UMPa: A Multi-objective, multi-level partitioner for communication minimization

Umit V. Catalyurek, Mehmet Deveci, Kamer Kaya, Bora Uçar

UMPa: A Multi-objective, multi-level partitioner for communication minimization

Ümit V. Çatalyürek, Mehmet Deveci, Kamer Kaya, and Bora Uçar

Abstract. We propose a directed hypergraph model and a refinement heuristic to distribute communicating tasks among the processing units in a distributed memory setting. The aim is to achieve load balance and minimize the maximum data sent by a processing unit. We also take two other communication metrics into account with a tie-breaking scheme. With this approach, task distributions causing an excessive use of network or a bottleneck processor which participates in almost all of the communication are avoided. We show on a large number of problem instances that our model improves the maximum data sent by a processor up to 34% for parallel environments with $4, 16, 64$ and 256 processing units compared to the state of the art which only minimizes the total communication volume.

1. Introduction

In parallel computing, the problem of distributing communicating tasks among the available processing units is important. To solve this problem, several graph and hypergraph models are proposed [**6, 7, 9, 12, 20**]. These models transform the problem at hand to a balanced partitioning problem. The balance restriction on part weights in conventional partitioning corresponds to the load balance in the parallel environment, and the minimized objective function corresponds to the total communication volume between processing units. Both criteria are crucial in practice for obtaining short execution times, using less power, and utilizing the computation and communication resources better.

In addition to the total data transfer, there are other communication metrics investigated before, e.g., total number of messages sent [**19**], or maximum volume of messages sent and/or received by a processor [**4, 19**]. Even with perfect load balancing and minimized total data transfer, there can be a bottleneck processing unit which participates in most of the data transfers. This can create a problem especially for data intensive applications where reducing the amount of data transferred by the bottleneck processing unit can improve the total execution time significantly.

2010 *Mathematics Subject Classification.* Primary 05C65, 05C70; Secondary 90C35.

This work was supported in parts by the DOE grant DE-FC02-06ER2775 and by the NSF grants CNS-0643969, OCI-0904809, and OCI-0904802.

In this work, given a task graph, our main objective is distributing its tasks evenly and minimizing the maximum amount of data sent by a processing unit. Previous studies addressing different communication cost metrics (such as [**4, 19**]) work in two phases. In the first phase, the total volume of communication is reduced, and in the second phase the other metrics are addressed. We propose a directed hypergraph model and partition the related hypergraph with a multi-level approach and a novel K-way refinement heuristic. While minimizing the primary objective function, our refinement heuristic also takes the maximum data sent and received by a processing unit and the total amount of data transfer into account by employing a tie-breaking scheme. Therefore, our approach is different from the existing studies in that the objective functions are minimized all at the same time.

The organization of the paper is as follows. In Section 2, the background material on graph and hypergraph partitioning is given. Section 2.3 shows the differences of the graph and hypergraph models and describes the proposed directed hypergraph model. In Section 3, we present our multi-level, multi-objective partitioning tool UMPa (pronounced as "Oompa"). Section 4 presents the experimental results, and Section 5 concludes the paper.

2. Background

2.1. Hypergraph partitioning. A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices. A net $n \in \mathcal{N}$ is a subset of vertices and the vertices in n are called its *pins*. The number of pins of a net is called the *size* of it, and the *degree* of a vertex is equal to the number of nets it is connected to. In this paper, we will use $\mathsf{pins}[n]$ and $\mathsf{nets}[v]$ to represent the pin set of a net n and the set of nets vertex v is connected to, respectively. Vertices can be associated with weights, denoted with $\mathsf{w}[\cdot]$, and nets can be associated with costs, denoted with $\mathsf{c}[\cdot]$.

A *K-way partition* of a hypergraph $\mathcal{H}$ is denoted as $\Pi=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K\}$ where

- parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for all $1 \le k < \ell \le K$,
- each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, i.e., $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \le k \le K$,
- union of K parts is equal to $\mathcal{V}$, i.e., $\bigcup_{k=1}^{K} \mathcal{V}_k = \mathcal{V}$.

Let W_k denote the total vertex weight in $\mathcal{V}_k$ (i.e., $W_k=\sum_{v\in\mathcal{V}_k}\mathsf{w}[v]$) and W_{avg} denote the weight of each part when the total vertex weight is equally distributed (i.e., $W_{avg}=(\sum_{v\in\mathcal{V}}\mathsf{w}[v])/K$). If each part $\mathcal{V}_k \in \Pi$ satisfies the *balance criterion*

$$W_k \le W_{avg}(1+\varepsilon), \quad \text{for } k=1,2,\ldots,K \tag{2.1}$$

we say that Π is *ϵ-balanced* where ε represents the maximum allowed imbalance ratio.

For a K-way partition Π, a net that has at least one pin (vertex) in a part is said to *connect* that part. The number of parts connected by a net n, i.e., *connectivity*, is denoted as λ_n. A net n is said to be *uncut* (*internal*) if it connects exactly one part (i.e., $\lambda_n = 1$), and *cut* (*external*), otherwise (i.e., $\lambda_n > 1$).

