A medium-grain method for fast 2D bipartitioning of sparse matrices

Daniël M. Pelt
Scientific Computing Group
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
D.M.Pelt@cwi.nl

Rob H. Bisseling
Mathematical Institute
Utrecht University
Utrecht, The Netherlands
R.H.Bisseling@uu.nl

***Abstract*—We present a new hypergraph-based method, the medium-grain method, for solving the sparse matrix partitioning problem. This problem arises when distributing data for parallel sparse matrix-vector multiplication. In the medium-grain method, each matrix nonzero is assigned to either a row group or a column group, and these groups are represented by vertices of the hypergraph. For an $m \times n$ sparse matrix, the resulting hypergraph has $m+n$ vertices and $m+n$ hyperedges.**

Furthermore, we present an iterative refinement procedure for improvement of a given partitioning, based on the medium-grain method, which can be applied as a cheap but effective postprocessing step after any partitioning method.

The medium-grain method is able to produce fully two-dimensional bipartitionings, but its computational complexity equals that of one-dimensional methods. Experimental results for a large set of sparse test matrices show that the medium-grain method with iterative refinement produces bipartitionings with lower communication volume compared to current state-of-the-art methods, and is faster at producing them.

***Keywords*-parallel computing, sparse matrix-vector multiplication, hypergraph**

I. Introduction

The *sparse matrix partitioning problem* naturally arises in the study of parallel sparse matrix-vector multiplication, where an $m \times n$ matrix A with N nonzeros ($N \ll mn$) is multiplied by a vector $\vec{v}$ of length n. The standard method of parallelizing this multiplication (see e.g. [1]) starts by distributing the N nonzero elements over the p available processors, creating a subset of nonzeros A_i for each processor, such that $A = \bigcup_i A_i$ and $A_i \cap A_j = \emptyset$ if $i \neq j$. Then, the product $\vec{u} = A\vec{v}$ is calculated in four steps: (1) the fanout; (2) the local multiplication; (3) the fanin; and finally (4) the summation of partial sums. During the fanout, communication is necessary if an input vector element v_j that needs to be multiplied with an element a_{ij} of the local subset is owned by a different processor. After the fanout, all elements of the local subset are multiplied by the corresponding input vector element and partial sums are created for each output vector element u_i. This operation is performed locally by all processors, and no communication is therefore needed. Following this local multiplication step, all nonzero partial sums need to be communicated to the processor that owns the corresponding output vector element. This step is called the fanin. Finally, each processor sums up the partial sums for each output vector element that it owns, to produce the output vector $\vec{u}$.

The time it takes to compute $\vec{u}$ depends mainly on two factors: the number of multiplications that need to be performed in step (2) and the number of elements that need to be communicated in steps (1) and (3). In step (2), every element of the local subset has to be multiplied and each processor can work in parallel, so the total time requirement of this local multiplication step is $\mathcal{O}(\max_i |A_i|)$. In order to minimize this computation time, $\max_i |A_i|$ is usually bound by a load imbalance constraint,

$$\max_i |A_i| \leq (1+\varepsilon)\frac{N}{p}, \tag{1}$$

where ε is the allowed load imbalance fraction.

In steps (1) and (3), the communication has to be minimized. Communication along a single column during the fanout step is only needed when the matrix elements of that column are assigned to different processors, and similarly for rows during the fanin step. In fact, the necessary communication cost of a single row or column i equals

$$C(i, A_0, \ldots, A_{p-1}) = \lambda_i - 1, \tag{2}$$

where λ_i is the number of processors that have one or more nonzeros of that row/column in their local subset. The total amount of communication, measured in data words, that is needed during the parallel sparse matrix-vector multiplication, is called the *communication volume* V and it is equal to the sum of all row and column communication costs:

$$V(A_0, \ldots, A_{p-1}) = \sum_i C(i, A_0, \ldots, A_{p-1}). \tag{3}$$

Given an $m \times n$ matrix A with N nonzeros and a number of processors p, the sparse matrix partitioning problem can be defined as follows: out of all partitionings of matrix A into p subsets A_i that obey the load imbalance constraint, eqn (1), find the one with the smallest communication volume V, eqn (3). Note that other communication metrics such as maximum communication volume per processor or the total number of messages sent may also be important, but this falls outside the scope of the current work, as we will

be concerned exclusively with minimizing the total communication volume. A recent package that takes other metrics into account during the matrix partitioning is UMPa [2].

II. Previous work

Most attempts to solve the sparse matrix partitioning problem in practice first translate the problem to a *hypergraph partitioning problem*, and then partition the resulting hypergraph. A hypergraph is a generalization of a graph, in which edges can connect more than two vertices with each other. Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of n vertices $\mathcal{V} = \{0, \ldots, n-1\}$ and a set of m *hyperedges* $\mathcal{N} = \{n_0, \ldots, n_{m-1}\}$. Each hyperedge, also called a *net*, is a subset of $\mathcal{V}$. The advantage of using hypergraphs is that we can employ existing hypergraph partitioners to solve the sparse matrix partitioning problem. Since hypergraph partitioning is NP-Complete [3], these solvers all use heuristics to try to find a good solution: no algorithm exists that can find the optimal solution for large problems in reasonable time.

Three different hypergraph models are commonly used for the translation of sparse matrices: the one-dimensional (1D) *row-net* and *column-net* models [4] and the two-dimensional (2D) *fine-grain* model [5], all by Çatalyürek and Aykanat. In the row-net model, each column of the matrix is converted to a vertex of the hypergraph, and each row is converted to a hyperedge. All columns that have a nonzero in a certain row are placed in the hyperedge corresponding to that row. The column-net model is the same, but with the roles of rows and columns reversed. The advantage of using the row-net or column-net model is that one source of communication during parallel sparse matrix-vector multiplication is completely eliminated. For instance, using the row-net model, no communication is necessary during the fanout phase, step (1). Of course, matrix rows can still be cut, so communication during fanin might still occur. The disadvantage of the row-net and column-net models is that they can be too restrictive: for certain matrix-vector multiplications, allowing some communication during the fanout in the row-net model can help reduce the overall needed communication volume.

In the fine-grain model, the matrix nonzeros correspond to the vertices of the hypergraph, and the rows and columns of the matrix are converted to hyperedges. This model is very general: every partitioning of a matrix A can be expressed as a partitioning of the corresponding fine-grain hypergraph, and vice versa. The fine-grain model does not suffer from the restrictiveness of the row-net and column-net models, but has the main disadvantage that it is larger: it contains N vertices, whereas the row-net and column-net models contain n and m vertices, respectively. Therefore, computations on the fine-grain model of a matrix take longer than computations on the row-net and column-net models of the same matrix, and furthermore the larger size can make it harder to find good solutions to the fine-grain hypergraph partitioning problem.

