Parallel Computing 39 (2013) 319–335

Contents lists available at SciVerse ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A new metric enabling an exact hypergraph model for the communication volume in distributed-memory parallel applications

O. Fortmeier [a,*], H.M. Bücker [a,1], B.O. Fagginger Auer [b], R.H. Bisseling [b]

[a] Institute for Scientific Computing, RWTH Aachen University, D-52056 Aachen, Germany
[b] Mathematical Institute, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands

ARTICLE INFO

Article history:
Received 28 February 2012
Received in revised form 5 May 2013
Accepted 21 May 2013
Available online 2 June 2013

Keywords:
Hypergraph
Unstructured triangulation
Communication volume metric
Finite elements
Combinatorial scientific computing

ABSTRACT

A hypergraph model for mapping applications with an all-neighbor communication pattern to distributed-memory computers is proposed, which originated in finite element triangulations. Rather than approximating the communication volume for linear algebra operations, this new model represents the communication volume exactly. To this end, a hypergraph partitioning problem is formulated where the objective function involves a new metric. This metric, the $\lambda(\lambda-1)$-metric, accurately models the communication volume for an all-neighbor communication pattern occurring in a concrete finite element application. It is a member of a more general class of metrics, which also contains more widely used metrics, such as the cut–net and the $(\lambda-1)$-metric. In addition, we develop a heuristic to minimize the communication volume in the new $\lambda(\lambda-1)$-metric. For the solution of several real-world finite element problems, experimental results based on this new heuristic demonstrate a small reduction in communication volume compared to a standard graph partitioner and do not show significant reductions in communication volume compared to a hypergraph partitioner using the common $(\lambda-1)$-metric. However, for this set of problems, the new approach does reduce actual communication times. As a by-product, we observe that it also tends to reduce the number of messages. Furthermore, the new approach dramatically reduces the communication volume for a set of sparse matrix problems that are more irregularly-structured than finite element problems.

1. Introduction

A parallel implementation of a finite element method on a distributed-memory computer demands a mapping of the underlying triangulation to the processes. The goal of this distribution is to minimize the data communication necessary between the processes when computation is carried out using this finite element triangulation. The distribution is only acceptable if the resulting computational load is evenly balanced among all participating processes. Modeling this data distribution as a graph partitioning problem has a long tradition in parallel computing; see the survey [1] and the references therein. In such an undirected graph model, the vertices represent the computational tasks while the edges represent the dependencies between these tasks. An edge in a graph connects exactly two vertices. In a hypergraph model, this is extended by hyperedges that connect any number of vertices and hence are capable of expressing more general connectivity

* Corresponding author. Present address: Bull GmbH, D-51149 Cologne, Germany. Tel.: +49 1735887589; fax: +49 22033051845.
E-mail address: fortmeier@sc.rwth-aachen.de (O. Fortmeier).
[1] Present address: Institute for Computer Science, Friedrich Schiller University Jena, D-07743 Jena, Germany.

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.05.003

information. This additional flexibility of the hypergraph model is exploited in various application areas of parallel computing including sparse matrix–vector multiplication [2], volume rendering [3], and scheduling [4]. Previous work on representing aspects of finite element triangulations using hypergraph models includes partitioning followed by local reorderings within each resulting part of the partitioning in an attempt to improve data locality [5].

Standard techniques for distributing finite element triangulations on distributed-memory computers focus on parallel sparse matrix–vector multiplication which is often the most time-consuming operation of a finite element method. Though, in practice, a large number of finite element codes carry out this operation differently, these approaches conceptually rely on assembling the stiffness matrix and employing any graph or hypergraph partitioning for the resulting sparse matrix–vector multiplication. Another class of techniques is based on performing graph partitioning on the dual graph of the finite element triangulation. Here, the dual graph represents the elements of the triangulations and their adjacencies. While both these classes of techniques have been successfully used in practice, they fall short in modeling the communication volume exactly. The communication volume is defined as the number of data words to be communicated between processes when computations are carried out using the underlying finite element triangulation. While all established models approximate the "true" communication volume, the first new contribution of the present paper is to develop a hypergraph model that represents the communication volume exactly. When preparing the revised version of this paper, we found out that an exact hypergraph model for a mesh is also mentioned in [6,7]. This model uses the $(\lambda-1)$-metric as described below and is available via a load balancing service called iZoltan developed within the Interoperable Technologies for Advanced Petascale Simulations project, defining an abstract data model and interfaces for parallel mesh data [8]. The mesh-based iZoltan service uses Zoltan [9]. Our work which was carried out independently from this related work is inspired by analyzing the communication pattern occurring in the parallel finite element solver Drops [10]. Previous approaches based on the undirected graph model [11,12] were not successful in modeling the communication volume of that software accurately.

The finite element software package Drops is being developed for the solution of two-phase flow problems where two immiscible fluids are interacting in three space dimensions. Drops is based on the level-set approach to capture the time-dependent boundary between the two fluids and discretizes the flow and the level set function on an adaptively refined unstructured tetrahedral mesh. Though adaptivity is an important issue in real-world finite element computations, we do not consider hypergraph models for dynamic partitioning [13,14]. By focusing on a hypergraph model for static partitioning, we defer the extension from static to dynamic partitioning for future research. An overview of algorithms and techniques to dynamically partition application data and work among processes for numerically solving partial differential equations is given in [15].

There are different communication volume metrics used in hypergraph partitioning models. The $(\lambda-1)$-metric is widely used for sparse matrix–vector multiplication [2]. For the distribution of finite element triangulations, we show that the $(\lambda-1)$-metric exactly models the communication volume when each data item is kept by a single owner during the computation and communicated when needed. However, it fails to model the communication volume for implementations like Drops where all neighbors share data among each other and communicate in an all-to-all fashion. To reflect this situation, we propose the novel $\lambda(\lambda-1)$-metric, which models the number of communicated data words exactly in that case. The new metric falls into the category *partitioning for complex objectives*, a term coined by Pinar and Hendrickson in 2001 [16]. Another recent example of complex objectives is the work by Kaya et al. [17] who try to achieve balance on the number of nonzeros on certain parts of sparse matrices. In this paper, we introduce the $\lambda(\lambda-1)$-metric, as well as a bipartitioning heuristic for the solution of the hypergraph partitioning problem with this new metric as the objective function.

In Section 2, we discuss the finite element triangulation and the different types of communication patterns required by the parallel software package Drops. Minimizing the number of communicated data words while, at the same time, balancing the computational load is reformulated as a hypergraph partitioning problem in Section 3. In Section 4, we show that such partitioning problems, with general communication volumes, can be minimized greedily using a standard hypergraph partitioner. To measure the performance of this strategy, we partition a finite element triangulation from a real-world application taken from [18] in Section 5 and measure the resulting communication volumes and communication times of Drops for up to 1024 processes. We compare partitionings obtained by the Mondriaan hypergraph partitioner [19] together with PaToH [20] to those generated by partitioning the undirected graph model using METIS [21]. Our new hypergraph model for the communication volume leads to communication volumes that are smaller than the undirected graph model. Furthermore, it tends to reduce the number of messages sent among all processes and also results in lower communication times.

2. Distribution of finite element triangulations

Triangulations of computational domains are crucial for various numerical techniques in scientific computing, in particular for finite element methods. In this section, we first consider triangulations with a focus on how they are represented on distributed-memory parallel computers. We then discuss the parallel execution of finite element computations on these triangulations with a focus on the communication volume. Afterward, we formulate a problem describing a "perfect" distribution of these triangulations in the sense of minimizing communication volume while balancing computational load. We stress that the aim of this section is to describe the communication volume of the finite element software package Drops exactly rather than approximately. Since the techniques implemented in Drops are also used elsewhere, the discussion is

intentionally formulated in a general notation. In the final subsection, we also give some remarks on implementations that differ from DROPS.

2.1. Distributed triangulations

Throughout this paper, we consider a triangulation

$$T = \{t_1, \ldots, t_K\}$$

of a three-dimensional domain $\Omega \subset \mathbb{R}^3$ by a tetrahedral mesh. The mesh consists of K tetrahedra t_i, whose neighborship relation is defined by the topological adjacency in the mesh. A tetrahedron consists of nodes, edges, and faces. Let

$$N = \{n_1, \ldots, n_L\}$$

denote the set of nodes and edges of the triangulation. We refer to all elements of N as "nodes" without distinguishing whether these are actually nodes or edges. The face between a tetrahedron t_i and a neighboring tetrahedron t_j is denoted by $t_i \cap t_j$.

We assume that the triangulation is represented by separate data structures for the tetrahedra, nodes, and faces which are distributed among $P \geqslant 2$ processes. We further assume that each tetrahedron $t \in T$ is stored by a unique process $1 \leqslant p \leqslant P$. That is, there is no tetrahedron stored by more than one process. Let the symbol T_p denote the set of tetrahedra which is stored by process p. Since a node or a face can belong to more than one tetrahedron, they are distributed differently than the tetrahedra. While a tetrahedron is assigned to exactly one process, nodes and faces can be assigned to multiple processes, yielding an overlap at process boundaries as illustrated in Fig. 1. The discussions and results given in this paper apply to three spatial dimensions; for clarity, the example in Fig. 1 is two-dimensional. This example illustrates the distribution of the triangulation $T = \{t_1, t_2, \ldots, t_{10}\}$ with $K = 10$ tetrahedra among $P = 3$ processes p_1, p_2, and p_3. The process boundaries between these sets of tetrahedra T_1, T_2, and T_3 consist of faces and nodes. For instance, the boundary of the two processes p_1 and p_2 is given by the two faces $t_1 \cap t_2$ and $t_7 \cap t_8$ and the three nodes n_1, n_5, and n_9. The process boundary between the two processes p_1 and p_3 is solely described by the node n_9.

This example illustrates the distribution of tetrahedra among processes which is formally described by the following definition.

