Spatiotemporal Graph and Hypergraph Partitioning Models for Sparse Matrix-Vector Multiplication on Many-Core Architectures

Nabil Abubaker, Kadir Akbudak, and Cevdet Aykanat

Abstract—There exist graph/hypergraph partitioning-based row/column reordering methods for encoding either spatial or temporal locality separately for sparse matrix-vector multiplication (SpMV) operations. Spatial and temporal hypergraph models in these methods are extended to encapsulate both spatial and temporal localities based on cut/uncut net categorization obtained from vertex partitioning. These extensions of spatial and temporal hypergraph models encode the spatial locality primarily and the temporal locality secondarily, and vice-versa, respectively. However, the literature lacks models that simultaneously encode both spatial and temporal localities utilizing only vertex partitioning for further improving the performance of SpMV on shared-memory architectures. In order to fill this gap, we propose a novel spatiotemporal hypergraph model that leads to a one-phase spatiotemporal reordering method which encodes both types of locality simultaneously. We also propose a framework for spatiotemporal methods which encodes both types of locality in two dependent phases and two separate phases. The validity of the proposed spatiotemporal models and methods are tested on a wide range of sparse matrices and the experiments are performed on both a 60-core Intel Xeon Phi processor and a Xeon processor. Results show the validity of the methods via almost doubling the Gflop/s performance through enhancing data locality in parallel SpMV operations.

Index Terms—Sparse matrix, sparse matrix-vector multiplication, data locality, spatial locality, temporal locality, hypergraph model, bipartite graph model, graph model, hypergraph partitioning, graph partitioning, Intel Many Integrated Core Architecture, Intel Xeon Phi.

1 Introduction

Sparse matrix-vector multiplication (SpMV) is a building-block for many applications. In this work, we focus on repeated SpMV operation of the form $y = Ax$, where the sparsity pattern of matrix A does not change. Thread-level parallelization of SpMV on today's many-core cache-coherent architectures highly necessitates utilizing both spatial locality and temporal locality in order to efficiently use the cache hierarchy. Here, spatial locality refers to the use of data elements within relatively close storage locations. That is, if a particular storage location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future. Temporal locality refers to the reuse of specific data within a relatively small time duration. That is, if at one point a particular memory location is referenced, then it is likely that the same location will be referenced in the near future. In terms of cache hierarchy, per-core cache sizes of today's processors vary from tens of kilobytes [1] to several megabytes. In this work, we focus on reordering-based methods for accelerating SpMV for any kind of cache hierarchy with any capacity.

1.1 Data Locality in Parallel SpMV

Here, we present data locality issues in SpMV. For the sake of clarity of the presentation, we assume that one or more rows are processed at a time and some kind of compression is used for indexing the nonzeros in such a way that indirect accesses are performed on x-vector entries. In other words, we assume that the sparse matrix is stored and processed using the CRS (Compressed Row Storage) scheme.

Temporal locality is not feasible in accessing nonzeros of the input matrix, because these nonzeros together with their indices are accessed only once, whereas spatial locality is already achieved because the nonzeros are stored and accessed consecutively.

Temporal locality in accessing y-vector entries is achieved on the highest level of memory, because partial results for the same y-vector entries are summed consecutively since nonzeros are accessed rowwise. Spatial locality is already achieved because y-vector entries are accessed consecutively.

Temporal locality is feasible in accessing x-vector entries, because these entries are accessed multiple times while processing nonzeros in different rows. Spatial locality is also feasible, because these entries are accessed irregularly depending on the index arrays. These two types of localities constitute the main bottleneck of rowwise SpMV.

Regarding the above-mentioned data locality characteristics, the possibility of exploiting spatial locality in accessing x-vector entries is increased by ordering columns with similar sparsity patterns nearby. The possibility of exploiting temporal locality in accessing x-vector entries is increased by ordering rows with similar sparsity pattern nearby. Simultaneously reordering the columns and rows with similar sparsity patterns nearby increases the possibility of exploiting both spatial and temporal localities in accessing x-vector entries.

• *Nabil Abubaker and Cevdet Aykanat are with the Department of Computer Engineering, Bilkent University, Ankara, Turkey. Kadir Akbudak is with Department of Applied Mathematics and Computational Science, Extreme Computing Research Center, KAUST, KSA*
E-mail: nabil.abubaker@bilkent.edu.tr, kadir.akbudak@kaust.edu.sa, aykanat@cs.bilkent.edu.tr. 27/07/2018 17:03:57

