On Partitioning Problems with Complex Objectives

Kamer Kaya[1], François-Henry Rouet[2], and Bora Uçar[3]

[1] CERFACS, Toulouse, France
Kamer.Kaya@cerfacs.fr
[2] Université de Toulouse, INPT (ENSEEIHT)-IRIT, France
frouet@enseeiht.fr
[3] CNRS and ENS Lyon, France
bora.ucar@ens-lyon.fr

Abstract. Hypergraph and graph partitioning tools are used to partition work for efficient parallelization of many sparse matrix computations. Most of the time, the objective function that is reduced by these tools relates to reducing the communication requirements, and the balancing constraints satisfied by these tools relate to balancing the work or memory requirements. Sometimes, the objective sought for having balance is a complex function of a partition. We mention some important class of parallel sparse matrix computations that have such balance objectives. For these cases, the current state of the art partitioning tools fall short of being adequate. To the best of our knowledge, there is only a single algorithmic framework in the literature to address such balance objectives. We propose another algorithmic framework to tackle complex objectives and experimentally investigate the proposed framework.

Keywords: Hypergraph partitioning, graph partitioning, sparse matrix partitioning, parallel sparse matrix computations.

1 Introduction

Hypergraph and graph partitioning tools are used to partition work for efficient parallelization of many sparse matrix computations. Roughly speaking, the vertices represent the data and the computations, and the (hyper)edges represent dependencies of the computations on the data. For a parallel system of K processors, partitioning the vertices into K disjoint parts can be used to partition the data and the total work among the processors by associating each part with a unique processor. Therefore, a successful application of such partitioning tools should assign almost equal work/data to processors and should reduce the communication costs. The first of these goals is attained by associating weights to vertices and then by guaranteeing that the K resulting parts have almost equal weights, defined as a function of individual vertex weights. The second goal is achieved by reducing a function related to the (hyper)edges that straddle two or more parts. There are a number of widely used tools, including MeTiS [9], Mondriaan [19], PaToH [6], Scotch [14], and Zoltan [5], to achieve these goals.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 334–344, 2012.

Sometimes the objective sought for having balance is simple. By this, we mean that one can assign weights to the vertices before partitioning, and then measure the weight of a part by simply adding up the weights of vertices in that part. For example, if a vertex represents a row of a sparse matrix, then the number of nonzeros in that row can be used as the weight of the corresponding vertex. In a given partition, the weight of a part corresponds to the total number of nonzeros assigned to that part, and hence balance can be obtained among processors easily by using the standard partitioning tools listed above. This standard approach, however, is not sufficient for many important classes of sparse matrix computations. We (in the accompanying technical report [11]) and Pınar and Hendrickson [15] discuss several class of computations (including the FETI class of domain decomposition-based solvers, iterative methods with incomplete LU or Cholesky preconditioners, overlapped Schwarz solvers, and direct methods based on multifrontal solvers) for which the standard approach falls short of achieving balance on the computational load of the processors. In these computations, the objective sought for balancing is a complex function of a partition and cannot be computed by looking at a set of a priori given vertex weights without taking a partition into account—resembling to the chicken-egg problem [15]. We give a toy example for this phenomena. Suppose that we want to partition a square matrix rowwise for efficient parallel computation of $y \leftarrow Ax$ where the input vector x and the output vector y are assigned to the processors conformally with the rows of A, i.e., a processor holding row i of A holds the vector entries y_i and x_i as well. Suppose also that we want to obtain balance on the number of nonzero entries with which the scalar multiply-add operations with the x-vector entries can be computed without communication. The objective is complex, because we cannot know which entries in a row will need an x-vector entry residing in another processor.

We discuss three special forms of sparse matrices in the next subsection. These forms embody most of the data partitioning approaches for efficient parallelization of sparse matrix computations with complex balance requirements. In Section 1.2, we survey the related work on similar problems and highlight our contributions. Section 2 includes the background material. In Section 3, we propose a general framework for the problem of partitioning for complex objectives and adjust it for two of the three matrix forms. For the third form, we just give a short summary and refer the reader to the technical report [11] for more details. A brief experimental evaluation is given in Section 4. We conclude the paper in Section 5 with a summary.

