Centro Euro-Mediterraneo sui Cambiamenti Climatici

Research Papers
Issue RP0281
June 2017

ASC - Advanced Scientific Computing Division

Driving NEMO towards Exascale Step 3: Performance Analysis of a Parallel-in-Time PDE Solver - based on MGRIT Algorithm.

By **L. D'Amore**
University of Naples Federico & Fondazione CMCC
luisa.damore@unina.it

V. Mele
University of Naples Federico II
valeria.mele@unina.it

G. Aloisio
University of Salento & Fondazione CMCC
giovanni.aloisio@unisalento.it

SUMMARY Parallel in Time (PinT) methods have received renewed interest over the last decade for improving the algorithmic scalability on emerging computer architectures of time-dependent large scale simulations. However, a PinT approach may exhibit higher overheads and not necessarily a space-and-time decomposition leads to a parallel algorithm with the highest performance. Here we consider MGRIT (MultiGrid-In-Time) algorithm, which is based on MultiGrid Reduction (MGR). We provide a mathematical model for analysing performances of MGRIT algorithm, by determining the benefits arising from the decomposition. A set of matrices (decomposition, execution and storage) highlights fundamental characteristics of the algorithm, such as the inherent parallelism, some sources of overhead and memory occupancy, respectively. The aim of the proposed performance analysis is to address the algorithmic strong and weak scaling of MGRIT, regarded as a parallel iterative algorithm proceeding along the time dimension. The analysis allows us to a-priori determine the correct number of MGRIT time-levels as well as the suitable number of processing elements for efficiently implementing the algorithm.

Keywords: PinT, MGRIT, Perfomance Analysis

INTRODUCTION AND DESCRIPTION OF THE WORK

The efficient numerical solution of large systems of discretized partial differential equations (PDEs), such that describing the ocean circulation models, requires hierarchical algorithms which ensure a reduction of both short- and long-frequencies error components. One of the most important advances during the last three decades, was due to the multigrid approach because of its optimal algorithmic scaling for both parallel communication and number of operations. The MultiGrid (MGR) algorithm requires a set of meshes defined on a hierarchy of grids with different sizes obtained by successive refinement. Each iteration typically consists of the following steps: smooth errors at current grid; transfer residual to next coarser grid; correct iterate using the coarser residual (recursively). According to the Domain Decomposition approach, parallelism is introduced by dividing the spatial domain of interest into subdomains (one for each processor). Each processor is then responsible for updating the unknowns associated within its subdomain only. In particular, in standard parallel MGR algorithm computations within a mesh are performed in parallel and each level in the hierarchy is addressed one at a time.

One approach to achieve parallelism in time direction is Multigrid-In-Time (MGRIT) algorithm [7], based on MGR techniques. The MGRIT algorithm calls a time-stepping solver that uses coarse time-grids to accelerate convergence to the solution on the finest time-grid. In a simple two-level setting, MGRIT is equivalent to the Parareal method [11], which is perhaps the most popular parallel-in-time method.

However, to achieve the full benefit of computing multiple time steps at once, space-and-time multigrid methods, where time is another dimension in the grid, should be considered.

While MGRIT is implemented in the software package XBraid[13], we are developing a multilevel parallel algorithm which is based on PETSc (Portable, Extensible Toolkit for Scientific computing) and fuses the MGRIT algorithm with the MGR in space methods [3, 6]. PETSc is characterized by a considerable endowment of implementations of numerical methods and algorithms, including those used to solve systems of linear equations. PETSc is modular, organized in different levels of abstraction, and it provides data structures that hide, to the final user, the complexity of the interprocess communications in distributed memory environments. The tools available in the PETSc software library, allow us to implement the multilevel parallel algorithm which uses:

- across the level, the MGRIT algorithm;
- within each level, the parallel algebraic multigrid method.

The aim of the proposed performance analysis is to address the algorithmic strong and weak scaling of MGRIT, regarded as a parallel iterative algorithm proceeding along the time dimension. The performance model in [9], instead, aims to select the best parallel configuration (i.e. how much parallelism to devote to space vs. time). Our aim is to give the MGRIT user the opportunity of predicting the performance gain that she/he can achieve when using the MGRIT approach instead of a time-marching integrator. We believe that both performance models could be employed for the successful implementation of the MGRIT algorithm.

MGRIT ALGORITHM. BASIC IDEA

Throughout this paper, we assume the reader to have some basic knowledge of geometric

multigrid. In particular, she/he should be familar with the fundamental principles (smoothing and coarse-grid correction) and with the recursive definition of multigrid cycles.
We analyze the performance of MGRIT algorithm with respect the parallelization in time dimension [9], using results presented in [1, 5], intentionally overlooking the spacial parallelism. To this aim, we give a brief introduction of MGRIT algorithm, by using the simplest viewpoint based on linear algebra operations.
For simplicity, we focus on a system of Ordinary Differential Equations (ODEs), such as in the method of lines for approximating a system of time-dependent PDEs, described as

$$\frac{du(t,x_j)}{dt} = f(t,u(t,x_j)) \quad j = 1,\dots,M. \quad (1)$$

with $u(0) = u_0$ and $t \in [0,T]$.
Let $t_i = i\delta_t$, $i = 0,1,...,N$ be a Fine-discretization of $[0,T]$ with spacing $\delta_t = \frac{T}{N}$ (this mesh will be called F-grid), and for each x_j, let u_i be an approximation of $u(t_i,x_j)$. This means that, in order to only highlight the dependence of $u(t,x)$ on t_i, we overcome its dependence on x_j. A numerical approximation of (1) based on one-step time marching is (see Figure 1):

$$\begin{cases} u_0 := u(0) \\ u_i = \Phi_i(u_{i-1}) + b_i, \quad i = 1,2,...,N. \end{cases} \quad (2)$$

For simplicity, we assume that the model equations in (1) are linear so that functions Φ_i (the so-called propagators) are linear, and we let $\Phi_i \equiv \Phi$ (i.e. Φ does not depend on t_i), $i = 1,2,\dots,N$. If we introduce the matrix $A \in \Re^{N\times N}$ where

$$A = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -\Phi & I & \cdots & 0 & 0 \\ 0 & -\Phi & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -\Phi & I \end{bmatrix} \quad (3)$$

the numerical scheme in (2) can also be expressed as the solution of the following system:

$$A \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{bmatrix} \quad (4)$$

where $b_0 = u(0)$. We denote the system in (4) as F-grid system.

