Mathematical Geoscience manuscript No.
(will be inserted by the editor)

Efficient partitioning and reordering of conforming virtual element discretizations for large scale Discrete Fracture Network flow parallel solvers

Stefano Berrone · **Alice Raeli**

Received: date / Accepted: date

Abstract Discrete Fracture Network models are largely used for very large scale geological flow simulations. For this reason numerical methods require an investigation of tools for efficient parallel solutions on High Performance Computing systems. In this paper we discuss and compare several partitioning and reordering strategies, that result to be highly efficient and scalable, overperforming the classical mesh partitioning approach used to partition a conforming mesh among several processes.

Keywords Partitioning strategies · Parallel solvers · Virtual Element Methods · Discrete Fracture Networks · Geological Problems

Mathematics Subject Classification (2000) [2020] 65N30 · 65N50 · 65Y05 · 68U20 · 68W10 · 86-08

1 Introduction

The flow in fractured media is a relevant topic in several engineering applications such as aquifers monitoring, disposal and geological storage of nuclear wastes, prevention of accidental dispersion of contaminants, oil and gas enhanced production and many other applications (Lei et al. 2017), (Council et al. 1996). When the rock matrix of the geological formation has a very small porosity, the contribution of the rock matrix surrounding fractures can have a marginal impact on the flow pattern. Intensity and direction of flow often depend almost uniquely on the distribution of fractures and on their hydraulic properties. *Discrete Fracture Network* models, DFN in the following, simulate transport and flow within fractured material (usually rocks)

S. Berrone (🖂)
E-mail: stefano.berrone@polito.it

A. Raeli (🖂)
E-mail: alice.raeli@polito.it
Politecnico di Torino
Dipartimento di Scienze Matematiche,
Corso Duca degli Abruzzi 24, 10129 Torino, Italy

using discrete computational strategies to approach the real solution (Neuman 2005), (Jaffré and Roberts 2012).

In Section 2 we briefly introduce the DFN flow formulation in an impervious rock matrix, following (Cacas et al. 1990), (Nordqvist et al. 1992), (Dershowitz and Fidelibus 1999), (Fidelibus 2007), (Berrone et al. 2017), modeling fractures as planar polygons. Brief information concerning the mesh creation and the degrees of freedom handling are given in Subsections 2.1 and 2.3 respectively.

Due to the stochastic nature of the DFN model generated starting from probabilistic distribution of fracture position, orientation, size and of hydraulic parameters, several large simulations are needed in order to perform uncertainty quantification for flow quantities of interest (Berrone et al. 2015a), (Berrone et al. 2018), (Canuto et al. 2019), (Pieraccini 2020). As a consequence of the stochastic generation of the networks, DFNs for practical applications are usually very complex. In fact, DFNs count a large number of fracture intersections with some critical properties, such as very narrow angles or multiple intersection zones. The generation of a mesh conforming to fracture intersections in a DFN is often a complicated and challenging process, so different numerical approaches possibly circumventing the problem could be applied: standard or mixed Finite Element Methods (Vohralik et al. 2007), (Hyman et al. 2014), (Sentís and Gable 2017), hybrid mortar methods (Pichot et al. 2010), (Benedetto et al. 2016a), optimization methods (Mustapha and Mustapha 2007),(Berrone et al. 2015b), and others (Nœtinger and Jarrige 2012). This work focuses on a Virtual Element discretization with a conforming polygonal mesh approach (Benedetto et al. 2016b) following the strategy presented in (Berrone et al. 2021) for the generation of the conforming mesh, nevertheless the implementation presented in this work is independent of the element geometry and can be easily extended to other approaches.

The number of mesh cells required for a DFN simulation depends on the number and the size of fractures, the density of the network, the range of scale-lengths generated by the network and on the accuracy of the approximation sought. Despite the large scale of realistic 3D DFN geological formations, an efficient parallel High Performance Computing approach enables flow and transport simulations. In (Berrone et al. 2015b),(Berrone et al. 2019) is presented a parallel *master/slaves* approach associated with an PDE (Partial Differential Equation) constrained optimization approach: this choice guarantees scalability and accuracy on the solution requiring a non-standard solver for the linear systems. The method used in this work is based on a more standard conforming discretization approach and proposes a parallelization of the global problem, based on an equitable distribution of the work load, that minimizes communications and may resorts to common solvers and preconditioners for the linear systems.

An efficient parallel DFN simulation necessitate a high-quality partitioning of the degrees of freedom, such that the computations are well-balanced with minimal communications between processes. A multi-level approach is proposed in (Ushijima-Mwesigwa et al. 2021) where the advantages of a DFN-based partitioning are presented in comparison with a classical mesh-partitioning approach. In Section 3 different partitioning strategies of the DFN among the processes are presented resorting to different types of DFN based graphs representation; the chosen tool for problem partitioning is the graph partitioning library METIS (Karypis and Kumar 1999). To each parallel process is assigned a subset of fractures such that these subsets are disjoint and all the fractures in the DFN are assigned only to one process. Once the DFN is partitioned among processes, in Section 4, we present a method to number the degrees of freedom with respect to the partitioned DFN. The objective of this ordering is to minimize the communications of the iterative method used to solve the linear system by PETSc (Balay et al. 2019) toolkit. The resolution method used in our context is a preconditioned conjugate gradient with a Jacobi preconditioner, other, more efficient, approaches can be used but are not tested in this paper, as we focus on the effect of the MPI parallel communications not on the efficiency of the linear solver.

Although in this work we refer to the specific case of DFNs, the methods presented are compatible with other problems on networks and graphs with a relevant cost of computational operations on nodes and edges; the same approach can be applied to other numerical methods possibly changing the structure of the mesh. In Section 5 we conclude with numerical tests that investigate the partitioning/reordering methods mentioned above.

2 The Discrete Fracture Network Discretization

In this section, we briefly introduce the notation used in the following. We assume a Darcy flow model inside the fractures and an impervious surrounding rock matrix. Moreover, we assume continuity of the hydraulic head and conservation of the flux at the fracture intersections. We do not provide deeper details of the flow model, as they are not relevant for the focus of the paper and the proposed methods very weakly depend on these modeling choices. Full details concerning the model used for the numerical tests can be found in (Berrone et al. 2017) and references therein.

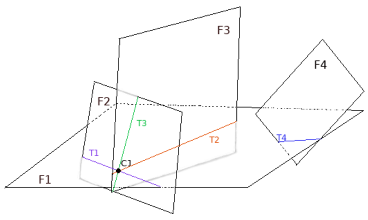


Fig. 1: A simple DFN with four planar rectangular fractures and four intersections.

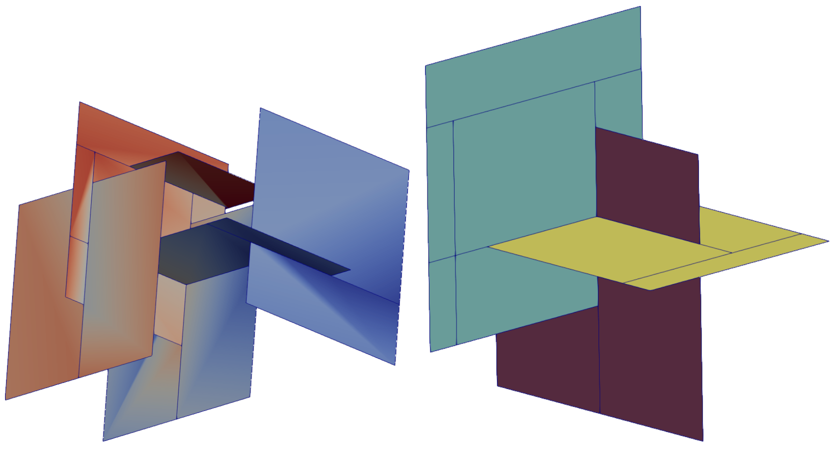
Let Ω be the DFN, given by the union of planar polygonal fractures $F_r, r = 1,...,\#\mathscr{F}$, intersecting each other to form traces $T_m, m = 1,...,\#\mathscr{T}$, see Figure 1. We refer to fractures and traces as *domains* and *interfaces*. A multiple domains network will represent the DFN $\Omega := \bigcup_{r=1,...,\#\mathscr{F}} F_r$.

Let $\mathscr{F}$ be the set of the fractures and $\mathscr{T}$ the set of traces; we assume that each trace T_m, for $m = 1,\ldots,\#\mathscr{T}$, is given by the intersection of two fractures $T_m = F_r \cap F_s$. This assumption induce a map between each trace index and a couple of fracture indexes,

$IT(m) = (r,s)$ with $r < s$, such that $F_r \cap F_s = T_m$. We define $\mathscr{CP}$ the set of cross points of the DFN; we assume that a cross point is a multiple intersection point between three traces, they so belong to three fractures as well. The induced intersection map is defined such that $ICP(t) = (r,s,q)$ with $r < s < q$, and $F_r \cap F_s \cap F_q = CP_t, \ \forall t = 1,\dots \#\mathscr{CP}$.

In Figure 1 an example with four fractures is given, so we have $\mathscr{F} := \bigcup_{r=1,\dots,4} F_i$. This network contains four traces and we can list the following induced maps: $IT(1) = (1,2)$, $IT(2) = (1,3)$, $IT(3) = (2,3)$ and $IT(4) = (1,4)$. Moreover, the point CP_1 given by the intersection of F_1, F_2 and F_3, such that $ICP(1) = (1,2,3)$, or by the intersection of traces T_1, T_2 and T_3, is a cross point.

In Figure 2 a DFN with six fractures is represented that will be often used in the following as example. We will refer to it as *Frac6* for the sake of simplicity.