1.1 Problem Definition

Consider the following three forms of an $m \times n$ sparse matrix A for a given integer $K > 1$:

$$A_{SB}=\begin{bmatrix} A_{11} & & A_{1S} \\ & \ddots & \vdots \\ & & A_{KK}A_{KS} \end{bmatrix} \quad (1) \quad A_{BL}=\begin{bmatrix} A_{11}\cdots A_{1K} \\ \vdots \ \ddots \ \vdots \\ A_{K1}\cdots A_{KK} \end{bmatrix} \quad (2) \quad A_{DB}=\begin{bmatrix} A_{11} & & A_{1S} \\ & \ddots & \vdots \\ & & A_{KK}A_{KS} \\ A_{S1}\cdots & & A_{SK}A_{SS} \end{bmatrix} \quad (3)\,.$$

The first form A_{SB} (1) is called the singly bordered block-diagonal form (by convention, we assume a columnwise border throughout the paper). The second one A_{BL} (2) is called the block form. The third one A_{DB} (3) is called the doubly bordered block diagonal form. These three forms have different uses in parallel sparse matrix computations. We assume that they are going to be used to partition the matrices among K processors. The following cases are common (see for example [3,8]). In the forms A_{SB} and A_{BL}, each processor holds a row stripe, i.e., processor k holds, respectively, $[A_{kk} \ \ A_{kS}]$ and $[A_{k1}\cdots A_{kk}\cdots A_{kK}]$. In A_{DB}, a processor holds the arrow-head formed by the blocks A_{kk}, A_{kS} and A_{Sk}, and perhaps parts of or all of A_{SS}.

The accompanying report [11] presents some applications which require a matrix to be put in one of the above three forms with the following complex partitioning requirements. In the A_{SB} form, the size of the border should be small, and the diagonal blocks should have an almost equal number of nonzeros. In the A_{BL} form, the total communication volume (the total number of nonzero off-diagonal column segments) should be small, the diagonal blocks should have an almost equal number of nonzeros, and each row stripe should have an almost equal number of nonzeros. The requirements for the A_{BL} form coincide with those of the toy example mentioned earlier. In the A_{DB} form, the size of the border should be small, the border blocks should have a balanced number of nonzeros, and the diagonal blocks should also have a balanced number of nonzeros.

1.2 Related Work and Contributions

The problem of partitioning for complex objectives was studied before for specific problems [4,13,18] and in a general setting [15] with some specific applications. The algorithmic framework in these studies is very similar and is called the predictor-corrector approach [13]. In the predictor-corrector approach, a partition is obtained by using the standard tools, with the standard (simple) objectives in the predictor step. Then, the partition is evaluated for the complex objectives and refinements to the current partition are performed in the corrector step. Certain methods [4,18] do not go back and forth between different objectives, rather they fix one of them and try to improve the others. The specific approach of [13] and the general framework of [15] apply move based improvement heuristics to improve the partition for all objectives.

There are a few difficulties and challenges that arise in the corrector step. Firstly, in order to efficiently compute and evaluate the complex functions, large two-dimensional data structures are required where one of the dimensions is K (also true for the simple objectives [2,16]). Secondly, efficient mechanisms

that avoid cycles in the move based improvement approaches are hard to design. Furthermore, ties among the gains of moves arise almost always, and effective and efficient tie-breaking mechanisms are hard to design for the K-way refinement scheme (see [1] for those for the recursive bisection based approaches). Therefore, vertices are usually visited in a random order and best moves are performed (see [2,13]). This heuristic, although it can be helpful, can also be very shortsighted.

Direct K-way partitioning methods can handle complex partitioning objectives. This can be accomplished by replacing the standard refinement heuristics with those for the complex objectives. However, this method is akin to the predictor-corrector approach and suffers from the same difficulties.