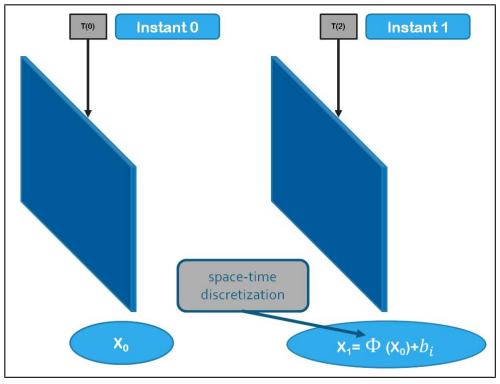

Figure 1: The role of the function Φ between two instants.

Let $\Delta T = m\delta_t$, with $m > 1$, be the step size of a coarse discretization of $[0,T]$, such that $t_j = j\Delta T$, where $j = 0,1,...,N_\Delta$ and $N_\Delta = \frac{N}{m}$. The Coarse-grid (C-grid) discretization of (2) is obtained by defining the so-called C-grid propagators Φ_Δ and we get the following C-grid linear system:

$$\hat{A} \cdot \begin{bmatrix} u_{\Delta_0} \\ u_{\Delta_1} \\ \vdots \\ u_{\Delta_{N_\Delta}} \end{bmatrix} = \begin{bmatrix} b_{\Delta_0} \\ b_{\Delta_1} \\ \vdots \\ b_{\Delta_{N_\Delta}} \end{bmatrix} \quad (5)$$

where $\hat{A} \in \Re^{N_\Delta \times N_\Delta}$ is

$$\hat{A} = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -\Phi_\Delta & I & \cdots & 0 & 0 \\ 0 & -\Phi_\Delta & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \cdots & -\Phi_\Delta & I \end{bmatrix} \quad (6)$$

Note that as $N_\Delta < N$, matrix $\hat{A}$ has fewer rows and columns than A.

Parareal algorithm, first introduced in [11], solves (2) combining the solution on the F-grid and on the C-grid. It can be viewed as a 2-grid scheme, as summarized in the following (some details in [10]):

Parareal algorithm:

1. *relax* on the F-grid (this operation is called F-relaxation),
2. project the residual to the C-grid
3. solve the Coarse problem,
4. project the Coarse solution on the corresponding points in the C-grid,
5. relax again on the F-grid.

MGRIT algorithm extends the Parareal approach on more grids: this means that MGRIT uses discretization, relaxation, restriction, and projection operators for each grid-level, according to different kinds of cycles (see Figure 2). The key difference is in a new relaxation operator called **FCF-relaxation** (see Figure 3), summarized as follows:

FCF-relaxation:

1. relax at points that are only on the Finest-grid,
2. relax at points that are only on the Coarse-grid,
3. relax again at points that are only on the Finest-grid.

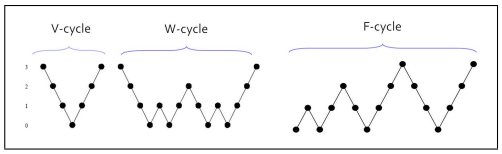

Figure 2: MGRIT defines V-, W- and F-cycle versions.

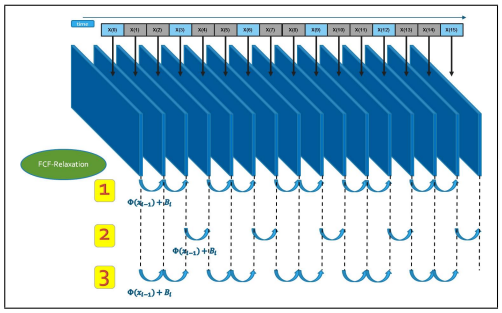

Figure 3: FCF-relaxation.

Let us introduce the following parameters:

- l: the current level,
- L: the coarsest level,
- m_l: the coarsening step at each level l, with $m_0 = 1$
- δ_l: the discretization time step at each level l, where $\delta_l = \delta_{l-1} \cdot m_l$
- N_l: the number of time steps for each l, with $N_0 > N_1 > ... > N_L$,
- A_l: the matrix at level l,
- $u^{(l)}$ and $b^{(l)}$: the solution and right hand side vectors at level l,
- R_I: the restriction/injection operator from a level to the coarser one,

- P: the *ideal interpolation* (see [7]) corresponding to an injection from the coarser level to a finer one, followed by an F-relaxation with a zero right-hand side (see [8]).

then MGRIT algorithm can be listed in Figure 4 and detailed in [7].

MGRIT(l)
if l is the coarsest level, L
- Solve coarse-grid system $A_L\mathbf{u}^{(L)} = \mathbf{g}^{(L)}$.

else
- Relax on $A_l\mathbf{u}^{(l)} = \mathbf{g}^{(l)}$ using FCF-relaxation.
- Compute and restrict residual using injection, $\mathbf{g}^{(l+1)} = R_I(\mathbf{g}^{(l)} - A_l\mathbf{u}^{(l)})$.
- Solve on next level: **MGRIT**$(l+1)$.
- Correct using "ideal interpolation", $\mathbf{u}^{(l)} \leftarrow \mathbf{u}^{(l)} + P\mathbf{u}^{(l+1)}$.

end

Figure 4: MGRIT algorithm

A MATRIX-BASED PERFORMANCE MODEL

The performance model we are going to introduce takes into account the dependence relationship among component parts of the given computational problem, let us say $\mathcal{B}_{N_r}$.

(**Dependency Group**) Let $(\mathcal{E}, \pi)$ be a group and let $\pi_\mathcal{E}$ be a strict partial order relation on $\mathcal{E}$, which is compatible with π. We say that any two elements of $\mathcal{E}$, let us say A and B, are dependent each other if $A\pi_\mathcal{E}B$, and we write $A \leftarrow B$. If A does not depend on B we write $A \nleftarrow B$. The group $(\mathcal{E}, \pi)$ equipped with $\pi_\mathcal{E}$ is called dependency group and it is denoted as $(\mathcal{E}, \pi, \pi_\mathcal{E})$.

