PERFORMANCE ANALYSIS OF THE MHM SIMULATOR IN A PETASCALE MACHINE

Antônio Tadeu Azevedo Gomes

Weslley da Silva Pereira

Roberto Pinto Souto

Frédéric Valentin

atagomes@lncc.br

weslleyp@lncc.br

rpsouto@lncc.br

valentin@lncc.br

LNCC - Laboratório Nacional de Computação Científica

25651-075 Petrópolis, RJ, Brasil

Diego Paredes

diego.paredes@pucv.cl

PUCV - Pontificia Universidad Catolica de Valparaiso

Valparaiso, Chile

Abstract. *We present the performance analysis of a parallel simulator that implements the multiscale hybrid-mixed (MHM) finite element method for the elastostatic equation. The formulation behind MHM comprises a global problem defined over the skeleton of a coarse mesh and a set of independent elasticity problems posed over element-wise submeshes. The performance analysis considers an isotropic elastic 3D model. For the strong scaling, we characterize whether it is more efficient to refine the submeshes and use polynomial degrees of lower order or vice-versa while keeping the same approximation error. We obtained the better efficiency— greater than 70% on 3,072 cores— when using higher-order polynomials. For the weak scaling, we fixed the high-order polynomials and varied the level of the refinement of the submeshes. We then increased the size of the problem by refining the global mesh at the same rate as the increase in the number of cores. The measurements suggest the configurations with refined submeshes offer the best efficiency— greater than 80% on 3,072 cores.* ***Keywords:*** *Scientific computing, Advanced numerical algorithms, Parallel and distributed computing*

CILAMCE 2017

Proceedings of the XXXVIII Iberian Latin-American Congress on Computational Methods in Engineering
P.O. Faria, R.H. Lopez, L.F.F. Miguel, W.J.S. Gomes, M. Noronha (Editors), ABMEC, Florianópolis, SC, Brazil,
November 5-8, 2017

1 INTRODUCTION

The numerical simulation of multiscale phenomena, such as fluid flows in porous media and wave propagation in nano-structures, provides effective ways to solve relevant problems in areas such as energy, climate/weather, and health. In this context, the use of Finite Element Methods (FEM) is particularly disseminated in industry and academia to simulate these phenomena. Despite the fact that classical FEM methods (e.g. the Galerkin method) are appropriate for many different problems, their accuracy may seriously decay when they face problems with highly multiscaled features. Because of that, many modern FEM techniques have emerged (Hou et al. 1997; Efendiev et al. 2015). Among them, of particular interest to this paper is the family of Multiscale Hybrid-Mixed (MHM) methods originally proposed in (Harder et al. 2013) and (Araya, Harder, Paredes, et al. 2013). The MHM methods appear as a highly competitive option to handle realistic multiscale boundary value problems (BVPs). From a mathematical viewpoint, the MHM methods naturally incorporate multiple scales (the MHM-levels) and provide solutions with high-order precision on coarse meshes. From a computational viewpoint, at a lower MHM-level a set of completely independent *local* problems can be posed over submeshes defined within each element of the coarse mesh. These local problems can then be computed in parallel to provide information to its upper MHM-level—the so-called *global* problem, which is posed over the skeleton of the coarse mesh. As a result, the MHM methods are naturally shaped to be used in parallel computing environments.

The MHM methods have been numerically validated for problems such as the Darcy equation (Harder et al. 2013), the Stokes/Brinkman models (Araya, Harder, Poza, et al. 2016), the advective-reactive dominated problems (Harder et al. 2015), and the linear elasticity problem (Harder, Madureira, et al. 2016). In this paper, we present a software library, developed in C++, specifically crafted for the MHM methods. The library offers hybrid parallelism by means of MPI and OpenMP and strongly explores the C++ *Standard Template Library* (STL) together with packages such as METIS, Eigen, Pardiso and MUMPS to leverage efficiency for some of the fundamental data structures and algorithms. Besides, we present a preliminary performance evaluation of an application that employs the library to solve the elastostatic equation, considering as our model problem an isotropic elastic three-dimensional model with constant physical coefficients defined in the unit cube. The evaluation takes into account the strong and weak scalability efficiency of the application. We collected scalability measurements of MHM simulations in the Santos Dumont HPC cluster at LNCC in Brazil. For such measurements, we explored a set of MHM configurations over our model problem. The key idea was to characterize:

- whether it is more efficient to coarsen the global mesh and refine the submeshes, or vice-versa, while capturing the same level of multiscale features; and
- whether it is more efficient to refine the submeshes and use polynomial degrees of lower order in the local and global problems, or vice-versa, while keeping the same approximation error.

The strong scaling measurements suggest that a balanced approach—with a modest level of global mesh refinement—leads to smaller execution times, and that the better efficiency—greater than 70% on 3,072 cores—is obtained when higher-order polynomials are used. We then fixed high-order polynomials and varied the level of the refinement of the submeshes for the weak scaling measurements. For each level of refinement we collected such measurements

by refining the global mesh (i.e. increasing the overall problem size) at the same rate as the increase in the number of cores. The weak scaling measurements suggest the configurations with refined submeshes offer the best efficiency—greater than 80% on 3,072 cores. As a final remark on such scaling measurements, the refinement of the submeshes, having fixed all the other parameters, does not reduce the approximation errors; rather, they stay in the same order. This is a consequence of the adopted model problem, which has constant physical coefficients in the domain. For more realistic scenarios, with highly heterogeneous coefficients, the refinement of the submeshes is expected to reduce the approximation errors up to the point in which further refinements do not capture additional multiscale features.

The remainder of this paper is structured as follows. Section 2 overviews the family of MHM methods. Section 3 presents the key computing primitives of the MHM software library and how they relate to the algorithm implied by the MHM methods. Section 4 shows the architecture and key design principles that underlie the MHM software library. Section 5 offers a performance evaluation of the library, using an elastostatic application as the test case. Finally, Section 6 presents some concluding remarks and perspectives for future work.