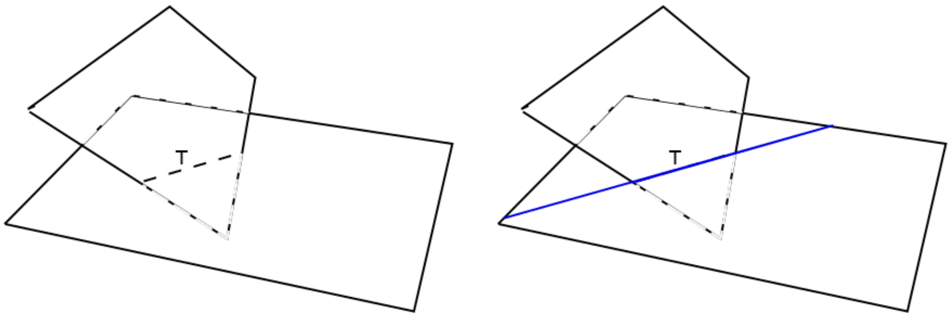
(a) Six Fractures Network (b) Cross point, intersection among three fractures.

Fig. 2: *Frac6*. A simple DFN with six fractures and six traces.

2.1 The Mesh

In this section we briefly recall the approach described in (Berrone et al. 2021) to get a polygonal conforming mesh on a DFN. Let us consider for sake of simplicity two intersecting fractures (Figure 3) and let us consider each fracture as a convex cell of a temporary mesh. These cells partially or totally crossed by the trace are split in polygonal convex sub-cells by the trace segments or its possible extensions (see Figure 2a and Figure 3b). This cutting process involves the minimum number of cells on the fracture and is repeated iteratively for all the traces and all the fractures. Full details of this approach can be found in (Berrone et al. 2021). The mesh obtained by this process is called *(almost) minimal mesh* (Figure 4a). The minimal mesh can be then refined using, for example, a uniform refinement or an a posteriori mesh refinement as described in (Berrone et al. 2021).

In the following we denote by $E_e \in \mathscr{E}$, $e = 1, \dots, \#\mathscr{E}$ a generic cell, and by N_e the set of its nodes.

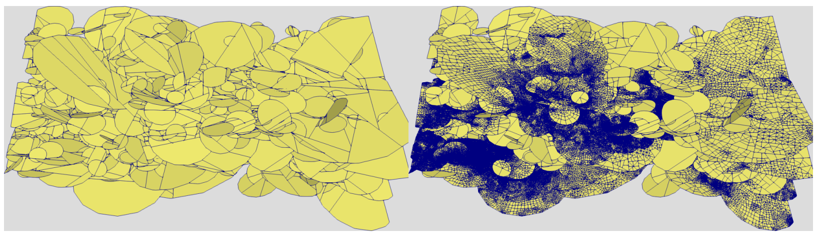


(a) Two intersecting fractures.

(b) Polygonal mesh on the two fractures induced by trace T.

Fig. 3: Polygonal minimal mesh creation.

2.2 The VEM Discretization



(a) Minimal mesh induced by traces.

(b) Mesh after 22 adaptive refinements (by momentum cut direction).

Fig. 4: *Frac*1000: Minimal and refined mesh for a VEM resolution. The mesh is refined each iteration by and adaptive mesh refinement algorithm.

The discretization approach applied here is the order 1 Virtual Element Method that allows the discretization of a second order partial differential equation on a conforming polygonal mesh, (Ahmad et al. 2013 Beirão da Veiga et al. 2013 Beirão da Veiga et al. 2014 Beirão da Veiga et al. 2015). The mesh generation and refinement process here considered yields to convex elements and all the tests proposed are provided on such kind of meshes. Due to the stochastic nature of the DFN the conforming polygonal mesh may contain elements characterized by a low quality due, for example, to a large aspect ratio or the coexistence in the same cell of short and long edges. Although badly shaped elements may affect the quality of a VEM solution, when low polynomial order elements are considered this property is not relevant as shown in (Berrone and Borio 2017a).

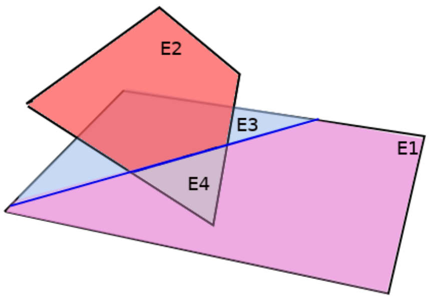
In order to test the proposed methods on large suitable meshes, in the last Subsection 5.5, we apply the proposed methods on a polygonal mesh obtained applying an *a posteriori* mesh refinement. In this context the problem is solved on progressively refined meshes, being the refinement based on information provided by an *a posteriori* error estimator (Fig. 4b).

2.3 The DOFs Handler

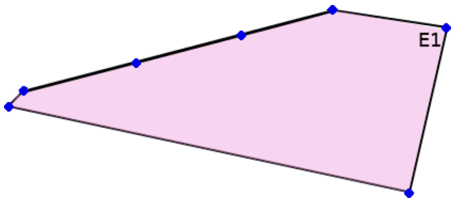
A node is a point in space, defined by its coordinates to which we associate a degree of freedom. A degree of freedom (Dof) can be defined in many different ways, in the following it will be the point-wise value of the discrete solution on the node considered, and the solution to our problem is written as a linear combination of Lagrangian basis functions ϕ_i:

$$u_h = \sum_{i=0}^{nDofs-1} Dof_i \phi_i, \tag{1}$$

where ϕ_i is the basis function related to the i-th Dof, and $nDofs$ is the total number of degrees of freedom. In Figure 5a is reported the minimal mesh for two intersecting fractures with four cells E_e, $e = 1,2,3,4$ and in Figure 5b the nodes of the cell E_1 are highlighted with blue bullets.



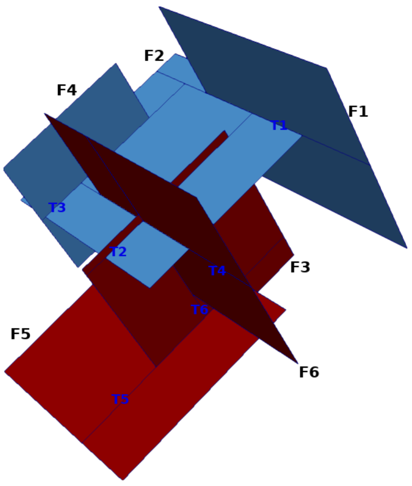
(a) Minimal mesh for two intersecting fractures.



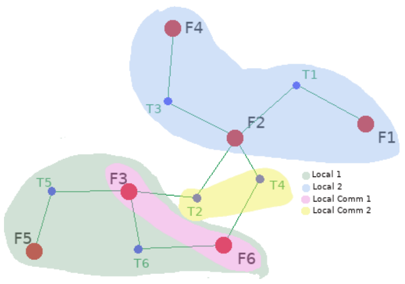
(b) Nodes (blue dots) induced on cell E_1.

Fig. 5: Minimal mesh for two intersecting fractures.

Using a conforming mesh, the degrees of freedom on the DFN include the degrees of freedom at the cross points in $\mathscr{CP}$, the remaining degrees of freedom on the traces in $\mathscr{T}$, and the remaining degrees of freedom inside the fractures in $\mathscr{F}$. Let [.] be



(a) *Frac6* partition between 2 processes. Red fractures are assigned to P_1, blue ones are assigned to P_2.



(b) Graph partitioning among 2 processes. Local and local communicating networks

Fig. 6: *Frac6* - 2 processes partition.

the operator such that $[\mathscr{CP}]$ is the number of degrees of freedom in $\mathscr{CP}$ ($[\mathscr{CP}] = \#\mathscr{CP}$), $[\mathscr{T}]$ is the number of degrees of freedom of the traces (including the degrees of freedom at the cross points), and $[\mathscr{F}] = nDofs$ is the number of degrees of freedom on the fractures (total number of degrees of freedom).

In the following each degree of freedom is uniquely related to an integer index $i = 0, \ldots, nDofs - 1$ and we refer to a possible permutation of this index set as a reordering. For sake of simplicity, let us assume we are dealing with a problem with homogeneous Dirichlet boundary conditions. For each node $\mathbf{x}$, not on the boundary of the DFN, let i be the corresponding Dof index, ϕ_i the corresponding Lagrangian basis function and $\mathscr{E}_{\mathbf{x}} = \{E \in \mathscr{E} \,|\, E \cap \mathbf{x} \neq \emptyset\}$ its support, i.e. the set of cells intersecting the node $\mathbf{x}$; we define $Neigh_i$ the set of the Dofs indices corresponding to the vertices of the cells in $\mathscr{E}_{\mathbf{x}}$ called *neighborhood* of the i-th Dof. Most of the Dofs on fractures have a neighborhood within the fracture, but Dofs on traces and cross points have adjacent cells on different fractures. The neighborhoods of the degrees of freedom of traces and cross points are relevant during the partitioning strategies presented in the following, and, when the connected fractures lie on different MPI processes, they are related to the part of the solution that requires communications between processes.

3 Parallel Partitioning

The discrete structure of a DFN (union of polygonal fractures) naturally implies several graph representations, (Hendrickson and Leland 1995), (Karypis and Kumar 1999), (Ushijima-Mwesigwa et al. 2021). In order to balance the computations in DFN flow simulations two main partitioning strategies are possible: the mesh-based partitioning, i.e. the partitioning of the Dofs of the full DFN based on the mesh connectivity, and the DFN-based ones, i.e. the partitioning of the geometrical objects (fractures, traces, cross points) and the corresponding Dofs based on objects connectivity.

In the following we present three different DFN-based graph partitioning strategies among all MPI processes (Prs). Let $\mathscr{P} = \cup_{i=1}^{\#\mathrm{Prs}} P_i$ be a partition of fractures, where P_i is the set of fracture assigned to i-th process of our parallel environment with $P_i \cap P_j = \emptyset, \forall i \neq j$. Once the DFN fractures are divided among the processes, auxiliary sub-networks are created in order to suitably manage the communications between processes.

