Parallel meshing, discretization and computation of flow in massive discrete fracture networks

PARALLEL MESHING, DISCRETIZATION AND COMPUTATION OF FLOW IN MASSIVE DISCRETE FRACTURE NETWORKS *

S. BERRONE†, S. SCIALÒ†, AND F. VICINI†

June 6, 2019

Abstract. In the present work an MPI parallel implementation of an optimization-based approach for the simulation of underground flows in large discrete fracture networks is proposed. The software is capable of parallel execution of meshing, discretization, resolution and post-processing of the solution. We describe how optimal scalability performances are achieved combining high efficiency in computations to an optimized use of MPI communication protocols. Also, a novel graph-topology for communications, called *multi-Master* approach, is tested, allowing for high scalability performances on massive fracture networks. Strong scalability and weak scalability simulations on random networks counting order of 10^5 fractures are reported.

Key words. Parallel scalability, MPI, Discrete Fracture Networks, Single-phase flows, PDE constrained optimization

AMS subject classifications. 65N30, 65N15, 65N50, 65J15, 68U20, 68W10, 68W40, 86-08

1. Introduction.

It is well known that underground flow simulations are particularly demanding from a computational point of view, mainly as a consequence of the size and geometrical complexity of computational domains of interest for practical applications. Moreover, due to the lack of direct measurements of subsoil characteristics, input data for simulations are typically given as probability distributions, thus demanding a large number of costly computations to derive reliable statistics on quantities of interest [17, 21, 24, 7].

The subsoil can be regarded as a porous material crossed by a network of intersecting fractures. Among the different models to describe the subsoil (see e.g. [40]), Discrete Fracture Network (DFN) models only represent underground fractures, neglecting the contribution of the surrounding rock matrix, such that flow can only occur through fractures and fracture intersections. Differently from homogenization techniques, DFN models explicitly represent the fractures and are thus capable of reproducing the topology of the fracture network and preferential pathways for the flow, thus being particularly suitable for the simulation of dispersion phenomena [37, 32, 41, 10]. Fractures in a DFN are modeled as intersecting planar polygons forming an extremely challenging computational domain, usually characterized by an intricate system of intersections. Further, DFNs present geometrical features at different scales, as e.g. small fractures intersecting with large faults, fractures intersecting with narrow angles or co-existence of very small and very large fracture intersections. This geometrical complexity and multi-scale nature poses severe constraints on the meshing strategies. Standard discretization techniques, based on the Finite Element Method (FEM), require a mesh conforming to the intersections between fractures in order to correctly enforce matching conditions at the interfaces. Meshes conforming to all the geometrical features in a DFN can not be generated for networks of practical interest, or would introduce such a large number of elements to make the resolution

*This work was supported by the CINECA-ISCRA project IsC58, HP10CDFLWH, by the INdAM-GNCS project "Analisi e sviluppo di metodi numerici su griglie poligonali/poliedriche per simulazioni in domini con geometrie complesse ed elevata eterogeneita" (2018) and by the MIUR project "Dipartimenti di Eccellenza 2018-2022".

†Dipartimento di Scienze Matematiche, Politecnico di Torino, Torino, IT, ({stefano.berrone, stefano.scialo, fabio.vicini}@polito.it).

of the corresponding discrete problem unaffordable. Further, a single simulation for a given geometry is not sufficient for applications, as it is the case for time dependent simulations or for uncertainty quantification purposes.

Next to recent works proposing advanced tools to produce conforming meshes for discrete fracture networks, [31, 41, 26], a large number of different approaches has been presented, aimed at relaxing the geometrical constraints on the mesh, thus overcoming the complexity of DFN simulations. We will define as *partially conforming* those DFN meshes in which fracture intersections do not cross mesh elements, as in a conforming mesh, but there is no matching at the intersections among element vertexes of the mesh of different fractures (see [9, Figs. 1,2]); and as *non-conforming* those meshes in which elements are free to arbitrarily cross fracture intersections (see [9, Figs. 3,17]). In [29] a survey on several unconventional numerical schemes for stationary DFN flow simulations is presented. In [20, 42] the DFN flow problem is dimensionally reduced to a mono-dimensional problem, whereas in [44] mixed finite elements are used on meshes partially conforming at fracture intersections and mortaring is used to enforce the required matching conditions. Polygonal methods have also been suggested as a way to easily generate conforming or partially conforming meshes: in [2, 3, 28, 27] the Virtual Element Method (VEM) is used for fracture networks and in [4] for fracture/matrix coupling, whereas [1] proposes the use of mimetic finite differences and [18] of Hybrid High Order Methods for fracture/matrix flow coupling. To completely overcome mesh related complexities in DFN simulations some authors suggest the use of non-conforming meshes. In this context, the use of the eXtended Finite Element Method (XFEM) has been proposed by many authors, see [25], to allow for irregular solutions within mesh elements, along with other techniques based e.g. on the cutFEM [16] or on Lagrange multipliers [35] to enforce conditions at the interfaces for fracture/matrix coupling. In Embedded Discrete Fracture Matrix (EDFM) models [36, 38], only the fractures exceeding a certain threshold size are explicitly represented, the others being homogenized, and the fractures and the porous matrix are represented on different computational meshes, adding suitable fracture/matrix connections for the coupling. Different approaches aimed at reducing cost and complexity of DFN simulations are based on a graph representation of the network of fractures, as proposed by [46, 33], also using graph theory tools [30] or machine learning [51] to estimate the flow.

The present work focuses on a novel approach for flow computations in large DFNs, first proposed in [11] and based on numerical optimization. The method allows the use of non-conforming meshes and the coupling conditions at fracture intersections are enforced through the minimization of a properly designed cost functional. The functional expresses the error in the fulfillment of the interface conditions, and the solution is obtained as the minimum of this functional constrained by the equations describing the flow on each fracture [12, 14]. One of the key aspects of the method resides in the fact that the minimization process requires to iteratively solve local and almost independent linear systems, each defined on a fracture of the network, and these fracture-local problems only need to share information at the interfaces. This allows a natural parallel implementation, with a high scalability potential.

A parallel implementation of this optimization-based method is reported in [15]. The algorithm is based on the MPI protocol for the communications among parallel processes and uses a *Master*/*Slave* topology to handle communications. According to this, the computing processes (*Slaves*) can not directly share the required data, as all communications flow through the *Master* process. The mentioned reference shows how this topology can reduce the number of communications in large networks. The

efficiency of the parallel algorithm is tested on networks with a number of fractures of the order of 10^3, envisaging the use of an improved communication topology to handle larger networks. For this approach "a posteriori" error estimates have been derived in [5] and applied for adaptive mesh refinement to large scale DFNs in [6]. Another parallel code for flow simulations in DFNs is proposed in [39], based mixed-hybrid finite elements on conforming meshes. Due to the use of conforming meshes, the main focus is on the parallel mesh generation and assembly of the discrete matrices, thus differing from both the approach in [15] and the one described here.

The present work proposes a new implementation of the optimization-based approach for flow simulations in large DFNs. In this code, the meshing of the domain, the assembly and the resolution of the discrete problem, as well as other pre-resolution and post-resolution tasks are implemented allowing for parallel computing on distributed memory architectures using MPI, taking full advantage of the peculiarities of the optimization approach. An effective organization of linear algebra operations and the use of functions for MPI communications capable of reducing the overhead related to communications at each iteration of the resolution process, yield optimal scalability performances to the proposed code. Further a *multi-Master* topology is implemented with a hierarchy of *Master* processes in order to remove possible bottlenecks caused by the overloading of a single process handling the communication phases of a large number of parallel computing processes. This extends optimal scalability properties to a large range of numbers of parallel processes, thus allowing to effectively and efficiently solve problems on extremely large networks using a strong parallel approach. Numerical examples on networks counting order of 10^5 fractures and using up to 128 parallel processes are used to document the performances of the code.

The structure of the manuscript is as follows: Section 2 describes the optimization based approach at the basis of the proposed algorithm, which is presented in detail in the following Section 3. Parallel performances of the code are shown in Section 4, and finally some conclusions are reported in Section 5.

2. Optimization formulation. Let us briefly recall the optimization based formulation of the DFN problem; full details can be found in [14] and references therein mentioned. Let Ω denote a fracture network, $\Omega := \bigcup_{i=1,\dots,I} F_i$, with each of the F_i representing a planar polygon in the three dimensional space, resembling one of the fractures in the network. Fractures are surrounded by an impervious matrix, such that flow only occurs in the fractures and through fracture intersections. Fracture intersections, also called traces, are denoted by S_m, $m = 1, \dots, M$, and we assume that each trace is given by the intersection of exactly two fractures, i.e. $S_m = \bar{F}_i \cap \bar{F}_j$. There is a map between each trace index and the couple of fracture indexes, denoted by $I_S : [1, \dots, M] \mapsto [1, \dots, I]^2$ and defined by $I_S(m) = (i, j)$ with $i < j$ such that $\bar{F}_i \cap \bar{F}_j = S_m$. We also define $\underline{I}_S(m) := i$ and $\overline{I}_S(m) := j$. Let us further indicate by $\mathcal{S}$ the set of all the traces in the network and by $\mathcal{S}_i$ the set of traces on F_i. The boundary $\partial\Omega$ of Ω is split in a Dirichlet part $\Gamma_D \neq \emptyset$ and a Neumann part $\Gamma_N = \partial\Omega \setminus \Gamma_D$. Functions b^D and b^N prescribe Dirichlet and Neumann boundary conditions on Γ_D and Γ_N, respectively. The boundary of each fracture ∂F_i can be consequently split in a Dirichlet part $\Gamma_{iD} := \partial F_i \cap \Gamma_D$ and a Neumann part $\Gamma_{iN} := \partial F_i \cap \Gamma_N$ with boundary conditions given by functions $b_i^D := {b^D}_{|\Gamma_{iD}}$ and $b_i^N := {b^N}_{|\Gamma_{iN}}$, respectively.

2.1. The continuous problem. The distribution of the hydraulic head H in a DFN is governed by the Darcy's law, which can be stated in weak form as follows:

for $i = 1, \dots, I$, given the functional spaces

$$V_i = \mathrm{H}^1_0(F_i) = \left\{ v \in \mathrm{H}^1(F_i) : v_{|\Gamma_{iD}} = 0 \right\}$$

and

$$V_i^D = \mathrm{H}^1_{\mathrm{D}}(F_i) = \left\{ v \in \mathrm{H}^1(F_i) : v_{|\Gamma_{iD}} = b_i^D \right\},$$

find $H_i := H_{|F_i} \in V_i^D$ such that, $\forall v \in V_i$,

$$\int_{F_i} K_i \nabla H_i \nabla v \mathrm{d}\Omega = \int_{F_i} q_i v \mathrm{d}\Omega + \int_{\Gamma_{iN}} b_i^N v_{|\Gamma_{iN}} \mathrm{d}\gamma + \sum_{S_m \in \mathcal{S}_i} \int_{S_m} \left[\!\left[\frac{\partial H_i}{\partial \hat{\nu}^i_{S_m}} \right]\!\right]_{S_m} v_{|S_m} \mathrm{d}\gamma \tag{2.1}$$

and, $\forall m = 1, \dots, M$, with $(i, j) = I_S(m)$

$$H_{i|S_m} - H_{j|S_m} = 0, \tag{2.2}$$

$$\left[\!\left[\frac{\partial H_i}{\partial \hat{\nu}^i_{S_m}} \right]\!\right]_{S_m} + \left[\!\left[\frac{\partial H_j}{\partial \hat{\nu}^j_{S_m}} \right]\!\right]_{S_m} = 0, \tag{2.3}$$

where $\mathbf{K}_i$ is a uniformly positive definite tensor representing the fracture hydraulic conductivity, q_i is a source term and $\left[\!\left[\frac{\partial H_i}{\partial \hat{\nu}^i_{S_m}} \right]\!\right]_{S_m}$ is the jump of the co-normal derivative along the unit vector normal to S_m with a fixed direction on F_i. Equations (2.2)-(2.3) represent the matching conditions at fracture intersections, imposing the continuity of the solution and the balance of fluxes, respectively.

