DOI: 10.15439/2014F340
ACSIS, Vol. 2

Implementation of a distributed parallel in time scheme using PETSc for a Parabolic Optimal Control Problem

Juan José Cáceres Silva*, Benjamín Barán*†‡ and Christian E. Schaerer†
*Science and Technology School, Catholic University, Asunción, Paraguay
†Polythechnic School, National University of Asunción, San Lorenzo, Paraguay, P.O.Box: 2111 SL.
‡Polythechnic School, East National University, Ciudad del Este, Paraguay

***Abstract*—This work presents a parallel implementation of the Parareal method using Portable Extensible Toolkit for Scientific Computation (PETSc). An optimal control problem of a parabolic partial differential equation with known boundary conditions and initial state is solved, where the minimized cost function relates the controller v usage and the approximation of the solution y to an optimal known function y^*, measured by $\|y\|$ and $\|y^*\|$, respectively. The equations that model the process are discretized in space using Finite Elements and in time using Finite Differences. After the discretizations, the problem is transformed to a large linear system of algebraic equations, that is solved by the Conjugate Gradient method. A Parareal preconditioner is implemented to speed up the convergence of the Conjugate Gradient.**

The main advantage in using the Parareal approach is to speed up the resolution time, when comparing to implementations that use only the Conjugate Gradient or GMRES methods. The implementation developed in this work offers a parallelization relative efficiency for the strong scaling of approximately 70% each time the process count doubles. For weak scaling, 75% each time the process count doubles for a constant solution size per process and 96% each time the process count doubles for a constant data size per process.

I. Introduction

MANY challenges of engineering design, such as heat dissipation, electromagnetic inversion, diffraction tomography among others, can be modeled as a parabolic optimal control problem [1]. The problem to be solved is [2, 3]:

$$\begin{cases} \textit{minimize} & J(y,v) \\ \quad s.t. & \nabla_t y = \Delta_x y + v\,, \end{cases} \tag{1}$$

with [3]

$$\begin{aligned} J(y,v) =& \frac{\alpha}{2}\int_\Omega\int_{t_0}^{t_f} \|y-y^*\|_2^2 dtdx + \frac{\beta}{2}\int_\Omega \|y(t_f)-y^*(t_f)\|_2^2 dx \\ &+ \frac{\gamma}{2}\int_\Omega\int_{t_0}^{t_f}\|v\|_2^2 dtdx\,, \end{aligned} \tag{2}$$

where y^* is the optimal condition for the function y, α is the weight for the general approximation of the function y, β is the weight for the approximation at the final instant of the function y and γ gives the cost of the controller usage.

The finite elements discretization using the Galerkin method yields the following state equation [4, 5]:

$$M\dot{\underline{z}} = K\underline{z} + B\underline{u} \tag{3}$$

where $\underline{z} \in \mathbb{R}^{\hat{q}}$ is the nodal representation of y, $\underline{u} \in \mathbb{R}^{\hat{p}}$ is the nodal representation of v, M is the mass matrix, K is the stiffness matrix and B is the coupling matrix. Using this discretization, the cost function (2) becomes:

$$\begin{aligned} J_h(\underline{z},\underline{u}) &= \frac{\alpha}{2}\int_{t_0}^{t_f}(\underline{z}-\underline{z}^*)^T M(\underline{z}-\underline{z}^*)\,dt \\ &+ \frac{\beta}{2}\left\{[\underline{z}(t_f)-\underline{z}^*(t_f)]^T M[\underline{z}(t_f)-\underline{z}^*(t_f)]\right\} + \frac{\gamma}{2}\int_{t_0}^{t_f}\underline{u}^T R\underline{u}\,dt \end{aligned} \tag{4}$$

where $\underline{z}^*$ is the nodal representation of y^* and R is the controller's mass matrix.

The finite differences discretization, using a time interval τ with $\hat{l}$ time instants, is based on equation [4]:

$$\begin{aligned} &F_1\underline{z}(i+1) = F_0\underline{z}(i) + \tau B\underline{u}(i+1);\ \text{for } 0 < i < \hat{l} \\ &\text{and } \underline{z}(0) = y_0 \end{aligned} \tag{5}$$

where $F_0, F_1 \in \mathbb{R}^{\hat{q}\times\hat{q}}$ are matrices defined by $F_0 = M$ and $F_1 = M + \tau K$. The arrangement of equation (3) for all times yield:

$$\mathbf{Ez} + \mathbf{Nu} = \mathbf{f}_3 \tag{6}$$

where $\mathbf{z} \in R^{\hat{l}\hat{q}}$ and $\mathbf{u} \in R^{\hat{l}\hat{p}}$. With a similar argument, equation (4) has the following form:

$$J_h^\tau(\mathbf{z},\mathbf{u}) = \frac{1}{2}(\mathbf{z}-\mathbf{z}^*)^T\mathbf{Q}(\mathbf{z}-\mathbf{z}^*) + \frac{1}{2}\mathbf{u}^T\mathbf{Gu} + (\mathbf{z}-\mathbf{z}^*)^T\mathbf{g}. \tag{7}$$

Using Lagrange multipliers [6] for minimizing equation (7) subject to equality constraint (6) and imposing first order optimality conditions [7, 8, 9], the following KKT system [3] with saddle point form [10] is obtained [2, 4, 11]:

$$\begin{bmatrix} \mathbf{Q} & 0 & \mathbf{E}^T \\ 0 & \mathbf{G} & \mathbf{N}^T \\ \mathbf{E} & \mathbf{N} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \mathbf{u} \\ \mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{0} \\ \mathbf{f}_3 \end{bmatrix} \tag{8}$$

A. Schur's Equation

In order to simplify the linear system (8), the variables $\mathbf{z}$ and $\mathbf{q}$ are solved in terms of the control variable $\mathbf{u}$ [12, 4, 10, 13]. Making $\mathbf{z} = \mathbf{E}^{-1}\mathbf{f}_3 - \mathbf{E}^{-1}\mathbf{N}\mathbf{u}$, $\mathbf{q} = \mathbf{E}^{-T}\mathbf{f}_1 - \mathbf{E}^{-T}\mathbf{Q}\mathbf{z}$, and substituting on the final equation from (8), gives [13, 14, 15]:

$$\mathbf{H}\mathbf{u} = \mathbf{f} \tag{9}$$

where $\mathbf{H} = \mathbf{G} + \mathbf{N}^T\mathbf{E}^{-T}\mathbf{Q}\mathbf{E}^{-1}\mathbf{N}$ and $\mathbf{f} = \mathbf{N}^T\mathbf{E}^{-T}(\mathbf{Q}\mathbf{E}^{-1}\mathbf{f}_3 - \mathbf{f}_1)$. [1]

Doing this, the reduced Schur Complement [16, 14] Doing this, the equation system (8) is reduced. This expression is known as the Schur complement for equation (8) [16].

From this point on, the problem to solve is (9), a linear equation system, lets say $Ax = b$ for a general form, where the matrix A (in this case matrix $\mathbf{H}$ from equation (9)) is symmetric positive definite [10, 7, 4].

II. Mathematical Solution Formulation

A. Conjugate Gradient

The Conjugate Gradient method is used to solve a generic equation $Ax = b$ where $A \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ is symmetric positive definite and $b \in \mathbb{R}^{\tilde{n}}$.

