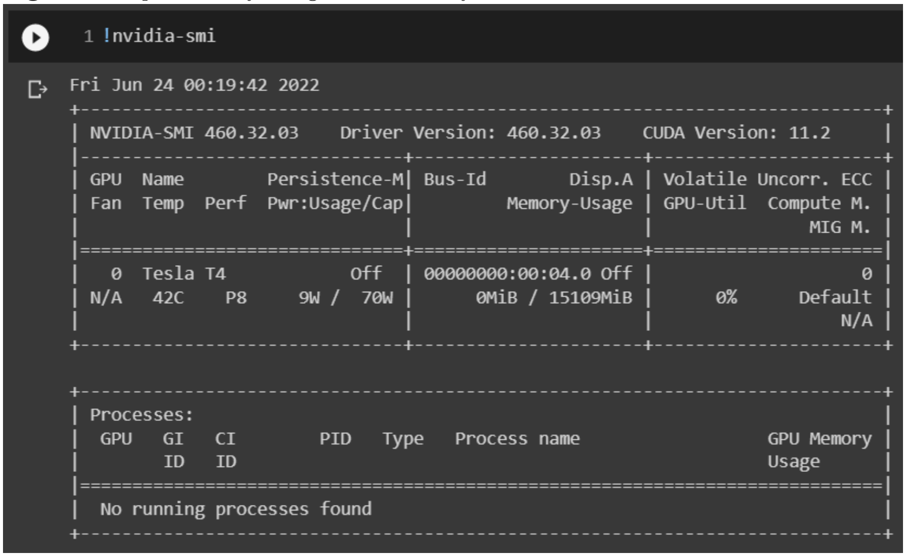
The proposed method implementation through CUDA is based on a one-dimensional grid and one-dimensional blocks. Moreover, numerical results are obtained through Google Colab that made available a Tesla T4 GPU as shown in Figure 2. The Tesla T4 Compute Capability (CC) is 7.5, and the CC category describes the GPU hardware structure and determines limits for the implementation phase.

First of all let us recall the time discretization (27) of the problem (1)

$$\bar{u}_n^{(\eta)} = F_n + \left(K_0 - \mu^{(\eta)} I\right)\bar{u}_n^{(\eta-1)} + \mu^{(\eta)} I \bar{u}_n^{(\eta)}, \quad \eta \geq 1 \tag{54}$$

where F_n stands for

Fig. 2 GPU provided by Google Colaboratory



```
!nvidia-smi
```

```
Fri Jun 24 00:19:42 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   42C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

$$F_n = u_0 + \sum_{j=0}^{n-1} \mathbf{K}_{n-j}\bar{u}_j + \bar{f}_n, \qquad n \geq 1. \tag{55}$$

Let $0 = t_0 < t_1 < \cdots < t_N = T$ be a time mesh for the time discretization with $t_n = t_0 + nh$. Therefore Algorithm 1 describes the implementation of the numerical method (27). The prefix 'h' expresses that the variables are memorized by the host, instead the prefix 'd' precedes the variables memorized by the device. Initially, lines 1 and 2 fix the dimension of blocks and the grid which allow to divide operations in kernels through the provided structure. Then the weights (9) and the parameters (38) are determined according to line 3. In particular, weights are memorized in a multidimensional matrix d_K $\in \mathbb{R}^{D\times D\times N}$ that contains N matrices of dimension $D \times D$. Instead that set of values is stored in the vector mu $= \left(\mu^{(1)}, \ldots, \mu^{(D)}\right)^T \in \mathbb{R}^D$.

Line 4 of the pseudocode summarizes the initialization of the starting term $\bar{u}_0 + \bar{f}\left(t_0\right)$ memorized into the variable d_unew. Then lines 5 describes the kernel that allows to copy the vector d_unew into the first column of the matrix d_un $= (\bar{u}_0, \ldots, \bar{u}_N) \in \mathbb{R}^{D\times(N+1)}$.

The proposed Richardson time–point NSRW method is based on interations on each subinterval between two consecutive discretization points. Then, the term (55) is computed through the kernel at line 8 and memorized into the vector d_F_n.

The iteration on the subinterval continues while the error is less than the tolerance and the maximum iterations number is not reached. Into the loop, the method is applied according to (54), and after its application lines 13–15 allow to calculate the error estimation in the point t_n at iteration $\eta =$Niter:

Algorithm 1 Algorithm related to the CUDA program
The times where the numerical solutions are computed are $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ (h=T/N), where $t_0 = 1$ and $t_N = T = 1$ in all experiments below, and for several values of N ($N = 50$, and 100).

