








# Performance Evaluation for a PETSc Parallel-in-Time Solver Based on the MGRIT Algorithm

Valeria Mele<sup>1</sup> , Diego Romano<sup>2</sup> , Emil M. Constantinescu<sup>3</sup> ,  
Luisa Carracciuolo<sup>2</sup> , and Luisa D'Amore<sup>1</sup> 

<sup>1</sup> University of Naples Federico II, Naples, Italy  
`valeria.mele@unina.it`

<sup>2</sup> Italian National Research Council - CNR, Rome, Italy

<sup>3</sup> Mathematics and Computer Science Division,  
Argonne National Laboratory, Chicago, IL, USA

**Abstract.** We herein describe the performance evaluation of a modular implementation of the MGRIT (MultiGrid-In-Time) algorithm within the context of the PETSc (the Portable, Extensible Toolkit for Scientific computing) library. Our aim is to give the PETSc users the opportunity of testing the MGRIT parallel-in-time approach as an alternative to the Time Stepping integrator (TS), when solving their problems arising from the discretization of linear evolutionary models. To this end, we analyzed the performance parameters of the algorithm in order to underline the relationship between the configuration factors and problem characteristics, intentionally overlooking any accuracy issue and spatial parallelism.

**Keywords:** Parallelism-in-time · Performance evaluation  
Multigrid reduction · MGRIT · Linear systems · PETSc

## 1 Introduction

Scientific applications in life science and in many other fields can benefit from the Parallel In Time (PINT) methods which have the potential to extract additional parallelism in many applications governed by evolutionary models, allowing for concurrency also along the temporal dimension. Consider as an example the analysis, the reconstruction and the denoising of ultrasound images arising from 2D/3D echocardiography [2, 18, 24]. In the European Exascale Software Initiative (EESI) 2014 roadmap, PINT approaches are recommended to the end of developing efficient applications for Exascale computing, thus taking a significant step beyond “traditional” HPC. On the other side, the deployment of application codes by means of scientific libraries, such as PETSc (the Portable, Extensible Toolkit for Scientific computing) [1], can be considered as a good investment [2] to maximize the availability of PinT algorithms for scientific applications.

Recent advances in PETSc regarded the improvement of multilevel, multidomain and multiphysics algorithms. The most relevant capabilities allow users to test different solvers (linear, nonlinear, and time stepping) for their complex simulations, without making premature choices about algorithms and data structures [1]. Nevertheless, PETSc does not provide any parallel-in-time support.

The MultiGrid-In-Time (MGRIT) algorithm is a PINT approach based on Multigrid Reduction (MGR) techniques [3]. Although we know that it is implemented in the software package XBraid [4], we are developing a modular multilevel parallel implementation [5] based on PETSc. The main goal of our approach is to provide a model predicting the performance gain achievable using the MGRIT approach instead of a timemarching integrator, independently of whether parallelism in the space dimension is introduced or not. The performance model in [7], instead, aims to selecting the best parallel configuration (i.e. how much parallelism is to be devoted to space vs. time). Therefore, we analyze the performance parameters of the algorithm in the mathematical framework presented in other works by the same authors [8]. We intentionally overlook spacial parallelism. In this way we describe the performance improvement regardless of the execution time needed to implement the characteristic function of the problem. We believe that both performance models could be employed for the successful implementation of the MGRIT algorithm [9].

In the second section we describe the basic idea of the algorithm to be implemented, summarizing the main results described in other works [3, 7, 10, 11]. In the third section we briefly define the tools of the performance evaluation framework we need. In the fourth one we give some details about the PETSc implementation of MGRIT and write a performance model to describe the expected performance gain, depending mainly on the number of processors, the number of time-discretization points and of grids levels. Finally, in the last section we introduce what we are currently working on and what are the next planned steps.

## 2 MGRIT Algorithm. Basic Idea

The basic idea of MGRIT comes from the two-grid formulation of the Parareal method [11] for solving an Ordinary Differential Equation (ODE) and its discretization

$$u_t = f(u, t), \text{ with } u(0) = u_0 \text{ and } t \in [0, T]. \quad (1)$$

$$u(t + \delta t) = \Phi(u(t), u(t + \delta t)) + g(t + \delta t). \quad (2)$$

where  $\Phi$  is a linear (or nonlinear) operator that encapsulates the chosen time stepping solver, and  $g$  incorporates all solution independent terms. The application of  $\Phi$  is either a matrix vector multiplication, e.g. forward Euler, or a spatial solver, e.g. backward Euler [13].