2 THE MHM METHODS

To present the MHM methods in a compact way, we consider the BVP to find $u : \Omega \mapsto \mathbb{R}^d$, $d \in \{2, 3\}$, so that $\mathcal{L}u = f$ in an open-bounded domain Ω with prescribed homogeneous Dirichlet boundary condition on the polygonal boundary $\partial\Omega$. Hereafter $\mathcal{L}$ is a bounded linear operator. Some definitions will be needed in the sequel.

First, let $\{\mathcal{T}_H\}_{H\geq 0}$ be a family of regular triangulations of Ω such that each mesh $\mathcal{T}_H$ decomposes in a set of globally-indexed elements $\{K_t\}_{t\in T}$, with $T \subset Z^+ = \{0, \ldots, N_t - 1\}$. This set defines a *mesh* with N_t elements. The label H refers to the *mesh size*, where H is the maximum diameter of any element in the mesh. Each element K has a boundary ∂K consisting of a set of faces F, and some of these faces (those who are internal to the mesh) may be shared between two elements. Denote by $\partial\mathcal{T}_H$ the set of ∂K, and by $\mathcal{E}_H$ the set in the triangulation of all globally-indexed faces $\{F_e\}_{e\in E}$, with $E \subset Z^+ = \{0, \ldots, N_e - 1\}$. $\mathcal{E}_H$ defines the *skeleton* of a mesh with N_e faces. Figure 1 illustrates these concepts for $\mathbb{R}^2$. We associate each face F to a unitary normal vector $\mathbf{n}$, taking care to ensure this is facing outward on $\partial\Omega$. Also, the unitary outward normal vector on a face F of element K is represented by $\mathbf{n}_F^K$.

Second, we define some function spaces:

- V_0 is the null space (with basis $\{v_0, \ldots, v_{N_{V_0}-1}\}$) of operator $\mathcal{L}\,|_K$ for all $K \in \mathcal{T}_H$. As an example, for the Darcy equation V_0 is formed by piecewise constants in each K (Harder et al. 2013), and then $N_{V_0} = N_t$;
- $V_0^\perp := \bigoplus_{K\in\mathcal{T}_H} V_0^\perp(K)$ is the orthogonal complement of V_0 such that it yields the well-posedness of $\mathcal{L}\,|_K$ for all $K \in \mathcal{T}_H$;
- $V := \bigoplus_{K\in\mathcal{T}_H} V_K$ is the space defined as the direct sum between V_0 and $V_0^\perp$. As an example, for the Darcy equation $V_K = H^1(K)$ (Harder et al. 2015);
- Λ is the space containing the Lagrange multipliers defined over $\partial\mathcal{T}_H$.

For these spaces, we also define the L^2 inner products $(\cdot,\cdot)_{\mathcal{T}_H} := \sum_{K\in\mathcal{T}_H}(\cdot,\cdot)_K$ and $(\cdot,\cdot)_{\partial\mathcal{T}_H} := \sum_{K\in\mathcal{T}_H}(\cdot,\cdot)_{\partial K}$, respectively.

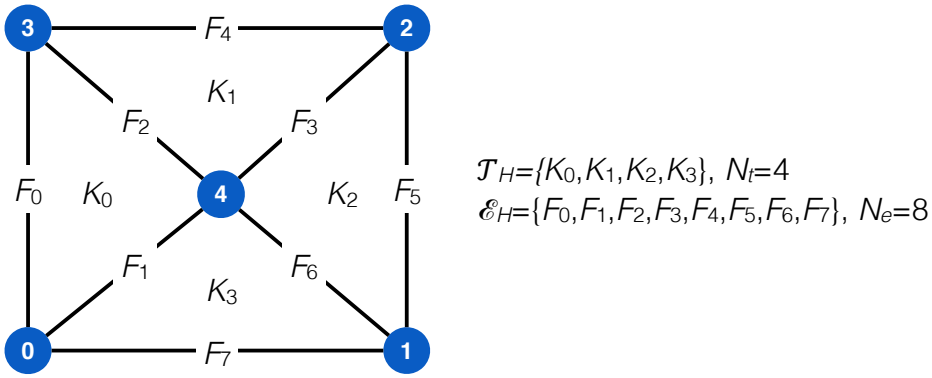


Figure 1: Mesh example in $\mathbb{R}^2$.

Third, we define the following finite-dimensional space needed for approximating the exact field u:

- $\Lambda_H^{m,l} \subset \Lambda$ is the space of discontinuous polynomial functions on $F_e \in \mathcal{E}_H$ of degree less than or equal to $l \geq 0$ and defined on a uniform partition of F_e in m parts ($m \geq 1$), with basis $\{\psi_0, \ldots, \psi_{L_l-1}\}$.

L_l is the total number of degrees of freedom (DoFs) for $\Lambda_H^{m,l}$, and its value depends on the adopted degree l of the piecewise polynomials in $\Lambda_H^{l,m}$, the total number of partitions m in each face F_e, and the total number of faces N_e in the mesh. See Fig. 2 for an example.

Given these definitions, a BVP can be written within the MHM terminology in terms of a collection of local problems—each K being associated with a different local problem—brought together by a global problem defined on $\mathcal{T}_H$. Specifically, the global formulation is as follows: Find $(\lambda_H, u_0) \in \Lambda_H^{m,l} \times V_0$ such that

$$\begin{cases} (T_h\lambda_H, \mu_l)_{\partial\mathcal{T}_H} + (u_0, \mu_l)_{\partial\mathcal{T}_H} &= -(\hat{T}_h f, \mu_l)_{\partial\mathcal{T}_H} \\ (\lambda_H, q_0)_{\partial\mathcal{T}_H} &= (f, q_0)_{\mathcal{T}_H} \end{cases} \tag{1}$$

for all $(\mu_l, q_0) \in \Lambda_H^{m,l} \times V_0$. In the above formulation, f is given; T_h and $\hat{T}_h$ are linear bounded operators driven by the local problems.