Definition 1 (P-way Tetrahedron Distribution). Given a triangulation T and a number of processes $P \geqslant 2$, a distribution of its tetrahedra among P processes denoted by $\mathcal{T} := \{T_1, \ldots, T_P\}$ is called a *P-way tetrahedron distribution of T* if the following three conditions hold:

1. Each process stores one or more tetrahedra, i.e.,

   $$T_p \subset T \text{ and } T_p \neq \emptyset \quad \text{for } 1 \leqslant p \leqslant P.$$

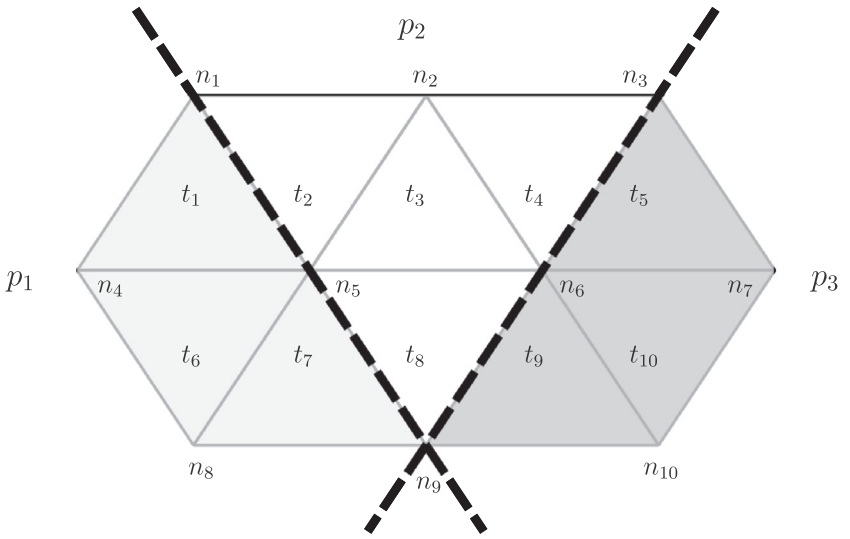

Fig. 1. A 3-way tetrahedron distribution $\mathcal{T} = \{T_1, T_2, T_3\}$ of a given triangulation $T = \{t_1, t_2, \ldots, t_{10}\}$ with nodes $N = \{n_1, \ldots, n_{10}\}$. The tetrahedra of this triangulation are distributed among three processes p_1, p_2, and p_3. The shading indicates the sets of tetrahedra T_1, T_2, and T_3 assigned to these processes: tetrahedra assigned to p_1 are colored light gray, those assigned to p_2 white, and those assigned to p_3 dark gray.

2. Processes store disjoint subsets of tetrahedra, i.e.,

$$T_p \cap T_q = \emptyset \quad \text{for } 1 \leqslant p < q \leqslant P.$$

3. The union of tetrahedra stored on all processes is the triangulation itself, i.e.,

$$\bigcup_{p=1}^{P} T_p = T.$$

A given P-way tetrahedron distribution induces a distribution of the nodes N. We assume that if a tetrahedron t is stored on a process p then all the nodes of t are also located at the same process. We formalize this node distribution as follows. Let $\mathcal{P}(S)$ denote the power set of a set S. Then, for a given P-way tetrahedron distribution $\mathcal{T}$, we introduce the mapping

$$\Lambda_{\mathcal{T}} : N \to \mathcal{P}(\{1, \ldots, P\})$$

which maps each node $n \in N$ to the set of processes that store a tetrahedron containing n. Let

$$\lambda_{\mathcal{T}}(n) := |\Lambda_{\mathcal{T}}(n)|$$

denote the number of processes on which a node n is stored. A node n that is stored at a single process, i.e., $\lambda_{\mathcal{T}}(n) = 1$, is called *local* whereas a node located at a process boundary for which $\lambda_{\mathcal{T}}(n) \geqslant 2$, is called *distributed*. For instance, in Fig. 1 with $L = 10$ nodes, the local node n_2 is stored only by process p_2. Thus, $\Lambda_{\mathcal{T}}(n_2) = \{p_2\}$ and $\lambda_{\mathcal{T}}(n_2) = 1$. Since the distributed node n_9 is found at all three processes, we have $\Lambda_{\mathcal{T}}(n_9) = \{p_1, p_2, p_3\}$ and $\lambda_{\mathcal{T}}(n_9) = 3$.

2.2. Distributed degrees of freedom

In triangulations arising from finite element discretizations of partial differential equations, degrees of freedom (DOF) are introduced to represent finite element functions which, in turn, describe a solution of an underlying problem. We assume that these DOF are located at nodes $n \in N$. The values of the DOF at a node n are represented by a vector $x_n \in \mathbb{R}^{c(n)}$ where $c(n)$ denotes the number of DOF. This number $c(n)$ may vary for different nodes. Now suppose that there is a P-way tetrahedron distribution of the triangulation together with its induced node distribution. Let $x_n^p \in \mathbb{R}^{c(n)}$ denote the value of the DOF of a node n assigned to process p. For a local node n, we have $x_n^p = x_n$. For distributed nodes, however, we store the distributed additive contributions: each of the processes $p \in \Lambda_{\mathcal{T}}(n)$ stores the portion x_n^p of x_n such that

$$x_n = \sum_{p \in \Lambda_{\mathcal{T}}(n)} x_n^p. \tag{1}$$

To evaluate (1) for a distributed node n, the processes $p \in \Lambda_{\mathcal{T}}(n)$ need to communicate their local values. We assume the following *all-neighbor approach* to transform the local values x_n^p into the global value x_n by all processes $p \in \Lambda_{\mathcal{T}}(n)$ such that each process $p \in \Lambda_{\mathcal{T}}(n)$ stores a copy of the global value, afterward. In this approach, each process $p \in \Lambda_{\mathcal{T}}(n)$ sends its x_n^p to all other processes in $\Lambda_{\mathcal{T}}(n)$. The communication pattern of the all-neighbor approach corresponds to an all-to-all broadcast within the group of processes $\Lambda_{\mathcal{T}}(n)$. Afterward, each process in $\Lambda_{\mathcal{T}}(n)$ is capable of evaluating the global value x_n according to formula (1) by adding the received data to its local value x_n^p.

We now consider the communication volume that is caused by distributed nodes. Given the number of processes, $\lambda_{\mathcal{T}}(n)$, on which node n is stored, let $f(\lambda_{\mathcal{T}}(n))$ denote the number of data words to be communicated between the processes in $\Lambda_{\mathcal{T}}(n)$ to evaluate a scalar entry of (1) for this node n. Then summing up the contributions of all $n \in N$ with the corresponding number of DOF gives the total communication volume

$$C_f(\mathcal{T}) := \sum_{n \in N} c(n) f(\lambda_{\mathcal{T}}(n)). \tag{2}$$

For the all-neighbor approach, the function f is given by

$$f_{\text{all-neigh}}(\lambda_{\mathcal{T}}(n)) := \lambda_{\mathcal{T}}(n) \cdot (\lambda_{\mathcal{T}}(n) - 1), \tag{3}$$

because each of the $\lambda_{\mathcal{T}}(n)$ processes in $\Lambda_{\mathcal{T}}(n)$ sends its local x_n^p to $\lambda_{\mathcal{T}}(n) - 1$ processes. Here, we assume that, rather than performing multiple send operations between any two processes separately, communication between these processes is carried out by collecting the data in a single message. Adding up the message volumes between all neighboring processes yields the total communication volume (2).

For parallel finite element simulations on a distributed triangulation, it is not uncommon that communication among neighboring processes resulting from computations of the form (1) is the only type of neighboring communication occurring in linear algebra operations. In the present paper, we will therefore assume that (2) represents the total communication volume exactly—an assumption which is valid for DROPS.

2.3. Problem of finding a distribution

After identifying the total communication volume of a given P-way tetrahedron distribution of a triangulation, the problem is now to find the "best" distribution of the tetrahedra among a set of P processes. Here, the meaning of "best" is understood as finding a distribution of the tetrahedra such that the total communication volume involved in the computations is minimized while, at the same time, the number of tetrahedra is evenly balanced among the processes. This problem is formally stated as follows.

Problem 1 (*Triangulation Distribution Problem*). Given a triangulation T with $c(n)$ DOF located at each node $n \in N$, a number of processes $P \geqslant 2$, a function $f(\lambda_{\mathcal{T}}(n))$ representing the number of data words to be communicated caused by a node $n \in N$ which is stored at $\lambda_{\mathcal{T}}(n)$ processes, and a balancing tolerance parameter $\varepsilon > 0$, find a P-way tetrahedron distribution $\mathcal{T} = \{T_1, \ldots, T_P\}$ of T which minimizes the total communication volume

$$C_f(\mathcal{T}) := \sum_{n \in N} c(n) f(\lambda_{\mathcal{T}}(n)),$$

while satisfying the balancing constraint

$$|T_p| \leqslant (1+\varepsilon)\frac{|T|}{P} \quad \text{for all } 1 \leqslant p \leqslant P. \tag{4}$$

A P-way tetrahedron distribution $\mathcal{T}$ satisfying (4) is called ε-*balanced*.

2.4. A related communication pattern

The all-neighbor approach implemented in DROPS involves a communication pattern for the evaluation of (1) for a distributed node n. However, there are alternatives based on different communication patterns. In all these patterns, the processes $p \in \Lambda_{\mathcal{T}}(n)$ need to send their local values to other processes. The objective is to transform the local values x_n^p into the global value x_n by all processes $p \in \Lambda_{\mathcal{T}}(n)$ such that each process $p \in \Lambda_{\mathcal{T}}(n)$ stores a copy of the global value, afterward.

An alternative to the all-neighbor approach is as follows. First, we define a particular process $p^* \in \Lambda_{\mathcal{T}}(n)$ for each node $n \in N$. The approach is based on [22,23], where this process p^* is called the "owner process" of the node n. Then, each process $p \in \Lambda_{\mathcal{T}}(n) \setminus \{p^*\}$ sends its x_n^p to the owner process that evaluates the global value x_n by formula (1). Afterward, the owner sends x_n back to all processes $p \in \Lambda_{\mathcal{T}}(n) \setminus \{p^*\}$. In this approach, referred to as the *owner approach*, all x_n^p are first sent by $\lambda_{\mathcal{T}}(n) - 1$ processes to the owner. Then, the owner sends the global x_n back to $\lambda_{\mathcal{T}}(n) - 1$ processes. Hence, the number of data words to be communicated between the processes in $\Lambda_{\mathcal{T}}(n)$ is given by

$$f_{\text{owner}}(\lambda_{\mathcal{T}}(n)) := 2(\lambda_{\mathcal{T}}(n) - 1). \tag{5}$$

Recall from (3) that the corresponding function of the all-neighbor approach, $f_{\text{all-neigh}}$, is different. So, the communication volumes for the two approaches denoted by $C_{f_{\text{owner}}}$ and $C_{f_{\text{all-neigh}}}$ also differ; see the definition in (2). The main focus of the present paper is on the all-neighbor approach. However, to judge it against a different approach we briefly take the owner approach into account for certain parts of the numerical experiments.

