HAL
open science

On Partitioning Two Dimensional Finite Difference Meshes for Distributed Memory Parallel Computers

Anael Grandjean, Bora Uçar

To cite this version:

Anael Grandjean, Bora Uçar. On Partitioning Two Dimensional Finite Difference Meshes for Distributed Memory Parallel Computers. 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014), Feb 2014, Turin, Italy. pp.9 - 16, 10.1109/PDP.2014.10 . hal-01111292

HAL Id: hal-01111292

https://inria.hal.science/hal-01111292v1

Submitted on 30 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On partitioning two dimensional finite difference meshes for distributed memory parallel computers

Anael Grandjean
LIP (UMR CNRS-ENS Lyon-INRIA-UCBL),
46, allée d'Italie, ENS Lyon, F-69364, France
Email:anael.grandjean@ens-lyon.fr

Bora Uçar
CNRS and LIP (UMR CNRS-ENS Lyon-INRIA-UCBL),
46, allée d'Italie, ENS Lyon, F-69364, France
Email: bora.ucar@ens-lyon.fr

***Abstract*—We investigate the problem of partitioning finite difference meshes in two dimensions among the processors of a parallel computer. The objective is to achieve a perfect load balance while minimizing the communication cost. There are well-known graph, hypergraph, and geometry-based partitioning algorithms for this problem. The known geometric algorithms have linear running time and obtain the best results for very special mesh sizes and processor numbers. We propose another geometric algorithm. The proposed algorithm is linear; is applicable to much more cases than some well-known alternatives; obtains better results than the graph partitioning algorithms; obtains better results than the hypergraph partitioning algorithms almost always. Our algorithm also obtains better results than a known asymptotically-optimal algorithm for some small number of processors. We also catalog related theoretical results.**

I. Introduction

In many physical applications, the finite difference method is used to approximate solutions of partial differential equations on a computational physical domain. In this method, the physical domain is discretized and a computational domain in the form of a mesh is obtained. Often times, the resulting computational domain is a two dimensional (2D) rectangular mesh, where the core of the computation is comprised of the interactions of each mesh point with its immediate neighbors. In a parallel computing environment, such meshes should be partitioned among the processors with the common objectives of load balance (e.g., almost equal number of mesh points per processors) and reduced communication cost (e.g., lower total communication volume). The underlying problem is usually cast as a graph or hypergraph partitioning problem, sometimes taking advantage of the underlying geometry. We propose a geometric algorithm for the 2D meshes. The proposed algorithm has a running time linearly proportional to the number of mesh points (a common trait in geometrical algorithms). It is applicable to almost all processor numbers and mesh sizes, whereas algorithms similar in nature are applicable to a restricted set of processor numbers and mesh sizes. It obtains better results than a well-known hypergraph partitioning tool almost always. We also demonstrate some theoretical results about partitioning 2D meshes. In particular we develop formulas for the total communication volume of some known methods, and discuss what can be obtained from a hypothetical optimal graph partitioning algorithm.

When formulating the mesh partitioning problem as the standard graph partitioning problem, the mesh points correspond to the vertices of the graph where two vertices are connected by an edge iff they correspond to two interacting mesh points. The graph partitioning method then searches for a balanced partition of the mesh points among a given number of parts with the objective of minimizing the number of interacting pairs that are assigned to different processors. There are a number of software libraries that provide necessary graph partitioning functions, see e.g., MeTiS [18] and Zoltan [7]. The objective function, known as the edge cut, is an approximate metric relating to the communication cost [9], [15], [16].

When formulating the mesh partitioning problem as the standard hypergraph partitioning problem, each mesh point corresponds to a vertex as in the graph case. Each mesh point corresponds also to a hyperedge which contains the vertices associated with the corresponding mesh point and its interacting neighbors. The interactions represented by a hyperedge are going to be computed by the processor that holds the associated mesh point. The hypergraph partitioning method then searches for a balanced partition of the mesh points among a given number of parts with the objective of minimizing the sum, over all hyperedges, of the number of parts among which the interacting vertices contained in a hyperedge are partitioned. Hypergraph partitioning functions are available in a number software libraries such as hMeTiS [19], Mondriaan [24], PaToH [10], and Zoltan [7]. The related objective function, known as the connectivity-1 metric, encodes the total communication volume exactly [9].

Most of the geometry based partitioning methods (see for example the methods discussed by Gilbert et al. [14]) are essentially graph partitioning methods that minimize the edge cut. Instead of using general purpose graph partitioning algorithms, these methods partition the computational domain by partitioning its physical domain using geometric objects such as lines, planes, and spheres. These methods work for any geometric mesh with certain characteristics (the finite difference methods we consider here have the required characteristics). There are also other geometric methods based on space filling curves (see for example a few of them implemented in Zoltan [7]) which aim to achieve load balance without explicit effort to reduce the communication cost. Another set of methods [5, Section 4.8] (also [3], [22]) achieve load balance and aim to minimize the communication cost by

minimizing the total communication volume and the maximum communication volume per processor.

Our contributions are as follows. We discuss when the partitioning methods presented by Bisseling [5, Section 4.8] and Bisseling and McColl [3] are applicable to solve the mesh partitioning problem that we address. We give an explicit formula for the total volume of communication resulting from such partitions. The formulae include a careful analysis of the boundary of the given finite mesh; from the discussions in the cited resources, one can obtain an asymptotical formula for an infinite mesh. We also explicitly state conditions on the number of processors and the mesh sizes on which the methods work. We propose an algorithm for partitioning 2D meshes. The proposed algorithm is an improvement of MeshPart of Uçar and Çatalyürek [22]. The proposed algorithm applies to much general cases than its predecessor and obtains better results almost always. For a small number of parts, it also obtains better results than the methods presented by Bisseling [5, Section 4.8] and Bisseling and McColl [3] (these latter methods are asymptotically better).

