A Parallel Implementation of the Mortar Element Method in 2D and 3D

Abdoulaye Samake, Silvia Bertoluzza, Micol Pennacchio, Christophe Prud'Homme, Chady Zaza

ESAIM: PROCEEDINGS, Vol. ?, 2013, 1-10
Editors: Will be set by the publisher

A PARALLEL IMPLEMENTATION OF THE MORTAR ELEMENT METHOD IN 2D AND 3D

Abdoulaye Samake[1], Silvia Bertoluzza[2], Micol Pennacchio[3], Christophe Prud'homme[4] and Chady Zaza[5]

Abstract. We present here the generic parallel computational framework in C++ called Feel++ for the mortar finite element method with the arbitrary number of subdomain partitions in 2D and 3D. An iterative method with block-diagonal preconditioners is used for solving the algebraic saddle-point problem arising from the finite element discretization. Finally we present a scalability study and the numerical results obtained using Feel++ library.

Keywords: Domain decomposition and mortar method and parallel computing

1. Introduction

Domain decomposition methods are becoming increasingly popular as a tool to solve problems arising in many different applications. The possibility of easily coupling different discretizations and/or different numerical methods in different subdomains without the need of imposing strong matching conditions is the main feature of the nonconforming version of such methods and adds a further advantage.

In this paper we consider an algorithm presented in [2] for solving the linear system arising from the *mortar* element method (initially proposed by C. Bernardi, Y. Maday and A. Patera in [8]) as well as another variant with Lagrange multipliers proposed by F. Ben Belgacem and Y. Maday in [6]. The main feature of the Mortar method is that the interface continuity conditions of the subdomains is taken into account in weak form by asking the jump of the finite element solution on the interface to be L_2-orthogonal to a well chosen Lagrange multiplier space. The approximation properties of such a method are optimal in the sense that the error is bounded by the sum of the subdomain approximation errors.

In order to make such techniques more competitive for real life applications, one has to deal with the problem of efficient implementation. As it happens with all domain decomposition methods (both conforming and non-conforming) the efficient implementation relies on parallelizing the solution process by assigning each subdomain to a processor and employing the preferred iterative scheme.

[1] Université Joseph Fourier Grenoble 1, / CNRS, Laboratoire Jean Kuntzmann / UMR 5224. Grenoble, F-38041, France e-mail: abdoulaye.samake@imag.fr

[2] IMATI CNR Italy,e-mail: Silvia.Bertoluzza@imati.cnr.it

[3] IMATI CNR Italy,e-mail: Micol.Pennacchio@imati.cnr.it

[4] Université de Strasbourg / CNRS, IRMA / UMR 7501. Strasbourg, F-67000, France, e-mail: prudhomme@unistra.fr

[5] Commissariat à l'Energie Atomique, DEN/DANS/DM2S/STMF/LMEC. CEA Cadarache, 13108 Saint Paul lez Durance, France. e-mail: chady.zaza@cea.fr

The paper is organized as follows. In section 2 we recall the mortar finite element method with Lagrange multipliers. In section 3 the computational framework is described and some remarks on the parallel implementation aspects are given. Finally, the numerical results showing strong and weak scaling on large architectures in 3D are presented in section 4 and we give brief conclusions in section 5.

2. The Mortar Method

Let Ω be bounded domain of $\mathbb{R}^d$, $d = 2, 3$. We denote $\partial\Omega$ its boundary and we assume that Ω is a union of L subdomains Ω_k:

$$\Omega = \bigcup_{k=1}^{L} \Omega_k. \tag{1}$$

We assume that the domain decomposition is geometrically conforming. It means that if $\gamma_{kl} = \overline{\Omega}_k \cap \overline{\Omega}_l$ $(k \neq l)$ and $\gamma_{kl} \neq \emptyset$, then γ_{kl} must either be a common vertex of Ω_k and Ω_l, or a common edge, or a common face. We define $\Gamma_{kl} = \gamma_{kl}$ as the interface between Ω_k and Ω_l. We note that $\Gamma_{kl} = \Gamma_{lk}$.
We consider the Dirichlet boundary value problem (2): find u satisfying

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega, \\ u &= g \quad \text{on} \quad \partial\Omega, \end{aligned} \tag{2}$$

where $f \in L^2(\Omega)$ and $g \in H^{1/2}(\partial\Omega)$ are given functions. The usual variational formulation of (2) reads as follows

Problem 2.1. Find $u \in H^1_g(\Omega)$ such that

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx \qquad \forall v \in H^1_0(\Omega). \tag{3}$$

where H^1_g denotes the space $H^1_g = \{u \in H^1(\Omega), u = g \text{ on } \partial\Omega\}$.