The set of external nets of a partition Π is denoted as $\mathcal{N}_E$. There are various cutsize definitions [**16**] for hypergraph partitioning. The one that will be used in this work, which is shown to accurately model the total communication volume [**7**], is called the *connectivity* metric and defined as:

$$\chi(\Pi) = \sum_{n \in \mathcal{N}} \mathsf{c}[n](\lambda_n - 1) \; . \tag{2.2}$$

In this metric, each cut net n contributes $\mathsf{c}[n](\lambda_n - 1)$ to the cutsize. The hypergraph partitioning problem can be defined as the task of finding a balanced partition Π with K parts such that $\chi(\Pi)$ is minimized. This problem is also NP-hard [**16**].

2.2. ***K*-way partitioning and multi-level framework.** Arguably, the multi-level approach [**3**] is the most successful heuristic for the hypergraph partitioning problem. Although, it has been first proposed for recursive-bisection based graph partitioning, it also works well for hypergraphs [**2, 5, 7, 13, 17**]. In the multi-level approach, a given hypergraph is coarsened to a much smaller one, a partition is obtained on the the smallest hypergraph, and that partition is projected to the original hypergraph. These three phases will be called the coarsening, initial partitioning, and uncoarsening phases, respectively. The coarsening and uncoarsening phases have multiple levels. In a coarsening level, similar vertices are merged to make the hypergraph smaller. In the corresponding uncoarsening level, the merged vertices are split, and the partition of the coarser hypergraph is refined for the finer one.

Most of the multi-level partitioning tools used in practice are based on recursive bisection. In recursive bisection, the multi-level approach is used to partition a given hypergraph into two. Each of these parts is further partitioned into two recursively until K parts are obtained in total. Hence, to partition a hypergraph into $K = 2^k$, the recursive bisection approach uses $K - 1$ coarsening, initial partitioning, and uncoarsening phases.

Several successful clustering heuristics are proposed to coarsen a hypergraph. Although their similarity metrics aim to reduce the cutsize, they cannot find an optimal solution, since the problem is NP-hard. Hence, an optimal partition of the coarser hypergraph may not be optimal for the finer one. To obtain better partitions, iterative-improvement-based heuristics are used to refine the coarser's partition after projecting it to finer. In practice, Kernighan-Lin (KL) [**15**] and Fiduccia-Mattheyses (FM) [**11**] based refinement heuristics that depend on vertex swaps and moves between two parts are used.

2.3. Task graph and communication volume metrics. Let $\mathcal{A} = (\mathcal{T}, \mathcal{C})$ be a task graph where $\mathcal{T}$ is the set of tasks to be executed, and $\mathcal{C}$ is the set of communications between pairs of tasks. We assume that the execution time of each task may differ, hence each task $t \in \mathcal{T}$ is associated with an execution time $exec(t)$. Each task $t_i \in \mathcal{T}$ sends a different amount of data $data(t_i)$ to each t_j such that $t_i t_j \in \mathcal{C}$. The communications between tasks may be uni-directional, That is $t_i t_j \in \mathcal{C}$ does not imply $t_j t_i \in \mathcal{C}$. In our parallel setting, we assume owner computes rule and hence, each task of $\mathcal{A}$ is executed by the processing unit to which it is assigned. Let $\mathcal{T}_i$ be the set of tasks assigned to processing unit P_i. Since it is desirable to distribute the tasks evenly, the computational load $\sum_{t \in \mathcal{T}_i} exec(t)$ should be almost the same for each P_i. In addition to that two heavily communicating tasks should be assigned to the same processing unit since less data transfer over the network is needed in this case. The total amount of data transfer throughout the execution of the tasks is called the *total communication volume* (*totV*). Note that when a task $t \in \mathcal{T}_i$ needs to send data to a set of tasks in $\mathcal{T}_j$, the contribution to $totV$ is $data(t)$, since it is enough to send t's data to P_j only once.

Although minimizing the total communication volume is important, it is sometimes preferable to reduce other communication metrics [**12**]. For example, in the context of one-dimensional partitioning of structurally unsymmetric sparse matrices for parallel matrix-vector multiplies, Uçar and Aykanat used a communication hypergraph model to reduce the maximum of number of messages and the maximum amount of data sent and received by a processor [**19**] (see also [**4**] and [**18**] for other communication metrics).

Let $SV[i]$ and $RV[i]$ be the volumes of communication sent and received by P_i, respectively. Hence, the total communication volume equals to $totV = \sum_i \mathsf{SV}[i] = \sum_i \mathsf{RV}[i]$. In addition to $totV$, we are interested in two other communication metrics: *maximum send volume* ($maxSV$), which equals to $\max_i(\mathsf{SV}[i])$; and *maximum send-receive volume* ($maxSRV$), which is $\max_i(\mathsf{SV}[i] + \mathsf{RV}[i])$.