We propose another approach for partitioning problems with complex objectives. The main idea is to use the recursive bisection based partitioning scheme and to evaluate the complex functions with respect to the existing coarser partition obtained as a result of the preceding bisections. Once the functions are evaluated, some weights can be assigned to the vertices, as the complex functions with respect to the coarser partition are now simple. This allows us to use available tools at each bisection step. The advantages of this framework is that one does not need to write a refinement routine, and the framework is easily applicable to graph and hypergraph models with differing objective functions. We will apply the framework with the standard hypergraph partitioning tools to address the complex partitioning problems for the A_{SB} and A_{BL} forms, and give a summary for that of the A_{DB} form.

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) $\mathcal{N}$. Every net $n_j \in \mathcal{N}$ is a subset of vertices. Weights can be associated with the vertices. We use $w(v_i)$ to denote the weight of the vertex v_i. Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part $\mathcal{V}_k$ is nonempty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. A K-way vertex partition of $\mathcal{H}$ is said to be balanced if $\frac{W_{max}}{W_{avg}} \leq (1 + \varepsilon)$, where $W_{max} = \max_k\{W(\mathcal{V}_k)\}$, $W(\mathcal{V}_k)$ is the weight of the part $\mathcal{V}_k$ defined as the sum of the weights of the vertices in $\mathcal{V}_k$, W_{avg} is the average part weight, and ε represents the allowed imbalance ratio.

In a partition Π of $\mathcal{H}$, a net that has at least one vertex in a part is said to *connect* that part. *Connectivity* λ_j of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* (*external*) if $\lambda_j > 1$, and *uncut* (*internal*) otherwise. The set of external nets of a partition Π is denoted as $\mathcal{N}_E$. The partitioning objective is to minimize the cutsize defined over the cut nets. There are various cutsize definitions. Two relevant definitions are:

$$cutsize(\Pi) = \sum_{n_j \in \mathcal{N}_E} 1 \qquad (4) \qquad cutsize(\Pi) = \sum_{n_j \in \mathcal{N}_E} (\lambda_j - 1) \qquad (5)\ .$$

The NP-complete hypergraph partitioning problem [12] is defined as the task of dividing the vertices of a hypergraph into K parts such that the cutsize is minimized, while the balance criterion given above is met.

A recent variant of the above problem is the multi-constraint formulation [2,7,10,17] in which a set of T weights is associated with each vertex v, i.e., $w(v,1),\ldots,w(v,T)$. Let $W(\mathcal{V}_k,t) = \sum_{v\in\mathcal{V}_k} w(v,t)$ denote the weight of part $\mathcal{V}_k$ for constraint t. Then, a partition Π is said to be balanced if $\forall t \in \{1,\ldots,T\}$ we have $\frac{W_{max}(t)}{W_{avg}(t)} \leq (1+\varepsilon(t))$, where $W_{max}(t) = \max_k\{W(\mathcal{V}_k,t)\}$, $W_{avg}(t) = \sum_{v_i\in\mathcal{V}} w(v_i,t)/K$, and $\varepsilon(t)$ is the allowed imbalance ratio for the constraint t.

Different interpretations and applications of hypergraph partitioning can be used to permute a matrix into the A_{BL}, A_{SB} and A_{DB} forms (see Section 3).

2.2 Recursive Bisection Based Hypergraph Partitioning

We recall some important concepts in the recursive bisection based K-way hypergraph partitioning methods (see also [6]). The number of parts K is assumed to be a power of 2 for the ease of presentation, otherwise this is not a requirement. In this partitioning method, the vertices of a given hypergraph are partitioned into two balanced parts recursively until K parts are obtained. The recursive calls form a tree (called the *bisection tree*) with K leaves. The first bisection, or the root of the bisection tree, corresponds to partitioning the vertices of the original hypergraph into two. The leaf nodes correspond to the parts, and the two parts having the same parent are said to be of the same bisection.

While optimizing the cutsize metric (4), after a bisection step, one discards the cut nets and forms the two hypergraphs with two parts of vertices and the internal nets. This is referred to as *discarding the cut nets* during bisections. On the other hand, while optimizing for the other cutsize metric (5), one *splits the cut nets*. Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be the two vertex partitions obtained at a bisection. Then for any net $n_j \cap \mathcal{V}_1 \neq \emptyset$, one puts a net containing the vertices $n_j \cap \mathcal{V}_1$ in the hypergraph containing vertices $\mathcal{V}_1$, and for any net $n_j \cap \mathcal{V}_2 \neq \emptyset$, one puts a net containing the vertices $n_j \cap \mathcal{V}_2$ in the hypergraph containing vertices $\mathcal{V}_2$.