Let us denote by $H^{1/2}(\Gamma_{kl})$ the trace space of one of the spaces $H^1(\Omega_k)$ or $H^1(\Omega_l)$ on the interface Γ_{kl}. We define two product spaces:

$$V = \prod_{k=1}^{L} H^1(\Omega_k), \quad \Lambda = \prod_{k=1}^{L} \prod_{\substack{0 \le l < k \\ |\Gamma_{kl}| \neq 0}} \left(H^{1/2}(\Gamma_{kl})\right)'. \tag{4}$$

The space Λ will be a trial space for the weak continuity conditions on the interfaces. We introduce the bilinear forms $a : V \times V \to \mathbb{R}$, $b : V \times \Lambda \to \mathbb{R}$ and the linear functional $f : V \to \mathbb{R}$:

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) &= \sum_{k=1}^{L} a_k(\mathbf{u}, \mathbf{v}), \quad a_k(\mathbf{u}, \mathbf{v}) = \int_{\Omega_k} \nabla u_k \cdot \nabla v_k \, dx, \\ b(\lambda, \mathbf{v}) &= \sum_{k=1}^{L} \sum_{\substack{l=0 \\ |\Gamma_{lk}| \neq 0}}^{L} b_{kl}(\lambda, \mathbf{v}), \quad b_{kl}(\lambda, v) = \langle \lambda_{kl}, v_k \rangle|_{\Gamma_{kl}}, \\ f(\mathbf{v}) &= \sum_{k=1}^{L} \int_{\Omega_k} f v_k \, dx, \end{aligned}$$

where $\lambda_{kl} = -\lambda_{lk}$ and $\langle \cdot, \cdot \rangle|_{\Gamma_{kl}}$ stands for the duality product between $\left(H^{1/2}(\Gamma_{kl})\right)'$ and $H^{1/2}(\Gamma_{kl})$. The bilinear form $a_k(\cdot, \cdot)$ corresponds to the Dirichlet problem in the subdomain Ω_k for the operator $-\Delta$.

The approach that we consider here is the mortar formulation with Lagrange multipliers presented in [2] that reads as follows:

Problem 2.2. Find $(\mathbf{u}, \lambda) \in V \times \Lambda$ such that

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) + b(\lambda, \mathbf{v}) &= f(\mathbf{v}), \\ b(\mu, \mathbf{u}) &= 0, \\ \forall (\mathbf{v}, \mu) &\in V \times \Lambda. \end{aligned} \tag{5}$$

If $\tilde{\mathbf{u}}$ and $\tilde{\lambda}$ denote the vectors of the components of $\mathbf{u}_h$ and λ_h, finite element approximations to $\mathbf{u}$ and λ, the discrete problem associated to the problem (2.2) is equivalent to a saddle-point system of the following form:

$$\mathcal{A} \begin{pmatrix} \tilde{\mathbf{u}} \\ \tilde{\lambda} \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}, \qquad \mathcal{A} = \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \tag{6}$$

where

$$A = \begin{pmatrix} A_1 & & 0 \\ & \ddots & \\ 0 & & A_L \end{pmatrix}, \qquad B^T = \begin{pmatrix} B1^T \\ \vdots \\ B_L^T \end{pmatrix}$$

and A_k corresponds to the stiffness matrix in the subdomain Ω_k.

Remark 2.3. Moreover, there are other *mortar* formulations namely *master-slave* constraint formulation in which the *weak continuity constraint* is directly taken into account in the approximations space called *constraint space*.

3. Computational Framework

3.1. Krylov Iterative solvers

We want to solve the saddle-point problem (6) using an iterative Krylov subspace method in parallel. Finding a good preconditioner for such problems is a delicate issue as the matrix $\mathcal{A}$ is indefinite and any preconditioning matrix $\mathcal{P}$ acting on the jump matrices B_{kl} would involve communications.
A survey on block diagonal and block triangular preconditioners for this type of saddle-point problem (6) can be found in [7, 12].
The matrix $\mathcal{A}$ arising in saddle-point problems is known to be spectrally equivalent to the block diagonal matrix:

$$\mathcal{P} = \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix}$$

where S is the Schur complement $-BA^{-1}B^T$ (see for instance [15]). While not being an approximate inverse of $\mathcal{A}$, the matrix $\mathcal{P}$ is an ideal preconditioner. Indeed it can be shown that $P(X) = X(X-1)(X^2 - X - 1)$ is an annihilating polynomial of the matrix $\mathcal{T} = \mathcal{P}^{-1}\mathcal{A}$. Therefore, assuming $\mathcal{T}$ non singular, the matrix $\mathcal{T}$ has only three eigenvalues $\{1, (1 \pm \sqrt{5})/2\}$. Thus an iterative solver using the Krylov subspaces constructed with $\mathcal{T}$ would converge within three iterations. In practice, computing the inverse of the exact preconditioner $\mathcal{P}$ is too expensive. Instead, one would rather look for an *inexact* inverse $\hat{\mathcal{P}}^{-1}$. When applying the preconditioner, the inexact inverse $\hat{\mathcal{P}}^{-1}$ would be determined following an iterative procedure for solving the linear system $\mathcal{P}\mathbf{x} = \mathbf{y}$.

This requires a class of iterative methods qualified as *flexible inner-outer preconditioned solvers* [18] or *inexact inner-outer preconditioned solvers* [13].

The *outer* iterations for solving the main problem would involve *inner* iterations for computing an inexact and *non-constant* preconditioner. Finding the relevant convergence parameters to this inner iterative procedure is a critical issue. On the one hand, $\hat{\mathcal{P}}^{-1}$ has to be computed in few iterations: the total number of iterations including the inner iterations should be less than without preconditioner. On the other hand for ensuring the stability of the outer iterations, it would be preferable to solve the inner iterations with as much accuracy as possible in order to keep an almost constant preconditioner. We refer the reader to [11] and references therein for theoretical results and experimental assessment with respect to the influence of the perturbation to the preconditioner. In this context, the choice a good preconditioner for solving the inner iterations can have a significant impact on the convergence of the outer iterations.