3. UMPa: A multi-objective partitioning tool for communication minimization

3.1. Directed hypergraph model. We propose modeling the task graphs with directed hypergraphs. Given a task graph $\mathcal{A}$, we construct the directed hypergraph model $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ as follows. For each task $t_i \in \mathcal{T}$, we have a corresponding vertex $v_i \in \mathcal{V}$ and a net $n_i \in \mathcal{N}$ where $\mathsf{pins}[n_i] = \{v_i\} \cup \{v_j \mid t_i t_j \in \mathcal{C}\}$, $\mathsf{w}[v_i] = exec(t_i)$, and $\mathsf{c}[n_i] = data(t_i)$. In this directed hypergraph model, the communication represented by a net n is flowing from its source vertex, which will be denoted as $s(n)$, to the target vertices $\mathsf{pins}[n] \setminus \{s(n)\}$. Given a partition Π, let $\delta(n, \mathcal{V}_i) = 1$ if $n \cap \mathcal{V}_i \neq \emptyset$, and 0, otherwise. Then the data sent and received by P_i are equal to $\mathsf{SV}[i] = \sum_{n, s(n) \in \mathcal{V}_i} \mathsf{c}[n](\lambda_n - 1)$ and $\mathsf{RV}[i] = \sum_{n, s(n) \notin \mathcal{V}_i} \mathsf{c}[n]\delta(n, \mathcal{V}_i)$, respectively. Our primary objective is to minimize $maxSV$, the maximum send volume. While doing this, we also take the maximum send-receive volume and the total communication volume into account. The total volume of communication corresponds to the cutsize definition (2.2) as in the standard hypergraph model. In other words, the sense of direction is not important for the total communication volume $totV$. On the other hand, the directions of the flow is crucial while minimizing $maxSV$ and $maxSRV$.

To optimize its metrics, UMPa follows the multi-level approach. Instead of a recursive bisection, it adopts a direct K-way partitioning. Given the hypergraph, UMPa gradually coarsens it, obtains an initial K-way partition for the coarsest hypergraph, and projects it into the original one by uncoarsening and refinement steps at each level.

3.2. Multi-level coarsening phase. In this phase, the original hypergraph is gradually coarsened in multiple levels by clustering subsets of vertices at each level. There are two types of clustering algorithms: matching-based ones and agglomerative ones. The matching-based ones put at most two similar vertices in a cluster, whereas the agglomerative ones allow any number of similar vertices. There are various similarity metrics—see for example [**1, 7, 14**]. All these metrics are defined only on two adjacent vertices (one of them can be a vertex cluster). Two vertices are adjacent if they share a net and they can be in the same cluster if the are adjacent.

In this work, we use an agglomerative algorithm and the absorption clustering metric using pins [**1, 8**]. For this metric, the similarity between two adjacent vertices u and v is

$$\sum_{n \in \mathsf{nets}[u] \cap \mathsf{nets}[v]} \frac{\mathsf{c}[n]}{|\mathsf{pins}[n]| - 1}$$

This is also the default metric in PaToH [**8**], a well-known hypergraph partitioner. In each level ℓ, we start with a finer hypergraph $\mathcal{H}^\ell$ and obtain a coarser one $\mathcal{H}^{\ell+1}$. If $\mathcal{V}_C \subset \mathcal{V}^\ell$ is a subset of vertices deemed to be clustered, we create the cluster vertex $u \in \mathcal{V}^{\ell+1}$ where $\mathsf{nets}[u] = \cup_{v \in \mathcal{V}_C} \mathsf{nets}[v]$. We also update the pin sets of the nets in $\mathsf{nets}[u]$ accordingly.

Since we need the direction, i.e., source vertex information for each net to minimize $maxSV$ and $maxSRV$, we always store the source vertex of a net $n \in \mathcal{N}$ as the first pin in $\mathsf{pins}[n]$. To maintain this information, when a cluster vertex u is formed in the coarsening phase, we put u to the head of $\mathsf{pins}[n]$ for each net n whose source vertex is in the cluster.

3.3. Initial partitioning phase. To obtain an initial partition for the coarsest hypergraph, we use PaToH [**8**], which is proven to produce high quality partitions with respect to total communication volume metric [**7**]. We execute PaToH ten times and get the best partition according to the $maxSV$ metric. We have several reasons to use PaToH. First, although our main objective is minimizing $maxSV$, since we also take $totV$ into account, it is better to start with an initial partition having a good total communication volume. Second, since $totV$ is the sum of the send volumes of all parts, as we observed in our preliminary experiments, minimizing it may also be good for both $maxSV$ and $maxSRV$. Also, as stated in [**2**], using recursive bisection and FM-based improvement heuristics for partitioning the coarsest hypergraph is favorable due to small net sizes and high vertex degrees.

3.4. K-way refinement of communication volume metrics. In an uncoarsening level, which corresponds to the ℓth coarsening level, we project the partition $\Pi^{\ell+1}$ obtained for $\mathcal{H}^{\ell+1}$ to $\mathcal{H}^\ell$. Then, we refine it by using a novel K-way refinement heuristic which is described below.