3. A hypergraph model for distributions of triangulations

In this section, we transform the triangulation distribution Problem 1 into an equivalent partitioning problem on a hypergraph. The corresponding hypergraph model enables a systematic approach to find a distribution of the tetrahedra among processes. To a triangulation $T = \{t_1, \ldots, t_K\}$ with nodes $N = \{n_1, \ldots, n_L\}$ we associate a hypergraph $H = (V, E)$ where V denotes the set of vertices and $E \subset \mathcal{P}(V)$ the set of hyperedges. A hyperedge is a nonempty subset of the set of vertices V. Every tetrahedron $t_i \in T$ is associated with a vertex $v_i \in V$. Thus, the hypergraph has K vertices $V = \{v_1, \ldots, v_K\}$. Every node $n_i \in N$ is associated with a hyperedge $e_i \in E$ such that there are L hyperedges and $E = \{e_1, \ldots, e_L\}$. A vertex v_i is in the hyperedge e_j if and only if the tetrahedron represented by v_i contains the node represented by e_j. Such a vertex $v_i \in e_j$ is called a pin of the hyperedge e_j. The number of pins of a hyperedge is referred to as hyperedge size. A hyperedge in a hypergraph represents connections between any number of vertices; this is necessary because more than two tetrahedra in T can intersect in a single node. Alternatively, the hyperedges can be interpreted as a means to capture the topological neighborship relation between tetrahedra in T. Two or more tetrahedra that are adjacent in T share a common node. The hyperedge representing that node then contains all the vertices representing these adjacent tetrahedra.

The hypergraph H associated to the triangulation T given in Fig. 1 is displayed in Fig. 2. Since $T = \{t_1, \ldots, t_{10}\}$ consists of $K = 10$ tetrahedra, this hypergraph H consists of 10 vertices $V = \{v_1, \ldots, v_{10}\}$, which are illustrated by circles. The $L = 10$ nodes $N = \{n_1, \ldots, n_{10}\}$ in T correspond to 10 hyperedges $E = \{e_1, \ldots, e_{10}\}$ in H. A hyperedge in this figure is shown by a square together with pins to the vertices belonging to that hyperedge. For instance, the node n_9 is adjacent to the tetrahedra t_7, t_8, and t_9. Thus, the associated hyperedge $e_9 = \{v_7, v_8, v_9\}$ connects the three corresponding vertices.

A distribution of tetrahedra in T induces a partitioning of the vertices in H. More precisely, for a given P-way tetrahedron distribution of T, there is a P-way hypergraph partitioning of H with the following property: if a tetrahedron is stored by

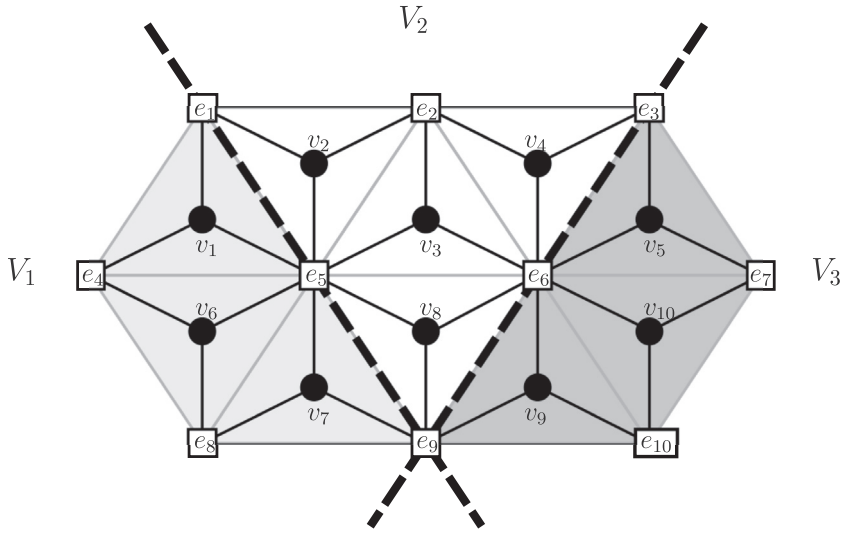

Fig. 2. A 3-way hypergraph partitioning $\mathcal{V} = \{V_1, V_2, V_3\}$ of the hypergraph H associated to the triangulation T from Fig. 1. The vertices are illustrated by circles whereas the hyperedges are drawn using squares and pins. The vertices of this hypergraph are distributed among three processes as indicated by the same shading used in Fig. 1.

process p, then the associated hypergraph vertex is also stored by process p. The following definition introduces a P-way hypergraph partitioning formally.

Definition 2 (P-way Hypergraph Partitioning). Given a hypergraph $H = (V, E)$ and a number $P \geqslant 2$, a partition of its vertices into P parts denoted by $\mathcal{V} := \{V_1, \ldots, V_P\}$ is called a *P-way hypergraph partitioning of H* if the following three conditions hold:

1. Each part is a nonempty set of vertices, i.e.,
   $$V_p \subset V \text{ and } V_p \neq \emptyset \quad \text{for } 1 \leqslant p \leqslant P.$$
2. Parts are disjoint, i.e.,
   $$V_p \cap V_q = \emptyset \quad \text{for } 1 \leqslant p < q \leqslant P.$$
3. The union of vertices of all parts is the vertex set itself, i.e.,
   $$\bigcup_{p=1}^{P} V_p = V.$$

Next, we focus on the communication volume and how it is transformed from the triangulation to the hypergraph. Recall from (2) that, in T, the communication volume depends on the number $\lambda_{\mathcal{T}}(n)$ of different processes on which a node n is stored. Processes in a tetrahedron distribution $\mathcal{T} = \{T_1, \ldots, T_P\}$ correspond to parts in the induced hypergraph partitioning $\mathcal{V} := \{V_1, \ldots, V_P\}$. Since a node n in T corresponds to a hyperedge e in H, the number $\lambda_{\mathcal{T}}(n)$ corresponds to the number of different parts in which e has vertices. This number is called the *connectivity* $\lambda_{\mathcal{V}}(e)$ of a hyperedge $e \in E$ and is defined by

$$\lambda_{\mathcal{V}}(e) := |\{1 \leqslant i \leqslant P \mid e \cap V_i \neq \emptyset\}|.$$

Since all hyperedges $e \neq \emptyset$, we have that $\lambda_{\mathcal{V}}(e) \geqslant 1$. This definition allows us to describe the total communication volume (2) in terms of the hypergraph H as

$$C_f(\mathcal{V}) := \sum_{e \in E} \sigma(e) f(\lambda_{\mathcal{V}}(e)). \tag{6}$$

Here, the weight $\sigma(e)$ of a hyperedge e is introduced as the number of DOF located at the node $n \in N$ to which e is associated:

$$\sigma(e) := c(n). \tag{7}$$

The function f in (6) is a *communication volume metric* and is given by

$$f_{\text{all-neigh}}(\lambda_{\mathcal{V}}(e)) := \lambda_{\mathcal{V}}(e) \cdot (\lambda_{\mathcal{V}}(e) - 1)$$

for the all-neighbor approach; cf. (3).

Weights cannot only be introduced for hyperedges but also for vertices. Let ϱ denote the function which assigns a weight $\varrho(v) > 0$ to each vertex (or tetrahedron) v. We will consider a hypergraph $H = (V, E, \varrho, \sigma)$ with a weighting function for vertices $\varrho : V \to \mathbb{R}_{>0}$, as well as a weighting function for hyperedges $\sigma : E \to \mathbb{R}_{>0}$.

Overall, the discussion in this section shows that the triangulation distribution Problem 1 is equivalent to the following hypergraph partitioning problem. This partitioning problem represents an exact model of the total communication volume, rather than just an approximation.

Problem 2 (Hypergraph Partitioning Problem). Given a hypergraph $H = (V, E, \varrho, \sigma)$, a number $P \geqslant 2$, a communication volume metric $f(\lambda_{\mathcal{V}}(e))$ depending on the connectivity $\lambda_{\mathcal{V}}(e)$ of a hyperedge $e \in E$, and a balancing tolerance parameter $\varepsilon > 0$, find a P-way hypergraph partitioning $\mathcal{V} = \{V_1, \ldots, V_P\}$ of H which minimizes the total communication volume

$$C_f(\mathcal{V}) := \sum_{e \in E} \sigma(e) f(\lambda_{\mathcal{V}}(e)),$$

while satisfying the balancing constraint

$$\varrho(V_p) \leqslant (1 + \varepsilon)\frac{\varrho(V)}{P} \quad \text{for all } 1 \leqslant p \leqslant P. \tag{8}$$

As usual, the weight of a set of vertices in (8) is defined as the sum of the weights of the vertices in that set. A P-way hypergraph partitioning $\mathcal{V}$ satisfying (8) is called *ε-balanced*.

3.1. Related communication volume metrics

Though the focus of the present paper is on the all-neighbor communication volume metric, we mention that different metrics are also commonly used in hypergraph partitioning. In particular, we consider two additional metrics in the numerical experiments below so that, overall, we are concerned with the following three metrics:

$$f_{\text{all-neigh}}(\lambda_{\mathcal{V}}(e)) := \lambda_{\mathcal{V}}(e) \cdot (\lambda_{\mathcal{V}}(e) - 1) \tag{9}$$
$$f_{\text{owner}}(\lambda_{\mathcal{V}}(e)) := 2(\lambda_{\mathcal{V}}(e) - 1), \tag{10}$$
$$f_{\text{cut-net}}(\lambda_{\mathcal{V}}(e)) := \min(\lambda_{\mathcal{V}}(e) - 1, 1). \tag{11}$$

The first two metrics, (9) and (10), represent the previously described approaches to transform the local values to global values; see the all-neighbor approach (3) and the owner approach (5), respectively. The metric of the owner approach (10) is a scaled version of the connectivity−1 metric, $f(\lambda_{\mathcal{V}}) = \lambda_{\mathcal{V}} - 1$, which is used for parallelization of sparse matrix–vector multiplication [2]. Moreover, the choice (11) is commonly known as the cut–net metric, $f(\lambda_{\mathcal{V}}) = \min(\lambda_{\mathcal{V}} - 1, 1)$, which is applied in [24]. The choice of $f(\lambda_{\mathcal{V}}) = \lambda_{\mathcal{V}}(\lambda_{\mathcal{V}} - 1)$ we propose in (9) is new in the context of partitioning finite element triangulations, to the best of our knowledge. However, a scaled version of this all-neighbor metric is briefly mentioned in [25] where the author conjectures that this metric "may be useful in distributed systems applications where it is desired to assign a unit cost to each processor-to-processor communication."