The MGRIT algorithm is detailed in several works [3, 7, 10, 13]. Briefly, let  $t_i = i\delta_t$ ,  $i = 0, 1, \dots, N$  be a discretization of  $[0, T]$  with spacing  $\delta_t = \frac{T}{N}$  (this mesh will be called F-grid), and  $t_j = j\Delta T$ , where  $j = 0, 1, \dots, N_\Delta$  with  $N_\Delta = \frac{N}{m}$

and  $m > 1$  (called C-grid). We rewrite the problem 2 on the F-grid, denoted as *Fine problem*:

$$A(u) = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -\Phi_0 & I & \cdots & 0 & 0 \\ 0 & -\Phi_1 & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \cdots & -\Phi_{N-1} & I \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_N \end{bmatrix} = g \tag{3}$$

that corresponds to a C-grid problem obtained by introducing the appropriate interpolation and restriction operators P and R (see definitions in [10] and description in [9]). Then the multigrid reduction approximates  $A_\Delta$  by  $B_\Delta$  which is based on a new coarse propagator  $\Phi_{\Delta,i}$  arising from the re-discretization of problem 2 on the C-grid, and which is less expensive to evaluate. In this way we have to solve the so-called *Coarse problem*:

$$B_\Delta = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -\Phi_{\Delta,0} & I & \cdots & 0 & 0 \\ 0 & -\Phi_{\Delta,1} & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \cdots & -\Phi_{\Delta,N-1} & I \end{bmatrix} \cdot \begin{bmatrix} u_{\Delta,0} \\ u_{\Delta,1} \\ \vdots \\ u_{\Delta,N} \end{bmatrix} = \begin{bmatrix} g_{\Delta,0} \\ g_{\Delta,0} \\ \vdots \\ g_{\Delta,0} \end{bmatrix} = g_\Delta \tag{4}$$

Let us assume that the model equations in (1) are linear so that the  $\Phi_i$  are linear, and we let  $\Phi_i \equiv \Phi, i = 1, 2, \dots, N$ . Then, Parareal can be derived as an approximate Schur-complement approach with F-relaxation (relaxation<sup>1</sup> applied on the so-called fine points, or F-points, that are the points on the F-grid and not also on the C-grid), i.e. a two-level multigrid method. Mainly MGRIT algorithm extends the Parareal approach on more grids. This means that it uses discretization, relaxation, restriction, and projection operators for each grid-level, according to different kinds of cycles. The key difference from Parareal relies on a new relaxation operator called the **FCF-relaxation**. In practice it is the application of the F-relaxation and the C-relaxation repeated one after another once or more times [7].

The MGRIT algorithm for solving the linear case, as detailed in [3], is listed in Algorithm 1.

where:

- $l$  is the current level,  $0 \leq l \leq L$  and  $L$  is the coarsest level,
- $m_l$  is the coarsening step at each level  $l$ , with  $m_0 = 1$ , and  $\delta_l$  is the discretization time step at each level  $l$ , where  $\delta_l = \delta_{l-1} \cdot m_l$
- $N_l$  is the number of time steps for each  $l$ , with  $N_0 > N_1 > \dots > N_L$  and  $A_l$  is the matrix at level  $l$
- $u^{(l)}$  and  $g^{(l)}$  are the solution and right hand side vectors at level  $l$ ,

---

<sup>1</sup> *Relaxation* meaning the solution of (3) by using an iterative method (see [12] for details).

**Algorithm 1.** MGRIT( $l$ ) - Linear MGRIT algorithm at level  $l$ 


---

```

if  $l$  is the coarsest level  $L$  then
  Solve the Coarse problem  $A_L \mathbf{u}^{(L)} = \mathbf{g}^{(L)}$ 
else
  Apply FCF-relaxation to  $A_l \mathbf{u}^{(l)} = \mathbf{g}^{(l)}$ 
  Compute and restrict residual using injection  $\mathbf{g}^{(l+1)} = R_{Inj}(\mathbf{g}^{(l)} - A_l \mathbf{u}^{(l)})$ 
  Recursively call MGRIT( $l + 1$ ) to solve on next level
  Correct using ideal interpolation  $\mathbf{u}^{(l)} \leftarrow \mathbf{u}^{(l)} + P \mathbf{u}^{(l+1)}$ 
end if

```

---

- $R_{Inj}$  is the restriction/injection operator from a level to the coarser one and  $P$  is the *ideal interpolation* (see [3]) corresponding to an injection from the coarser level to a finer one, followed by an F-relaxation with a zero right-hand side (see [13]).

### 3 Preliminary Concepts and Definitions

The increasing need for parallel and scalable software, ready to exploit the new exascale architectures, leads to the development of many performance models, mainly based on architecture features [19–23, 26] or especially made for chosen algorithm classes [25, 27–29]. The model we present here is mainly focused on the dependencies among the computational tasks of the algorithm and is meant to be as general as possible.

We start by giving the definition of *dependency relation* on a set.