Given a partition Π, let a vertex be a *boundary vertex* if it is in the pin set of at least one cutnet. Let $\Lambda(n,p) = |\mathsf{pins}[n] \cap \mathcal{V}_p|$ be the number of pins of net n in part p, and $\mathsf{part}[u]$ be the current part of u. The proposed heuristic runs in multiple passes where in a pass it visits each boundary vertex u and either leaves it in $\mathsf{part}[u]$, or moves it to another part according to some move selection policy. Algorithm 1 shows a pass of the proposed refinement heuristic. For each visited boundary vertex u and for each available part p other than $\mathsf{part}[u]$, the heuristic computes how the communication metrics are affected when u is moved to p. This is accomplished in three steps. First, u is removed from $\mathsf{part}[u]$, and the leave gains on the send/receive volumes of the parts are computed (after line 1). Second, u is put into a candidate part p and the arrival losses on the send/receive volumes are computed (after line 2). Last, the maximum send, maximum send-receive, and total volumes are computed for this move (after line 4).

3.4.1. *Move selection policy and tie-breaking scheme.* Our move selection policy given in Algorithm 2 favors the moves with the maximum gains on $maxSV$ and never allows a move with negative gain on the same metric. To take other metrics

Algorithm 1: A pass for K-way refinement

```
  Data: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, boundary[], part[], SV[], RV[], $\lambda$, $\Lambda$
  for each unlocked $u \in$ boundary do
      $receiveGain \leftarrow 0$
      $uToPartU \leftarrow 0$
      sendGain[] $\leftarrow 0$
1     for each $n \in$ nets[$u$] do
          if $s(n) = u$ then
              sendGain[part[$u$]] $\leftarrow$ sendGain[part[$u$]] $+ (\lambda_n - 1)$c[$n$]
              if $\Lambda(n, \text{part}[u]) > 1$ then
                  $receiveGain \leftarrow receiveGain -$ c[$n$] $uToPartU \leftarrow uToPartU +$ c[$n$]
          else if $\Lambda(n, \text{part}[u]) = 1$ then
              sendGain[part[$s(n)$]] $\leftarrow$ sendGain[part[$s(n)$]] $+$ c[$n$]
              $receiveGain \leftarrow receiveGain +$ c[$n$]
      $(bestMaxSV, bestMaxSRV, bestTotV) \leftarrow (maxSV, maxSRV, totV)$
      $bestPart \leftarrow$ part[$u$]
      for each part $p$ other than part[$u$] do
          if $p$ has enough space for vertex $u$ then
              $receiveLoss \leftarrow 0$
              sendLoss[] $\leftarrow 0$
2             sendLoss[$p$] $\leftarrow$ sendGain[part[$u$]] $+ uToPartU$
3             for each $n \in$ nets[$u$] do
                  if $s(n) = u$ then
                      if $\Lambda(n, p) > 0$ then
                          sendLoss[$p$] $\leftarrow$ sendLoss[$p$] $-$ c[$n$]
                          $receiveLoss \leftarrow receiveLoss -$ c[$n$]
                  else if $\Lambda(n, p) = 0$ then
                      sendLoss[part[$s(n)$]] $\leftarrow$ sendLoss[part[$s(n)$]] $+$ c[$n$]
                      $receiveLoss \leftarrow receiveLoss +$ c[$n$]
4             $(moveSV, moveSRV) \leftarrow (-\infty, -\infty)$
5             for each part $q$ do
                  $\Delta_S \leftarrow$ sendLoss[$q$] $-$ sendGain[$q$]
                  $\Delta_R \leftarrow 0$
                  if $q = part[u]$ then
                      $\Delta_R \leftarrow receiveGain$
                  else if $q = p$ then
                      $\Delta_R \leftarrow receiveLoss$
                  $moveSV \leftarrow \max(moveSV,$ SV[$q$] $+ \Delta_S)$
                  $moveSRV \leftarrow \max(moveSRV,$ SV[$q$] $+ \Delta_S +$ RV[$q$] $+ \Delta_R)$
              $moveV \leftarrow totV + receiveLoss - receiveGain$
6             MoveSelect($moveSV, moveSRV, moveV, p,$
                          $bestMaxSV, bestMaxSRV, bestTotV, bestPart$)
      if $bestPart \neq$ part[$u$] then
          move $u$ to $bestPart$ and update data structures accordingly
```

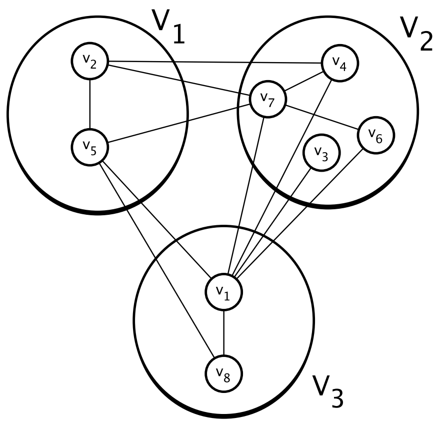
into account, we use a tie-breaking scheme which is enabled when two different moves of a vertex u have the same $maxSV$ gain. In this case, the move with $maxSRV$ gain is selected as the best move. If the gains on $maxSRV$ are also equal then the move with maximum gain on $totV$ is selected. We do not allow a vertex move without a positive gain on any of the communication metrics. As the experimental results show, this move selection policy and tie-breaking scheme have positive impact on all the metrics.