Let us introduce the the quantity

$$U_i^m := \left[\!\left[\frac{\partial H_i}{\partial \hat{\nu}^i_{S_m}} \right]\!\right]_{S_m} + \alpha H_{i|S_m}, \quad U_i^m \in \mathrm{H}^{-\frac{1}{2}}(S_m)$$

for $\alpha > 0$, and the quadratic functional J defined as:

$$J(H, U) = \sum_{m=1}^{M} \left\| H_{i|_{S_m}} - H_{j|_{S_m}} \right\|^2_{\mathrm{H}^{\frac{1}{2}}(S_m)} + \left\| U_i^{S_m} + U_j^{S_m} - \alpha \left(H_{i|_{S_m}} + H_{j|_{S_m}} \right) \right\|^2_{\mathrm{H}^{-\frac{1}{2}}(S_m)}, \tag{2.4}$$

being U the function given by the cartesian product of functions U_i^m, for $i = 1, \dots, I$ and $S_m \in \mathcal{S}_i$, $U \in [\mathrm{H}^{-\frac{1}{2}}(\mathcal{S})]^2$. Then problem (2.1)-(2.3) is equivalent to the following PDE-constrained optimization problem [12]:

$$\min_{U \in [\mathrm{H}^{-\frac{1}{2}}(\mathcal{S})]^2} \quad J(H, U) \tag{2.5}$$

$$\text{such that, } \forall i = 1, \dots, I, \ \forall v \in V_i :$$

$$\int_{F_i} K_i \nabla H_i \nabla v \, \mathrm{d}\Omega + \alpha \sum_{S_m \in \mathcal{S}_i} \int_{S_m} H_{i|S_m} v_{|S_m} \, \mathrm{d}\gamma =$$

$$\int_{F_i} q_i v \, \mathrm{d}\Omega + \int_{\Gamma_{iN}} b_i^N v_{|\Gamma_{iN}} \, \mathrm{d}\gamma + \sum_{S_m \in \mathcal{S}_i} \int_{S_m} U_i^m v_{|S_m} \mathrm{d}\gamma. \tag{2.6}$$

2.2. The discrete problem. Let us now introduce on each fracture F_i of the DFN Ω a triangulation $\mathcal{T}_i$ and also, on each trace of each fracture F_i, a discretization $\mathcal{T}_i^m$. We remark that each of these meshes can be defined independently of all the

others. Then, on the elements of $\mathcal{T}_i$ we define a finite element space of local piecewise basis functions, such that the discrete version of H_i on F_i can be defined as $h_i := \sum_{k=1}^{N_i} h_{i,k}\varphi_{i,k}$, $i = 1, \dots, I$, being N_i the number of Degrees Of Freedom (DOFs) of h_i. Similarly, on the mesh $\mathcal{T}_i^m$ of each trace $S_m \in \mathcal{S}_i$ we introduce a set of basis functions, such that the discrete version of function U_i^m is set as $u_i^m := \sum_{k=1}^{N_i^m} u_{i,k}^m \psi_{i,k}^m$, now N_i^m denoting the number of DOFs of u_i^m. In the following will use the same symbol h_i (or u_i^m) to denote both the discrete function and the vector of its DOFs. We then define vector $h \in \mathbb{R}^{N_F}$, $N_F = \sum_{i=1}^{I} N_i$ as $h := (h_1; \dots; h_I)$, $u_i \in \mathbb{R}^{N_i^S}$, $N_i^S = \sum_{S \in \mathcal{S}_i} N_i^m$, $u_i := (u_i^{m_1}; \dots; u_i^{m_{\#\mathcal{S}_i}})$, and $u \in \mathbb{R}^{N^S}$, $N^S = \sum_{i=1}^{I} N_i^S$, $u := (u^1_{\underline{I}_S(m)}; u^1_{\overline{I}_S(m)}; \dots; u^M_{\underline{I}_S(m)}; u^M_{\overline{I}_S(m)})$. All vectors are column vectors and, here and in the following, syntax $(v; w)$ denotes vertical concatenation of v and w.

The Darcy equation (2.6) can be discretized in a classical way by the finite element method on each fracture as:

$$A_i h_i = q_i + \mathcal{B}_i u_i, \quad i = 1, \dots, I \tag{2.7}$$

where $A_i \in \mathbb{R}^{N_i \times N_i}$ is defined by

$$(A_i)_{k\ell} = \int_{F_i} K_i \nabla\varphi_{i,k} \nabla\varphi_{i,\ell} \, \mathrm{d}\Omega + \alpha \sum_{m=1}^{M_i} \int_{S_m} \varphi_{i,k|S_m} \varphi_{i,\ell|S_m} \, \mathrm{d}\gamma,$$

matrix $\mathcal{B}_i \in \mathbb{R}^{N_i \times N_{\mathcal{S}_i}}$ collects the integrals of the product of basis functions $\{\varphi_{i,k|S_m}\}$, $k = 1, \dots, N_i$, with $\{\psi_{i,k}^m\}$, $k = 1, \dots, N_i^m$, and $q_i \in \mathbb{R}^{N_i}$ is the vector deriving from the discretization of forcing terms and boundary conditions. The block-diagonal matrix $\mathbb{A} = \mathrm{diag}(A_i)_{i=1,\dots,I} \in \mathbb{R}^{N^F \times N^F}$ is then formed from matrices A_i, whereas matrix $\mathcal{B} \in \mathbb{R}^{N^F \times N^S}$, is defined as $\mathcal{B} := (\mathcal{B}_1 R_1; \dots; \mathcal{B}_I R_I)$, i.e. is obtained collecting column-wise matrices $\mathcal{B}_i R_i \ \forall i = 1, \dots, I$, where matrix R_i acts extracting from u the DOFs corresponding to u_i; now equations (2.7) can be compactly re-written as

$$\mathbb{A} h = q + \mathcal{B} u \tag{2.8}$$

where q is obtained grouping vectors q_i columnwise.

The discrete functional is obtained replacing, in equation (2.4) functions H and U with their discrete versions and using $\mathrm{L}^2(S_m)$ norms in place of the $\mathrm{H}^{\frac{1}{2}}(S_m)$ and $\mathrm{H}^{-\frac{1}{2}}(S_m)$ norms:

$$J(h,u) = \frac{1}{2} \sum_{i=1}^{I} \sum_{S_m \in \mathcal{S}_i} \left(\int_{S_m} \Big(\sum_{k=1}^{N_i} h_{i,k}\varphi_{i,k|S_m} - \sum_{k=1}^{N_j} h_{j,k}\varphi_{j,k|S_m}\Big)^2 \mathrm{d}\gamma + \right.$$
$$\left. \int_{S_m} \Big(\sum_{k=1}^{N_i^m} u_{i,k}^m \psi_{i,k}^m + \sum_{k=1}^{N_j^m} u_{j,k}^m \psi_{j,k}^m - \alpha \sum_{k=1}^{N_i} h_{i,k}\varphi_{i,k|S_m} - \alpha \sum_{k=1}^{N_j} h_{j,k}\varphi_{j,k|S_m}\Big)^2 \mathrm{d}\gamma \right). \tag{2.9}$$

The functional (2.9) is re-written in compact algebraic form as follows:

$$J(h,u) := \frac{1}{2} \left(h^T G^h h - \alpha h^T B^h u - \alpha u^T B^u h + u^T G^u u \right) \tag{2.10}$$

where matrix $G^h \in \mathbb{R}^{N^F \times N^F}$ collects integrals on the traces of the product between functions $\{\varphi_{i,k|S_m}\}$, $k = 1, \dots, N_i$, $i = 1, \dots, I$; matrix $G^u \in \mathbb{R}^{N^S \times N^S}$ collects integrals of the product between functions $\{\psi_{i,k}^m\}$, $k = 1, \dots, N_i^m$, and matrices B^h and

B^u, $B^h = (B^u)^T \in \mathbb{R}^{N^F \times N^S}$ collect integrals of the product between basis functions of h and of u.

The discrete optimization problem is then stated as follows:

$$\begin{aligned} &\min \quad J(h,u) \\ &\text{s.t.} \quad \mathbb{A}h - \mathcal{B}\,u = q. \end{aligned} \tag{2.11}$$

Optimality conditions for this problem are given by the system of equations

$$\mathbb{A}h = q + \mathcal{B}\,u \tag{2.12}$$

$$\mathbb{A}^T p = G^h h - \alpha B^h u \tag{2.13}$$

$$0 = \mathcal{B}^T\, p + G^u u - \alpha B^u h = \nabla J(u), \tag{2.14}$$

which corresponds to the following saddle point problem:

$$\mathcal{A} = \begin{pmatrix} G^h & -\alpha B^u & \mathbb{A}^T \\ -\alpha B^h & G^u & \mathcal{B}^T \\ \mathbb{A} & \mathcal{B} & O \end{pmatrix}; \quad \mathcal{A} \begin{pmatrix} h \\ u \\ -p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ q \end{pmatrix} \tag{2.15}$$

PROPOSITION 2.1. *Matrix $\mathcal{A}$ in equation (2.15) is non singular, and the unique solution $(h^\star, u^\star, p^\star)$ of (2.15) is the minimizer of (2.11).*

The proof of Proposition 2.1 can be easily found as a particular case of the proof given in [13].