The three hypergraph models for sparse matrices can also be interpreted as three different ways of translating the matrix A to another matrix B, followed by application of the canonical row-net model for translating a sparse matrix to a hypergraph. The row-net model then has $B = A$, the column-net model $B = A^T$, the transposed of A, and the fine-grain model has $B = F(A)$, the $(m+n) \times N$ fine-grain matrix with two nonzeros f_{ik} and $f_{m+j,k}$ corresponding to the kth nonzero a_{ij} in A, with $0 \leq k < N$. The incidence matrix $F(A)$ is illustrated in [6, Fig. 12.5].

Uçar and Aykanat [7] present enhanced hypergraph models for parallel sparse matrix–vector multiplication that include the matrix and both the input and output vectors, enhancing the row-net, column-net, and fine-grain models by adding extra vertices for the vector components. This allows for minimizing the communication volume in case the input and output vectors have to be distributed in the same way, which may cause extra communication for matrices with zeros on the main diagonal. In other work [8], the same authors introduce a method for simultaneously partitioning two matrices A and M aimed at preconditioned iterative methods for linear system solving. Here, A is the matrix from the original problem $A\vec{x} = \vec{b}$ to be solved and M is the preconditioner, an approximation of A^{-1}; the corresponding preconditioned system is $AM\vec{z} = \vec{b}$. Both matrices are used in the solver in a multiplicative manner, by first computing $\vec{x} = M\vec{z}$ and then $\vec{y} = A\vec{x}$, yielding $\vec{y} = AM\vec{z}$. The simultaneous method computes a 1D partitioning of both matrices, where the work load (expressed as the number of nonzeros) can be balanced either for both multiplications together, or separately, and where the partitioning of the rows of M is the same as that of the columns of A (and that of the components of $\vec{x}$). The simultaneous partitioning problem is solved by formulating it in terms of a *composite hypergraph*, composed from simpler hypergraphs (an enhanced column-net model for M and an enhanced row-net model for A), which are connected by suitably merging vertices corresponding to vector components.

Çatalyürek and Aykanat [9] introduce the *coarse-grain* method, which in a first phase applies the column-net hypergraph model to obtain a row partitioning of the matrix into p parts and then performs a multi-constraint column partitioning into q parts, yielding a 2D Cartesian partitioning into $p \times q$ parts.

It is difficult to choose the best method to use, given a matrix to partition. Çatalyürek, Aykanat, and Uçar [10] propose a partitioning recipe that chooses a partitioning method according to some matrix characteristics.

We call the new hypergraph method that we present in this paper the *medium-grain* method, as it fits between the fine-grain and the coarse-grain approach. The fine-grain, medium-grain, and coarse-grain approaches are all 2D in

nature. The fine-grain approach treats nonzeros individually, the coarse-grain approach treats whole rows or columns, whereas the medium-grain approach treats groups of nonzeros from the same row or column as an atomic entity.

PaToH [4], hMetis [11], and Mondriaan [12] are some of the most popular sequential hypergraph partitioners available at this moment. A parallel hypergraph partitioner is Zoltan-PHG [13]. MLpart [14] is a multilevel hypergraph partitioner developed for circuit design applications. Of these, Mondriaan is specifically designed to solve the matrix partitioning problem, while the others are general hypergraph partitioners that can also be used to partition matrices. While these partitioners are different in solution quality and execution time, all use a similar method to find good initial solutions, namely the Kernighan–Lin method [15] with the optimizations of Fiduccia–Mattheyses [16]. In order to solve large instances, all these partitioners use a multilevel method [17]: large instances are coarsened to smaller ones, the resulting smaller instances are partitioned, and the solution is refined to produce a solution for the original large instance. Furthermore, these partitioners are all based on recursive bisection: instead of being partitioned into k parts directly, the hypergraph is recursively split into two parts until k parts are obtained. The Mondriaan matrix partitioner (version 1.0 until 3.11) employs as default the *localbest* method, which splits the matrix using both the row-net and column-net model, and returns the bipartitioning with the lowest communication volume of the two. The medium-grain method has become the default in version 4.0 of Mondriaan. An overview of hypergraph-based sparse matrix partitioning methods and a comparison of 1D and 2D approaches can be found in [6].

III. Method

A. Composite model

In this paper, we propose a new method, based on a composite hypergraph model, for solving the matrix partitioning problem: the *medium-grain* method. This method combines the advantages of the existing row-net, column-net, and fine-grain hypergraph models, while avoiding their disadvantages. The method is able to produce 2D partitionings directly, thereby avoiding the main disadvantage of the row-net and column-net model, but it still imposes sets of nonzeros from the same row or column to be assigned to a single processor, avoiding the main disadvantage of the fine-grain model, where each nonzero is assigned individually.

To translate an $m \times n$ matrix A to the new, composite hypergraph model, we first split it into two parts: A^r and A^c, each of size $m \times n$. Each nonzero a_{ij} of A is placed in either A^r or A^c, in row i and column j. In other words, we ensure that $A = A^r + A^c$. Afterwards, A^r and A^c are

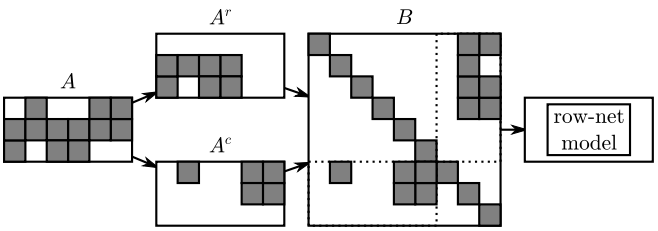


Figure 1: Translation of a 3×6 matrix A with $N = 12$ nonzeros to a new hypergraph. First, we split A into two disjoint parts A^r and A^c. We combine A^r and A^c to form the matrix B, with ones on the main diagonal. Finally, we form the hypergraph by applying the row-net model to B.

combined to form a new $(m+n) \times (m+n)$ matrix B:

$$B = \begin{bmatrix} I_n & (A^r)^T \\ A^c & I_m \end{bmatrix}, \tag{4}$$

where I_m is the identity matrix of size $m \times m$.

The diagonal of B is only used for calculating the communication volume, not for calculating the load imbalance. This is done by considering the diagonal nonzeros of B as dummy nonzeros. If a subset of the nonzeros in a single row of A is assigned to A^r, and the remaining subset of the row to A^c, the block I_m of B connects both subsets with each other, such that the corresponding communication volume is still correct if they are assigned to different processors. The same holds for columns of A and block I_n of B.

The composite hypergraph $\mathcal{H}_B = (\mathcal{V}_B, \mathcal{N}_B)$ is obtained by applying the row-net model to the matrix B. In the row-net model, each column of B becomes a vertex of the hypergraph: $\mathcal{V}_B = \{0, \ldots, m+n-1\}$. The vertex weight of the vertex corresponding to column j of B is equal to $nz_c(j) - 1$, where $nz_c(j)$ is the number of nonzero elements in column j, and one is subtracted to exclude the dummy diagonal element. Each row i of B is converted to a hyperedge n_i, containing all vertices j for which $b_{ij} \neq 0$. In other words, $\mathcal{N}_B = \{n_0, \ldots, n_{m+n-1}\}$ and $n_i = \{j : b_{ij} \neq 0\}$. The entire process of translating A to a hypergraph in the new method is illustrated in Fig. 1.