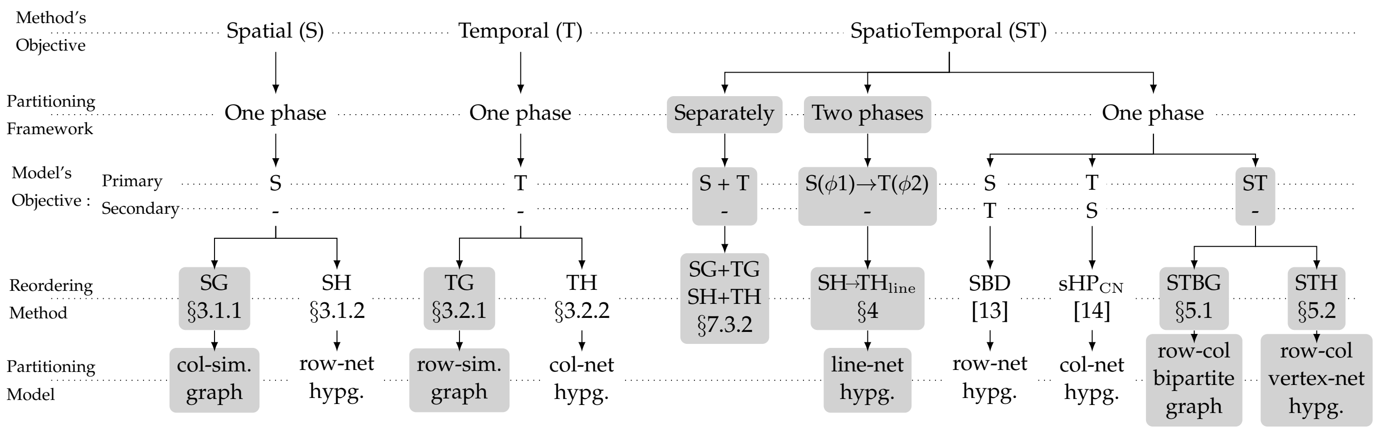


Fig. 1: A taxonomy for reordering methods used to exploit data locality in SpMV operations. Shaded leaves denote proposed methods.

1.2 Contributions

In this work, we mainly focus on reordering models that exploit spatiotemporal locality for parallel SpMV operations on many-core architectures. Here, spatiotemporal locality refers to exploiting both spatial and temporal localities. We present Fig. 1 which contains a taxonomy of reordering methods for exploiting spatial, temporal and spatiotemporal localities in CRS-based SpMV. We should note here that the fine-grain hypergraph models are out of scope of this paper due to their high partitioning overheads [14] so the existing works [14], [23] are not included in this taxonomy. Table 1 shows the list of notations and abbreviations used in this figure, as well as throughout the paper.

In Section 3, we discuss methods that aim at exploiting only one type of locality. We summarize the existing Spatial Hypergraph (SH) and Temporal Hypergraph (TH) methods. Moreover, we propose to use similarity graphs in graph partitioning (GP) based methods to exploit spatial and temporal localities separately, which are referred to here as Spatial Graph (SG) and Temporal Graph (TG), respectively. To our knowledge, similarity graph models are not used for improving SpMV performance through reordering columns/rows of a sparse matrix, although various similarity graph models have been proposed in different contexts such as sparse matrix-matrix multiplication [2], declustering for distributed databases [3] and parallel text retrieval [4].

In Section 4, we propose a new two-phase framework for exploiting both spatial and temporal localities. In the first phase, we use a column reordering method for encoding spatial locality in order to find an assignment of x-vector entries into lines (blocks). In the second phase, we use a row reordering method for encoding temporal locality among the lines identified in the first phase in order to exploit locality on the access of lines instead of single words. According to this framework, we propose and implement the Temporal Hypergraph on lines of data entries (SH→TH$_{\text{line}}$ or TH$_{\text{line}}$ shortly) method that uses the SH method in the first phase and uses the TH method in the second phase.

In Section 5, we propose two new one-phase methods that simultaneously encode spatial and temporal localities in accessing x-vector entries. The first method is based on bipartite graph partitioning and will be referred to as SpatioTemporal Bipartite Graph (STBG). The second method is based on hypergraph partitioning (HP) and will be referred to as SpatioTemporal Hypergraph (STH).

In Sections 3 and 5, after presenting each method, we also provide a brief insight on how the method works so that the partitioning objective of minimizing the cutsize in the graph and hypergraph models relate to reducing the number of x-vector entries that are not reused due to loss of spatial and/or temporal localities.

In order to empirically verify the validity of the proposed methods, we use Sparse Library (SL) [5], [6], [7], which is highly optimized for performing SpMV on shared-memory architectures. We conduct experiments on both 60-core Xeon Phi and Xeon processors using a wide-range of sparse matrices arising in different applications. The results given in Section 7 show that the proposed spatiotemporal methods that aim at simultaneously exploiting both types of localities substantially perform better than the non-simultaneous methods. The results also show the superiority of the HP-based methods over the GP-based methods.