Algorithm 2: MoveSelect

```
Data: $moveSV, moveSRV, moveV, p,$
      $bestMaxSV, bestMaxSRV, bestTotV, bestPart$
$select \leftarrow 0$
if $moveSV < bestMaxSV$ then
    $select \leftarrow 1$                                  ▷Main objective
1 else if $moveSV = bestMaxSV$ then
    if $moveSRV < bestMaxSRV$ then
        $select \leftarrow 1$                              ▷First tie break
2 else if $moveSV = bestMaxSV$ then
    if $moveSRV = bestMaxSRV$ then
        if $moveV < bestTotV$ then
            $select \leftarrow 1$                          ▷Second tie break
if $select = 1$ then
    $bestMaxSV \leftarrow moveSV$
    $bestMaxSRV \leftarrow moveSRV$
    $bestTotV \leftarrow moveV$
    $bestPart \leftarrow p$
```

Figure 1 shows a sample graph with 8 vertices and 13 edges partitioned into 3 parts. Assume that this is a partial illustration of boundary vertices, and any move will not violate the balance criteria. Each row in the table contains a possible vertex move and the changes on the communication volume metrics. In the initial configuration, $maxSV = 6$, $maxSRV = 9$, and $totV = 12$. If we move v_3 from the partition $\mathcal{V}_2$ to the partition $\mathcal{V}_3$, we reduce all metrics by 1. On the other hand, if we move v_3 to $\mathcal{V}_1$, we decrease $maxSV$ and $maxSRV$, but $totV$ does not change. In this case, since its gain on $totV$ is better, the tie-breaking scheme favors the move v_3 to $\mathcal{V}_3$. Moreover, the moves v_4 to $\mathcal{V}_1$, v_6 to $\mathcal{V}_3$ and v_7 to $\mathcal{V}_3$ are other move examples where tie-breaking scheme is used. Note that we allow all the moves in the first 13 rows of the table including these two. However, we do not allow the ones in the last three rows.

3.4.2. *Implementation details.* During the gain computations, the heuristic uses the connectivity information between nets and parts stored in data structures λ and Λ. These structures are constructed after the initial partitioning phase, and then maintained by the uncoarsening phase. Since the connectivity information changes after each vertex move, when a vertex u is moved, we visit the nets of u and update the data structures accordingly. Also, when new vertices become boundary vertices, they are inserted into boundary array and visited in the same pass.



Vertex	Part	$maxSV$	$maxSRV$	$totV$
v_1	$\mathcal{V}_1$	-1	$+1$	-2
v_1	$\mathcal{V}_2$	-2	-2	-3
v_2	$\mathcal{V}_2$	0	-1	-1
v_2	$\mathcal{V}_3$	-1	$+1$	0
v_3	$\mathcal{V}_1$	-1	-1	0
v_3	$\mathcal{V}_3$	-1	-1	-1
v_4	$\mathcal{V}_1$	-1	-1	0
v_4	$\mathcal{V}_3$	-1	$+1$	$+1$
v_5	$\mathcal{V}_3$	0	0	-1
v_6	$\mathcal{V}_1$	-1	0	$+1$
v_6	$\mathcal{V}_3$	-1	0	0
v_7	$\mathcal{V}_1$	-1	$+1$	0
v_7	$\mathcal{V}_3$	-1	-1	0
v_5	$\mathcal{V}_2$	$+2$	$+2$	-1
v_8	$\mathcal{V}_1$	0	0	0
v_8	$\mathcal{V}_2$	$+2$	$+2$	$+1$

FIGURE 1. A sample partitioning and some potential moves with their effects on the communication volume metrics. The initial values are $maxSV = 6$, $maxSRV = 9$ and $totV = 12$. A negative value in a column indicates a reduction on the corresponding metric.

If at least one move with a positive gain on $maxSV$ is realized in a refinement pass, the heuristic continues with the next pass. Otherwise, it stops. For efficiency purposes, throughout the execution of a pass, we restrict the number of moves for each vertex u. If this number is reached, we lock the vertex and remove it from the boundary. In our experiments, the maximum number of moves per vertex is 4.

Let $\rho = \sum_{n \in \mathcal{N}} |\mathsf{pins}[n]|$ be the number of pins in a hypergraph. The time complexity of a pass of the proposed refinement heuristic is $\mathcal{O}(\rho K + |\mathcal{V}|K^2)$ due to the gain computation loops at lines 3 and 5. To store the numbers of pins per part for each net, Λ, we use a 2-dimensional array. Hence, the space complexity is $\mathcal{O}(K|\mathcal{N}|)$. This can be improved as shown in [**2**].