In this work the outer iterations will be carried out with a flexible preconditioned Biconjugate Gradient Stabilized Method (`FBICGSTAB`) (see for instance algorithm (7.7) in [19]) and the flexible preconditioned Generalized Minimal Residual Method (`FGMRES(m)`) with restart (see [20]):

Algorithm 1 Flexible Preconditioned `FGMRES(m)`

for $k = 1, 2, \ldots$ maxiter **do**
  $\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$
  $\beta = \|\mathbf{r}_0\|_2$
  $\mathbf{v}_1 = \mathbf{r}_0/\beta$
  $\mathbf{p} = \beta\mathbf{e}_1$
  for $j = 0, 1, \ldots$ m **do**
    solve $\mathcal{P}\mathbf{z}_j = \mathbf{v}_j$
    $\mathbf{w} = \mathcal{A}\mathbf{z}_j$
    for $i = 1, 2, \ldots j$ **do**
      $h_{i,j} = (\mathbf{w}, \mathbf{v}_i)$
      $\mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$
    end for
    $h_{j+1,j} = \|\mathbf{w}\|_2$
    $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
    for $i = 1, 2, \ldots j-1$ **do**
      $h_{i,j} = c_i h_{i,j} + s_i h_{i+1,j}$
      $h_{i+1,j} = -s_i h_{i,j} + c_i h_{i+1,j}$
    end for
    $\gamma = \sqrt{h_{j,j}^2 + h_{j+1,j}^2}$
    $c_j = h_{j,j}/\gamma; \quad s_j = h_{j+1,j}/\gamma$
    $h_{j,j} = \gamma; \quad h_{j+1,j} = 0$
    $p_j = c_j p_j; \quad p_{j+1} = -s_j p_j$
    if $|p_{j+1}| \leq \varepsilon$ **then**
      exit loop
    end if
  end for
  $\mathcal{Z}^m \longleftarrow [\mathbf{z}_1 \cdots \mathbf{z}_m]$
  $\mathcal{H}^m \longleftarrow \left(h_{i,j}\right)_{1 \leq i \leq j+1; 1 \leq j \leq m}$
  $\mathbf{y} = \mathrm{Argmin}_{\mathbf{q}} \|\mathbf{p} - \mathcal{H}^m\mathbf{q}\|_2$
  $\mathbf{x} = \mathbf{x}_0 + \mathcal{Z}^m\mathbf{y}$
  if $|p_{j+1}| \leq \varepsilon$ **then**
    exit loop
  else
    $\mathbf{x}_0 = \mathbf{x}$
  end if
end for

Algorithm 2 Flexible Preconditioned `FBICGSTAB`

$\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$
$\tilde{\mathbf{r}}_0 = \mathbf{r}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
$\mathbf{v}_0 = \mathbf{r}_0$
$\rho_0 = \alpha = \omega_0 = 1$
for $j = 0, 1, \ldots$ maxiter **do**
  $\rho_{j+1} = (\tilde{\mathbf{r}}_0, \mathbf{r}_j)$
  $\beta = (\rho_{j+1}/\rho_j) \times (\alpha/\omega_j)$
  $\mathbf{p}_{j+1} = \mathbf{r}_j + \beta(\mathbf{p}_j - \omega_j\mathbf{v}_j)$
  solve $\mathcal{P}\hat{\mathbf{p}} = \mathbf{p}_{j+1}$
  $\mathbf{v}_{j+1} = \mathcal{A}\hat{\mathbf{p}}$
  $\alpha = \rho_{j+1}/(\hat{\mathbf{r}}_0, \mathbf{v}_{j+1})$
  $\mathbf{s} = \mathbf{r}_j - \alpha\mathbf{v}_{j+1}$
  solve $\mathcal{P}\hat{\mathbf{s}} = \mathbf{s}$
  $\mathbf{t} = \mathcal{A}\hat{\mathbf{s}}$
  $\omega_{j+1} = (\mathbf{t}, \mathbf{s})/(\mathbf{t}, \mathbf{t})$
  $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha\hat{\mathbf{p}} + \omega\hat{\mathbf{s}}$
  $\mathbf{r}_{j+1} = \mathbf{s} - \omega_{j+1}\mathbf{t}$
end for

Regarding the preconditioning we will focus on two approximations of $\mathcal{P}$:

$$\mathcal{P}_I = \begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix} \qquad \text{and} \qquad \mathcal{P}_S = \begin{pmatrix} A & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

In the first preconditioner the exact inverse of $\mathcal{P}_I$ is computed at each iteration using the $(I)LU$ factorization of the block diagonal matrix A. This preconditioner only acts on the diagonal blocks A_k. As a result, solving the linear system $\mathcal{P}_I\mathbf{x} = \mathbf{y}$ does not involve any communication between the subdomains. However, since $\mathcal{P}_I$ does not act on the jump matrices B_{kl}, it is very likely to become less effective as the number of subdomains increases.