Remark: Since $\pi_\mathcal{E}$ is transitive, any two elements of $\mathcal{E}$, let us say A and B, are independent if there is no any relationship between them. In this case we write $A \nleftarrow B$ and $B \nleftarrow A$, or even $A \nleftrightarrow B$.

(**Dependency Matrix**) Given $(\mathcal{E}, \pi, \pi_\mathcal{E})$, the matrix $M_D = (d_{i,j})$, where $d_{i,j} \in (\mathcal{E}, \pi)$, of size $r_D \times c_D$, whose non zero elements are such that $\forall i \in [0, r_D - 1]$

$$d_{i,j} \nleftrightarrow d_{i,s} \quad , \quad \forall s,\, j \in [0, c_D - 1] \qquad (7)$$

and $\forall i \in [1, r_D - 1], \quad \exists q \in [0, c_D - 1]$ s.t.

$$d_{i,j} \leftarrow d_{i-1,q} \quad , \quad \forall j \in [0, c_D - 1], \qquad (8)$$

while the others elements are set equal to zero, is said the dependency matrix.

Remark: c_D is said the concurrency degree of $(\mathcal{E}, \pi, \pi_\mathcal{E})$ and r_D is the said the dependency degree of $\mathcal{E}$. Matrix M_D is unique unless an exchange between the rows is performed. This means that r_D is well determined.
Concurrency degree measures the intrinsic concurrency among sub-problems of $(\mathcal{E}, \pi, \pi_\mathcal{E})$. It is obtained as the number of columns of M_D. Therefore, if there are only non zero elements $(\mathcal{E}, \pi, \pi_\mathcal{E})$ has the highest intrinsic concurrency.

Let $S(\mathcal{B}_{N_r})$ denote the solution of $\mathcal{B}_{N_r}$. Here, for the sake of simplicity, we assume that $S(\mathcal{B}_{N_r})$ exists and it is unique.

(**Decomposition**) Any finite set of computational problems $\mathcal{B}_{N_k}$ where $N_i < N_r$, $i = 0, \ldots, k$, and

$$\sum_{i=0}^{k-1} N_i \geq N_r$$

and such that $\mathcal{B}_{N_r} \leftarrow \mathcal{B}_{N_i}$ is called a decomposition of $\mathcal{B}_{N_r}$. $\mathcal{B}_{N_i}$ denotes a sub-problem of $\mathcal{B}_{N_r}$. A decomposition of $\mathcal{B}_{N_r}$ is denoted as

$$D_k(\mathcal{B}_{N_r}) = \{\mathcal{B}_{N_0}, \ldots, \mathcal{B}_{N_{k-1}}\} \qquad (9)$$

The set of all the decompositions of $\mathcal{B}_{N_r}$ is denoted as $\mathcal{DB}_{N_r}$.

Remark (**Decomposition matrix**): In order to capture interactions among component parts (or sub-problems) of $\mathcal{B}_{N_r}$, we use the dependency matrix on $D_k(\mathcal{B}_{N_r})$. More precisely, we introduce the group $(D_k(\mathcal{B}_{N_r}), g_{sol})$ where g_{sol} is any application between the solutions $S(\mathcal{B}_{N_i})$ and $S(\mathcal{B}_{N_j})$ of two elements of $D_k(\mathcal{B}_{N_r})$, equipped with the strict partial order relation $\pi_{D_k(\mathcal{B}_{N_r})}$. Then, we construct the (unique) **decomposition matrix** M_D corresponding to the decomposition $D_k(\mathcal{B}_{N_r})$. Given $D_k(\mathcal{B}_{N_r})$, c_D is the (unique) concurrency degree of $\mathcal{B}_{N_r}$ and r_D is the (unique) dependency degree of $\mathcal{B}_{N_r}$. Concurrency degree measures the intrinsic concurrency among sub-problems of $\mathcal{B}_{N_r}$. If there are not empty elements, the problem $\mathcal{B}_{N_r}$ has the highest intrinsic concurrency.
The decomposition matrix allows us to identify the concurrency that is available in a problem and decompose it into parts that can be executed in parallel. The next step is to take these parts and assign them (i.e., the mapping step) onto the computing machine.

In the literature, an algorithm is any procedure consisting of finite number of unambiguous rules that specify a finite sequence of operations to reach a solution to a problem or a specific class of problems. Here we define an algorithm as a proper set of operators which solves $\mathcal{B}_{N_r}$.
Let $\mathcal{M}_P$ be the reference computing machine consisting of $P \geq 1$ processing elements with specific logical-operational capabilities.

(**Computing Operators**) An operator I^j of $\mathcal{M}_P$ is the correspondence between $\Re^s$ and $\Re^t$, where $s, t \in \mathbb{N}$ are positive integers.

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 .
Finally, we say that operators, properly organized, provide the solution to $\mathcal{B}_{N_r}$, as stated in the following

(**Solvable Problems**) $\mathcal{B}_{N_r}$ is solvable in $\mathcal{M}_P$ if

$$\exists D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r}$$

such that

$$\forall \mathcal{B}_{N_j} \in D_k \quad \exists I^j \in Cop_{\mathcal{M}_P},$$

and

$$I^j[\mathcal{B}_{N_j}] = S(\mathcal{B}_{N_j})$$

that is, if it exists any relation

$$\theta : \mathcal{B}_{N_j} \in D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r} \longmapsto I^j \in Cop_{\mathcal{M}_P}. \tag{10}$$

In particular, we say that decomposition is elementary if θ is a function. From now on, we consider solvable problem $\mathcal{B}_{N_r}$ and elementary decompositions $D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r}$ fixed [1] .

(**Algorithm**) An Algorithm solving $\mathcal{B}_{N_r}$, indicated as $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$ is a sequence of elements (not necessarily distinct) of $Cop_{\mathcal{M}_P}$

$$A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P} = \{I^{i_0}, I^{i_1}, ...I^{i_k}\}$$

such that

$$I^{i_k} \circ I^{i_{k-1}} \circ ... \circ I^{i_0}[\mathcal{B}_{N_r}] = S(\mathcal{B}_{N_r}),$$

where

$$I^{i_j} \in Cop_{\mathcal{M}_P} \text{ and } j \in [0, card(Cop_{\mathcal{M}_P}) - 1]$$

and such that there is a bijective correspondence

$$\gamma : \mathcal{B}_{N_\nu} \in D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r} \longleftrightarrow I^{i_j} \in A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P} \tag{11}$$

[1] There is no loss of generality.

