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 $[\cdot]$ rounds $\cdot$ 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 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 more load than average, one can use $\lceil K/(1+\varepsilon) \rceil$ 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 $(p_x,p_y)$. Fig. 1. A 2D mesh resulting from the use of five-point stencil.
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

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

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 known to be $NP$-hard for general graphs [8].

D. Hypergraph partitioning

A hypergraph $H = (V, \mathcal{N})$ consists of a set of vertices $V$ and a set of nets (hyperedges) $\mathcal{N}$. Every net is a subset of vertices. Given a hypergraph $H = (V, \mathcal{N})$, $\Pi = \{V_1, \ldots, V_K\}$ is called a $K$-way partition of the vertex set $V$ if each part is non-empty, parts are pairwise disjoint, and the union of parts gives $V$. The partitioning constraint is to maintain a balance on the number of vertices per part. In a partition $\Pi$ of $H$, a net that contains at least one vertex in a part is said to connect that part. Connectivity $\lambda_j$ 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

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

This problem is known to be $NP$-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 $\ell_1$-sphere defined by a center point $(x_c, y_c)$ and a radius $\rho$. Such a diamond contains all mesh points with a Manhattan distance at most $\rho$ of the center, i.e., a point $(x, y)$ is included in a diamond having the center $(x_c, y_c)$ and the radius $\rho$ if $|x - x_c| + |y - y_c| \leq \rho$. A digital diamond of radius $\rho$ 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 $\rho$). Figure 2(a) shows the partitioning of the $13 \times 13$ mesh into 13 parts, each represented by a digital diamond of radius 2.

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 $\frac{X}{\rho}$ and $\frac{Y}{\rho}$ 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 $\rho$. This gives us $\beta = -\frac{\rho + 1}{\rho}\alpha$ and $\delta = -\frac{\rho + 1}{\rho}\gamma$. Then $\alpha = \frac{\rho K}{X}$ and $\gamma = \frac{\rho K}{Y}$.

Next we give the total volume of communication formula for 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 $(4\rho + 4)K - 2\left[\frac{X}{(2\rho + 1)(\rho + 1)}\right].$

**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 $\rho$ from the border of the mesh. First, let us count the centers at distance $\rho$ 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 $1 + \left[\frac{X}{(2\rho + 1)(\rho + 1)}\right]$ centers at distance $\rho$ from the left border.
Basic diamonds works only if assuming problems that can be solved with basic diamonds, again parts, each being a basic diamond of radius $2$.

Lemma 4. The volume of a basic diamond is $\frac{\alpha \rho^2}{2}$, 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 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. There are $\frac{Y}{2\rho}$ centers at the left and right borders save one volume of communication, resulting in a total saving of $Y/\rho$. 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 $\frac{X \times Y}{P \times Q}$ 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 $16 \times 16$ which is then extended to a 6-way partitioning of a mesh of size $16 \times 24$ 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 $NP$-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 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 $\text{edgecut}(L) > 2ab$. This system has a unique minimum for the edge cut when $a = b = S$. The optimal edge cut is therefore $4\sqrt{\frac{S^2}{PQ}}PQ - 2X - 2Y$, 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 $d$th 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...
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 \( \lambda \) 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 \( X/P \) and \( Y/Q \) are integers. The proposed algorithm (Fig. 5) runs in three steps.

In the first step, we partition the mesh of size \( X \times Y \) into \( 4 \times 4 \) parts. The \( XY \) closest points to the point \((1, 1)\) are assigned to the part 1. The \( XY \) 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 \( XY \) 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 \( \text{bfsColor} \). The \( \text{bfsColor} \) routine accepts six arguments: the mesh, the indices \( x, y \), the starting part number \( p \), the target size of the part, and the number of parts, \( m \). It then finds \( m \) times \( \text{targetSize} \) 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 \( \text{bfs} \) search like an algorithm starting from \((c_x, c_y)\) and stopping when \( m \times \text{targetSize} \) 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 the second step, we partition the mesh of size \( X \times Y \) 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 \( \text{move} \). The subroutine \( \text{move} \) 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 \( \text{move} \), every point \((i, j)\) such that \((i - X/P) \) 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 \( XY \) 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 \( 2X/P \times 3Y/Q \). All parts but 2 and 3 are the same as in the previous partition. Every point \((i, j)\) such that \((i - (Q-3)Y/Q) \)
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 $(n_x, n_y)$

1. **The first phase**
   1. targetsize ← $\frac{XY}{K}$
   2. pmesh ← $-1$ for all $(x,y) \in X \times Y$
   3. bfsColor(pmesh, 1, 1, 1, targetsize, 1)
   4. bfsColor(pmesh, $\frac{X}{P}$, $\frac{Y}{Q}$, 2, targetsize, 1)
   5. bfsColor(pmesh, 1, $\frac{2Y}{Q}$, 3, targetsize, 1)
   6. bfsColor(pmesh, $\frac{2X}{P}$, 1, 4, targetsize, 1)

2. **The second phase**
   7. if $Q > 2$ then **The second phase**
   8. move(pmesh, $y'$, $\frac{Y}{Q-2}$, $\frac{Y}{Q}$, 2)
   9. bfsColor(pmesh, $\frac{2X}{P}$, $\frac{Y}{Q-2}$, 5, targetsize, 2)
   10. move(pmesh, $y'$, $(2Q-3)\frac{Y}{Q}$, 2, 3)