The **local network** $\mathscr{L}_i$ of the i-th process is an ordered set containing the set of fractures P_i and the set of internal traces $\mathscr{T}_{P_i} = \{T_m \in \mathscr{T} \,|\, IT(m) = (r,s), F_r, F_s \in P_i\}$, i.e. $\mathscr{L}_i = (P_i, \mathscr{T}_{P_i})$.

The **local communicating network** $\mathscr{LC}_i$ is an ordered set containing a set of fractures and a set of traces that are involved in communications during resolution. Let $\mathscr{F}_{LC,P_i} = \{F_s \in P_i \,|\, \exists T_m \in \mathscr{T} : IT(m) = (r,s), F_r \notin P_i\}$ be the fractures in P_i that share a trace with a fracture not belonging to P_i having a fracture index smaller than the one of the fracture in P_i. Moreover, let $\mathscr{T}_{LC,P_i} = \{T_m \in \mathscr{T} \,|\, IT(m) = (r,s), F_r \in P_i, F_s \notin P_i\}$ be the set of traces shared by a fracture of P_i and a fracture not belonging to P_i with a fracture index larger than the one in P_i. We define $\mathscr{LC}_i = (\mathscr{F}_{LC,P_i}, \mathscr{T}_{LC,P_i})$. Furthermore, the process that owns a trace in its local communicating network is the one that handles its corresponding Dofs.

We explore in detail the *Frac6* example in Figure 6a for a two processes partition: red highlighted fractures lie on $\mathscr{L}_1$, light blue ones belong to $\mathscr{L}_2$. In Figure 6b $\mathscr{L}_1$ and $\mathscr{L}_2$ are disjointed in their graph representation (highlighted two disjointed zones). The traces T_5 and T_6 lie on process 1 and they are totally local, so they belong to $\mathscr{L}_1$. Analogously $T_1, T_3 \in \mathscr{L}_2$. The traces T_2 and T_4 are not local as long they rely two fractures in different processes. Following the construction strategy ($IT(2) = (2,3), IT(4) = (2,6)$) in both cases the two traces are handled by process 2: their Dofs are indexed with the Dofs of fracture F_2, whereas the process 1 set up the two fractures F_3 and F_6 to receive Dof indices on these traces from the process 2 (pink subset in Figure 6b). We have $\mathscr{LC}_1 = \{F_3, F_6\}$, and $\mathscr{LC}_2 = \{T_2, T_4\}$.

We define the **cut** C of a network partition $\mathscr{P}$ as the number of traces that connect fractures belonging to different processes:

$$C = |\mathscr{C}|, \quad \mathscr{C} = \{T_m \in \mathscr{T} | IT(m) = (r,s), F_r \in P_i, F_s \in P_j, i \neq j\}. \tag{2}$$

We note that C is the sum of the number of all the traces contained in local communicating networks of the partitioning.

Let D_i be the number of degrees of freedom of the sub-network associated to the partition $P_i \in \mathscr{P}$, we define the **imbalance** of a partition $\mathscr{P}$ as:

$$I = \frac{\min_{P_i \in \mathscr{P}} D_i}{\max_{P_i \in \mathscr{P}} D_i} = \frac{D_{min}}{D_{max}}. \tag{3}$$

Closer the quantity I is to one, more balanced is the partitioning.

The METIS library (Karypis and Kumar 1999) used to partition the graph requires the graph stored in its *adjacency format*. METIS toolkit also requires a *partitioning* *objective* to be chosen between *edge-cut minimization* and *communication volume minimization*. By adding *weights* to the nodes of a graph we quantify the amount of the computations entrusted to each process of the partition managing those nodes also if an edge-cut minimization is applied, whereas the weights on edges quantify the amount of communication between processes.

3.1 Partitioning Strategies

In this section we focus on the partition of the DFN among the computational processes aiming at maximizing data locality in order to minimize communications and at balancing the computational load among the processes. Six partitioning strategies are presented and analyzed in the following.

3.1.1 Partition Graph Pg and Partition Weighted Graph Wg

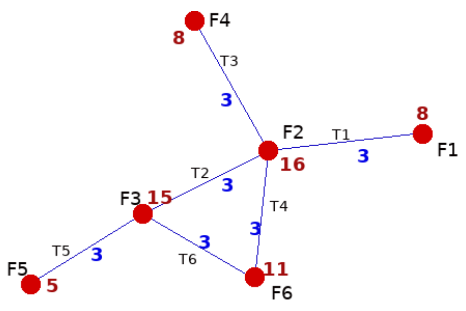


Fig. 7: *Frac6*, adjacency graph representation. F stands for fractures and T for traces. Red numbers are the weights on nodes and blue ones the weights on edges.

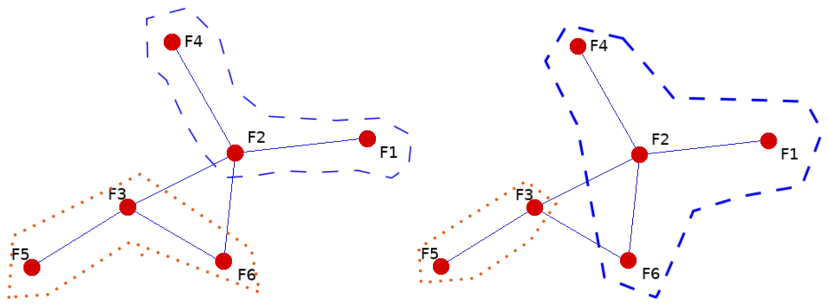
In the simplest graph representation of a DFN the fractures are graph-nodes and the traces are graph-edges (see Figure 7), (Berrone et al. 2015b), (Ushijima-Mwesigwa et al. 2021); weights on nodes and edges are set to 1. We denote this partitioning as Pg (Partition Graph); this approach aims at minimizing the number of the cut traces (cut-minimization), so the number of communicating traces, but the chosen traces to be cut does not necessarily minimize the amount of data communication during the resolution process. Moreover, we do not consider in the partitioning the amount of computations required by each process.

In order to avoid this problem we can resort to a *weighted* partition (Wg) of the graph attaching weights to nodes and edges in the following way:

- $\forall F_r \in \mathscr{F}$ the associated node weight is $[F_r]$;
- $\forall T_m \in \mathscr{T}$ the associated edge weight is $[T_m]$.

By an edge-cut minimization this weighted graph we limit the amount of data communication, whereas the node-weights balance the workload of the processes.

We will present in numerical results that the partition time of the weighted version is almost preserved with respect to the non-weighted version, but the imbalance value I is significantly improved in front of a negligible deterioration of the cut C.



(a) Pg cut minimization partitioning strategy among two processes. (b) Wg weighted partitioning strategy among two processes.

Fig. 8: *Frac6* partitioning strategies comparison. Nodes in the dotted region lie in P_1, dashed ones in P_2.

In Figure 8 an example of the two partitions for *Frac*6 is presented. We note differences on local networks $\mathscr{L}_0$ and $\mathscr{L}_1$. Increasing the number of the fractures the weighting effect becomes more evident.

3.1.2 Partition Bipartite Graph Pb and Partition Bipartite Weighted Graph Wb

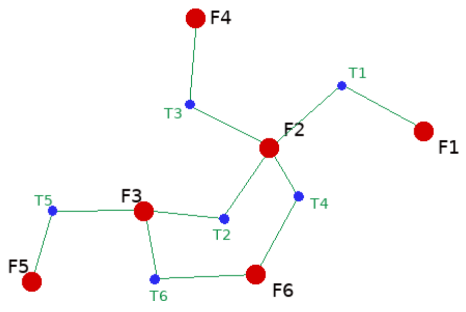


Fig. 9: *Frac6* bipartite graph representation. F stands for fractures and T for traces.

Most of the amount of data communication between processes depend on the number of degrees of freedom on the traces in $\mathscr{C}$. In order to increase the impact of the degrees of freedom on the traces on the partitioning process, we introduce a bipartite graph representing the DFN. In the bipartite graph there are two sets of nodes, the first set representing the fractures and the second one representing the traces. Each element of the first set is only connected with elements of the second one and other way round. The edges correspond to connections between traces and fractures.

When a cut occurs on an edge connecting a trace T_m, with $IT(m) = (r,s)$, and a fracture F_s, the trace T_m is associated to $\mathscr{LC}_i$ and the fracture F_s is associated to $\mathscr{LC}_j$ with $i \neq j$. In Figure 9 we provide the bipartite graph corresponding to the DFN *Frac6*. We denote this partitioning as *Pb* (Partition Bipartite). As before we also explore the advantages of a weighted version when we provide weights on nodes and edges (*Wb* Weighted Bipartite). We assume that the number of degrees of freedom $[T_m]$ on traces is proportional to the number of connections between the trace and the fractures which intersects. We set the edges weights as $[T_m], \forall T_m \in \mathscr{T}$. In this approach we increase the possibility of the partitioning to entrust traces to different processes and in the weighted version we can distinguish the amount of communication related to the connectivity of the Dofs on the trace and the two connected fractures that can be very different when the size of the elements on the two fractures is different. This may happen, for example, when the two fractures have a strong gap in the transmissivity.

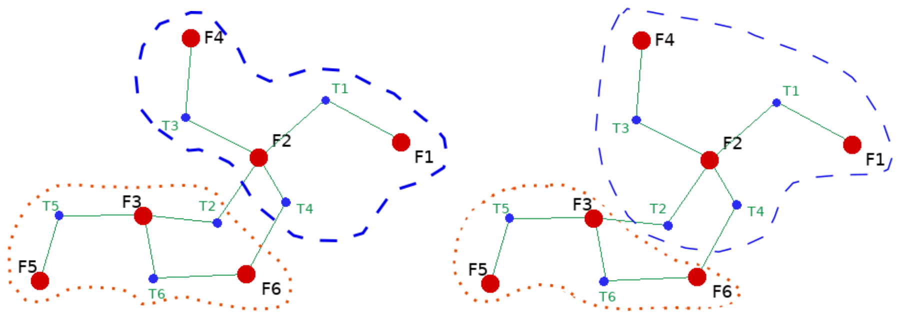


Fig. 10: *Frac6*, *Pb* (left) and *Wb* (right) partitionings. Nodes in the dotted region are assigned to P_1, dashed ones are assigned to P_2.