Every sub-sequence of $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$ is a sub-algorithm of $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$.

For simplicity of notations and when there is no ambiguity, we indicate algorithm briefly as $A_{k,P}$.

(**Equals Algorithms**) Two algorithms

$$A^i_{k,P} = \{I^{i_0}, I^{i_1}, ...I^{i_k}\}, \quad A^j_{k,P} = \{I^{j_0}, I^{j_1}, ...I^{j_k}\}$$

are said to be equals if $\forall s \in [0,k], \quad I^{i_s} \equiv I^{j_s}$.

Note that two equals algorithms have the same cardinality.

The number and size of sub-problems into which a problem is decomposed determines the granularity of the decomposition. A decomposition into a large number of small computational problems is called fine-grained and a decomposition into a small number of large computational problems is called coarse-grained. Here, granularity has a major consequence in the level of detail that is required for an algorithm to be analysed with this approach. However, among the commonly used decomposition techniques, it is worth noting that recursive decomposition is the most suitable for our performance model especially for a real-world algorithm.

(**Granularity set of an Algorithm**) Given $A_{k,P}$, the subset $\mathcal{G}(A_{k,P})$ of $A_{k,P}$ made of distinct operators of $A_{k,P}$ defines the granularity set of $A_{k,P}$. Two algorithms

$$A^i_{k,P} = \{I^{i_0}, I^{i_1}, ...I^{i_k}\}, \quad A^j_{k,P} = \{I^{j_0}, I^{j_1}, ...I^{j_k}\}$$

have the same granularity if $\mathcal{G}(A^i_{k,P}) \equiv \mathcal{G}(A^j_{k,P})$.

Let $AL_{\mathcal{B}_{N_r}}$ (or simply AL) be the set of algorithms that solve $\mathcal{B}_{N_r}$, obtained by varying $\mathcal{M}_P$, P and $D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}$.
Let associate each algorithm of AL to an elementary decomposition, that is, let introduce the surjective correspondence

$$\varphi : A_{k,P} \in AL \longrightarrow D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}, \qquad (12)$$

which induces on AL an equivalence relationship ϱ of AL in itself, such that

$$\varrho(A_{k,P}) = \{\widetilde{A_{k,P}} \in AL : \varphi(\widetilde{A_{k,P}}) \quad = \quad \varphi(A_{k,P})\} \qquad (13)$$

Therefore, $\varrho(A_{k,P})$ is the set of algorithms of AL associated with the same decomposition $D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}$. Hence, ϱ induces the quotient set $\frac{AL}{\varrho}$, whose elements are disjoints and finite subsets of AL determined by ϱ, that is they are *equivalence classes* under ϱ.

In the following we consider A as a representative of its equivalence class in AL.

(**Complexity**) The number elements of $A_{k,P}$ is said complexity of $A_{k,P}$. It is denoted as $C(A_{k,P})$. That is

$$C(A_{k,P}) := card(A_{k,P}) \quad .$$

Remark: $C(A_{k,P})$ equals to the number of non empty elements of M_D, i.e. the decomposition matrix defined on $D_k(\mathcal{B}_{N_r})$. So, each algorithm belonging to the same equivalence class according to ϱ has the same complexity. An integer is then associated with each element $\varrho(A_{k,P})$ of quotient set $\frac{AL}{\varrho}$ which induces an ordering relation between the equivalence classes in $\frac{AL}{\varrho}$: therefore there is a minimum and a maximum complexity for algorithms that solve the problem $\mathcal{B}_{N_r}$. By virtue of the bijective correspondence γ in (11), it holds that

$$card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r})) = k, \qquad (14)$$

$\forall\, A_{k,P} \in \varrho(A_{k,P})$.

Remark (**Similar Algorithms**): Given $\mathcal{B}_{N_r} \simeq \mathcal{B}_{N_q}$

and their relative similar decompositions $D'_{k_i}(\mathcal{B}_{N_r})$ and $D''_{k_j}(\mathcal{B}_{N_q})$ $k_i, k_j \in \mathbb{N}$, algorithms belonging to $\varphi^{-1}(D'_{k_i}(\mathcal{B}_{N_r}))$ (see (12)) are similar to algorithms belonging to $\varphi^{-1}(D''_{k_j}(\mathcal{B}_{N_q}))$.

Remark: Since we can associate operator I^{i_k} of $A_{k,P}$ to each subproblem according to γ, operators of A inherit the dependencies that exist between subproblems of $\mathcal{B}_{N_r}$, but not independencies, because two operators may depend - for example - on the availability of computing units of $\mathcal{M}_P$ during their execution.

*Remark (***Execution matrix***)*: we introduce the group $(A_{k,P}, \diamond, \pi_{A_{k,P}})$ where $\pi_{A_{k,P}}$ is the strict partial order relation between any two elements of $A_{k,P}$ that guarantees that two elements cannot be performed in any arbitrary order and simultaneously, that is on two different processing elements of M_D. The condition that two elements cannot be performed in any arbitrary order induces the inheritance of dependencies between decomposition subproblems and algorithm operators, while the condition that that two elements cannot be performed simultaneously - relating to availability of resources - adds possible reasons for dependency between operators, which depend on the machine on which algorithm A is intended to run. We construct matrix M_D of order $r_E \times c_E$, where $c_E = P^2$ as dependency matrix (see Definition 3). The number of columns of this matrix will be the maximum number of sub-algorithms that can be performed simultaneously on $\mathcal{M}_P$. In the following, we denote this matrix as **execution matrix** and we refer to it by using the symbol $M_E = (e_{i,j})$ or simply M_E. This matrix can be placed in analogy with the execution graphs that are often used to describe the sequence of steps of algorithm A on a given machine for a particular input or a particular configuration.

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

Matrix M_E is unique unless an exchange between the rows is performed. This means that r_E is well determined.

In conclusion, by looking at the rows of M_E for determining those that have only one non empty element and those that have more that one non empty element, allows us to distinguish between sequential and parallel part of $A_{k,P}$.