```
 1: dim3 dimBlock(BLOCK_SIZE);
 2: dim3 dimGrid( (int)ceil(float(D)/float(dimBlock.x)));
 3: Calculation of weights d_K0 ∈ $\mathbb{R}^{D\times D}$ and d_K ∈ $\mathbb{R}^{D\times D\times N}$, and parameters
    mu ∈ $\mathbb{R}^D$;
 4: starting<<<dimGrid, dimBlock>>> ( d_unew, D,$t_0$);
 5: copyvetmat<<<dimGrid, dimBlock>>>(d_un,d_unew,D,N,0);
 6: for $n = 1, \ldots, N$ do                                ▷ $N$ # of discrete points
 7:     $t = t_0 + nh$;
 8:     lagterm<<<dimGrid, dimBlock>>>(d_un,d_F$_n$,h,d_K,t,D,N,n);
 9:     copiavet<<<dimGrid, dimBlock>>>(d_F$_n$,d_uold,D);
10:     err=1, Niter = 0;
11:     while (err>toll) && (Niter<Nmax) do
12:         richardson<<<dimGrid,                              dimBlock>>>
    (mu[Niter],d_uold,d_unew,d_F$_n$,h, d_K0,D);
13:         error<<<dimGrid, dimBlock>>>(d_uold,d_unew,d_err,D);
14:         thrust::device_ptr<float> new_d(d_err);
15:         err = thrust::reduce(new_d, new_d + D);
16:         copiavet<<<dimGrid, dimBlock>>>(d_unew,d_uold,D);
17:         Niter++;
18:     end while
19:     end while
20:     copyvetmat<<<dimGrid, dimBlock>>>(d_un,d_unew,D,N,n);
21: end for
22: end for
```

$$\mathrm{EE}_n^{(\eta)} = \|\bar{u}_n^{(\eta)} - \bar{u}_n^{(\eta-1)}\|, \qquad 1 \le n \le N, \quad \eta \ge 1. \tag{56}$$

In particular, at line 13 the kernel *error* allows to memorize the difference between components of $\bar{u}_n^{(\eta)}$ and $\bar{u}_n^{(\eta-1)}$, stored in the vector d_err. The function *thrust::device_ptr* allocates the new vector d_unew on the device, then the function *thrust::reduce* returns the error estimation (56).

Indeed, the approximated d_unew solution related to the discretization point t_n is copied into d_uold at line 16. Finally, at line 19, the new approximation related to the last iteration m_n is memorized into the matrix d_un.

Table 1 summarizes the variables related to Algorithm 1.

Table 1 Variables involved in Algorithm 1

Name	**Type**	**Dimension**	**Referring to the Proposed Method**
D	scalar value	1	system dimension
N	scalar value	1	number of discretization intervals
h	scalar value	1	stepsize
t_0	scalar value	1	first discretization point
mu	vector	D	parameters in (38)
d_K0	matrix	D × D	weights matrix related to the first discretization point
d_K	multidimesional matrix	D × D × N	weights in (9)
d_F_n	vector	D	F_n defined in (55)
d_uold	vector	D	$\bar{u}_n^{(\eta-1)}$ in (54)
d_unew	vector	D	$\bar{u}_n^{(\eta)}$ in (54)
d_un	matrix	D × (N + 1)	matrix $\left(\bar{u}_n^{(m_n)}\right)_{n=0,\dots,N}$ with m_n defined in (42)
err	scalar	1	error estimation in (56)

6 Numerical Results

As performance metric related to parallel implementations we considered the Speed–Up, that consists of the rate of the runtime (in milliseconds) for the sequential and parallel implementations.

$$\text{Speed–Up} = \frac{\text{CPUtime}}{\text{GPUtime}} \tag{57}$$

For the Speed–Up assessment, the following subsections present several tests applied to practical problems. The numerical results aim to evaluate the ability of parallel computing to improve the velocity of the problem resolution. Moreover, they also allow for assessing the accuracy of the proposed approach and for empirically evaluating the convergence order of the method.

Results will be presented through tables that describe, in addition to Speed–Up, the dimension D of the system (1), sequential (CPU time) and parallel (GPU time) implementation times measured in milliseconds, and the error E referred to the exact solution $\bar{u}\,(t)$

$$\text{E} = \max_{n=0,\dots,N} \|\bar{u}_n^{(m_n)} - \bar{u}\,(t_n)\,\|, \tag{58}$$

where m_n, defined in (42), represents the iteration that allows to satisfy the tollerance in the point t_n. Finally, tables show the Error Estimation EE, aka, the maximal value of errors $\text{EE}_n^{(m_n)}$ defined in (56)

$$\text{EE} = \max_{1\le n\le N} \|\text{EE}_n^{(m_n)}\| = \max_{1\le n\le N} \|\bar{u}_n^{(m_n)} - \bar{u}_n^{(m_n-1)}\|. \tag{59}$$

Notice that E stands for the true error yielded by the numerical method, while EE is nothing but an estimation of it which is really useful as stoping criteria in cases where analytical solution is not available at all. In our experiments the analytical solution is actually known, despite that we show the values of EE.

We will also show the mean of iterations' number at each timestep t_n, i.e.:

$$m = \frac{m_1 + \cdots + m_N}{N} \tag{60}$$

Moreover, let us notice that throughout the present section, the error will be measured in the sup norm.