4. Experimental results

UMPa is tested on a computer with 2.27GHz dual quad-core Intel Xeon CPUs and 48GB main memory. It is implemented in C++ and compiled with g++ version 4.5.2.

To obtain our data set, we used several graphs from the testbed of 10th DIMACS implementation challenge [**10**]. We remove relatively small graphs containing less than 10^4 vertices, and also extremely large ones. There are 123 graphs in our data set from 10 graph classes. For each graph, we execute UMPa and other algorithms 10 times. The results in the tables are the averages of these 10 executions.

To see the effect of UMPa's K-way partitioning structure and its tie-breaking scheme, we compare it with two different refinement approaches and PaToH. The first approach is partitioning the hypergraph into K with PaToH's recursive bisection scheme and refining it by using the proposed K-way refinement algorithm

TABLE 1. The relative performance of UMPa and PaToH+refinement without tie breaking. The performance are computed with respect to that of PaToH.

	PaToH + Refinement			UMPa			UMPa		
	No tie breaking			No tie breaking			With tie breaking		
K	$maxSV$	$maxSRV$	$totV$	$maxSV$	$maxSRV$	$totV$	$maxSV$	$maxSRV$	$totV$
4	0.93	1.05	1.06	0.73	0.83	0.93	0.66	0.77	0.84
16	0.93	1.06	1.04	0.84	0.94	1.11	0.73	0.83	0.98
64	0.91	1.04	1.02	0.86	0.98	1.12	0.76	0.87	1.00
256	0.91	1.03	1.01	0.89	1.00	1.10	0.81	0.91	1.02
Avg.	0.92	1.05	1.03	0.83	0.93	1.06	0.74	0.84	0.96

without employing the tie-breaking scheme. The second approach is using UMPa but again without tie breaking. To remove tie breaking, we remove the **else** statements at lines labeled with 1 and 2 of Algorithm 2.

Table 1 gives the average performance of all these approaches normalized with respect to PaToH's performance. Without tie breaking, refining PaToH's output reduces the maximum send volume by 8%. However, it increases the maximum send-receive and total volumes by 5% and 3%, respectively. Hence, we do not suggest using the refinement heuristic alone and without tie breaking. On the other hand, if it is used in the multi-level structure of UMPa, we obtain better results even without a tie-breaking scheme.

Table 1 shows that UMPa's multi-level structure helps to obtain 17% and 7% less volumes than PaToH's partitions in terms of $maxSV$ and $maxSRV$, respectively. But since PaToH minimizes the total communication volume, there is a 6% overhead on the $totV$. Considering 17% reduction on $maxSV$, this overhead is acceptable. However, we can still reduce all the communication metrics 9%-to-10% more by employing the proposed tie-breaking scheme. For $K = 4$, this leads us a 34% better maximum send volume, which is impressive since even the total communication volume is 16% less compared with PaToH. Actually, for all K values, UMPa manages to reduce $maxSV$ and $maxSRV$ on the average. The percent of improvement reduces with the increasing K. This may be expected since when K is large, the total volume will be distributed into more parts, and the maximum send or send-receive volume will be less. Still, on the average, the reductions on $maxSV$, $maxSRV$, and $totV$ are 26%, 16%, and 4%, respectively.

Tables 2 and 3 show performance of PaToH and UMPa in terms of the communication metrics and time. There are 20 graphs in each table selected from 10 graph class in DIMACS testbed. For each graph class, we select the two (displayed consecutively in the tables) for which UMPa obtains the best and worst improvements on $maxSV$. The numbers given in the tables are averages of 10 different executions. For all experiments with $K = 16$ parts, as Table 2 shows, UMPa obtains a better $maxSV$ value than PaToH on the average. When $K = 4, 64$, and 256, PaToH obtains a better average $maxSV$ only for 16, 4, and 1 graphs, out of 123, respectively.

There are some instances in the tables for which UMPa improves $maxSV$ significantly. For example, for graph ut2010 in Table 2, the $maxSV$ value is reduced from 1506 to 330 with approximately 78% improvement. Furthermore, for the same graph, the improvements on $maxSRV$ and $totV$ are 75% and 67%, respectively.

Table 2. The maximum send and send-receive volumes, and the total volume for PaToH and UMPa when $K = 16$. The times are given in seconds. There are 20 graphs in the table where two graphs with the best and the worst improvements on $maxSV$ are selected from each class. Each number is the average of 10 different executions.