Concurrency degree of $\mathcal{B}_{N_r}$ provides an upper limit to the maximum number of independent sub-algorithms executable simultaneously on the machine. Dependency degree provides a lower limit to number of rows of M_E.

Let $tcalc$ be the execution time for one floating point operation. In the following we assume that

$$\forall I^{i_j} \in Cop_{\mathcal{M}_P}, \quad t_{i_j} = \alpha_{i_j}^{calc} \cdot tcalc, \quad \alpha_{i_j}^{calc} \in \Re \tag{15}$$

(**Row execution time**) The quantity

$$T_r(A_{k,P}) := \max_{j \in [0, c_E - 1]} t_{r_j}$$

is said execution time of the row r of M_E (which is a sub-algorithm of $A_{k,P}$).

Let $\alpha_r^{calc} := \max_{j \in [0, c_E - 1]} \alpha_{r_j}^{calc}$ then

$$T_r(A_{k,P}) = \max_{j \in [0, c_E - 1]} \alpha_{r_j}^{calc} \cdot tcalc = \alpha_r^{calc} \cdot tcalc$$

(**Execution time**) The quantity

$$T(A_{k,P}) := \sum_{r=0}^{r_E - 1} T_r(A_{k,P}) \tag{16}$$

is said execution time of A.

If $\alpha_{sum}^{calc} := \sum_{r=0}^{r^E - 1} \alpha_r^{calc}$ then

$$T(A) = \alpha_{sum}^{calc} \cdot tcalc \quad .$$

Remark: Let

- $r_{seq} \leq r_E$ denote the number of rows of M_E with only one non-empty element (sequential sub-algorithms of $A_{k,P}$).
- $r_{par} = r_E - r_{seq}$, with $r_{par} \leq r_E$, denote the number of rows of M_E with more than one non empty element.

Within the sequence $i = 0, \ldots, r_E - 1$, numbering the r_E rows of M_E, two subsequences of indices come out: $\{i_q\}_{q \in [0, r_{seq}-1]}$, and $\{i_r\}_{r \in [0, r_{par}-1]}$

(**Execution time**) The quantity

$$T_{par}(A_{k,P}) := \sum_{r=0}^{r_{par}-1} T_{i_r}(A_{k,P}) \qquad (17)$$

is said execution time of $A_{k,P}$.

(**Sequential Execution time**) The quantity

$$T_{seq}(A_{k,P}) := \sum_{i_q=0}^{r_{seq}-1} T_{i_q}(A_{k,P}) \qquad (18)$$

is said sequential execution time of $A_{k,P}$.

Equation (16) can be written as

$$T(A_{k,P}) = T_{seq}(A_{k,P}) + T_{par}(A_{k,P}) \quad . \qquad (19)$$

Expression (19) states that, by looking at matrix M_E, the model allows to immediately let know how much large are the parallel and the sequential parts of execution time of $A_{k,P}$.

Given $A_{k,P}$ with $P > 1$ and M_E of order $N_P^E = r_{E_P} \times P$, let $Sp(A_{k,P})$ denote speedup of $A_{k,P}$, that is

$$Sp(A_{k,P}) := \frac{T(A_{k,1})}{T(A_{k,P})} \qquad (20)$$

Let $tcalc$ be the execution time for one floating point operation. In the following we assume that

$$\forall I^{i_j} \in Cop_{\mathcal{M}_P}, \quad t_{i_j} = \alpha_{i_j}^{calc} \cdot tcalc, \quad \alpha_{i_j}^{calc} \in \Re \qquad (21)$$

(**Row execution time**) The quantity

$$T_r(A_{k,P}) := \max_{j \in [0, c_E - 1]} t_{r_j}$$

is said execution time of the row r of M_E (which is a sub-algorithm of $A_{k,P}$).
Let $\alpha_r^{calc} := \max_{j \in [0, c_E - 1]} \alpha_{r_j}^{calc}$ then

$$T_r(A_{k,P}) = \max_{j \in [0, c_E - 1]} \alpha_{r_j}^{calc} \cdot tcalc = \alpha_r^{calc} \cdot tcalc$$

If $\alpha_{sum}^{calc} := \sum_{r=0}^{r^E - 1} \alpha_r^{calc}$ then

$$T(A) = \alpha_{sum}^{calc} \cdot tcalc \quad .$$

Remark: Let

- $r_{seq} \leq r_E$ denote the number of rows of M_E with only one non-empty element (sequential sub-algorithms of $A_{k,P}$).
- $r_{par} = r_E - r_{seq}$, with $r_{par} \leq r_E$, denote the number of rows of M_E with more than one non empty element.

Theorem 1 *Let*

$$R^{calc}(A_{k,P}) := \frac{\alpha_{sum}^{calc}}{r_E} \geq 1 \quad .$$

R^{calc} is the parameter of the algorithm A depending on the most computationally intensive sub algorithms of A. It holds

$$T(A_{k,P}) = R^{calc}(A_{k,P}) \cdot r_E \cdot tcalc \qquad (22)$$

Following result shows that the execution time of $A_{k,P}$ is bounded below by the problem dependence degree.

Theorem 2 *If $c_E = 1$ (i.e. $A_{k,P}$ is sequential), it is*

1. $C(A_{k,P}) = r_E$,
2. $T(A_{k,P}) = C(A_{k,P}) \cdot R^{calc}(A_{k,P}) \cdot tcalc$.

Next result formally states that if $\mathcal{B}_{N_r}$ is fully decomposed, the execution time of a fully parallel algorithm $A_{k,P} \in AL_{\mathcal{B}_{N_r}}$ on $\mathcal{M}_P$ is P times smaller than the execution time of $A_{k,1}$ on $\mathcal{M}_1$ where $\mathcal{M}_1$ and $\mathcal{M}_P$ differ only on the number of processing units.