In Figure 10 *Pb* and *Wb* partitionings are represented for the *Frac*6 example. We observe that both strategies for this very small test produce the same local networks $\mathscr{L}_0$ and $\mathscr{L}_1$, however the METIS assignment of the nodes differs on T_2.

3.1.3 Partition Tripartite Graph Pt and Partition Tripartite Weighted Graph Wt

In this approach we set as nodes of the graph the fractures, the traces and the cross points: they are respectively managed as two, one and zero dimensional domains. We call tripartite graph the new induced graph on which the edges correspond to connections between traces and cross points, traces and fractures, cross points and fractures. This approach has a larger number of nodes and further increase the possibility to distribute nodes among processes, moreover the number of edges generated by this graph representation is larger and can change the Dofs distribution among processes

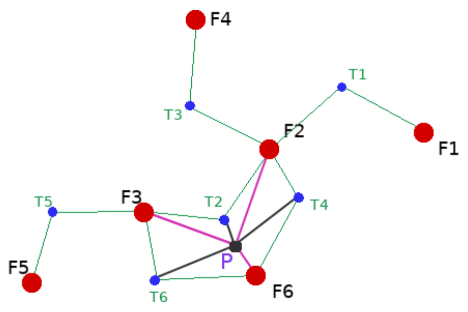


Fig. 11: *Frac6*. Tripartite graph. F denotes fractures, T stands for traces and P corresponds to cross points with multiple intersection. The adjacency matrix considers as connections the green, pink and black edges.

if a partition that minimizes the edges-cut is applied. We notice on the example given in Figure 11 that a cross point is connected with fractures and traces. We denote this partitioning as Pt (Partition Tripartite) in the following.

Its weighted counterpart Wt (Weighted Tripartite) is built such that:

- $\forall F_r \in \mathscr{F}$ the associated node weight is $[F_r]$;
- $\forall T_m \in \mathscr{T}$ the corresponding node weight is $[T_m]$;
- $\forall CP_t \in \mathscr{CP}$ the associated node weights are set to 1;
- for each edge of the graph connecting a trace to a fracture the corresponding weight is $[T_m]$, the number of Dofs on the trace;
- for each edge of the graph connecting a cross point to a fracture, or to a trace, the associated edge weight is $[CP_t] * degree(CP_t)$; $[CP_t] = 1$ is the number of Dofs on cross point, and $degree(CP_t)$ is the degree of the node CP_t in the graph; for example in Figure 11 $degree(CP_t) = 6$ as it intersects three fractures and three traces of the DFN.

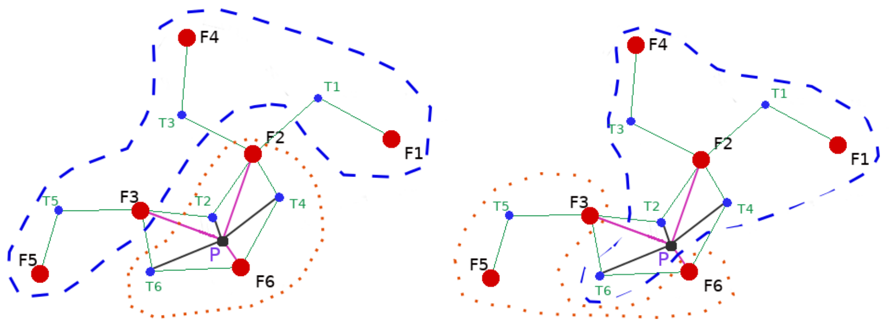


Fig. 12: *Frac6*, Pt (left) and Wt (right) partitionings. Nodes in the dotted region lie in P_1, dashed ones in P_2.

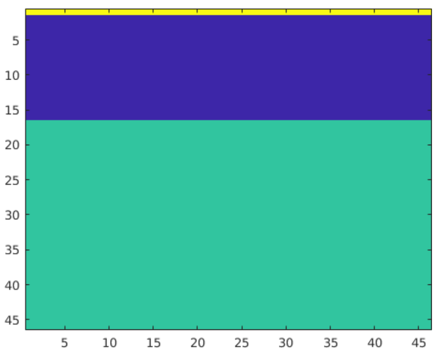
In Figure 12 we present the $Frac6$ partitioning example, the unweighted and weighted versions differ in their local networks $\mathscr{L}_0$ and $\mathscr{L}_1$.

4 Indexing Dofs

In this section we focus on global indexing strategies of the degrees of freedom, unique among all the processes. We present a first serial strategy to enumerate the Dofs of the DFN in Algorithm 1. This strategy is among the simplest to be applied to a DFN, and by changing slightly this approach we provide a parallel efficient global indexing. These algorithms first set indices on domain interfaces of increasing geometrical dimension, then they fill the remaining degrees of freedom with a simple incremental order on fractures.

```
1: foreach $CP_t \in \mathscr{CP}$ do
       Assign dof index $d$. $d++$
   end
2: foreach $T_m \in \mathscr{T}$ do
       foreach node on $T_m$ without index do
           Assign dof index to $d$. $d++$
       end
   end
3: foreach $F_r \in \mathscr{F}$ do
       foreach node on $F_r$ without index do
           Assign dof index to $d$. $d++$
       end
   end
```

Algorithm 1: Degrees of Freedom: Serial Assignment.


(a) Outline structure: the yellow line refers to cross point test functions, the blue stripe to traces and the green one to fractures internal Dofs.

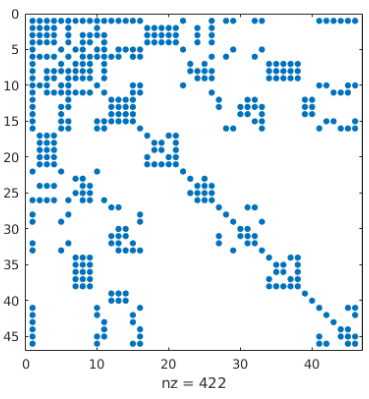
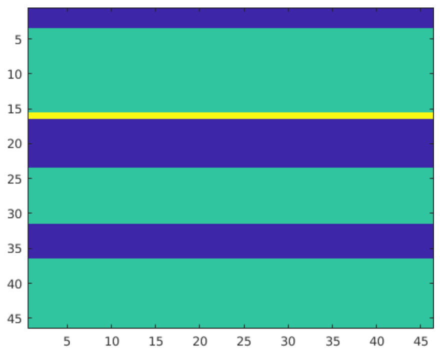

(b) Matrix sparsity pattern.

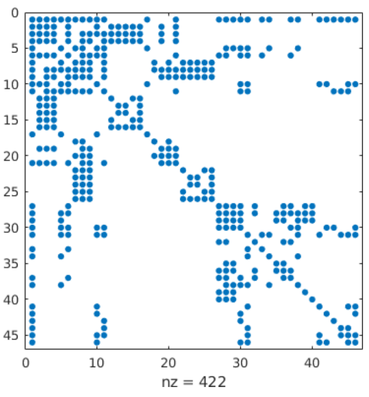
Fig. 13: $Frac6, 46$ degrees of freedom distribution for regular DofHandler.

Given $[\mathscr{T}] - [\mathscr{CP}] = M$ the number of degrees of freedom on traces that are not connected to cross points and $[\mathscr{F}] - M - [\mathscr{CP}] = K$ the internal degrees of freedom on fractures; the resulting sparse matrix has the first $[\mathscr{CP}]$ rows concerning the cross points interactions, then M rows concerning the remaining traces Dofs and the last K rows concerning the remaining fracture Dofs. This strategy agglomerates the most communicating rows in the higher part of the matrix, see Figure 13. This approach is not convenient for a PETSc parallelization (unless different restrictions are imposed) because PETSc subdivides the matrix among the processes in horizontal contiguous stripes. In order to perform the matrix-vector products needed by the PCG method each process requires updating components of the vector that were computed by other processes at the previous iteration. The elements of the vector that are involved in communications are the elements whose indices are outside the diagonal block of the stripe assigned to the process. In Figure 13b the sparsity pattern of the matrix highlights the presence of many nonzero elements outside the diagonal blocks for all the processes. The matrix partition resulting in this case overloads the first process of data communications with almost all the other processes highlighting the negative effect of the latency. The SpeedUp of Krylov subspace methods depends on the global synchronization during the matrix-vector product communication (Saad 2003), (Ghysels et al. 2013), (Kaya et al. 2013).

4.1 The Reordered DofHandler

We propose a reordering for degrees of freedom in order to enumerate consecutively the Dofs of the geometrical objects contained in the local network of each process. Moreover the degrees of freedom on local communicating traces (including Dofs at cross points) are handled by the process that owns them.


(a) Outline structure: the yellow line refers to cross point test functions, the blue stripe to traces and the green one to fractures internal Dofs.



(b) Matrix sparsity pattern.

Fig. 14: $Frac6,46$ degrees of freedom distribution for reordered DofHandler. 3 processes partition.