**Definition 1 (Dependency relation).** *Let  $\mathcal{E}$  be a set and let  $\pi_{\mathcal{E}}$  be a strict partial order relation on  $\mathcal{E}$  describing a dependency relation between the elements. We say that any element of  $\mathcal{E}$ , say  $A$ , depends on another element of  $\mathcal{E}$ , say  $B$ , if  $A \pi_{\mathcal{E}} B$ , and we write  $A \leftarrow B$ . If  $A$  and  $B$  do not depend on each other we write  $A \not\leftarrow B$ .*

Then, consider the set of all the computational problems  $\Gamma$  and any element  $B_N \in \Gamma$  where  $N$  is the input data size, called the *problem size*. Any  $B_N$  can always be decomposed in at least one finite set of other computational problems, that we call *decomposition of  $B_N$* . Given a decomposition in  $k$  subproblems  $B_{N_i}$ , called  $D_k$ , and, taking into account the dependencies among the subproblems, we build a *dependency matrix*  $\mathcal{M}_{D_k}$  where in each row we essentially put subproblems independent of one another and dependent on those in the previous rows. Let us introduce the dependency relation  $\pi_{D_k}$  such that  $B_{N_i} \pi_{D_k} B_{N_j}$  with  $i \neq j$  if and only if the solution of  $B_{N_j}$  must be found before the one of  $B_{N_i}$ .

**Definition 2 (Dependency Matrix).** *Given the partially ordered set  $(D_k, \pi_{D_k})$ , the matrix  $M_{D_k}$ , of size  $r_{D_k} \times c_{D_k}$ , whose elements  $d_{i,j}$ , are s.t.  $\forall i \in [0, r_{D_k} - 1]$  and  $\forall s, j \in [0, c_{D_k} - 1]$  it is  $d_{i,j} \leftrightarrow d_{i,s}$  and s.t.  $\forall i \in [1, r_{D_k} - 1] \exists q \in [0, c_{D_k} - 1]$  s.t.  $d_{i,j} \leftarrow d_{i-1,q} \quad \forall j \in [0, c_{D_k} - 1]$ , while the other elements are set equal to zero, is called the *dependency matrix*.*

Given  $D_k$ ,  $c_{D_k}$  is the *concurrency degree* of  $B_N$ , and  $r_{D_k}$  is the *dependency degree* of  $B_N$ , according to the actual decomposition, so that the dependency degree measures the amount of dependencies intrinsic to the chosen decomposition. The number and size of sub-problems a problem is decomposed into determine the granularity of the decomposition. Granularity has a major consequence in the level of detail required for an algorithm to be analysed with this approach.

The decomposition matrix allows us to identify some properties of the algorithm design, such as the concurrency available in a problem when we choose a decomposition rather than another. So the first question must be about how to decompose the problem. That is pretty obvious, but it can lead to algorithm characteristics that we want to emphasize and possibly exploit.

At this point we define the Scale up, using the cardinality of two decompositions.

**Definition 3 (Scale Up).** *Let us consider the following two decomposition  $D_{k_i}$  and  $D_{k_j}$  of  $B_N$ , with cardinalities  $k_j \neq k_i$ , the ratio  $SC(D_{k_i}, D_{k_j}) := \frac{k_i}{k_j}$  is called scale-up factor of  $D_{k_j}$  measured with respect to  $D_{k_i}$ .*

The next step is to assign the identified subproblems to the computing machine. First we introduce the machine  $\mathcal{M}_P$  equipped with  $P \geq 1$  processing elements with specific logical-operational capabilities<sup>2</sup> called computing operators of  $\mathcal{M}_P$ , and denoted by the function<sup>3</sup>  $I[\cdot] : B_N \in \Gamma \rightarrow S(B_N) \in S$  where  $S$  is the set of the solutions of all the problems in  $\Gamma$  and  $S(B_N)$  is the solution of  $B_N$ . Given  $\mathcal{M}_P$ , the set without repetitions  $Cop_{\mathcal{M}_P} = \{I^j\}_{j \in [0, q-1]}$ , where  $q \in \mathbb{N}$ , characterizes logical-operational capabilities of the machine  $\mathcal{M}_P$ .

**Definition 4 (Algorithm).** *Given the problem decomposition  $D_k$ , an algorithm solving  $B_N$  on  $\mathcal{M}_P$ , is the partially ordered set  $(A_{k,P}, \pi_{A_{k,P}})$ , with not necessarily distinct elements, where  $A_{k,P} = \{I^{i_0}, I^{i_1}, \dots, I^{i_k}\}$  such that  $I^{i_j} \in Cop_{\mathcal{M}_P}$  and*

$$\forall B_{N_\nu} \in D_k(B_N) \quad \exists ! I^{i_j} \in A_{k,P} : I^{i_j}[B_{N_\nu}] = S(B_{N_\nu}). \tag{5}$$

*There is a bijective correspondence  $\gamma : B_{N_\nu} \in D_k \longleftrightarrow I^{i_j} \in A_{k,P}$ . Every ordered subset of  $A_{k,P}$  is called sub-algorithm of  $A_{k,P}$ .*

By virtue of the property 5, operators of  $A_{k,P}$  inherit the dependencies existing between subproblems in  $D_k$ , but not the independencies, because, for example, two operators may depend on the availability of computing units of  $\mathcal{M}_P$  during their executions [6].