The formulation of the total communication volume (6) does not distinguish between hyperedges that connect multiple parts and those that contain only vertices of a single part. That is, the sum runs over all hyperedges, regardless of their connectivity. However, hyperedges containing only vertices of a single part do not cause any communication at all. The formulation of the total communication volume takes this into account by requiring the property

$$f(1) = 0$$

of a communication volume metric. More precisely, if the connectivity of a hyperedge is $\lambda_{\mathcal{V}}(e) = 1$, then the corresponding term in the sum (6) vanishes due to the factor $\lambda_{\mathcal{V}}(e) - 1$ in the metrics (9)–(11). Hence, this term in the sum does not contribute to the total communication volume. For instance, the term corresponding to e_4 in the partitioning given in Fig. 2 vanishes because $\lambda_{\mathcal{V}}(e_4) = 1$.

4. Recursive bisection for the hypergraph partitioning problem

The hypergraph partitioning Problem 2 is known to be NP-hard for nondecreasing f with $f(1) = 0$ and $f(2) > 0$; see [26,27] where the special case of a bisection of an undirected graph is considered. Therefore, this problem is commonly solved by heuristic approaches.

A powerful heuristic is based on recursively partitioning the hypergraph into two parts. The following theorem establishes the foundation for such a recursive bisection.

Theorem 1. *Let $\mathcal{V} = \{V_1, \ldots, V_{P-1}, V_P\}$ be a P-way partitioning of a hypergraph $H = (V, E, \varrho, \sigma)$. Suppose we split $V_P = U \cup W$ as a disjoint union to obtain the $(P+1)$-way hypergraph partitioning $\mathcal{V}' = \{V_1, \ldots, V_{P-1}, U, W\}$, then*

$$C_f(\mathcal{V}') = C_f(\mathcal{V}) + \sum_{e \in E(U,W)} \sigma(e)\,(f(\lambda_{\mathcal{V}}(e)+1) - f(\lambda_{\mathcal{V}}(e))), \tag{12}$$

where

$$E(U,W) := \{e \in E \mid e \cap U \neq \emptyset \wedge e \cap W \neq \emptyset\}.$$

Proof. Note that $E(U,W)$ consists precisely of all hyperedges that are cut or cut further in the splitting of V_P into U and W. All other hyperedges are oblivious to the splitting of V_P, i.e., the number of different parts they connect does not change:

$$\lambda_{\mathcal{V}'}(e) = \begin{cases} \lambda_{\mathcal{V}}(e)+1 & \text{if } e \in E(U,W), \\ \lambda_{\mathcal{V}}(e) & \text{otherwise.} \end{cases} \tag{13}$$

Therefore,

$$\begin{aligned} C_f(\mathcal{V}') - C_f(\mathcal{V}) &= \sum_{e \in E} \sigma(e)(f(\lambda_{\mathcal{V}'}(e)) - f(\lambda_{\mathcal{V}}(e))) = \sum_{e \in E(U,W)} \sigma(e)(f(\lambda_{\mathcal{V}'}(e)) - f(\lambda_{\mathcal{V}}(e))) + \sum_{e \in E \setminus E(U,W)} \sigma(e)(f(\lambda_{\mathcal{V}'}(e)) - f(\lambda_{\mathcal{V}}(e))) \\ &= \sum_{e \in E(U,W)} \sigma(e)(f(\lambda_{\mathcal{V}}(e)+1) - f(\lambda_{\mathcal{V}}(e))) + 0, \end{aligned}$$

which shows the desired result. □

The theorem can be seen as a generalization of [19, Theorem 2.2], which deals with the case of $f(\lambda_{\mathcal{V}}) = \lambda_{\mathcal{V}} - 1$ in the context of sparse matrix–vector multiplication with a two-dimensional partitioning. This, in turn, generalizes an earlier result [2] for a one-dimensional matrix partitioning.

Algorithm 1. Algorithm outline for generating an ε-balanced P-way partitioning $\mathcal{V}$ of a given hypergraph $H = (V, E, \varrho, \sigma)$. This algorithm solves Problem 2 by greedily minimizing $C_f(\mathcal{V})$ for a given metric f.

1: Let $\mathcal{V} \leftarrow \{V\}$ and $\lambda_{\mathcal{V}}(e) \leftarrow 1$ for all $e \in E$.
2: **while** $|\mathcal{V}| < P$ **do**
3: Select the largest part $V' \in \mathcal{V}$ and determine the bipartitioning imbalance $\varepsilon' < \varepsilon$ from $\mathcal{V}, P$, and ε (e.g., using [19, Algorithm 1]).
4: Extract the subhypergraph H' of H corresponding to V'.
5: Obtain the vertex weights of H' through restriction $\varrho' := \varrho|_{V'}$.
6: Calculate the hyperedge weights σ' of H' using (14).
7: Create an ε'-balanced partitioning of $H' = (V', E', \varrho', \sigma')$ into two parts $V' = U \cup W$.
8: Update the partitioning by $\mathcal{V} \leftarrow (\mathcal{V} \setminus \{V'\}) \cup \{U, W\}$.
9: Let $\lambda_{\mathcal{V}}(e) \leftarrow \lambda_{\mathcal{V}}(e) + 1$ for all $e \in E(U,W)$ according to (13).
10: **end while**
11: return $\mathcal{V}$

In particular, (12) shows us that we can minimize the communication volume C_f for any valid f by using recursive bipartitioning (i.e. generating a partitioning by recursively cutting the set of all vertices into two) as outlined in Algorithm 1. We start out with the partitioning $\mathcal{V} = \{V\}$, consisting of one part containing all vertices. Then, until we have created P parts, we select the largest $V' \in \mathcal{V}$ and cut it up into two parts by bipartitioning the subhypergraph $H' := (V', E', \varrho', \sigma')$ of H, with

$$E' := \{e \cap V' \mid e \in E \wedge e \cap V' \neq \emptyset\}.$$

For more details on how to perform hypergraph bipartitioning based on this theorem (in particular load balancing), we would like to refer the reader to [19, Algorithm 1] derived from [19, Theorem 2.2]. We provide H' with vertex weights obtained from restricting the nodes V to the set V', i.e., $\varrho' := \varrho|_{V'}$, and hyperedge weights σ' determined by (12):

$$\sigma'(e \cap V') := \sigma(e)\,(f(\lambda_{\mathcal{V}}(e)+1) - f(\lambda_{\mathcal{V}}(e))), \quad \text{for all } e \cap V' \in E'. \tag{14}$$

Now we find a bipartitioning $V' = U \cup W$ such that the total cost of all cut hyperedges,

$$\sum_{e \in E'(U,W)} \sigma'(e),$$

is minimal, and replace V' in the partitioning $\mathcal{V}$ by U and W. According to (12) this will yield an increase of the communication volume that is as small as possible, and therefore provides a heuristic for minimizing the total communication volume for the partitioning into P parts, for any metric f. The only tool we require is a standard hypergraph bipartitioner in step 4.

The new hyperedge weights (14) in each bipartitioning step of Algorithm 1 depend on the metric f. For the all-neighbor metric, these new hyperedge weights are given by

$$\sigma'_{\text{all-neigh}}(e \cap V') = \sigma(e)((\lambda_{\mathcal{V}}(e)+1)\lambda_{\mathcal{V}}(e) - \lambda_{\mathcal{V}}(e)(\lambda_{\mathcal{V}}(e)-1)) = 2\,\sigma(e)\,\lambda_{\mathcal{V}}(e).$$

In summary, by inserting the different metrics (9)–(11) into (14), we arrive at the following three hyperedge weights:

$$\begin{aligned}
\sigma'_{\text{all-neigh}}(e \cap V') &= 2\,\sigma(e)\,\lambda_{\mathcal{V}}(e) \quad \text{for } \lambda_{\mathcal{V}}(e) \geqslant 1,\\
\sigma'_{\text{owner}}(e \cap V') &= 2\,\sigma(e) \quad \text{for } \lambda_{\mathcal{V}}(e) \geqslant 1,\\
\sigma'_{\text{cut-net}}(e \cap V') &= \begin{cases} \sigma(e) & \text{if } \lambda_{\mathcal{V}}(e) = 1,\\ 0 & \lambda_{\mathcal{V}}(e) > 1. \end{cases}
\end{aligned}$$

From inspection of these hyperedge weights, we conclude that the partitioning strategy for the all-neighbor metric is straightforward in case of sequential partitioning. However, it will make parallelization of the partitioning process more difficult, because we need the $\lambda_{\mathcal{V}}(e)$ values of all hyperedges $e \cap V' \in E'$ in the hypergraph. Hence, after the partitioning of any part $V' \in \mathcal{V}$, the $\lambda_{\mathcal{V}}$ values of H will need to be updated, resulting in a global synchronization step after each partitioning. For the owner or cut–net metrics this problem is less severe, as $\sigma'(e \cap V')$ becomes independent of $\lambda_{\mathcal{V}}(e)$ once $\lambda_{\mathcal{V}}(e) > 1$.

5. Numerical experiments

In the remainder of this article, we present numerical experiments for two different sets of problems. Since the main focus of this article is on modeling the communication involved in the finite element solver DROPS, we begin our discussion in the first subsection with a description of the corresponding results. In a second subsection, we consider a set of artificially constructed hypergraphs that, compared to those arising from DROPS, exhibit a larger variation in their hyperedge sizes.