TABLE 1: List of notations and abbreviations

Concept	Meaning
$\mathcal{G}$	Graph model
$\mathcal{H}$	Hypergraph model
S*	Spatial; used with graph or hypergraph models
T*	Temporal; used with graph or hypergraph models
$\mathcal{V}$	Set of Vertices
$\mathcal{E}$	Set of Edges (for a graph model)
$\mathcal{N}$	Set of Nets (for a hypergraph model)
d*	Data; used with a vertex or a net, or with sets $\mathcal{V}$ and $\mathcal{N}$
t*	Task; used with a vertex or a net, or with sets $\mathcal{V}$ and $\mathcal{N}$
GP	Graph Partitioning
HP	Hypergraph Partitioning
SG	Spatial Graph
TG	Temporal Graph
SH	Spatial Hypergraph
TH	Temporal Hypergraph
SH→TH$_{\text{line}}$ (TH$_{\text{line}}$)	Temporal Hypergraph on Lines (blocks) of data entries
STBG	SpatioTemporal Bipartite Graph
STH	SpatioTemporal Hypergraph

* *used as a superscript*

2 Preliminaries

Graph and hypergraph partitioning approaches have been used in the literature to attain row/column reordering for exploiting spatial and/or temporal locality in the

SpMV operation [13], [14], [27], [28]. Here, we briefly explain how the methods presented in this work utilize the graph/hypergraph partitioning framework to obtain row/column reordering for SpMV on many-core architectures.

A K-way partition $\Pi(\mathcal{V}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ on vertices of the graph and hypergraph models presented in this paper is decoded as a partial reordering on the corresponding matrix dimension(s) as described as follows. For $k = 1, 2, \ldots, K-1$:

(i) In spatial methods, the columns corresponding to the vertices in $\mathcal{V}_{k+1}$ are reordered after the columns corresponding to the vertices in $\mathcal{V}_k$.
(ii) In temporal methods, the rows corresponding to the vertices in $\mathcal{V}_{k+1}$ are reordered after the rows corresponding to the vertices in $\mathcal{V}_k$.
(iii) In spatiotemporal methods, both row and column reorderings described in (i) and (ii) are applied.

The row/column orderings obtained by these methods are referred to as partial orderings because the rows/columns corresponding to vertices in a part are reordered arbitrarily. On the other hand, by keeping the number of vertices per part sufficiently small, the rows/columns corresponding to the vertices in a part are considered to be reordered nearby.

In a given partition of a graph, an edge $e_{i,j}$ that connects a pair of vertices in two different parts is said to be *cut*, and *uncut* otherwise. In a given partition of a hypergraph, a net n_i that connects vertices in more than one part is said to be *cut*, and *uncut* otherwise. In graph and hypergraph models, the relevant cutsize definitions are as follows respectively:

$$\text{Graph}: \quad \mathit{edgecut}(\Pi(\mathcal{V})) = \sum_{v_i \in \mathcal{V}_k \wedge v_j \in \mathcal{V}_{\ell \neq k}} w(e_{i,j}), \tag{1}$$

$$\text{Hypergraph}: \quad \mathit{cutsize}(\Pi(\mathcal{V})) = \sum_{n_i \in \mathcal{N}} w(n_i)|\mathit{con}(n_i)|. \tag{2}$$

In (2), $\mathit{con}(n_i)$ denotes the connectivity set of n_i, that is, the set of parts that have at least one vertex connected by n_i. In (1) and (2), $w(e_{i,j})$ and $w(n_i)$ denote the weight of edge $e_{i,j}$ and net n_i, respectively.

In Fig. 2, we present a sample 6-by-8 sparse matrix for the purpose of illustrating the main features of the graph and hypergraph models. As seen in the figure, letters a, b, c,... are used to denote columns of matrix A, whereas numbers 1, 2, 3,... are used to denote rows of matrix A. We also include the x-vector entries to better show the correspondence between them and the columns of the matrix when discussing column reordering methods.

Algorithm 1 The thread-level rowwise parallelization of SpMV operation based on CRS

Require: A matrix stored in $irow$, $icol$ and val arrays, x and y vectors.

```
1: //Each iteration performs $< r_i, x >$ as a task
2: for $i = 1$ to # of rows of $A$ in parallel do
3:     $sum = 0.0$
4:     for $k = irow[i]$ to $irow[i+1] - 1$ do
5:         $sum = sum + val[k]\ x[icol[k]]$
6:     $y[i] = sum$
```

$$\begin{array}{c|cccccccc|}
 & x_a & x_b & x_c & x_d & x_e & x_f & x_g & x_h \\
 & a & b & c & d & e & f & g & h \\
\hline
1 & \times & & & \times & & & & \\
2 & \times & & \times & & & & & \\
3 & & \times & & & & & & \times \\
4 & & & \times & & & & \times & \\
5 & & & & & \times & \times & \times & \\
6 & & & & & \times & \times & & \\
\hline
\end{array}$$

Fig. 2: Sample matrix A.