On this work, the iterative algorithm defined in [17] is applied to the input matrix A and vector b, with error tolerance ε, initial guess x_0 and iteration limit for convergence max_i, as follows:

Algorithm 1 Conjugate Gradient

```
Input: $A, b, \varepsilon, x_0, max_i$
Output: $x$
 1: $r_0 \leftarrow b - Ax_0$
 2: $p_0 \leftarrow r_0$
 3: $i \leftarrow 0$
 4: while $r_{i+1} \geq \varepsilon \wedge i < max_i$ do
 5:     $\alpha_i \leftarrow \frac{r_i^T r_i}{p_i^T A p_i}$    ▷ In our implementation, $Ap_i$ is calculated by Algorithm 2
 6:     $x_{i+1} \leftarrow x_i + \alpha_i p_i$
 7:     $r_{i+1} \leftarrow r_i - \alpha_i A p_i$    ▷ In our implementation, $Ap_i$ is calculated by Algorithm 2
 8:     $\beta_i \leftarrow \frac{r_{i+1}^T r_i}{r_i^T r_i}$
 9:     $p_{i+1} \leftarrow r_{i+1} + \beta_i p_i$
10:     $i \leftarrow i + 1$
11: end while
12: if $r_{i+1} < \epsilon$ then
13:     return $x_i$    ▷ Convergent
14: else
15:     return $n.c.$    ▷ Not convergent
16: end if
```

[1]Recall that $\mathbf{u}, \mathbf{b} \in \mathbb{R}^{\hat{l}\hat{p}}$, $\mathbf{H}, \mathbf{G} \in \mathbb{R}^{\hat{l}\hat{p}\times\hat{l}\hat{p}}$, $\mathbf{N} \in \mathbb{R}^{\hat{l}\hat{q}\times\hat{l}\hat{p}}$ and $\mathbf{E}, \mathbf{Q} \in \mathbb{R}^{\hat{l}\hat{q}\times\hat{l}\hat{q}}$.

B. Using the Conjugate Gradient

In order to use Algorithm 1, the input matrix $\mathbf{H}$ must be previously computed, which requires a great computational work [5]. To avoid building matrix $\mathbf{H}$, steps 5) and 7) from Algorithm 1 are performed using Algorithm 2.

Let $\mathbf{s}$ be a generic input vector, and matrices $\mathbf{G}, \mathbf{N}, \mathbf{E}$ and $\mathbf{Q}$ as defined in Section I. The value of the product $\mathbf{H}\mathbf{s}$ is found using only matrix-vector operations, to avoid matrix-matrix operations that require more computational resources [18]. Algorithm 2 describes these matrix-vector operations [7].

Algorithm 2 Matrix-Vector Product $\mathbf{H}\mathbf{s}$

```
Input: $\mathbf{G}, \mathbf{N}, \mathbf{E}, \mathbf{Q}, \mathbf{s}$
Output: $\mathbf{x}$    ▷ $\mathbf{x} = \mathbf{H}\mathbf{s} = \mathbf{G}\mathbf{s} + \mathbf{N}^T\mathbf{E}^{-T}\mathbf{Q}\mathbf{E}^{-1}\mathbf{N}\mathbf{s}$
 1: $\mathbf{s}_1 \leftarrow \mathbf{G}\mathbf{s}$
 2: $\mathbf{s}_2 \leftarrow \mathbf{N}\mathbf{s}$
 3: $\mathbf{s}_3 \leftarrow \mathbf{E}^{-1}\mathbf{s}_2$ ▷ $\mathbf{E}\mathbf{s}_3 = \mathbf{N}\mathbf{s}$, in our implementation, solved by Algorithm 4
 4: $\mathbf{s}_4 \leftarrow \mathbf{Q}\mathbf{s}_3$    ▷ $\mathbf{s}_4 = \mathbf{Q}\mathbf{E}^{-1}\mathbf{s}_2$
 5: $\mathbf{s}_5 \leftarrow \mathbf{E}^{-T}\mathbf{s}_4$    ▷ $\mathbf{E}^T\mathbf{s}_5 = \mathbf{Q}\mathbf{s}_3$, in our implementation, solved by Algorithm 4
 6: $\mathbf{x} \leftarrow \mathbf{s}_1 + \mathbf{N}^T\mathbf{s}_5$    ▷ $\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{N}^T\mathbf{s}_5$
```

The direct implementation of Algorithm 2 can be unviable since steps 3) and 5) require inverse matrices [7]. To avoid this, the steps 3) and 5) from the Algorithm 2 can be solved using an inner Conjugate Gradient. This step will have a high computational cost because it will be done for each iteration of the outer Conjugate Gradient.

The idea is to replace steps 3) and 5) from Algorithm 2 using the Parareal method [7].

C. Parareal

The Parareal method [19, 9] is an iterative method used to solve a time dependant equation, based on a time domain decomposition $[t_0, t_f]$ in $\hat{k}$ *coarse* time intervals, each of size $\Delta T = (t_f - t_0)/\hat{k}$, with $T_0 = t_0$ and $T_k = t_0 + k\Delta T$ for $1 \leq k \leq \hat{k}$. This sets the solution for each instant T_k with $1 \leq k \leq \hat{k}$ using the *multiple-shooting* technique [20, 21] that requires the parallel resolution of the equation $\mathbf{z} = \mathbf{E}^{-1}\mathbf{b}$ for each (T_{k-1}, T_k) subinterval. To accelerate each multiple-shooting iteration, the residual equations are preconditioned by a coarse time grid discretization, with a time interval ΔT [7].

An approximation $\mathbf{E}_n^{-1}$ for $\mathbf{E}^{-1}$, is based on n Richardson's iterations [22], through the Parareal algorithm, where the Richardson's algorithm is used as an external iteration for a Schur's complement problem [7, 16, 23]. The matrix $\mathbf{E}_n$ is used to approximate the solution $\mathbf{z}$ by $\mathbf{z}_n = \mathbf{E}_n^{-1}\mathbf{b}$, and the main interest is that $\mathbf{z}_n = \mathbf{E}_n^{-1}\mathbf{b}$ and $\mathbf{z}_n \to \mathbf{z}$ as $n \to \infty$, in practical situations n is bounded [4].

Let $\hat{m} = (T_k - T_{k-1})/\tau$, $j_{k-1} = (T_{k-1} - T_0)/\tau$ and Z_k be the solution for the instant T_k, defined by solving from time T_{k-1} to T_k using the Finite Difference discretization scheme on the fine grid [24] (for each time instant, of size τ) with initial values Z_{k-1} in T_{k-1} and right hand side vector

$\mathbf{b} = [\underline{b}(j_{k-1}+1)^T, \ldots, \underline{b}(j_{k-1}+\hat{m})^T]^T$. The solution of each coarse interval is given by:

$$F_1 \otimes Z_k = F_0^{\Delta} \otimes Z_{k-1} + S_k \tag{10}$$

where, $\otimes$ represents the Kronecker product [25], $F_0^{\Delta} = (F_0 F_1^{-1})^{\hat{m}-1} F_0 \in \mathbb{R}^{\hat{q}\times\hat{q}}$, $Z_0 = 0$, the matrices F_0 y F_1 as set in (5) and

$$S_k = \sum_{m=1}^{\hat{m}} (F_1^{-1} F_0)^{\hat{m}-m} [F_0 Z_{k-1} - \underline{b}(j_{k-1}+m)] \tag{11}$$

Imposing continuity, $F_1 \otimes Z_k - F_0^{\Delta} \otimes Z_{k-1} - S_k = 0$ on the instants T_k, for $1 \le k \le \hat{k}$, the system $\mathbf{CZ} = \mathbf{S}$ is obtained [9, 7]:

$$\underbrace{\begin{bmatrix} F_1 & & & \\ -F_0^{\Delta} & F_1 & & \\ & \ddots & \ddots & \\ & & -F_0^{\Delta} & F_1 \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_{\hat{k}} \end{bmatrix}}_{\mathbf{Z}} = \underbrace{\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{\hat{k}} \end{bmatrix}}_{\mathbf{S}} \tag{12}$$

The case where the coarse solution in T_k with initial data $Z_{k-1} \in \mathbb{R}^{\hat{q}}$ in T_{k-1} is obtained after using a Finite Differences step for the coarse time interval $G_1 Z_k = G_0 Z_{k-1}$ is considered, where the matrices $G_1 = (M + K\Delta T)$ and $G_0 = M \in \mathbb{R}^{\hat{q}\times\hat{q}}$ are defined.