The organization of this paper is as follows. In the next section we discuss necessary background material. Section III contains a survey of the known algorithms and theoretical results relating to those algorithms. We also collect some related discrete geometrical results in the same section. Whereas there are numerous investigations of theoretical and practical nature on the edge cut optimizing partitioning, there are only a few studies focusing on the communication volume optimizing partitions. Among the theoretical results in the paper, those that are with a proof are ours. Our novel algorithm for partitioning 2D meshes is presented in Section IV and evaluated in Section V, before concluding the paper.

II. Background and problem definition

We use $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ as the standard ceiling and floor functions. The notation $[a]$ rounds a to the nearest integer.

We assume that the processors of the parallel system are indexed using a 2D virtual processor-mesh topology according to the given computational mesh. The K processors will be organized in a $P \times Q$ topology. Here, the processor k, for $1 \leq k \leq K$ is associated with two integers (p, q) where $p \in \{0, \ldots, P-1\}$ and $q \in \{0, \ldots, Q-1\}$ so that $k = p + qP + 1$. Throughout the discussion, we use part and processor interchangeably, and suppose that when a mesh is partitioned into K parts, each part would be assigned to a unique processor. By a processor, we mean a processing element in a parallel computer.

A. Meshes corresponding to five-point stencil operations and their partitioning

Assume that a rectangular 2D domain is discretized with the five point stencil and a mesh of size $X \times Y$ is obtained whose points are placed at integer locations. Two points (x_1, y_1) and (x_2, y_2) of the resulting mesh are neighbors iff $|x_1 - x_2| + |y_1 - y_2| = 1$. Figure 1 shows a sample mesh. These meshes are used to obtain finite difference approximations of

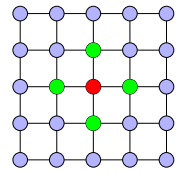


Fig. 1. A 2D mesh resulting from the use of five-point stencil.

the derivatives at the discrete points, where the approximation at a point is improved using the approximations at the point itself and its neighbors.

In order to efficiently parallelize the finite difference computations in a distributed memory setting, one has to achieve load balance among processors and reduce the communication cost. One can achieve load balance by equally partitioning the computations among the processors. Since the interior mesh points are involved in an equal number of interactions, one can achieve load balance by assigning almost equal number of mesh points to the processors. Furthermore, in the case where there are many unknowns associated with each mesh point, one balances the memory requirements of the processors as well. Although in practical settings one can allow slight imbalance among processors, we address the exact load balance case for two reasons. First, if one allows the maximum loaded processor to have ε more load than average, one can use $\lfloor K/(1+\varepsilon) \rfloor$ processors and leave the others idle. If for example $K = 1024$ and $\varepsilon = 0.03$ (a typical value, see the 10th DIMACS implementation challenge [1]), one can leave about 30 processors empty, wasting resources. Second, the partitioning problem is easier to analyze in the perfect load balance case, which helps a practitioner gauge different algorithms. The case with some allowed imbalance is not any easier algorithmically [13]. In order to reduce the communication cost, one has to minimize a complex function of the total communication volume, the total number of messages, and the maximum volume and number of messages per processor, in sends [21] or in sends and receives [4]. In order to do so, usually a two-step approach is followed where the first step reduces the total communication volume (other metrics are handled in the second step while keeping the total volume of communication intact). We address the total communication cost metric here as in the first step of more sophisticated approaches. Furthermore, as our target problem concerns regular meshes, the other communication cost metrics are very likely to be reduced with the reduced total communication volume (the communications are among the processes that have neighboring regions of the meshes).

B. Cartesian partitioning

One of the most simple partitioning methods is the Cartesian partitioning method. Given a mesh of size $X \times Y$ to be partitioned among $P \times Q$ processors, this methods partitions the x and y coordinates by, respectively, $P-1$ and $Q-1$ planes, where the planes partitioning a coordinate are equally spaced (e.g., X/P). This way, given a mesh point (n_x, n_y), one simply computes $p_x = \lfloor P \times n_x / X \rfloor$, $p_y = \lfloor Q \times n_y / Y \rfloor$, and assign the point (n_x, n_y) to the part (p_x, p_y).

The following fact is easy to see as a point on one side of the plane is separated from only one of its neighbors.

Fact 1. *The Cartesian partitioning of a mesh of size $X \times Y$ among $P \times Q$ processors necessitates a total communication volume of*

$$2\left((P-1)Y + (Q-1)X\right) \ . \tag{1}$$

The maximum volume of communication of a part is

$$2\left(X/P + Y/Q\right) \ .$$

C. Graph partitioning

Let $G = (V, E)$ be a graph with the vertex set V and edge set E. The standard graph partitioning asks for a partition of the vertices of G into K parts, for a given integer $K \geq 2$, such that the parts have almost equal number of vertices, and the total number of edges having vertices in two different parts, referred to as edge cut, is minimized. In general, one uses an imbalance parameter in the problem definition to define an allowable upper bound for the part weights. This problem is $\mathcal{NP}$-hard for general graphs [8].

D. Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$. Every net is a subset of vertices. Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part is non-empty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. The partitioning constraint is to maintain a balance on the number of vertices per part. In a partition Π of $\mathcal{H}$, a net that contains at least one vertex in a part is said to connect that part. *Connectivity* λ_j of a net n_j denotes the number of parts connected by n_j. A particular partitioning objective function, called the connectivity-1 metric, is to minimize the cutsize defined as follows

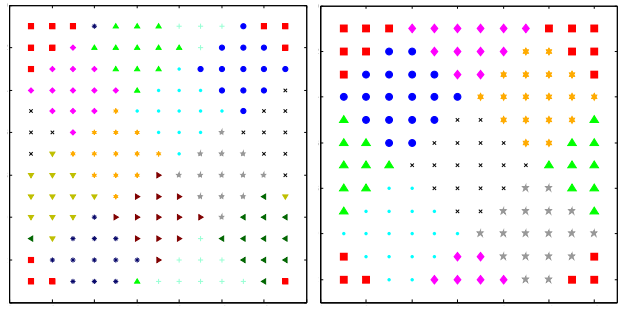
$$cutsize(\Pi) \quad = \quad \sum_{n_j \in \mathcal{N}} (\lambda_j - 1) \ . \tag{2}$$

This problem is known to be $\mathcal{NP}$-hard [20].