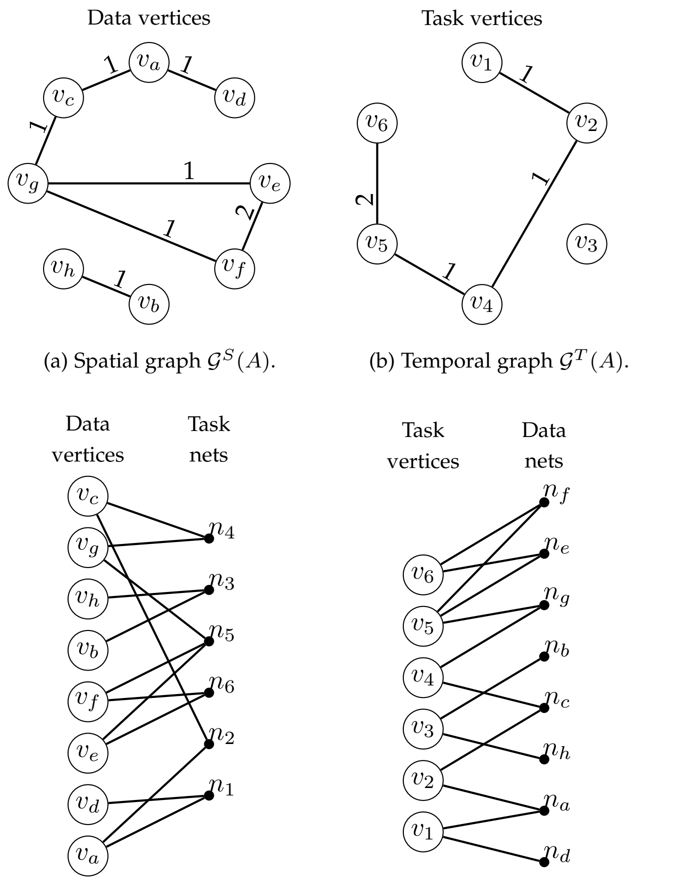
Algorithm 1 is presented to better relate the entities in the proposed graph-theoretic models with the data accesses and computations in the CRS-based SpMV operation. For the sake of clarity of presentation, the following assumptions are used in Algorithm 1 and in the insights given to show the correctness of the methods: The memory alignment of A, x and y arrays are ignored. Fully associative cache is assumed since the scope of this work is to decrease only capacity misses due to accessing x-vector entries.

In Algorithm 1, the inner-product tasks of the SpMV operation are represented by the iterations of the outer *for*-loop (lines 2-6), whereas the irregular accesses to the x-vector entries are represented by the indirect indexing in line 5. In the spatial methods presented in this work, the vertices of the graph or hypergraph models represent x-vector entries. These methods aim at reducing the latency due to irregular accesses to the x-vector entries by reordering the different x-vector entries used by the same and/or consecutive inner-products (iterations) nearby. In the temporal methods presented in this work, the vertices of the graph or hypergraph model represent inner-product tasks. These methods aim at reusing the x-vector entries brought to the cache during the previous iteration by reordering the inner-products so that those using similar x-vector entries are executed consecutively. In the proposed spatiotemporal methods, both graph and hypergraph models contain vertices that represent inner-product tasks and x-vector entries separately. So these methods combine the aims of the spatial and temporal methods.

The methods presented in this work produce reorderings for the rows and/or columns of matrix A as well as the x- and/or y-vector entries involved in the $y = Ax$ operation given in Algorithm 1. The spatial methods presented in Section 3.1 produce conformal reorderings for the columns of A and the entries of the x vector. The temporal methods presented in Section 3.2 produce conformal reorderings for the rows of A and the entries of the output y vector. The spatiotemporal methods presented in Sections 4, 5 and 7.3.2 produce reorderings for both rows and columns of A which respectively induce conformal reorderings for the entries of the y and x vectors.

3 Exploiting Spatial and Temporal Localities Separately

In this section, we discuss graph and hypergraph models that encode either spatial or temporal locality in accessing x-vector entries.



(a) Spatial graph $\mathcal{G}^S(A)$. (b) Temporal graph $\mathcal{G}^T(A)$.

(c) Spatial hypergraph $\mathcal{H}^S(A)$. (d) Temporal hypergraph $\mathcal{H}^T(A)$.

Fig. 3: Graph and hypergraph models (of the sample matrix given in Fig. 2) for exploiting spatial and temporal localities separately.

3.1 Spatial Locality

Here, we present two reordering methods based on GP and HP that encode spatial locality.

3.1.1 *Similarity Graph Model $\mathcal{G}^S$ (SG Model)*

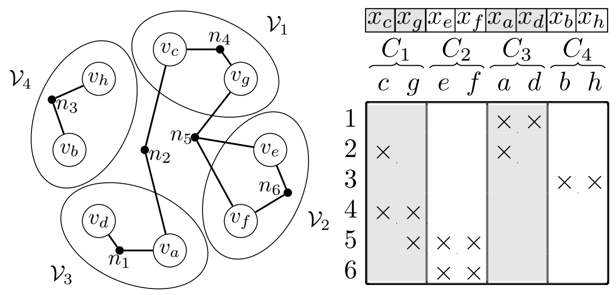
For a given matrix $A=(a_{i,j})$, the similarities between the use of x-vector entries by the inner product tasks are represented–in terms of the number of shared rows between columns–as a similarity graph $\mathcal{G}^S(A) = (\mathcal{V}^d, \mathcal{E})$. A row is said to be shared between two columns if both of these two columns have a nonzero on that row. Here, calligraphic letters are used to denote sets, e.g., $\mathcal{V}$ and $\mathcal{E}$ denote the sets of vertices and edges, respectively.