A coarse grid propagator based on G_0 and G_1 is used in the Parareal algorithm to precondition (12) [9]. The coarse grid propagation system $\mathbf{Z}^{i+1} = \mathbf{Z}^i + \mathbf{E}_g^{-1}\mathbf{R}^i$ is defined as:

$$\underbrace{\begin{bmatrix} Z_1^{i+1} \\ Z_2^{i+1} \\ \vdots \\ Z_{\hat{k}}^{i+1} \end{bmatrix}}_{\mathbf{Z}^{i+1}} = \underbrace{\begin{bmatrix} Z_1^{i} \\ Z_2^{i} \\ \vdots \\ Z_{\hat{k}}^{i} \end{bmatrix}}_{\mathbf{Z}^{i}} + \underbrace{\begin{bmatrix} G_1 & & & \\ -G_0 & G_1 & & \\ & \ddots & \ddots & \\ & & -G_0 & G_1 \end{bmatrix}^{-1}}_{\mathbf{E}_g^{-1}} \underbrace{\begin{bmatrix} R_1^{i} \\ R_2^{i} \\ \vdots \\ R_{\hat{k}}^{i} \end{bmatrix}}_{\mathbf{R}^{i}} \tag{13}$$

for $0 \le i \le (n-1)$, where the residue $\mathbf{R}^i = \left[{R_1^i}^T, \ldots, {R_{\hat{k}}^i}^T\right]^T \in \mathbb{R}^{\hat{k}\hat{q}}$ in (13) is defined as $\mathbf{R}^i = \mathbf{S} - \mathbf{CZ}^i$, where $\mathbf{Z}^i = \left[{Z_1^i}^T, \ldots, {Z_{\hat{k}}^i}^T\right]^T \in \mathbb{R}^{\hat{k}\hat{q}}$ and $\mathbf{Z}^0 = [0^T, \ldots, 0^T]^T$. Each R_j^i vector stands for the i-th iteration of the residue, on the T_j time instant. Likewise, each Z_j^i vector stands for the i-th iteration of the solution, on the T_j time instant.

Now, $\mathbf{z}_n = \mathbf{E}_n^{-1}\mathbf{b}$ is defined. Let $\mathbf{z}_n$ be the nodal representation of a piecewise linear function $\underline{z}^n$ in the time dimension with respect to the fine space discretization parameterized by τ in $[t_0, t_f]$. Because $\mathbf{z}_n \in \mathbb{R}^{(\hat{l}+\hat{k}-1)\hat{q}}$ is continuous in each coarse subinterval $[T_{k-1}, T_k]$, the function $\underline{z}^n$ can be discontinuous on the points T_k, with $1 \le k \le \hat{k}-1$. On each $[T_{k-1}, T_k]$ subinterval, $\underline{z}^n$ is defined by solving from the instant T_{k-1} to the instant T_k using the Finite Differences scheme with fine time intervals τ and initial data Z_{k-1}^n in T_{k-1}. The equation that describes the solution on the fine intervals, starting from a coarse interval is:

$$F_1 \underline{z}^n(i+1) = F_0 \underline{z}^n(i) - \underline{b}(i+1); \text{ for } T_{k-1} \le t < T_k; \text{ y } \underline{z}^n(T_{k-1}) = Z_{k-1}^n. \tag{14}$$

The vector $\mathbf{z}_n$ is obtained computing (14) for $2 \le k \le \hat{k}$. With this algorithm the steps 3) and 5) from Algorithm 2 can be solved, and therefore it can find the product $\mathbf{Hs}$. In the program, the input vector $\mathbf{s}$ for Algorithm 2 will be each vector p_k from Algorithm 1, used on the outer iteration of the Conjugate Gradient.

III. Implementation

The user defines the spatial discretization size $\hat{q}$, the fine time discretization size $\hat{l}$, the coarse time discretization size $\hat{k}$, the initial condition y_0 and the target solution y^*.

To help the convergence rate of the outer Conjugate Gradient, an initial guess $\mathbf{u}_0$ is found through:

$$\mathbf{u}_0 = \mathbf{N}\mathbf{E}^{-T}\mathbf{Q}\mathbf{E}^{-1}\mathbf{f}_3 - \mathbf{N}^T\mathbf{E}^{-T}\mathbf{f}_1 \,. \tag{15}$$

The program implemented on this work uses the `main` structure of Algorithm 3.

Algorithm 3 Main

```
Input: $\hat{q}, \hat{l}, \hat{k}, \hat{m}, y_0, y^*$
Output: $\mathbf{u}$
1: $[M, K, B] = finiteElements(\hat{q}, y_0, y^*)$   ▷ Call to the function that does the space discretization, as described in Section I
2: $[\mathbf{E}, \mathbf{Q}, \mathbf{N}, \mathbf{b}] = fineGrid(M, K, B, \hat{l})$   ▷ Call to the function that does the fine time discretization, as described in Section I
3: $[\mathbf{C}, \mathbf{E}_g] = coarseGrid(M, K, \hat{k}, \hat{m})$   ▷ Call to the function that does the coarse time discretization, as described in Section II-C
4: $[\mathbf{u}_0] = preconditioner(\mathbf{G}, \mathbf{E}, \mathbf{Q}, \mathbf{N}, \mathbf{b})$   ▷ Call to a function that calculates (15)
5: $[\mathbf{u}] = cg(\varepsilon, \mathbf{u}_0, max_i, \hat{k}, \hat{m}, \mathbf{G}, \mathbf{E}, \mathbf{Q}, \mathbf{N}, \mathbf{b}, \mathbf{C}, \mathbf{E}_g)$ ▷ Call to Algorithm 1
```

With the defined problem data, the finite elements matrices are generated. Next, the matrices of the time discretization are built, and the system (8) can be formulated. The matrices from the time discretization are considered as the fine grid matrices, because they have every time instant from the problem. Afterward, the coarse grid matrices are generated from the finite elements matrices. The coarse grid matrices are needed for the application of the Parareal method, as shown in equations (12) and (13).

With all the matrices created, the Conjugate Gradient method is executed to resolve $\mathbf{Hu} = \mathbf{b}$. On each Conjugate Gradient's iteration i, the product $\mathbf{H}p_i$ must be computed as described in Algorithm 1. To perform the product of matrix $\mathbf{H}$ by a vector, the Algorithm 2 is called. The steps 3) and 5) form Algorithm 2 are solved using the Parareal method. When the product $\mathbf{H}p_i$ is computed, the outer Conjugated Gradient's execution resumes.

For instance, to appreciate the benefits of the Parareal method, consider a space discretization grid with $\hat{q} = 2 \times 2 = 4$ elements, a fine time discretization with $\hat{l} = 10000$ time instants and a coarse time discretization with $\hat{k} = 10$ time instants. As a consequence, each process gets $\hat{m} = 1000$ time instants of the fine grid. For this example, the solution of the system $\mathbf{Ez} = \mathbf{b}$ involves $4 \cdot 10 \cdot 1000 \times 4 \cdot 10 \cdot 1000$ equations and variables, while the approximation $\mathbf{E}_n \mathbf{z}_n = \mathbf{b}$ is a linear system of only $4 \cdot 10 \times 4 \cdot 10$ equations and variables.

IV. ALGORITHMS

The pseudocode of the function used for the Parareal method (Algorithm 4) and its dependences are presented next, given that its implementation is the main contribution of this work. Besides, Algorithm 4 shows how the message passing is managed to maintain a low communication cost among the parallel processes.