Let  $AL_{B_N}$  (or simply  $AL$ ) be the set of algorithms that solve  $B_N$ , obtained by varying  $\mathcal{M}_P$ ,  $P$  and  $D_k$ . Let us associate each algorithm of  $AL$  to a decomposition suited for  $\mathcal{M}_P$ , which means that we introduce the surjective correspondence

<sup>2</sup> Such as basic operations (arithmetic, . . .), special functions evaluations (sin, cos, . . .), solvers (integrals, equations system, non linear equations. . .).

<sup>3</sup> An operator can be an algorithm itself, from a finer granularity point of view.

$\psi : A_{k,P} \in AL \longrightarrow D_k$ , which induces an equivalence relationship  $\varrho$  of  $AL$  to itself, such that

$$\varrho(A_{k,P}) = \{\widetilde{A_{k,P}} \in AL : \psi(\widetilde{A_{k,P}}) = \psi(A_{k,P})\}. \tag{6}$$

Therefore,  $\varrho(A_{k,P})$  is the set of algorithms of  $AL$  associated with the same decomposition  $D_k$ <sup>4</sup>. Hence,  $\varrho$  induces the quotient set  $\frac{AL}{\varrho}$ , the elements of which are disjoint subsets of  $AL$  determined by  $\varrho$ , that is they are *equivalence classes* under  $\varrho$ . In the following we consider  $A_{k,P}$  as a representative of its equivalence class in  $AL$ .

**Definition 5 (Complexity).** *The cardinality of  $A_{k,P}$  is called complexity of  $A_{k,P}$ . It is denoted as  $C(A_{k,P})$ . That is  $C(A_{k,P}) := \text{card}(A_{k,P}) = k$ .*

Notice that, by virtue of the property 5, it holds that

$$\text{card}(A_{k,P}) = \text{card}(D_k(\mathcal{B}_{N_r})) = k, \quad \forall A_{k,P} \in \varrho(A_{k,P}). \tag{7}$$

and so  $C(A_{k,P}) = k$  equals the number of non empty elements of  $M_{D_k}$  (see Definition 2). This means that each algorithm belonging to the same equivalence class according to  $\varrho$  has the same complexity. Thus an integer (the complexity) is associated with each element  $\varrho(A_{k,P})$  of quotient set  $\frac{AL}{\varrho}$  and induces an ordering relation between the equivalence classes in  $\frac{AL}{\varrho}$ . Therefore there is a minimum complexity for algorithms that solve the problem  $\mathcal{B}_{N_r}$ .

Let us better define the order relation on the algorithm set, as a second dependency relation  $\pi_{A_{k,P}}$  such that  $I^{ij} \pi_{A_{k,P}} I^{ir}$  with  $j \neq r$  if and only if  $I^{ij}$  solves a subproblem dependent on the one solved by  $I^{ir}$ , and/or the execution of  $I^{ij}$  needs to wait for the execution of  $I^{ir}$  to use the same computing resource.

**Definition 6 (Execution matrix).** *Given the partially ordered set  $(A_{k,P}, \pi_{A_{k,P}})$ , the matrix  $\mathcal{E}_{k,P}$ , of size  $r_{\mathcal{E}_{k,P}} \times c_{\mathcal{E}_{k,P}}$ , with<sup>5</sup>  $c_{\mathcal{E}_{k,P}} = P$ , whose elements  $e_{i,j}$ , are s.t.  $\forall i \in [0, r_{\mathcal{E}_{k,P}} - 1]$  and  $\forall s, j \in [0, P - 1]$  it is  $e_{i,j} \leftrightarrow e_{i,s}$  and s.t.  $\forall i \in [1, r_{\mathcal{E}_{k,P}} - 1] \exists q \in [0, P - 1]$  s.t.  $e_{i,j} \leftarrow e_{i-1,q} \quad \forall j \in [0, P - 1]$ , while the other elements are set equal to zero, is called the execution matrix.*

Inside an equivalency class, the algorithm solving a problem according to a decomposition that is executed on a machine with just one processor is a *sequential algorithm* and its execution matrix has just one column, since  $P = 1$ . In general, the execution matrix size describes the *cost* of the algorithm. In case of empty spaces in the matrix, they represent the algorithm *overhead*.

The number of rows  $r_{\mathcal{E}_{k,P}}$  is directly related to the execution time of the algorithm executed with that number  $P$  of processing units. Note that in order to compare two algorithms we must ensure that they are described by the same

<sup>4</sup> In this set the algorithms can have different value of  $P$ .