III. A SURVEY OF KNOWN METHODS AND RESULTS

A. Partitioning five-point meshes

Bisseling [5, Section 4.8] and Bisseling and McColl [3] discuss partitioning two dimensional domains using digital diamonds. A digital diamond is a closed ℓ_1-sphere defined by a center point (c_x, c_y) and a radius ρ. Such a diamond contains all mesh points with a Manhattan distance at most ρ of the center, i.e., a point (p_x, p_y) is included in a diamond having the center (c_x, c_y) and the radius ρ if $|p_x - c_x| + |p_y - c_y| \leq \rho$. A digital diamond of radius ρ thus contains $2\rho^2 + 2\rho + 1$ mesh points. In order to partition a two-dimensional mesh of size $X \times Y$ into K parts with digital diamonds, one first finds the radius by using the equation $K(2\rho^2 + 2\rho + 1) = XY$, assuming $\rho < \min\{X, Y\}$. Then, the center points of the K diamonds are placed periodically at the integer linear combinations of $(\rho, \rho+1)$ and $(-\rho-1, \rho)$. Then, each mesh point is assigned to the closest center point (at a Manhattan distance less than ρ). Figure 2(a) shows the partitioning of the 13×13 mesh into 13 parts, each represented by a digital diamond of radius 2.



(a) 13-way partitioning of the 13×13 mesh using digital diamonds. (b) 8-way partitioning of the 12×12 mesh using basic diamonds.

Fig. 2. Examples with different partitioning methods for 2D meshes. Different color and symbol pairs are used to show different parts.

Apart from the equation $XY = K(2\rho^2 + 2\rho + 1)$, one also has to satisfy more constraints to be able to partition a finite mesh. The next lemma gives these constraints. We assume $XY \geq K$ so that there is at least one mesh point per part.

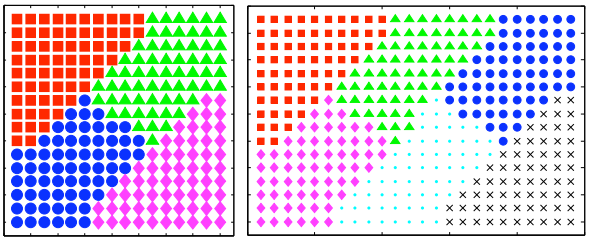
Lemma 1. *Partitioning with digital diamonds is possible only if $2\rho^2 + 2\rho + 1 = XY/K$, and $\frac{\rho K}{Y}$ and $\frac{\rho K}{X}$ are integers.*

Proof: If the first center is at $(1, 1)$, the point $(X+1, 1)$ and the point $(1, Y+1)$ have to be centers too, otherwise some parts will overlap, and we will not have a proper partition. Using the characterization of the centers given above (from Bisseling and McColl [3]) we have: $(X, 0) = \alpha(\rho, \rho+1) + \beta(-\rho-1, \rho)$ and $(Y, 0) = \gamma(\rho, \rho+1) + \delta(-\rho-1, \rho)$. For some $\alpha, \beta, \gamma, \delta$ integers and a radius ρ. This gives us $\beta = -\frac{\rho+1}{\rho}\alpha$ and $\delta = -\frac{\rho+1}{\rho}\gamma$. Then $\alpha = \frac{\rho K}{Y}$ and $\gamma = \frac{\rho K}{X}$. ■

Next we give the total volume of communication formula for the digital diamonds.

Lemma 2. *The total volume of communication resulting from partitioning a mesh of size $X \times Y$ with the digital diamonds into K parts is $(4\rho+4)K - 4 - 2\left\lfloor \frac{X}{(2\rho+1)(\rho+1)} \right\rfloor - 2\left\lfloor \frac{Y}{(2\rho+1)(\rho+1)} \right\rfloor$.*

Proof: We will proceed to show that the total volume of communication can be computed by first computing the total volume of communication assuming the round-around connections, rendering the mesh a torus, and then by subtracting the volume of communication on the border of the mesh. The volume for a perfect diamond is $4\rho + 4$, and hence we have a total volume of communication of $(4\rho + 4)K$, assuming the round-around connection. We gain volume on the border (when we ignore the round-around connections) only when a vertex is connected to a part with only one edge which is proper to the torus. For the diamond decomposition this happens only when a center is at distance exactly ρ from the border of the mesh. First, let us count the centers at distance ρ from the left border of the mesh. As $(1, 1)$ is a center, the point $(\rho+1, \rho+2)$ is a center, and the next one is $(\rho+1, (\rho+2)(\rho+1) + \rho^2 + 1)$. Therefore, we have $1 + \left\lfloor \frac{X}{(2\rho+1)(\rho+1)} \right\rfloor$ centers at distance ρ from the left border.



(a) Quadrisecting a 16×16 mesh.

(b) Quadrisection is extended to 6-way partitioning of a mesh of size 16×24.

Fig. 3. The MeshPart algorithm's [22] output.

This is exactly the same for the other borders. Subtracting from $(4\rho+4)K$ gives the announced formula. ■

In order to generalize the digital diamonds so as to cover finite meshes of certain sizes, one trims off a layer of points from the north- and south-eastern borders [5, Section 4.8]. The resulting diamond-like shape is called the basic diamond. A basic diamond with a radius ρ contains $2\rho^2$ mesh points. In order to partition a two-dimensional mesh of size $X\times Y$ into K parts, one first finds the radius by using the equation $K2\rho^2 = X \times Y$, assuming $\rho < \min\{X, Y\}$. Then the center points of K basic diamonds are placed periodically at the integer linear combinations of (ρ, ρ) and $(-\rho, \rho)$, and the mesh points are assigned to the parts each corresponding to a basic diamond. Figure 2(b) shows the partitioning of the 12×12 mesh into 8 parts, each being a basic diamond of radius 2.

The next lemma gives the constraints on the partitioning problems that can be solved with basic diamonds, again assuming $\rho > 0$ (there is at least one mesh point per part).