6.1 Test problem 1

The first test problem consists of the system (1) with $T = 1$, where

$$\bar{u}_0 = \mathbf{0} \in \mathbb{R}^D, \qquad \bar{f}(t) = \left(t + \frac{4\sqrt{t^3}}{15\Gamma(\alpha)}, t, \ldots, t, t - \frac{4\sqrt{t^3}}{15\Gamma(\alpha)} \right)^T \in \mathbb{R}^D, \tag{61}$$

$$\mathbf{K}(t-s) = \frac{(t-s)^{\alpha-1}}{\Gamma(\alpha)} A, \tag{62}$$

with $1 < \alpha < 2$, and the matrix A is the three–diagonal matrix

$$A = \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & 1 & -2 & 1 \\ 0 & \cdots & \cdots & 1 & -2 \end{pmatrix} \in \mathcal{M}_{D\times D}(\mathbb{R}), \tag{63}$$

representing the one dimensional Laplacian discretization by means of the classical second order finite differences scheme (without re–scaling with spatial mesh length).

The exact solution to the test problem 1 is

$$\bar{u}(t) = (t, \ldots, t)^T \in \mathbb{R}^D. \tag{64}$$

Tables 2 and 3 present the results provided by sequential and parallel implementation of the numerical method for $N = 50$ and $N = 100$, respectively. Notice that the values of Speed–Up increase as the system dimension D increases. In particular, the values of Speed–Up provided by the case $N = 100$ are better than the ones provided by the case $N = 50$.

6.2 Test problem 2

The second test problem stands for a more complex problem than the previous one where, instead of the considering a single value of α, there are different

Table 2 Richardson time–point NSRW method with $\alpha = 1.5$, $T = 1$, tol $= 1e - 3$, and BLOCK_SIZE = 32, N= 50. The iteration mean in (60) is $m = 1.84$

D	**CPU (ms)**	**GPU (ms)**	**E**	**EE**	**Speed–Up**
64	27	19	7.397711e-03	8.935034e-04	1.4211
128	112	34	7.397473e-03	8.953214e-04	3.2941
256	448	105	7.397532e-03	8.971691e-04	4.2667
512	1866	230	7.397532e-03	9.009838e-04	8.1130
1024	7849	505	7.397652e-03	8.934736e-04	15.5426
2048	31828	1012	7.397532e-03	8.934438e-04	31.4506
4096	151330	2251	7.397532e-03	8.934438e-04	67.2279

Table 3 Richardson time–point NSRW method with $\alpha = 1.5$, $T = 1$, tol $= 1e - 3$, and BLOCK_SIZE = 32, N= 100. The iteration mean in (60) is $m = 1.5$

D	**CPU (ms)**	**GPU (ms)**	**E**	**EE**	**Speed–Up**
64	118	67	3.715694e-03	9.936988e-04	1.7612
128	448	134	3.715575e-03	9.935200e-04	3.3433
256	1764	298	3.715456e-03	9.936690e-04	5.9195
512	7825	577	3.715456e-03	9.936094e-04	13.5615
1024	33322	1738	3.715575e-03	9.935200e-04	19.1726
2048	149889	3705	3.715456e-03	9.935200e-04	40.4559
4096	672455	8643	3.715456e-03	9.962916e-04	77.8034

values related to each single integral equation of the system. It, indeed, consists of the system (1) in which

$$\bar{u}_0 = \mathbf{0} \in \mathbb{R}^D, \tag{65}$$

and the vector $\bar{f}(t)$ has the following elements:

$$f_1(t) = t + 2\frac{t^{\alpha_1+1}}{\Gamma(\alpha_1)\,\alpha_1(\alpha_1+1)} - \frac{t^{\alpha_2+1}}{\Gamma(\alpha_2)\,\alpha_2(\alpha_2+1)}, \tag{66}$$

$$\begin{aligned} f_i(t) = t - \frac{t^{\alpha_{i-1}+1}}{\Gamma(\alpha_{i-1})\,\alpha_{i-1}(\alpha_{i-1}+1)} + 2\frac{t^{\alpha_i+1}}{\Gamma(\alpha_i)\,\alpha_i(\alpha_i+1)} - \\ -\frac{t^{\alpha_{i+1}+1}}{\Gamma(\alpha_{i+1})\,\alpha_{i+1}(\alpha_{i+1}+1)} \qquad i = 2,\ldots,D-1, \end{aligned} \tag{67}$$

$$f_D(t) = t - \frac{t^{\alpha_{D-1}+1}}{\Gamma(\alpha_{D-1})\,\alpha_{D-1}(\alpha_{D-1}+1)} + 2\frac{t^{\alpha_D+1}}{\Gamma(\alpha_D)\,\alpha_D(\alpha_D+1)}. \tag{68}$$

Then the convolution operator has the form

$$\mathbf{K}(t-s) = A\ I(t-s), \tag{69}$$

where the matrix A is defined in (63), and the matrix $I(t-s)$ represents a diagonal matrix in which the diagonal elements are

$$I_{ii}(t-s) = \frac{(t-s)^{\alpha_i-1}}{\Gamma(\alpha_i)} \qquad i = 1,\ldots,D. \tag{70}$$

These hypotheses allow to know that the analytical solution is same as in the problem test 1, that is (64).