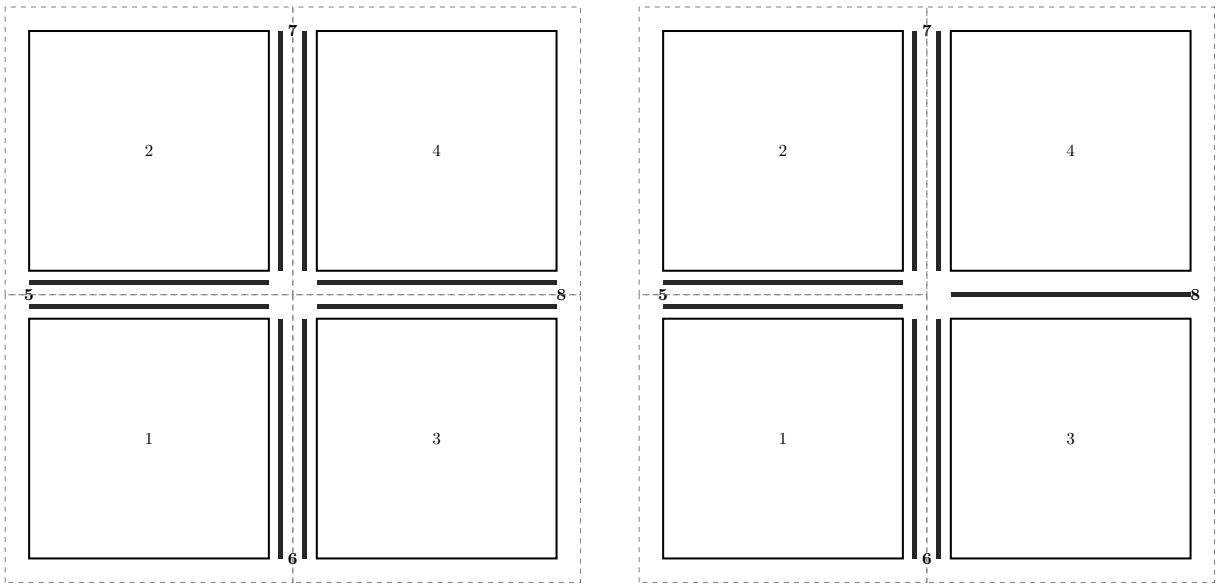
In the second preconditioner the exact inverse of the block diagonal matrix A is also computed so that the exact Schur complement $S = -BA^{-1}B^T$ is readily available. Instead of taking S we choose an approximation $\hat{S}$ such that $\hat{\mathbf{x}} = \hat{S}^{-1}\mathbf{y}$ is an approximate solution to the linear system $S\mathbf{x} = \mathbf{y}$ following an iterative procedure. This inner procedure is also carried out with a `BICGSTAB` algorithm preconditioned with the diagonal of S (Jacobi preconditioner $\mathcal{M}_J$) or with $\mathcal{M}_S^{-1} = BAB^T$.

Remark 3.1. The Krylov methods `FBICGSTAB` and `FGMRES(m)` are both adapted to our saddle-point problem, but the only major difference between these methods is that `FBICGSTAB` presents sometimes *breakdowns* unlike `FGMRES(m)`.

3.2. **Parallel implementation**

The parallel implementation is designed using the message passing interface(`MPI`) library. The objective of the parallel implementation is to minimize the amount of communications with respect to the parallel operations involved in the linear solver, namely matrix-vector products and dot products. One of interests of this mortar parallel implementation is that there's no communication at cross-points(in 2D and 3D) and cross-edges(in 3D), which reduces considerably communications between subdomains.

Assuming a constant number of internal dofs in each subdomain, it is rather straightforward to bind a subdomain to each process. Each process would own its subdomain mesh $\mathcal{T}_{h_k}$, functional space X_{h_k}, stiffness matrix A_k and unknown u_k. Regarding the mortars, the choice is less obvious. In order to decrease the amount of communications in the matrix-vector products, we have used technique developed in [2] which consists in duplicating the data at the interfaces between subdomains. If Γ_{kl} is such an interface, then the Lagrange multiplier vector λ_{kl} and its associated trace mesh $\mathcal{T}_{h_k,l}$ and trace space $\mathcal{M}_{h_k,l}$ are stored in both the processors dealing Ω_k and Ω_l. Although the data storage is increased a little bit, the communications will be reduced significantly.



1.1. One subdomain per cluster

1.2. Subdomains 3 and 4 on the same cluster

FIGURE 1. Domain decompositions

As an example, consider the splitting of the unit square into four little squares, as in FIGURE 1., where the dash rectangles denote clusters and the bold segments correspond to the mortar interfaces. Note, that when

neighboring subdomains belong to different clusters, there are two copies of the mortar interface variables stored in different clusters. Consider the interfaces as shown in the picture. The matrix $\mathcal{A}$ has the following form:

$$\mathcal{A} = \left(\begin{array}{cccc|cccc} A_1 & & & & B_{15}^T & B_{16}^T & \mathbf{0} & \mathbf{0} \\ & A_2 & & & B_{25}^T & \mathbf{0} & B_{27}^T & \mathbf{0} \\ & & A_3 & & \mathbf{0} & B_{36}^T & \mathbf{0} & B_{38}^T \\ & & & A_4 & \mathbf{0} & \mathbf{0} & B_{47}^T & B_{48}^T \\ \hline B_{15} & B_{25} & \mathbf{0} & \mathbf{0} & & & & \\ B_{16} & \mathbf{0} & B_{36} & \mathbf{0} & & & & \\ \mathbf{0} & B_{27} & \mathbf{0} & B_{38} & & & & \\ \mathbf{0} & \mathbf{0} & B_{47} & B_{48} & & & & \end{array}\right)$$