Lemma 3. *Basic diamonds works only if $K\rho^2 = XY$, and $\frac{X}{2\rho}$ and $\frac{Y}{2\rho}$ are integers.*

Proof: As in the proof of Lemma 1, we want $(X, 0) = \alpha(\rho, \rho)+\beta(-\rho, \rho)$ and $(Y, 0) = \gamma(\rho, \rho)+\delta(-\rho, \rho)$. This gives that $\alpha = -\beta$ and hence $X = 2\alpha\rho$, so that $\frac{X}{2\rho}$ should be an integer. Similar conclusion holds for $\frac{Y}{2\rho}$. ■

Next we give the total volume of communication formula for the basic diamonds.

Lemma 4. *The total volume of communication volume resulting from partitioning a mesh of size $X \times Y$ with the basic diamonds into K parts is $(4\rho+2)K - \frac{X}{\rho} - \frac{Y}{\rho}$.*

Proof: We use the same calculation method as in Lemma 2. The volume of a basic diamond is $4\rho + 2$. There are no centers saving communication at the bottom border. $\frac{Y}{2\rho}$ centers at the left and right borders save one volume of communication, resulting in a total saving of Y/ρ. There are $\frac{X}{2\rho}$ centers at the top border that save communication, where each one saves 2, because of the shape of the basic diamond. Subtracting the savings from the total volume assuming the torus connection gives the result. ■

Uçar and Çatalyürek [22] propose another algorithm for partitioning a five point mesh of size $X \times Y$ among $P \times Q$ parts. The proposed algorithm, called MeshPart, first partitions a square submesh of $4 \times \frac{X\times Y}{P\times Q}$ points into four, and then extends this partition to the original mesh. Figure 3(a) displays a 4-way partitioning of a mesh of size 16×16 which is then extended to a 6-way partitioning of a mesh of size 16×24 shown in Fig. 3(b). The parts at the corners are kept intact after the quadrisection step.

Uçar and Çatalyürek [22] state that the total communication volume of their algorithm is

$$(3PQ-(P+Q)-1)X/P+(P-1)(3Q-5)+(Q-1)(3P-5),$$

for a mesh of size $X \times Y$ to be divided into $P \times Q$ parts. For the algorithm to work, and the total communication volume formula above to hold, there are a number of restrictions: $X/P = Y/Q$, and $XY/(PQ)$ should be divisible by 32.

We note that the discussed three methods can be used to partition a scaled version of the mesh (or with proper dimensions). However, this would require a rebalancing, which will compound comparisons, and theoretical investigations.

B. Partitioning meshes with graphs

Consider a 2D mesh generated using the 5-points stencil. This mesh can be partitioned with a standard graph partitioning algorithm by representing each mesh point as a vertex and each interacting pair as an edge. The graph partitioning algorithm will then obtain a balanced partition of the mesh points among given K processors and will minimize the edge cut. Although the general problem is $\mathcal{NP}$-hard, in certain cases the regular mesh problem is not. These are highlighted below.

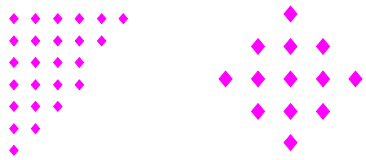
Lemma 5. *In partitioning a mesh of size $X\times Y$ among $P\times Q$ processors, if $X/P = Y/Q$ then the minimum edge cut is obtained by a Cartesian partitioning.*

Proof: We use again the fact that the total edge cut can be computed by first computing the total edge cut assuming torus connections and then by subtracting the cut edges which are proper to the torus. The optimal edge cut on the torus is at least the optimal edge cut for a part times the number of parts. Hence, the optimal edge cut of a mesh is at least the optimal edge cut for a part times the number of parts minus the edges of the torus which do not belong to the mesh.

We prove that the optimal edge cut of a part of size S^2 is $4S$. Let L be a part of size S^2, a be the number of planes where x is constant in which there is a point of L, b be the number of planes where y is constant in which there is a point of L. We have $ab \geq S^2$ and $edgecut(L) > 2ab$. This system has a unique minimum for the edge cut when $a = b = S$. The optimal edge cut is therefore $4\sqrt{\frac{XY}{PQ}}PQ - 2X - 2Y$, which is the edge cut of the Cartesian partitioning. ■

Bezrukov and Rovan [2, Theorem 3] show that the minimum edge cut in a partition of a cubic mesh (in d dimension, with all dimensions having equal length) is asymptotically equivalent to that of the Cartesian partitioning. When the number of parts is the dth power of an integer, the Cartesian partitioning gives the optimal edge cut—see the lower bound formula of Bezrukov and Rovan [2, Theorem 1].

Since in a Cartesian partitioning the edge cut is equal to the half of the total communication volume (resulting from the same partition), graph partitioning with the objective of minimizing the edge cut can at its best obtain a total



(a) At a corner. (b) Interior.

Fig. 4. Pictorial description of the optimal shapes [25]; triangle-like in 4(a) and diamond in 4(b).

communication volume equal to that of the Cartesian partitioning (1), while partitioning meshes. In practical settings, a little imbalance among the part weights is allowed so that the partitions would differ.

Another related result is given by Bollobás and Leader [6]. They show that a set S of points inside a d-dimensional cube (all dimensions of length n) touches at least n^{d-1} edges if $n^d/4 \leq |S| \leq 3n^d/4$. Their result therefore can be used to show that the Cartesian partitioning into 4 parts gives the best edge cut. Hence, the optimal graph partitioning can only achieve a total communication volume of $4X$ when partitioning a mesh of size $X \times X$ among 4 processors.