Once a hypergraph is set up, the solution to the hypergraph partitioning Problem 2 can be computed by Algorithm 1 for any of the three communication volume metrics (9)–(11). We will use the following partitioning approaches:

all-neigh	We determine a hypergraph partitioning $\mathcal{V}_{\text{all-neigh}}$ by the solution of Problem 2 using $f = f_{\text{all-neigh}}$.
owner	We determine a hypergraph partitioning $\mathcal{V}_{\text{owner}}$ by the solution of Problem 2 using $f = f_{\text{owner}}$.
cut–net	We determine a hypergraph partitioning $\mathcal{V}_{\text{cut-net}}$ by the solution of Problem 2 using $f = f_{\text{cut-net}}$.
graph	We determine a partitioning $\mathcal{V}_{\text{graph}}$ using a standard (undirected) graph partitioning formulation in which vertices are identical to those defined by the hypergraph model, while the edges differ. An edge of this graph model represents adjacent tetrahedra sharing a common face. The resulting P-way graph partitioning problem is solved by the library METIS (v. 5.0.2) [21], using the high-quality `PartGraphKway()` function with the edge-cut metric. The details of this approach are described in [28] where it is referred to as the "triangulation graph partitioning model".

As a shorthand notation, we refer to such a partitioning by its index; for instance, we use all-neigh to indicate the partitioning $\mathcal{V}_{\text{all-neigh}}$. For the all-neigh, owner, and cut–net partitionings, we use the Mondriaan matrix partitioner [19], together with PaToH [20] as external bipartitioner. For all four partionings, we carried out 50 runs with a partitioning algorithm using different random seeds. We then determine the minimum communication volumes over these 50 runs. We did not use the internal hypergraph bipartitioner of Mondriaan, as it assumes unit hyperedge weights (which is incompatible with (14)). Therefore, Mondriaan will perform Algorithm 1 except for the bipartitioning step at line 4, which is done using PaToH. For the owner partitionings, we also use Mondriaan's vector partitioning functionality [19, Section 3] to minimize the total number of data words sent and received by each processor.

All results are gathered on a compute cluster at the Center for Computing and Communication at RWTH Aachen University. This cluster consists of dual socket nodes (Bullx Blade B500) equipped with two Intel Westmere (X5675) processors each. The processors run at a clock rate of 3.06 GHz and the nodes are connected by a QDR Infiniband network.

5.1. Results for the finite element solver DROPS

For this set of experiments, we investigate three different problems which vary in the number of tetrahedra and, hence, in the number of vertices, hyperedges, and DOF. In all these problems, the underlying computational domain Ω represents a measurement cell [29] which is used to study the behavior of levitated droplets [30]. We recursively apply different numbers of local refinements to the triangulation yielding the problem sizes illustrated in Table 1. For each problem size, this table presents the number of vertices $|V|$, the number of hyperedges $|E|$, the total number of pins $\Delta_{\text{pin}} := \sum_{e\in E}|e|$, the minimum hyperedge size $\Delta_{\min} := \min_{e\in E}|e|$, the maximum hyperedge size $\Delta_{\max} := \max_{e\in E}|e|$, and the median of the hyperedge sizes Δ_{med} of the resulting hypergraphs. Throughout the experiments we assume a situation where a single DOF is stored at each node $n \in N$. So by (7), all hyperedge weights are set to 1. The weights of vertices are designed to encode information on the triangulation. In this article, we consider a single triangulation which is a special case of a multi-level triangulation. The latter enters the picture via the three problem sizes that correspond to three levels of refinement. A weighting function for the vertices of a multi-level triangulation is introduced in [11]. We follow that approach by setting the vertex weight to the number of tetrahedra represented by that vertex.

We investigate the total communication volume that occurs when evaluating the sum in (1) for all DOF. Recall from (6) that, in terms of the hypergraph, this communication volume is given by $C_f(\mathcal{V})$ where $\mathcal{V}$ is some given partition. The com-

munication volume metric f then represents a communication pattern that is applied when evaluating (6) on a given partition $\mathcal{V}$. We consider $C_{f_{\text{owner}}}(\mathcal{V})$ and $C_{f_{\text{all-neigh}}}(\mathcal{V})$ when evaluating this sum with the communication pattern $f = f_{\text{owner}}$ and $f = f_{\text{all-neigh}}$, respectively. We compare these two communication volumes for three different partitionings graph, owner, and all-neigh representing different tetrahedron distributions.

The total communication volumes $C_{f_{\text{owner}}}(\mathcal{V})$ and $C_{f_{\text{all-neigh}}}(\mathcal{V})$ are illustrated for these partitioning approaches in Fig. 3. A tolerance of $\varepsilon = 5\,\%$ for the balance condition in (8) is used for all partitioning approaches. All partitioners are able to satisfy the balancing constraint for all problems, with the exception of the owner partitioner for the medium problem divided into $P = 1024$ parts. Therefore, we do not show any data for this particular case.

From Fig. 3 we find that, compared to graph, the communication volume is almost always smaller when using a hypergraph-based approach. For instance, consider the medium problem for $P = 1024$ and $C_{f_{\text{all-neigh}}}(\mathcal{V})$ in Fig. 3(d). Here, we observe that the communication volume decreases from 261,774 for the graph approach to 250,456 for the hypergraph-based all-neigh approach, corresponding to a saving of 4.5%. On average the all-neighbor communication volume obtained by all-neigh is 3% lower than the volume obtained by graph, while the all-neighbor communication volume obtained by owner is very close to that of all-neigh. If we focus on the differences between the two hypergraph approaches, the minimum of volumes for a given number of processes is—almost always—achieved if the corresponding objective function is used in the partitioning. That is, $C_{f_{\text{all-neigh}}}(\mathcal{V})$ is minimal for the partitioning all-neigh and $C_{f_{\text{owner}}}(\mathcal{V})$ is minimal for the partitioning owner. When comparing the results in Fig. 3, we also notice that the volumes obtained by the owner partitioning are smaller than those of all-neigh. This observation is sound, since $f_{\text{owner}}(\lambda_{\mathcal{V}}) \leqslant f_{\text{all-neigh}}(\lambda_{\mathcal{V}})$ holds for all $\lambda_{\mathcal{V}}$.

The formulation of the triangulation distribution Problem 1 and the equivalent hypergraph partitioning Problem 2 aim at finding a partitioning of the triangulation minimizing the communication volume. Though not explicitly addressed by the objective function of this partitioning problem, we next focus on the communication time. More precisely, we consider the time needed for determining the sum (1) for all nodes $n \in N$. Given any P-way partitioning, DROPS is capable of evaluating the sum (1). However, the only communication pattern used in DROPS is currently the all-neighbor pattern. A novel communication library is currently being developed that will allow future versions of DROPS to evaluate this sum by the owner communication pattern. Therefore, in Fig. 4, we present the communication time for evaluating this sum 1000 times using the all-neighbor communication pattern. Here, we see that, for a small number of processes, the performance of all three partitioning approaches is nearly identical, while for a large number of processes the partitioning all-neigh tends to result in less communication time than both graph and owner. In most cases, the longest communication time is observed when using the hypergraph-based approach owner. Distributing the triangulation by graph yields shorter communication times than the distribution obtained by owner but longer times than the one determined by all-neigh. Thus, modeling the exact communication volume by the hypergraph model, as done by all-neigh, also tends to be advantageous to reduce communication time, which forms a major overhead in parallel computing.

In addition to the communication volume and the communication times discussed in previous paragraphs, we now consider another quantity of interest. In Table 2, we present the number of messages which are needed to compute the sum (1) for all $n \in N$. In Table 2(a), we consider the number of messages using the all-neighbor communication pattern. This communication pattern is implemented in DROPS and the values in that part of the table are not only those that result from an analysis of a given partitioning but also coincide with the actual number of messages sent in the implementation. In contrast, the numbers given in Table 2(b) refer to the owner communication pattern and are theoretically extracted from a given partitioning of the hypergraph. They are not practically observed in the implementation since the current version of DROPS is not capable of determining the sum in (1) by the owner approach.

Comparing the two partitioning approaches all-neigh and graph for the all-neighbor communication pattern in Table 2(a), we do not find that, in general, one of the two partitionings leads to a smaller number of messages. However, the all-neigh partitioning approach tends to reduce the number of messages when the number of parts P becomes large. In Table 2(b), we compare owner and graph for the owner communication pattern. Here, the partitioning graph yields a smaller number of messages. The only exceptions are the medium case using $P = 32$ and the large case using $P = 16$. A number of messages in Table 2(b) is always larger than the corresponding number in Table 2(a). That is, the all-neighbor communication pattern to determine the sum in (1) results in less messages than the owner communication pattern. This is mainly caused by the fact that two messages needs to be exchanged between the owner process of a DOF located at n and the $\lambda(n) - 1$ processes, see Section 2.4.

The time it takes to generate a hypergraph partitioning is comparable for owner and all-neigh (within 3% of each other on average). For instance, consider the hypergraph partitioner based on a combination of Mondriaan and PaToH to partition the hypergraph into 256 parts. Here, the partitioning owner needs 0.34 s, 2.1 s, and 11.3 s for the small, medium and large

Table 1
Problem sizes and characteristics of hypergraphs representing three finite element meshes based on the computational domain of a measurement cell.