A column partitioning of B can easily be converted to a partitioning of A, as follows. After the hypergraph partitioning, each column k of B is assigned to a single processor p_k. Because of the way B is constructed, we see that each column j of A^c is assigned to p_j and each row i of A^r to p_{i+n}. The corresponding partitioning of A is created by setting the processor to which the nonzero a_{ij} is assigned to p_j if a_{ij} was put in A^c and to p_{i+n} if it was put in A^r. This process is illustrated in Fig. 2. The final partitioning of A is two-dimensional, but it keeps sets of nonzeros from the same row or column together, since all nonzeros in a single column of A^c are assigned to the same processor, and similarly for the nonzeros of a single row of A^r.

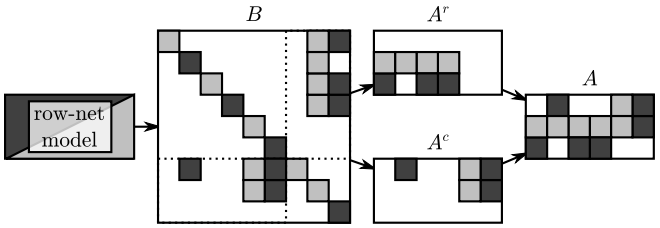


Figure 2: Translation of a partitioning of the row-net hypergraph to a partitioning of A. Each nonzero is given a shade of gray corresponding to the part to which it is assigned. The number of parts is $p = 2$. The row-net partitioning is converted into a partitioning of B, where every column is assigned to a single processor. This partitioning is copied to A^c and A^r, resulting in a partitioning where every column of A^c is assigned to a single processor, and every row of A^r. By combining both, we obtain a 2D partitioning of A.

We will now show that the communication volume of the resulting partitioning of A is equal to that of the original partitioning of B, provided the columns of B containing only the diagonal nonzero are assigned to a part owning one of its neighboring columns (in order to avoid unnecessary communication). Here, a column j' is a neighbor of column j if it contains a nonzero $a_{ij'}$ such that a_{ij} is also nonzero.

Let $A = A^r + A^c$ be an $m \times n$ sparse matrix, where A^r and A^c are $m \times n$ sparse matrices with disjoint sets of nonzeros, and let B be the $(m + n) \times (m + n)$ sparse matrix defined by eqn (4). Let the columns of B, except those containing only a diagonal nonzero, be partitioned into p sets. Every remaining column is assigned to a part owning one of its neighboring columns, if such a column exists, and to an arbitrary part otherwise. This defines a partitioning of the nonzeros of B into p disjoint sets $\{B_0, \ldots, B_{p-1}\}$, with $B = \cup_{k=0}^{p-1} B_k$.

Define p sets of nonzeros of A by

$$\begin{aligned} A_k = &\{a_{ij} \in A^r : b_{j,i+n} \in B_k\} \cup \\ &\{a_{ij} \in A^c : b_{i+n,j} \in B_k\}, \end{aligned} \tag{5}$$

for all $k \in \{0, \ldots, p-1\}$. Then, the sets $\{A_0, \ldots, A_{p-1}\}$ form a partitioning of A, as follows. The sets A_k are disjoint, because the sets B_k are disjoint and A^r and A^c are disjoint. If a nonzero a_{ij} is in A^r, it has a corresponding nonzero $b_{j,i+n}$ in B and hence in a B_k, so that a_{ij} is in A_k. If it is in A^c, it has a corresponding nonzero $b_{i+n,j}$ in B and hence in a B_k, so that a_{ij} is in A_k. Thus, every nonzero is contained in a subset A_k, and therefore the set $\{A_0, \ldots, A_{p-1}\}$ is a partitioning of A.

Furthermore, the communication volume for a parallel sparse matrix-vector multiplication by A using $\{A_0, \ldots, A_{p-1}\}$ is equal to that of a multiplication by B using $\{B_0, \ldots, B_{p-1}\}$:

$$V(A_0, \ldots, A_{p-1}) = V(B_0, \ldots, B_{p-1}). \tag{6}$$

To prove eqn (6), we look at a single row i of A. Without loss of generality, we assume that row i is non-empty; the case of an empty row is trivial. Define $\lambda_i(A)$ as the number of parts owning nonzeros in row i in the partitioning of A. Define $\delta_i(B) = 1$ if b_{ii} is assigned to a different part than all the other nonzeros in row i in the partitioning of B, and $\delta_i(B) = 0$ otherwise.

The volume caused by row i of A equals

$$\begin{aligned} C(i, A_0, \ldots, A_{p-1}) &= \lambda_i(A) - 1 \\ &= \lambda_i(A^c) - 1 + \delta_{i+n}(B), \end{aligned} \tag{7}$$

because the nonzeros of row i in A^c are assigned to $\lambda_i(A^c)$ parts and those in A^r are all assigned to one part, namely the owner of $b_{i+n,i+n}$, yielding the term $\delta_{i+n}(B)$.

By inspecting row $i + n$ of B, using eqn (4) and eqn (5), we see that

$$\lambda_{i+n}(B) = \lambda_i(A^c) + \delta_{i+n}(B). \tag{8}$$

By combining eqn (7) and eqn (8), we see that the communication volume caused by row i of A equals

$$\begin{aligned} C(i, A_0, \ldots, A_{p-1}) &= \lambda_{i+n}(B) - 1 \\ &= C(i + n, B_0, \ldots, B_{p-1}), \end{aligned} \tag{9}$$

the volume caused by row $i + n$ of matrix B.

In the special case that row i of A^r is empty, the only nonzero in column $i + n$ of B is the diagonal nonzero. Our column partitioning of B then ensures that $\delta_{i+n}(B) = 0$, so that eqn (7) still holds. In the special case that row i of A^c is empty, $\delta_{i+n}(B) = 1$ by convention, and both row i of A and row $i + n$ of B do not cause communication.

We conclude that the communication volume caused by row i of A is equal to that of row $i+n$ of B. A similar proof can be made for a column j of A and row j of B. Since the total communication volume of the partitioned matrix A is the sum of the communication volumes of each individual row and column, this finishes the proof of eqn (6).

Note that the columns of B only containing the diagonal nonzero do not influence the resulting partitioning of A. We can therefore simply remove them from B and apply the row-net model to the resulting matrix, thereby ensuring that they do not cause any additional communication volume in B. Similarly, we can remove rows of B containing only the diagonal nonzero to reduce the problem size, without changing the resulting partitioning of A. Using the above proof, we also see that if all nonzeros of A are placed in A^c, the medium-grain method reduces to the row-net model. Similarly, if all nonzeros are placed in A^r, the medium-grain method reduces to the column-net model.

Considering load balance, it is easy to see from eqn (5) that the number of nonzeros in A_k equals the number of nondummy nonzeros in B_k. Hence, the load imbalance constraint for A (eqn (1)) is automatically satisfied when the corresponding constraint for B is satisfied.