In Algorithm 2, each process numbers its Dofs with a local numbering, then the local numberings are concatenated in the global indexing. Interfaces indices at traces and cross points are set by the process owning the lowest index fracture and are skipped by the other processes that will inherits the global numbering later. Recalling that $\forall CP_t \in F_r \cap F_s \cap F_q$ the intersection map $ICP(t) = (r,s,q)$ is such that $r < s < q$, and for a trace $T_m \in F_r \cap F_s$ we have $IT(m) = (r,s)$ with $r < s$, the algorithm can be sketched in the following way:

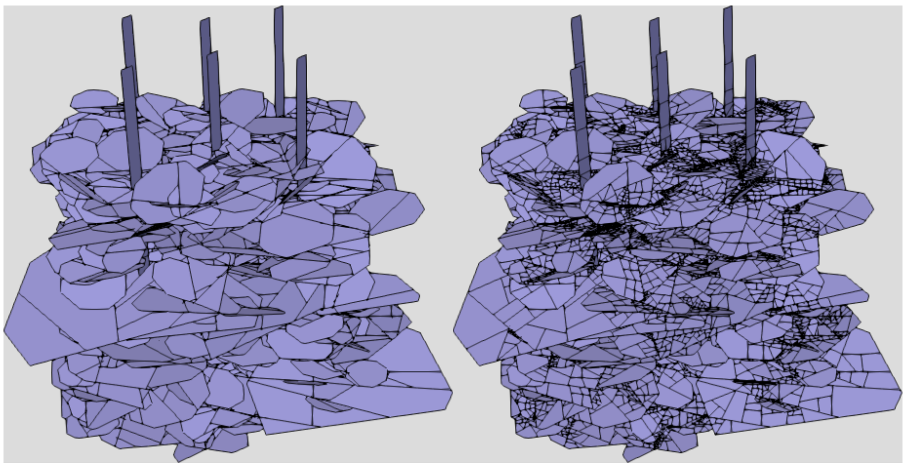
```
Data: Each Process $i$ Call
foreach $CP_t \in \mathscr{CP}$, compute $ICP(t)$ do
    if $F_r \in \mathscr{L}_i$ then
        Assign dof index to $d_{local}$; $d_{local}++$;
end
foreach $T_m \in \mathscr{T}$, compute $IT(m)$ do
    if $F_r \in \mathscr{L}_i$ then
        foreach node on $T_m$ without index do
            Assign dof index to $d_{local}$; $d_{local}++$;
        end
end
foreach $F_k \in \mathscr{L}_i$ do
    foreach node on $F_k$ without index do
        if $F_k \in \mathscr{LC}_i$ then
            Prepare $F_k$ to receive.
        end
        Assign dof index to $d_{local}$; $d_{local}++$;
    end
end
```

Algorithm 2: Pre-communicating phase, degrees of freedom local assignment.

The global indices are then computed from local ones adding to them an offset corresponding to the sum of indices counted on the previous processes. Each process determines the global indices for the degrees of freedom previously set, then indices of Dofs not managed by the process will be received by a different one containing the interface in the local communicating network.

This approach ensures a communicating part of the matrix to each process, balancing the communications required by the solver at each iteration.

In Figure 14 we provide a representation of the matrix previously seen in Figure 13 partitioned among 3 processes by this algorithm, the communicating data (Figure 14a) are sketched in the first part of each horizontal stripe. Moreover, in Figure 14b, the sparsity pattern is presented; we remark that the $Frac6$ is a small test on which the advantages of partitioning and reordering strategies are not evident, however the third processes highlights a larger diagonal local block. Further details are presented for a larger DFN in Section 5.



(a) $Frac512$: Minimal mesh. (b) $Frac512$: refined mesh n=100.

Fig. 15: $Frac512$: initial minimal mesh (left) uniformly refined until the number of degrees of freedom reaches $[\mathscr{F}] \geq n * 512$ (right).

Table 1: DFN Numerical notations.

$[\mathscr{F}]$	Total number of degrees of freedom
$\#\mathscr{S}$	Cardinality of the set $\mathscr{S}$
n	Average number of degrees of freedom required on each fracture
C	Cut
I	Imbalance
$Part.Time(s)$	Partitioning time, in seconds
$Res.Time(s)$	Solution time, in seconds

5 Numerical Results

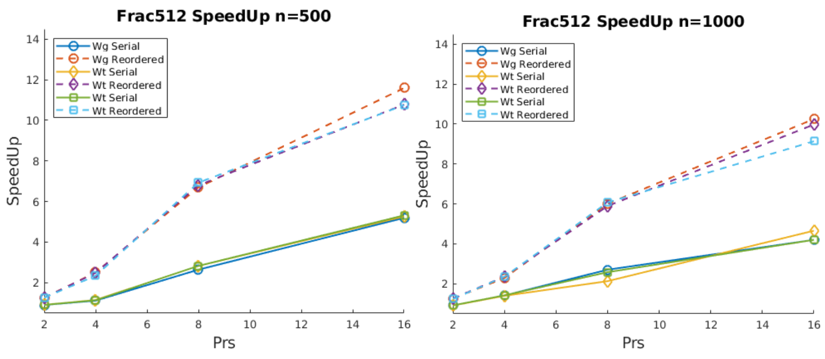
The tests presented in this section concern DFNs with different number of fractures: 512, 1000, 2000 and 4000. We denote the corresponding DFNs as $Frac512$, $Frac1000$, and so on. The linear systems of the tests presented are solved using the PETSc preconditioned conjugate gradient iterative method (*KSPCG*) with a Jacobi diagonal preconditioner. This choice exploits the symmetry and coercivity of the Darcy problem and the corresponding symmetric positive definiteness of the VEM discretization matrix, nevertheless we remark that the partitioning and renumbering methods discussed do not rely on this properties.

5.1 Reordering Analysis

In this section we investigate the performances of the solver coupled to the partitioning strategies introduced in Subsection 5.2 (Table 2) applied to a VEM discretization

Table 2: DFN Partitioning strategies notations.

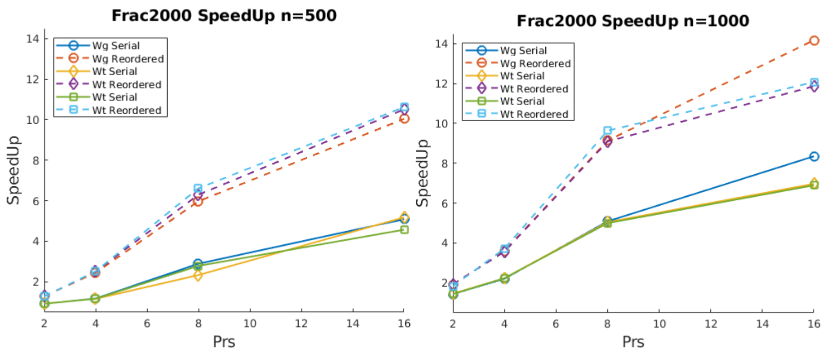
Pg	Partitioning induced graph. The weights on nodes and edges are 1. Fractures as nodes, traces as edges
Wg	Weighted partitioning induced graph Fractures as nodes, traces as edges. The weights are $[F_r]$ and $[T_m]$ on the associated nodes and edges.
Pb	Partitioning bipartite graph. The weights on nodes and edges are 1. Fractures and traces as nodes, connections between fractures and traces are edges
Wb	Weighted partitioning bipartite graph Fractures and traces as nodes, connections between fractures and traces are edges. The weights are $[F_r]$ and $[T_m]$ on the nodes associated to fractures and traces respectively, $[T_m]$ on the edges connecting traces to fractures.
Pt	Partitioning tripartite graph. The weights on nodes and edges are 1. Fractures, traces and cross points as nodes, connections between these objects are edges
Wt	Weighted partitioning tripartite graph Fractures, traces and cross points as nodes, connections between these objects are edges. The weights are: $[F_r]$ on fracture nodes; $[T_m]$ on trace nodes and on edges connecting trace nodes and fracture nodes; 6 on edges connecting cross points to fractures and traces; 1 on nodes representing cross points.



(a) $Frac512\,n = 500$: SpeedUp. (b) $Frac512\,n = 1000$: SpeedUp.

Fig. 16: SpeedUp: $Frac512$, weighted partitioning strategies base, bipartite and tripartite. Comparison of reordered and serial DofHandler numbering strategies. $n = 500$, left, $n = 1000$, right.

to the Darcy problem on fractures. In particular we focus on the different performances when the numbering of the Dofs is performed by the simple Algorithm 1 and by the Algorithm 2. The mesh on which the partition is applied is the minimal mesh (Berrone et al. 2021) on which we refine uniformly until the number of degrees of freedom satisfies $[\mathscr{F}] \geq \#\mathscr{F} * n$, where $n = 500, 1000$. This approach provides a number of degrees of freedom high enough to justify a parallel approach with few processes. In Figure 15 is depicted the final mesh for $Frac512$ with $n = 100$. For this analysis we use the weighted versions of the partitioning strategies aiming at balancing the computational load and minimizing communications among processes.



(a) $Frac2000 n = 500$: SpeedUp. (b) $Frac2000 n = 1000$: SpeedUp.

Fig. 17: SpeedUp: $Frac2000$, weighted partitioning strategies base, bipartite and tripartite. Comparison of reordered and serial dofhandler numbering strategies. $n = 500$, left, $n = 1000$, right.

The Dofs are numbered by the basic sequential Algorithm 1 and by the reordered Algorithm 2, then the resolution times are compared.

Given t_p the resolution time required by p processes, we define the SpeedUp for our parallel resolution as $S_p = \frac{t_1}{t_p}$.

In Figures 16 and 17 we report the SpeedUp for the cases $Frac512$ and $Frac2000$. As expected the reordered dofhandler displays a clear improved behavior. Figures 16 and 17 clearly highlights a loss of parallel performances for the case with 16 processes due to the overloading of shared resources on the CPU. In order to investigate this phenomenon we will present results obtained using different sockets in Subsection 5.3. The tests are performed on Intel Xeon CPUs with 16 cores [1], the 16 processes tests are performed using all the cores of the CPU.

Once the efficiency of reordering strategy is proved in terms of SpeedUp, in the following sections the reordered DofHandler is used. The matrix sparsity pattern is presented in Figure 22.