It is possible to obtain an unconstrained minimization problem equivalent to (2.11) by formally replacing $h = \mathbb{A}^{-1}(\mathcal{B}\,u + q)$ in (2.10). To this end, we set:

$$\begin{aligned} \hat{J}(u) = \frac{1}{2} u^T (&\mathcal{B}\,\mathbb{A}^{-T} G^h \mathbb{A}^{-1}\,\mathcal{B} + G^u - \alpha\,\mathcal{B}^T\,\mathbb{A}^{-T} B - \alpha B^T \mathbb{A}^{-1}\,\mathcal{B}) u \\ &+ q^T \mathbb{A}^{-T} (G^h \mathbb{A}^{-1}\,\mathcal{B} + G^u - \alpha B) u \\ &= \frac{1}{2} u^T \hat{G} u + \hat{q} u, \end{aligned} \tag{2.16}$$

where $B := B^h = (B^u)^T$, and $\hat{G}$ is symmetric positive definite, see [15] for more details. Then, solving problem (2.11) is equivalent to solve equation $\hat{G}u = -\hat{q}$, i.e. minimizing the unconstrained functional (2.16) via, e.g., a gradient scheme. What is of interest for the present work is that this does not require the direct computation of matrix $\hat{G}$, but only involves the resolution of fracture-local problems. Indeed, the application of a gradient scheme to solve $\hat{G}u = -\hat{q}$ corresponds to the following algorithm: starting from a tentative u^0, solve in cascade, for $n = 0, 1, \ldots$:

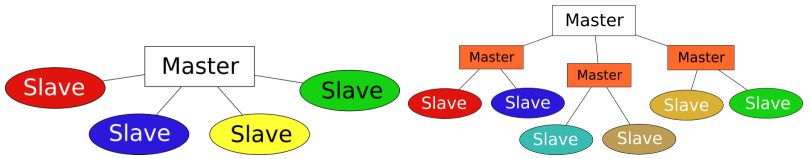
$$\mathbb{A}h = \mathcal{B}\,u^n \tag{2.17}$$

$$\mathbb{A}^T p = G^h h - \alpha B^h u^n \tag{2.18}$$

$$g^n = \mathcal{B}^T\, p + G^u u - \alpha B^u h \tag{2.19}$$

$$u^{n+1} = u^n + \lambda^n g^n \tag{2.20}$$

where g^n is the gradient direction $\nabla \hat{J}(u^n)$ at iteration n and λ^n is computed through an exact line search, [15]. Given the block-diagonal structure of matrix $\mathbb{A}$, problems (2.17)-(2.18) can be solved independently on each fracture of the DFN. A conjugate gradient scheme will actually be used to solve the optimization problem, and thus a



(a) Master-Slave Topology (b) Tree Topology

Figure 1: MPI Topologies

conjugate direction d^n is introduced, computed from g^n as $d^n = -g^n + \beta^n d^{n-1}$, for $n = 1, \ldots,$ being β^n a scalar quantity ensuring the conjugacy condition $(d^n)^T \hat{G}\, d^{n-1} = 0$, see Algorithm 3.1 in Section 3.5 for more details.

The structure of the iterative solver and the sub-division of this problem into sub-problems is detailed in the next Section 3.

3. Parallel implementation. Efficiency in memory management and in the use of computational power is critical in DFN flow simulations for engineering applications, which typically involve large scale domains, random parameters and are time dependent. The proposed approach is strongly parallel to achieve the desired efficiency, allowing for high speed computations without compromising the accuracy and the reliability of the results.

This section is devoted to the detailed description of the implementation of the parallel algorithm for flow simulations in DFNs formulated as a PDE-constrained optimization problem. The parallel code is written in the C++ language and relies on the MPI protocol version 3 (see e.g. [50]) for communications on distributed memory architectures, and on the Eigen library [22] for linear algebra operations.

A key aspect of the program is the parallel environment topology. A *MasterSlave* topology has been choosen in place of a *PointToPoint* communication approach as also proposed in [15]. In a *MasterSlave* topology (Figure 1a), one of the parallel processes (the *Master*) is uniquely devoted to receive and send the data that the other processes (called computing processes or *Slaves*) need to share. Point-to-point communications would require every process to directly share data with a number of other processes, which depends on the connectivity of the network and on its partitioning. For example, when large faults are present in a DFN, entirely crossing the network, if point-to-point communications are implemented, almost all the processes might need to share data among each other, thus yielding a growth of the number of communications proportional to the square of the number of processes. On the other side, with a *MasterSlave* topology the number of communications always scales like the number of processes, independently of the geometry of the network.

The *MasterSlave* approach has the drawback of a possible overloading of the *Master* process when a large number of parallel processes is used, thus creating unwanted bottlenecks. For this reason a new tree-type topology is implemented, allowing to handle a hierarchy of *Master* processes, with multiple bottom-level *Master* processes up to a single top-level *Master* process, as sketched in Figure 1b, where one top-level and three bottom level *Master* processes are used. This configuration is called *Multi-Master* communication topology. Now, each bottom-level *Master* process manages the communication of a fraction of the computing processes and the highest level *Master* process handles the communications of the lower-lever *Masters*.

The whole code can be subdivided in seven different tasks, as follows: *1)* DFN geometry import; *2)* main connected component computation; *3)* main connected component distribution among parallel processes; *4)* mesh generation; *5)* discrete problem matrices assembly; *6)* discrete problem resolution; *7)* solution post-processing and export.

3.1. DFN geometry import. Typically a fracture network is described as a set of fractures, identified by the coordinates of their vertexes in the three-dimensional space. Since geometry is provided in a single file, no parallel implementation is used for this phase. Both textual and binary input files are supported. If a bounding box is specified, fractures are cut accordingly, storing information on the fracture edges lying on each face of the box. Boundary conditions and other properties of the fractures are imported at this phase. All the data on the fractures are then communicated to all the parallel processes.

3.2. Main connected component computation. Usually, fracture networks are stochastically generated, thus the imported "raw" geometrical data might contain fractures or networks of fractures disconnected from the main connected component. The main connected component is defined as the largest set of connected fractures with a non empty portion of Dirichlet boundary. The determination of the main connected component is performed in parallel and is based on the computation of a local adjacency matrix.

Each fracture of the raw imported network is labeled with an integer i ranging between one to the total number of imported fractures I_T. Then, fracture indexes are split among the $n_{\mathcal{P}}+1$ parallel processes $\mathcal{P}_k$, $k=0,\dots,n_{\mathcal{P}}$, and sets $\mathcal{I}_k$ of fracture indexes are assigned to processes $\mathcal{P}_k$, balancing the quantity $\mathcal{X}_k = \sum_{i\in\mathcal{I}_k} I_T - (i+1)$, such that $\mathcal{X}_k \sim \frac{I_T(I_T+1)}{2(n_{\mathcal{P}}+1)}$. Each process $\mathcal{P}_k$ builds a local adjacency matrix A_k^D by checking if each of the fractures F_i, $i\in\mathcal{I}_k$ has intersections with fractures F_j, $j = i+1,\dots,I_T$. Thus, balancing the quantity $\mathcal{X}_k$, $k=0,\dots,n_{\mathcal{P}}$ corresponds to balance the number of checks that each process has to do.

The computation of the main connected component is then performed in parallel. On each process $\mathcal{P}_k$ a local connected component is computed and stored in an array $c_k\in\mathbb{R}^{I_T}$, initialized as $c_k(i)=i$. At each position j, $j=1,\dots,I_T$ of c_k the fracture index ℓ is stored, defined as $\ell = \arg\min_{i\in\mathcal{I}_k} \bar{F}_i\cap\bar{F}_j \neq\emptyset$. The array c_k can be easily built starting from the local adjacency matrix A_k^D. Then, for each $i\in\mathcal{I}_k$, if $c_k(i)\neq i$, we set $j=c_k(i)$ and, recursively, $c_k(i)=c_k(j)$ and $j=c_k(i)$, while $c_k(i)\neq c_k(j)$. Subsequently an *MPI_ Allreduce* operation is used to compute the array c, defined at each position $i=1,\dots,I_T$, as $c(i)=\min_{k=0,\dots,n_{\mathcal{P}}} c_k(i)$. Finally, again, for each $i\in\mathcal{I}_k$, if $c(i)\neq i$, we set $j=c_k(i)$, and, recursively, $c_k(i)=c_k(j)$ and $j=c_k(i)$, while $c_k(i)\neq c_k(j)$. At the end of this operation the connected component is given taking the largest set of fractures Ω_ζ, where for $\zeta=1,\dots,I_T$, $\Omega_\zeta = \{F_i, i=1,\dots,I_T : c(i)=\zeta\}$. The DFN Ω is then set equal to the main connected component identified in such way.

3.3. DFN partitioning. Partitioning is aimed at minimizing the communications, balancing, at the same time, the computational load assigned to each computing process, and is performed using the METIS library [34]. In a *Multi-Master* topology, the tree of the processes is first built (see Figure 1b), thus defining a hierarchy of processes, with $n_{\mathcal{P}}^M > 0$ *Master* processes, organized in different levels, and $n_{\mathcal{P}}^S \geq 0$ *Slave* processes at the bottom level. If the number of *Slave* processes is zero the code is executed in serial. We assume that the *Master-Slave* tree is a completely balanced tree.

To each *Master*, the set $\mathcal{P}_k^{\downarrow}$ of processes is then assigned, which might contain either lower level *Masters*, either a set of *Slave* processes, such that $\mathcal{P}_k$, $k = 0, \dots, n_{\mathcal{P}}^M - 1$ manages the communications among all the processes in $\mathcal{P}_k^{\downarrow}$. The set $\mathcal{P}_k^{\uparrow}$, contains, instead, the unique process at higher level for process $\mathcal{P}_k$, $k = 1, \dots, n_{\mathcal{P}}$ (i.e. the higher level *Master*), and it is empty for $\mathcal{P}_0$.

On the *Master* process $\mathcal{P}_0$ the DFN is first split through the METIS balanced graph partitioning tool into a number of parts equal to the number of processes in $\mathcal{P}_0^{\downarrow}$, and then each part is assigned to a different process in $\mathcal{P}_0^{\downarrow}$. In the case of Figure 1b, $\mathcal{P}_0^{\downarrow}$ contains three *Master* processes, and then Ω is first split into three parts. The operation is repeated on each *Master* $\mathcal{P}_k \in \mathcal{P}_0^{\downarrow}$, splitting the assigned portion of the network into sub-networks among the processes in $\mathcal{P}_k^{\downarrow}$, i.e. at the lower level, until the level of the *Slave* processes is reached. In the example of Figure 1b, on each of the three lower level *Master* processes the DFN is split between two computing processes, although, in practical applications the number of *Slaves* is much larger than the number of *Masters*. This procedures ensures that the communications at each level of the topology graph are minimized, at the cost of multiple calls to the METIS library. However, this approach is not expensive compared to the whole resolution process and it is capable of optimizing the partitioning of the graph. Information on the connectivity among the various parts of the original DFN are stored during the partitioning process, to be used for the communication phases of the algorithm.

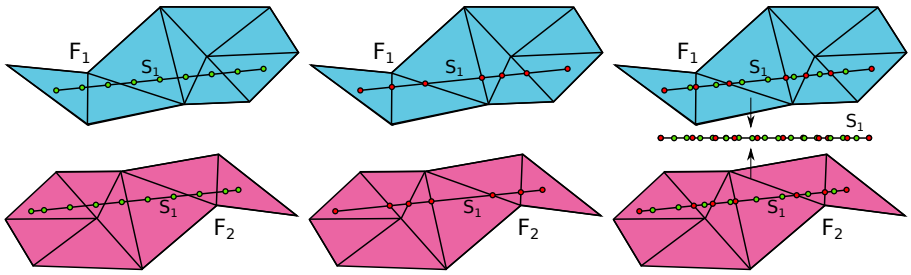
An estimate of the number of degrees of freedom per fracture and per trace is computed to evaluate the computational cost related to the resolution of linear systems on the fractures and the cost of communications related to each trace. These information can then be used by METIS in order to perform the balanced partitioning of the DFN.