C. Partitioning meshes with hypergraphs

Consider a 2D mesh generated using the 5-points stencil. In order to formulate the mesh partitioning problem as the standard hypergraph partitioning problem, the hypergraph is created as follows. Each mesh point corresponds to a vertex. Each mesh point corresponds also to a hyperedge which contains the vertices associated with the corresponding mesh point and its interacting points. The hypergraph partitioning algorithm will then obtain a balanced partition of the mesh points among the K processors and will minimize the connectivity-1 metric which corresponds to the total communication volume. As for each point n_{ij}, if the corresponding net h_{ij} connects λ parts, a message from the processor that holds n_{ij} will be sent to other processors corresponding to the parts in the connectivity set of h_{ij}. This is because of the fact that the vertex n_{ij} will be in a net corresponding to a vertex in the net h_{ij}.

Not much is known about the problem of partitioning the hypergraph corresponding to a 5-point stencil mesh among K parts. Let S be a set of mesh points. Define the vertex boundary of S as the set of points not in S and are neighbors of a vertex in S. Then, the hypergraph partitioning cutsize (2) or the total communication volume, can be stated as the sum of the cardinalities of the vertex boundaries of each part. Once this correspondence is established, one can find some results in the literature.

Lemma 6 (Theorem 1 of Wang and Wang [25])**.** *Order the mesh points according to their Manhattan distance to the origin, in a nondecreasing order. Let S be the set of the first F points in the said order. Then S has the minimum vertex boundary among all point sets of cardinality F.*

The theorem can be used to show that a part placed at the corner of a finite 2D mesh will look like as shown in Fig. 4(a).

Lemma 7 (Theorem 2, Wang and Wang [25])**.** *Take a mesh point as the origin and order the mesh points according to their Manhattan distance to the the chosen origin. Let S be the set of first F points. Then S has the minimum vertex boundary among all point sets of cardinality F in an infinite mesh.*

The theorem can be used to show that a single part inside a finite 2D mesh will look like as shown in Fig. 4(b) or as in digital diamonds of Bisseling and McColl [3].

These results are related to optimizing the shape of a single part (or all in an infinite mesh). To the best of our knowledge, there is no work that examines the total vertex boundary, or the total communication volume, where a given finite mesh is partitioned into K equal cardinality parts.

IV. Partitioning algorithms for 2D meshes

Here we are going to partition a two dimensional mesh of size $X \times Y$ into $P \times Q$ parts, assuming that $\frac{X}{P}$ and $\frac{Y}{Q}$ are integers. The proposed algorithm (Fig. 5) runs in three steps.

In the first step, we partition the mesh of size $\frac{2X}{P} \times \frac{2Y}{Q}$ into 4 parts. The $\frac{XY}{PQ}$ closest points to the point $(1,1)$ are assigned to the part 1. The $\frac{XY}{PQ}$ closest points to the point $(2X/P, 2Y/Q)$ are assigned to the part 2. Those two parts are optimal, as they are triangle-like—see Fig. 4(a). Among the remaining points, the $\frac{XY}{PQ}$ closest points to $(1, 2Y/Q)$ are assigned to the part 3. The rest are assigned to the part 4. We perform these four assignments using a subroutine called *bfsColor*. The bfsColor routine accepts six arguments: the mesh, the indices of center point (c_x, c_y), the starting part number p, the target size of the part, and the number of parts, m. It then finds m times targetSize many points, closest to the center point (c_x, c_y) and colors them with a unique color starting from p. The algorithm can be implemented easily by a bfs search like an algorithm starting from (c_x, c_y) and stopping when $m \times targetSize$ many points are colored. Note that this is essentially a heuristic trying to create spherical structures as suggested by Lemmas 6 and 7. The resulting partition at the end of this step is shown in Fig. 6(a). In this figure, and in the algorithm, the part at the bottom left corner is 1, at the bottom right corner is 3, at the top right corner is 2, and the one at the top left corner is 4. This step runs in time $\mathcal{O}(\frac{XY}{PQ})$.

In the second step, we partition the mesh of size $\frac{2X}{P} \times \frac{3Y}{Q}$ into 6 parts. All points in parts 1 and 4 remain as they are computed in the first step. The algorithm then calls the subroutine $move$. The subroutine $move$ takes four arguments as input: the mesh, the axis along which to move the parts, the length of the movement, and the parts to move. It basically takes a number of parts and moves them by a certain amount. With the subroutine $move$, every point (i, j) such that $(i, j - \frac{Y}{Q})$ was assigned to the old part 2 (respectively 3) in the previous step is assigned to the new part 2 (respectively 3). Among the remaining points, the $\frac{XY}{PQ}$ ones closest to the point $(1,1)$ are assigned to the part 5. All the remaining points are then assigned to part 6. Now we partition the mesh of size $\frac{2X}{P} \times Y$. All parts but 2 and 3 are the same as in the previous partition. Every point (i, j) such that $(i, j - \frac{(Q-3)Y}{Q})$