The first pseudocode presented corresponds to the Parareal method. The same naming conventions as in Section II-C are used. The input parameters for the `Parareal` function are: the vector $b = \mathbf{b}$, the matrix $E_g = \mathbf{E}_g$, the matrix $C = \mathbf{C}$, the vector of the initial approximated solution Z_0, the coarse intervals count $\hat{k}$, the fine intervals by coarse interval count $\hat{m}$ and the error tolerance ε.

The output of the `Parareal` function is an approximation to $z = \mathbf{z} \leftarrow \mathbf{E}^{-1}\mathbf{b}$ as described on Section II-C. Algorithm 4 calls the functions `fineSolver` (Algorithm 5) and `marching` (Algorithm 6) to be next described in this section.

Algorithm 4 Parareal

```
Input: $b, E_g, C, \hat{k}, \hat{m}, Z_0, \varepsilon$
Output: $y$
 1: $S \leftarrow fineSolver(b, \hat{k}, \hat{m})$        ▷ Call to Algorithm 5
 2: $Z \leftarrow Z_0$
 3: $R \leftarrow S$   ▷ $\mathbf{R}^1 \leftarrow \mathbf{S} - \mathbf{CZ}^0$, communication of $S^k$ to the next process
 4: while $\|r_i\| > \varepsilon$ do
 5:     $coarse \leftarrow E_g^{-1}R$        ▷ $aux \leftarrow \mathbf{E}_g^{-1}\mathbf{R}^i$
 6:     $Z \leftarrow Z + coarse$        ▷ $\mathbf{Z}^{i+1} \leftarrow \mathbf{Z}^i + aux$
 7:     $R \leftarrow S - C \times Z$        ▷ $\mathbf{R}^{i+1} \leftarrow \mathbf{S} - \mathbf{CZ}^i$
 8: end while
 9: $y \leftarrow marching(b, x_{i-1}, \hat{k}, \hat{m})$        ▷ Call to Algorithm 6
10: return $y$
```

Algorithm 5 shows how the jumps vector $\mathbf{S}$ is generated according to equation (11), that saves only the final elements of the coarse time interval.

Algorithm 5 fineSolver

```
Input: $b, \hat{k}, \hat{m}$
Output: $S$
1: for all $k < \hat{k}$ do        ▷ parallel loop, distributed in $\hat{k}$ processes
2:     $s \leftarrow \vec{0}$
3:     for all $i < \hat{m}$ do        ▷ local loop, calculated on each process
4:         $s \leftarrow F_1^{-1}(F_0 s - b(k,i))$        ▷ Equation (10)
5:     end for
6:     $S(k) = s$
7: end for
8: return $S$
```

With this information the iterative loop of the Parareal algorithm is performed, the loop computes vector $\mathbf{Z}^i$ iteratively, as indicated in equation (13), until the solution of the coarse grid $\mathbf{Z}^n$ is found, when the required tolerance is reached.

After finding the coarse solution, the function `marching` is called, so that each process can extend its initial coarse solution to their own fine time intervals $\underline{z}^n$. Joining the solution of every process, the approximated general solution $\mathbf{z}_n$ is found.

Algorithm 6 marching

```
Input: $b, coarse, \hat{k}, \hat{m}$
Output: $y$
1: for all $k < \hat{k}$ do        ▷ parallel loop, distributed in $\hat{k}$ processes
2:     $z \leftarrow coarse(k)$
3:     for all $i < \hat{m}$ do        ▷ local loop, calculated on each process
4:         $z \leftarrow F_1^{-1}(F_0 z - b(i,k))$        ▷ Equation (14)
5:         $y(k,i) = z_i$
6:     end for
7: end for
8: return $y$
```

The functions `fineSolver` and `marching` are similar, because both solve the problem on the coarse time intervals. The `fineSolver` function finds its fine grid solution to calculate the final coarse instants (used as a preconditioner for the Parareal). The function `marching` finds the fine grid solution given an initial condition $\mathbf{Z}^n$ (the coarse grid solution), to complete the global solution.

As it was mentioned previously, some special attention is needed when a process requires some data that belongs to another process. The algorithms were designed to reduce the data communication between processes. With the proposed solution, the data communication between processes is needed only on the `Parareal` function, when the coarse data grid is propagated according to equation (13). Next, the experimental results of the implementation are presented.

V. Numerical experiments

The results of the experiments using the implementation of the Parareal method are presented in this Section. The experiments are based on the reference paper [7], used for validation.

The hardware used is a cluster of four Dell PowerEdge R710 nodes, with 2 processors of 4 cores Intel Xeon E5530 of 2.4GHz, Intel 5530 chipset, 8GB DDR3 of 1066 MHz RAM memory, connected in a Giga-Ethernet (1Gbps) LAN, as shown in Figure 1.

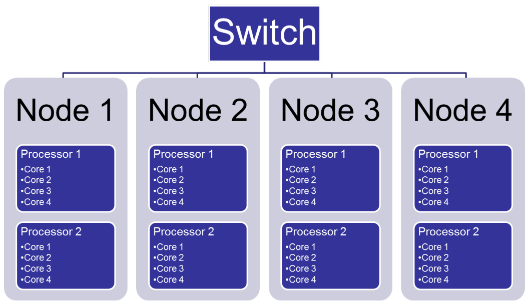

Fig. 1. Experimental platform

In this set of experiments, there are at most 4 nodes used and, for simplicity, only the first processor of each node is used. Therefore, only 4 cores per node are used.

A. Definitions

The optimal control problem to be solved for the experiments is given by the following 2D heat equation:

$$\begin{cases} z_t - z_{xx} = v, & x \in \Omega, \quad 0 < t \\ z(t,0) = 0, & x \in \partial\Omega, \quad 0 \leq t \\ z(0,x) = 0, & x \in \partial\Omega\,, \end{cases} \tag{16}$$

where $\Omega = [0,1] \times [0,1]$. The selected target function is $y^*(x_1,x_2) = x_1(1-x_1)e^{-x_1}x_2(1-x_2)e^{-x_2}$ for $t \in [0,1]$. The selected problem, the problem sizes and considerations are used as in the reference paper [7].

As naming conventions for the experiments, pCG(n,p) represents the execution of the modified Conjugate Gradient, using the Parareal method, for n nodes, each with p parallel processes. In all the experiments, a single process is run in each core. Times values are shown in seconds for all tables.

For each comparative table all tests are remade, so there may be some time differences in different tables that run the same setup of pCG(n,p). Those differences are unavoidable [26], but the variations are small in general, so they are acceptable anyway.

It should be mentioned that in the conducted experiments, the peak FLOPS / average FLOPS ratio [27] was not larger than 1.06 in any experiment.

B. Validation

The implementation of the pCG(n,p) of this work is compared to the reference work (IFOM) [7]. As [7] is a theoretical work about Parareal, there is no execution time; on the contrary, there are only data about the required iterations needed for the resolution of the problems. The values of Table I are given in the format $iter_e(iter_i)$, where $iter_e$ is the external iteration count (Conjugate Gradient) and $iter_i$ is the inner iteration count (Parareal).

TABLE I
Comparision of iterations of [7]'s implementation and our implementation pCG(n,p), for inner tolerance values ε_i, outer tolerance $\varepsilon_o = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, inner grid size 13×13, $\tau = 1/512$, $\hat{k} = 32$, $\Delta T/\tau = 16$ and $n.c.$ means that the system does not converge in 100 iterations.

ε_i	IFOM	pCG$(1,p)$		
		$p=1$	$p=2$	$p=4$
10^{-12}	16(586)	16(586)	16(580)	16(578)
10^{-10}	17(510)	17(510)	17(502)	17(504)
10^{-8}	18(442)	18(442)	18(414)	18(424)
10^{-7}	18(362)	18(362)	18(364)	20(404)
10^{-6}	21(338)	21(338)	21(342)	19(340)
10^{-5}	24(274)	24(274)	24(280)	n.c.
10^{-4}	28(220)	28(220)	63(312)	43(258)

Table I shows that the iteration count obtained through the implementation of pCG$(1,1)$ is consistent with the theoretical results expected from [7]. Some other tests were also made, which compare the solutions u_{IFOM} of IFOM and u_{pCG} of pCG(n,p), and the error $\epsilon = \|u_{IFOM} - u_{pCG}\|/\|u_{IFOM}\|$ was smaller than 10^{-6} on all the cases. Furthermore, checking the execution of the Parareal solver on pCG$(1,1)$, the error of each iteration's result was less than 10^{-6}.