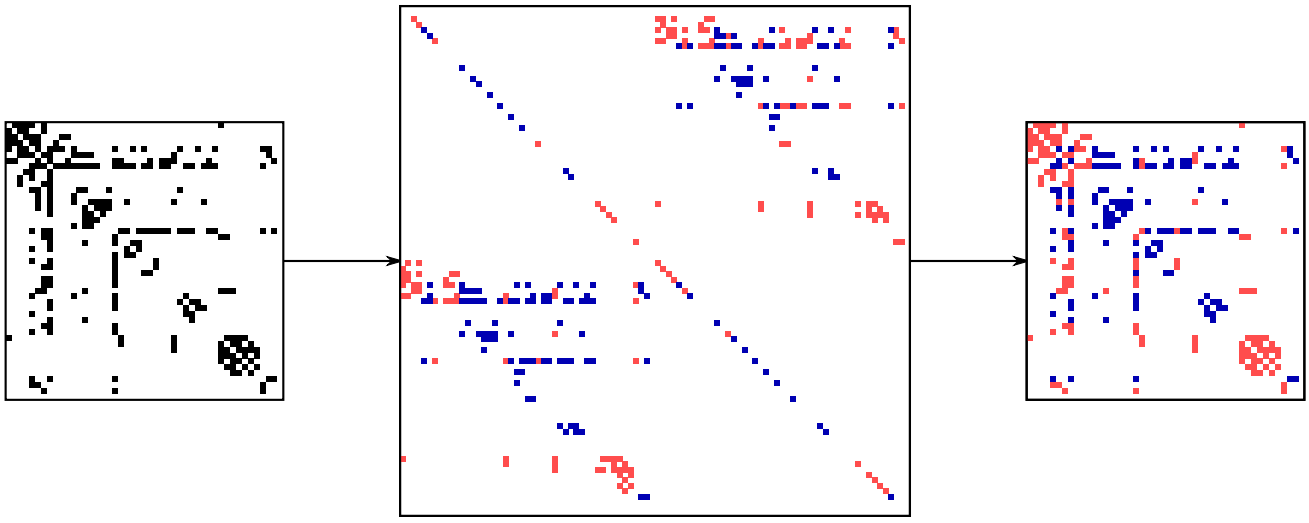


Figure 3: This figure shows the use of the medium-grain method to bipartition the `gd97_b` matrix (47×47, 264 nonzeros) from [18], with an allowed load imbalance $\varepsilon = 0.03$. The original matrix A is shown on the left, and the B matrix, partitioned by column, is shown in the middle. Each nonzero is given a color corresponding to the part to which it is assigned. The corresponding 2D partitioning of the original matrix is shown on the right. The communication volume is equal to 11, which was shown to be optimal in [19]. The partitioning of B was found using the Mondriaan software, version 3.11, with its internal hypergraph partitioner. The best communication volumes found for A in 100 runs using the row-net, column-net, and fine-grain models were 31, 31, and 12, respectively. In 19 out of 100 runs using the medium-grain method, the optimal volume of 11 was found.

The 2D medium-grain method presented in our paper can be viewed as tearing a single matrix A apart by some simple strategy into two matrices A^c and A^r, followed by simultaneous 1D partitioning of A^c and A^r using a composite hypergraph model, see Uçar and Aykanat [8]. In this application of the composite model, the columns of A^c and the rows of A^r are partitioned, with the load balanced for both matrices together, and with connections to both an input vector (of length n) and an output vector (of length m). Both connections are needed because the matrices are glued back together into $A = A^r + A^c$.

Alternatively, the medium-grain method can also be viewed as performing one extra, initial level of coarsening in the multi-level fine-grain method by merging all nonzeros (vertices) from a column of A^c into a single vertex, and all nonzeros from a row of A^r into a single vertex as well, creating hyperedges in a suitable fashion, and then running the partitioner on the resulting hypergraph with at most $m + n$ vertices.

B. Initial split of A

The question remains how to split A into A^r and A^c. This is an important problem, since the split determines what partitionings of A are possible in the subsequent hypergraph partitioning, and it will influence the final communication volume. Note that we perform this split heuristically: it is not *a priori* clear what the optimal method of splitting is.

To come up with a good heuristic, we first note that it is easy to prevent communication within a single row of A by putting all elements of that row in A^r. Similarly, we can prevent communication within a single column by putting it entirely in A^c. Of course, it is impossible to do both: if there is a nonzero a_{ij} in row i and column j, we cannot put row i entirely in A^r *and* put column j entirely in A^c.

An alternative is a compromise: we give each row and column of A a certain *score* that indicates its probability of being cut in a good partitioning, a low score meaning that we expect all nonzeros of that row or column to be assigned to the same processor in a good partitioning. In this paper, we use the number of nonzeros in a row or column as its score. Empirically, we found this to be a good choice, and intuitively it makes sense: in a good partitioning, it is more likely that small rows and columns are uncut, since it is harder to keep all nonzeros of a large row or column assigned to the same processor. We define $s_r(i)$ as the score of row i, and $s_c(j)$ as the score of column j.

The use of scores leads to a simple heuristic for the initial split: we place nonzero a_{ij} in A^r if $s_r(i) < s_c(j)$, and in A^c if $s_r(i) > s_c(j)$. In other words, for every nonzero, the row or column with the lowest score 'wins'. In case of a tie ($s_r(i) = s_c(j)$), we experimentally found that making a globally preferred choice is best: all ties are placed in either A^r or A^c. In the case of a rectangular matrix, the globally preferred choice is based on the matrix dimensions: if A has

more rows than columns, we place ties in A^r, and otherwise in A^c. For square matrices, we randomly pick A^r or A^c at the start of the algorithm as the globally preferred choice. Finally, a small improvement can be made by noting that if a row of A consists of only a single nonzero, it is better to place that nonzero in A^c, since the row will always be uncut. Columns of A that consist of a single nonzero can be treated similarly. The resulting algorithm is given as Algorithm 1.

After running Algorithm 1, we obtain another small improvement by modifying rows and columns that are completely placed in A^r and A^c, respectively, except for one nonzero. If every nonzero of row i is placed in A^r except for a_{ij}, we move a_{ij} from A^c to A^r, so we are ensured that there is no communication volume caused by row i in the final partitioning. Columns with only one nonzero in A^r, and the rest in A^c, are handled in a similar way. The whole procedure of partitioning a matrix with the medium-grain method is shown in Fig. 3.