At the end of the partitioning process a set of fracture indexes is assigned to each computing process $\mathcal{P}_k$, $k = n_{\mathcal{P}}^M, \dots, n_{\mathcal{P}}$, denoted by $\mathcal{I}_k^{\text{loc}}$. Fractures F_i, $i \in \mathcal{I}_k^{\text{loc}}$ are called process-local fractures for $\mathcal{P}_k$, whereas indexes of fractures that are not process-local, but that form at least one trace with a local fracture are stored in the set $\mathcal{I}_k^{\text{ext}}$, i.e. $j \in \mathcal{I}_k^{\text{ext}}$ if $j \notin \mathcal{I}_k^{\text{loc}}$ and $\bar{F}_j \cap \bar{F}_r \neq \emptyset$, for some $r \in \mathcal{I}_k^{\text{loc}}$. We then define, for each computing process $\mathcal{P}_k$ the set $\mathcal{I}_k^{\text{tot}} = \mathcal{I}_k^{\text{loc}} \cup \mathcal{I}_k^{\text{ext}}$. Further, process-local fractures are split between communicating fractures, whose indexes are collected in the set $\mathcal{I}_k^{\text{com}}$, that are fractures F_i, $i \in \mathcal{I}_k^{\text{loc}}$ that have at least a trace in common with a fracture F_j, $j \in \mathcal{I}_k^{\text{ext}}$, and non communicating fractures, whose indexes are collected in $\mathcal{I}_k^{\text{noc}}$, that are fractures F_i, $i \in \mathcal{I}_k^{\text{loc}}$ forming traces only with other fractures F_j, $j \in \mathcal{I}_k^{\text{loc}}$.

The indexes of the traces associated to each process are collected in the index set $\mathcal{M}_k^{\text{loc}}$, such that $m \in \mathcal{M}_k^{\text{loc}}$ if $S_m \in \mathcal{S}_i$ for some $i \in \mathcal{I}_k^{\text{loc}}$. Trace indexes in $\mathcal{M}_k^{\text{loc}}$ are split into two sets: non-communicating trace set, $\mathcal{M}_k^{\text{noc}}$, collecting indexes of traces formed by two process-local fractures, i.e. $m \in \mathcal{M}_k^{\text{noc}}$ if $I_S(m) \subset \mathcal{I}_k^{\text{loc}}$, and communicating trace set, $\mathcal{M}_k^{\text{com}} := \mathcal{M}_k^{\text{loc}} \setminus \mathcal{M}_k^{\text{noc}}$.

3.4. Mesh generation. The generation of the computational mesh is performed by each computing process $\mathcal{P}_k$, $k = n_{\mathcal{P}}^M, \dots, n_{\mathcal{P}}$ on the set of local fractures F_i, $i \in \mathcal{I}_k^{\text{loc}}$ and local traces S_m, $m \in \mathcal{M}_k^{\text{loc}}$. As already mentioned, the mesh for the hydraulic head h_i on each fracture F_i is independent from the mesh on the other fractures and from the position of fracture intersections. The triangulation on the fractures is obtained using the Triangle library [45].

On each trace S_m of each fracture F_i a mesh is built for the discretization of function u_i^m, which is, in general, independent from the mesh for the hydraulic head and from the mesh of the same trace on fracture F_j, $(i, j) = I_S(m)$. Thus, thanks to



(a) Discretization (b) Induced (c) Union and Interface

Figure 2: Example meshes for trace S_1 on F_1 and F_2

the proposed optimization approach, the meshing process can be performed in parallel and becomes a trivial task. Only a simple communication phase is required at the end of the meshing process to build a support mesh on each trace, for the evaluation of the integrals on the traces of basis functions (or restriction of basis functions) defined on the different meshes.

Concerning the mesh on the traces, we will distinguish among: *Discretization* mesh, *Induced* mesh, and *Union* mesh, see Figure 2. Each of these meshes is defined differently depending if the trace S_m is seen as an object of fracture F_i or F_j, $(i,j) = I_S(m)$, $m = 1,\dots,M$: in the following we will denote as S_m^- trace S_m as an object of F_i and as S_m^+ as an object of F_j, recalling that $i < j$. The Discretization mesh can be arbitrarily chosen, as for example a mesh of equally spaced nodes, as shown in Figure 2a. In this case the nodes on S_m^- and S_m^+ are staggered, i.e. shifted of a given quantity (typically half of the interval length), with the exception of the first and last node, which always coincide with the two extremes of the trace. As another example, the Discretization mesh might, instead, coincide with the Induced mesh. The Induced mesh is simply obtained taking as nodes the intersection points between the trace and the elements of the fracture mesh, Figure 2b. Finally the Union mesh is given by the union of the Discretization and Induced mesh, see Figure 2c. Another mesh is defined, called *Interface* mesh, which is unique for each trace, and is given on S_m by the union of the Union mesh of S_m^- and S_m^+, as depicted in Figure 2c. This mesh is only used as a support for integration purposes, and its construction requires the communication of the two Union meshes for each trace S_m, $m \in \mathcal{M}_k^{\text{com}}$. Values of basis functions $\{\varphi_{i,k|S_m}\}_{k=1,\dots,N_i}$ and $\{\psi_{i,k}^m\}_{k=1,\dots,N_i^m}$ $\forall i \in I_S(m)$, for $m = 1,\dots,M$ in each quadrature node of the Interface mesh are computed on each process and then shared through an MPI communication by both processes sharing the trace.

3.5. Discrete problem matrix assembly. As mentioned in Section 2.2, equations (2.17)-(2.19) can be split, at each iteration, into sub-problems which can be solved in parallel. Problem (2.17)-(2.19) can be written fracture-wise in the following way: for $i = 1,\dots,I$:

$$A_i[h_i]^n = \mathcal{B}_i[u_i]^n + q_i, \tag{3.1}$$

$$A_i^T[p_i]^n = G_i^h[h_i^+]^n - \alpha B_i^h[u_i^+]^n, \tag{3.2}$$

$$[g_i]^n = \mathcal{B}_i^T[p_i]^n + G_i^u[u_i^+]^n - \alpha B_i^u[h_i^+]^n \tag{3.3}$$

where the array h_i^+ is obtained appending column-wise to h_i vectors h_j, for $j = 1, \dots, I$ such that $j \neq i$ and $\bar{F}_i \cap \bar{F}_j \neq \emptyset$; u_i^+ is obtained in a similar way, grouping column-wise vectors u_i^m and u_j^m for $S_m \in \mathcal{S}_i$, $(i,j) = I_S(m)$. Matrices G_i^h and B_i^u are the sub-matrices obtained from G^h and B^u, respectively, extracting the rows relative to the position of the degrees of freedom of h_i in h, and the columns relative to the position of the DOFs of h_i^+ in h, and similarly matrices G_i^u and B_i^h, are obtained from G^u and B^h extracting the rows corresponding to the DOFs of u_i in u and columns corresponding to the DOFs of u_i^+ in u; finally g_i is the local gradient direction. The parallel implementation proposed in [15] is based on this fracture-wise splitting of the network.

For implementation purposes a different organization of computations appears more effective, with vectors and matrices assembled in blocks as explained in the following, capable of optimizing the performances of the library used for linear algebra computations and the efficiency of MPI communications. On each process $\mathcal{P}_k$, $k = n_{\mathcal{P}}^M, \dots, n_{\mathcal{P}}$, the following vectors are defined: vector $h_k^{\text{com}} \in \mathbb{R}^{N_k^{\text{com}}}$, grouping column-wise vectors h_i, $i \in \mathcal{I}_k^{\text{com}}$; vector $h_k^{\text{noc}} \in \mathbb{R}^{N_k^{\text{noc}}}$, grouping vectors h_i, $i \in \mathcal{I}_k^{\text{noc}}$; and vector $h_k^{\text{tot}} \in \mathbb{R}^{N_k^{\text{tot}}}$, obtained collecting all the vectors h_i for $i \in \mathcal{I}_k^{\text{tot}}$, such that h_k^{tot} contains h_k^{com}, h_k^{noc}, and finally vector h_k^{ext}, which is given grouping vectors h_i, $i \in \mathcal{I}_k^{\text{ext}}$. Other vectors are also assembled, named $u_k^{\text{com}} \in \mathbb{R}^{N_k^{S,\text{com}}}$, obtained grouping u_i^m and u_j^m for $m \in \mathcal{M}_k^{\text{com}}$ with $(i,j) = I_S(m)$, and vectors $u_k^{\text{loc}} \in \mathbb{R}^{N_k^{S,\text{loc}}}$, grouping vectors u_i^m and u_j^m for $m \in \mathcal{M}_k^{\text{loc}}$, $(i,j) = I_S(m)$. Matrix $A_k^{\text{com}} \in \mathbb{R}^{N_k^{\text{com}} \times N_k^{\text{com}}}$ is created as a block diagonal matrix with blocks formed by matrices A_i with $i \in \mathcal{I}_k^{\text{com}}$ and similarly matrix $A_k^{\text{noc}} \in \mathbb{R}^{N_k^{\text{noc}} \times N_k^{\text{noc}}}$ is a block diagonal matrix with blocks A_i, for $i \in \mathcal{I}_k^{\text{noc}}$. Matrix $\mathcal{B}_k^{\text{com}} \in \mathbb{R}^{N_k^{\text{com}} \times N_k^{S,loc}}$ is instead formed extracting from matrix $\mathcal{B}$ the rows corresponding to the position of the DOFs of h_k^{com} in h and the columns corresponding to the position of the DOFs of u_k^{loc} in u; analogously is obtained $\mathcal{B}_k^{\text{noc}} \in \mathbb{R}^{N_k^{\text{noc}} \times N_k^{S,loc}}$ from $\mathcal{B}$. Matrix $G_k^{h,\text{com}} \in \mathbb{R}^{N_k^{\text{com}} \times N_k^{\text{tot}}}$ is extracted from G^h taking the rows corresponding to the position of h_k^{com} in h and the columns corresponding to the position of h_k^{tot} in h, and similarly for $G_k^{h,\text{com}} \in \mathbb{R}^{N_k^{\text{noc}} \times N_k^{\text{tot}}}$. Finally matrix $B_k^{h,\text{com}} \in \mathbb{R}^{N_k^{\text{com}} \times N_k^{S,loc}}$ is created extracting rows from matrix B^h corresponding again to the position of h_k^{com} in h and to the columns corresponding to the position of u_k^{loc} in u, and similarly for $B_k^{h,\text{noc}} \in \mathbb{R}^{N_k^{\text{noc}} \times N_k^{S,loc}}$. Then, at iteration n, problem (2.17)-(2.18) is re-written, on each process $\mathcal{P}_k$, as:

$$A_k^{\text{com}}[h_k^{\text{com}}]^n = q_k^{\text{com}} + \mathcal{B}_k^{\text{com}}[u_k^{\text{loc}}]^n \tag{3.4}$$

$$A_k^{\text{noc}}[h_k^{\text{noc}}]^n = q_k^{\text{noc}} + \mathcal{B}_k^{\text{noc}}[u_k^{\text{loc}}]^n \tag{3.5}$$

$$(A_k^{\text{com}})^T[p_k^{\text{com}}]^n = G_k^{h,\text{com}}[h_k^{\text{tot}}]^n - \alpha B_k^{h,\text{com}}[u_k^{\text{loc}}]^n \tag{3.6}$$

$$(A_k^{\text{noc}})^T[p_k^{\text{noc}}]^n = G_k^{h,\text{noc}}[h_k^{\text{tot}}]^n - \alpha B_k^{h,\text{noc}}[u_k^{\text{loc}}]^n. \tag{3.7}$$

This implementation allows for an improved use of the memory, as it avoids multiple copies of the same data, with respect to a fracture-wise organization, in which arrays h_i^+ and u_i^+ would contain data already present in h_i and u_i, and, in general several copies of the same data would be present on each process.