Tables 4 and 5 summarize the obtained results for $N = 50$ and $N = 100$, respectively. We also report in these tables the mean of iteration (60) related to each system dimension because the selection of $\alpha_1, \dots, \alpha_D$ through pseudo-casual numbers influences the total number of iterations.

Table 4 Richardson time–point NSRW method with $T = 1$, tol $= 1e - 3$, and BLOCK_SIZE = 32, N= 50.

D	**CPU (ms)**	**GPU (ms)**	**m**	**E**	**EE**	**Speed–Up**
64	32	25	3.22	6.361306e-03	9.817481e-04	1.2800
128	128	50	2.96	7.957757e-03	6.324649e-04	2.5600
256	522	131	2.96	7.530391e-03	8.974820e-04	3.9847
512	2318	239	3.58	7.871866e-03	9.982884e-04	9.6587
1024	10036	546	3.90	7.064283e-03	9.021387e-04	18.3810
2048	44654	1069	3.98	7.480860e-03	7.445198e-04	41.7717
4096	188029	2378	2.94	8.184791e-03	9.403415e-04	79.0702

Table 5 Richardson time–point NSRW method with $T = 1$, tol $= 1e - 3$, and BLOCK_SIZE = 32, N= 100.

D	**CPU (ms)**	**GPU (ms)**	**m**	**E**	**EE**	**Speed–Up**
64	111	81	2.84	3.215253e-03	9.885132e-04	1.3704
128	470	188	2.78	3.967643e-03	9.888783e-04	2.5000
256	2195	384	2.82	3.755748e-03	9.220243e-04	5.7161
512	9242	606	2.86	3.920734e-03	9.138733e-04	15.2508
1024	43657	1811	2.96	3.520787e-03	7.510614e-04	24.1066
2048	182070	3876	3.62	3.721952e-03	9.990931e-04	46.9737
4096	1330830	8949	2.00	4.073441e-03	6.577969e-04	148.7127

Tables 4 and 5 show the increase of the Speed–Up as system dimension D growths. In particular, in the case N= 100 and D= 4096 we obtain a Speed–Up equal to 148.7127. We also notice that the obtained Speed–Ups are better than the ones of the previous numerical experiment.

Table 6 Assumptions on the grid for obtaining the A matrix for the test problem 3.

D	**D_x**	**D_y**
32	4	8
64	8	8
128	8	16
256	16	16
512	16	32
1024	32	32
2048	32	64
4096	64	64

6.3 Test Problem 3

The last problem considered to test the proposed approach has the following specifications

$$\bar{u}_0 = \mathbf{0} \in \mathbb{R}^D, \tag{71}$$

$$\mathbf{K}\,(t-s) = A\;I\,(t-s)\,, \tag{72}$$

where $I(t-s)$ is diagonal matrix defined in (70), and the matrix A represent now the approximation matrix obtained by second order finite difference scheme to the two dimensional Laplacian operator with Neumann boundary conditions. In particular, by considering D_x discretization points in the x–axis direction and D_y discretization points in the y–axis direction, the matrix A belongs to $\mathcal{M}_{D\times D}(\mathbb{R})$ with $D = D_x D_y$ and is defined by

$$(\Delta_\tau \bar{u})_{i,j} = \frac{\bar{u}_{i+1,j} + \bar{u}_{i-1,j} - 4\bar{u}_{i,j} + \bar{u}_{i,j+1} + \bar{u}_{i,j-1}}{\tau^2}, \tag{73}$$

for $i = 1, \ldots, D_x, \quad j = 1, \ldots, D_y$. Once again the spatial mesh length does not play any role in this approach, this is why it is taken as 1 and number of discretization points D_x and D_y chosen as in Table 6.

In this problem, in order to derive the same analytical solution of the previous test problems

$$\bar{u}\,(t) = (t, \ldots, t)^T \in \mathbb{R}^D, \tag{74}$$

the source term $\bar{f}(t) = (f_1(t), f_2(t), \ldots, f_D(t))^T$, comes defined by

$$f_i\,(t) = t - \sum_{i=1}^{D} a_{ij} \frac{t^{\alpha_i+1}}{\Gamma\,(\alpha_i)\,\alpha_i\,(\alpha_i+1)} \qquad i = 1, \ldots, D, \tag{75}$$

where, a_{ij} stand for the entries of the matrix A, and as done in the test problem 2, the values $\alpha_1, \ldots, \alpha_D$ are generated through pseudo–random numbers and are related to each integral integral equation of the system.