C. Efficiency

The following concepts are used for the efficiency analysis of the implementation. *Strong Scaling* is defined as the variation of the resolution time as the number of processes changes, while having a fixed problem size[2] [28]. *Weak Scaling* is defined as the variation of the resolution time as the number of processes changes, while having a fixed problem size per process[3] and therefore, the problem size is proportional to the number of processes [28].

The product $c = n \cdot p$ represents the total processes used, remembering that n is the amount of nodes used and p is the amount of processes run per node. Considering the strong scaling, an increase in the number of processes c, decreases the problem size in each process. Conversely, considering the weak scaling, an increase in the number of processes c, increases the total problem size.

A well recognized metric used to describe the scaling of a program is the parallelism efficiency. The absolute efficiency of the parallelism is:

$$e_c = \frac{t_s}{ct_c} \tag{17}$$

[2]Related to the Amdahl's Law[29].

[3]Related to the Gustafson's Law [30].

where e_c is the absolute efficiency of the parallelism in c processes, t_s is the execution time of the best known serial solution and t_c is the execution time of the program using c processes [28]. In this analysis, the best known serial solution is the one developed on this work, so the equality $t_s = t_1$ is used.

Another metric proposed in this work is the relative efficiency when the number of processes doubles ϵ_c, this is defined as:

$$\epsilon_c = \begin{cases} (e_c)^{1/log_2 c}, & c > 1 \\ 1, & c = 1 \end{cases} \tag{18}$$

The only values used for the experiments were $\epsilon_1 = 1$, $\epsilon_2 = e_2$, $\epsilon_4 = (e_4)^{1/2}$, $\epsilon_8 = (e_8)^{1/3}$ and $\epsilon_{16} = (e_{16})^{1/4}$.

1) Strong scaling: The absolute efficiency for the strong scaling is calculated as:

$$e_c = \frac{time(\text{pCG}(1,1))}{c \cdot time(\text{pCG}(n,p))}. \tag{19}$$

The problem size for the first test is $\hat{q} = 13 \times 13$ and $\hat{l} = 512$, as it is used in [7]. To illustrate the problem size for this configuration, the dimension of matrix $\mathbf{G}$ is 199680×199680 and the dimension of matrix $\mathbf{E}$ is 86528×86528.

TABLE II
TIMES OF pCG$(1, p)$, FOR DIFFERENT INNER TOLERANCE ε_i VALUES, OUTER TOLERANCE $\varepsilon_e = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, INNER GRID SIZE 13×13, $\tau = 1/512$, $\hat{k} = 32$ AND $n.c.$ MEANS THAT THE SYSTEM DOES NOT CONVERGE IN 100 ITERATIONS.

ε_i	pCG$(1,p)$		
	$p=1$	$p=2$	$p=4$
10^{-12}	9.075293	5.867213	4.406441
10^{-10}	9.0619	5.964406	4.398958
10^{-8}	8.99508	5.988965	4.353884
10^{-7}	8.826598	5.693643	4.540562
10^{-6}	9.293346	6.345327	4.236906
10^{-5}	9.908053	6.485624	n.c.
10^{-4}	10.695609	14.248947	7.465788

Table II shows the execution times of the experiments presented in Table I. Fixing the values of p, it can be noticed that the time values do not differ greatly until an inner tolerance of 10^{-5}. When the required precision of the inner solver (Parareal) is increased, the inner iterations count increases, while the outer iterations count (Conjugate Gradient) decreases. Although there are less iterations needed for the low precision cases, the external iterations count increase leads to greater execution times. For a case with inner tolerance of 10^{-5}, the algorithm does not converge. At the same time, for lower precision cases the convergence rate of the outer Conjugate Gradient is lowered.

The inner tolerance value of 10^{-6} is used as a reference, because it is coherent with the outer tolerance value of 10^{-6}. Therefore the base tolerance value of 10^{-6} is chosen for the following tests.

To build Table III, the absolute efficiency e_c of the parallelism is calculated from the data of Table II, following (19).

TABLE III
EFFICIENCY OF pCG$(1, p)$, FOR DIFFERENT INNER TOLERANCE ε_i VALUES, OUTER TOLERANCE $\varepsilon_e = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, INNER GRID SIZE 13×13, $\tau = 1/512$, $\hat{k} = 32$ AND $n.c.$ MEANS THAT THE SYSTEM DOES NOT CONVERGE IN 100 ITERATIONS.

ε_i	pCG$(1,p)$		
	$p=1$	$p=2$	$p=4$
10^{-12}	1	0.77339	0.51489
10^{-10}	1	0.75966	0.51500
10^{-8}	1	0.75097	0.51650
10^{-7}	1	0.77513	0.48599
10^{-6}	1	0.73230	0.54836
10^{-5}	1	0.76384	n.c.
10^{-4}	1	0.37531	0.35815

The efficiency values from Table III show that the average relative efficiency for a single cluster node is approximately $\epsilon = 0.73$ each time the number of processes doubles.

The next step is to calculate the efficiency for multiple cluster nodes. In order to have a viable test, the spatial grid must be larger. The total execution times and the relative efficiency ϵ are calculated with a grid of $\hat{q} = 19 \times 19$. Figure 2 is obtained using the average efficiency of the different inner tolerances $\varepsilon_i = \{10^{-12}, 10^{-10}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$.

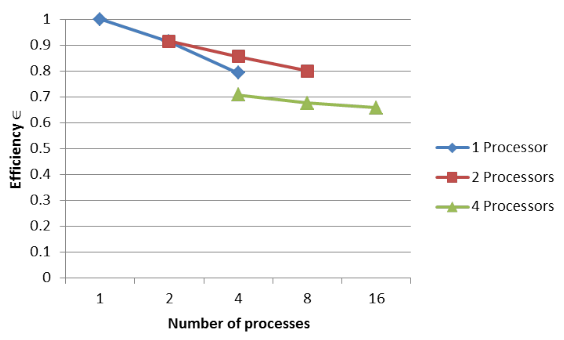

Fig. 2. Average strong scaling (Total time)

The average strong scaling shown in Figure 2 is approximately $\epsilon = 0.7$ each time the number of processes doubles. Considering only the times of the solver (without taking into account the time used to build the matrices and the execution of the external Conjugate Gradient preconditioner), the main contribution of this work can be noticed. The Figure 3 shows the times of the Parareal solver.

On this test, the average relative efficiency on a single cluster node is approximately $\epsilon = 0.79$, that is higher than the one calculated on Table III. This indicates that the strong scaling efficiency increases as the problem size grows.

The next test for the strong scaling is for a big sized problem, that cannot be solved in a single cluster node, and that is near the size limit that two nodes can solve.

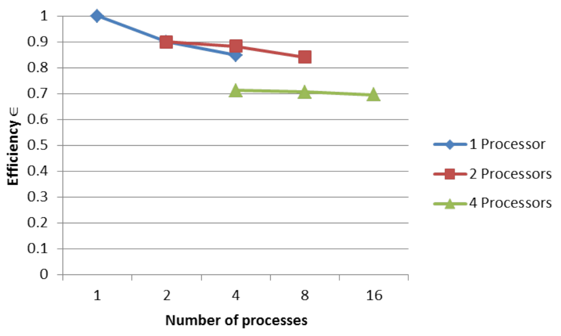

Fig. 3. Average strong scaling (considering only the solver)

The values $\hat{q} = 19 \times 19$ and $\hat{l} = 8192$ are chosen. With these values, the size of matrix $\mathbf{E}$ is 2957312×2957312 (approximately $3 \cdot 10^6 \times 3 \cdot 10^6 = 9 \cdot 10^{12}$ elements) and the size of matrix $\mathbf{G}$ is 6537216×6537216 (approximately $6.54 \cdot 10^6 \times 6.54 \cdot 10^6 \approx 4.3 \cdot 10^{13}$ elements).