Equations (3.4) and (3.5) are split since after the resolution of problems (3.4), the data in the array h_k^{out} are sent to the *Master* process by each *Slave* process. A non blocking MPI send operation is used, and then each *Slave* process can proceed with the resolution of problem (3.5), before receiving array h_k^{in} from the *Master*, which is required for the resolution of (3.6), where it appears into array h_k^{tot}. This allows to

Algorithm 3.1 Parallel CG algorithm

for each computing process $\mathcal{P}_k$, $k = n_{\mathcal{P}}^M, \dots, n_{\mathcal{P}}$ **do**
  Choose an initial guess $[u_k^{\text{loc}}]^0$
  Solve $A_k^{\text{com}}[h_k^{\text{com}}]^0 = q_k^{\text{com}} + \mathcal{B}_k^{\text{com}}[u_k^{\text{loc}}]^0$
  Communication: MPI Send $[h_k^{\text{out}}]^0$
  Solve $A_k^{\text{noc}}[h_k^{\text{noc}}]^0 = q_k^{\text{noc}} + \mathcal{B}_k^{\text{noc}}[u_k^{\text{loc}}]^0$
  Communication: MPI Recv $[h_k^{\text{in}}]^0$
  Solve $(A_k^{\text{com}})^T[p_k^{\text{com}}]^0 = G_k^{h,\text{com}}[h_k^{\text{tot}}]^0 - \alpha B_k^{h,\text{com}}[u_k^{\text{loc}}]^0$
  Solve $(A_k^{\text{noc}})^T[p_k^{\text{noc}}]^0 = G_k^{h,\text{noc}}[h_k^{\text{tot}}]^0 - \alpha B_k^{h,\text{noc}}[u_k^{\text{loc}}]^0$
  Compute $[g_k^{\text{loc}}]^0 = (\mathcal{B}_k^{\text{tot}})^T[p_k^{\text{tot}}]^0 + G_k^{u,\text{loc}}[u_k^{\text{loc}}]^0 - \alpha B_k^{u,\text{tot}}[h_k^{\text{tot}}]^0$
  Compute $[\beta_{N,k}]^0 = ([g_k^{\text{loc}}]^0)^T[g_k^{\text{loc}}]^0$
  Communication: MPI All-Reduce $[\beta_N]^0 = \sum_{k=1}^{n_{\mathcal{P}}}[\beta_{N,k}]^0$
  Set $[d_k^{\text{loc}}]^0 = -[g_k^{\text{loc}}]^0$
  Communication: MPI Send $[d_k^{\text{out}}]^0$ and Recv $[d_k^{\text{in}}]^0$
  Set $n = 0$
  while $[g_k^{\text{loc}}]^n \neq 0$ **do**
    Solve $A_k^{\text{com}}[\delta h_k^{\text{com}}]^n = \mathcal{B}_k^{\text{com}}[\delta d_k^{\text{loc}}]^n$
    Communication: MPI Send $[\delta h_k^{\text{out}}]^n$
    Solve $A_k^{\text{noc}}[\delta h_k^{\text{noc}}]^n = \mathcal{B}_k^{\text{noc}}[\delta d_k^{\text{loc}}]^n$
    Communication: MPI Recv $[\delta h_k^{\text{in}}]^n$
    Solve $(A_k^{\text{com}})^T[\delta p_k^{\text{com}}]^n = G_k^{h,\text{com}}[\delta h_k^{\text{tot}}]^n - \alpha B_k^{h,\text{com}}[d_k^{\text{loc}}]^n$
    $(A_k^{\text{noc}})^T[\delta p_k^{\text{noc}}]^n = G_k^{h,\text{noc}}[\delta h_k^{\text{tot}}]^n - \alpha B_k^{h,\text{noc}}[d_k^{\text{loc}}]^n$
    $[\delta g_k^{\text{loc}}]^n = (\mathcal{B}_k^{\text{tot}})^T[\delta p_k^{\text{tot}}]^n + G_k^{u,\text{loc}}[d_k^{\text{loc}}]^n - \alpha B_k^{u,\text{tot}}[\delta h_k^{\text{tot}}]^n$
    Compute $[\lambda_{N,k}]^n = ([d_k^{\text{loc}}]^n)^T[g_k^{\text{loc}}]^n$ and $[\lambda_{D,k}]^n = ([d_k^{\text{loc}}]^n)^T[\delta g_k^{\text{loc}}]^n$
    Communication: MPI All-Reduce $[\lambda_N]^n = \sum_{k=1}^{n_{\mathcal{P}}}[\lambda_{N,k}]^n$, $[\lambda_D]^n = \sum_{k=1}^{n_{\mathcal{P}}}[\lambda_{D,k}]^n$
    Compute $[\lambda]^n = \frac{[\lambda_N]^n}{[\lambda_D]^n}$
    Update $[g_k^{\text{loc}}]^{n+1} = [g_k^{\text{loc}}]^n + [\lambda]^n[\delta g_k^{\text{loc}}]^n$
    Set $[\beta_D]^{n+1} = [\beta_N]^n$ and compute $[\beta_{N,k}]^{n+1} = ([g_k^{\text{loc}}]^{k+1})^T[g_k^{\text{loc}}]^{k+1}$
    Communication: MPI All-Reduce $[\beta_N]^{n+1} = \sum_{k=1}^{n_{\mathcal{P}}}[\beta_{N,k}]^{n+1}$
    Compute $[\beta]^{n+1} = \frac{[\beta_N]^{n+1}}{[\beta_D]^{n+1}}$
    Update $[d_k^{\text{loc}}]^{n+1} = -[g_k^{\text{loc}}]^{n+1} + [\beta]^{n+1}[d_k^{\text{loc}}]^n$
    Communication: MPI Send $[d_k^{\text{out}}]^{n+1}$ and Recv $[d_k^{\text{in}}]^{n+1}$
    Set $n = n + 1$
  end while
end for

maximize the parallel performances of the algorithm, as communication time can be shadowed, on each *Slave* process, by the resolution of the linear system related to the non communicating fractures. Despite no communication occurs between equations (3.6) and (3.7) they are split to save memory, reusing matrices A_k^{com} and A_k^{noc}, or their factorizations. Finally equation (2.19) is written on each process $\mathcal{P}_k$, $k = 1, \dots, n_{\mathcal{P}}$ as:

$$g_k^{\text{loc}} = (\mathcal{B}_k^{\text{tot}})^T p_k^{\text{tot}} + G_k^{u,\text{loc}} u_k^{\text{loc}} - \alpha B_k^{u,\text{tot}} h_k^{\text{tot}} \tag{3.8}$$

where g_k^{loc} is the gradient direction computed for the local traces, and matrices $G_k^{u,\text{loc}}$

Algorithm 3.2 Communication phase

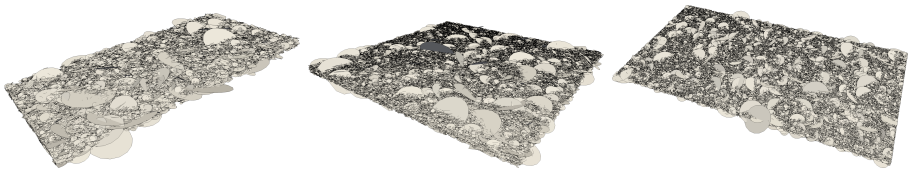
1: **for** $k = 0, \dots, n_{\mathcal{P}}^M - 1$ **do**
2: **if** $n = 0$ **then**
3: Initialize MPI indexed arrays h_k^{com} and u_k^{com}
4: Initialize the MPI_SEND_INIT and MPI_RCV_INIT buffers
5: **end if**
6: Receive $[h_\ell^{\text{out}}]^n$ and $[u_\ell^{\text{out}}]^n$ from processes $\mathcal{P}_\ell \in \mathcal{P}_k^{\downarrow}$
7: Assemble $[h_k^{\text{out}}]^n$ and $[u_k^{\text{out}}]^n$
8: Send $[h_k^{\text{out}}]^n$ and $[u_k^{\text{out}}]^n$ to process $\mathcal{P}_k^{\uparrow}$
9: Receive $[h_k^{\text{in}}]^n$ and $[u_k^{\text{in}}]^n$ from process $\mathcal{P}_k^{\uparrow}$
10: send $[h_\ell^{\text{in}}]^n$ and $[u_\ell^{\text{in}}]^n$ to processes $\mathcal{P}_\ell \in \mathcal{P}_k^{\downarrow}$
11: **end for**

and $B_k^{u,\text{tot}}$ are extracted from G^u and B^u, respectively, analogously to what described above.

Let us define, for each trace S_m on fracture F_i a set of *active* DOFs, i.e. the subset of DOFs of h_i which determines the value of $h_{i|S_m}$, i.e. the solution on the trace. Since coefficients of matrices G^h and B^h only depend from those basis functions whose support has a non empty intersection with a trace, for efficiency reasons, at each iteration of the scheme (3.4)-(3.7) only the set of active DOFs in arrays h_k^{tot} on each process $\mathcal{P}_k$ needs to be updated. More in details, $\mathcal{P}_k$ has to send to the *Master* a subset of the array h_k^{com}, denoted as h_k^{out}, containing the set of all active DOFs of its communicating fractures F_i, $i \in \mathcal{I}_k^{\text{com}}$, and, in turn, receives from the *Master* a subset of h_k^{ext}, denoted as h_k^{in}, containing the active DOFs of fractures F_i, $i \in \mathcal{I}_k^{\text{ext}}$. Analogously, concerning trace-variables, the part of u_k^{com} sent to the *Master* is denoted as u_k^{out} and the part received is denoted u_k^{in}, now u_k^{out} and u_k^{in} defining the whole u_k^{com} (actually, as shown in the next paragraph, the increment of u_k^{com} is communicated at each iteration and not u_k^{com} itself). The organization of arrays in blocks on each process, as here proposed, also yields efficient MPI communications. Using persistent MPI communication protocols (*MPI_SEND_INIT* and *MPI_RECV_INIT* instructions, see for example [48]) and the *MPI_Type_indexed* [49] for h_k^{tot}, and u_k^{loc} it is possible to define, at the first iteration, the indexes to the active DOFs in h_k^{com}, h_k^{ext} and u_k^{com}, allowing for an optimized indexing and a minimization of communication overhead. In contrast, a fracture-wise organization would require repeated indexing into fracture-local arrays, with a detrimental impact on the efficiency of MPI communications.

3.6. Problem resolution. The optimization problem (2.16) is solved via a conjugate gradient scheme whose parallel implementation is detailed Algorithm 3.1. The notation $\delta(\cdot)$ is introduced to denote the increment of a variable between two subsequent iterations of the method, as, e.g. $[\delta h_k^{\text{com}}]^n$ is the variation of h_k^{com} on process $\mathcal{P}_k$ between iterations n and $n+1$, i.e. $[\delta h_k^{\text{com}}]^n = [h_k^{\text{com}}]^{n+1} - [h_k^{\text{com}}]^n$. Algorithm 3.2 reports the steps of a communication phase for a *Master* process, at a generic iteration number $n \geq 0$. The definition of arrays h_k^{out}, h_k^{in} and u_k^{out}, u_k^{in} is extended to *Master* processes $\mathcal{P}_k$, $k = 1, n_{\mathcal{P}}^M - 1$ as the set of DOFs that need to be sent and received, respectively, from a different process. Steps 8-9 are not performed by the top level *Master* ($\mathcal{P}_0^{\uparrow} = \emptyset$).