<sup>5</sup> In general  $c_E \leq P$ , but we can exclude cases where dependencies between subproblems do not allow us to use all the computing units available, i.e. in which  $c_E < P$ , because they can easily be taken back to the case where  $c_E = P$ .

kind of operators, or with the same granularity. In particular, if all the operators have the same execution time  $t$ , the *algorithm execution time* is proportional to  $r_{\mathcal{E}_{k,P}}$  and the *Speed Up* can be defined for an algorithm, in its equivalency class, as the ratio between its complexity and the number of rows. More specifically, if  $A_{k,P}$  is an algorithm built according to the decomposition  $D_k$  and executed on a machine with  $P$  processing units, we give the following definitions

**Definition 7 (Execution time).** *The quantity  $T(A_{k,P}) := r_{\mathcal{E}_{k,P}} \cdot t$  is called execution time of  $A_{k,P}$ .*

**Definition 8 (Speed Up).** *Given the algorithms  $A_{k,P}$  executed with  $P$  computing units the ratio  $S(A_{k,P}) := \frac{C(A_{k,P})}{r_{\mathcal{E}_{k,P}}}$  is called Speed Up of  $A_{k,P}$  in its equivalency class.*

This rewrites the classical speed up formula, so we can say that the ideal value is  $P$ , and we can also show that, varying  $P$ , it is limited by the concurrency degree of the problem in the same decomposition.

Briefly, given two different decompositions  $D_{k_i}$  and  $D_{k_j}$ , with  $k_j \neq k_i$ , given two different machines with two different number of processors  $P_1 = 1$  and  $P > 1$ , for the two corresponding algorithms we define the *General Speed Up* of the parallel one respect to the sequential one, as the product of the *Scale Up* between the two decompositions and the classical speed up of the parallel one.

**Definition 9 (General Speed Up).** *The ratio*

$$GS(A_{k_j,P}, A_{k_i,1}) := SC(D_{k_i}, D_{k_j}) \cdot S(A_{k_j,P}) = \frac{k_i}{k_j} \cdot \frac{k_j}{r_{\mathcal{E}_{A_{k_j,P}}}} = \frac{r_{\mathcal{E}_{A_{k_i,1}}}}{r_{\mathcal{E}_{A_{k_j,P}}}}$$

*is called General Speed Up of  $A_{k_j,P}$  respect to  $A_{k_i,1}$ .*

Note that the ideal value of the General Speed Up is not limited by the number of processing units  $P$ .

## 4 The PETSc Based Implementation of MGRIT for the Linear Case

At the top of the PETSc hierarchy there are the object to solve ODEs and nonlinear systems, built on other objects needed to solve linear systems. In particular, the TS (TimeStepping) library provides a framework to solve ODEs and DAEs arising from the discretization of time-dependent PDEs. Users shall essentially provide the F function, the G function (if nonzero), the initial condition and the Jacobian.

We are now developing a kind of “parallel TS”, based on MGRIT, to be compared with the already provided sequential ones. The idea is to “simply” solve the linear system using a linear solver with a multigrid preconditioner.

The first step of the implementation is to provide the data structure to handle the time dimension together with the space ones, in the context of the PETSc

DM or Distributed objects. This means (1) to provide the basic operations for the new type, (2) to handle the coarsening factor, and (3) to provide the user interface to the function which describes the way of operating for  $\Phi$ , that is the spacial solver, and the time discretization calls.

Everything about the coarsening of the grids along the levels, the distribution of the points among the processes and the communications is handled by the PETSc DA (DistributedArray) object linked to the solvers in a fully transparent fashion. Users can tune the behavior of the solver and thus the actual structure of the scheme through the option setting (including tolerance and initial guess for all the operators involved) at runtime.

The second step is the implementation of F- and C-relaxations that must be set as down and up smoothers of the multigrid scheme, tunable (even at runtime) by the user to fit his/her own problem, according to the PETSc design. Users will still control all the parameters and solver choices even at runtime.

#### 4.1 The Performance Model

First, we notice that the application of  $\Phi$  is the dominant task. In case of explicit time stepping each application of  $\Phi$  corresponds to a matrix-vector product, whose execution time will be constant. In case of implicit time stepping, each application of  $\Phi$  equates itself to a system solver. Using an optimal space solver and fixing the stopping tolerance or the number of iterations and the initial guess choice for the spacial solver, the work required for one time step evaluation can be considered constant across all time levels (and associated time step sizes)<sup>6</sup> [10].

Let  $\Phi_{i,j}$  be the subproblem of evaluating the function  $\Phi$  at any instant  $u_{\delta_{i,j}}$ , with  $i = 0, \dots, L$  and  $j = 1, \dots, N_L$ , and  $\phi_{i,j}$  the operator to solve it. Notice that there is no evaluation at the first instant of each grid.