Theorem 3 *$\forall \mathcal{B}_{N_r}$ fully decomposed and $\forall A_{k,P}$ perfectly parallel algorithm that solves it on $\mathcal{M}_P$ with $P > 1$, if*

$$Cop_{\mathcal{M}_1} \equiv Cop_{\mathcal{M}_p}, \quad Xop_{\mathcal{M}_1} \equiv Xop_{\mathcal{M}_p} \quad ,$$

it follows that:

$$T(A_{k,P}) = \frac{T(A_{k,1})}{P} \cdot R^{calc}(A_{k,P}) \cdot tcalc \quad . \quad (23)$$

Theorem 4 *For matrices M_E of algorithms in $\varrho(A_{k,P})$, it holds*

$$c_E \leq c_D \quad (24)$$

and

$$r_E \geq r_D. \quad (25)$$

Moreover, let us consider $A^i_{k,P}$ and $A^j_{k,P}$ two algorithms belonging to $\varrho(AA_{k,P})$, and their matrices M^i_E and M^j_E. We have:

- $c^i_E < c^j_E \Rightarrow r^i_E \geq r^j_E$;
- $c^i_E > c^j_E \Rightarrow r^i_E \leq r^j_E$.

Theorem 5 *It holds that*

$$T(A_{k,P}) \geq r_D \cdot R^{calc}(A_{k,P})\, tcalc \quad , \quad (26)$$

and it assumes its minimum value when $r_E = r_D$.

Remark: The minimum execution time is proportional to dependency degree of $\mathcal{B}_{N_r}$, that is when the number of computing units is equal to concurrency degree of $\mathcal{B}_{N_r}$.

Theorem 6 *It holds*

$$T(A_{k,P}) = (r_{seq} + r_{par}) \cdot R^{calc}(A_{k,P}) \cdot tcalc \quad . \quad (27)$$

Let Q denote the cost of $A_{k,P}$. The cost is defined as the product of the execution time and the number of processors utilized. It follows that

Theorem 7 *It holds*

$$Q(A_{k,P}) = c_E \cdot r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc \quad . \quad (28)$$

Theorem 8 *It holds*

$$Sp(A_{k,P}) = \frac{C(A_{k,1})}{r_{E_{A_{k,P}}} \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}} \quad . \quad (29)$$

Remark: If $A_{k,P}$ is perfectly parallel than $Sp(A_{k,P}) = c_E$.

PERFORMANCE ANALYSIS OF MGRIT ALGORITHM

Let $\Phi(M, \delta_l)$ be the subproblem of evaluating Φ and $\phi(M, \delta_l)$ the operator to solve it: all the most time consuming operators in the MGRIT algorithm involve the Φ evaluation. Execution time of $\phi(M, \delta_l)$ depends on the size of the spacial grid, i.e. the set $(x_j)_{j=1,M}$, and on the step size of discretization in time direction, i.e. δ_l. In this work we assume M be fixed. This means that we may assume that the execution time of ϕ is fixed. We completely overlook the accuracy of the results,

since it strictly depends on the characteristics of Φ. For the sake of brevity we face here only the V-Cycle, as described in Figure 4.

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. As described in Figure 4, we note that if L is the coarsest level, and the solver of the system on the coarsest-grid is sequential, it will involve at least one $\phi(M, \delta_L)$-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 $\phi(M, \delta_L)$.
Otherwise, for each level $l < L$,

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

Let us introduce the matrix $\mathcal{M}_D$, which is the so-called *dependency matrix* of the time-space problem to solve, according to its decomposition in the space subproblems $\Phi(M, \delta_i)$, for $i = 1, \ldots, L$, and $j = 1, \ldots, Np$ where $Np \in \{N_{F_l}, N_{l+1}, N_L\}$.

The *concurrency degree* of the problem decomposed in this way is the number of columns of $\mathcal{M}_D$, i.e. the maximum number of simultaneous Φ evaluations, that is

$$c_{\mathcal{D}} = max\{N_{F_0}, N_2\}$$

To get the *dependency degree* we need $r_{\mathcal{D}}$, i.e. the number of rows of $\mathcal{M}_D$. We have:

- 3 rows for each FCF-relaxation, that means $3 \cdot (L-1)$ rows,
- 1 row for each residual computation, that means $L-1$ rows (this are the longest rows in the matrix, that is with the biggest numbers of columns),
- $N_L - 1$ rows for the coarsest-grid solver,
- 1 row for each ideal interpolation (F-relaxation), that means $L-1$ rows.

So, $r_{\mathcal{D}} = 5 \cdot (L-1) + N_L - 1$.

Consider now a computing architecture with P processing elements, where $P = \frac{c_{\mathcal{D}}}{np}$ where $np \in \mathbb{N}$ and $P \leq N_L$. These conditions state that the points on the finest grid are equally distributed among the processors, that is $c_{\mathcal{D}}$ is multiple of P and that on the coarsest grid each processors holds at least one point.
We define the *execution matrix* of MGRIT made of the operators $\phi(M, \delta_i)_j^k$, where $i = 1, \ldots, L$, $j = 1, \ldots P$. $k = 1, \ldots, Npl$ and $Npl \in \{\frac{N_{F_l}}{P}, \frac{N_{l+1}}{P}, N_L\}$. The matrix is defined considering that, for each level, the number of points of the grid is a multiple of P. This is without loss of generality, as otherwise the number of rows is still the same, just with some empty elements.
Consider the algorithm, let us say A_{seq}, which solves (2) with the same discretization in time on the finest grid but without introducing MGR or any parallelism. A_{seq} is made of N_0

executions of $\phi(M, \delta_1)$, leading to the execution matrix $\mathcal{E}_1$ with one column and N_0 rows. We prove that

Proposition: Let execute MGRIT algorithm on a computing architecture with $P < N_L$ processing elements, where $P = \frac{N_0}{np}$ and $np \in \mathbf{N}$. Let t_ϕ be the execution time of $\phi(M, \delta_l)$, $\forall l \in [1, L]$. Then the speed-up $S(MGRIT, A_{seq})$ of MGRIT with respect to A_{seq} is bounded above as follows:

$$S(MGRIT, A_{seq}) \leq \frac{N_0}{5 \cdot (L-1) + N_L - 1} \tag{30}$$

Proof: The MGRIT execution matrix (37) has P columns and

$$r_{\mathcal{E}_P} = \sum_{l=1}^{L-1} \left(3 \cdot \frac{N_{F_l}}{P} + 2 \cdot \frac{N_{l+1}}{P} \right) + N_L - 1 \geq$$

$$\geq r_{\mathcal{D}} = 5 \cdot (L-1) + N_L - 1$$

rows, since it consists of

- $L-1$ FCF-relaxations with:
  1. $\frac{N_{F_l}}{P}$ rows for F-relaxation,
  2. $\frac{N_{l+1}}{P}$ rows for C-relaxation,
  3. $\frac{N_{F_l}}{P}$ rows for F-relaxation,
- $\frac{N_{l+1}}{P}$ rows for each $l < L$ residual computation,
- $N_L - 1$ rows for the solver on the coarsest-grid,
- $\frac{N_{F_l}}{P}$ row for each ideal interpolation (F-relaxation), i.e. $L-1$ rows.