Let us consider the matrix-vector multiplication procedure with the matrix $\mathcal{A}$ and the vector $(\tilde{u}, \tilde{\lambda})$, where $\tilde{u}$ and $\tilde{\lambda}$ have the following component-wise representation, according to the decomposition and the enumeration in FIGURE 1.1.: $\tilde{\mathbf{u}} = (u_1^T, u_2^T, u_3^T, u_4^T)^T$ and $\tilde{\lambda} = (\lambda_5^T, \lambda_6^T, \lambda_7^T, \lambda_8^T)^T$. The resulting vector $(\tilde{\mathbf{v}}, \tilde{\mu}) = \mathcal{A} \cdot (\tilde{\mathbf{u}}, \tilde{\lambda})$ can be computed as

$$\left(\begin{array}{c} v_1^{(1)} \\ v_2^{(2)} \\ v_3^{(3)} \\ v_4^{(4)} \\ \hline \mu_5^{(1,2)} \\ \mu_6^{(1,3)} \\ \mu_7^{(2,3)} \\ \mu_8^{(3,4)} \end{array}\right) = \left(\begin{array}{c} A_1 u_1^{(1)} + B_{15}^T \lambda_5^{(1)} + B_{16}^T \lambda_6^{(1)} \\ A_2 u_2^{(2)} + B_{25}^T \lambda_5^{(2)} + B_{27}^T \lambda_7^{(2)} \\ A_3 u_3^{(3)} + B_{36}^T \lambda_6^{(3)} + B_{38}^T \lambda_8^{(3)} \\ A_4 u_4^{(4)} + B_{47}^T \lambda_7^{(4)} + B_{48}^T \lambda_8^{(4)} \\ \hline B_{15} u_1^{(1)} + B_{25} u_2^{(2)} \\ B_{16} u_1^{(1)} + B_{36} u_3^{(3)} \\ B_{27} u_2^{(2)} + B_{47} u_4^{(4)} \\ B_{38} u_3^{(3)} + B_{48} u_4^{(4)} \end{array}\right) \tag{7}$$

where the upper indices denote the cluster(the processor), in which this variable is stored. Two upper indices mean that this variable is stored in both processors. Note that $\lambda_i^{(k)} \equiv \lambda_i^{(l)}$ and so far we need communications only when computing μ_i. For example

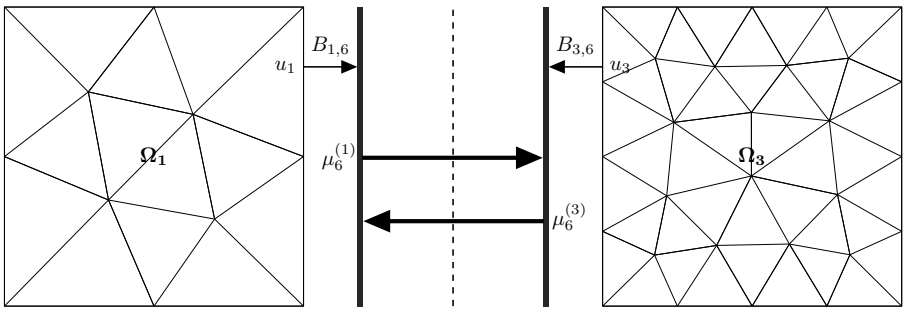


FIGURE 2. Communications for jump matrix multiplication

$$\mu_6 = \mu_6^{(1)} + \mu_6^{(3)}, \qquad \mu_6^{(1)} = B_{16} u_1 \quad \text{and} \quad \mu_6^{(3)} = B_{36} u_3.$$

We see that $\mu_6^{(1)}$ and $\mu_6^{(3)}$ are computed in parallel, and then should be interchanged and summed(see the representations in FIGURE 3. and more explicitly in FIGURE 2.) .