Algorithm 1 Initial split algorithm

```
1: function SPLIT(A, m, n)
2:     if m > n then
3:         w ← r
4:     else if m < n then
5:         w ← c
6:     else
7:         w ← random r or c
8:     s_r(·) ← nz_r(·)          ▷ nz_r(i): number of nonzeros in row i
9:     s_c(·) ← nz_c(·)          ▷ nz_c(j): number of nonzeros in col j
10:    for every nonzero a_ij do
11:        if nz_c(j) = 1 then
12:            place a_ij in A^r
13:        else if nz_r(i) = 1 then
14:            place a_ij in A^c
15:        else if s_r(i) < s_c(j) then
16:            place a_ij in A^r
17:        else if s_r(i) > s_c(j) then
18:            place a_ij in A^c
19:        else
20:            place a_ij in A^w
21:    return A^r, A^c
```

C. Iterative refinement

The medium-grain method allows different ways of translating a matrix A to a matrix B, by varying the split of A into A^r and A^c. By exploiting this freedom, we can improve the partitioning quality of the medium-grain method. As an example, after partitioning a matrix, we can use information of the outcome to improve the initial split in a new run of partitioning using the medium-grain method. Here, we use this idea to perform *iterative refinement* on a partitioning of the medium-grain method. This iterative refinement procedure can also be applied to bipartitionings of other methods, as a cheap postprocessing step to improve the communication volume.

Bipartitioning a matrix A results in two sets of nonzeros A_0 and A_1, with the nonzeros assigned to processor 0 in A_0, and those assigned to processor 1 in A_1. Afterwards, we can create a new matrix B by placing all nonzeros of A_0 in A^r, and those of A_1 in A^c. Note that the resulting matrix is different from the one obtained by applying Algorithm 1 to A. In the new matrix B, we assign the columns of A^c to one processor, and the columns of $(A^r)^T$ to the other. By doing this, we ensure that the resulting partitioned matrix B has the same communication volume and load balance as the original bipartitioning of A.

After construction of the new matrix B, we create the corresponding bipartitioned hypergraph, and perform a single run of the Kernighan–Lin method on it. Note that we do not use a multi-level method, but only refine the current bipartitioning to a better one. This also means that our iterative refinement method does not increase the partitioning time significantly. After the refinement, we can repeat the process: translate the refined bipartitioning to a new matrix B, and apply refinement on that matrix. After this iteration, we can again repeat the process, and so on, thereby improving the solution at every iteration. Since the Kernighan–Lin method only decreases the communication volume or keeps it the same, the entire iterative refinement procedure is monotonically non-increasing.

Of course, it is also possible to create the new matrix B by placing all nonzeros of A_0 in A^c instead of A^r, and those of A_1 in A^r instead of A^c. Here we used both possibilities: we start by using the first option, where the nonzeros of A_0 are placed in A^r. Once iterative refinement does not reduce the communication volume any more, we switch to the second option, where the nonzeros of A_0 are placed in A^c. We keep applying iterative refinement with this second option, until this does not reduce the communication volume any more, after which we switch back to the first option. We keep switching between the options until no further reduction can be obtained. The iterative refinement method is summarised in Algorithm 2.

Iterative refinement based on the medium-grain method somewhat resembles the so-called V-cycle refinement included in hMetis [11], which is a multi-level postprocessing procedure with a restricted coarsening (respecting the current partitioning) followed by Kernighan–Lin refinement at all levels. This can improve the quality of the solution at the expense of increased computation time. Our refinement procedure involves only one level, the finest, and is relatively cheap as it does not involve coarsening. Furthermore, it exploits the splitting freedom offered by the medium-grain method as a way to encode information from the current partitioning in several different ways (for instance, different directions). Both refinement procedures are monotonically non-increasing with respect to the communication volume.

IV. Experiments

We implemented the new medium-grain method and the iterative refinement procedure as subroutines of the Mon-

Algorithm 2 Iterative refinement

```
1: function ITERATIVE_REFINE(A, m, n, A_0, A_1)        ▷ A = A_0 ∪ A_1
2:     dir ← 0
3:     V_0 ← V(A_0, A_1)
4:     k ← 1
5:     done ← False
6:     while not done do
7:         if dir = 0 then
8:             A^r ← A_0
9:             A^c ← A_1
10:        else
11:            A^c ← A_0
12:            A^r ← A_1
13:        create B using eqn (4)
14:        B_0 ← B(∗, 0 : n − 1)
15:        B_1 ← B(∗, n : m + n − 1)
16:        apply single run of Kernighan–Lin to B
17:        determine A_0 and A_1 using eqn (5)
18:        V_k ← V(A_0, A_1)
19:        if V_k = V_{k−1} then
20:            dir ← 1 − dir
21:        if k > 1 and V_k = V_{k−2} then
22:            done ← True
23:        k ← k + 1
24:    return A_0, A_1
```

driaan software package, version 4.0. This package is also able to partition matrices using the *row-net*, *column-net*, and *fine-grain* models, making comparisons between the different methods easy to perform. The Mondriaan software also includes the *localbest* method, which splits the matrix using both the row-net and column-net model, and returns the bipartitioning with the lowest communication volume of the two. All programs were compiled by GCC version 4.6.3 and executed on an Intel Core i7-2600K 3.4 GHz processor with 16 Gbyte of RAM, under the Fedora 16 Linux operating system (Linux kernel 3.6.6, x86_64).