In $\mathcal{G}^S(A)$, there exists a data vertex $v_i^d \in \mathcal{V}^d$ for each column c_i of A. There exists an edge $e_{i,j} \in \mathcal{E}$ if columns c_i and c_j share at least one row. We associate vertices with unit weights. We associate an edge $e_{i,j}$ with a weight $w(e_{i,j})$ equal to the number of shared rows between columns c_i and c_j. That is,

$$w(e_{i,j}) = |\{h : a_{h,i} \neq 0 \wedge a_{h,j} \neq 0\}|.$$

An edge $e_{i,j}$ with weight $w(e_{i,j})$ means that x_i and x_j will be accessed together during each of the $w(e_{i,j})$ inner-products of rows shared between columns c_i and c_j.

Fig. 3a shows the SG model $\mathcal{G}^S$ of the sample matrix given in Fig. 2. As seen in the figure, there are 8 data vertices and 7 edges. The edge $e_{e,f}$ has a weight $w(e_{e,f}) = 2$ because columns c_e and c_f share both rows r_5 and r_6.



(a) 4-way partitioning of $\mathcal{H}^S(A)$. (b) Reordered matrix $\hat{A}$.

Fig. 4: Four-way partition $\Pi(\mathcal{V}^d)$ of the spatial hypergraph given in Fig. 3c and matrix $\hat{A}$, which is obtained via reordering matrix A according to this partition.

A brief insight on how the method works can be given as follows: Assume that a cache line contains L words and each part in $\Pi(\mathcal{V}^d)$ of $\mathcal{G}^S$ contains exactly L vertices. Since the columns corresponding to the L words in a part are reordered consecutively, the corresponding x-vector entries are located consecutively, possibly on the same line. So we can assume that an uncut edge $e_{i,j}$ corresponds to assigning x_i and x_j to the same line, whereas a cut edge $e_{i,j}$ induces the allocation of x_i and x_j to two different lines.

Consider an uncut edge $e_{h,\ell}$. During each of the $w(e_{h,\ell})$ inner-products of rows shared between columns c_h and c_ℓ, the line that contains both x_h and x_ℓ will be reused. Consider a cut edge $e_{i,j}$. During each of the $w(e_{i,j})$ inner-products of rows shared between columns c_i and c_j, the line that contains x_i or the line that contains x_j may not be reused. So, a cut edge $e_{i,j}$ will incur at most $w(e_{i,j})$ extra cache misses due to loss of spatial locality. Thus, minimizing the edgecut according to (1) relates to reducing the number of cache misses due to loss of spatial locality.

3.1.2 *Hypergraph Model $\mathcal{H}^S$ (SH Model)*

For a given matrix A, the requirement of x-vector entries by the inner-product tasks are represented as a hypergraph $\mathcal{H}^S(A) = (\mathcal{V}^d, \mathcal{N}^t)$. For each column c_j of A, there exists a data vertex $v_j^d \in \mathcal{V}^d$. For each row r_i of A, there exists a task net $n_i^t \in \mathcal{N}^t$. Net n_i^t connects vertices corresponding to the columns that share row r_i, that is,

$$Pins(n_i^t) = \{v_j^d : a_{i,j} \neq 0\}. \tag{3}$$

Net n_i^t encodes the fact that $|Pins(n_i^t)|$ x-vector entries corresponding to the vertices in $Pins(n_i^t)$ will be accessed together during the inner-product $< r_i, x >$. We associate vertices and nets with unit weights. $\mathcal{H}^S$ (A) is topologically equivalent to the row-net model [8] of matrix A.

Fig. 3c shows the $\mathcal{H}^S$ model of the sample matrix given in Fig. 2. In the figure, the task net n_5^t connects data vertices v_e^d, v_f^d and v_g^d because the inner product task associated with row r_5 requires the x-vector entries x_e, x_f and x_g.

A brief insight on how the method works can be given as follows: Assume that a cache line contains L words and each part in $\Pi(\mathcal{V}^d)$ of $\mathcal{H}^S$ contains exactly L vertices. Since the columns corresponding to the L words in a part are reordered consecutively, the corresponding x-vector entries are located consecutively in the memory, possibly on the same line. A net n_i^t with connectivity set $con(n_i^t)$ means that inner-product $< r_i, x >$ requires the x-vector entries corresponding to the columns represented by the vertices in $con(n_i^t)$. These x-vector entries are stored in $|con(n_i)|$ different lines. Hence, at most $|con(n_i^t)|$ cache misses occur during the inner product $<r_i, x>$. So, minimizing the cutsize according to (2) corresponds to minimizing the number of cache misses due to loss of spatial locality.

Fig. 4 shows a four-way partition $\Pi(\mathcal{V}^d)$ of $\mathcal{H}^S(A)$ model of the sample matrix given in Fig. 2 and the permuted matrix $\hat{A}$. The row order of $\hat{A}$ is the same as that of A. The partial column order of $\hat{A}$ is obtained from $\Pi(\mathcal{V}^d)$ as described in the beginning of Section 2. Hence, in Fig. 4a, column sets $C_1, \ldots, C_4$ are obtained via decoding the respective parts $\mathcal{V}_1, \ldots, \mathcal{V}_4$ in Fig. 4b and the x-vector entries are reordered accordingly.