A version of the code based on different C++ libraries for linear algebra is de-



(a) DFN 64K (b) DFN 128K (c) DFN 256K

Figure 3: DFNs used for strong scalability tests

scribed in [8], where the NVIDIA CUDA Toolkit [43] is used.

4. Results. This Section reports the performances of the proposed algorithm, in terms of achieved speedups in iteration time when the number of parallel processes increases. More in details, by iteration time we mean the average time per iteration over 1000 iterations of Algorithm 3.1. The pre-processing time, needed to perform tasks from 1 to 5 described at the beginning of Section 3, is negligible with respect to resolution time (task 6), and further, pre-processing is readily parallel as it involves almost no communications.

The *strong scalability* and the *weak scalability* performances of the algorithm will be shown. In strong scalability tests the dimension of the problem is fixed (number of fractures and mesh parameter) and the time to obtain the solution is measured when the number of parallel processes increases. Performances are measured in terms of the speedup $S_p := t_1/t_p$, corresponding to the ratio between the serial execution time t_1 and the time with p parallel computing processes t_p; and in terms of efficiency $E_p := S_p/p$ representing the achieved speedup divided by the ideal speedup $p/1$. On the other hand, in weak scalability tests both problem size (number of fractures, with a fixed mesh parameter) and the number of parallel processes grow with a fixed ratio. A measure of the performances is obtained, in this case, comparing the ratio between computing time and number of processes (or, equivalently, problem size) with the expected fixed one. Scalability performances are computed with respect to the number of *Slave* processes, not taking into account the number of *Masters*. This is done in order to provide an insight of the optimality of the performances with respect to the actual number of computing processes, and to better highlight how performances vary as the number of processes changes, being, instead, the number of *Master* processes fixed for each simulation.

The networks used for the simulations are generated from random probability distribution functions concerning the size, the position, the orientation, the number and the hydraulic transmissivity of fractures, adapted from the data available in [47]. The resulting networks display a large variability in terms of fracture sizes, which span about four orders of magnitude; also, highly connected fractures, having more than 10^2 traces, coexist with scarcely connected fractures, with only few traces.

An extensive set of simulations is reported, performed on the partition $A1$ of cluster *Marconi*, located at the Italian HPC center CINECA [19]. The machine is composed by 720 nodes, with 2x18-cores Intel Xeon E5-2697 v4 (Broadwell) processors at frequency 2.30 GHz, and 128 GB of RAM per node. The OpenMPI implementation of the MPI communication protocol version 3 is used, [50].

Table 1: Strong scalability tests - Network data

Id	Tot. Fracs	Conn. Fracs	Traces	Min Trcs	Max Trcs
64K	64186	64178	122543	1	390
128K	167321	129552	248422	1	392
256K	328206	253150	482755	1	442

Table 2: Strong scalability tests - DOFs data

Id	Mesh	Fracture DOF's	Trace DOF's	Total DOF's
64K	100	$6.07 \cdot 10^6$	$6.13 \cdot 10^5$	$6.68 \cdot 10^6$
128K	100	$1.23 \cdot 10^7$	$1.24 \cdot 10^6$	$1.35 \cdot 10^7$
256K	100	$2.39 \cdot 10^7$	$2.41 \cdot 10^6$	$2.64 \cdot 10^7$

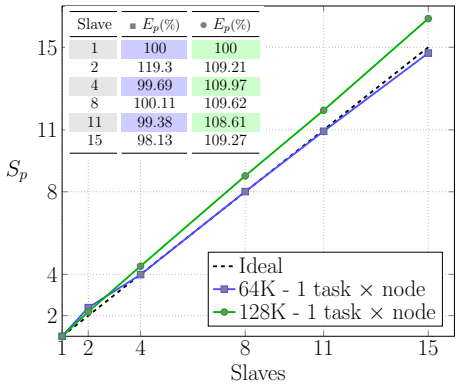


Figure 4: Strong scalability on DFN 64K and 128K

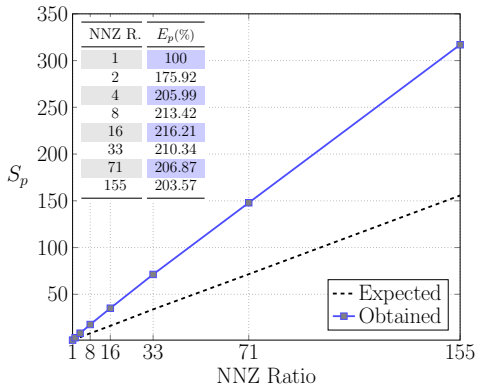
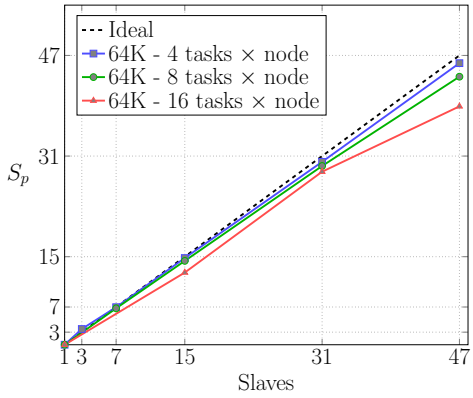
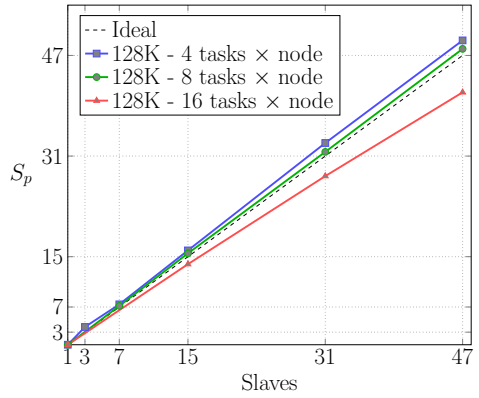


Figure 5: Sparse matrix-vector product performance test



(a) DFN 64K



(b) DFN 128K

Figure 6: Strong scalability with different task×node configurations

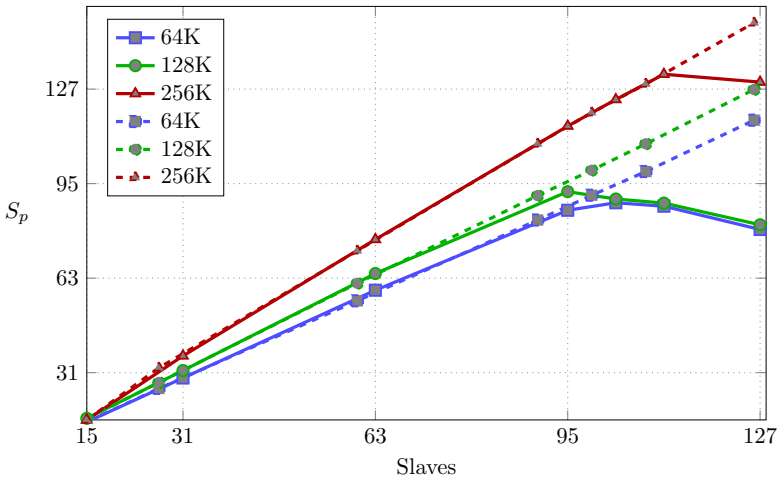


Figure 7: Strong scalability with 8 tasks×node and one (solid line) or four (dashed line) *Master* processes

4.1. Strong Scalability. Strong scalability results are shown for three different networks with an increasing number of fractures. The three networks, labelled 64K, 128K and 256K, are displayed in Figure 3. Table 1 reports, for each DFN the initial number of fractures, (column *Tot. Fracs*), i.e. before the definition of the main connected component, the number of connected fractures (column *Conn. Fracs*) and their traces (column *Traces*) and the minimum and maximum number of traces per fracture (columns *Min Trcs* and *Max Trcs*, respectively). The chosen mesh parameter and the resulting degrees of freedom on the fractures, on the traces, and their sum, are shown in Table 2. Linear finite elements are used for the description of the discrete hydraulic head h on the fractures, and trace-meshes with staggered nodes and piecewise-constant basis functions are used to describe function u on the traces. The diameter of the mesh on each trace S_m, $m = 1, \dots, M$, is chosen equal to the average between the maximum diameter of the triangular elements intersected by the trace on F_i and F_j, $(i, j) = I_S(m)$.

Figure 4 plots the speedup S_p for p ranging between 1 and 15, for the 64K and 128K networks, also reporting, in the embedded table, the corresponding efficiency values. Simulations are performed ensuring that no more than a single process is allocated on each computing node of the cluster, thus avoiding any possible memory-related access conflict between processes. An optimal speedup is achieved for the smaller network, whereas a speedup exceeding the optimal one is observed for the larger DFN. This circumstance is not an isolated case, as shown in the following, and is given by the combined effect of a very low communication cost, achieved by the present implementation, with the behaviour of the cost of linear algebra operations when the size of the involved matrices varies. This behaviour is described by the example in Figure 5, reporting how the cost of sparse matrix-vector product performed with Eigen changes when the same matrix is split in several sub-matrices taking chunks of rows, keeping instead fixed the size of the vector. The test is performed on a single core, and does not involve communications: starting from an initial reference matrix, seven subsequent simulations are performed, each time defining a sub-matrix taking one half

of the number of rows of the matrix at the previous simulation, and measuring the time required to perform the sparse matrix vector product. This mimics what happens in Algorithm 3.1 when the network is split among an increasing number of parallel processes. Speedup measurements for this test are obtained dividing the time for the computation of the matrix-vector product for the initial reference matrix by the time required to perform the same operation on each fraction of the reference matrix. In Figure 5 the achieved speedups are shown with respect to the *NNZ Ratio*, defined as the ratio between the non zero elements of the initial reference matrix and the number of non zero elements of the considered fractions of this matrix; also efficiency values are shown in the embedded table, defined, as usual, as the ratio between the achieved speedup and the ideal one. It can be seen that a speedup much larger than optimal is obtained: as an example, when the number of non zero elements of the matrix is reduced of a factor 71 the computing time is reduced of a factor close to 150. Consequently, efficiency values exceeding 200% are reached. This effect becomes less relevant when the number of non-zero elements further decreases. A similar behaviour is documented for the Eigen library in [23] in the case of full matrices, and also applies to other linear algebra libraries. As the DFN is partitioned among a larger number of parallel processes, the size of the matrices in equations (3.4)-(3.8) is reduced, as less fractures are present in sets $\mathcal{I}_k^{\text{loc}}$, $k = n_{\mathcal{P}}^M, \ldots, n_{\mathcal{P}}$, and thus the number of non zero entries of the matrices on each process reduces. The cost of communications is negligible with respect to the cost of linear algebra computations, and well shadowed by the operations on the non communicating fractures, especially when the number of parallel processes is relatively small. Thus a speedup higher than the ideal one can be achieved, and this is more evident, as expected, on larger networks, where the cost of linear algebra operations is more relevant. When more than a single process per node is allocated, cache memory access conflicts of different processes on the same computing node can deteriorate the speedup, with an impact that increases on larger networks, for which more memory is used, see Figure 6.