5.2 Partitioning Analysis

In this section we compare the performances of the six partitioning strategies presented in Section 3 for different DFNs on a mesh with approximately $[\mathscr{F}] \geq \#\mathscr{F} * n$, with $n = 100$ degrees of freedom. Here we only consider the reordered version of the Dofs indices.

In Table 3 we report the resolution times for the serial problems. In Tables 4, 5 and 6 we report the results comparing the three partitioning strategies in their unweighted and weighted versions. In the tables the second column reports the number of processes involved for each test case specified on first column. The columns 3

[1] `https://ark.intel.com/content/www/it/it/ark/products/120492/intel-xeon-gold-6130-processor-22m-cache-2-10-ghz.html`

Table 3: Serial resolution times.

DFN	Res. Time(s)
512	5.24397
1000	15.2265
2000	13.1338
4000	57.7411

Table 4: Partition results for graph partitioning Pg and the weighted one Wg.

DFN	**Prs**	C		I		Part. Time(s)		Res. Time(s)	
		Pg	Wg	Pg	Wg	Pg	Wg	Pg	Wg
512	2	38	50	0.827	0.970	0.020	0.022	2.676	2.645
512	4	90	125	0.642	0.973	0.023	0.026	1.818	1.403
512	8	149	166	0.549	0.918	0.025	0.025	0.903	0.730
512	16	360	275	0.534	0.884	0.054	0.028	0.479	0.420
1000	2	34	45	0.639	0.944	0.036	0.038	8.849	7.976
1000	4	78	112	0.599	0.933	0.041	0.050	4.840	3.991
1000	8	130	169	0.512	0.910	0.058	0.071	2.820	2.129
1000	16	213	296	0.385	0.812	0.081	0.102	1.372	1.191
2000	2	88	93	0.969	0.955	0.057	0.058	6.726	6.794
2000	4	152	227	0.884	0.945	0.060	0.067	3.482	3.618
2000	8	293	339	0.760	0.941	0.067	0.068	1.979	1.857
2000	16	455	551	0.608	0.890	0.084	0.076	1.092	1.000
4000	2	86	101	0.901	0.965	0.118	0.155	28.971	28.583
4000	4	177	204	0.827	0.970	0.120	0.128	15.022	14.141
4000	8	372	501	0.731	0.930	0.136	0.144	8.200	7.608
4000	16	556	652	0.537	0.950	0.137	0.141	5.427	4.629

and 4 report the cut C for unweighted and weighted partitioning, respectively. The columns 5 and 6 the imbalance I. $Part.Time(s)$ on columns 7 and 8 is the partitioning time employed by METIS, the last two columns report the resolution time employed by the PETSc.

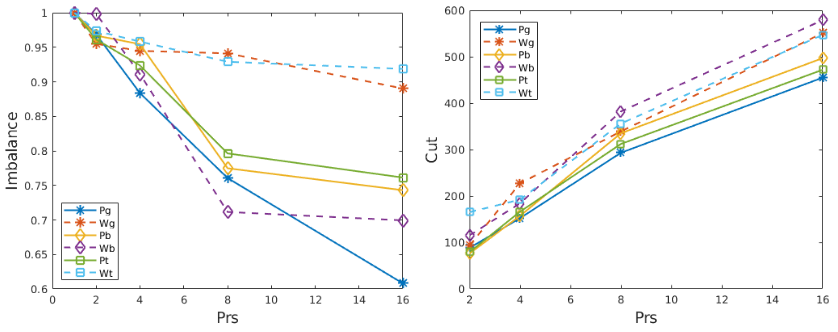


Fig. 18: $Frac2000$: imbalance (left) and cut (right) increasing the number of processes, $n = 100$.

Table 5: Partition results for bipartite partitioning Pb and the weighted one Wb.

DFN	**Prs**	C		I		Part. Time(s)		Res. Time(s)	
		Pb	Wb	Pb	Wb	Pb	Wb	Pb	Wb
512	2	40	60	0.864	0.988	0.020	0.022	2.682	2.648
512	4	80	151	0.783	0.937	0.022	0.028	1.612	1.412
512	8	145	219	0.701	0.887	0.024	0.030	0.816	0.748
512	16	211	290	0.500	0.836	0.026	0.032	0.474	0.452
1000	2	45	50	0.888	0.980	0.039	0.039	8.122	7.959
1000	4	90	102	0.786	0.943	0.041	0.053	4.090	4.001
1000	8	140	180	0.711	0.896	0.053	0.052	2.389	2.144
1000	16	239	315	0.553	0.863	0.062	0.070	1.399	1.245
2000	2	76	93	0.967	0.962	0.056	0.057	7.006	6.782
2000	4	157	226	0.954	0.952	0.061	0.065	3.336	3.379
2000	8	335	420	0.775	0.914	0.063	0.073	1.821	1.837
2000	16	497	611	0.743	0.867	0.079	0.080	1.060	1.008
4000	2	83	92	0.992	0.986	0.118	0.119	28.428	28.045
4000	4	176	195	0.859	0.936	0.121	0.124	14.781	14.699
4000	8	348	494	0.820	0.929	0.130	0.138	7.512	7.303
4000	16	603	767	0.676	0.902	0.135	0.155	5.070	4.663

Table 6: Partition results for tripartite partitioning Pt and the weighted one Wt.

DFN	**Prs**	C		I		Part. Time(s)		Res. Time(s)	
		Pt	Wt	Pt	Wt	Pt	Wt	Pt	Wt
512	2	50	70	0.872	0.986	0.021	0.024	2.752	2.645
512	4	93	143	0.801	0.963	0.022	0.026	1.579	1.368
512	8	136	136	0.864	0.864	0.025	0.025	0.816	0.741
512	16	219	290	0.466	0.836	0.027	0.032	0.478	0.452
1000	2	45	45	0.886	0.961	0.037	0.039	8.004	8.192
1000	4	87	112	0.805	0.947	0.040	0.057	4.448	4.018
1000	8	140	189	0.760	0.896	0.053	0.064	2.402	2.203
1000	16	232	308	0.594	0.863	0.063	0.058	1.360	1.132
2000	2	79	165	0.961	0.974	0.055	0.065	6.628	6.922
2000	4	166	192	0.924	0.958	0.062	0.064	3.361	3.501
2000	8	312	356	0.796	0.929	0.066	0.069	2.029	1.827
2000	16	472	547	0.761	0.919	0.073	0.080	1.091	0.962
4000	2	83	98	0.973	0.979	0.117	0.118	30.125	31.120
4000	4	175	282	0.934	0.951	0.121	0.134	14.415	13.857
4000	8	358	453	0.826	0.927	0.129	0.138	7.543	7.215
4000	16	577	728	0.743	0.880	0.138	0.149	5.225	5.106

For each partitioning strategy, comparing the results of the unweighted and weighted versions we can see that the cut C is increasing for the weighted versions because the partitioner aims at minimizing the number of degrees of freedom on the cut traces instead of the number of the cut traces. The cut C represents the number of communicating traces of the DFN, that are edges of the graph representing the DFN only for the Pg and Wg strategies. In the weighted version the cuts concentrate on more shorter traces. We note that an increased number of communicating traces arises in the bipartite and tripartite case, as long as they focus on the data load and not on the quantity of the cut edges.

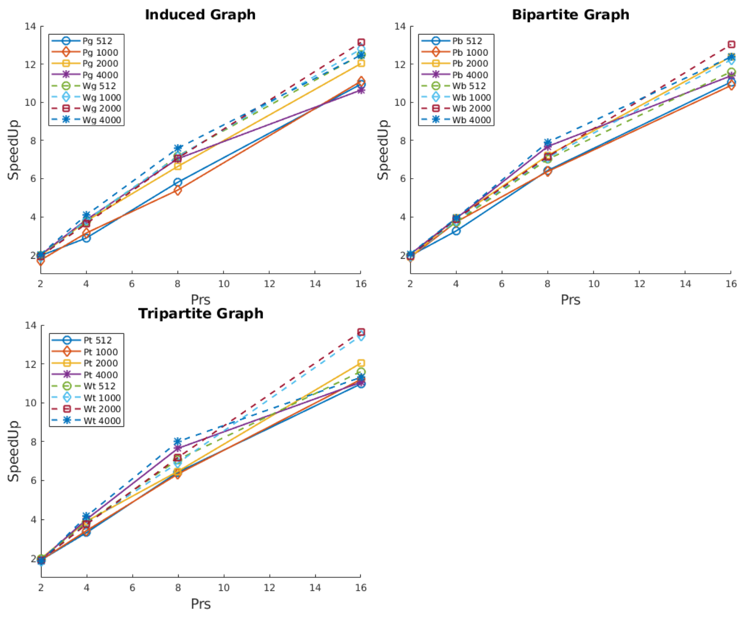


Fig. 19: SpeedUp comparison among partitioning strategies. The average number of degrees of freedom on each fracture is $n = 100$.

Concerning the imbalance I, we see that the weighted versions have an imbalance closer to one and less dependent on the number of processes, that denotes a more uniform distribution of the workload among the processes. In Figure 18 we report for $Frac2000$ plots of imbalance for all the partitioning strategies, the advantages in terms of Dofs imbalance is evident for base graph and tripartite graph in front of a negligible loss for the cut C.

From columns $7-8$ $(Part.Time(s))$ we can see that the partitioning time is almost equal for all the partitioning strategies.

In Figure 19 we compare the SpeedUp for the weighted and non-weighted strategies, we can observe that they are quite similar and the weighted version behaves slightly better. In this particular case the total number of Dofs is not so large to overload the shared resources of the CPU and the degradation seen in Figure 17 is slightly appreciable for the $Frac4000$.