TABLE IV
TIMES OF pCG$(1, p)$, INNER TOLERANCE $\varepsilon_i = 10^{-6}$ VALUES, OUTER TOLERANCE $\varepsilon_e = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, INNER GRID SIZE 19×19, $\hat{l} = 8192$, AND $o.o.m.r.$ MEANS THAT THE SYSTEM RUNS OUT OF MEMORY DURING THE RESOLUTION OF THE PROBLEM.

$\hat{k}$	pCG(n, p)					
	$n = 2$			$n = 4$		
	$p = 1$	$p = 2$	$p = 4$	$p = 1$	$p = 2$	$p = 4$
512	o.o.m.r.	o.o.m.r.	o.o.m.r.	506.84255	263.27448	229.03597
256	958.6446	596.9837	483.9645	436.1471	224.44206	183.52355
128	919.7547	471.1759	529.52744	430.2823	224.12969	183.02848
64	926.5926	529.6313	526.7008	428.6131	229.998	187.0255
32	939.2415	523.5229	557.8653	453.0177	242.8749	199.8105
16	922.1194	559.6586	514.7818	513.2409	272.3714	229.5808

It is not possible to compute the efficiency in a classic sense for Table IV, given that there is no serial solution that can solve the proposed problem cases as the system runs out of memory when the matrices are built in a single node. Therefore, to obtain the efficiency, the base execution time may be considered as the one that solves the problem with the least amount of cluster nodes, and the least amount of cores of each node. In the tests shown in Table IV, for $\hat{k} = 512$ and $\hat{m} = 16$ the base case will be pCG$(4, 1)$ and for all the other tested conditions of $\hat{k}$ and $\hat{m}$, it will be pCG$(2, 1)$.

Considering the data of Table V, the average relative efficiency can be established as $\epsilon = 0.7$ each time the number of processes doubles.

From the data presented, the general strong scaling obtained is approximately $\epsilon = 0.7$ each time the number of processes doubles.

As a noteworthy detail to keep in mind, in general, the efficiency decays faster when the number of nodes increases than when the number of processes per node increases. This occurs because the LAN's connection to the new nodes adds latency to the computations and has a lower data transfer rate than the local bus on each node. In general, the LAN's data transfer rate is not enough to keep the same efficiency in processes on different nodes as compared to processes on a same node. There is also a bus bandwidth from the central memory in each node that limits the efficiency of increasing the number of processes per node, as adding more processes will decrease the bandwidth available for each process after a certain point. The limit will depend on the equipment specifications.

TABLE V
EFFICIENCY OF pCG$(1, p)$, INNER TOLERANCE $\varepsilon_i = 10^{-6}$ VALUES, OUTER TOLERANCE $\varepsilon_e = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, INNER GRID SIZE 19×19, $\hat{l} = 8192$, AND $o.o.m.r.$ MEANS THAT THE SYSTEM RUNS OUT OF MEMORY DURING THE RESOLUTION OF THE PROBLEM.

$\hat{k}$	pCG(n, p)					
	$n = 2$			$n = 4$		
	$p = 1$	$p = 2$	$p = 4$	$p = 1$	$p = 2$	$p = 4$
512	o.o.m.r.	o.o.m.r.	o.o.m.r.	1	0.96257	0.55323
256	1	0.80291	0.49520	0.54950	0.53390	0.32647
128	1	0.97602	0.43423	0.53439	0.51296	0.31407
64	1	0.87475	0.43981	0.54046	0.50359	0.30965
32	1	0.89704	0.42091	0.51832	0.48340	0.29379
16	1	0.82382	0.44782	0.44917	0.42319	0.25103

2) *Weak scaling:* The efficiency for the weak scaling is computed as:

$$e_c = \frac{time(\text{pCG}(1, 1))}{time(\text{pCG}(n, p))}. \tag{20}$$

Given that the total problem size grows as the number of used processes c does, there is no need to multiply the denominator by $c = n \cdot p$.

To compute the weak scaling, the problem size per process must be fixed. As a first experimental option, the number of elements from vector $\mathbf{u}$ can be fixed; this is, the size of the solution found by each process $\hat{q} \cdot \hat{m}$ is constant and the number of coarse intervals $\hat{k}$ is shifted to obtain several configurations. Let $\hat{q} = 19 \times 19$, $\hat{m} = 32$ be the fixed size per process, when the total execution times are measured, the relative efficiency ϵ is computed. Figure 4 shows the average of the relative efficiency for the coarse instants per node $\hat{k}/n = \{1, 2, 4, 8, 16\}$.

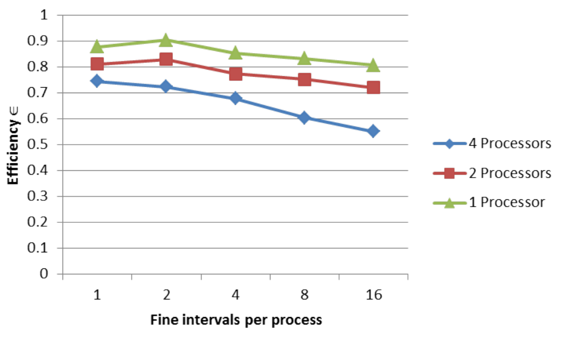

Fig. 4. Average weak scaling (Total time)

The average weak scaling form Figure 4 is approximately $\epsilon = 0.75$ each time the number of processes doubles. Considering only the execution time of the function pCG(n, p), the weak scaling is approximately $\epsilon = 0.85$ each time the number of processes doubles, with an increasing efficiency as the problem size increases, as it can be observed in Figure 5 (this is a meaningful improvement with respect to the efficiency of the whole program, when the scaling efficiency drops to $\epsilon = 0.29$ for the pCG$(4, 4)$ with $\hat{k} = 256$).

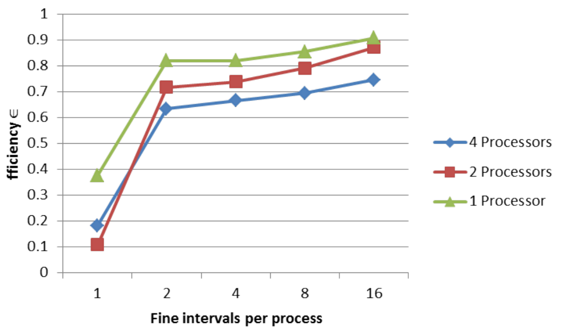

Fig. 5. Average weak scaling (considering only the solver)

The problem with the pCG's preconditioner is that it uses standard PETSc operations to solve the inverse of matrix $\mathbf{E}$. In particular for the test pCG$(4, 4)$ with $\hat{k} = 256$, the time to build the matrices is 9.21 seconds, 332 seconds for the preconditioner and 48.7 seconds for the pCG function. This clearly shows that solving a single time the inverse of matrix $\mathbf{E}$ by the standard means is much slower than the multiple solutions done by the pCG function.

The problem with the first consideration for the fixed problem size is that the matrices involved on the problem have their size squared compared to the solution vector. Then, as a second experimental option, the fixed data size per process is the size of the matrices stored on each process. With this approach, and setting the parameters $\hat{q} = 19 \times 19$ and $\hat{m} = 32$, each execution time is measured, then the relative efficiency ϵ is computed. The average of the relative efficiency for the coarse instants per node $\hat{k}/n = \{1, 2, 4, 8, 16\}$ is shown in Figure 6.