It means that execution time of MGRIT is

$$T_P(MGRIT) = r_{\mathcal{E}_P} \cdot t_\phi \tag{31}$$

where the r.h.s. of (31) is

$$\sum_{l=1}^{L-1} \left(3 \cdot \frac{N_{F_l}}{P} + 2 \cdot \frac{N_{l+1}}{P} \right) + N_L - 1 \quad . \tag{32}$$

As $\mathcal{E}_1$ has N_0 rows, the execution time of the sequential-in-time algorithm is

$$T_1(A_{seq}) = N_0 \cdot t_\phi \tag{33}$$

Thus it holds that

$$S(MGRIT, A_{seq}) = \frac{r_{\mathcal{E}_1} \cdot t_\phi}{r_{\mathcal{E}_P} \cdot t_\phi} \tag{34}$$

where the r.h.s. of (34) is

$$\frac{N_0}{\sum_{l=1}^{L-1} \left(3 \cdot \frac{N_{F_l}}{P} + 2 \cdot \frac{N_{l+1}}{P} \right) + N_L - 1} \tag{35}$$

and the (30) follows.

Main outcome of (30) and (32) is to give a-priori estimate of the maximum number of multigrid levels and the number of processors to allocate in order to efficiently implement the iterative MGRIT algorithm. This analysis is carried out regardless the execution of ϕ. Indeed, this operation could be performed in parallel or sequentially in space and such choice only affects the unit time t_ϕ.

DISCUSSION AND FUTURE WORK

The aim of the proposed performance analysis is to address the algorithmic strong and weak scaling of MGRIT, regarded as a parallel iterative algorithm proceeding along the time dimension. The performance model in [9], instead, aims to select the best parallel configuration (i.e. how much parallelism to devote to space vs. time). We believe that both models could be employed for the successful implementation of the MGRIT algorithm instead of a time-marching integrator.

We are currently set up the PETSc objects

and the operators of the structure that is Interpolation, Restriction, Smoothers, and Coarse-solver for performing numerical experiments in order to validate the previous analysis. In the linear case, the scalable linear equations solvers (KSP) component provides an easy-to-use interface to the combination of a Krylov subspace iterative method and a preconditioner (in the KSP and PC components, respectively) or a sequential direct solver. Then we'll build and validate the MGRIT algorithm for the non-linear case, using the SNES components. The idea is to introduce, at the end, a new level of parallelism along the space dimensions, handled separately from the time one.

Finally, we will also consider the memory accesses matrix for taking into account the communications that significantly affect the software speed up and limit the real number of processors and the number of grid levels to use.

APPENDIX

$$\mathcal{M}_D = \begin{bmatrix} \Phi(M,\delta_0)_1 & \Phi(M,\delta_0)_2 & \cdots & \Phi(M,\delta_0)_{N_{F_0}} \\ \Phi(M,\delta_0)_1 & \Phi(M,\delta_0)_2 & \cdots & \Phi(M,\delta_0)_{N_1} \\ \Phi(M,\delta_0)_1 & \Phi(M,\delta_0)_2 & \cdots & \Phi(M,\delta_0)_{N_{F_0}} \\ \Phi(M,\delta_0)_1 & \Phi(M,\delta_0)_2 & \cdots & \Phi(M,\delta_0)_{N_1} \\ \Phi(M,\delta_1)_1 & \Phi(M,\delta_1)_2 & \cdots & \Phi(M,\delta_1)_{N_{F_1}} \\ \Phi(M,\delta_1)_1 & \Phi(M,\delta_1)_2 & \cdots & \Phi(M,\delta_1)_{N_2} \\ \Phi(M,\delta_1)_1 & \Phi(M,\delta_1)_2 & \cdots & \Phi(M,\delta_1)_{N_{F_1}} \\ \Phi(M,\delta_1)_1 & \Phi(M,\delta_1)_2 & \cdots & \Phi(M,\delta_1)_{N_2} \\ \Phi(M,\delta_2)_1 & \Phi(M,\delta_2)_2 & \cdots & \Phi(M,\delta_2)_{N_{F_2}} \\ \Phi(M,\delta_2)_1 & \Phi(M,\delta_2)_2 & \cdots & \Phi(M,\delta_2)_{N_3} \\ \Phi(M,\delta_2)_1 & \Phi(M,\delta_2)_2 & \cdots & \Phi(M,\delta_2)_{N_{F_2}} \\ \Phi(M,\delta_2)_1 & \Phi(M,\delta_2)_2 & \cdots & \Phi(M,\delta_2)_{N_3} \\ \vdots & \vdots & \cdots & \vdots \\ \Phi(M,\delta_L)_2 & & & \\ \vdots & & & \\ \Phi(M,\delta_L)_{N_L} & & & \\ \vdots & \vdots & \cdots & \vdots \\ \Phi(M,\delta_2)_1 & \Phi(M,\delta_2)_2 & \cdots & \Phi(M,\delta_2)_{N_{F_2}} \\ \Phi(M,\delta_1)_1 & \Phi(M,\delta_1)_2 & \cdots & \Phi(M,\delta_1)_{N_{F_l}} \\ \Phi(M,\delta_0)_1 & \Phi(M,\delta_0)_2 & \cdots & \Phi(M,\delta_0)_{N_{F_0}} \end{bmatrix} \tag{36}$$