Problem	$\lvert V\rvert$	$\lvert E\rvert$	Δ_{pin}	Δ_{min}	Δ_{med}	Δ_{max}
small	5059	12,042	59,847	1	15.5	36
medium	16,247	101,462	374,947	1	17.5	36
large	66,616	551,425	1,914,326	1	18.0	36

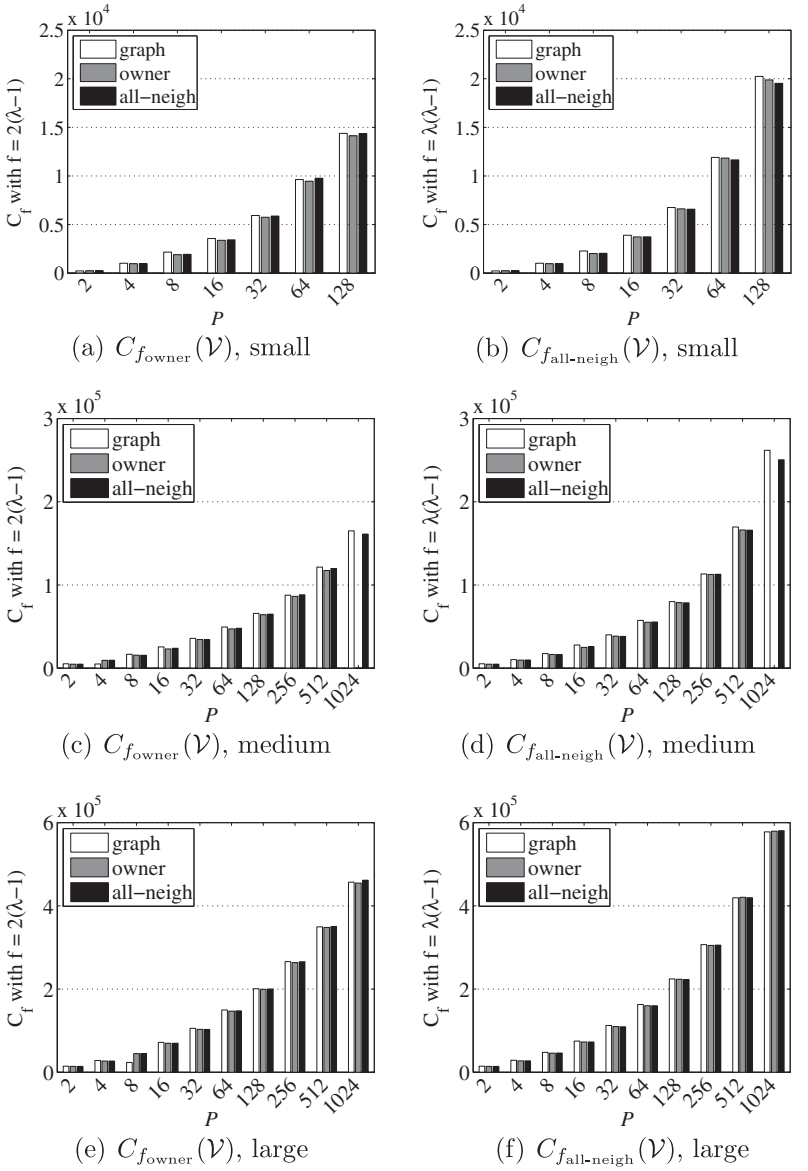

Fig. 3. Communication volumes for the owner communication pattern $C_{f_{\text{owner}}}(\mathcal{V})$ (left) and for the all-neighbor communication pattern $C_{f_{\text{all-neigh}}}(\mathcal{V})$ (right) when varying the partitioning approach $\mathcal{V}$ among graph, owner, and all-neigh. Here, the sum in (1) is determined for all $n \in N$, considering the three finite element problems from Table 1.

problem, respectively. The corresponding timings for all-neigh are given by 0.35 s, 2.1 s, and 11.6 s. The partitioning graph based on METIS required only around 1.6% of these hypergraph partitioning times. In real-life finite element computations there is a trade-off between the time spent in determining a distribution of the tetrahedra and the time spent in determining the sum (1). In our findings, we illustrate that determining a "better" distribution of the tetrahedra among the processes by the more time consuming hypergraph model yields better communication times for linear algebra operations in the simulation, such as determining the sum (1). This may be advantageous if a static triangulation is used, i.e., a single triangulation is given for the whole simulation. However, this may become disadvantageous if an adaptive triangulation strategy is used where the triangulation constantly changes. In this case, many tetrahedral decompositions need to be determined and only a few linear algebra operations are performed on each triangulation. Here, a fast but less accurate graph partitioning may be the right choice to reduce the overall simulation time.

5.2. Results for artificial problems with larger hyperedge sizes

Recall from Fig. 3 that the communication volume of the partitioning approach all-neigh is similar to that of owner. This is explained by the fact that, in the hypergraphs arising from the triangulation distribution problem, the resulting hyperedge

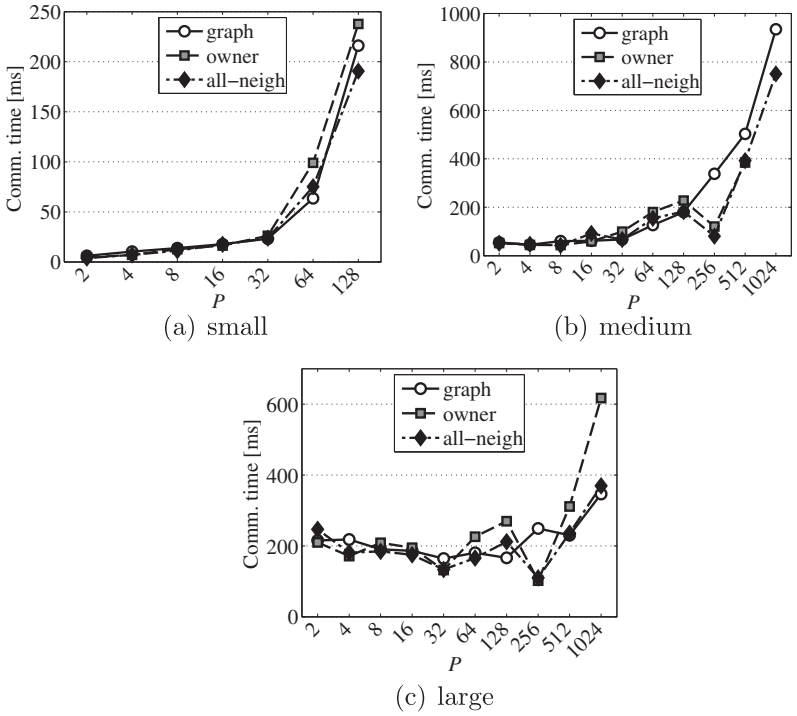

Fig. 4. Communication time for determining the sum in (1) 1000 times for all nodes. The sum in (1) is evaluated using the all-neighbor communication pattern when varying the partitioning approaches among graph, owner, and all-neigh.

Table 2
Number of messages sent among the processes while computing the sum in (1) using the all-neighbor communication pattern (top) and the owner communication pattern (bottom). Here, we use the partitioning approach corresponding to the communication pattern under consideration and compare with the graph approach. Best result in **bold**.

(a) *All-neighbor communication pattern*

		P									
Problem	Approach	2	4	8	16	32	64	128	256	512	1024
Small	graph	2	6	**20**	**62**	186	544	1690	–	–	–
	all-neigh	2	6	24	66	**178**	**502**	**1510**	–	–	–
Medium	graph	2	12	**46**	**156**	406	**898**	2008	4772	10,972	27,050
	all-neigh	2	12	48	174	**392**	918	**2004**	**4572**	**10,770**	**25,810**
Large	graph	2	12	**48**	148	**396**	**914**	**1934**	**4158**	**9036**	20,000
	all-neigh	2	12	50	**146**	412	952	2010	4322	9126	**19,812**

(b) *Owner communication pattern*

		P									
Problem	Approach	2	4	8	16	32	64	128	256	512	1024
Small	graph	4	12	**40**	**124**	**316**	**900**	**2388**	–	–	–
	owner	4	12	48	132	324	968	3320	–	–	–
Medium	graph	4	24	88	**292**	720	**1588**	**3412**	**7348**	**15,184**	–
	owner	4	24	88	304	**712**	1664	3592	7748	16,948	–
Large	graph	4	24	**92**	292	**728**	**1652**	**3540**	**7472**	**15,404**	**31,900**
	owner	4	24	100	**288**	744	1752	3684	7792	16,176	34,216

sizes are moderate. Indeed, any hyperedge size is no larger than the maximal number of different tetrahedra that can meet at a single node which is given by $\Delta_{max} = 36$ as already reported in Table 1. The minimum, maximum, and median hyperedge sizes given in that table indicate that the adaptive refinement taking place in the finite element discretization does not have a significant effect on the structure of the resulting hypergraphs. This is indeed true since the adaptive refinement, though typically causing the tetrahedra to dramatically change in their spatial distribution, mainly preserves the topology of their neighborship. Thus, the resulting hypergraphs are quite regularly structured. This way, the greedy approach of Algorithm 1 does not have the chance to differ significantly for all-neigh and owner.

Table 3
Problem sizes and characteristics of hypergraphs constructed from sparse matrices [31,32] together with the number of parts $P = \lceil\sqrt{|V|}\rceil$.

Matrix	$\|V\|$	$\|E\|$	Δ_{pin}	Δ_{min}	Δ_{med}	Δ_{max}	P
commanche_dual	7920	7920	31,680	4	4	4	89
aug3d	24,300	24,300	69,984	1	2	6	156
mario001	38,434	38,434	206,156	1	4	7	197
skirt	12,598	12,598	196,520	1	15	33	113
helm3d01	32,226	32,226	428,444	3	19.5	37	180
fe_tooth	78,136	78,136	905,182	3	17.5	39	280
sparsine	50,000	50,000	1,548,988	6	32	56	224
ex3stal	16,782	16,782	678,998	9	43	339	130
polyDFT	46,176	46,176	3,690,048	1	205	340	215
memplus	17,758	17,758	126,150	1	50	353	134
poli_large	15,575	15,575	33,074	1	47	491	125
lp_cre_b	77,137	9,648	260,785	1	121	844	278
guptal	31,802	31,802	2,164,210	3	499.5	8413	179
mipl	66,463	66,463	10,352,819	4	388	66,395	258
Stanford_Berkeley	683,446	683,446	7,583,376	1	495	83,448	827

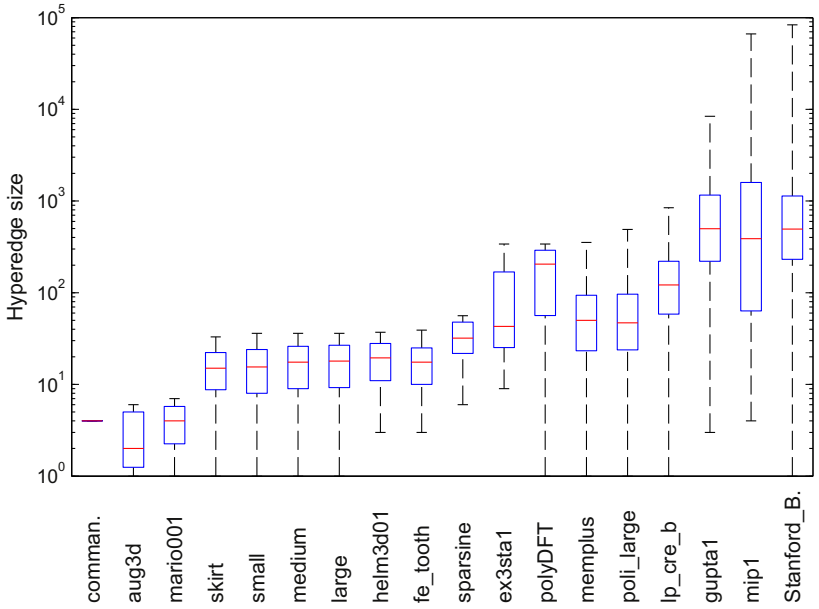

Fig. 5. Characteristics of hyperedge sizes. The hypergraphs are sorted by increasing maximum hyperedge size Δ_{max}. The bottom and top of the blue boxes are the 25th and 75th percentile. The red band near the middle of each box is the median hyperedge size Δ_{med}. The ends of the whiskers represent the minimum and maximum hyperedge sizes Δ_{min} and Δ_{max}, respectively. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

To illustrate potential differences in the partitioning strategies, we next also investigate hypergraphs with a greater variation in hyperedge size. We construct these hypergraphs from sparse matrices originating in a variety of fields, including but not limited to finite element simulations. The matrix `polyDFT` stems from the Tramonto density functional theory code [31] while all remaining matrices belong to the University of Florida sparse matrix collection [32]. We interpret each matrix column as a vertex with weight 1 and each matrix row as a hyperedge, containing all columns that have a nonzero entry in that row. The weight of a hyperedge is set to 1.