Figure 3(a) illustrates an example of the distribution of the DoFs in the case of the Darcy equation for both u_0 and λ_H (with $l = 1$ and $m = 1$) in the mesh of Figure 1.

The local operators T_h and $\hat{T}_h$ in (1) are responsible for the upscaling process. To precise

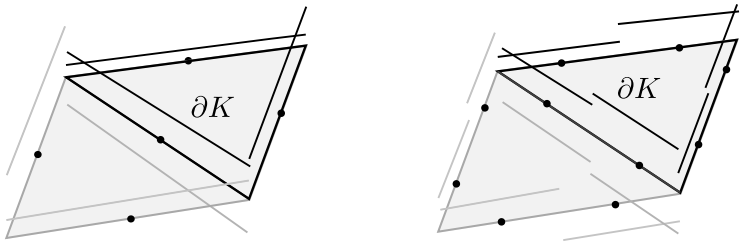


Figure 2: $\Lambda_H^{1,0}$ (left) and $\Lambda_H^{2,0}$ (right) interpolation spaces on faces in $\mathbb{R}^2$ (Harder et al. 2015).

them, let:

- $V_h(K_t) \subset V_0^{\perp}(K_t)$ be a finite-dimensional space (with basis $\{\phi_0^t, \ldots, \phi_{D_{K_t}-1}^t\}$) defined within an element K_t in the mesh;
- $a(u_h, w_h)_{K_t}$ be a bilinear form that sets up T_h and $\hat{T}_h$, being defined (as usual) as the action of $\mathcal{L}u_h$ on w_h in K_t, with $u_h, w_h \in V_h(K_t)$.

At the local level, each element K_t is considered a domain on its own and as such may have an internal triangulation, thus defining a *submesh* within K_t. Each of these submeshes may have a different characteristic size $h \geq 0$, depending on the high-contrast aspects of the coefficients involved in operator $\mathcal{L}$ for each K_t. The number of DoFs D_{K_t} in $V_h(K_t)$ will therefore depend on the degree of the polynomial functions in $V_h(K_t)$ and h. For the purposes of this paper, we assume the solution to each local problem is approximated using the continuous Galerkin method with a Lagrange basis for $\mathbb{P}_h^k$ (the space of piecewise polynomials of degree less than or equal to k); D_{K_t} is then determined by the degree k of the polynomials and the number of elements in the submesh. Figure 3(b) illustrates an example of $V_h(K_t)$ with quadratic interpolation $\mathbb{P}_h^2$ for element K_3 in the mesh of Figure 1. In the example, K_3 has been internally triangulated generating a submesh with 4 elements.

Each local problem comprises a set of local *subproblems*. One such subproblem defines $\hat{T}_h$ by computing $\eta_t^f := \hat{T}_h f \in V_h(K_t)$ in each $K_t \in \mathcal{T}_H$ as follows: Find $\eta_t^f \in V_h(K_t)$ such that

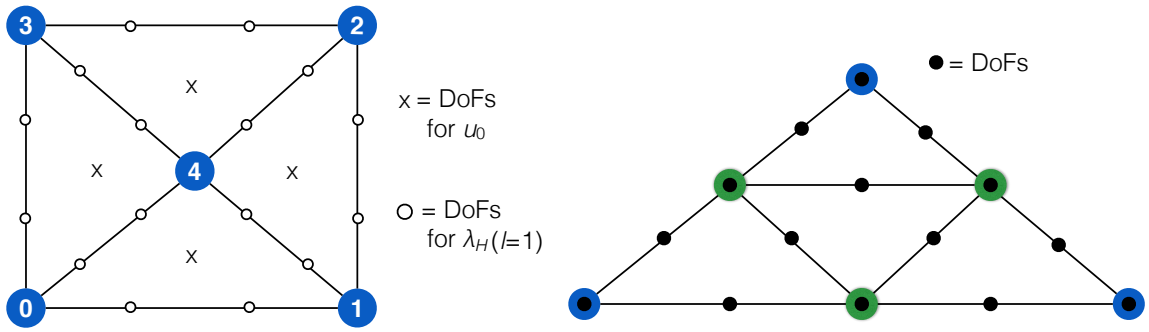
$$a(\eta_t^f, w_h)_{K_t} = (f, w_h)_{K_t} \quad \text{for all } w_h \in V_h(K_t). \tag{2}$$

The other N_I subproblems define T_h by computing $\eta_t^i := T_h\psi_i$ from the following local problems: For $i \in \{0, \ldots, N_I - 1\}$, find $\eta_t^i \in V_h(K_t)$ such that

$$a(\eta_t^i, w_h)_{K_t} = -(\psi_i, w_h)_{\partial K_t} \quad \text{for all } w_h \in V_h(K_t), \tag{3}$$

where ψ_i changes its sign according to $\mathbf{n} \cdot \mathbf{n}_F^{K_t}$, and N_I is defined by the number of faces in K_t and by the degree l of the polynomials and m in $\Lambda_H^{m,l}$. Finally, from the solution of Equations (1), (2) and (3), and using that $\lambda_H = \sum_{i=0}^{N_I-1} k_i\psi_i$ with $k_i \in \mathbb{R}$, the exact field u is approximated by

$$\begin{aligned} u_{H,h} &:= u_0 + T_h\lambda_H + \hat{T}_h f \\ &= u_0 + \sum_{K_t \in \mathcal{T}_H} \left[\sum_{i=0}^{N_I-1} k_i\eta_t^i + \eta_t^f \right]. \end{aligned} \tag{4}$$



(a) Global problem. (b) Local problem.

Figure 3: DoFs distribution in MHM.