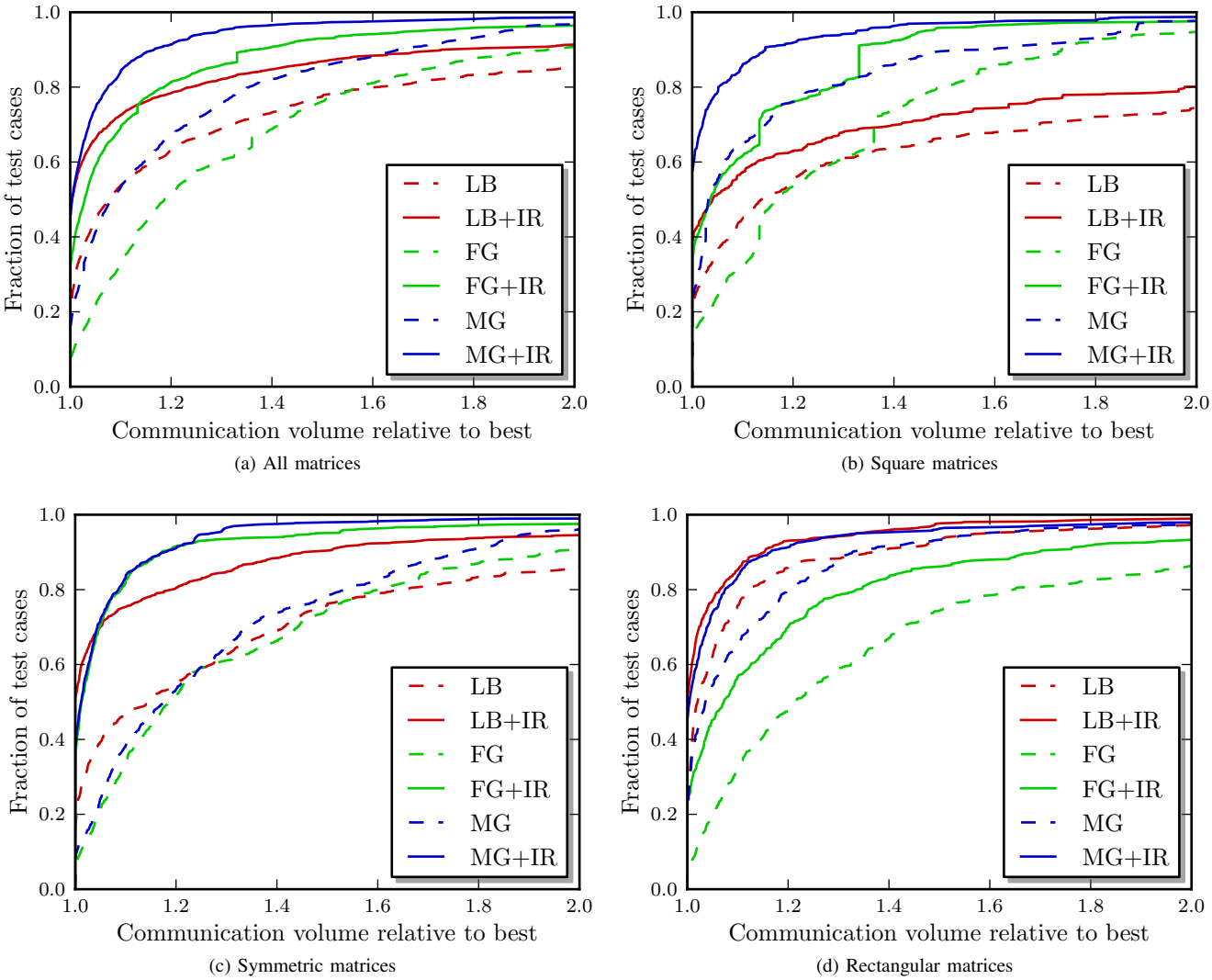
To test the new medium-grain method and the iterative refinement procedure, we bipartitioned all matrices from the University of Florida sparse matrix collection [18] with between 500 and 5,000,000 nonzeros. At the time of experimentation, there were 2267 matrices that satisfy these constraints. Three matrices (`LargeRegFile`, `ASIC_680k`, and `rajat29`) were excluded because partitioning them took an extremely long time, resulting in a test set of 2264 matrices, of which 582 are rectangular matrices, 1007 are structurally symmetric matrices, and 675 are square non-symmetric matrices. Specifically, the symmetric matrices have a nonzero pattern symmetry equal to one in the University of Florida sparse matrix collection, while the square non-symmetric matrices have a nonzero pattern symmetry smaller than one. For all matrices, we calculate the average communication volume and partitioning time of 10 runs of the Mondriaan software, comparing the *localbest* method, *fine-grain* method, and the *medium-grain* method, both with and without iterative refinement. We set the allowed load imbalance for every experiment to $\varepsilon = 0.03$, the value used in experiments in earlier studies [4], [6], [12]. The structurally symmetric matrices are partitioned without imposing a symmetric partitioning.

To compare the different bipartitioning methods, we use *performace profiles*, which were introduced by Dolan and Moré [20] as a tool to compare different methods for a certain metric over a large test set. Performance profiles were also used by Çatalyürek, Aykanat, and Uçar [10] to compare different matrix partitioning methods. In our case, a performance profile shows, for each partitioning method, the fraction of test matrices for which the resulting communication volume is within some factor of the lowest volume found by all methods. For example, if the performance profile of a method shows a fraction of 0.9 at factor 2, it shows that for 90% of the test matrices, the communication volume of the method was less than or equal to two times the lowest communication volume. For a given factor, the method with the highest fraction is best. Matrices for which the lowest communication volume found by all methods was equal to zero were removed from the test set, since they cannot be represented in the performance profile.

The performance profiles for the communication volume of the *localbest* method, *fine-grain* method, and the *medium-grain* method, both with and without *iterative refinement*, are shown in Fig. 4. Using the profile of *all* matrices (Fig. 4a), we can see that for around 90% of the matrices, the medium-grain method with iterative refinement produces a bipartitioning with a communication volume at most 1.2 times the best volume. The next best method, fine-grain with iterative refinement, has at most 1.2 times the best volume for around 80% of all matrices. The fine-grain method *without* iterative refinement produces partitionings with at most 1.2 times the best volume for around 50% of the matrices. From Fig. 4a we can conclude that the medium-grain method with iterative refinement produces, on average, bipartitionings with the lowest communication volume, and that applying iterative refinement improves communication volume significantly for all methods.

If we look at the performance profile of only the square unsymmetric matrices (Fig. 4b), only the symmetric matrices (Fig. 4c), and only the rectangular matrices (Fig. 4d), we see that the relative ranking of the methods is different for each matrix type. For square unsymmetric matrices, the localbest method performs relatively badly, while the medium-grain method with iterative refinement performs relatively well. For symmetric matrices, the iterative refinement procedure has the largest impact on the communication volume, and the difference between the medium-grain method and the fine-grain method is small. Finally, for rectangular matrices, the localbest method without iterative refinement produces the best bipartitionings of all methods without iterative refinement, and with iterative refinement it is tied with the medium-grain method.

In Fig. 5, the performance profile is shown for the partitioning time, for all matrices. As expected from the size



(a) All matrices

(b) Square matrices

(c) Symmetric matrices

(d) Rectangular matrices

Figure 4: Performance profile plots comparing the communication volume, using Mondriaan's internal hypergraph partitioner, of the *localbest* (LB), *fine-grain* (FG), and *medium-grain* (MG) methods, both with and without *iterative refinement* (IR).

of the corresponding hypergraphs, the medium-grain method and localbest method are faster than the fine-grain method. An interesting result is that the medium-grain method is even faster than the localbest method. An explanation for the relatively small partitioning time of the medium-grain method is that for many matrices, the number of vertices of the hypergraph in the medium-grain method is lower than $m + n$, since many columns of the matrix B (eqn (4)) will only consist of the diagonal dummy nonzero. Figure 5 also shows that the iterative refinement procedure does not greatly increase the partitioning time.

The geometric means of the communication volume and partitioning time of the different methods are shown in Table I. For each matrix, the volumes and times were normalized with respect to the volume and time of the localbest method without iterative refinement (the default method of Mondriaan version 3.11). We then take the geometric mean of the resulting normalized communication volumes and partitioning times to obtain a single average normalized value for each method. The results are in agreement with the performance profiles of Fig. 4, with the medium-grain method with iterative refinement having the lowest average communication volume. Specifically, the communication volume of the medium-grain method with iterative refinement is, on average, 27% lower than the volume of the localbest method, and the partitioning time is 28% less. The second-best method, fine-grain with iterative refinement, produces bipartitionings with, on average, 23% lower volume than the localbest method, but is almost two

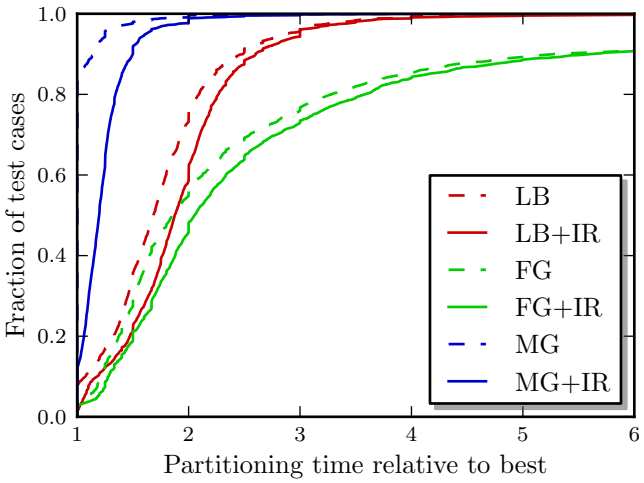


Figure 5: Performance profile plot comparing the partitioning time for all matrices using Mondriaan's internal hypergraph partitioner. For explanation of the abbreviations, see Fig. 4.