Table 7 Richardson time–point NSRW method with $T = 1$, tol $= 1e-3$, and BLOCK_SIZE = 32, N= 50.

D	CPU (ms)	GPU (ms)	m	E	EE	Speed–Up
64	51	31	4.74	2.254719e-02	9.942949e-04	1.6452
128	195	63	4.76	2.192962e-02	9.448826e-04	3.0952
256	779	149	4.52	2.038664e-02	9.826720e-04	5.2282
512	3268	272	4.72	2.128011e-02	9.753406e-04	12.0147
1024	13317	588	4.92	2.158207e-02	8.717850e-04	22.6480
2048	59041	1148	4.94	2.006114e-02	7.664412e-04	51.4294
4096	262844	2557	3.90	2.280349e-02	9.278730e-04	102.7939

Tables 7 and 8 present the results related to the test problem 3 with N= 50 and N= 100, respectively.

Table 8 Richardson time–point NSRW method with $T = 1$, tol $= 1e - 3$, and BLOCK_SIZE = 32, N= 100.

D	**CPU (ms)**	**GPU (ms)**	**m**	**E**	**EE**	**Speed–Up**
64	127	78	2.40	1.106864e-02	9.861588e-04	1.6282
128	488	200	2.51	1.069969e-02	9.885132e-04	2.4400
256	2203	391	2.62	1.010352e-02	9.793937e-04	5.6342
512	9274	597	2.80	1.047248e-02	9.895116e-04	15.5084
1024	41374	1787	2.86	1.053202e-02	9.916872e-04	23.1528
2048	178755	3821	2.91	1.028883e-02	9.425953e-04	46.7823
4096	1190617	9022	2.96	1.112318e-02	8.166134e-04	131.9682

Also in this case, the increase of the Speed–Up as system dimension growths confirms the benefits of the parallelization. In fact, for D= 4096, the sequential times are more than a hundred times slower than their respective parallels with both N= 50 and N= 100.

6.4 Numerical results assessments

As mentioned above, another aim of numerical tests consists of the experimental assessment of the convergence order. The analysis related to the results arisen from the three test problems, are shown in Table 9 which presents the results obtained to a system of dimension D= 32 as the step size of the time discretization decreases of a factor equal to 2. The increase of the discretization points N gives rise to the decrease of the error. The error in logarithmic scale, showed in Figure 3, follows the slope of order 1 with the exception of the case N= 3200 where there is a little deviation.

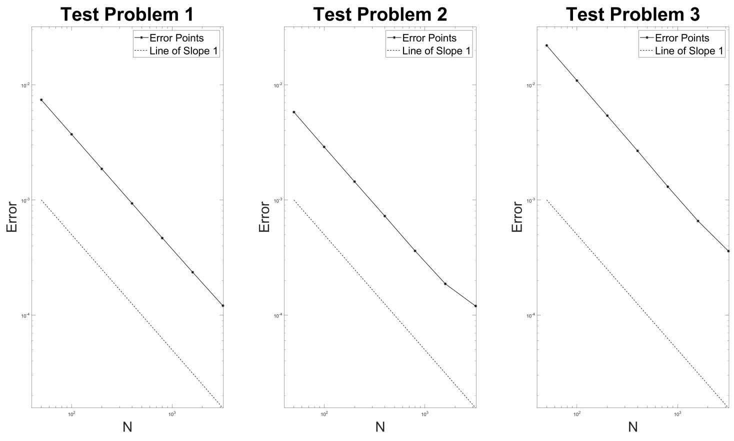
Then the assessment of the convergence, theoretically analyzed in Section 4, is experimentally confirmed. Moreover, Figure 3 represents the errors in logarithmic scale and allows deducing that the slope related to error points in Table 9 follows the slope of order 1. The only exception consists of the second test problem at the case $N = 3200$.

Table 9 Error, and Iteration Average for test problems 1, 2 and 3 (respectively TP1, TP2, and TP3) based on the increase of the discretization points with $D = 32$, tol= $1e-6$, $T = 1$, and BLOCK_SIZE = 32. In the test problem 1, the parameter α assumes the value 1.5.

	N	**50**	**100**	**200**	**400**	**800**	**1600**	**3200**
TP1	E	7.398e-03	3.72e-03	1.87e-03	9.33e-04	4.67e-04	2.36e-04	1.21e-04
	m	3.64	3.55	4.09	2.52	2.62	2.60	2.26
TP2	E	5.79e-03	2.88e-03	1.44e-03	7.25e-04	3.61e-04	1.87e-04	1.20e-04
	m	5.04	4.49	4.30	3.82	3.59	3.44	3.27
TP3	E	2.20e-02	1.09e-02	5.39e-03	2.67e-03	1.30e-03	6.56e-04	3.60e-04
	m	6.56	4.49	3.78	3.25	3.02	2.73	2.10

According to the results presented in previous subsections and summarized in Figure 4, the three test problems provide a considerable improvement in

Fig. 3 Empirical evaluation of the convergence order related to test problems.