5.3 Multi-Sockets Analysis

As already noted in the previous sections when the number of partitions is higher than 8 and the number of Dofs is quite large our tests clearly highlight a loss of SpeedUp. This phenomenon can be attributed to an overloading of the common resources of the CPUs. In order to confirm this interpretation we repeat the same tests entrusting only a process to each CPU. Being the communications between different sockets more expensive (Balay et al. 2019) we focus on a larger problem that is the $Frac4000$ on

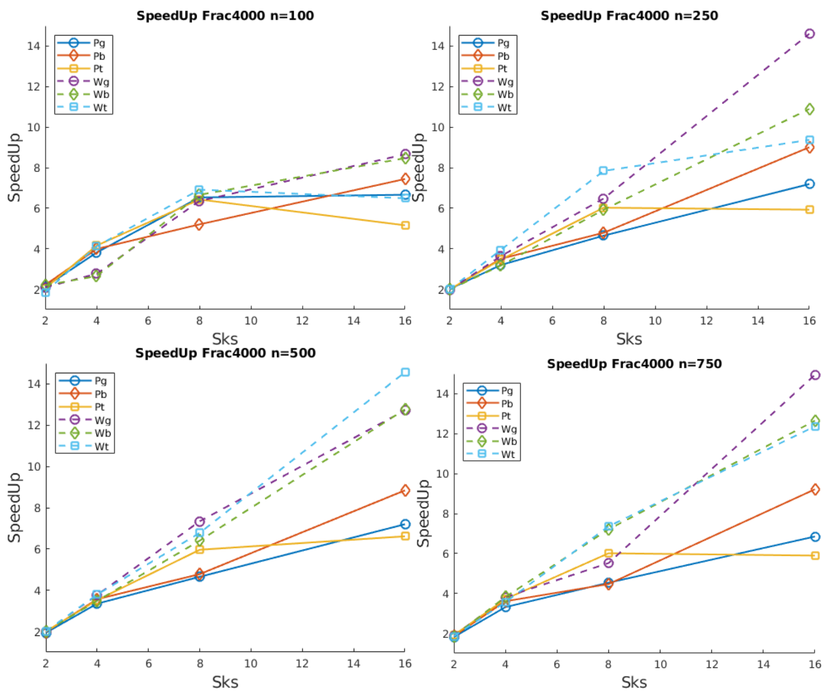


Fig. 20: Multi-socket SpeedUp for $Frac4000$, $[\mathscr{F}]$: 332174, 1260833 , 1889468 and 2789051. The average number of degrees of freedom on each fracture is n; partitioning strategies comparison for their weighted and unweighted versions.

which we increase the number of degrees of freedom $[\mathscr{F}] \geq \#\mathscr{F} * n$ from $n = 100$ to $n = 250$, $n = 500$ and $n = 750$. In Figure 20 we report the SpeedUp results, that highlight better performances increasing the number of degrees of freedom due to the increased workload of the processes with respect to the cost of communications. In this case the degeneration of performances passing from 8 to 16 processes is less relevant because there is not competition between processes in the use of the shared resources of the CPU.

5.4 Mesh Partitioning

In this section we consider the standard partitioning of the Dofs based on the connectivity of the mesh. In particular we construct the adjacency matrix of the graph of the degrees of freedom, i.e., we have one graph-node for each mesh-node (Dof) and one graph-edge for each mesh-edge. We apply a partitioning of this graph using a cut-edge minimization strategy; with this approach the amount of data communicated at each iteration among the processes should be minimized.

The partitioning of the Dofs is handled by METIS. Then the resulting set of indices on each process is numbered in a contiguous way (see Figures 22a, 22b).

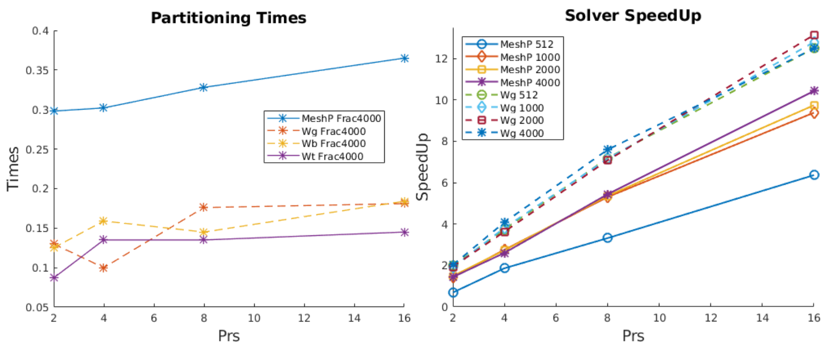
The partitioning time employed by METIS for this approach is much higher with respect to the ones of the previously presented partitioning strategies due to the larger graph to be partitioned, see Figure 21a. Nevertheless, the solver SpeedUp is quite similar, see Figure 21b. When the number of processes increases the DFN-based graph partitionings present a clear better overall behavior, (Ushijima-Mwesigwa et al. 2021). In Table 7 we report the partitioning time and the resolution time for the same mesh ($n = 100$) used for the test cases of Tables 4, 5 and 6. We can observe that the resolution time for the mesh partitioning is always larger with respect to the proposed partitioning strategies. This behavior can be explained observing the sparsity pattern of the matrix. In Figure 22 we report the spy for the serial DofHandler, for the mesh partitioning and for the weighted graph partitionings proposed. The large number of off-diagonal block elements of the mesh partitioning approach is responsible for a lower efficiency of the matrix vector products, whereas in the reordered weighted versions we have similar structures with marginal differences and a higher clustering of the nonzeros elements on the rows that helps improving the matrix vector product.

Table 7: Mesh Partitioning.

DFN	**Prs**	Part. Time(s)	Res.Time(s)
512	1	0	5.025
512	2	0.052	3.564
512	4	0.055	1.836
512	8	0.060	1.010
512	16	0.068	0.570
1000	1	0	14.789
1000	2	0.102	10.239
1000	4	0.105	5.363
1000	8	0.112	2.789
1000	16	0.135	1.577
2000	1	0	12.648
2000	2	0.133	8.517
2000	4	0.146	4.638
2000	8	0.146	2.360
2000	16	0.170	1.299
4000	1	0	56.103
4000	2	0.298	39.084
4000	4	0.302	21.548
4000	8	0.328	10.316
4000	16	0.365	5.382

5.5 An adaptive VEM mesh refinement test

In this last section we investigate the interplay between partitioning and mesh refinement, and, in particular, the degeneration of the performances of the proposed partitioning strategies when the partitioning is performed on a starting mesh that is



(a) *Frac*4000: Partitioning times employed by METIS comparison.
(b) Mesh partition SpeedUp. Comparison with weighted graph partition.

Fig. 21: Induced mesh partitioning results for partitioning times and solver SpeedUp.

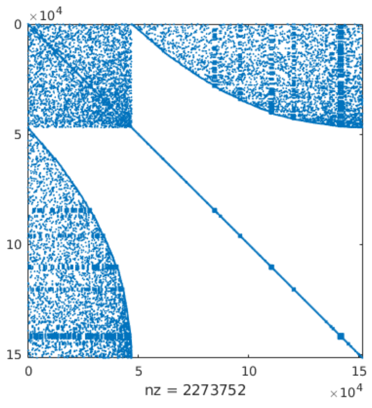
subject to an iterative mesh refinement based on *a posteriori* error estimates, (Berrone and Borio 2017b). In Figure 23 the convergence curves of the relative error are reported on the left; on the right the growth of Dofs number due to the mesh refinement, (Berrone et al. 2021). As long as it is not possible to provide the adaptive refinement *a priori*, a computationally costly, but performing approach would apply a partitioning of the DFN during each refinement step. Another approach is to partition the DFN every fixed predefined refinement steps (for example each five refinements). However we aim at observing the behavior of our partitioning strategies without a re-partitioning phase to see how rapidly the initial partitioning degenerate. We apply the partitioning strategies on the minimal mesh at the beginning of the computation. We do not expect, as long as the refining is not uniform, that the imbalance can remain constant during the resolution. Curves presented in Figure 24 suggest to apply a re-partitioning each five refinement iterations.

6 Conclusions

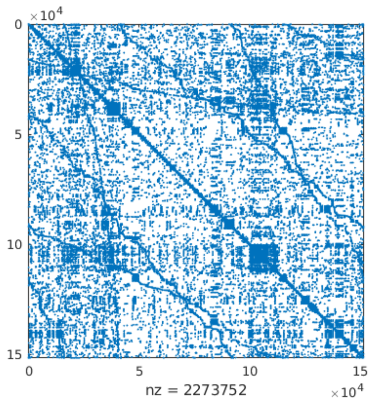
In this article we present three partitioning strategies for the DFN flow simulations. For each of them, we analyze several performances parameters, among them the cut, the imbalance, the partitioning time and the resolution time. The weighted partitioning strategies in general perform better, moreover they have similar behavior, with in general slight loss in performances for the bipartite graph partitioning. The proposed partitioning strategies are computationally cheaper than the classical mesh induced one in partitioning time and perform better in resolution time.

The reordering aims at avoiding that one or few processes are overloaded of communications with respect to the other processes, trying to equally distribute the communication among all the processes as presented in Section 4.

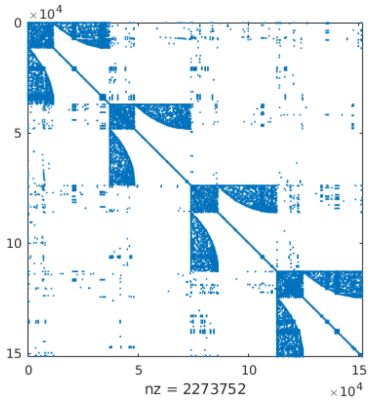
For problems with a large number of degrees of freedom we also compare the SpeedUp on processes running on the same socket and on different CPUs clearly highlighting a degradation of the SpeedUp when the CPU and memory access re-



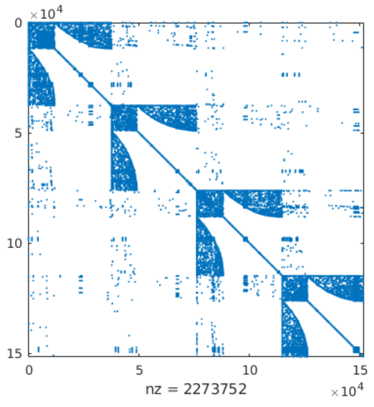
(a) Serial DofHandler matrix structure.