   11. $y \leftarrow$ zeros($\frac{P}{Q}$)
   12. for $x = 1$ to $\frac{P}{Q}$ do
   13. \quad $y(x) \leftarrow 1$
   14. while pmesh($x,y(x)$) = 1 or pmesh($x,y(x) - 1$) = 4 do
   15. \quad $y(x) \leftarrow y(x) + 1$
   16. for $i = 4$ to $Q$ do
   17. \quad for $x = 1$ to $\frac{P}{Q}$ do
   18. \quad \quad if $j = 0$ to $\frac{Y}{Q} - 1$ do
   19. \quad \quad \quad pmesh($x,y(x),(i-(3)\frac{Y}{Q}+j)$) ← pmesh($x,y(x)+j$) + 2($i-3$)
   20. \quad \quad \quad \end{for}
   21. \quad \quad \quad $y(x) \leftarrow y(x) + 1$
   22. \quad \quad for $j = 0$ to $\frac{Y}{Q}$ - 1 do
   23. \quad \quad \quad pmesh($x,y(x),(i-(3)\frac{Y}{Q}+j)$) ← pmesh($x,y(x)+j$) + 2($i-3$)
   24. \quad \quad \quad \end{for}
   25. \quad \quad PMesh(x) \leftarrow zeros(Y)
   26. \quad for $y = 1$ to $Y$ do
   27. \quad \quad $x(y) \leftarrow 1$
   28. \quad \quad while pmesh($x(y), y$) < $2Q + 1$ do
   29. \quad \quad \quad $x(y) \leftarrow x(y) + 1$
   30. \quad \quad \quad \end{while}
   31. \quad \quad \quad for $i = 4$ to $P$ do
   32. \quad \quad \quad \end{for}
   33. \quad \quad \quad for $k = 0$ to $\frac{Y}{Q} - 1$ do
   34. \quad \quad \quad \quad pmesh($x(y) + (i-(3)\frac{Y}{Q} + k,y)$) ← 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 processor $pmesh(i, y(i) + n) + 2k$, 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 $O(\frac{XY}{Q})$.

In the third step, we do the same operations as in the second step for the other dimension to partition the mesh of size $\frac{2X}{P} \times Y$. 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 $(i-(3)\frac{Y}{Q}, j)$ 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 mesh of size $\frac{2X}{P} \times Y$. All parts but the ones in $L$ remain intact. Every point $(i, j)$ such that $(i-(3)\frac{Y}{Q}, j)$ 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 $O(\frac{XY}{Q})$, 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]
algorithm. The results are shown in Table I. In the second set of experiments, we have \(\frac{X}{P+Q} = 2\rho^2\) where \(\rho\) 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 Mesh-Part 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
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.

**TABLE IV**

<table>
<thead>
<tr>
<th>Mesh size</th>
<th>( K )</th>
<th>PaToH Basic diamonds</th>
<th>MovePart</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128</td>
<td>4</td>
<td>161</td>
<td>130</td>
</tr>
<tr>
<td>256x256</td>
<td>4</td>
<td>264</td>
<td>257</td>
</tr>
<tr>
<td>512x512</td>
<td>4</td>
<td>584</td>
<td>513</td>
</tr>
<tr>
<td>1024x1024</td>
<td>4</td>
<td>1227</td>
<td>1025</td>
</tr>
<tr>
<td>2048x2048</td>
<td>4</td>
<td>2254</td>
<td>2049</td>
</tr>
<tr>
<td>2048x2048</td>
<td>64</td>
<td>1199</td>
<td>772</td>
</tr>
<tr>
<td>2048x2048</td>
<td>256</td>
<td>625</td>
<td>388</td>
</tr>
<tr>
<td>2048x2048</td>
<td>1024</td>
<td>329</td>
<td>196</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mesh size</th>
<th>( K )</th>
<th>PaToH Basic diamonds</th>
<th>MovePart</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x128</td>
<td>4</td>
<td>111</td>
<td>125</td>
</tr>
<tr>
<td>64x128</td>
<td>16</td>
<td>85</td>
<td>66</td>
</tr>
<tr>
<td>64x128</td>
<td>64</td>
<td>42</td>
<td>34</td>
</tr>
<tr>
<td>256x512</td>
<td>4</td>
<td>421</td>
<td>512</td>
</tr>
<tr>
<td>256x512</td>
<td>16</td>
<td>399</td>
<td>258</td>
</tr>
<tr>
<td>256x512</td>
<td>64</td>
<td>199</td>
<td>130</td>
</tr>
<tr>
<td>256x512</td>
<td>256</td>
<td>94</td>
<td>66</td>
</tr>
<tr>
<td>1024x2048</td>
<td>4</td>
<td>1673</td>
<td>2048</td>
</tr>
<tr>
<td>1024x2048</td>
<td>16</td>
<td>1482</td>
<td>1026</td>
</tr>
<tr>
<td>1024x2048</td>
<td>64</td>
<td>803</td>
<td>514</td>
</tr>
<tr>
<td>1024x2048</td>
<td>256</td>
<td>412</td>
<td>258</td>
</tr>
<tr>
<td>1024x1024</td>
<td>8</td>
<td>1414</td>
<td>1026</td>
</tr>
<tr>
<td>1024x1024</td>
<td>32</td>
<td>850</td>
<td>514</td>
</tr>
<tr>
<td>1024x1024</td>
<td>128</td>
<td>394</td>
<td>258</td>
</tr>
<tr>
<td>1024x1024</td>
<td>512</td>
<td>215</td>
<td>130</td>
</tr>
</tbody>
</table>

**REFERENCES**