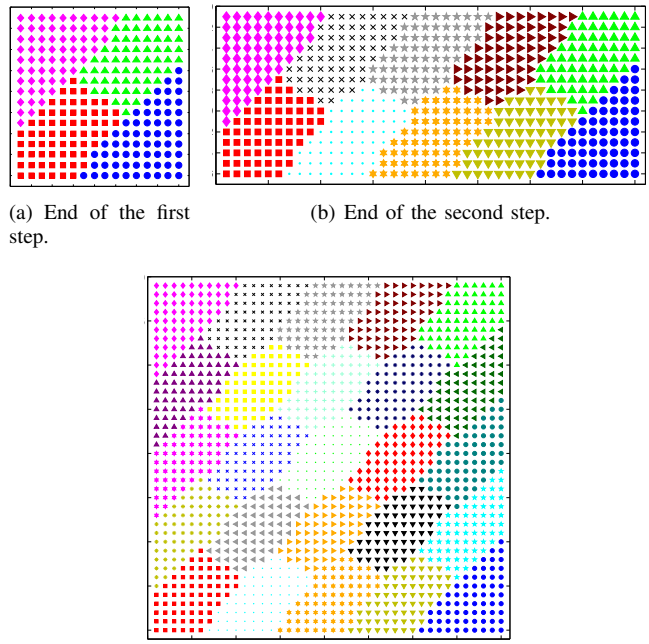
Input: X, Y; the size of the mesh
Input: K: the number of processors
Input: P, Q: processor mesh sizes, were $K = PQ$
Output: pmesh: pmesh(x, y) gives the processor assignment for the point (n_x, n_y)

```
    ►The first phase
targetsize ← XY/PQ
pmesh ← −1 for all (x, y) ∈ X × Y
bfscolor(pmesh, 1, 1, 1, targetsize, 1)
bfscolor(pmesh, 2X/P, 2Y/Q, 2, targetsize, 1)
bfscolor(pmesh, 1, 2Y/Q, 3, targetsize, 1)
bfscolor(pmesh, 2X/P, 1, 4, targetsize, 1)
if Q > 2 then    ►The second phase
    move(pmesh,'y', Y/Q, [2, 3])
    bfscolor(pmesh, 2X/P, 3Y/Q, 5, targetsize, 2)
    move(pmesh,'y', (Q−3)Y/Q, [2, 3])
    y ← zeros(2X/P)
    for x = 1 to 2X/P do
        y(x) ← 1
        while pmesh(x, y(x)) = 1 or pmesh(x, y(x)) = 4 do
            y(x) ← y(x) + 1
    for i = 4 to Q do
        for x = 1 to 2X/P do
            for j = 0 to Y/Q − 1 do
                pmesh(x, y(x) + (i − 3)Y/Q + j) ← pmesh(x, y(x) +
                j) + 2(i − 3)
if P > 2 then    ►The third phase
    L = {l | l mod 2 = 1 and l < 2Q}
    move(pmesh,'x', X/P, L)
    bfscolor(pmesh, 2X/P, Y, 2Q + 1, targetsize, Q)
    move(pmesh,'x', (P−3)X/P, L)
    x ← zeros(Y)
    for y = 1 to Y do
        x(y) ← 1
        while pmesh(x(y), y) < 2Q + 1 do
            x(y) ← x(y) + 1
    for i = 4 to P do
        for y = 1 to Y do
            for k = 0 to X/P − 1 do
                pmesh(x(y) + (i − 3)X/P + k, y) ← pmesh(x(y) +
                k, y) + (i − 3)Q
```

Fig. 5. The proposed MovePart algorithm to partition a five-point mesh into $P \times Q$ parts.

was assigned to the old part 2 (respectively 3) is assigned to the new part 2 (respectively 3). On each row, $\frac{Y}{Q}$ consecutive points belong to the parts 5 or 6. We create a vector y containing the coordinates of the leftmost of these points. Every point $(i, y(i) + k\frac{Y}{Q} + n)$, for $k = 1, \ldots Q - 3$ not yet colored is assigned to the part $pmesh(i, y(i) + n) + 2k$, with $n < \frac{Y}{Q}$. The resulting partition is show in Fig. 6(b). In this figure the five bottom parts are the parts $1, 5, 7, 9, 3$, and they will be moved in the next step. This step runs in time $\mathcal{O}(\frac{2X}{P}Y)$.

In the third step, we do the same operations as in the second step for the other dimension to partition the mesh of size $\frac{3X}{P} \times Y$. Let L be the set of odd integers smaller than $2Q$. All parts but the ones in L stay intact. Every point (i, j) such that $(i - \frac{Y}{Q}, j)$ is assigned to a part in L is assigned to that part. The $\frac{XY}{PQ}$ closest not-colored points to the point (X, Y) are assigned to the processor $2Q + 1$. We repeat this Q times such that we assign $\frac{XY}{PQ}$ points to each of $3Q$ parts. Now we partition the



(a) End of the first step.

(b) End of the second step.

(c) End of the third step.

Fig. 6. Partitioning a 40×40 mesh into 25 parts using the proposed algorithm.

mesh of size $\frac{2X}{P} \times Y$. All parts but the ones in L remain intact. Every point (i, j) such that $(i - \frac{(P-3)X}{P}, j)$ was assigned to a part in L is assigned to that part. On each column of the mesh, $\frac{X}{P}$ consecutive points belong to a part in L. We create a vector x containing the coordinates of the highest of those points. Every point $(x(j) + k\frac{X}{P} + m, j)$ for $k = 1, \ldots, P - 3$ not yet colored is assigned to the part $pmesh(x(j) + m, j) + Qk$,with $m < \frac{X}{P}$. Final result is show in Fig. 6(c). This step runs in time $\mathcal{O}(XY)$, which makes the algorithm running time be linearly proportional to the number of mesh points.

The proposed algorithm has the advantage of being not restricted to any mesh or processor counts. It only requires that X/P and Y/Q be integers. It works better for cases where X/P and Y/Q are close to each other, as in this case the shapes are close to being diamonds. These conditions are usually satisfied, as the given number of processors K can be factored in two integers P and Q so that X/P and Y/Q are both integers and are close to each other.

V. Experimental results

As we do not have a formula for the total communication volume resulting from the proposed algorithm, we did some experimental investigations. In the experiments, we compared the proposed algorithm with the hypergraph partitioning tool PaToH [10], used through its Matlab interface [23], the basic diamonds algorithm [3], and the MeshPart algorithm [22]. PaToH is run with the "quality" setting, with a very tight balance parameter. We call the proposed 2D mesh partitioning algorithm MovePart for lack of a better name.

We made three sets of experiments. In the first set, we have $\frac{X}{P} = \frac{Y}{Q}$ and XY is divisible by 32 in order to compare the proposed MovePart algorithm with the MeshPart [22]

TABLE I
COMPARISON OF DIFFERENT ALGORITHMS IN TERMS OF THE TOTAL COMMUNICATION VOLUME, WITH $P = Q = \sqrt{K}$.