		LB	LB+IR	MG	MG+IR	FG	FG+IR
Com. Vol.	Rec	1.00	**0.94**	1.02	0.96	1.28	1.11
	Sym	1.00	0.75	0.80	**0.67**	0.88	0.69
	Sqr	1.00	0.77	0.68	**0.62**	0.76	0.66
	All	1.00	0.80	0.81	**0.73**	0.93	0.77
Time	Rec	1.00	1.05	**0.53**	0.60	1.08	1.18
	Sym	1.00	1.14	**0.64**	0.79	1.55	1.70
	Sqr	1.00	1.08	**0.66**	0.75	1.23	1.32
	All	1.00	1.10	**0.62**	0.72	1.32	1.43

Table I: Geometric means of the communication volume and partitioning time using Mondriaan's internal hypergraph partitioner, calculated relative to the *localbest* method without iterative refinement. Results are shown for the rectangular (Rec), symmetric (Sym), and square unsymmetric (Sqr) matrices, and all matrices (All) of the test set. The best (lowest) value of each row is shown in boldface. For explanation of the method abbreviations, see Fig. 4.

times slower than the medium-grain method with iterative refinement. Table I also shows that partitioning with iterative refinement is roughly 10% slower than partitioning without iterative refinement, but produces results with roughly 20% lower communication volume.

In Fig. 6a, the performance profile plot is shown of the communication volume for bipartitioning of all matrices, obtained using PaToH [4] as hypergraph partitioner instead of the internal partitioner of Mondriaan. For the medium-grain and localbest methods, the results are similar to those obtained using Mondriaan's internal partitioner (Fig. 4a). The fine-grain method, however, performs better using PaToH, both with and without iterative refinement. Using PaToH, the

	p	LB	LB+IR	MG	MG+IR	FG	FG+IR
Vol	2	1.00	0.81	0.76	**0.67**	0.71	**0.67**
Cost	2	1.00	0.82	0.78	**0.69**	0.73	**0.69**
Vol	64	1.00	0.86	0.89	**0.80**	0.87	**0.80**
Cost	64	1.00	0.78	0.75	**0.68**	0.72	**0.68**

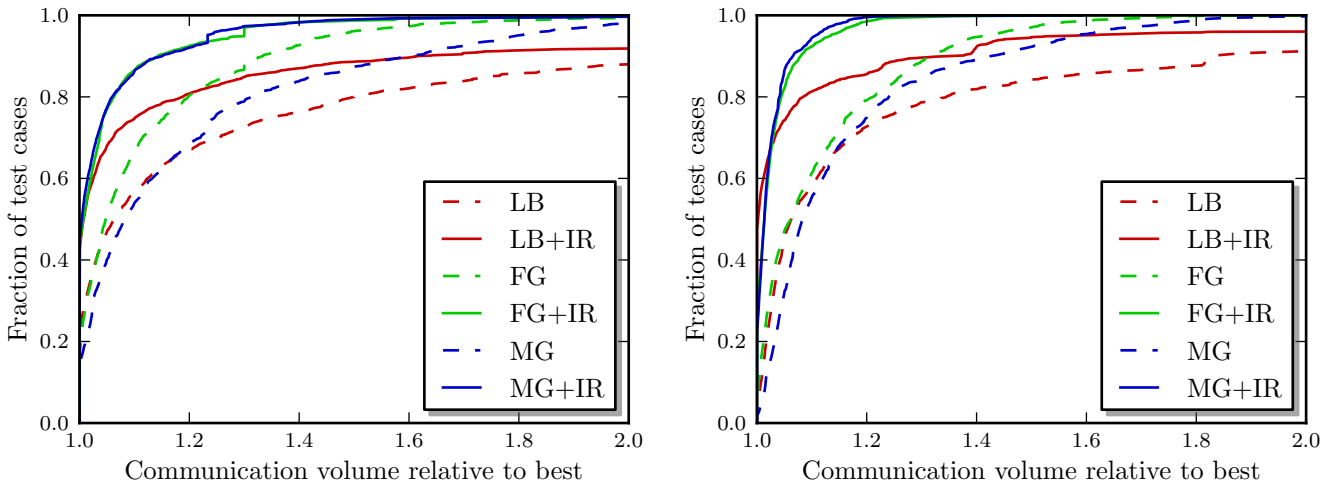
Table II: Geometric means of the communication volume (**Vol**) and BSP cost (**Cost**) for $p = 2$ and $p = 64$, using PaToH as hypergraph partitioner, calculated relative to the *localbest* method without iterative refinement. The best (lowest) value of each row is shown in boldface. For explanation of the method abbreviations, see Fig. 4.

fine-grain method with iterative refinement performs as well as the medium-grain method with iterative refinement. It is important to note, however, that both produce bipartitionings with significantly lower communication volume compared to the fine-grain method without iterative refinement, the current state-of-the-art method. Furthermore, the results for the partitioning time using PaToH as partitioner are similar to those using Mondriaan's internal partitioner, with the medium-grain method being significantly faster than the other methods.

The medium-grain method can also be used in a recursive bisection scheme to obtain partitionings into p parts. For $p = 64$ and using PaToH as partitioner, the performance profile plot of the resulting communication volume is shown in Fig. 6b. Results are similar to partitioning with $p = 2$ using PaToH (Fig. 6a), although the iterative refinement procedure has an even larger impact for $p = 64$. Geometric means of the communication volume for $p = 2$ and $p = 64$ are given in Table II. We also show a different metric, the BSP (Bulk Synchronous Parallel) cost, defined as the sum of the maximum number of data words that are sent or received by a single processor during the fan-in and fan-out phase of a parallel matrix-vector multiplication [1]. The medium-grain method and the fine-grain method, both with iterative refinement, produce partitionings with the lowest BSP cost, similar to the results for the communication volume.