3.2 Temporal Locality

Here, we present two reordering methods based on GP and HP that encode temporal locality.

3.2.1 Similarity Graph Model $\mathcal{G}^T$ (TG Model)

For a given matrix A, the similarities between inner product tasks associated with each row are represented–in terms of the number of shared columns–as a similarity graph $\mathcal{G}^T(A) = (\mathcal{V}^t, \mathcal{E})$. In $\mathcal{G}^T(A)$, there exists a task vertex $v_i^t \in \mathcal{V}^t$ for each row r_i of A. There exists an edge $e_{i,j} \in \mathcal{E}$ if rows r_i and r_j share at least one column. We associate vertices with unit weights. We associate an edge $e_{i,j}$ with a weight $w(e_{i,j})$ equal to the number of shared columns between rows r_i and r_j. That is,

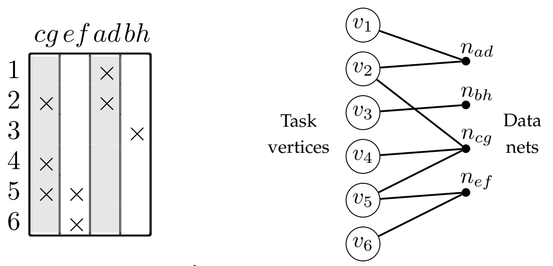
$$w(e_{i,j}) = |\{h : a_{i,h} \neq 0 \wedge a_{j,h} \neq 0\}|.$$

An edge $e_{i,j}$ with weight $w(e_{i,j})$ means that $w(e_{i,j})$ x-vector entries corresponding to the columns shared between rows r_i and r_j will be accessed during each of the inner products $<r_i, x>$ and $<r_j, x>$.

Fig. 3b shows the $\mathcal{G}^T$ model of the sample matrix given in Fig. 2. As seen in the figure, there are 6 task vertices and 4 edges. Edge $e_{5,6}$ has a weight $w(e_{5,6}) = 2$ because the rows r_5 and r_6 have nonzeros in the common columns c_e and c_f.

A brief insight on how the method works can be given as follows under the assumption that each line stores one word: Since the rows corresponding to the vertices in a part are reordered consecutively, the respective inner product operations can reuse the required x-vector entries. So we can assume that an uncut edge $e_{i,j}$ induces the processing of inner product tasks $<r_i, x>$ and $<r_j, x>$ consecutively, whereas a cut edge $e_{i,j}$ means that the inner-product tasks $<r_i, x>$ and $<r_j, x>$ are not processed consecutively.

Consider an uncut edge $e_{h,\ell}$. During the inner products $< r_h, x >$ and $< r_\ell, x >$, $w(e_{h,\ell})$ x-vector entries corresponding to the columns shared between rows r_h and r_ℓ will possibly be reused due to exploiting temporal locality. Consider a cut edge $e_{i,j}$. During the inner-products $<r_i, x>$ and $<r_j, x>$, $w(e_{i,j})$ x-vector entries corresponding to the columns shared between rows r_i and r_j may not be reused. So, a cut edge $e_{i,j}$ will incur at most $w(e_{i,j})$ extra cache misses due to loss of temporal locality. So, minimizing the edgecut according to (1) relates to minimizing the number of cache misses.



(a) Compressed matrix $\hat{A}_{\text{line}}$. (b) $\mathcal{H}^T_{\text{line}}$ model.

Fig. 5: Compressed matrix $\hat{A}_{\text{line}}$ and its temporal hypergraph model $\mathcal{H}^T_{\text{line}}$ for exploiting spatial and temporal localities in two partitioning phases.

3.2.2 Hypergraph Model $\mathcal{H}^T$ (TH Model)

For a given matrix A, the dependencies of the inner-product tasks on the x-vector entries are represented as a hypergraph $\mathcal{H}^T(A) = (\mathcal{V}^t, \mathcal{N}^d)$. In $\mathcal{H}^T(A)$, there exists a task vertex $v_i^t \in \mathcal{V}^t$ for each row r_i of A. For each column c_j of A, there exists a data net $n_j^d \in \mathcal{N}^d$. Net n_j^d connects the vertices representing the rows that share column c_j, that is,

$$Pins(n_j^d) = \{v_i^t : a_{i,j} \neq 0\}. \tag{4}$$

We associate vertices and nets with unit weights. $\mathcal{H}^T$ (A) is topologically equivalent to the column-net model [8] of matrix A.

Fig. 3d shows the $\mathcal{H}^T$ model of the sample matrix given in Fig. 2. In Fig. 3d, data net n_f connects task vertices v_5 and v_6 because the inner product tasks $<r_5, x>$ and $<r_6, x>$ both require the x-vector entry x_f.