Let  $N_{F_l} := N_l - N_{l+1}$  be the number of F-points and  $N_{l+1}$  be the number of C-points at each level  $l$  of MGRIT algorithm. The relation between the number of F-points and C-points depends on the coarsening factor  $m$  that can be the same for all levels or possibly different for each one. In Algorithm 1, we note that if  $L$  is the coarsest level, and the solver of the system on the coarsest-grid is sequential, this will involve at least one  $\phi$ -execution for each time step on the  $L$ -th grid. It means that if  $L$  is the coarsest level there are  $N_L$  executions of  $\phi$ . Otherwise, for each level  $l < L$ ,

- the FCF-relaxation involves  $N_{F_l}$  F-relaxation steps (or  $\phi$ -executions), which can be performed in parallel,  $N_{l+1}$  C-relaxation steps (or  $\phi$ -executions), which can be performed in parallel, F-relaxation steps (or  $\phi$ -executions), which can be performed in parallel,
- computing the residual requires one  $\phi$ -execution for each time step on the  $(l + 1)$ -th grid, that is  $N_{l+1}$ , which can be performed in parallel,

---

<sup>6</sup> For the sake of brevity we discuss here the execution of only one V-Cycle, as described in Algorithm 1. The number of iterations of the multigrid cycles can be considered later.



- the ideal interpolation requires  $N_{F_l}$  F-relaxation steps (or  $\phi$ -executions), which can be performed in parallel.

Let us now define the *dependency matrix*  $\mathcal{M}_D$  (see Definition 2 in Sect. 3) of the time-space problem to be solved, according to its decomposition in the space subproblems  $\Phi_{i,j}$ , for  $i = 0, \dots, L$  and  $j = 1, \dots, Np$  where  $Np \in \{N_{F_l}, N_{l+1}, N_L\}$  and where in each row we essentially put subproblems independent of one another and dependent on those in the previous rows (the  $\mathcal{M}_D$  matrix is well described in [9]).

The *concurrency degree* of the problem decomposed in this way is  $c_D$ , i.e. the maximum number of simultaneous  $\Phi$  evaluations. Since  $N_{F_l} > N_{l+1}$  and  $N_{F_l} > N_{F_{l+1}}$ , which means that the number of F-points at any level is greater than the number of C-points at the same level and greater than the F-points at the next level,  $c_D = N_{F_0}$ .

The *dependency degree* is  $r_D = 5 \cdot L + N_L$ , since, with  $L+1$  levels, we have (1) 3 rows for each FCF-relaxation, that means  $3 \cdot L$  rows, (2) 1 row for each residual computation, that means  $L$  rows, which are the longest rows in the matrix, or with the largest numbers of columns, (3)  $N_L$  rows for the coarsest-grid solver, (4) 1 row for each ideal interpolation (F-relaxation), that means  $L$  rows.

Consider now a computing architecture with  $P$  processing elements, where  $P = \frac{c_D}{np}$  (this condition states that the points on the finest grid are equally distributed among the processors, that is  $c_D$  is a multiple of  $P$ ) and  $np \in \mathbf{N}$  and  $P \leq N_L$  (this condition states that on the coarsest grid each processor holds at least one point).

We can define the *execution matrix*  $\mathcal{E}_P$  of MGRIT (see Definition 6, in Sect. 3), consisting of the operators  $\phi_{i,k \cdot P+j}$ , with  $i = 0, \dots, L$  and  $j = 1, \dots, P$  and  $k = 1, \dots, \frac{N_{F_l}}{P}$  for F-relaxation or  $k = 1, \dots, \frac{N_{l+1}}{P}$  for C-relaxation and residual computation, considering that, for each level, the number of points of the grid is a multiple of  $P^7$  (the  $\mathcal{E}_P$  matrix is well described in [9]).

Consider now the algorithm  $A_{N_0,1}$ , which solves (2) with the same discretization in time on the finest grid (same initial guess and same tolerance) but without introducing MGR or any parallelism, that means using a sequential time-stepping approach with the same discretization techniques and parameters as used by MGRIT on the finest grid.  $A_{N_0,1}$  is made of  $N_0$  executions of  $\phi$ , leading to the execution matrix  $\mathcal{E}_1$  with one column and  $N_0$  rows (the  $\mathcal{E}_1$  matrix is well described in [9]).

---

<sup>7</sup> This is without loss of generality, as, otherwise, the number of rows is still the same but with just some empty elements.

We can prove the following (proof in [9]):

**Theorem 1.** *Let us assume that MGRIT algorithm runs on a computing architecture with  $P \leq N_L$  processing elements, where  $P = \frac{N_0}{np}$  and  $np \in \mathbf{N}$ . Let  $t_\phi$  be the execution time of  $\phi$ ,  $\forall l \in [0, L]$ .*

*Let us say that it reaches the same accuracy as  $A_{N_0,1}$  in  $\nu$  iterations. Then the general speed-up  $GS(MGRIT_{N_{MGRIT},P}, A_{N_0,1})$  of MGRIT with respect to  $A_{N_0,1}$  is*

$$GS(MGRIT_{N_{MGRIT},P}, A_{N_0,1}) = \frac{N_0}{\nu \cdot \left( \sum_{l=0}^{L-1} \left( 3 \cdot \frac{N_{Fl}}{P} + 2 \cdot \frac{N_{l+1}}{P} \right) + N_L \right)} \quad (8)$$

## 5 Conclusions and Future Work

Summarizing, we introduced a mathematical framework to propose a speed-up model for our implementation of MGRIT algorithm. It describes the impact of several factors (i.e. the number of time steps and the number of processors) on the dependencies among operators and thus on the algorithm performance, regardless of the execution of  $\phi$ . Any choice related to its implementation affects the unit time  $t_\phi$  and/or the numerical accuracy of the results. The required accuracy will limit one or more parameters in a way that is beyond the scope of this paper. If  $\Phi$  is nonlinear, each application becomes an iterative nonlinear solver, whose conditioning usually depends on the time step size [13].