Mesh size	K	MeshPart	Cartesian	PaToH	MovePart
64x64	4	226	256	252	**222**
128x128	4	450	512	504	**444**
128x128	64	3066	3584	3353	**3020**
256x256	4	898	1024	1015	**878**
256x256	64	5866	7168	6736	**5790**
256x256	256	13050	15360	13893	**12716**
512x512	4	1794	2048	2051	**1752**
512x512	64	11466	14336	13648	**11412**
512x512	256	24810	30720	28135	**24414**
512x512	1024	53754	63488	56306	**52076**
1024x1024	4	3586	4096	4194	**3500**
1024x1024	64	22666	28672	28279	**22574**
1024x1024	256	48330	61440	58598	**47988**
1024x1024	1024	101866	126976	114223	**100062**
2048x2048	4	7170	8192	8463	**6996**
2048x2048	64	45066	57344	56890	**44952**
2048x2048	256	95370	122880	117996	**94956**
2048x2048	1024	198090	253952	234477	**196404**
average		0.831	1.000	0.965	0.821

TABLE II
COMPARISON OF DIFFERENT ALGORITHMS IN TERMS OF THE TOTAL COMMUNICATION VOLUME. FOR THE NON-SQUARE PROCESSOR NUMBERS, THE FACTORIZATIONS $8 = 2 \times 4$, $32 = 4 \times 8$, $128 = 8 \times 16$, AND $512 = 16 \times 32$ ARE USED FOR THE VIRTUAL PROCESSOR MESH.

Mesh size	K	Cartesian	PaToH	Basic diamonds	MovePart
64x128	4	384	368	510	**324**
64x128	16	1152	1005	1044	**996**
64x128	64	2688	2166	**2152**	2460
256x512	4	1536	1424	2048	**1284**
256x512	16	4608	4217	4116	**3884**
256x512	64	10752	9062	**8296**	9180
256x512	256	23040	18322	**16848**	20156
1024x2048	4	6144	6225	8190	**5124**
1024x2048	16	18432	16525	16404	**15404**
1024x2048	64	43008	38978	**32872**	36060
1024x2048	256	92160	78710	**66000**	77756
1024x1024	8	8192	7719	8200	**7188**
1024x1024	32	20480	18967	**16432**	17516
1024x1024	128	45056	38816	**32992**	38364
1024x1024	512	94208	77167	**66496**	80828
average		1.00	0.894	0.929	0.855

TABLE III
COMPARISON OF THE ALGORITHMS. FOR THE CARTESIAN PARTITIONING CASE, $30 = 5 \times 6$, $120 = 10 \times 12$, AND $480 = 20 \times 24$ FACTORIZATIONS ARE USED FOR THE VIRTUAL PROCESSOR MESH.

Mesh size	K	Cartesian	PaToH	MovePart
200x300	30	4400	4091	**3626**
200x300	120	9800	8556	**8184**
400x600	30	8800	8412	**7172**
400x600	120	19600	17140	**15922**
400x600	480	41200	34567	**34144**
average		1.00	0.894	0.823

algorithm. The results are shown in Table I. In the second set of experiments, we have $\frac{XY}{PQ} = 2\rho^2$ where ρ is an integer in order to compare MovePart with the basic diamonds. Table II contains those results. In the third set of experiments, we have the only requirement that $\frac{X}{P}$ and $\frac{Y}{Q}$ are integers so that MeshPart and basic diamonds are not applicable, but hypergraph and Cartesian partitioning methods are. Table III contains the comparison of the proposed method with the hypergraph partitioning tool PaToH and the Cartesian partitioning method. In these three tables the best result is in bold face for each row, the results are normalized with respect to the Cartesian partitioning and the average ratios are given at the last row.

As Table I shows, the proposed MovePart algorithm is always better than the Cartesian and hypergraph partitioning methods. It is always, with the exception of the second problem instance, better than the MeshPart algorithm. We see that the hypergraph partitioning method gives better results than the Cartesian partitioning method, but gains only 3% whereas the proposed MovePart gains 18% on the average.

As Table II shows, the basic diamonds are better than the proposed MovePart algorithm for large (here larger than 16 is sufficient) values of PQ. Yet, for small numbers the proposed algorithm obtains better results. We note that this fact does not depend on the mesh size, i.e., there are infinitely many cases in which the proposed MovePart algorithm is better than the basic diamonds (and there are infinitely many cases where the converse is also true). The hypergraph partitioning method is sometimes better than MovePart which loses its effectiveness when $\frac{X}{P}$ and $\frac{Y}{Q}$ are very different. Nevertheless, MovePart still gains 4% in average over the hypergraph partitioning method, while the basic diamond is still pretty bad in average because of its poor results on small values of PQ.

Table III gives the results of experiments where we compare the proposed MovePart algorithm with the hypergraph partitioning tool PaToH. In general, even when $\frac{X}{P}$ and $\frac{Y}{Q}$ are not equal (but have a ratio smaller than two between them) our algorithm has better results than PaToH. As said before, the running time of the proposed algorithm is linear. Given that PaToH is more time consuming, we think that the proposed MovePart method can be preferable (the difference between the total communication volume is small).

The maximum volume of messages of a processor, the maximum number of messages of a processor, and the total number of messages are also important communication cost metrics in distributed memory parallel computing systems. However, due to the nature of the problem and the partitioning methods, among those metrics only the maximum volume per processor is exciting (the others are always close to each other). We give in Table IV the maximum volume of sends/receives of a processor with different partitioning algorithms. We do not comment much on these numbers, we rather give them for reference after noting that they follow the trends in the total volume of communication metric. It is important to note however that the basic diamonds balance all these communication cost metrics across processors (whenever it works), as it creates parts with the same shape.

VI. CONCLUSION

We investigated the partitioning of 2D meshes arising in 5-point stencil-based computations for distributed-memory parallel computers. The partitioning objectives were to balance the number of mesh points across processors and minimize the total communication volume. We collected some theoretical results, and provided some more so as to understand the difficulty of the underlying problem and gain insights. Based

TABLE IV
MAXIMUM COMMUNICATION VOLUME HANDLED BY A SINGLE PROCESSOR IN RECEIVES OR IN SENDS.