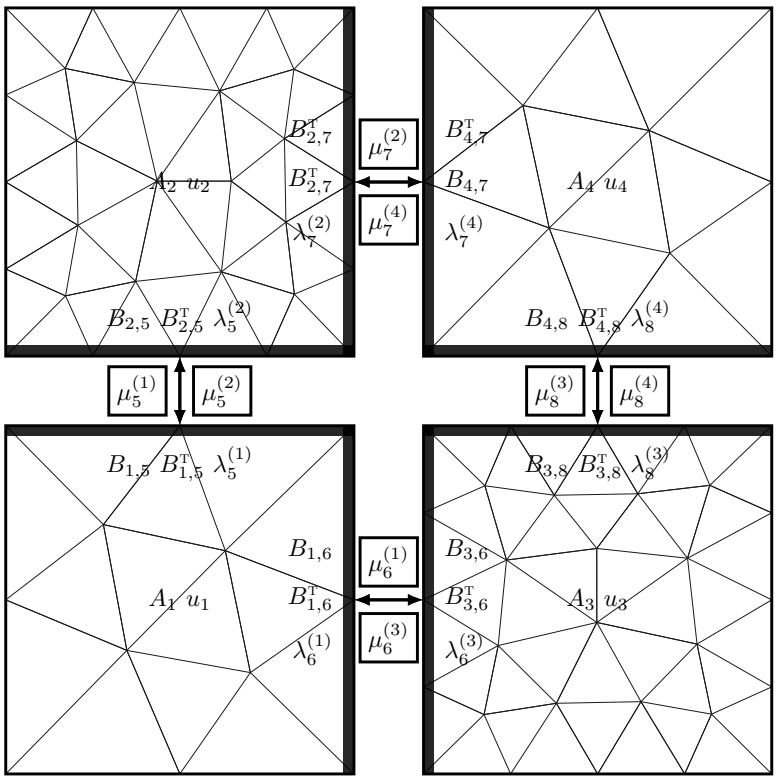


FIGURE 3. Communications for parallel matrix-vector multiplication

For the FEEL++ [16, 17] implementation the communications are handled explicitly by the user and we use PETSC [3–5, 14] sequentially even though the code is parallel using `MPI` communicators. This technique requires explicitly sending and receiving complex data structures such as mesh data structures, PETSC vectors and elements of functions space(traces) using `Boost.MPI` and `Boost.Serialization` [1].

4. NUMERICAL RESULTS

We present in this section the numerical results of the parallel implementation of the mortar element method described in the section (3) using FEEL++ and `Boost.MPI` libraries. We consider here the problem (2) in 3D with $g = \sin(\pi x)\cos(\pi y)\cos(\pi z)$ the exact solution and $f = -\Delta g = 3\pi^2 g$ the corresponding right hand side. The problem is solved in the parallelepiped $\Omega = [0, L_x]\times[0, L_y]\times[0, L_z]$, $L_x, L_y, L_z > 0$. The following numerical results are obtained using P_2 finite element approximations in each subdomain Ω_k with $h_{\Omega_k} = 0.1$ in the strong scaling study(see subsection 4.1) and $h_{\Omega_k} = 0.075$ in the weak scaling one(see subsection 4.2), $k = 1, \cdots, L$. The stopping criterion is such that the residual norm for the Krylov solver is less than $\varepsilon = 10^{-7}$.

The simulations have been performed at UJF/LJK on Syrah. Syrah cluster is made of 2 nodes named syrah-local and Grenache-local. Each node has 2 x Intel Xeon L5640 2.27GHz granted 12 Cores with 128Gb of RAM on Syrah-local and 64Gb of RAM on Grenache-local. The cluster has a total resource of 24 Cores and 192Gb of RAM.

In the context of High Performance Computing(HPC), there are two common notions of scalability to evaluate the efficiency of the parallel computing.

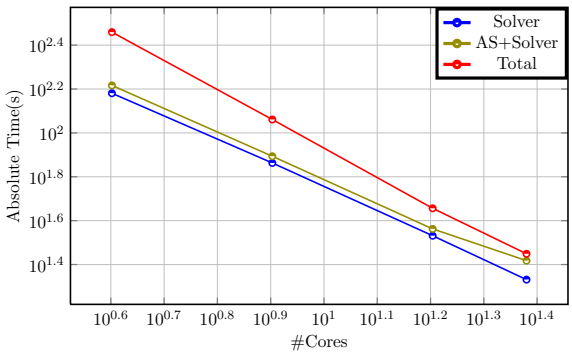
The first is the ***strong scaling***, which is defined as how the solution time varies with the number of cores for a fixed total problem size. The goal is to minimize time to solution for a given problem by keeping the problem

size fixed and increasing the number of cores.
The second is the ***weak scaling***, which is defined as how the solution time varies with the number of cores for a fixed problem size per core. The goal is to solve the larger problems by keeping the work per core fixed and increasing the number of cores.

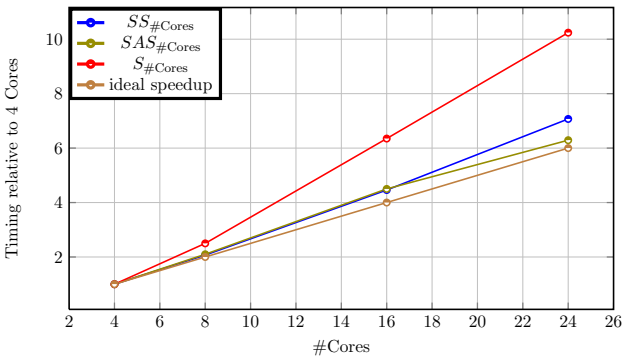
4.1. **Strong Scaling**

We present here the strong scaling results corresponding to the partition of the global domain Ω into $L_x \times L_y \times L_z$ subdomains (1 subdomain per core) with the fixed lengths $L_x = L_y = L_z = 1$. We plot in FIGURE 4.1. the absolute solve time and assembly+solver time and total time versus the number of cores in `loglog` axis and in FIGURE 4.2. the speedup and ideal speedup versus number of cores. The total number of degrees of freedom is approximately equal to 300.000 and all the the measured timings are expressed in seconds.

We define the speedup by the following formula: $S_p = T_r/T_p$ where p is the number of cores and T_r the execution time of the parallel algorithm with r cores($r = 4$ is our reference number of cores) and T_p the execution time of the parallel algorithm with p cores($r < p$). Analogously we define by SS_p the corresponding speedup for only the solver time and by SAS_p for assembly and solver .