performance from sequential to parallel executions. The runtime saved allows the application of the proposed method to fields in which the response times represents a key point.

In fact, nowadays a widespread application field of the analyzed problem is image or video processing. Systems of type (1) rather frequently appear in problems related to image and video processing (image filtering, clustering, video restoration,...). In this framework one of the main drawbacks is the size of images and/or frames to be processed which give rise to computations with very large system of equations (linear or even non–linear). What is harder, in many instances results are required to get ready in real time: restoration of damaged patches, dis–occluding hidden parts of frames in video,... In this regard see for instance the recent paper [43] where linear Volterra equations of type (1) are proposed for video restoration. In this context, the response time is crucial and for related algorithms the possibility of reducing the run time represents a great challenge and any advantage is welcome. Moreover, the convergence analysis confirms the goodness of the proposed approach.

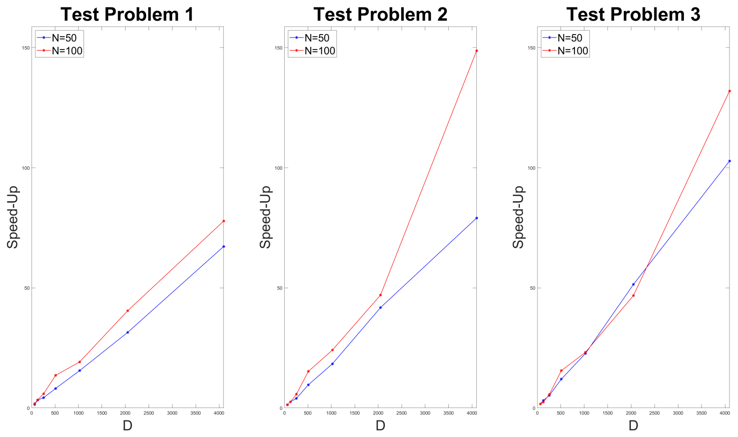
7 Conclusions and future works

NSWR methods and their parallel implementation by means of GPUs have shown a very good performance, particularly if compared to sequential implementation. The good performance extends further classical fractional integral equations but for more general linear systems of Volterra equations as the ones shown in Section 6.

What is more the numerical results reached in Section 6 are perfectly in line with the theoretical results in Section 4 related to the convergence

The statement of the Theorem 1 may be extended without no significant differences to a wider class of equations of type (1). In fact one may assume

Fig. 4 Graphical representations of Speed–Up results



that $\mathbf{K}(t)$ admits an element–wise Laplace transform $\widetilde{\mathbf{K}}(z) = (\tilde{k}_{i,j}(z))_{1\le i,j\le D}$ whose entries turn out to be analytic in the complex sector $S_\varphi := \{z \in \mathbb{C} : \|\arg(-z)\| < \varphi\}$, $0 < \varphi < \pi/2$, and for which there exist $M, \nu > 0$ such that

$$\left\|\tilde{k}_{i,j}(z)\right\| \le \frac{M}{\|z\|^\nu}, \qquad 1 \le i,j \le D, \quad z \notin S_\varphi.$$

In that case Corollary 3.2 in [16] extends (12) and says that there exists $C > 0$ such that

$$\|\mathbf{K}_n\| \le C t_n^{\nu-1}\tau, \quad 1 \le n \le N, \quad \text{and} \quad \|\mathbf{K}_0\| \le C\tau^{\nu-1}.$$

Under this hypothesis Theorem 1 follows in similar fashion.

We finally highlight that this paper focuses on the accuracy and Speed–Up analysis of the proposed parallelized approach for general purposes, that is for linear systems of general Volterra equations but not for particular applications. The application of this approach in the field of images processing represents a future development of the present work.

8 Declarations

8.1 Ethical Approval

Not Applicable

8.2 Availability of supporting data

Not Applicable

8.3 Competing interests

Not Applicable

8.4 Funding

The authors D. Conte and C. Valentino are members of the GNCS group. This work has been partially supported by GNCS-INDAM project and by the Italian Ministry of University and Research (MUR), through the PRIN 2017 project (No. 2017JYCLSF) "Structure preserving approximation of evolutionary problems".

8.5 Author's contribution

D. Conte, V. Carmine, and E. Cuesta. shared writing, and preparing the main manuscript text. V. Carmine took mostly over the codes performance. All authors reviewed the manuscript.

8.6 Acknowledgements

The author E. Cuesta would like to thank to GIR (Research Recognized Group) of the University of Valladolid, and IMUVa (Mathematic Institute of the University of Valladolid).