	PaToH				UMPa			
Graph	$maxSV$	$maxSRV$	$totV$	Time	$maxSV$	$maxSRV$	$totV$	Time
coPapersDBLP	62,174	139,600	673,302	91.45	53,619	117,907	842,954	145.47
as-22july06	1,506	5,063	12,956	0.63	1,144	3,986	13,162	2.70
road_central	500	999	3,926	112.64	279	576	2,810	27.85
smallworld	12,043	24,020	188,269	3.09	10,920	21,844	174,645	19.27
delaunay_n14	119	235	1,500	0.19	115	236	1,529	0.88
delaunay_n17	351	706	4,100	1.09	322	655	4,237	2.54
hugetrace-00010	2,113	4,225	25,809	93.99	2,070	4,144	28,572	43.39
hugetric-00020	1,660	3,320	20,479	60.96	1,601	3,202	22,019	29.51
venturiLevel3	1,774	3,548	19,020	27.41	1,640	3,282	20,394	16.01
adaptive	2,483	4,967	27,715	54.00	2,345	4,692	29,444	29.33
rgg_n_2_15_s0	146	293	1,519	0.34	119	254	1,492	1.03
rgg_n_2_21_s0	1,697	3,387	19,627	37.86	1,560	3,215	20,220	16.66
tn2010	2,010	3,666	13,473	1.26	1,684	3,895	56,780	1.54
ut2010	1,506	2,673	3,977	0.43	330	677	1,303	0.82
af_shell9	1,643	3,287	17,306	14.83	1,621	3,242	18,430	8.64
audikw1	15,119	29,280	145,976	161.23	11,900	24,182	159,640	77.16
asia.osm	63	125	409	40.43	30	62	323	7.67
belgium.osm	141	281	1,420	4.80	120.6	243	1,406	1.96
memplus	986	7,138	7,958	0.23	686	3,726	10,082	0.72
t60k	155	310	1,792	0.29	148.5	297	1,890	0.99

When $K = 256$ (Table 3) for the graph memplus, UMPa obtains approximately 50% improvement on $maxSV$ and $maxSRV$. Although $totV$ increases 26% at the same time, this is acceptable considering the improvements on the first two metrics.

Table 4 shows the relative performance of UMPa in terms of execution time with respect to PaToH. As expected, due to the complexity of K-way refinement heuristic, UMPa is slower than PaToH especially when the number of parts is large.

5. Conclusions and future work

We proposed a directed hypergraph model and a multi-level partitioner UMPa for obtaining good partitions in terms of multiple communication metrics where the maximum amount of data sent by a processing unit is the main objective function to be minimized. UMPa uses a novel K-way refinement heuristic employing a tie-breaking scheme to handle multiple communication metrics. We obtain significant improvements on a large number of graphs for all K values.

We are planning to speed up UMPa and the proposed refinement approach by implementing them on modern parallel architectures. We are also planning to investigate partitioning for hierarchical memory systems, such as cluster of multi-socket, multi-core machines with accelerators.

Table 3. The maximum send and send-receive volumes, and the total volume for PaToH and UMPa when $K = 256$. The times are given in seconds. There are 20 graphs in the table where two graphs with the best and the worst improvements on $maxSV$ are selected from each class. Each number is the average of 10 different executions.

	PaToH				UMPa			
Graph	$maxSV$	$maxSRV$	$totV$	Time	$maxSV$	$maxSRV$	$totV$	Time
coPapersCiteseer	7,854	16,765	577,278	224.09	5,448	11,615	579,979	658.21
coPapersDBLP	14,568	34,381	1,410,966	143.97	10,629	23,740	1,371,425	1038.86
as-22july06	1,555	7,128	28,246	1.01	617	4,543	33,347	12.62
smallworld	1,045	2,078	232,255	4.55	877	1,751	208,860	36.24
delaunay_n20	301	600	57,089	17.98	279	566	58,454	68.85
delaunay_n21	420	844	80,603	35.01	398	813	83,234	107.35
hugetrace-00000	407	814	74,563	55.51	415	831	80,176	123.66
hugetric-00010	502	1,004	91,318	92.45	477	955	97,263	167.69
adaptive	753	1,505	143,856	96.60	735	1,472	152,859	224.30
venturiLevel3	568	1,137	107,920	49.97	564	1,132	114,119	132.02
rgg_n_2_22_s0	799	1,589	145,902	151.30	724	1,495	147,331	249.23
rgg_n_2_23_s0	1,232	2,432	219,404	347.32	1,062	2,168	221,454	446.78
ri2010	3206	5,989	281,638	0.72	2,777	5,782	279,941	8.66
tx2010	5,139	9,230	124,033	8.47	3,011	7,534	117,960	15.55
af_shell10	898	1,792	174,624	89.90	885	1,769	184,330	158.04
audikw1	4,318	8,299	680,590	322.57	3,865	7,607	692,714	822.73
asia.osm	72	146	4,535	72.37	66	135	4,484	18.79
great-britain.osm	104	209	11,829	50.52	82	168	11,797	25.51
finan512	199	420	36,023	2.75	192	437	36,827	27.70
memplus	1,860	7,982	15,785	0.49	946	4,318	19,945	8.25

Table 4. The relative performance of UMPa with respect to PaToH in terms of execution time. The numbers are computed by using the results of 10 executions for each of the 123 graphs in our data set.