4.1. Absolute Solve and Total time versus #Cores



4.2. Speedup versus #Cores

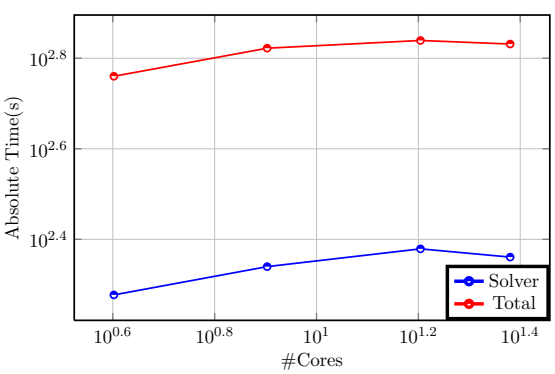
FIGURE 4. Strong scaling

Remark 4.1. We observe in FIGURE 4.2. that the speedup related to the total computational time is very over the ideal speedup. This is due to the fact that the functions spaces construction and matrix factorization timings decrease significantly in the strong scaling, as well as the communications between subdomains which are few in only 24 cores. We expect to find the normal behavior on the large scale architectures on which we can take many subdomains with more consistent problem size.

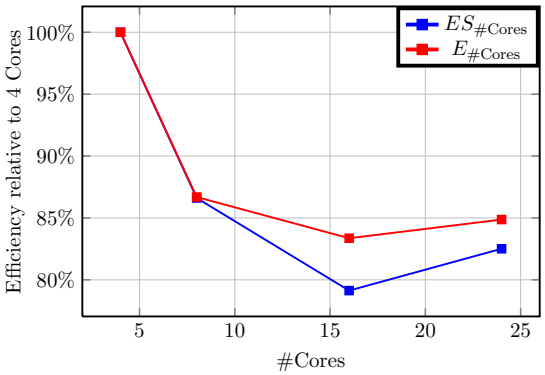
4.2. **Weak Scaling**

We present here the the weak scaling results corresponding to the partition of the global domain Ω into $L_x \times L_y \times L_z$ subdomains (1 subdomain per processor) with $L_x \times L_y \times L_z = \#\text{Cores}$. We plot in FIGURE 5.1. the absolute solve time and total time versus the number of cores in `loglog` axis and in FIGURE 5.2. the efficiency relative to four cores versus number of cores. We denote by E_p the efficiency relative to four cores for the total time on p cores and ES_p the efficiency relative to four cores for the solver time on p cores. The number of degrees of freedom is approximately equal to 12.000 per subdomain and all the measured timing are expressed in seconds.

Remark 4.2. The plots in FIGURE 5.1. clearly show that the absolute solve and total time does not increase significantly when the number of cores and the problem size increase by keeping the problem size per core. This confirms the results expected for the weak scaling study.



5.1. Absolute Solve and Total time versus #Cores



5.2. Efficiency versus #Cores

FIGURE 5. Weak scaling

4.3. **Convergence Results**

We summarize in the following Tables TABLE 1. and TABLE 2. the behavior of L_2 and H^1 errors of the numerical solution relative to the analytical solution of our problem using the mortar finite formulation described in the section 2 according to the maximum mesh size $h \in \{0.2, 0.1, 0.05, 0.025\}$ and the Lagrange polynomial orders P_N, $N \in \{1, 2, 3\}$. The tests are performed in the nonconforming case where the characteristic mesh size in the subdomain Ω_k is $h_{\Omega_k} = h + \delta_k$, $k = 1, \cdots, L$, with $\delta_k = 0.001$ the small perturbation. All the tests are achieved with 2, 4, 8 and 16 number of subdomains. We denote by u the exact solution of our problem and u_h^N the discrete solution obtained by using the characteristic size equal to h and the piecewise polynomials of degree less than or equal to N. We denote $\|\cdot\|_0$ the L^2-norm and $\|\cdot\|_1$ the H^1-norm. We plot in FIGURE 6.1. and FIGURE 6.2. the L^2 and H^1 norms of the error relative to the exact solution versus the characteristic mesh sizes in `loglog` axis.

TABLE 1. L^2 Convergence results

h	$\|u - u_h^1\|_0$	$\|u - u_h^2\|_0$	$\|u - u_h^3\|_0$
0.2	$2.80 \cdot 10^{-2}$	$2.67 \cdot 10^{-3}$	$2.16 \cdot 10^{-4}$
0.1	$6.69 \cdot 10^{-3}$	$2.83 \cdot 10^{-4}$	$9.58 \cdot 10^{-6}$
0.05	$1.66 \cdot 10^{-3}$	$3.24 \cdot 10^{0}$	$5.29 \cdot 10^{-7}$
0.025	$4.00 \cdot 10^{-4}$	$3.91 \cdot 10^{-6}$	$3.10 \cdot 10^{-8}$