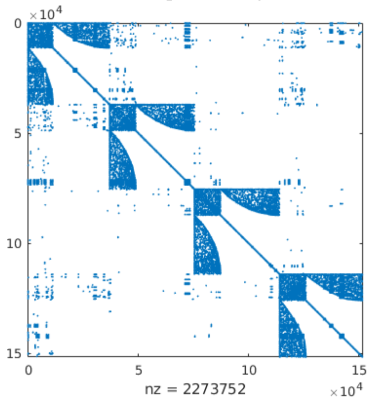
(b) *MeshP* partitioning.



(c) *Wg* partitioning.



(d) *Wb* partitioning.



(e) *Wt* partitioning.

Fig. 22: *Frac*2000: Sparsity pattern. Partitioning among 4 processes

sources are overloaded. The proposed methods preserve the SpeedUp with a higher dimension of the DFN problem.

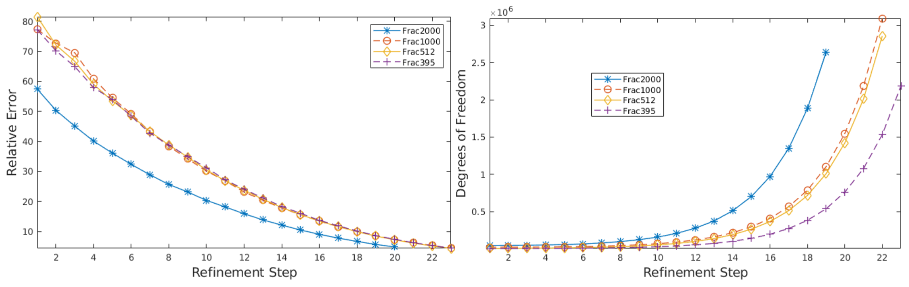


Fig. 23: Relative error (left) and increasing degrees of freedom (right) during an adaptive mesh refinement.

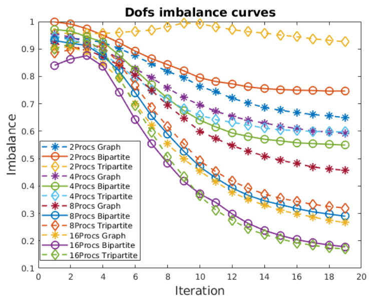


Fig. 24: *Frac*2000: imbalance curves during VEM resolution.

Acknowledgements This work is supported by the MIUR project “Dipartimenti di Eccellenza 2018-2022” (CUP E11G18000350001), PRIN project “Virtual Element Methods: Analysis and Applications” (201744KLJL_004) and by INdAM-GNCS. Computational resources supported by HPC@polito.it and SmartData@polito.

References

Ahmad B, Alsaedi A, Brezzi F, Marini L D, Russo A (2013) Equivalent projectors for virtual element methods. Computers & Mathematics with Applications 66:376–391

Balay S, Abhyankar S, Adams M, Brown J, Brune P, Buschelman K, Dalcin L, Dener A, Eijkhout V, Gropp W, et al. (2019) Petsc users manual

Beirão da Veiga L, Brezzi F, Cangiani A, Manzini G, Marini L D, Russo A (2013) Basic principles of virtual element methods. Mathematical Models and Methods in Applied Sciences 23(01):199–214

Beirão da Veiga L, Brezzi F, Marini L D, Russo A (2014) The hitchhiker’s guide to the virtual element method. Mathematical Models and Methods in Applied Sciences 24(08):1541–1573

Beirão da Veiga L, Brezzi F, Marini L D, Russo A (2015) Virtual element methods for general second order elliptic problems on polygonal meshes. Mathematical Models and Methods in Applied Sciences 26(04):729–750

Benedetto M F, Berrone S, Borio A, Pieraccini S, Scialò S (2016a) A hybrid mortar virtual element method for discrete fracture network simulations. Journal of Computational Physics 306:148–166

Benedetto M F, Berrone S, Scialò S (2016b) A globally conforming method for solving flow in discrete fracture networks using the virtual element method. Finite Elements in Analysis and Design 109:23–36

Berrone S, Borio A (2017a) Orthogonal polynomials in badly shaped polygonal elements for the Virtual Element Method. Finite Elements in Analysis & Design 129:14–31, ISSN 0168-874X

Berrone S, Borio A (2017b) A residual *a posteriori* error estimate for the Virtual Element Method. Math Models Methods Appl Sci 27(8):1423–1458, ISSN 0218-2025

Berrone S, Borio A, D'Auria A (2021) Refinement strategies for polygonal meshes applied to adaptive vem discretization. Finite Elements in Analysis and Design 186:103502, ISSN 0168-874X

Berrone S, Canuto C, Pieraccini S, Scialò S (2015a) Uncertainty quantification in discrete fracture network models: Stochastic fracture transmissivity. Computers & Mathematics with Applications 70(4):603–623, ISSN 0898-1221

Berrone S, Canuto C, Pieraccini S, Scialò S (2018) Uncertainty quantification in discrete fracture network models: Stochastic geometry. Water Resources Research 54(2):1338–1352

Berrone S, Pieraccini S, Scialò S (2017) Non-stationary transport phenomena in networks of fractures: effective simulations and stochastic analysis. Computer Methods in Applied Mechanics and Engineering 315:1098–1112

Berrone S, Pieraccini S, Scialo S, Vicini F (2015b) A parallel solver for large scale dfn flow simulations. SIAM Journal on Scientific Computing 37(3):C285–C306

Berrone S, Scialó S, Vicini F (2019) Parallel meshing, discretization, and computation of flow in massive discrete fracture networks. SIAM Journal on Scientific Computing 41(4):C317–C338

Cacas M C, Ledoux E, de Marsily G, Tillie B, Barbreau A, Durand E, Feuga B, Peaudecerf P (1990) Modeling fracture flow with a stochastic discrete fracture network: calibration and validation: 1. the flow model. Water Resources Research 26(3):479–489

Canuto C, Pieraccini S, Xiu D (2019) Uncertainty quantification of discontinuous outputs via a non-intrusive bifidelity strategy. Journal of Computational Physics 398:108885

Council N R, et al. (1996) Rock fractures and fluid flow: contemporary understanding and applications. National Academies Press

Dershowitz W, Fidelibus C (1999) Derivation of equivalent pipe network analogues for three-dimensional discrete fracture networks by the boundary element method. Water Resources Research 35(9):2685–2691

Fidelibus C (2007) The 2d hydro-mechanically coupled response of a rock mass with fractures via a mixed bem–fem technique. International journal for numerical and analytical methods in geomechanics 31(11):1329–1348

Ghysels P, Ashby T J, Meerbergen K, Vanroose W (2013) Hiding global communication latency in the gmres algorithm on massively parallel machines. SIAM Journal on Scientific Computing 35(1):C48–C71

Hendrickson B, Leland R W (1995) A multi-level algorithm for partitioning graphs. SC 95(28):1–14

Hyman J D, Gable C W, Painter S L, Makedonska N (2014) Conforming delaunay triangulation of stochastically generated three dimensional discrete fracture networks: A feature rejection algorithm for meshing strategy. SIAM Journal on Scientific Computing 36(4):A1871–A1894

Jaffré J, Roberts J E (2012) Modeling flow in porous media with fractures; discrete fracture models with matrix-fracture exchange. Numerical Analysis and Applications 5(2):162–167

Karypis G, Kumar V (1999) A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1):359–392

Kaya K, Uçar B, Çatalyürek Ü V (2013) Analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication. In International Conference on Parallel Processing and Applied Mathematics, Springer, 174–184

Lei Q, Latham J P, Tsang C F (2017) The use of discrete fracture networks for modelling coupled geomechanical and hydrological behaviour of fractured rocks. Computers and Geotechnics 85:151–176

Mustapha H, Mustapha K (2007) A new approach to simulating flow in discrete fracture networks with an optimized mesh. SIAM Journal on Scientific Computing 29(4):1439–1459

Neuman S P (2005) Trends, prospects and challenges in quantifying flow and transport through fractured rocks. Hydrogeology Journal 13(1):124–147

Nœtinger B, Jarrige N (2012) A quasi steady state method for solving transient darcy flow in complex 3d fractured networks. Journal of Computational Physics 231(1):23–38

Nordqvist A W, Tsang Y, Tsang C, Dverstorp B, Andersson J (1992) A variable aperture fracture network model for flow and transport in fractured rocks. Water Resources Research 28(6):1703–1713

Pichot G, Erhel J, de Dreuzy J R (2010) A mixed hybrid mortar method for solving flow in discrete fracture networks. Applicable Analysis 89(10):1629–1643

Pieraccini S (2020) Uncertainty quantification analysis in discrete fracture network flow simulations. GEM-International Journal on Geomathematics 11(1):1–21

Saad Y (2003) Iterative methods for sparse linear systems. SIAM

Sentís M L, Gable C W (2017) Coupling lagrit unstructured mesh generation and model setup with tough2 flow and transport: a case study. Computers & Geosciences 108:42–49

Ushijima-Mwesigwa H, Hyman J D, Hagberg A, Safro I, Karra S, Gable C W, Sweeney M R, Srinivasan G (2021) Multilevel graph partitioning for three-dimensional discrete fracture network flow simulations. Mathematical Geosciences :1–26

Vohralik M, Maryška J, Severỳn O (2007) Mixed and nonconforming finite element methods on a system of polygons. Applied numerical mathematics 57(2):176–193