3 THE LIBRARY PRIMITIVES

The MHM software library is built on top of key computing primitives that bear correspondence with the major mathematical building blocks of the MHM method: SPLITPROBLEM, SOLVELOCALPROBLEM, REDUCELOCALPROBLEMS, and COMPUTESOLUTION. These primitives are combined to implement a MHM simulator as depicted abstractly in Algorithm 1.

Algorithm 1 Pseudocode of *main* procedure for a MHM simulator.

Require: $\mathcal{T}_H = \{K_t\}_{t \in T}$ ▷ Mesh with indexed elements
  $P_{\mathcal{T}_H} \leftarrow$ SPLITPROBLEM($\mathcal{T}_H$) ▷ $P_{\mathcal{T}_H} := \{(K_t, \{F_t\}\}_{t \in T}$
  $S_{\mathcal{T}_H} \leftarrow \emptyset$
  for all $p_t \in P_{\mathcal{T}_H}$ **do** ▷ $p_t := (K_t, \{F_t\})$
    $s_t \leftarrow$ SOLVELOCALPROBLEM(p_t) ▷ $s_t := (\eta_t^f, \{\eta_t^i\}_{i \in \{0,\ldots,N_I-1\}})$
    $S_{\mathcal{T}_H} \leftarrow S_{\mathcal{T}_H} \cup s_t$
  end for
  $(u_0, \lambda_H) \leftarrow$ REDUCELOCALPROBLEMS($S_{\mathcal{T}_H}$)
  $u_{H,h} \leftarrow$ COMPUTESOLUTION(u_0, λ_H, $S_{\mathcal{T}_H}$)

The SPLITPROBLEM primitive divides the original problem posed over a mesh resulting from a triangulation $\mathcal{T}_H$ into a global problem (the 1st MHM-level) and a set of local problems (the 2nd MHM-level), one for each element K_t in the mesh. This primitive returns a set $P_{\mathcal{T}_H}$ of pairs $(K_t, \{F_t\})$, where $\{F_t\}$ represents the set of all faces of K_t.

The SOLVELOCALPROBLEM computes the contributions η_t^f and $\{\eta_t^i\}_{i \in \{0,\ldots,N_I-1\}}$ from a specific element K_t in the mesh to be used in the global problem. Functions η_t^f and η_t^i, $i \in \{0, \ldots, N_I - 1\}$ are searched as the solution of (2) and (3), respectively, in the following form:

$$\eta_t^f = \sum_{j=0}^{D_{K_t}-1} c_j^t \phi_j^t \quad \text{and} \quad \eta_t^i = \sum_{j=0}^{D_{K_t}-1} d_{i,j}^t \phi_j^t. \tag{5}$$

The coefficients c_j^t and $\{d_{i,j}^t\}_{i \in \{0,\ldots,N_I-1\}}$ are the actual numerical results computed by the SOLVELOCALPROBLEM primitive through the solution of a set of linear systems of the form $\underline{\mathbf{A}}\mathbf{C} = \mathbf{F}$ and $\{\underline{\mathbf{A}}\mathbf{D}_i = \mathbf{N}_i\}_{i \in \{0,\ldots,N_I-1\}}$. It is worth noting that these systems share the same matrix $\underline{\mathbf{A}} := [a(\phi_j^t, \phi_i^t)]_{i,j \in \{0,\ldots,D_{K_t}-1\}}$, albeit with different load vectors. This feature means that $\underline{\mathbf{A}}$ needs to be factorized only once (for linear problems), and therefore the increase in the value of degree l or m in the approximation space $\Lambda_H^{m,l}$ only increases the number of matrix-vector multiplications, which can be trivially parallelized even within the context of a single local problem by means of vectorized computations. Moreover, if the coefficients associated with the operator $\mathcal{L}$ are not time-dependent, $\underline{\mathbf{A}}$ needs to be assembled and factorized only during the first timestep in a time marching scheme.

The REDUCELOCALPROBLEMS primitive gathers the contributions $\{\eta_t^i\}_{i \in \{0,\ldots,N_I-1\}}$ and η_t^f from the subproblems to compute u_0 and λ_H from Equation (1). The underlying linear system associated to (1) reads:

$$\begin{pmatrix} \underline{\mathbf{A}} & \underline{\mathbf{B}}^{\mathrm{T}} \\ \underline{\mathbf{B}} & \underline{\mathbf{0}} \end{pmatrix} \begin{pmatrix} \mathbf{L} \\ \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{E} \\ \mathbf{F} \end{pmatrix}, \tag{6}$$

with:

$$\underline{\mathbf{A}} := [(\psi_j, \eta_t^i)]_{i,j\in\{0,\ldots,L_l-1\}},$$
$$\underline{\mathbf{B}} := [(\psi_j, \upsilon_i)]_{i\in\{0,\ldots,N_{V_0}-1\},j\in\{0,\ldots,L_l-1\}},$$
$$\mathbf{E} := [-(\psi_j, \eta_t^f)]_{j\in\{0,\ldots,L_l-1\}},$$
$$\mathbf{F} := [(\upsilon_j, f)]_{j\in\{0,\ldots,N_{V_0}-1\}}.$$

The computed entries $[k_j]_{j\in\{0,\ldots,L_l-1\}}$ in $\mathbf{L}$ define the solution to λ_H in $\Lambda_H^{m,l}$; likewise, the computed entries $[b_j]_{j\in\{0,\ldots,N_{V_0}-1\}}$ in $\mathbf{P}$ define the solution u_0 in V_0 through

$$u_0 = \sum_{j=0}^{N_{V_0}-1} b_j \upsilon_j. \tag{7}$$

Finally, owing to the COMPUTESOLUTION primitive, the two-level numerical solution $u_{H,h}$ reads

$$u_{H,h} = \sum_{j=0}^{N_{V_0}-1} b_j \upsilon_j + \sum_{K_t\in\mathcal{T}_H} \sum_{j=1}^{D_{K_t}-1} \Big[\sum_{i=0}^{N_I-1} k_i d_{i,j}^t \phi_j^t + c_j^t \phi_j^t \Big]. \tag{8}$$

where we used (7) and (5) in (4).