A brief insight on how the method works can be given as follows under the assumption that each line stores one word: Since the rows corresponding to the vertices in a part are reordered consecutively, the corresponding inner-product operations can reuse the required x-vector entries. A net n_j^d with connectivity set $con(n_j^d)$ means that x-vector entry x_j is required by the inner-products consisting of the rows represented by the vertices in $con(n_j^d)$. Assuming that x-vector entries are reused only in processing the rows belonging to the same part and each line can store one word, $|con(n_j^d)|$ cache misses occur due to accessing x_j. So, minimizing the cutsize according to (2) corresponds to minimizing the number of cache misses due to loss of temporal locality.

4 Exploiting Spatial and Temporal Localities in Two Phases

The correctness of the temporal methods TG and TH discussed in Section 3.2 is based on the assumption that each cache line stores only one word. In order to exploit both spatial and temporal localities and avoid this assumption, we propose to utilize the spatial and temporal methods respectively proposed in Sections 3.1 and 3.2 in a two-phase approach. In the first phase, a spatial method is applied to find a reordering on the x-vector entries and then this reordering is utilized to determine an allocation of x-vector entries to blocks (cache lines). That is, for a cache line of size L words, the first L x-vector entries are assigned to the first line, the second L x-vector entries are assigned to the second line, etc. In the second phase, a temporal method is applied by utilizing this information about the assignment of x-vector entries to lines.

Although all of the four combinations of spatial and temporal methods are valid, we only consider the hypergraph models and we propose a new spatiotemporal method SH$\rightarrow$TH$_{\text{line}}$ which exploits temporal locality on the access of lines instead of single words.

For a given matrix A, consider $\mathcal{H}^S(A) = (\mathcal{V}^d, \mathcal{N}^t)$ and its K-way vertex-partition $\Pi(\mathcal{V}^d)$. When K is large enough, we obtain an order $\Gamma(\mathcal{V}^d)$ from $\Pi(\mathcal{V}^d)$ on data vertices $\mathcal{V}^d$ of $\mathcal{H}^S(A)$. The reordering $\Gamma(\mathcal{V}^d)$ is used to assign x-vector entries to cache lines, possibly in a cache-oblivious manner since K is large enough. Consider applying the same reordering $\Gamma(\mathcal{V}^d)$ to the columns of the A matrix to obtain reordered matrix $\hat{A}$ (e.g., as shown in Fig. 4b). That is, column reordering and x-vector reordering are conformal to each other so that the k^{th} column stripe of $\hat{A}$ corresponds to the k^{th} line of the x-vector. Then we perform a column-wise compression of the $p\times q$ $\hat{A}$ matrix to construct a $p\times q_L$ matrix $\hat{A}_{\text{line}}$ (as shown in Fig. 5a), where $q_L = \lceil \frac{q}{L} \rceil$ is the number of lines required to store the x-vector. For each $k = 1, 2, ..., L$, we compress the k^{th} column stripe into a single column with sparsity pattern being equal to the union of sparsities of all columns that lie in the k^{th} column stripe. Note that the compression of $\hat{A}$ is performed for only building the hypergraph model, and it has no effect on the CRS data structure utilized during the local SpMV operations.

In $\hat{A}_{\text{line}}$, there exists a nonzero $a_{i,k}$ if at least one column in the k^{th} column stripe of $\hat{A}$ has a nonzero in row i. $\hat{A}_{\text{line}}$ summarizes the requirement of x-vector lines by the inner product tasks. Hence, we can easily construct a temporal hypergraph model $\mathcal{H}^T_{\text{line}}(A) = \mathcal{H}^T(\hat{A}_{\text{line}}) = (\mathcal{V}^t, \mathcal{N}^d_{\text{line}})$. In $\mathcal{H}^T_{\text{line}}(A)$, there exists a task vertex $v^t_i \in \mathcal{V}^t$ for each row r_i of A. For each consecutive L data vertices in $\Gamma(\mathcal{V}^d)$, there exists a data line net $n^d_j \in \mathcal{N}^d_{\text{line}}$. We associate vertices and nets with unit weights.

Fig. 5b shows the $\mathcal{H}^T_{\text{line}}$ model of the sample matrix given in Fig. 2. In Fig. 5b, the data net n_{cg} connects task vertices v_2, v_4 and v_5 because the inner product tasks associated with rows r_2, r_4 and r_5 require the line containing x_c and x_g.

In a partition $\Pi(\mathcal{V}^t)$ of vertices of $\mathcal{H}^T_{\text{line}}$, minimizing the cutsize according to (2) corresponds to minimizing the number of cache misses due to loss of temporal locality. The correctness of $\mathcal{H}^T_{\text{line}}$ can be derived from the explanation given for $\mathcal{H}^T$ in Section 3.2.2 with omitting the assumption that each line stores only one word.

5 Exploiting Spatial and Temporal Localities Simultaneously

In the GP- and HP-based spatiotemporal methods proposed in this section, the dependencies of the inner-product tasks on the x-vector entries and the requirements of x-vector entries by the tasks are represented as a bipartite graph and a hypergraph, respectively.