K	4	16	64	256	Avg.
Relative time	1.02	1.29	2.01	5.76	1.98

References

1. C. J. Alpert and A. B. Kahng, *Recent directions in netlist partitioning: A survey*, VLSI Journal **19** (1995), no. 1–2, 1–81.
2. Cevdet Aykanat, B. Barla Cambazoglu, and Bora Uçar, *Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices*, Journal of Parallel and Distributed Computing **68** (2008), no. 5, 609–625.
3. S. T. Barnhard and H. D. Simon, *Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurrency: Practice and Experience **6** (1994), no. 2, 67–95.
4. R. H. Bisseling and W. Meesen, *Communication balancing in parallel sparse matrix-vector multiplication*, Electronic Transactions on Numerical Analysis **21** (2005), 47–65.
5. T. N. Bui and C. Jones, *A heuristic for reducing fill-in sparse matrix factorization*, Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 445–452.

6. Ü. V. Çatalyürek and C. Aykanat, *A hypergraph model for mapping repeated sparse matrix-vector product computations onto multicomputers*, Proc. International Conference on High Performance Computing, December 1995.
7. ______, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems **10** (1999), no. 7, 673–693.
8. Ü. V. Çatalyürek and C. Aykanat, *Patoh: A multilevel hypergraph partitioning tool, version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at http://bmi.osu.edu/ umit/software.htm, 1999.
9. Ü. V. Çatalyürek, Cevdet Aykanat, and Bora Uçar, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, SIAM Journal on Scientific Computing **32** (2010), no. 2, 656–683.
10. *10th DIMACS implementation challenge: Graph partitioning and graph clustering*, 2011, http://www.cc.gatech.edu/dimacs10/.
11. C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, Proc. 19th Design Automation Conference, 1982, pp. 175–181.
12. Bruce Hendrickson and Tamara G. Kolda, *Graph partitioning models for parallel computing*, Parallel Computing **26** (2000), 1519–1534.
13. Bruce Hendrickson and Robert Leland, *A multilevel algorithm for partitioning graphs*, Proc. Supercomputing (New York, NY, USA), ACM, 1995.
14. G. Karypis, *Multilevel hypergraph partitioning*, Tech. Report 02-25, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, 2002.
15. B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal **49** (1970), no. 2, 291–307.
16. T. Lengauer, *Combinatorial algorithms for integrated circuit layout*, Wiley–Teubner, Chichester, U.K., 1990.
17. Aleksandar Trifunovic and William Knottenbelt, *Parkway 2.0: A parallel multilevel hypergraph partitioning tool*, Proc. ISCIS, LNCS, vol. 3280, Springer Berlin / Heidelberg, 2004, pp. 789–800.
18. Bora Uçar and Cevdet Aykanat, *Minimizing communication cost in fine-grain partitioning of sparse matrices*, Computer and Information Sciences - ISCIS 2003 (A. Yazici and C. Şener, eds.), Lecture Notes in Computer Science, vol. 2869, Springer Berlin / Heidelberg, 2003, pp. 926–933.
19. Bora Uçar and Cevdet Aykanat, *Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies*, SIAM J. Sci. Comput. **25** (2004), 1837–1859.
20. C. Walshaw, M. G. Everett, and M. Cross, *Parallel dynamic graph partitioning for adaptive unstructured meshes*, Journal of Parallel Distributed Computing **47** (1997), 102–108.

Appendix A. DIMACS Challenge Results

Table 5. The best maximum send volumes for UMPa for the challenge instances. X means UMPa failed to obtain a partition with the desired imbalance value.

	Parts									
Graph	2	4	8	16	32	64	128	256	512	1,024
as365						1,080	790	590	421	316
asia.osm						41	46	60	92	93
auto						2,044	1,497	1,070	733	501
coauthorsciteseer		10,066	7,773	5,313	3,216	2,006				
delaunay_n15			189	154	121	90	70			
er-fact1.5-scale23				5,707,503	3,933,216	2,091,986	1,154,276	622,913		
g3_circuit	1,266	1,630			1,151	938		536		
great-britain.osm					134	114	92	78		58
hugebubbles-00010		3,012			1,948	1,522		822	609	
hugetric-00000	1,274	2,206			1,117	804		458		
kkt_power				6,162	6,069	4,508		3,078	2,088	
kron_g500-logn17	36,656	53,381	55,314	49,657	42,272					
kron_g500-logn21						459,454	351,785	245,355	168,870	X
m6	1,487		2,034		1,427		762	568		
nlpkkt160		71,708	55,235	49,700	36,483	25,107				
nlr			2,380		1,563		847	623	447	
rgg_n_2_18_s0			516	431	330	248	195			

The Ohio State University, Dept. of Biomedical Informatics,
Email address: `umit@bmi.osu.edu`

The Ohio State University, Dept. of Biomedical Informatics,
Email address: `mdeveci@bmi.osu.edu`

The Ohio State University, Dept. of Biomedical Informatics,
Email address: `kamer@bmi.osu.edu`

LIP, ENS Lyon, Lyon 69364, France,
Email address: `bora.ucar@ens-lyon.fr`