4 IMPLEMENTATION DETAILS

The MHM software library allows for the implementation of hybrid parallel applications through the MPI and OpenMP standards. It explores the C++ STL library for implementing dynamic (`std::vector`) and associative (`std::map`) array-based data structures as well as iterators over them, and also for devising searching and sorting algorithms; the aim was to combine both efficiency and safety in the code. Besides, the MHM software library employs the template-based Eigen library[1] for common linear algebra operations.

The R MPI processes that make part of a MHM application are omnipotent; each MPI process $r \in R$ is responsible for running its own *share* of the SPLITPROBLEM primitive. This share corresponds to the execution of SPLITPROBLEM over a partition $\mathcal{T}_H^r$ of $\mathcal{T}_H$. Such a partitioning configures a static process scheduling that is kept until the end of the simulation. The software library offers two partitioning strategies: one based on a round-robin distribution of the elements of $\mathcal{T}_H$ between the MPI processes, and another based on the multilevel recursive bisection algorithm implemented in the METIS package.[2]

The result $P_{\mathcal{T}_H^r}$ of the SPLITPROBLEM primitive at MPI process $r \in R$ is taken by this process as a set of local problems to be solved—one for each $p_t \in P_{\mathcal{T}_H^r}$. From this point on the MPI processes start to execute the SOLVELOCALPROBLEM primitive for all p_t they are responsible for. We employed the Pardiso package[3] within this primitive for solving the

[1] `http://eigen.tuxfamily.org`
[2] `http://glaros.dtc.umn.edu/gkhome/views/metis`
[3] `http://www.pardiso-project.org`

linear systems that compute c_j^t and $\{d_{i,j}^t\}_{i\in\{0,\ldots,N_I-1\}}$ in Equation (5). The MHM software library allows the use of either the LDL^T or LU factorization methods provided within Pardiso, depending on whether the resulting matrix in the local problem is positive-definite or not. (For the elasticity problem used in our performance evaluation, LU factorization must be used.) In principle, within each MPI process its set of local problems is scheduled by an OpenMP loop so that each local problem is solved in a single core, with the intention to make as many local problems be solved in parallel in a single computing resource as the amount of cores the resource provides.

An MPI barrier is used for making all MPI processes wait for each other as they finish solving all their corresponding local problems. Once all local problems are solved, each MPI process $r \in R$ is responsible for running its own share of the REDUCELOCALPROBLEMS primitive. A share of the REDUCELOCALPROBLEMS primitive corresponds to the distributed assembly of the global linear system shown in Equation (6), and its solution through LDL^T factorization using the parallel sparse direct MUMPS solver.[4] The result from the REDUCELOCALPROBLEMS primitive triggers at each MPI process the immediate execution of its own share of the COMPUTESOLUTION primitive. After that, the application finishes.

5 PERFORMANCE EVALUATION

We collected measurements by running a parallel simulator implemented with the MHM software library in the Santos Dumont HPC cluster at LNCC in Brazil, which has the following per-node configuration: 2x CPU Intel Xeon E5-2695v2 Ivy Bridge (12 cores each CPU), 2.4GHZ, 64GB DDR3 RAM. The nodes of the cluster are interconnected through an Infiniband FDR network with a fat-tree topology, and they share a distributed file system based on Lustre v2.1. The C++ code was compiled with Intel® icpc compiler, version 17.0.1 build 20161005, with the optimization flag '`-O2`' enabled. We used Eigen version 3.3-rc1, Intel® MKL Pardiso that comes with Intel® Parallel Studio XE 2017.1.132, and MUMPS version 5.0.2. The MPI implementation is the OpenMPI version 2.0.1.

We solved the following elastostatic equation defined in the unit cube Ω with a boundary $\partial\Omega$ as a model problem: to find the displacement $\boldsymbol{u} : \Omega \to \mathbb{R}^d$ such that

$$\begin{cases} -\mathbf{div}\mathcal{A}\,\mathcal{E}(\boldsymbol{u}) = \boldsymbol{f} \quad \text{in } \Omega, \\ \qquad \boldsymbol{u} = \mathbf{0} \quad \text{on } \partial\Omega, \end{cases} \tag{9}$$

where $\boldsymbol{f}$ is a given function with values in $\mathbb{R}^d$, $d \in \{2, 3\}$. The linearized strain tensor is given by the symmetric part of the gradient

$$\mathcal{E}(\boldsymbol{u}) := \frac{1}{2}\big(\nabla\boldsymbol{u} + \nabla^T\boldsymbol{u}\big).$$

and the rigidity tensor $\mathcal{A}$ acts on the space $\mathbb{R}^{d\times d}_{\text{sym}}$ of symmetric matrices. In this paper we considered the homogeneous isotropic elastic domain case so that $\mathcal{A}$ can be determined by Lamé coefficients (μ, λ); for the model problem we set $\mu = 1$ and $\lambda = 1$. Besides, we also set

[4] `http://mumps.enseeiht.fr`

$\boldsymbol{f}$ such that the exact solution $\boldsymbol{u} = [u_1, u_2, u_3]^T$ to the model problem is given by

$$u_1(x, y, z) = \sin(2\pi x)\sin(2\pi y)\sin(2\pi z),$$
$$u_2(x, y, z) = 2\sin(2\pi x)\sin(2\pi y)\sin(2\pi z),$$
$$u_3(x, y, z) = 3\sin(2\pi x)\sin(2\pi y)\sin(2\pi z).$$

The weak formulations derived from the MHM method for the model problem can be briefly stated in terms of Equations (1), (2), and (3), with:

- V_0 corresponding to the space of the local rigid body modes in each $K_t \in \mathcal{T}_H$,
- $V_h(K_t)$ corresponding to the space $[H^1(K_t)]^d$ whose functions are L^2 orthogonal with respect to the local rigid body modes within K_t, and
- $a(\boldsymbol{u}_h, \boldsymbol{w}_h)_{K_t} := \int_{K_t} \mathcal{A}\,\mathcal{E}(\boldsymbol{u}_h) : \mathcal{E}(\boldsymbol{w}_h)$

For the sake of illustration, in Fig. 4 we depict one of the simplest scenarios we have experimented with: a 192-tetrahedra global mesh along with its submeshes (59 tetrahedra per element) as well as the corresponding numerical solution to our model problem.