V. Conclusion and Future Work

In this paper, we presented a new method for bipartitioning sparse matrices, the *medium-grain* method. This method has several advantages compared to existing methods. It is able to produce 2D bipartitionings of matrices directly, while the row-net and column-net models can only produce 1D bipartitionings, and are therefore restrictive. Although the fine-grain model is also 2D in nature, it is very large: for an $m \times n$ matrix with N nonzeros, the number of vertices in the hypergraph equals N. The size of the fine-grain model makes computation on it relatively slow, and makes it harder to find good solutions to the underlying hypergraph partitioning problem. The number of vertices in the hypergraph of the



(a) $p = 2$ (b) $p = 64$

Figure 6: Performance profile plots comparing the communication volume for all matrices using PaToH as hypergraph partitioner. For explanation of the abbreviations, see Fig. 4.

medium-grain method equals $m+n$, which is usually much smaller than N. Therefore, computations on the hypergraph of the medium-grain method are much faster than on the fine-grain model, and it is easier to find good solutions to the underlying hypergraph partitioning problem. Furthermore, the smaller hypergraph size enables us to partition larger matrices than is possible with the fine-grain model. We also presented a cheap iterative refinement procedure, based on the medium-grain method, which can be applied to improve the quality of the solution after any partitioning method.

Results of experiments performed in this paper on a set of 2264 matrices with up to 5,000,000 nonzeros seem to confirm the advantages of the medium-grain method. For bipartitioning using Mondriaan's internal hypergraph partitioner, the medium-grain method produced partitionings with lower communication volume than the fine-grain method and the localbest method, which is a combination of the row-net and column-net models and represents the best 1D method for $p = 2$. The medium-grain method with iterative refinement produces partitionings with, on average, 27% lower communication volume than the localbest method, and 22% lower than the fine-grain method, both without iterative refinement. Only in the case of rectangular matrices that are far from square, i.e. with $m \ll n$ or $m \gg n$, the medium-grain method does not yield benefits for $p = 2$, because a 1D partitioning in the proper direction (the row direction for $m > n$) will often already be very good, and the localbest method will choose this direction. Using PaToH as hypergraph partitioner, the medium-grain method with iterative refinement and the fine-grain method with iterative refinement both produce partitionings with the lowest communication volume on average.

In addition to obtaining partitionings with the lowest communication volume, the medium-grain method is also faster in producing these partitionings, taking, on average, 28% less time than the localbest method, and 45% less time than the fine-grain method. The results also show that the iterative refinement procedure can be used to improve bipartitionings of all methods, without increasing the computation time significantly, especially for symmetric matrices. Specifically, for all methods, bipartitioning with iterative refinement is roughly 10% slower than bipartitioning without iterative refinement, but produces bipartitionings with roughly 20% lower communication volume.

The quality and speed of the medium-grain method is such that we have made it the default option in Mondriaan 4.0. The quality of the partitionings produced by the medium-grain method might be further improved by using a different initial split algorithm. The current splitter, although able to outperform existing models and methods, may not be the best possible choice. It is important to note that every possible bipartitioning of a matrix A into sets of nonzeros A_0 and A_1, including the optimal one, can be translated to a column partitioning of an extended matrix B, by placing all nonzeros of A_0 in A^r and all nonzeros of A_1 in A^c, or vice versa. So in theory, an ideal initial split algorithm will allow the medium-grain method to represent the optimal bipartitioning of matrix A. Whether a practical algorithm can be found that is able to produce optimal initial splits for all matrices remains a subject for further research.

The medium-grain method is aimed at finding a good partitioning for parallel computations. It can be implemented sequentially, as has been done in this paper, by using a sequential hypergraph partitioner such as PaToH [4] or

the internal hypergraph partitioner of Mondriaan [12]. To avoid a potential sequential bottleneck, it may be desirable to parallelize the partitioning itself as well; this can be done by employing a parallel hypergraph partitioner such as Zoltan [13] and parallelizing Algorithms 1 and 2.

The initial split by Algorithm 1 of the matrix A into A^r and A^c can be parallelized by first broadcasting score values so that the owner of nonzero a_{ij} knows both scores $s_r(i)$ and $s_c(j)$, then deciding on inclusion of nonzeros in either A^r or A^c, thus creating B, and after that moving each nonzero of B to the responsible processor in the input distribution required by the parallel hypergraph partitioner (typically a row, column, or 2D Cartesian distribution). The output is a partitioning of B, in distributed format, which can easily be transformed to a partitioning of A in any desired distribution by suitably moving nonzeros. The iterative refinement by Algorithm 2 can be parallelized in similar fashion, using a parallel Kernighan–Lin run.

As explained in Section III-C, it is also possible to apply the medium-grain method in an iterative way: after partitioning a matrix, we can use information of the outcome to improve the initial split in a new run of partitioning using the medium-grain method. Instead of using this idea for iterative refinement only, as we did in this paper, one can also design a full iterative method, where a full multilevel partitioning is performed in each iteration. This would present an entirely new method of solving the sparse matrix partitioning problem, where one could trade computation time for solution quality, by using more or less iterations. Furthermore, other ways of using information of a previous iteration might improve results for both iterative refinement and full iterative methods.

REFERENCES

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

[2] Ü. V. Çatalyürek, M. Deveci, K. Kaya, and B. Uçar, "UMPa: A multi-objective, multi-level partitioner for communication minimization," in *Graph Partitioning and Graph Clustering*, ser. Contemporary Mathematics, D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, Eds., vol. 588. AMS, 2013, pp. 53–65.

[3] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*. John Wiley and Sons, Chichester, UK, 1990.

[4] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.

[5] ——, "A fine-grain hypergraph model for 2D decomposition of sparse matrices," in *Proc. Eighth International Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001)*. IEEE Press, Los Alamitos, CA, 2001, p. 118.

[6] R. H. Bisseling, B. O. Fagginger Auer, A. N. Yzelman, T. van Leeuwen, and Ü. V. Çatalyurek, "Two-dimensional approaches to sparse matrix partitioning," in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Chapman & Hall / CRC Press, 2012, pp. 321–349.

[7] B. Uçar and C. Aykanat, "Revisiting hypergraph models for sparse matrix partitioning," *SIAM Review*, vol. 49, no. 4, pp. 595–603, 2007.

[8] ——, "Partitioning sparse matrices for parallel preconditioned iterative methods," *SIAM Journal on Scientific Computing*, vol. 29, no. 4, pp. 1683–1709, 2007.

[9] Ü. V. Çatalyürek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proc. Supercomputing 2001*. ACM Press, New York, 2001, p. 28.

[10] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 656–683, 2010.

[11] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. 36th ACM/IEEE Conference on Design Automation*. ACM Press, New York, 1999, pp. 343–348.

[12] 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.

[13] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proc. IEEE International Parallel and Distributed Processing Symposium 2006*. IEEE Press, Los Alamitos, CA, 2006, p. 124.

[14] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Improved algorithms for hypergraph bipartitioning," in *Proc. Asia and South Pacific Design Automation Conference*. ACM Press, New York, 2000, pp. 661–666.

[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, pp. 291–307, 1970.

[16] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th IEEE Design Automation Conference*. IEEE Press, Los Alamitos, CA, 1982, pp. 175–181.

[17] T. N. Bui and C. Jones, "A heuristic for reducing fill-in in sparse matrix factorization," in *Proc. Sixth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Philadelphia, PA, 1993, pp. 445–452.

[18] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, p. 1:1–1:25, 2011.

[19] D. M. Pelt, "Matrix Partitioning: Optimal bipartitioning and heuristic solutions." Master's thesis, Utrecht University, 2010.

[20] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.