3 A Framework for Complex Partitioning Objectives

We propose a framework within which standard tools of the hypergraph partitioning problem can be used effectively to address complex partitioning objectives. In this framework, we follow the recursive bisection paradigm. The first bisection is performed as it would be done for a simple objective case. Then, the subsequent recursive bisection steps use the partial (or coarse) partition information to set secondary constraints and use multi-constraint bisection routines. At each bisection step, the two parts will satisfy a balance constraint approximately, as the real balance can only be determined after the bisection. The abstract framework is given in Alg. 1, where concrete instantiations for the complex partitioning problems described in Section 1.1 are elaborated on in the next

subsections. The initial call has the arguments $R = [1, \ldots, m]$, $C = [1, \ldots, n]$, $K = 2^\ell$ for some integer ℓ, $low = 1$, and $up = K$ for an $m \times n$ matrix A.

The advantage of this approach over the predictor-corrector approach is that it enables multi-level refinement (by harnessing such heuristics available in the standard tools) whereas the predictor-corrector approach does not. Writing down a multi-level refinement heuristic for the predictor-corrector approach will indeed be troublesome for the reasons outlined in Section 1.2. On the other hand, when the secondary constraints are not as important as the first one or when different and very loose imbalance ratios are used, predictions might well turn out to be acceptable, and one would not need to reduce the solution space by using multiple constraints.

3.1 The Singly Bordered Form

The off-the-shelf method to permute a matrix into the singly bordered form as shown in (1) is to use the column-net hypergraph model. In this model, an $m \times n$ matrix A is represented with a hypergraph $\mathcal{H} = (\mathcal{R}, \mathcal{C})$, where for each row i of A there is a vertex v_i in $\mathcal{R}$, for each column j of A there is a net n_j in $\mathcal{C}$, and $v_i \in n_j$ iff $a_{ij} \neq 0$. Each vertex v_i is assigned a weight of $|\{j : a_{ij} \neq 0\}|$, i.e., the number of nonzeros in the corresponding row. Then, partitioning this hypergraph into K parts under the objective function (4) can be used to permute the matrix A into the singly bordered form [3, Section 5]. The rows corresponding to the vertices in part k are permuted before the rows corresponding to the vertices in part ℓ for $1 \leq k < \ell \leq K$. This defines a row permutation where the permutation of the rows in a common block is arbitrary. The column permutation is found as follows. The columns corresponding to the nets that are internal to the part k are permuted before those corresponding to the nets internal to the part ℓ for $1 \leq k < \ell \leq K$. Then the coupling columns are permuted to the end. With this approach, one thus reduces the number of coupling columns and obtains balance on the number of nonzeros in the row stripes $[A_{kk} \;\; A_{kS}]$. This does not however imply balance on the diagonal blocks A_{kk}. In [3, Section 5], unit weighted vertices in an unconventional partitioning formulation in which one enforces balance on internal nets is used (the tool is not publicly available). This results in a singly bordered form where the diagonal blocks have a balanced number of rows as well as balanced a number of columns (but balance on the diagonal blocks is not addressed).

Our alternative is to use the outlined recursive bisection based framework to minimize the number of coupling columns while trying to obtain balance on the number of nonzeros in the diagonal blocks as well as in the row stripes. For this purpose, for each row vertex v_i, we associate two weights (after the first bisection) in the third line of Alg. 1:

$$w(v_i, 1) = |\{j : a_{ij} \neq 0\}| \,,$$
$$w(v_i, 2) = |\{j : a_{ij} \neq 0 \text{ and column } j \text{ is not cut yet}\}| \,.$$

Here $w(v_i, 1)$ is the number of nonzeros in row i and kept the same throughout the bisections to have balance in the row stripes $[A_{kk} \;\; A_{kS}]$. On the other hand