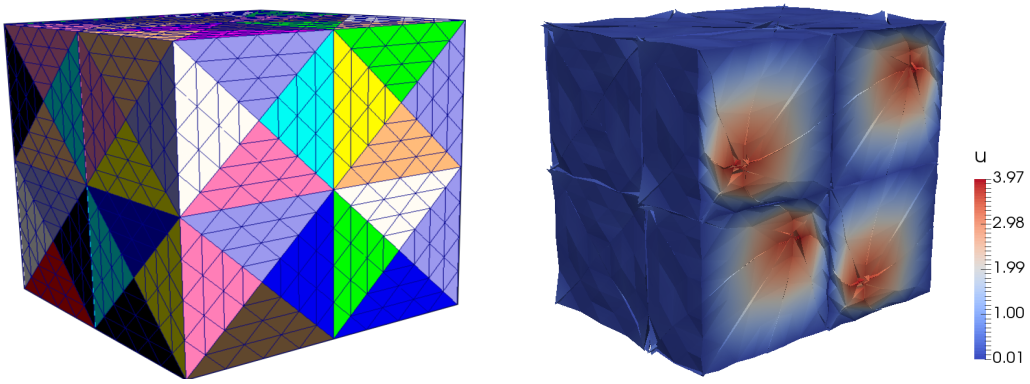


Figure 4: Mesh (left) and the isosurfaces of $|\boldsymbol{u}_{Hh}|$ at $x = 0.75$ (right).

Our simulation scenarios comprised global meshes varying from 24 to $98,304$ tetrahedra (i.e. with $H = \{1, 0.5, 0.25, 0.125, 0.0625\}$). In each of these scenarios, we explored a large set of MHM configurations over our model problem. Firstly, we fixed the characteristic length h of $\mathcal{T}_h^K$ and the polynomial degrees k and l, and varied the coarsening level H of the global mesh and the face partitioning level m. The idea was to characterize, from the point of view of computational performance, to what extent it is is better to coarsen the global mesh and refine the face partitioning—with the consequent refinement of the submeshes—or vice-versa, while capturing the same level of multiscale features (as determined by h and k). These experiments considered *minimal* submeshes. A minimal submesh is the coarsest submesh that respects the adopted partitioning of the faces in the global mesh, as determined by m in $\mathbf{\Lambda}_H^{m,l}$.

Fig. 5 shows the execution times with $H = \{0.5, 0.25, 0.125, 0.0625\}$ and $m = \{1, 4, 16, 64\}$, for $h = 0.0625$, $k = 3$ and $l = 1$, using an increasing number of cores. We can observe that a balanced approach, with only a few face partitions and a modest level of global mesh refinement, leads to smaller execution times.

Next, we fixed H and varied m, k and l (but respecting the constraint $k = l + 2$). The idea was to characterize, again from the point of view of computational performance, to what

extent it is better to refine the face partitioning and use polynomial degrees of lower order or vice-versa, while keeping the same approximation error. (In this case, of about 10^{-3} in the H^1-broken norm.) Fig. 6 shows the execution times for $H = 0.125$, $m = \{1, 4\}$, $k = \{4, 5\}$ and $l = \{2, 3\}$, using an increasing number of cores. We can observe that increasing the polynomial degree l (with a corresponding increase in the polynomial degree of k) instead of increasing the face partitioning m leads to smaller execution times.

Based on the previous results, we defined a set of MHM configurations over which we collected strong and weak scaling measurements. In Tab. 1, we summarize those configurations that led to approximations with errors below 1% in the H1-broken norm. The configurations with a '$*$' have minimal submeshes. The remainder configurations in the table have more refined submeshes, to account for richer levels of multiscale features.

Fig. 7 shows the strong scaling efficiency of all configurations in Tab. 1. For all config-

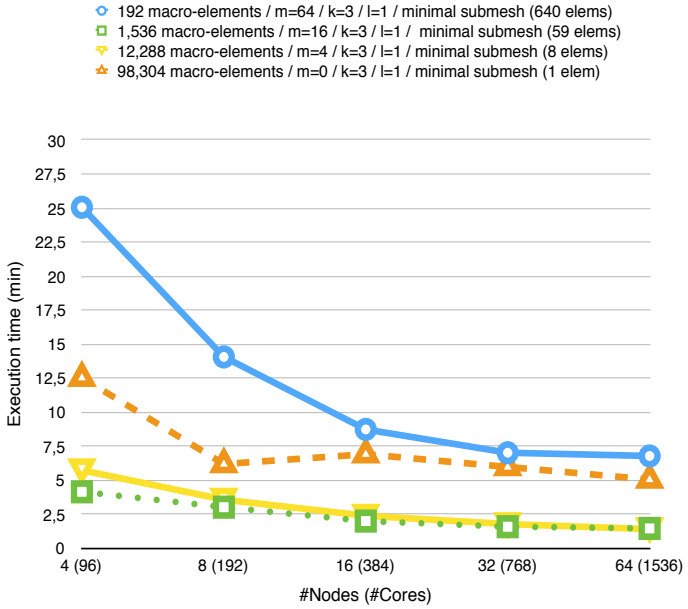


Figure 5: Execution times for MHM simulator using $h = 0.0625$ and $k = 3$ and $l = 1$.