The main topics we are now focusing on are the following:

- definition of a *memory access matrix* to take into account the communications that can significantly affect the *software speed up* limiting the number of processing elements and grid levels to be used,
- parallel implementation of  $\Phi$ , to handle different levels of parallelism, exploiting the capabilities of heterogeneous architectures, such as multicore clusters and GPUs, to efficiently treat the parallelism in the spacial dimension [14–17],
- validation of all the results arising from this designing approach through the execution of the resulting software prototype on a suited set of problems. The validation activities should provide the PETSc users with the needed guidelines to efficiently use the new TS object to solve their problems.

**Acknowledgments.** The research was carried out during a collaboration between the University of Naples Federico II (Naples, Italy) and the Argonne National Laboratory (Chicago, Illinois, USA).

It has received funding from European Commission under H2020-MSCA-RISE NASDAC project (grant agreement n. 691184).

This work was also supported by GNCS INdAM.

## References

1. Balay, S., et al.: *Petsc User Manual. Revision 3.7 Report number ANL-95/11 Rev. 3.7 127241*, United States: N. p., 2016. Web (2016). <https://doi.org/10.2172/1255238>
2. Murli, A., Boccia, V., Carracciolo, L., D'Amore, L., Laccetti, G., Lapegna, M.: Monitoring and migration of a PETSc-based parallel application for medical imaging in a grid computing PSE. In: Gaffney, P.W., Pool, J.C.T. (eds.) *Grid-Based Problem Solving Environments. ITIFIP*, vol. 239, pp. 421–432. Springer, Boston, MA (2007). [https://doi.org/10.1007/978-0-387-73659-4\\_25](https://doi.org/10.1007/978-0-387-73659-4_25)
3. Falgout, R.D., Friedhoff, S., Kolev, T.V., MacLachlan, S.P., Schroder, J.B.: Parallel time integration with multigrid. *SIAM J. Sci. Comput.* **36**(6), C635–C661 (2014). <https://doi.org/10.1137/130944230>
4. XBraid: Parallel multigrid in time. <http://llnl.gov/casc/xbraid>
5. Carracciolo, L., D'Amore, L., Mele, V.: Toward a fully parallel multigrid in time algorithm in PETSc environment: a case study in ocean models. In: *IEEE proceedings of International Conference on High Performance Computing & Simulation (HPCS) 2015*, Amsterdam, pp. 595–598 (2015). <https://doi.org/10.1109/HPCSim.2015.7237098>
6. Tjaden, G.S., Flynn, M.J.: Detection and parallel execution of independent instruction. *IEEE Trans. Comput.* **19**(10), 889–895 (1970). <https://doi.org/10.1109/T-C.1970.222795>
7. Gahvari, H., et al.: A performance model for allocating the parallelism in a multigrid-in-time solver. In: *Proceedings of 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS)*, Salt Lake City, UT, 2016, art. no. 7836411, pp. 22–31. IEEE Press (2017). <https://doi.org/10.1109/PMBS.2016.008>
8. D'Amore, L., Mele, V., Laccetti, G., Murli, A.: Mathematical approach to the performance evaluation of matrix multiply algorithm. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) *PPAM 2015. LNCS*, vol. 9574, pp. 25–34. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-32152-3\\_3](https://doi.org/10.1007/978-3-319-32152-3_3)
9. Mele, V., Costantinescu, E.M., Carracciolo, L., D'Amore, L.: A PETSc parallel-in-time solver based on MGRIT algorithm. *Concurrency Comput.: Practice Exp.* e4928 (2018). <https://doi.org/10.1002/cpe.4928>
10. Schroder, J.B., Falgout, R.D., Manteuffel, T.A., O'Neill, B.: Multigrid reduction in time for nonlinear parabolic problems: a case study. *SIAM J. Sci. Comput.* **39**(5), S298–S322 (2017)
11. Lions, J.L., Maday, Y., Turinici, G.: A parareal in time discretization of PDEs. *Comptes Rendus de l'Academie des Sci. - Ser. I - Math.* **332**, 661–668 (2001). [https://doi.org/10.1016/S0764-4442\(00\)01793-6](https://doi.org/10.1016/S0764-4442(00)01793-6)
12. Gander, M.J., Vandewalle, S.: Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput.* **29**, 556–578 (2007). <https://doi.org/10.1137/05064607X>
13. Falgout, R.D., Friedhoff, S., Kolev, T.V., MacLachlan, S.P., Schroder, J.B., Vandewalle, S.: Multigrid methods with space-time concurrency. *SIAM J. Sci. Comput.* (2015). <https://doi.org/10.1007/s00791-017-0283-9>
14. Cuomo, S., De Michele, P., Piccialli, F.: 3D data denoising via nonlocal means filter by using parallel GPU strategies. *Comput. Math. Methods Med.* **2014**, 14 (2014). <https://doi.org/10.1155/2014/523862>. Article ID 523862