References

[1] Stinga, P.R.: User's guide to the fractional Laplacian and the method of semigroups. Fractional Differential Equations, vol. 2, pp. 235–266. De Gruyter. Anatoly Kochubei and Yuri Luchko Edts., (2019). https://doi.org/10.1515/9783110571660-012

[2] Prüss, J.: Evolutionary Integral Equations and Applications, 1st edn. Modern Birkhäuser Classics. Birkhäuser, Basel, (2012). https://doi.org/10.1007/978-3-0348-0499-8

[3] Cuesta, E., Durán, A., Kirane, M.: On evolutionary integral models for image restoration. In: Tavares J., Natal Jorge R. (Edts) Developments in Medical Image Processing and Computational Vision. Lecture Notes in Computational Vision and Biomechanics, vol. 19, pp. 241–260. Springer, (2015). https://doi.org/10.1007/978-3-319-13407-915

[4] Hilfer, R.: Applications of Fractional Calculus in Physics. World Scientific, Singapore, (2000). https://doi.org/10.1142/3779

[5] Podlubny, I.: Fractional Differential Equations. Mathematics in Science and Engineering. Academic Press, (1999)

[6] Kilbas, A.A., Srivastava, H.M., Trujillo, J.J.: Theory and Applications of Fractional Differential Equations. North Holland Mathematics Studies 204. Elsevier B. V., (2006). https://doi.org/10.1016/S0304-0208(06)80001-0

[7] Das, S.: Functional Fractional Calculus, 2nd edition Springer, (2011). https://doi.org/10.1007/978-3-642-20545-3

[8] Cuesta, E., Kirane, M., Alsaedi, A., Ahmad, B.: On the sub–diffusion fractional initial value problem with time variable order. Adv. Nonlinear Anal. **10**(1), 1301–1315 (2021). https://doi.org/anona-2020-0182

[9] Cuesta, E., Ponce, R.: Well–posedness, regularity, and asymptotic behavior of continuous and discrete solutions of linear fractional integro–differential equations with time–dependent order. Electron. J. Differential Equations **2018**(173), 1–27 (2018)

[10] Banjai, L., Lubich, Ch.: An error analysis of Runge–Kutta convolution quadrature. BIT **51**(3), 483–496 (2011). https://doi.org/10.1007/s10543-011-0311-y

[11] Banjai, L., Lubich, Ch., Melenk, J.M.: Runge–Kutta convolution quadrature for operators arising in wave propagation. Numer. Math. **119**(1), 1–20 (2011). https://doi.org/10.1007/s00211-011-0378-z

[12] Calvo, M.P., Cuesta, E., Palencia, C.: Runge–Kutta convolution quadrature methods for well–posed equations with memory. Numer. Math. **107**, 589–614 (2007). https://doi.org/10.1007/s00211-007-0107-9

[13] Cuesta, E., Lubich, Ch., Palencia, C.: Convolution quadrature time discretization of fractional diffusion–wave equations. Math. Comput. **75**(254), 673–696 (2006). https://doi.org/10.1090/S0025-5718-06-01788-1

[14] Cuesta, E., Palencia, C.: A numerical method for an integro–differential equation with memory in Banach spaces: Qualitative properties. SIAM J. Numer. Anal. **41**(4), 1232–1241 (2003). https://doi.org/10.1137/S0036142902402481

[15] Lubich, Ch.: Fractional linear multistep methods for Abel–Volterra integral equations of the second kind. Math. Comput. **45**(172), 463–469 (1985). https://doi.org/10.2307/2008136

[16] Lubich, Ch.: Convolution quadrature and discretized operational calculus I. Numer. Math. **52**, 129–145 (1988). https://doi.org/10.1007/BF01398686

[17] Lubich, Ch.: On the multistep time discretization of linear initial–boundary value problems and their boundary integral equations. Numer. Math. **67**, 365–389 (1994). https://doi.org/10.1007/s002110050033

[18] Brunner, H.: Collocation Methods for Volterra Integral and Related Functional Differential Equations. Cambridge Monographs on Applied and Computational Mathematics (15). Cambridge University Press, (2004). https://doi.org/10.1017/CBO9780511543234

[19] Cardone, A., Conte, D., D'Ambrosio, R., Paternoster, B.: Collocation methods for Volterra integral and integro–differential equations: A review. Axioms **7**(45), 1–19 (2018). https://doi.org/AngelamariaCardone1,*ID,DajanaConte1ID,RaffaeleD\OT1\textquoterightAmbrosio2IDandBeatricePaternoster1I

[20] Cardone, A., Conte, D., Paternoster, B.: Stability of two-step spline collocation methods for initial value problems for fractional differential equations. Commun. Nonlinear Sci. **115**, 106726 (2022). https://doi.org/10.1016/j.cnsns.2022.106726

[21] Conte, D., D'Ambrosio, R., Paternoster, B.: Two–step diagonally–implicit collocation based methods for Volterra integral equations. Appl. Numer. Math. **62**(10), 1312–1324 (2012). https://doi.org/10.1016/j.apnum.2012.06.007

[22] Conte, D., Prete, I.D.: Fast collocation methods for Volterra integral equations of convolution type. J. Comput. Appl. Math. **196**, 652–663 (2006). https://doi.org/10.1016/j.cam.2005.10.018

[23] López–Fernández, M., Palencia, C.: On the numerical inversion of the Laplace transform of certain holomorphic mappings. Appl. Numer. Math. **51**(2-3), 289–303 (2004). https://doi.org/10.1016/j.apnum.2004.06.015

[24] Capobianco, G., Conte, D., del Prete, I., Russo, E.: Stability analysis of fast numerical methods for Volterra integral equations. Electron. Trans. Numer. Anal. **30**, 305–322 (2008)

[25] Hairer, E., Lubich, Ch., Schlichte, M.: Fast numerical solution of nonlinear Volterra convolution equations. SIAM J. Numer. Anal. **6**(3), 532–541 (1985). https://doi.org/10.1137/0906037

[26] López–Fernández, M., Lubich, Ch., Schädle, A.: Adaptive, fast, and oblivious convolution in evolution equations with memory. SIAM J. Sci. Comp. **30**(2), 1015–1037 (2008). https://doi.org/10.1137/060674168

[27] Schädle, A., López–Fernández, M., Lubich, Ch.: Fast and oblivious convolution quadrature. SIAM J. Scient. Comput. **28**(2), 621–639 (2006).