Mesh size	K	PaToH	MovePart
128x128	4	161	130
128x128	64	65	52
200x300	30	182	144
200x300	120	96	74
256x256	4	264	257
256x256	64	130	100
256x256	256	74	52
400x600	120	183	144
400x600	480	108	74
512x512	4	584	513
512x512	64	282	196
512x512	256	145	100
512x512	1024	66	52
1024x1024	4	1227	1025
1024x1024	64	538	388
1024x1024	256	276	196
1024x1024	1024	150	100
2048x2048	4	2254	2049
2048x2048	64	1199	772
2048x2048	256	625	388
2048x2048	1024	339	196

Mesh size	K	PaToH	Basic diamonds	MovePart
64x128	4	111	128	98
64x128	16	85	66	84
64x128	64	42	34	44
256x512	4	421	512	386
256x512	16	399	258	324
256x512	64	199	130	164
256x512	256	94	66	84
1024x2048	4	1673	2048	1538
1024x2048	16	1482	1026	1284
1024x2048	64	803	514	644
1024x2048	256	412	258	324
1024x1024	8	1414	1026	1156
1024x1024	32	850	514	644
1024x1024	128	394	258	324
1024x1024	512	215	130	164

on theoretical observations, we proposed a linear-time geometric heuristic, called MovePart method. Geometric approaches alternative to MovePart include diamond-like partitioning methods, called basic diamonds [5, Section 4.8] and [3], and MeshPart [22]. There are restrictions on when diamonds and MeshPart can be applied. The proposed algorithm MovePart removes most of those restrictions and hence can be applied to a wide range of problems. It was demonstrated to be more effective than the hypergraph partitioning by about 10% and to obtain results that are on the average around a factor 0.82 of the well-known Cartesian partitioning approach.

We discussed mainly distributed-memory parallel computing environments. The main application of the fast partitioning heuristics similar to the proposed MovePart algorithm would target hierarchical parallel computers in which each processing element is a many- or multi-core node. In such systems, the partitioning algorithms will minimize the inter-node communication. They should be combined with the state-of-the-art performance increasing methods (e.g., [11], [12], [17]). Since the proposed algorithm yields connected mesh parts (like many of the alternatives), it should facilitate the application of mentioned methods.

REFERENCES

[1] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, Eds., *Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop, February 13-14, 2012. Georgia Institute of Technology, Atlanta, GA*. AMS, 2013, vol. 588.

[2] S. L. Bezrukov and B. Rovan, "On partitioning grids into equal parts," *Computers and Artificial Intelligence*, vol. 16, no. 2, 1997.

[3] R. H. Bisseling and W. F. McColl, "Scientific computing on bulk synchronous parallel architectures," in *Technology and Foundations: Information Processing '94, Vol. I*, ser. IFIP Transactions A, B. Pehrson and I. Simon, Eds., vol. 51. Elsevier, Amsterdam, 1994, pp. 509–514.

[4] R. H. Bisseling and W. Meesen, "Communication balancing in parallel sparse matrix-vector multiplication," *ETNA*, vol. 21, pp. 47–65, 2005.

[5] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*. Oxford University Press, 2004.

[6] B. Bollobás and I. Leader, "Edge-isoperimetric inequalities in the grid," *Combinatorica*, vol. 11, no. 4, pp. 299–314, 1991.

[7] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, U. Catalyurek, D. Bozdag, W. Mitchell, and J. Teresco, *Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User's Guide*, Sandia National Lab., NM, 2007.

[8] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is NP-hard," *Inf. Proc. Lett.*, vol. 42, pp. 153–159, 1992.

[9] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE T. Par. and Dist. Sys.*, vol. 10, no. 7, pp. 673–693, 1999.

[10] ——, "PaToH: A multilevel hypergraph partitioning tool, version 3.0," Dept. Computer Eng., Bilkent Univ., Tech. Rep. BU-CE-9915, 1999.

[11] M. Christen, O. Schenk, and H. Burkhart, "PATUS: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures," in *IPDPS 2011*, Anchorage, Alaska, USA, May 2011, pp. 676–687.

[12] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, and K. A. Yelick, "Optimization and performance modeling of stencil computations on modern microprocessors," *SIAM Review*, vol. 51, pp. 129–159, 2009.

[13] A. E. Feldmann, "Fast balanced partitioning is hard even on grids and trees," *Theoretical Computer Science*, vol. 485, pp. 61–68, 2013.

[14] J. R. Gilbert, G. L. Miller, and S.-H. Teng, "Geometric mesh partitioning: Implementation and experiments," *SIAM J. Sci. Comput.*, vol. 19, no. 6, pp. 2091–2110, 1998.

[15] B. Hendrickson, "Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract)," in *Proc. of the 5th Int. Symp. on Solving Irregularly Structured Problems in Parallel*, ser. IRREGULAR '98. London, UK, UK: Springer-Verlag, 1998, pp. 218–225.

[16] ——, "Load balancing fictions, falsehoods and fallacies," *Applied Mathematical Modelling*, vol. 25, pp. 99–108, 2000.

[17] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams, "An auto-tuning framework for parallel multicore stencil computations," in *IPDPS 2010*, Atlanta, Georgia, USA, April 2010, pp. 1–12.

[18] G. Karypis and V. Kumar, *MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0*, Univ. Minnesota, Dept. Comp. Sci. Eng., Minneapolis, 1998.

[19] G. Karypis, V. Kumar, R. Aggarwal, and S. Shekhar, *hMeTiS: A Hypergraph Partitioning Package Version 1.0.1*, Univ. Minnesota, Dept. Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[20] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. Chichester, U.K.: Wiley–Teubner, 1990.

[21] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. Sci. Comput.*, vol. 25, pp. 1837–1859, 2004.

[22] B. Uçar and Ü. V. Çatalyürek, "On the scalability of hypergraph models for sparse matrix partitioning," in *PDP 2010*, M. Danelutto, J. Bourgeois, and T. Gross, Eds. IEEE Computer Society, 2010, pp. 593–600.

[23] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat, "A matrix partitioning interface to PaToH in MATLAB," *Parallel Computing*, vol. 36, no. 5-6, pp. 254–272, 2010.

[24] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Review*, vol. 47, no. 1, pp. 67–95, 2005.

[25] D.-L. Wang and P. Wang, "Discrete isoperimetric problems," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 860–870, 1977.