Figure 7 shows the achieved speedup for a number of computing processes ranging between 15 and 127, using eight cores on each node of the cluster, for the 64K, 128K and 256K networks. Solid lines represent the speedups obtained with a single *Master* process. A scalability higher than the ideal optimal value is again observed, more evident for the larger 256K network. However, when the number of parallel processes increases, the cost of the communications becomes more and more relevant, up to overweight the cost related to linear algebra manipulations. This happens earlier for the smaller networks, for the reduced cost of linear algebra operations. The reason of the decay of the parallel performances of the method, however, lies in the overload of the *Master* process, as demonstrated observing the results in dashed lines, where the same tests are repeated using a two level *Multi-Master* topology, with one top-level *Master* process and three bottom level Master processes (as in Figure 1b) and an increasing number of *Slave* processes. It can be seen that now the decay of speedup performances is completely cured on all the considered networks. The performances in the pre-decay range, i.e. $15 - 63$ *Slaves* for the 64K and 128K networks and $15 - 104$ *Slaves* for the 256K network are unaffected by the change of the communication topology, again showing the negligible weight of communications in the proposed algorithm when the *Master* process is under-saturated.

Results in Table 3 show the effect of an increment in the connectivity of the network on the scalability performances. A new DFN, labelled 64K* is introduced, having about the same number of fracture as the 64K network, but with about 1.7×10^6 traces, i.e. more than 10 times the number of traces of the 64K DFN. Using the

Table 3: Strong scalability test on a high density network and on a different architecture

Slaves	64K		64K*		64K*(MM)	
	S_p	E_p (%)	S_p	E_p (%)	S_p	E_p (%)
8	7.94	99.28	7.96	99.48	7.96	99.48
16	15.70	98.10	14.45	90.34	15.01	93.85

Table 4: Weak scalabilty tests: data on DFN families

ℓ	I_ℓ	Samples	Tot. Fracs	Conn. Fracs	Traces	DOFs
1	1000	13	1037	1035	1901	$1.33 \cdot 10^5$
2	2000	25	2073	2070	3896	$2.67 \cdot 10^5$
3	4000	19	4260	4254	8197	$5.51 \cdot 10^5$
4	8000	20	8366	8357	15991	$1.08 \cdot 10^6$
5	16000	11	16125	16116	30747	$2.08 \cdot 10^6$
6	32000	16	32000	31987	60923	$4.12 \cdot 10^6$
7	64000	12	77302	64535	123848	$8.34 \cdot 10^6$
8	128000	10	167321	129479	248275	$1.67 \cdot 10^7$

same mesh parameter of the previous simulations, 5.86×10^6 DOFs for the hydraulic head (i.e. approximately the same number as DFN 64K) and 1.4×10^7 DOFs for the variables on the traces are obtained, which is over one order of magnitude more than the 64K network. Despite the sensible increase in the number of traces and trace DOFs, and the consequent increase of the cost of communications, the scalability performances of the code are only marginally affected, as it can be noticed comparing columns 64K and 64K* of Table 3, where speedup and efficiency values are reported for the 64K and 64K* networks, respectively. Scalability performances on the 64K* network can be further improved introducing a *multi-Master* topology, as shown in column 64K*(MM). For memory capacity reasons, scalability data are computed using the time with four computing processes as reference, and using a single core per node. The results of Table 3 are computed on the partition *A2* of the cluster *Marconi*, equipped with Intel Xeon Phi7250 1.4 GHz (KnightLandings) processors, a different architecture from that used for the previous tests, thus also showing the good scalability of the code on different architectures.

4.2. Weak Scalability. Weak scalability tests are performed on families $\mathcal{D}_\ell$, $\ell = 1, \dots, 8$ of randomly generated DFNs: each family is characterized by networks having approximately the same number of fractures, whereas all other properties, such as the position, orientation, density and the hydraulic conductivity of the fractures, are randomly generated starting from the same probability distributions, based on the data in [47]. For each family $\mathcal{D}_\ell$, an initial total number of fractures is fixed and then various samples are generated, according to the distributions of the various parameters: information on each DFN family can be found in Table 4, where the number of networks considered (column *Samples*), the initial total number of fractures (column *Tot. Fracs*), the average number of connected fractures (column *Conn.*

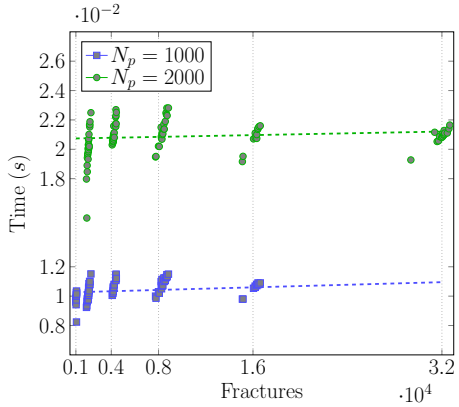


Figure 8: Weak Scalability - Case $N_p = 1000$ and $N_p = 2000$

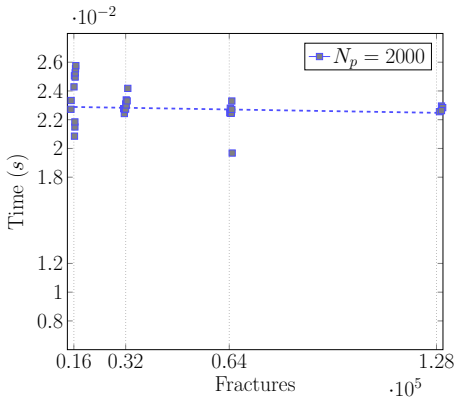


Figure 9: Weak Scalability - Case $N_p = 2000$ with 8 cores per node

Fracs), the average number of traces (column *Traces*) and the average total number of degrees of freedom (column *DOFs*) are reported. It can be seen that networks with a number of fractures ranging between about $10^3 = I_1$ and $128 \times 10^3 = I_8$ are used, and the total number of DOFs spans two orders of magnitude. Given p the number of parallel computing processes and I_ℓ the number of fractures characterizing family $\mathcal{D}_\ell$, $\ell = 1, \ldots, 8$, three different tests are performed, each with an approximately fixed ratio $N_p := I_\ell/p$, obtained varying p between 1 and 63, and choosing ℓ accordingly.

Weak scalability performances are reported in Figure 8 for values of $N_p \sim 1000$ and $N_p \sim 2000$ and a number of computing processes ranging between 2 and 16, with a single process allocated on each computing node of the cluster. Each point in the picture represents the value of iteration time for one of the networks, and the dashed lines correspond to the best fitting line of the data for each of the two N_p-values considered. Similar results are obtained for both values of N_p. It can be seen that best fitting lines are nearly parallel and horizontal, thus showing that iteration time is, on average, constant when the number of processes increases and N_p is kept almost constant. We also observe that the variability in computing time among DFNs belonging to the same family tends to decrease as the number of fractures in the network increases.

For values of $N_p \lesssim 1000$ the scalability performances start to gradually deteriorate, suggesting that, for optimal performances, each parallel process should handle not less than 10^5 DOFs.

Figure 9 plots the results obtained with $N_p = 2000$, by using eight cores per node and a number of parallel computing processes ranging between 7 and 63. Again, each point in this figure corresponds to the computing time of a different network and the dashed line is the best fitting of the data. The fitting line is almost horizontal, slightly decreasing, and this can be probably related again to the super-scalability behaviour observed in terms of strong scalability, for the larger networks.

5. Conclusions. A new implementation of an optimization-based approach for the simulation of subsurface flows has been described and tested. Mesh generation, problem discretization and resolution as well as other pre-processing and post-processing operations, are all performed in parallel by the proposed code. Full im-

plementation details are given, as, in particular, on the organization of algebraic operations, on the use of MPI communications and on the chosen graph topology for the parallel environment. Results are shown in terms of both strong and weak scalability on massive networks of over than 10^5 fractures, using up to 128 parallel processes, obtaining optimal scalability performances. In contrast to other numerical schemes for DFN flow simulations, where meshing time can be orders of magnitude larger than resolution time, the proposed approach achieves negligible meshing time, without compromising resolution time, thus being a new efficient tool for underground flow simulations at the reservoir scale.

Similar scalability performances are expected in the resolution of non stationary advection-dispersion problems, where, following the method described in [14], a problem analogous to (2.11) needs to be solved at each time-step. In the framework of time-dependent simulations, excellent scalability properties are of paramount importance for the practical applicability of the method. The same setting, as proposed in the present work, can be used for the parallel implementation of the optimization method for fracture-matrix flow simulations on non conforming meshes with the DFM approach, proposed in [13]. In this case, the balancing of the load among the computing processes results to be more difficult, as now bi-dimensional and three-dimensional problems need to solved simultaneously.

REFERENCES

[1] Antonietti, Paola F., Formaggia, Luca, Scotti, Anna, Verani, Marco, and Verzott, Nicola, *Mimetic finite difference approximation of flows in fractured porous media*, ESAIM: M2AN, 50 (2016), pp. 809–832, https://doi.org/10.1051/m2an/2015087.

[2] M. Benedetto, S. Berrone, A. Borio, S. Pieraccini, and S. Scialò, *A hybrid mortar virtual element method for discrete fracture network simulations*, J. Comput. Phys., 306 (2016), pp. 148–166, https://doi.org/http://dx.doi.org/10.1016/j.jcp.2015.11.034.

[3] M. F. Benedetto, A. Borio, and S. Scialò, *Mixed virtual elements for discrete fracture network simulations*, Finite Elements in Analysis & Design, 134 (2017), pp. 55–67, https://doi.org/10.1016/j.finel.2017.05.011.

[4] S. Berrone, A. Borio, C. Fidelibus, S. Pieraccini, S. Scialò, and F. Vicini, *Advanced computation of steady-state fluid flow in discrete fracture-matrix models: Fem–bem and vem–vem fracture-block coupling*, GEM - International Journal on Geomathematics, (2018), https://doi.org/10.1007/s13137-018-0105-3, https://doi.org/10.1007/s13137-018-0105-3.

[5] S. Berrone, A. Borio, and S. Scialò, *A posteriori error estimate for a PDE-constrained optimization formulation for the flow in DFNs*, SIAM J. Numer. Anal., 54 (2016), pp. 242–261, https://doi.org/http://dx.doi.org/10.1137/15M1014760.

[6] S. Berrone, A. Borio, and F. Vicini, *A posteriori error estimate for a PDE-constrained optimization formulation for the flow in DFNs*. Submitted for publication.

[7] S. Berrone, C. Canuto, S. Pieraccini, and S. Scialò, *Uncertainty quantification in discrete fracture network models: Stochastic geometry*, Water Resources Research, 54 (2018), pp. 1338–1352, https://doi.org/10.1002/2017WR021163.

[8] S. Berrone, A. D'Auria, and F. Vicini, *Performance analysis of GPGPUs for flow simulations in Discrete Fracture Networks*, GEM - International Journal on Geomathematics, 10 (2019), p. 8, https://doi.org/10.1007/s13137-019-0121-y.