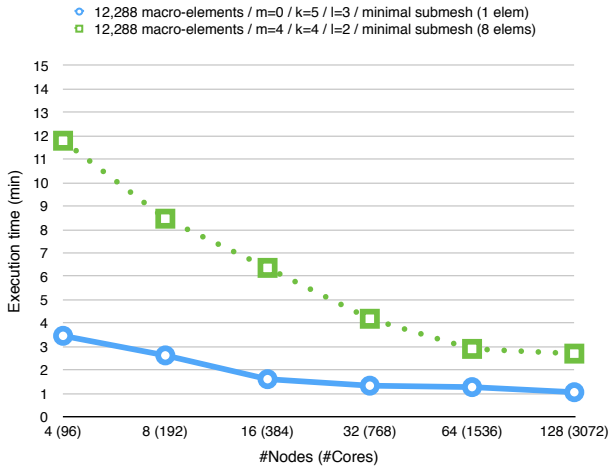


Figure 6: Execution times for MHM simulator using $H = 0.125$.

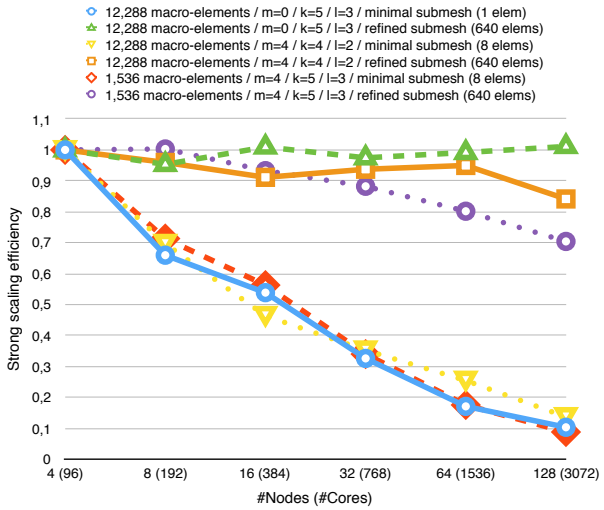


Figure 7: Strong scaling measurements.

urations with minimal submesh sizes the strong scaling quickly degrades, running at $< 50\%$ efficiency on more than 384 cores. For the configurations with refined submeshes the results are much better, all of them running at $> 70\%$ efficiency on $3,072$ cores (currently, the maximum allowed on the Santos Dumont HPC cluster). This is an expected result since the refinement of the submeshes—which is crucial to capture multiscale features—does not affect the size of the global system; the size of the local problems, however, does increase with refined submeshes and so does the relative importance of the computation of the local problems (which are embarrassingly parallel) to the total execution time of the MHM simulator.

For the weak scaling efficiency, we fixed the face partitioning and the polynomial degrees to the values of the scenario with the best strong scaling efficiency (i.e. $m = 1$, $k = 5$, $l = 3$), and varied the level of the refinement of the submeshes. We then increased the size of the problem by refining the global mesh (192, $1,536$ and $12,288$ elements) at the same rate as the increase in the number of cores. The graph on Fig. 8 shows the collected weak scaling measurements. The configuration with minimal submesh quickly degrades, whereas for refined submeshes the configurations run at $> 80\%$ efficiency on $3,072$ cores. As a final remark on the aforementioned scaling measurements, it can be noticed in Tab. 1 that the refinement of the submeshes, having fixed all the other parameters, does not reduce the approximation errors; rather, they stay at the same order. This is a consequence of the adopted model problem, whose physical coefficients

Table 1: MHM configurations with $\frac{\|u-u_{Hh}\|_1}{\|u\|_1} < 10^{-2}$

Coarse mesh	m in $\mathbf{\Lambda}_H^{m,l}$	l in $\mathbf{\Lambda}_H^{m,l}$	Submesh	k in $\mathbf{V}_h$	Total DoFs	$\|\|u - u_{Hh}\|\|_1$
$1,356$	4	3	8^*	5	$1,055,232$	4.51×10^{-4}
$1,356$	4	3	640	5	$70,838,784$	9.83×10^{-4}
$12,288$	1	3	1^*	5	$1,286,784$	8.37×10^{-4}
$12,288$	1	3	640	5	$563,610,240$	1.91×10^{-3}
$12,288$	4	2	8^*	4	$3,672,576$	1.00×10^{-3}
$12,288$	4	2	640	4	$296,741,376$	2.39×10^{-3}

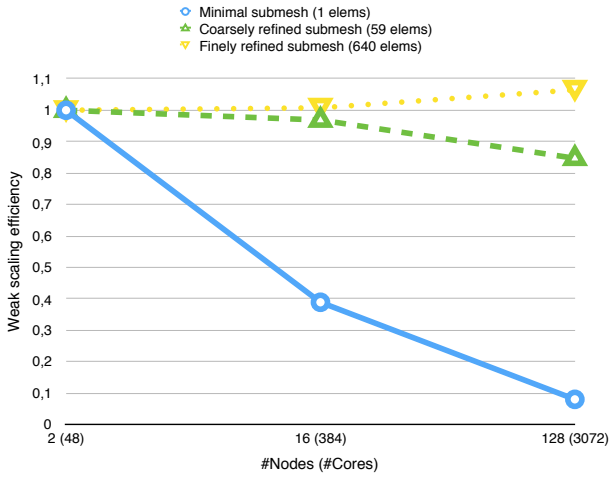


Figure 8: Weak scaling measurements.

are constant in the domain. For more realistic scenarios, with highly-heterogeneous coefficients, the refinement of the submeshes is expected to reduce the approximation errors up to the point in which further refinements do not capture additional multiscale features.

6 CONCLUSIONS

In this paper we presented our ongoing work on the implementation of scalable simulators specifically crafted for the MHM methods. We presented a preliminary evaluation of our software library, using an elastostatic application as the test case, and the collected results are promising with regard to its scalability.