The set of hypergraphs constructed from sparse matrices is given in Table 3. This table summarizes the problem sizes as well as hypergraph characteristics. The rows are sorted by increasing maximum hyperedge size Δ_{max}. Here and throughout this section, we use this ordering to divide these matrices into two classes. The first class, given in the top part of Table 3, consists of matrices whose maximum hyperedge size is moderate. Matrices belonging to the second class are given in the bottom part and exhibit considerably larger maximum hyperedge sizes.

To justify this classification, Fig. 5 gives more details on the hyperedge sizes. For each matrix, this boxplot depicts several hyperedge size characteristics. The figure does not only show the data for all the matrices given in Table 3 but also contains the corresponding data for the three hypergraphs arising from the finite element solver DROPS; compare Table 1 with the

Table 4
Communication volume $C_{f_{\text{cut-net}}}(\mathcal{V})$ for the matrices from Table 3 divided into $P = \lceil\sqrt{|V|}\rceil$ parts, with 5% imbalance. Best result in **bold**.

Matrix	cut–net	owner	all-neigh
commanche_dual	**1698**	1.03×	1.07×
aug3d	**4985**	1.12×	1.17×
mario001	**7544**	1.05×	1.07×
skirt	**5033**	1.03×	1.11×
helm3d01	**20,741**	1.04×	1.08×
fe_tooth	**36,475**	1.02×	1.07×
sparsine	**48,841**	1.02×	1.02×
ex3stal	**15,843**	1.05×	1.06×
polyDFT	**42,212**	1.05×	1.06×
memplus	**7298**	1.16×	1.22×
poli_large	**136**	1.67×	2.60×
lp_cre_b	**2554**	1.71×	2.39×
guptal	**31,192**	1.02×	1.02×
mipl	**66,257**	1.00×	1.00×
Stanford_Berkeley	**23,798**	1.26×	1.73×

Table 5
Communication volume $C_{f_{\text{owner}}}(\mathcal{V})$ for the matrices from Table 3 divided into $P = \lceil\sqrt{|V|}\rceil$ parts, with 5% imbalance. Best result in **bold**.

Matrix	cut–net	owner	all-neigh
commanche_dual	1.04×	**1816**	1.02×
aug3d	1.44×	**5838**	1.02×
mario001	1.54×	**8189**	1.01×
skirt	1.10×	**6193**	1.04×
helm3d01	1.58×	**34,857**	1.02×
fe_tooth	1.37×	**52,735**	1.03×
sparsine	2.18×	**304,685**	1.04×
ex3stal	2.93×	60,716	**0.98**×
polyDFT	3.35×	**130,734**	1.04×
memplus	2.44×	**12,451**	1.05×
poli_large	3.14×	**554**	1.07×
lp_cre_b	2.44×	**17,251**	1.34×
guptal	4.76×	**92,323**	1.22×
mipl	12.50×	**179,405**	1.04×
Stanford_Berkeley	3.02×	**56,129**	1.19×

Table 6
Communication volume $C_{f_{\text{all-neigh}}}(\mathcal{V})$ for the matrices from Table 3 divided into $P = \lceil\sqrt{|V|}\rceil$ parts, with 5% imbalance. Best result in **bold**.

Matrix	cut–net	owner	all-neigh
commanche_dual	1.10×	1.01×	**3752**
aug3d	2.32×	1.01×	**12,140**
mario001	2.28×	1.00×	**16,800**
skirt	1.25×	1.01×	**14,597**
helm3d01	2.46×	1.00×	**105,702**
fe_tooth	1.94×	**0.99**×	148,569
sparsine	4.35×	**0.97**×	2,756,078
ex3stal	8.93×	1.10×	**316,068**
polyDFT	11.54×	1.13×	**676,827**
memplus	10.67×	1.10×	**68,310**
poli_large	22.58×	1.68×	**2251**
lp_cre_b	10.13×	1.35×	**213,318**
guptal	12.78×	2.65×	**1,203,979**
mipl	139.31×	1.09×	**1,071,572**
Stanford_Berkeley	53.33×	1.25×	**449,926**

three matrices labeled "small," "medium" and "large" in Fig. 5. Two classes of hypergraphs can clearly be distinguished in this figure. The matrices starting from the left up to sparsine fall into the first class. Since their hyperedge size characteristics are similar to those of the three DROPS hypergraphs we expect similar results, i.e., the resulting communication volume

Table 7
Number of messages sent with owner communication pattern, for the matrices from Table 3 divided into $P = \lceil\sqrt{|V|}\rceil$ parts, with 5% imbalance. Best result in **bold**.

Matrix	cut–net	owner	all-neigh
commanche_dual	1.08×	451	**0.99**×
aug3d	1.91×	1527	**1.00**×
mario001	2.37×	1069	**0.99**×
skirt	1.23×	532	**0.89**×
helm3d01	2.40×	2631	**0.94**×
fe_tooth	2.49×	3499	**0.92**×
sparsine	1.69×	25,737	**0.96**×
ex3stal	1.66×	7607	**0.99**×
polyDFT	2.77×	6933	**0.86**×
memplus	3.29×	4084	**0.96**×
poli_large	3.35×	948	**0.92**×
lp_cre_b	2.65×	14,254	**0.93**×
guptal	1.17×	26,843	**0.48**×
mipl	6.20×	7054	**0.89**×
Stanford_Berkeley	6.47×	20,530	**0.88**×

Table 8
Number of messages sent with all-neighbor communication pattern, for the matrices from Table 3 divided into $P = \lceil\sqrt{|V|}\rceil$ parts, with 5% imbalance. Best result in **bold**.

Matrix	cut–net	owner	all-neigh
commanche_dual	1.12×	1.02×	**447**
aug3d	2.31×	1.02×	**1537**
mario001	2.68×	1.01×	**1070**
skirt	1.40×	1.13×	**485**
helm3d01	2.70×	1.06×	**2713**
fe_tooth	2.94×	1.09×	**3491**
sparsine	1.53×	1.06×	**31,601**
ex3stal	1.64×	1.01×	**8975**
polyDFT	3.57×	1.16×	**6701**
memplus	1.52×	1.11×	**11,644**
poli_large	8.06×	1.77×	**1389**
lp_cre_b	2.29×	1.39×	**32,639**
guptal	1.60×	1.59×	**19,946**
mipl	1.00×	1.00×	**66,306**
Stanford_Berkeley	4.94×	1.59×	**85,913**

and message number should not depend too much on the partitioning. However, the second class is quite different. Here, the maximum hyperedge sizes are considerably larger and also the variability of the hyperedge sizes is larger. So, for that class, we could expect differences in the communication volume and message number when applying the greedy approach of Algorithm 1 to different partitionings.

We create a partition $\mathcal{V}$ of the vertices V of each $|E| \times |V|$ matrix with Δ_{pin} nonzeros into $P = \lceil\sqrt{|V|}\rceil$ parts, with imbalance $\varepsilon = 5\,\%$. The recorded communication volumes and message counts are averaged over 50 partitioning runs using three partitionings: cut–net, owner, and all-neigh. Since some of these hypergraphs do not stem from problems based on meshes, we do not use METIS to determine a partitioning graph.

A comparison of the communication volumes is carried out in Tables 4–6, which list the obtained average volumes $C_{f_{\text{cut-net}}}$, $C_{f_{\text{owner}}}$, and $C_{f_{\text{all-neigh}}}$ for the three different partitioning strategies. To facilitate a comparison of the three partitioning strategies, the communication volumes of two partitioning approaches are written as multiplicative factors. For example, consider the first row of Table 4. Here, the absolute communication volume $C_{f_{\text{cut-net}}}$ of 1698 is given for the approach cut–net. The corresponding value of $C_{f_{\text{cut-net}}}$ is about 1.07 times as large for all-neigh. In these tables, the smallest communication volumes are denoted in bold font. An analysis of these tables indicates that each partitioning strategy yields the smallest volume for the corresponding metric in almost all cases. In particular, Table 6 shows that all-neigh is effective for minimizing $C_{f_{\text{all-neigh}}}$. However, for problems with a regular structure (those with small hyperedge sizes, such as the finite element matrix fe_tooth), the all-neigh partitioning strategy does not yield any improvement over owner in terms of $C_{f_{\text{all-neigh}}}$. For more complicated matrices (such as guptal), the all-neigh strategy is very effective. When varying the partitioning approaches in Tables 4–6, the differences in communication volume tend to be small for the class of matrices with a moderate maximum hyperedge size. At the same time, the matrices in the bottom part of these tables that correspond to the class with larger maximum hyperedge size tend to show significantly larger differences in communication volume.