$$\mathcal{E}_P = \begin{bmatrix} \phi(M,\delta_0)_1^1 & \phi(M,\delta_0)_2^1 & \cdots & \phi(M,\delta_0)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_0)_1^{\frac{N_{F_0}}{P}} & \phi(M,\delta_0)_2^{\frac{N_{F_0}}{P}} & \cdots & \phi(M,\delta_0)_P^{\frac{N_{F_0}}{P}} \\ \phi(M,\delta_0)_1^1 & \phi(M,\delta_0)_2^1 & \cdots & \phi(M,\delta_0)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_0)_1^{\frac{N_1}{P}} & \phi(M,\delta_0)_2^{\frac{N_1}{P}} & \cdots & \phi(M,\delta_0)_P^{\frac{N_1}{P}} \\ \phi(M,\delta_0)_1^1 & \phi(M,\delta_0)_2^1 & \cdots & \phi(M,\delta_0)_P^0 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_0)_1^{\frac{N_{F_0}}{P}} & \phi(M,\delta_0)_1^{\frac{N_{F_0}}{P}} & \cdots & \phi(M,\delta_0)_P^{\frac{N_{F_0}}{P}} \\ \phi(M,\delta_0)_1^1 & \phi(M,\delta_0)_2^1 & \cdots & \phi(M,\delta_0)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_0)_1^{np} & \phi(M,\delta_0)_2^{np} & \cdots & \phi(M,\delta_0)_P^{np} \\ \phi(M,\delta_1)_1^1 & \phi(M,\delta_1)_2^1 & \cdots & \phi(M,\delta_1)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_1)_1^{\frac{N_{F_1}}{P}} & \phi(M,\delta_1)_1^{\frac{N_{F_1}}{P}} & \cdots & \phi(M,\delta_1)_P^{\frac{N_{F_1}}{P}} \\ \phi(M,\delta_1)_1^1 & \phi(M,\delta_1)_2^1 & \cdots & \phi(M,\delta_1)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_1)_1^{\frac{N_2}{P}} & \phi(M,\delta_1)_1^{\frac{N_2}{P}} & \cdots & \phi(M,\delta_1)_P^{\frac{N_2}{P}} \\ \phi(M,\delta_1)_1^1 & \phi(M,\delta_1)_2^1 & \cdots & \phi(M,\delta_1)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_1)_1^{\frac{N_{F_1}}{P}} & \phi(M,\delta_1)_2^{\frac{N_{F_1}}{P}} & \cdots & \phi(M,\delta_1)_P^{\frac{N_{F_1}}{P}} \\ \phi(M,\delta_1)_1^1 & \phi(M,\delta_1)_1^1 & \cdots & \phi(M,\delta_1)_P^1 \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_1)_1^{\frac{N_1}{P}} & \phi(M,\delta_1)_2^{\frac{N_1}{P}} & \cdots & \phi(M,\delta_1)_P^{\frac{N_1}{P}} \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_L)_1^2 & & & \\ \vdots & & & \\ \phi(M,\delta_L)_1^{N_L} & & & \\ \vdots & \vdots & \cdots & \vdots \\ \phi(M,\delta_1)_1^1 & \phi(M,\delta_1)_2^1 & \cdots & \phi(M,\delta_1)_P^1 \\ \vdots & & & \\ \phi(M,\delta_1)_0^{\frac{N_{F_1}}{P}} & \phi(M,\delta_1)_1^{\frac{N_{F_1}}{P}} & \cdots & \phi(M,\delta_1)_P^{\frac{N_{F_1}}{P}} \\ \phi(M,\delta_0)_1^1 & \phi(M,\delta_0)_2^1 & \cdots & \phi(M,\delta_0)_P^1 \\ \vdots & & & \\ \phi(M,\delta_0)_1^{\frac{N_{F_0}}{P}} & \phi(M,\delta_0)_2^{\frac{N_{F_0}}{P}} & \cdots & \phi(M,\delta_0)_P^{\frac{N_{F_0}}{P}} \end{bmatrix} \tag{37}$$

Bibliography

[1] R. Arcucci, L. D'Amore, V. Mele *Mathematical Approach to the Performance Evaluation of Three Dimensional Variational Data Assimilation*, AIP Proceedings 2017.

[2] S.Balay, S. Abhyankar,M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, H. Zhang, *PETSc Web Pages* http://www.mcs.anl.gov/petsc.

[3] L. Carracciuolo, L. D'Amore, V. Mele. *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.

[4] L. Carracciuolo, L. D'Amore, A. Murli, *Towards a parallel component for imaging in PETSc programming environment: A case study in 3-D echocardiography* Parallel Computing, vol 32, n. 1, pp. 67-83, 2006.

[5] L. D'Amore, V. Mele, G. Laccetti, A. Murli. *Mathematical Approach to the Performance Evaluation of Matrix-matrix Multiply Algorithm on a Two Level Parallel Architecture*, LNCS 9574, Springer, pp. 25-34, 2016.

[6] L. D'Amore, A. Murli, V. Boccia, L. Carracciuolo, *Insertion of PETSc in the NEMO stack software driving NEMO towards exascale computing*, International Conference on High Performance Computing and Simulation (HPCS), 2014, pp.724,731, 21-25 July 2014.

[7] R. D. Falgout, S. Friedhoff, Tz. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel Time Integration with Multigrid*, SIAM Journal on Scientific Computing, vol. 36, n. 6, pp. C635-C661, 2014.

[8] R. D. Falgout,S. Friedhoff, Tz. V. Kolev, S. P. MacLachlan, and J. B. Schroder, S. Vandewalle, *Multigrid methods with space-time concurrency*, SIAM Journal on Scientific Computing, LLNL-JRNL-678572, submitted.

[9] H. Gahvari, V. A. Dobrev, R. D. Falgout, T. V. Kolev, J. B. Schroder, M. Schulz, U. Meier Yang (2016). *A performance model for allocating the parallelism in a multigrid-in-time solver.* In Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems). IEEE Press, Piscataway, NJ, USA, 22-31.

[10] M. J. Gander, S. Vandewalle,*Analysis of the parareal time-parallel time-integration method*, SIAM J. Sci. Comput., 29 (2007), pp. 556-578.

[11] J.-L. Lions, Y. Maday, G. Turinici, *A parareal in time discretization of PDEs*, Comptes Rendus de l'Academie des Sciences - Series I - Mathematics, vol. 332, pp. 661-668, 2001. Available at: http://dx.doi.org/10.1016/S0764-4442(00)01793-6

[12] B. Smith, *The portable extensible toolkit for scientific computing*, 2013, Lectures at the The Argonne Training Program on Extreme-Scale Computing, August 2013.

[13] *XBraid: Parallel multigrid in time*. http://llnl.gov/casc/xbraid.