The work described herein offers many opportunities for future work. Firstly, the test case employed in Section 5 makes use of constant physical coefficients, which allowed us to easily measure the approximation errors for the sake of better delimiting the scalability scenarios. For real-world cases (e.g. in geosysmic applications) high-contrast coefficients are the norm though. The MHM methods allow for different face partitioning levels per face of the coarse mesh, thus providing the adequate mathematical framework for dealing with precisely such sort of cases. As a consequence, however, the computational effort from the SOLVELOCALPROBLEM primitive instances to solve each local problem is not roughly the same anymore; METIS does allow us to better share this computational effort between different MPI processes, but does not solve the problem of imbalanced OpenMP computations within each MPI process. We are currently studying the use of the new *task* directives that have been introduced in version 4 of the OpenMP standard to cope with this issue. Along the same lines, we are investigating the use of coprocessors (starting with GPGPUs) to reduce the time taken by the execution of the large amount of SOLVELOCALPROBLEM primitive instances in MHM simulations. In this sense, a hybrid MPI-OpenMP-CUDA approach has already been explored by (Jacobsen et al. 2013) and the results did not show a significant advantage in performance over a hybrid MPI-CUDA approach for the specific problem tackled therein (Navier–Stokes equations solved through a multigrid method), even though the authors speculate that with higher GPU counts per node or with different domain decomposition strategies a hybrid MPI-OpenMP-CUDA approach may exhibit higher efficiency.

Secondly, as we have previously mentioned in this paper, the global problem is a key scalability bottleneck in the MHM methods. We have experimented with Intel® Parallel Direct Sparse Solver for Clusters[5] and PaStiX[6] to solve Equation (6) in the REDUCELOCALPROBLEMS primitive, but MUMPS offered the best compromise between memory footprint and computational time. In the future we intend to experiment with other parallel sparse linear solvers; in particular, the hybrid approaches presented in (Agullo et al. 2013) and (Yamazaki et al. 2013) combine direct and iterative methods and seem fit to solve linear systems similar to the one illustrated in Equation (6).

ACKNOWLEDGEMENT

These research results have received funding from the ATOS/Bull company and the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement no. 689772. The simulations presented herein used the HPC resources provided by LNCC's SDumont supercomputer (http://sdumont.lncc.br).

REFERENCES

Agullo, E., L. Giraud, A. Guermouche, A. Haidar, and J. Roman (2013). "Parallel algebraic domain decomposition solver for the solution of augmented systems". In: *Advances in Engineering Software* 60–61. CIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing, pp. 23–30. ISSN: 0965-9978. DOI: http://dx.doi.org/10.1016/j.advengsoft.2012.07.004. URL: http://www.sciencedirect.com/science/article/pii/S0965997812000981.

Araya, R., C. Harder, D. Paredes, and F. Valentin (2013). "Multiscale Hybrid-Mixed Method". In: *SIAM J. Numer. Anal.* 51.6, pp. 3505–3531. DOI: http://dx.doi.org/10.1137/120888223.

Araya, R., C. Harder, A. Poza, and F. Valentin (2016). *Multiscale Hybrid-Mixed Method for the Stokes and Brinkman Equations – The Method*. HAL 01347517.

Efendiev, Y., R. Lazarov, M. Moon, and K. Shi (2015). "A spectral multiscale hybridizable discontinuous Galerkin method for second order elliptic problems". In: *Computer Methods in Applied Mechanics and Engineering* 292. Special Issue on Advances in Simulations of Subsurface Flow and Transport (Honoring Professor Mary F. Wheeler), pp. 243–256. ISSN: 0045-7825. DOI: http://dx.doi.org/10.1016/j.cma.2014.09.036. URL: http://www.sciencedirect.com/science/article/pii/S0045782514003594.

Harder, C., A. L. Madureira, and F. Valentin (2016). "A Hybrid-Mixed Method for Elasticity". In: *ESAIM: Math. Model. Num. Anal.* 50.2, pp. 311–336. DOI: http://dx.doi.org/10.1051/m2an/2015046.

Harder, C., D. Paredes, and F. Valentin (2013). "A family of Multiscale Hybrid-Mixed finite element methods for the Darcy equation with rough coefficients". In: *Journal of Computational Physics* 245, pp. 107–130. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/j.jcp.2013.03.019. URL: http://www.sciencedirect.com/science/article/pii/S0021999113001976.

[5] https://software.intel.com/mkl-developer-reference-c-cluster-sparse-solver
[6] http://pastix.gforge.inria.fr

Harder, C., D. Paredes, and F. Valentin (2015). "On a Multiscale Hybrid-Mixed Method for Advective-Reactive Dominated Problems with Heterogeneous Coefficients". In: *Multiscale Modeling & Simulation* 13.2, pp. 491–518. DOI: 10.1137/130938499. URL: http://dx.doi.org/10.1137/130938499.

Hou, T. Y. and X.-H. Wu (1997). "A Multiscale Finite Element Method for Elliptic Problems in Composite Materials and Porous Media". In: *Journal of Computational Physics* 134.1, pp. 169–189. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1997.5682. URL: http://www.sciencedirect.com/science/article/pii/S0021999197956825.

Jacobsen, Dana A. and Inanc Senocak (2013). "Multi-level parallelism for incompressible flow computations on GPU clusters". In: *Parallel Computing* 39.1, pp. 1–20. ISSN: 0167-8191. DOI: http://dx.doi.org/10.1016/j.parco.2012.10.002. URL: http://www.sciencedirect.com/science/article/pii/S0167819112000804.

Yamazaki, I., X. S. Li, F.-H. Rouet, and B. Ucar (2013). "On Partitioning and Reordering Problems in a Hierarchically Parallel Hybrid Linear Solver". In: *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum* 00.undefined, pp. 1391–1400. DOI: doi.ieeecomputersociety.org/10.1109/IPDPSW.2013.170.