A detail to have in mind is that to allow the size of the matrices to be constant, on all the tests, the only possible combinations of n and p are those that make $c = n \cdot p$ a perfect square. This occurs because $\hat{k}$ must be divisible by c and the matrices's sizes are proportional to the square of $\hat{k}$. Therefore, the number of elementes of matrix $\mathbf{E}$, $size(\mathbf{E})$ will be set as reference.

Most iterative Krylov subspace methods have a computational complexity of $\mathcal{O}(sol_s \cdot iter)$ where sol_s is the solution size and $iter$ the number of iterations needed for the convergence of the algorithm [31]. The number of iterations using the Conjugate Gradient method is bounded by $1 \leq iter \leq sol_s$ because it can be used as a direct method [32]. The first experimental option tests the lower $iter$ bound and the second experimental option tests the upper $iter$ bound, because the number of iterations needed for the convergence is unknown before the execution of the solver.

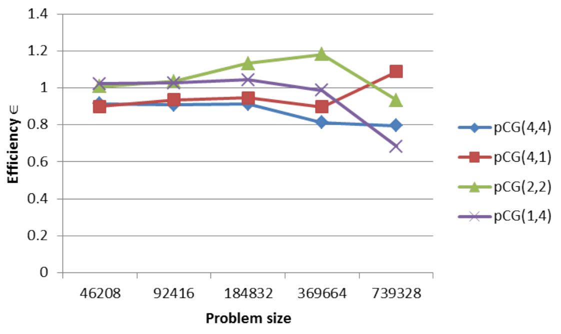

Fig. 6. Average weak scaling (Total time)

Because the number of iterations the solver used to converge in each experiment is greater than 1 and less than $\hat{l}\hat{p}$ (equal to sol_s), the first experimental option will give a scaling less than 1, and the second experiment can give a scaling greater than 1. This can be seen in Figure 6, where in some cases the scaling is greater than 1.

On average, for the second experimental option, the relative efficiency of the weak scaling is approximately $\epsilon = 0.96$ each time the number of processes doubles.

The same process as in the first experimental option is used for the solver time, Figure 7 shows the weak scaling of the pCG function.

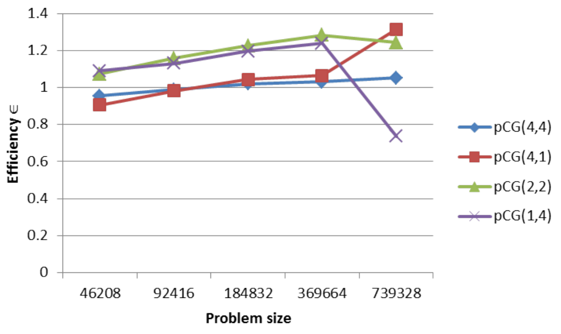

Fig. 7. Average weak scaling (considering only the solver)

It can be noticed in Figure 7 that most of the resolution times (not considering the preconditioner) scale on a supraliear way. In this context, the average relative efficiency is approximately $\epsilon = 1.09$ each time the number of processes doubles.

On both weak scaling considerations, the efficiency lost as the number of nodes increases is greater than the efficiency lost as the number of cores used per node increases. This is expected as it was analyzed on the strong scaling, due to the data transfer among cores on a single node is faster than the transfer among cores on different nodes, because the data has to travel through the switch holding the LAN.

The analysis continues comparing the problem resolution using or not the Parareal method.

D. *Parareal vs No Parareal*

To solve the external Conjugate Gradient, the steps 3) and 5) from Algorithm 2 must be solved by some iterative method. Three options to solve those steps are compared, *a*) the Parareal implementation done for this work *b*) the PETSc's implementation of the Conjugate Gradient, and *c*) the PETSc's implementation of GMRES.

The parameters used for the test are $\hat{q} = 9 \times 9$, $\hat{l} = 256$, $\hat{m} = 16$, while the values $\hat{k} = \{16, 32\}$ are considered. The Conjugate Gradient did not converge for any test case from this set, therefore only the Parareal and GMRES methods are compared.

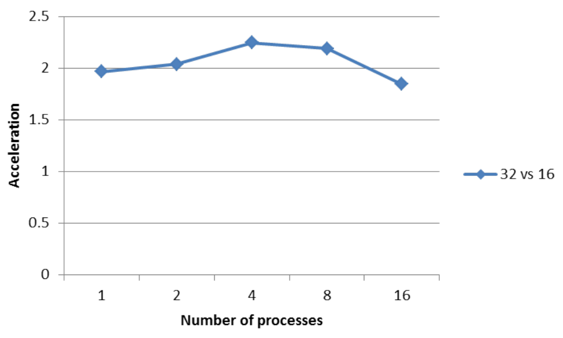

Fig. 8. Acceleration of Parareal vs GMRES (Total time)

The quotient from the resolution time of the GMRES and the the Parareal methods gives the speed-up for the cases $\hat{k} = 16$ and $\hat{k} = 32$. Then, the quotient from the cases $\hat{k} = 32$ and $\hat{k} = 16$ yield the acceleration obtained as the problem size doubles. This is presented in Figures 8 and 9.

Figures 8 and 9 show that the execution of the Parareal takes less time than the execution of the GMRES for every problem size tested, since the acceleration is higher than 1 for every case. Indeed, when the problem size increases the quotient from the execution time of the GMRES and the execution time from the Parareal increases with an average of 2 when the solution size doubles.

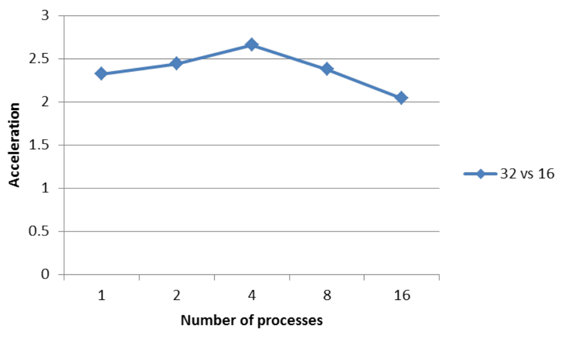

Fig. 9. Acceleration of the Parareal vs GMRES (Solver only)

As hinted, a considerable time difference was expected, after analysing the strong scaling tests, and for that reason the problem sizes for the test were small. The acceleration shown in Figures 8 and 9 are from linear systems where the size of matrix $\mathbf{E}$ are 20736×20736 (when $\hat{k} = 16$) and 41472×41472 (when $\hat{k} = 32$) respectively.

The main time load for the preconditioner of the external Conjugate Gradient is the solution of a system $\mathbf{E}x = b$. With the execution time of the preconditioner ($t_{precond}$), the resolution time of the system using GMRES (t_{GMRES}) can be estimaded by $t_{precond} \cdot iter_{cg} \approx t_{GMRES}$, where $iter_{cg}$ is the iterations needed for the convergence of the pCG (even if it uses the Parareal, as the iteration count for the external Conjugate Gradient is stable for every inner method tested).

The time approximation variation var (of the approximated value versus the measured value) found from the tests of this section, calculated by $var = |t_{GMRES} - t_{precond} \cdot iter_{cg}|/t_{GMRES}$, is at most 20%.

To emphasize the time reduction achieved using the Parareal method, the time of pCG$(4,4)$ with $\hat{q} = 19 \times 19$ and $\hat{l} = 8196$ is analyzed. In this case, building the matrices takes 9.21 seconds, the external preconditioner takes 332 seconds and the pCG function takes 48.7 seconds (with 16 iterations). Solving this pCG$(4,4)$ case with the GMRES solver would take $332 \cdot 16 - 48.7 = 5263$ seconds more (almost one hour and a half compared to 49 seconds when using the Parareal).