15. Cuomo, S., De Michele, P., Piccialli, F.: A (multi) GPU iterative reconstruction algorithm based on Hessian penalty term for sparse MRI. *Int. J. Grid Utility Comput.* **9**(2), 139–156 (2018). <https://doi.org/10.1504/IJGUC.2018.091720>
16. Piccialli, F., Cuomo, S., De Michele, P.: A regularized MRI image reconstruction based on Hessian penalty term on CPU/GPU systems. *Procedia Comput. Sci.* **18**, 2643–2646 (2013). <https://doi.org/10.1016/j.procs.2013.06.001>. ISSN 1877–0509
17. D’Amore, L., Marcellino, L., Mele, V., Romano, D.: Deconvolution of 3D fluorescence microscopy images using graphics processing units. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011. LNCS, vol. 7203, pp. 690–699. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31464-3\\_70](https://doi.org/10.1007/978-3-642-31464-3_70)
18. Maddalena, L., Petrosino, A., Laccetti, G.: A fusion-based approach to digital movie restoration. *Pattern Recogn.* **42**(7), 1485–1495 (2009)
19. Gregoretti, F., Laccetti, G., Murli, A., Oliva, G., Scafuri, U.: MGF: a grid-enabled MPI library. *Future Gen. Comput. Syst.* **24**(2), 158–165 (2008)
20. Laccetti, G., Lapegna, M., Mele, V., Romano, D., Murli, A.: A double adaptive algorithm for multidimensional integration on multicore based HPC systems. *Int. J. Parallel Program.* **40**(4), 397–409 (2012). <https://doi.org/10.1007/s10766-011-0191-4>
21. Laccetti, G., Lapegna, M., Mele, V., Romano, D.: A study on adaptive algorithms for numerical quadrature on heterogeneous GPU and multicore based systems. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013. LNCS, vol. 8384, pp. 704–713. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-55224-3\\_66](https://doi.org/10.1007/978-3-642-55224-3_66)
22. Laccetti, G., Lapegna, M., Mele, V., Montella, R.: An adaptive algorithm for high-dimensional integrals on heterogeneous CPU-GPU systems. *Concurrency Comput.: Practice Exp.* **2018**, e4945 (2018). <https://doi.org/10.1002/cpe.4945>
23. Laccetti, G., Lapegna, M., Mele, V.: A loosely coordinated model for heap-based priority queues in multicore environments. *Int. J. Parallel Program.* **44**(4), 901–921 (2016). <https://doi.org/10.1007/s10766-015-0398-x>
24. D’Amore, L., Casaburi, D., Galletti, A., Marcellino, L., Murli, A.: Integration of emerging computer technologies for an efficient image sequences analysis. *Integr. Comput.-Aided Eng.* **18**(4), 365–378 (2011). <https://doi.org/10.3233/ICA-2011-0382>
25. Arcucci, R., D’Amore, L., Celestino, S., Laccetti, G., Murli, A.: A scalable numerical algorithm for solving Tikhonov regularization problems. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015. LNCS, vol. 9574, pp. 45–54. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-32152-3\\_5](https://doi.org/10.1007/978-3-319-32152-3_5)
26. Boccia, V., Carracciuolo, L., Laccetti, G., Lapegna, M., Mele, V.: HADAB: enabling fault tolerance in parallel applications running in distributed environments. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011. LNCS, vol. 7203, pp. 700–709. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31464-3\\_71](https://doi.org/10.1007/978-3-642-31464-3_71)
27. Murli, A., Cuomo, S., D’Amore, L., Galletti, A.: Numerical regularization of a real inversion formula based on the Laplace transform’s eigen function expansion of the inverse function. *Inverse Probl.* **23**(2), 713 (2007)

28. D'Amore, L., Campagna, R., Mele, V., Murli, A., Rizzardi, M.: ReLaTive. An Ansi C90 software package for the real Laplace transform inversion. *Numer. Algorithms* **63**(1), 187–211 (2013). <https://doi.org/10.1007/s11075-012-9636-0>
29. Murli, A., D'Amore, L., Laccetti, G., Gregoretti, F., Oliva, G.: A multi-grained distributed implementation of the parallel Block Conjugate Gradient algorithm. *Concurrency Comput. Practice Exp.* **22**(15), 2053–2072 (2010). <https://doi.org/10.1002/cpe.1548>