[9] S. Berrone, C. Fidelibus, S. Pieraccini, and S. Scialò, *Simulation of the steady-state flow in discrete fracture networks with non-conforming meshes and extended finite elements*, Rock Mechanics and Rock Engineering, 47 (2014), pp. 2171–2182, https://doi.org/http://dx.doi.org/10.1007/s00603-013-0513-5.

[10] S. Berrone, C. Fidelibus, S. Pieraccini, S. Scialò, and F. Vicini, *Unsteady advection-diffusion simulations in complex Discrete Fracture Networks with an optimization approach*, Journal of Hydrology, 566 (2018), pp. 332–345, https://doi.org/10.1016/j.jhydrol.2018.09.031.

[11] S. Berrone, S. Pieraccini, and S. Scialò, *A PDE-constrained optimization formulation for discrete fracture network flows*, SIAM J. Sci. Comput., 35 (2013), pp. B487–B510, https://doi.org/http://dx.doi.org/10.1137/120865884.

[12] S. Berrone, S. Pieraccini, and S. Scialò, *An optimization approach for large scale simulations of discrete fracture network flows*, J. Comput. Phys., 256 (2014), pp. 838–853, https://doi.org/http://dx.doi.org/10.1016/j.jcp.2013.09.028.

[13] S. Berrone, S. Pieraccini, and S. Scialò, *Flow simulations in porous media with immersed intersecting fractures*, Journal of Computational Physics, 345 (2017), pp. 768 – 791, https://doi.org/https://doi.org/10.1016/j.jcp.2017.05.049.

[14] S. Berrone, S. Pieraccini, and S. Scialò, *Non-stationary transport phenomena in networks of fractures: Effective simulations and stochastic analysis*, Computer Methods in Applied Mechanics and Engineering, 315 (2017), pp. 1098 – 1112, https://doi.org/http://dx.doi.org/10.1016/j.cma.2016.12.006.

[15] S. Berrone, S. Pieraccini, S. Scialò, and F. Vicini, *A parallel solver for large scale DFN flow simulations*, SIAM J. Sci. Comput., 37 (2015), pp. C285–C306, https://doi.org/http://dx.doi.org/10.1137/140984014.

[16] E. Burman, P. Hansbo, M. G. Larson, and K. Larsson, *Cut finite elements for convection in fractured domains*, Computers & Fluids, (2018), https://doi.org/https://doi.org/10.1016/j.compfluid.2018.07.022.

[17] M. C. Cacas, E. Ledoux, G. de Marsily, B. Tillie, A. Barbreau, E. Durand, B. Feuga, and P. Peaudecerf, *Modeling fracture flow with a stochastic discrete fracture network: calibration and validation: 1. the flow model*, Water Resour. Res., 26 (1990), pp. 479–489, https://doi.org/http://dx.doi.org/10.1029/WR026i003p00479.

[18] F. Chave, D. Di Pietro, and L. Formaggia, *A hybrid high-order method for darcy flows in fractured porous media*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1063–A1094, https://doi.org/10.1137/17M1119500.

[19] Cineca, *MARCONI - cineca's tier0 system*, 2018. Retrieved at: https://www.cineca.it/en/content/marconi (Accessed October 2018).

[20] W. S. Dershowitz and C. Fidelibus, *Derivation of equivalent pipe networks analogues for three-dimensional discrete fracture networks by the boundary element method*, Water Resource Res., 35 (1999), pp. 2685–2691, https://doi.org/http://dx.doi.org/10.1029/1999WR900118.

[21] J. R. D. Dreuzy, P. Davy, and O. Bour, *Hydraulic properties of two-dimensional random fracture networks following a power law length distribution: 2., permeability of networks based on log-normal distribution of apertures*, Water Resour. Res., 37 (2001), pp. 2079–2095, https://doi.org/http://dx.doi.org/10.1029/2001WR900010.

[22] *Eigen documentation*, 2018. Retrieved at: http://eigen.tuxfamily.org/dox/ (Accessed October 2018).

[23] *Eigen benchmark*, 2018. Retrieved at: http://eigen.tuxfamily.org/index.php?title=Benchmark (Accessed October 2018).

[24] C. Fidelibus, G. Cammarata, and M. Cravero, *Hydraulic characterization of fractured rocks. In: Abbie M, Bedford JS (eds) Rock mechanics: new research.*, Nova Science Publishers Inc., New York, 2009.

[25] B. Flemisch, A. Fumagalli, and A. Scotti, *A review of the XFEM-Based Approximation of Flow in Fractured Porous media*, in Advances in Discretization Methods, vol. 12 of SEMA SIMAI Springer Series, Springer International Publishing, Switzerland, 2016, pp. 47–76.

[26] A. Fourno, T.-D. Ngo, B. Noetinger, and C. L. Borderie, *Frac: A new conforming mesh method for discrete fracture networks*, Journal of Computational Physics, 376 (2019), pp. 713 – 732, https://doi.org/https://doi.org/10.1016/j.jcp.2018.10.005.

[27] A. Fumagalli and E. Keilegavlen, *Dual Virtual Element Methods for Discrete Fracture Matrix Models*. to appear in Oil & Gas Science and Technology.

[28] A. Fumagalli and E. Keilegavlen, *Dual virtual element method for discrete fractures networks*, SIAM Journal on Scientific Computing, 40 (2018), pp. B228–B258, https://doi.org/10.1137/16M1098231.

[29] A. Fumagalli, E. Keilegavlen, and S. Scialò, *Conforming, non-conforming and non-matching discretization couplings in discrete fracture network simulations*, J. Comput. Phys., (2018), https://doi.org/10.1016/j.jcp.2018.09.048. In press.

[30] A. Hobè, D. Vogler, M. P. Seybold, A. Ebigbo, R. R. Settgast, and M. O. Saar, *Estimating flow rates through fracture networks using combinatorial optimization*. Available at: arXiv:1801.08321, 2018.

[31] J. Hyman, C. Gable, S. Painter, and N. Makedonska, *Conforming delaunay triangulation of stochastically generated three dimensional discrete fracture networks: A feature rejection algorithm for meshing strategy*, SIAM Journal on Scientific Computing, 36 (2014), pp. A1871–A1894, https://doi.org/http://dx.doi.org/10.1137/130942541.

[32] J. D. Hyman, S. Karra, N. Makedonska, C. W. Gable, S. L. Painter, and H. S.

Viswanathan, *dfnworks: A discrete fracture network framework for modeling subsurface flow and transport*, Computers & Geosciences, 84 (2015), pp. 10 – 19, https://doi.org/http://dx.doi.org/10.1016/j.cageo.2015.08.001.

[33] S. Karra, D. O'Malley, J. D. Hyman, H. S. Viswanathan, and G. Srinivasan, *Modeling flow and transport in fracture networks using graphs*, Physical Review E, 97 (2018), https://doi.org/10.1103/PhysRevE.97.033304.

[34] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392, https://doi.org/10.1137/S1064827595287997.

[35] M. Köppel, V. Martin, J. Jaffré, and J. E. Roberts, *A lagrange multiplier method for a discrete fracture model for flow in porous media*, Computational Geosciences, (2018), https://doi.org/10.1007/s10596-018-9779-8.

[36] L. Li and S. H. Lee, *Efficient Field-Scale Simulation of Black Oil in a Naturally Fractured Reservoir Through Discrete Fracture Networks and Homogenized Media*, SPE Reservoir Evaluation & Engineering, 11 (2008), https://doi.org/10.2118/103901-PA.

[37] N. Makedonska, S. L. Painter, Q. M. Bui, C. W. Gable, and S. Karra, *Particle tracking approach for transport in three-dimensional discrete fracture networks*, Computational Geosciences, 19 (2015), pp. 1123–1137, https://doi.org/http://dx.doi.org/10.1007/s10596-015-9525-4.

[38] A. Moinfar, A. Varavei, K. Sepehrnoori, and R. Johns, *Development of an Efficient Embedded Discrete Fracture Model for 3D Compositional Reservoir Simulation in Fractured Reservoirs*, SPE Journal, 19 (2014), https://doi.org/10.2118/154246-PA.

[39] H. Mustapha, A. Ghorayeb, K. Mustapha, and P. Saramito, *An efficient parallel mixed method for flow simulations in heterogeneous geological media*, International Journal of Computer Mathematics, 87 (2010), pp. 607–618, https://doi.org/10.1080/00207160802158728.

[40] S. P. Neuman, *Trends, prospects and challenges in quantifying flow and transport through fractured rocks*, Hydrogeol. J., 13 (2005), pp. 124–147, https://doi.org/10.1007/s10040-004-0397-2.

[41] T. D. Ngo, A. Fourno, and B. Noetinger, *Modeling of transport processes through large-scale discrete fracture networks using conforming meshes and open-source software*, Journal of Hydrology, 554 (2017), pp. 66 – 79, https://doi.org/https://doi.org/10.1016/j.jhydrol.2017.08.052.

[42] B. Nœtinger, *A quasi steady state method for solving transient Darcy flow in complex 3D fractured networks accounting for matrix to fracture flow*, J. Comput. Phys., 283 (2015), pp. 205–223, https://doi.org/http://dx.doi.org/10.1016/j.jcp.2014.11.038.

[43] Nvidia CUDA, *CUDA toolkit Documentation*. https://docs.nvidia.com/cuda-toolkit, 2008.

[44] G. Pichot, J. Erhel, and J. de Dreuzy, *A generalized mixed hybrid mortar method for solving flow in stochastic discrete fracture networks*, SIAM Journal on scientific computing, 34 (2012), pp. B86 – B105, https://doi.org/http://dx.doi.org/10.1137/100804383.

[45] J. R. Shewchuk, *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, in Applied Computational Geometry: Towards Geometric Engineering, M. C. Lin and D. Manocha, eds., vol. 1148 of Lecture Notes in Computer Science, Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry.

[46] S. Srinivasan, J. Hyman, S. Karra, D. O'Malley, H. Viswanathan, and G. Srinivasan, *Robust system size reduction of discrete fracture networks: a multi-fidelity method that preserves transport characteristics*, Computational Geosciences, (2018).

[47] Svensk Kärnbränslehantering AB, *Data report for the safety assessment sr-site*, 2010. Tech. Rep. TR-10-52, Stockholm, Sweden.

[48] The Open MPI Project, *MPI_ Send_ init(3) man page (version 3.0.2)*, 2017. Retrieved at: https://www.open-mpi.org/doc/v3.0/man3/MPI_Send_init.3.php (Accessed October 2018).

[49] The Open MPI Project, *MPI_ Type_ indexed(3) man page (version 2.0.4)*, 2017. Retrieved at: https://www.open-mpi.org/doc/v2.0/man3/MPI_Type_indexed.3.php (Accessed October 2018).

[50] The Open MPI Project, *Open MPI Documentation*, 2017. Retrieved at: https://www.open-mpi.org/doc/ (Accessed October 2018).

[51] M. Valera, Z. Guo, P. Kelly, S. Matz, V. A. Cantu, A. G. Percus, J. D. Hyman, G. Srinivasan, and H. S. Viswanathan, *Machine learning for graph-based representations of three-dimensional discrete fracture networks*, Computational Geosciences, 22 (2018), pp. 695–710, https://doi.org/10.1007/s10596-018-9720-1.