Algorithm 1. $RB(A, R, C, K, low, up)$

Input: A: a sparse matrix. R: row indices. C: column indices. K: number of parts. low, up: id of the lowest and highest numbered parts
Output: *partition*: partition information for the rows
1: form the column-net model of the matrix $A(R, C)$
2: **if** this is not the first bisection step **then**
3: use previous bisection information to set up the secondary constraints
4: partition into two $\langle R_1, R_2 \rangle \leftarrow$BisectRows$(A(R, C))$ ▶ with standard tools
5: set $partition(R_1) \leftarrow low$ and set $partition(R_2) \leftarrow up$
6: create the two column sets, using net splitting or net discarding, giving C_1 and C_2
7: $RB(A, R_1, C_1, K/2, low, (low + up - 1)/2)$ ▶ recursive bisection
8: $RB(A, R_2, C_2, K/2, (low + up - 1)/2 + 1, up)$ ▶ recursive bisection

$w(v_i, 2)$ relates to the diagonal block weight, and by changing $w(\cdot, 2)$ at every bisection we make these weights become closer to the exact weight that will be seen at the end. Although each bisection step obtains two parts with a balanced $W(\cdot, 2)$, two parts from two different bisections are related only indirectly. The coupling columns are discarded at the sixth line of the framework, as we are interested in the cut-net metric (4).

3.2 The Block Form

The off-the-shelf method for this problem is to use the column-net hypergraph model with the objective function (5). The row permutation is done as in the previous section. The column permutation is determined in a post-process [4,18]. A straightforward post-process would be to first permute the internal columns as is done in the previous section and then to permute a coupling column j to the block which has the minimum number of nonzeros in the diagonal (so far) among those blocks that the coupling column j touches.

The proposed recursive bisection based framework can be used as follows. As the objective function is (5), we use the net splitting methodology. While doing so, we keep the copy with the higher number of nonzeros as the main copy (uncut nets are already main copies) with an intent to assign the associated columns to one of the parts that will be resulting from the recursive calls on the part of the main copy. That is, the net splitting operation marks either the copy in C_1 or the copy in C_2 of a cut net as the main one (assuming the cut net was a main copy). Consider a split net n_j whose main copy is put in C_1. Then in the following bisection $RB(A, R_1, C_1, \ldots)$, the vertex $i \in R_1$ for which $a_{ij} \neq 0$ will bear a weight of one for the split net n_j (that nonzero entry is in the diagonal block), whereas no vertex in R_2 will bear a weight for the same net. Formally, we propose assigning the following weights to the vertices:

$$w(v_i, 1) = |\{j : a_{ij} \neq 0\}| \,,$$
$$w(v_i, 2) = |\{j : a_{ij} \neq 0 \text{ and column } j \text{ is a main copy}\}| \,.$$

As before, keeping $w(\cdot, 1)$ always equal to the number of nonzeros in the corresponding row results in balance in the row stripes $[A_{k1}, \ldots, A_{kK}]$, whereas $w(\cdot, 2)$ will approximate the number of nonzeros in the diagonal blocks. Therefore, the last level bisections will be almost accurate. Again the weights of two distant parts will be loosely related.

3.3 The Doubly Bordered Form

We have instantiated the framework for the problem of permuting a sparse symmetric matrix into A_{DB} form with the complex objectives stated in Section 1.1. The details are in the accompanying technical report [11]. We give a short summary of what is achieved by the framework. The standard tools (based on graph partitioning methods) are demonstrated to be susceptible to drastic imbalances (up to 9, on diagonal blocks, and 32, on the border blocks, fold imbalance were reported in a 128-way partitioning of a matrix). On the other hand, the framework with some proper definition of vertex weights was able to improve the balance on the diagonal blocks always (the maximum was about 183%) and the balance on the border blocks by about (the maximum was about 137%). This comes however with an increase of about 35% in the border size on average.