E. *Parallelization suitability*

When the strong scaling was analyzed, it was found that an increase in the problem size would give a better parallelization efficiency on each node.

Extending that idea, on a sufficiently small problem the cost of creating new processes and the communication costs among the processes would be higher than the time benefit obtained with the parallelization. The parallelization is convenient when the solution of the pCG(1,1) takes longer than every other analyzed solution of pCG(n,p). The tests are done with grid sizes that gives problem sizes of approximately twice as big each time.

Table VI is used to find the convenient problem size for the parallelization on a single node.

TABLE VI
TIMES OF PCG(n,p), OUTER TOLERANCE $\varepsilon_e = 10^{-6}$, INNER TOLERANCE $\varepsilon_i = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, $\hat{l} = 512$ AND $\hat{k} = 32$.

$\hat{q}$	pCG(n,p)		
	$n = 1$		
	$p = 1$	$p = 2$	$p = 4$
4×4	1.6396871	2.6615668	2.7016241
7×7	2.434376	2.5144259	2.447761
9×9	3.673742	3.319042	2.524157

The convenient minimum problem size on a single node for matrix $\mathbf{E}$ is 41472×41472 elements ($\hat{q} = 9 \times 9$ and $\hat{l} = 512$). In a similar fasion, multiple nodes are analyzed on Table VII.

Table VII shows that, in general, the parallelization is suitable starting on a matrix $\mathbf{E}$ of 86528×86528 elements ($\hat{q} = 13 \times 13$ and $\hat{l} = 512$).

TABLE VII
TIMES OF pCG(n, p), OUTER TOLERANCE $\varepsilon_e = 10^{-6}$, INNER TOLERANCE $\varepsilon_i = 10^{-6}$, $\alpha = 1$, $\beta = 12$, $\gamma = 10^{-5}$, $\hat{l} = 512$ AND $\hat{k} = 32$.

$\hat{q}$	pCG(n, p)					
	$n = 2$			$n = 4$		
	$p = 1$	$p = 2$	$p = 4$	$p = 1$	$p = 2$	$p = 4$
7×7	3.012665	2.7350261	2.472511	3.914736	3.915797	3.9322839
9×9	3.723882	2.760471	2.400935	3.842674	3.819844	4.711632
13×13	6.70257	4.045551	2.946394	6.058765	4.413698	4.337244

VI. CONCLUSIONS

- The experiments of Section V-D shows that the execution of the program using the Parareal method is considerably faster than executions that use the CG or GMRES methods. Not only it does perform better, but it accelerates for larger problem sizes.
- The experiments of Section V-C present a relative efficiency of around $\epsilon = 0.7$ each time the number of processes doubles for the strong scaling. At the same time, for the weak scaling, the relative efficiency is $\epsilon = 0.75$ each time the number of processes doubles for a constant solution size per process, and $\epsilon = 0.96$ each time the number of processes doubles for a constant data size per process.
- The experiments of Section V-C find that for the used hardware (described in Section V), the parallelization begins to be convenient for solution size of 40000 elements.

In summary, this paper presented a parallel efficient alternative in PETSc to solve a parabolic optimal control problem using the Parareal mthod. Experimental results above summarized demonstrate the advantages of this proposal over classical methods as the Conjugate Gradient and GMRES ones in a computing cluster.

REFERENCES

[1] J. Carlsson, "Optimal Control of Partial Differential Equations in Optimal Design," PhD Dissertation, KTH, School of Computer Science and Communication (CSC), Numerical Analysis and Computer Science, NADA, 2008.

[2] G. Biros and O. Ghattas, "Parallel Lagrange-Newton-Krylov-Schur methods for PDE-Constrained optimization I. The Krylov-Schur solver," *SIAM Journal on Scientific Computing*, no. 27, p. 687–713, 2005.

[3] J. L. Lions, *Optimal control of systems governed by partial differential equations*. Springer, 1971.

[4] T. P. Mathew, M. Sarkis, and C. E. Schaerer, "Analysis of block parareal preconditioners for parabolic optimal control problems," *SIAM*, no. 32, pp. 1180–1200, 2010.

[5] J. J. C. Silva, "Implementación de un esquema de paralelización temporal distribuida usando petsc," Diploma thesis, Ciencias y Tecnología - Universidad Católica Nuestra Señora de la Asunción, feb 2014.

[6] G. Strang, *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1998.

[7] X. Du, M. Sarkis, C. E. Schaerer, and D. Szyld, "Inexact and truncated Parareal-in-time Krylov subspace methods for parabolic optimal control problems," *Electronic Transactions on Numerical Analysis*, 2013.

[8] J. Fritz, *Partial Differential Equations*. Springer, 1982.

[9] J.-L. Lions, Y. Maday, and G. Turinici, "Résolution d'edp par un schéma en temps "pararéel"," *C. R. Acad. Sci. Paris Sér. I Math.*, vol. 332, no. 7, pp. 661–668, 2001.

[10] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, no. 14, pp. 1–137, 2005.

[11] T. Rees, M. Stoll, and A. Wathen, "All-at-once preconditioning in PDE-constrained optimization," *Kybernetika*, no. 46, p. 341–360, 2010.

[12] P. Neittaanmäki and D. Tiba, *Optimal Control of Nonlinear Parabolic Systems*. Marcel Dekker, Inc., 1994.

[13] O. Bashir, K. Willcox1, O. Ghattas, B. van Bloemen Waanders, and J. Hill, "Hessian-based model reduction for large-scale systems with initial condition inputs," *International Journal for Numerical Methods in Engineering*, vol. 73, no. 6, pp. 844–868, 2008.

[14] T. P. Mathew, *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*. Springer, 2008.

[15] W. Samyono, "Hessian matrix-free Lagrange-Newton-Krylov-Schur-Schwarz methods for elliptic inverse problems," PhD Dissertation, Old Dominion University, 2006.

[16] J. Gallier, "The Schur Complement and Symmetric Positive Semidefnite (and Defnite) Matrices," *Penn Engineering*, 2010.

[17] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[18] R. D. Falgout and U. M. Yang, "hypre: a library of high performance preconditioners," *Numerical Solution of Partial Differential Equations on Parallel Computers*, 2002.

[19] M. J. Gander and S. Vandewalle, "Analysis of the parareal time-parallel time-integration method," *SIAM Journal on Scientific Computing*, no. 29, pp. 556–578, 2007.

[20] C. E. Schaerer and E. Kaszkurewicz, "The shooting method for the solution of ordinary differential equations: A control-theoretical perspective," *International Journal of Systems Science*, vol. 32, no. 8, 2001.

[21] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 3rd ed. Springer-Verlag, 2002.

[22] E. H.-L. Liu, "Fundamental Methods of Numerical Extrapolation With Applications," *MIT Open Course Ware*, 2006.

[23] F. Zhang, "The Schur Complement and Its Applications," *Springer*, 2005.

[24] C. E. Schaerer, E. Kaszkurewicz, and N. Mangiavacchi, "A Multilevel Schwarz Shooting Method for the solution of the Poisson Equation in Two Dimensional Incompressible Flow Simulations," *Applied Mathematics and Computation*, vol. 153, no. 3, pp. 803–831, 2004.

[25] M. S. Gockenbach, *Partial differential equations: analytical and numerical methods*. SIAM, 2002.

[26] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[27] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.3, 2012.

[28] A. Kaminsky, *BIG CPU, BIG DATA: Solving the World's Toughest Computational Problems with Parallel Computing*. Rochester Institute of Technology, 2013.

[29] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *AFIPS spring joint computer conference*, 1967.

[30] J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, 1988.

[31] C. Vogel, *Computational Methods for Inverse Problems*. SIAM, 2002.

[32] Y. Saad, "Krylov subspace methods for solving large unsymmetric linear systems," *Mathematics of Computation*, 1981.