Next, we also analyze the number of messages for the owner and all-neighbor communication patterns in Tables 7 and 8. Here, we see that using the all-neigh partitioning strategy results in having to send the smallest number of messages for both communication patterns. This makes the $\lambda(\lambda-1)$-metric interesting for partitioning on systems where creating messages is time consuming. Similarly to the communication volume, the gains in the number of messages are small for regularly structured matrices but large for matrices with more irregular structure. Take `aug3d` as an illustrative example for the class of matrices with moderate maximum hyperedge size. In both tables there is almost no difference in the message number between all-neigh and owner. On the other hand, `guptal` taken from the class of matrices with larger maximum hyperedge size shows, in both tables, a considerable difference in the message number when comparing all-neigh and owner.

Therefore, in summary, the approach from Section 4 provides an effective way to minimize different communication volume metrics. Furthermore, the $\lambda(\lambda-1)$-metric is also useful for minimizing the total number of messages, but for problems where the hyperedge sizes are small, the gains are also small.

6. Conclusion

In this work, we have introduced a hypergraph model for the distribution of a finite element triangulation to processes of a distributed-memory computer. The crucial feature of this hypergraph model is its capability to represent the communication volume exactly rather than giving an approximation. To this end, we presented a new metric for hypergraph partitioning, the $\lambda(\lambda-1)$-metric, which was inspired by the all-neighbor communication pattern occurring in the finite element software package DROPS. This metric is an alternative to the common $(\lambda-1)$-metric, which accurately measures the communication volume in an owner communication pattern as well as in parallel sparse matrix–vector multiplication. Both metrics are special instances of a more general family of metrics, defined by a cost function $f(\lambda)$ which is a non-decreasing function of the connectivity λ of a hyperedge of the hypergraph. The three metrics studied in our work can be ordered by increasing dependence on the connectivity as: $\min(\lambda-1,1), 2(\lambda-1)$, and $\lambda(\lambda-1)$. Furthermore, we have given an explicit formula (12) for hypergraph bipartitioning with the new metric which enables implementation using a standard hypergraph partitioner, provided it has the option of setting hyperedge weights.

The experimental results for the finite element problems show that, though small, there are differences in the communication volume between graph and hypergraph partitioning. Compared to a graph partitioning, we observed reductions in the communication volume of up to about 5% for hypergraph partitioning. Also, the number of messages as well as the communication time needed to evaluate linear algebra operations on the distributed triangulation tend to be reduced when using a hypergraph model rather than an undirected graph model. On the other hand, the difference in communication volume between the $2(\lambda-1)$-metric and the $\lambda(\lambda-1)$-metric is almost negligible. Still, the communication time obtained from a partitioning based on the all-neighbor metric is smaller than when using a partitioning based on the owner metric. For 1024 processes we observed roughly a factor of two between these two communication times, see Fig. 4(c).

In finite element meshes, the number of neighbors of a given node does not vary very much, which accounts for the small differences observed in communication volume and which explains the continued use in many applications of graph partitioning instead of the more accurate hypergraph partitioning. In situations, however, where the number of neighbors of a vertex varies wildly, see Table 6, the new metric is suited for applications that follow the all-neighbor approach in their implementation.

Acknowledgements

The development of the finite element solver DROPS was supported by the Deutsche Forschungsgemeinschaft (DFG) within the SFB 540 "Model-based Experimental Analysis of Kinetic Phenomena in Fluid Multi-phase Reactive Systems," and is being developed in a collaboration with the chair for numerical mathematics (LNM) at RWTH Aachen University. We thank Alin Bastea, our colleagues of the Center for Computing and Communication at RWTH Aachen University, and Ali Rostami for helping us to gather data and prepare it for publication. Finally, we also thank the three anonymous reviewers whose comments and suggestions improved the quality of this paper.

References

[1] K. Schloegel, G. Karypis, V. Kumar, Graph partitioning for high-performance scientific simulations, in: J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, A. White (Eds.), Sourcebook of Parallel Computing, Morgan Kaufmann, San Francisco, 2003, pp. 491–541.
[2] Ü.V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 673–693.
[3] B.B. Cambazoglu, C. Aykanat, Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids, IEEE Transactions on Parallel and Distributed Systems 18 (2007) 3–16.
[4] G. Khanna, N. Vydyanathan, T. Kurc, Ü. Çatalyürek, P. Wyckoff, J. Saltz, P. Sadayappan, A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O, Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005), vol. 2, IEEE Computer Society, Washington, DC, USA, 2005, pp. 792–799.
[5] M. Strout, N. Osheim, D. Rostron, P. Hovland, A. Pothen, Evaluation of hierarchical mesh reorderings, in: G. Allen, J. Nabrzyski, E. Seidel, G. van Albada, J. Dongarra, P. Sloot (Eds.), Computational Science – ICCS 2009, Lecture Notes in Computer Science, vol. 5544, Springer, Berlin, 2009, pp. 540–549.
[6] C. Chevalier, E. Boman, An accurate hypergraph model for mesh partitioning, in: Poster at SIAM Workshop on Combinatorial Scientific Computing (CSC 2009), October 29–31, Seaside, CA, 2009.

[7] E.G. Boman, Ü.V. Çatalyürek, C. Chevalier, K.D. Devine, Parallel partitioning, coloring, and ordering in scientific computing, in: U. Naumann, O. Schenk (Eds.), Combinatorial Scientific Computing, CRC Press, 2012, pp. 351–371.
[8] K. Devine, L. Diachin, J. Kraftcheck, K.E. Jansen, V. Leung, X. Luo, M. Miller, C. Ollivier-Gooch, A. Ovcharenko, O. Sahni, M.S. Shephard, T. Tautges, T. Xie, M. Zhou, Interoperable mesh components for large-scale, distributed-memory simulations, Journal of Physics: Conference Series 180 (1) (2009) 012011.
[9] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, C. Vaughan, Zoltan data management services for parallel dynamic applications, Computing in Science and Engineering 4 (2) (2002) 90–97.
[10] S. Groß, A. Reusken, Numerical methods for two-phase incompressible flows, Springer Series in Computational Mathematics, first ed., vol. 40, Springer, Berlin, Heidelberg, 2011.
[11] O. Fortmeier, T. Henrich, H.M. Bücker, Modeling data distribution for two-phase flow problems by weighted graphs, in: Proceedings of the Workshop on Parallel Systems and Algorithms, PARS'10, VDE, 2010, pp. 31–38.
[12] O. Fortmeier, A. Bastea, H.M. Bücker, Graph model for minimizing the storage overhead of distributing data for the parallel solution of two-phase flows, in: Proceedings of the IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing PDSEC'11, 2011, pp. 1308–1316.
[13] Ü.V. Çatalyürek, E. Boman, K.D. Devine, D. Bozdağ, R. Heaphy, L.A. Riesen, Hypergraph-based dynamic load balancing for adaptive scientific computations, in: IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), IEEE, 2007, pp. 1–11.
[14] Ü.V. Çatalyürek, E.G. Boman, K.D. Devine, D. Bozdağ, R.T. Heaphy, L.A. Riesen, A repartitioning hypergraph model for dynamic load balancing, Journal of Parallel and Distributed Computing 69 (2009) 711–724.
[15] J.D. Teresco, K.D. Devine, J.E. Flaherty, Partitioning and dynamic load balancing for the numerical solution of partial differential equations, in: A.M. Bruaset, P. Bjørstad, A. Tveito (Eds.), Numerical Solution of Partial Differential Equations on Parallel Computers, Springer, Berlin, Heidelberg, 2006, pp. 55–88.
[16] A. Pinar, B. Hendrickson, Graph partitioning for complex objectives, in: Proceedings Eighth International Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001), IEEE Press, Los Alamitos, CA, 2001, p. 121.
[17] K. Kaya, F.H. Rouet, B. Uçar, On partitioning problems with complex objectives, in: M. Alexander et al. (Eds.), Euro-Par 2011: Parallel Processing Workshops, Lecture Notes in Computer Science, vol. 7155, Springer, Berlin/Heidelberg, 2012, pp. 334–344.
[18] O. Fortmeier, H.M. Bücker, Parallel re-initialization of level set functions on distributed unstructured tetrahedral grids, Journal of Computational Physics 230 (12) (2011) 4437–4453.
[19] B. Vastenhouw, R.H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix–vector multiplication, SIAM Review 47 (1) (2005) 67–95.
[20] Ü.V. Çatalyürek, C. Aykanat, PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH http://bmi.osu.edu/~umit/software.htm (1999).
[21] G. Karypis, V. Kumar, METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 4.0, Technical Report, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1998.
[22] E.S. Seol, M.S. Shephard, Efficient distributed mesh data structure for parallel automated adaptive analysis, Engineering with Computers 22 (3) (2006) 197–213.
[23] O. Sahni, M. Zhou, M.S. Shephard, K.E. Jansen, Scalable implicit finite element solver for massively parallel processing with demonstration to 160K cores, in: Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and, Analysis, SC'09, 2009, pp. 68:1–68:12.
[24] Y.F. Hu, K.C.F. Maguire, R.J. Blake, A multilevel unsymmetric matrix ordering algorithm for parallel process simulation, Computers and Chemical Engineering 23 (2000) 1631–1647.
[25] L.A. Sanchis, Multiple-way network partitioning, IEEE Transactions on Computers 38 (1) (1989) 62–81.
[26] M. Garey, D. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science 1 (3) (1976) 237–267.
[27] T.N. Bui, C. Jones, Finding good approximate vertex and edge partitions is NP-hard, Information Processing Letters 42 (3) (1992) 153–159.
[28] O. Fortmeier, Parallel re-initialization of level set functions and load balancing for two-phase flow simulations, Ph.D. Thesis, Computer Science Department, RWTH Aachen University, 2011.
[29] E. Gross-Hardt, E. Slusanschi, H.M. Bücker, A. Pfennig, C.H. Bischof, Practical shape optimization of a levitation device for single droplets, Optimization and Engineering 9 (2) (2008) 179–199.
[30] E. Gross-Hardt, A. Amar, S. Stapf, A. Pfennig, B. Blümich, Flow dynamics measured and simulated inside a single levitated droplet, Industrial & Engineering Chemistry Research 1 (2006) 416–423.
[31] L.J.D. Frink, A.G. Salinger, M.P. Sears, J.D. Weinhold, A.L. Frischknecht, Numerical challenges in the application of density functional theory to biology and nanotechnology, Journal of Physics: Condensed Matter 14 (46) (2002) 12167–12187.
[32] T.A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Transactions on Mathematical Software 38 (1) (2011) 1:1–1:25.