https://doi.org/10.1137/050623139

[28] Oancea, B., Andrei, T., Dragoescu, R.M.: GPGPU Computing. In: Proceedings of the CKS International Conference, 2012. arXiv:1408.6923, (2014)

[29] Burrage, K.: Parallel and Sequential Methods for Ordinary Differential Equations. Numerical Analysis and Scientific Computation. Oxford University Press, New York, (1995)

[30] Burrage, K., Dyke, C., Pohl, B.: On the performance of parallel weveform relaxations for differential systems. Appl. Numer. Math. **20**(1-2), 39–55 (1996). https://doi.org/10.1016/0168-9274(95)00116-6

[31] Sand, J., Burrage, K.: A Jacobi waveform relaxation method for ODEs. SIAM J. Sci. Comp. **20**(2), 534–552 (1998). https://doi.org/10.1137/S1064827596306562

[32] Courvoisier, Y., Gander, M.J.: Optimization of Schwarz waveform relaxation over short time windows. Numer. Algorithms **64**, 221–243 (2013). https://doi.org/10.1007/s11075-012-9662-y

[33] Lent, J.V., Vandewalle, S.: Multigrid waveform relaxation for anisotropic partial differential equations. Numer. Algorithms **31**(1), 361–380 (2002). https://doi.org/10.1023/A:1021191719400

[34] Zhang, H., Jiang, Y.-L.: A note on the H^1–convergence of the overlapping Schwarz waveform relaxation method for the heat equation. Numer. Algorithms **66**, 299–307 (2014). https://doi.org/10.1007/s11075-013-9734-7

[35] Capobianco, G., Conte, D.: An efficient and fast parallel method for Volterra integral equations of Abel type. J. Comput. Appl. Math. **189**(1-2), 481–493 (2006). https://doi.org/10.1016/j.cam.2005.03.056

[36] Capobianco, G., Conte, D., del Prete, I.: High performance parallel numerical methods for Volterra equations with weakly singular kernels. J. Comput. Appl. Math. **228**(2), 571–579 (2009). https://doi.org/10.1016/j.cam.2008.03.027

[37] Cardone, A., Messina, E., Russo, E.: A fast iterative method for discretized Volterra–Fredholm integral equations. J. Comput. Appl. Math. **189**(1-2), 568–579 (2006). https://doi.org/10.1016/j.cam.2005.05.018

[38] Conte, D., D'Ambrosio, R., Beatrice, P.: GPU–acceleration of waveform relaxation methods for large differential systems. Numer. Algorithms **71**(2), 293–310 (2016). https://doi.org/10.1007/s11075-015-9993-6

[39] Califano, G., Conte, D.: Optimal Schwarz waveform relaxation for fractional diffusion–wave equations. Appl. Numer. Math. **127**, 125–141 (2018). https://doi.org/10.1016/j.apnum.2018.01.002

[40] Conte, D., Paternoster, B.: Parallel methods for weakly singular Volterra integral equations on GPUs. Appl. Numer. Math. **114**, 30–37 (2017). https://doi.org/10.1016/j.apnum.2016.04.006

[41] Capobianco, G., Crisci, M.R., Russo, E.: Nonstationary waveform relaxation methods for Abel integral equations. J. Integral Equations Appl. **16**(1), 53–65 (2004). https://doi.org/10.1216/jiea/1181075258

[42] Mishr, I., Karakaya, Z.: Teaching parallel computing concepts using real-life applications. Int. J. Engineer. Edu. **32**(2), 772–781 (2016).

[43] Cuesta, E., Finat, J., Sánchez, J.: Grey-level intensity measurements processing by means of Volterra equations and Least Squares Method for video restoration. Phys. Scr. **In press**, 1–25 (2023).