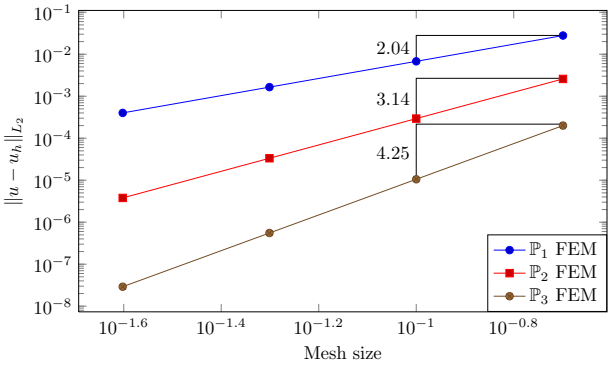
TABLE 2. H^1 Convergence results

h	$\|u - u_h^1\|_1$	$\|u - u_h^2\|_1$	$\|u - u_h^3\|_1$
0.2	$7.92 \cdot 10^{-1}$	$1.09 \cdot 10^{-1}$	$1.10 \cdot 10^{-2}$
0.1	$3.72 \cdot 10^{-1}$	$2.44 \cdot 10^{-2}$	$1.08 \cdot 10^{-3}$
0.05	$1.83 \cdot 10^{-1}$	$5.88 \cdot 10^{-3}$	$1.25 \cdot 10^{-4}$
0.025	$8.93 \cdot 10^{-2}$	$1.43 \cdot 10^{-3}$	$1.49 \cdot 10^{-5}$

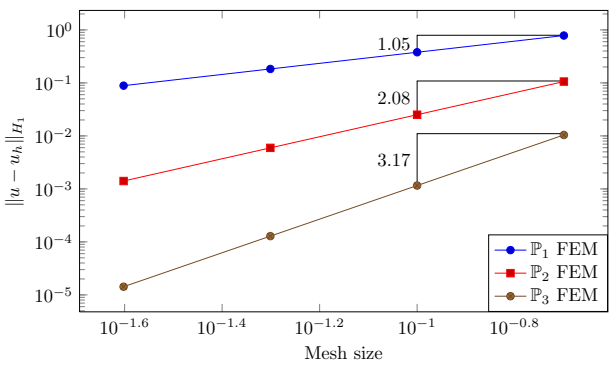
Remark 4.3. Note that the convergence results above clearly shows that our formulation checked the best convergence orders certified by the finite element theory. In addition, the above convergence results are obtained in the nonconforming configuration which illustrates the flexibility of the *mortar* element method.

5. CONCLUSIONS

This paper clearly shows that our parallel computational framework scales on the small scale architectures for solving the linear system arising from the *mortar* finite element method in 2D and 3D with the arbitrary number of subdomain partitions using the block-diagonal preconditioners. Our current work is focused on the implementation of the substructuring preconditioners [9, 10] for the discrete Steklov-Poincaré operator on



6.1. L^2 convergence



6.2. H^1 convergence

FIGURE 6. Convergence curves

interfaces so that the number of iterations is maintained as low as possible that is needed for a good scaling on very large scale architectures.

ACKNOWLEDGEMENTS

The authors would like to thank Vincent Chabannes for many fruitful discussions. Abdoulaye Samake and Christophe Prud'homme acknowledge the financial support of the project ANR HAMM ANR-2010-COSI-009.

REFERENCES

[1] Boost c++ libraries. http://www.boost.org.
[2] G.S. Abdoulaev, Y. Achdou, Y.A. Kuznetsov, and C. Prud'homme. On a parallel implementation of the mortar element method. *RAIRO-M2AN Modelisation Math et Analyse Numerique-Mathem Modell Numerical Analysis*, 33(2):245–260, 1999.
[3] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
[4] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc.
[5] Satish Balay, Victor Eijkhout, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
[6] F. Ben Belgacem and Y. Maday. The mortar element method for three-dimensional finite elements. *R.A.I.R.O. Modl. Math. Anal.*, 31:289–302, 1997.
[7] M. Benzi, G.H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14(1):1–137, 2005.
[8] C. Bernardi, Y. Maday, and A. Patera. A new nonconforming approach to domain decomposition:the mortar element method. *Nonlinear Partial Differential Equations and their Applications*, 1993.
[9] S. Bertoluzza and M. Pennacchio. Preconditioning the mortar method by substructuring: The high order case. *Applied Numerical Analysis & Computational Mathematics*, 1(2):434–454, 2004.
[10] Silvia Bertoluzza and Micol Pennacchio. Analysis of substructuring preconditioners for mortar methods in an abstract framework. *Applied Mathematics Letters*, 20(2):131 – 137, 2007.
[11] Jie Chen, L.C. McInnes, and H. Zhang. Analysis and practical use of flexible BICGSTAB. Technical Report ANL/MCS-P3039-0912, Argonne National Laboratory, 2012.
[12] H.C. Elman, D.J. Silvester, and A.J. Wathen. *Finite Elements and Fast Iterative Solvers:with Applications in Incompressible Fluid Dynamics: with Applications in Incompressible Fluid Dynamics*. Numerical Mathematics and Scientific Computation. OUP Oxford, 2005.
[13] G.H. Golub and Q. Ye. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM Journal on Scientific Computing*, 21(4):1305–1320, 1999.

[14] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, 2005.
[15] M.F. Murphy, G.H. Golub, and A.J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM Journal on Scientific Computing*, 21(6):1969–1972, 2000.
[16] Christophe Prud'homme. A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations. Scientific Programming, 14(2):81-110, 2006.
[17] Christophe Prud'homme. Life: Overview of a unified C++ implementation of the finite and spectral element methods in 1d, 2d and 3d. In Workshop On State-Of-The-Art In Scientific And Parallel Computing, Lecture Notes in Computer Science, page 10. Springer-Verlag, 2007.
[18] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.
[19] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.
[20] Youcef Saad. A flexible inner-outer preconditioned gmres algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, March 1993.