In a recent study [20], the framework is adapted to attain some other complex partitioning objectives for the A_{DB} form (in which the border size should be small and linear system solves with the diagonal blocks and border blocks should be balanced). In that study, it has been demonstrated in practical experiments that the improved load balance can result in reduced execution time in spite of the increased border size.

4 Experiments

We have used three rectangular, nine square and pattern unsymmetric, and 23 pattern symmetric matrices from University of Florida sparse matrix collection (`www.cise.ufl.edu/research/sparse/matrices/`). The names and the properties of these matrices can be seen in [11]. We have partitioned the matrices into $K = \{32, 64, 128\}$ parts using the hypergraph partitioning tool PaToH [6]. As PaToH includes randomized algorithms, we run each experiment 10 times and report the average result. Below, we give a summary of results and refer the reader to [11] for detailed results, including those for the A_{DB} form.

4.1 The Singly Bordered Form

We compare the framework with the standard method (SM) of partitioning the column-net hypergraph model on all matrices in the data set. Both of the approaches obtained good balance on the number of nonzeros per row stripe (both are less than 0.04 on average). In 60 instances (among $35 \times 3 = 105$ partitioning instances), both of the methods obtained balance on the number of nonzeros in the diagonal blocks within 10% of the perfect balance—those

instances are discarded. We normalized the cutsize and the imbalance obtained by the framework to those obtained by SM on the remaining instances. Some statistical indicators, the minimum, the median, the maximum, the average and the geometric mean (gmean), of these results are given in Table 1.

Table 1. Statistical indicators of the ratio of the results of the framework to the results of SM. "Cutsize" refers to the number of coupling columns and "Imbal(A_{kk})" refers to the imbalance on the number of nonzeros on the diagonal blocks.

	min	median	max	avg	gmean
Cutsize	0.78	1.08	1.74	1.11	1.10
Imbal(A_{kk})	0.36	0.64	1.57	0.70	0.67

As seen in Table 1, the framework obtains 30% better balance on the number of nonzeros in the diagonal blocks, on average. This comes with an increase of about 11% on the number of coupling columns. This degradation in the cutsize is expected as the standard method has only one constraint. Previously, the average increase in the cutsize with the metric (4) is reported to be around 34% (compare tables 2 and 5 in [2]) in the two-constraint case. We think therefore that the increase of 11% in the cutsize is well spent to reduce the imbalance of the diagonal blocks by 30%.

4.2 The Block Form

We compare the framework with the standard method (SM) of partitioning the column-net hypergraph model with the objective of minimizing the cutsize given in (5) on all square matrices of the data set. In all of these experiments, both of the approaches obtained balance on the number of nonzeros per row stripes quite satisfactorily (both are less than 0.06 on average). In 85 partitioning instances, both of the methods obtained balance on the number of nonzeros in the diagonal blocks within 10% of the perfect balance—those instances are discarded. We normalized the cutsize and the imbalance obtained by the framework to those obtained by SM on the remaining instances. Some statistical indicators, the minimum, the median, the maximum, the average and the geometric mean (gmean), of these results are given in Table 2.

Table 2. Statistical indicators of the ratio of the results of the framework to the results of SM. "Cutsize" refers to the total volume of communication and "Imbal(A_{kk})" refers to the imbalance on the number of nonzeros on the diagonal blocks.

	min	median	max	avg	gmean
Cutsize	1.00	1.10	1.53	1.15	1.16
Imbal(A_{kk})	0.35	0.78	2.00	0.84	0.75

As in the singly bordered case, we were expecting an increase in the cutsize. Compared to again previously reported results [2], 15% increase in the cutsize is acceptable. This resulted in 16% improvement in the balance on the number of nonzeros on the diagonal blocks. The small geometric mean indicates a few outliers with a large value. We have looked at the results closely and spotted only a few such cases (on one matrix the standard method and the framework obtained, respectively 0.08 and 0.10 imbalance, implying 25% better balance in favor of the standard method). Upon discarding those, the framework resulted in about 20% better balance, on average.

5 Conclusion

We have discussed three sparse matrix forms (the block form, the singly bordered block diagonal form, and the doubly bordered block diagonal form). These forms exemplify a broad range of sparse matrix computations whose efficient parallelization need some complex partitioning objectives to be attained. We have presented a framework to address such kinds of complex partitioning objectives, and evaluated the framework within hypergraph partitioning methods. We presented results for the singly bordered and block forms and reported results from [11] and another current study [20] in which the framework of the current paper was adapted to meet some objectives in the doubly bordered block diagonal form. In all cases, the framework is demonstrated to be able to trade an increase in the objective function with better load balance, and it improved running times in practical experiments with the doubly bordered form.

In general, the proposed framework is more effective with the increasing number of parts. The case $K = 2$, in particular, is not addressed at all.

References

1. Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning: A survey. Integration, the VLSI Journal 19, 1–81 (1995)
2. Aykanat, C., Cambazoglu, B.B., Uçar, B.: Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. J. Parallel Distr. Com 68, 609–625 (2008)
3. Aykanat, C., Pınar, A., Çatalyürek, Ü.V.: Permuting sparse rectangular matrices into block-diagonal form. SIAM J. Sci. Comput. 25, 1860–1879 (2004)
4. Bisseling, R.H., Meesen, W.: Communication balancing in parallel sparse matrix-vector multiplication. ETNA 21, 47–65 (2005)
5. Boman, E., Devine, K., Fisk, L.A., Heaphy, R., Hendrickson, B., Vaughan, C., Catalyurek, U., Bozdag, D., Mitchell, W., Teresco, J.: Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User's Guide. Sandia National Laboratories, Albuquerque, NM (2007)
6. Çatalyürek, Ü.V., Aykanat, C.: PaToH: A multilevel hypergraph partitioning tool, ver. 3.0. Tech. Rep. BU-CE-9915, Bilkent Univ., Dept. Computer Eng. (1999)
7. Çatalyürek, Ü.V., Aykanat, C., Uçar, B.: On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. SIAM J. Sci. Comput. 32, 656–683 (2010)

8. Hendrickson, B., Kolda, T.G.: Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. SIAM J. Sci. Comput. 21, 2048–2072 (2000)
9. Karypis, G., Kumar, V.: MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. Univ. Minnesota, Dept. Comp. Sci. Eng. (1998)
10. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. Tech. Rep. 98-019, Univ. Minnesota, Dept. Comp. Sci. Eng. (1998)
11. Kaya, K., Rouet, F.H., Uçar, B.: On partitioning problems with complex objectives. Tech. Rep. RR-7546, INRIA, France (2011)
12. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. Wiley–Teubner, Chichester (1990)
13. Moulitsas, I., Karypis, G.: Partitioning algorithms for simultaneously balancing iterative and direct methods. Tech. Rep. 04-014, Univ. Minnesota, Dept. Comp. Sci. Eng. (2004)
14. Pellegrini, F.: SCOTCH 5.1 User's Guide. Laboratoire Bordelais de Recherche en Informatique (LaBRI) (2008)
15. Pınar, A., Hendrickson, B.: Partitioning for complex objectives. In: IPDPS 2001, CDROM, p. 121. IEEE Computer Society, Washington, DC (2001)
16. Sanchis, L.A.: Multiple-way network partitioning with different cost functions. IEEE T. Comput. 42, 1500–1504 (1993)
17. Schloegel, K., Karypis, G., Kumar, V.: Parallel Multilevel Algorithms for Multiconstraint Graph Partitioning. In: Bode, A., Ludwig, T., Karl, W.C., Wismüller, R. (eds.) Euro-Par 2000. LNCS, vol. 1900, pp. 296–310. Springer, Heidelberg (2000)
18. Uçar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM J. Sci. Comput. 25, 1827–1859 (2004)
19. Vastenhouw, B., Bisseling, R.H.: A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Rev. 47, 67–95 (2005)
20. Yamazaki, I., Li, X.S., Rouet, F.H., Uçar, B.: Combinatorial problems in a parallel hybrid linear solver. In: Becker, M., Lotz, J., Mosenkis, V., Naumann, U. (eds.) Abstracts of 5th SIAM Workshop on Combinatorial Scientific Computing. pp. 87–89. RWTH Aachen University (2011)