5.1 Bipartite Graph Model $\mathcal{G}^{ST}$ (STBG Model)

In $\mathcal{G}^{ST}(A) = (\mathcal{V} = \mathcal{V}^d \cup \mathcal{V}^t, \mathcal{E})$, there exists a data vertex $v^d_j \in \mathcal{V}^d$ for each column c_j of A. For each row r_i of A, there exists a task vertex $v^t_i \in \mathcal{V}^t$. There exists an edge $e_{i,j} \in \mathcal{E}$ if there is a nonzero $a_{i,j}$. We associate vertices and edges

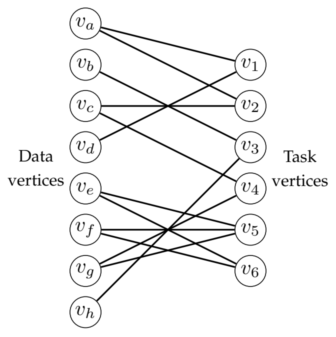


Fig. 6: STBG: Bipartite Graph model $\mathcal{G}^{ST}(A)$ for exploiting Spatial and Temporal localities simultaneously.

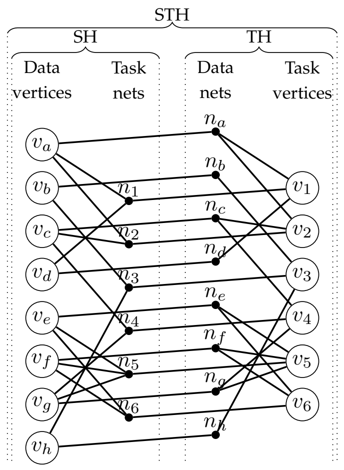


Fig. 7: STH: Hypergraph model $\mathcal{H}^{ST}(A)$ (of the sample matrix given in Fig. 2) for exploiting Spatial and Temporal localities simultaneously.

with unit weights. Fig. 6 shows the $\mathcal{G}^{ST}$ model of the sample matrix given in Fig. 2. In Fig. 6, the data vertex v_a is adjacent to the task vertices v_1 and v_2 because the inner product tasks associated with rows r_1 and r_2 require the x-vector entry x_a. In Fig. 6, the task vertex v_5 is adjacent to data vertices v_e, v_f and v_g because the inner product task associated with row r_5 depends on the x-vector entries x_e, x_f and x_g.

A brief insight on how the method works can be given as follows: Consider a task vertex v^t_i that is adjacent to D data vertices. Among the D edges connecting v^t_i to the data vertices, C cut edges mean that the inner-product task $<r_i, x>$ will not access at most C x-vector entries consecutively. On the other hand, $D - C$ uncut edges mean that $< r_i, x >$ will access $D - C$ x-vector entries consecutively. Similarly, consider a data vertex v^d_j that is adjacent to T task vertices. Among the T edges connecting v^d_j to task vertices, C cut edges mean that x-vector entry x_j will not be reused by at most C tasks due to loss of temporal locality. On the other hand, $T - C$ uncut edges mean that x-vector entry x_j will be reused by $T - C$ tasks since the rows corresponding to tasks in the same part are reordered consecutively. So, minimizing the edgecut according to (1) relates to reducing the number of cache misses due to loss of both spatial and temporal localities.

In $\mathcal{G}^{ST}$, an edge $e_{i,j}$ can be interpreted as encoding spatial and temporal localities simultaneously. However, the $\mathcal{G}^{ST}$ model overestimates the number of cache misses due to the deficiency of the graph model in encoding multi-way relations as also discussed in a different context in [8].

5.2 Hypergraph Model $\mathcal{H}^{ST}$ (STH Model)

In $\mathcal{H}^{ST}(A)=(\mathcal{V}=\mathcal{V}^d \cup \mathcal{V}^t, \mathcal{N}=\mathcal{N}^t \cup \mathcal{N}^d)$, there exists a data vertex $v_j^d \in \mathcal{V}^d$ and a data net $n_j^d \in \mathcal{N}^d$ for each column c_j of A. For each row r_i of A, there exists a task vertex $v_i^t \in \mathcal{V}^t$ and a task net $n_i^t \in \mathcal{N}^t$. Task net n_i^t connects the data vertices corresponding to columns that share row r_i, as well as task vertex v_i^t. Data net n_j^d connects the task vertices corresponding to rows that have a nonzero at column c_j, as well as data vertex v_j^d. That is,

$$\begin{aligned} Pins(n_j^d) &= \{v_i^t : a_{i,j} \neq 0\} \cup \{v_j^d\}, \\ Pins(n_i^t) &= \{v_j^d : a_{i,j} \neq 0\} \cup \{v_i^t\}. \end{aligned} \tag{5}$$

We associate vertices and nets with unit weights. Fig. 7 shows the $\mathcal{H}^{ST}$ model of the sample matrix given in Fig. 2. In the figure, task net n_5 connects data vertices v_e, v_f and v_g because the inner product task associated with row r_5 requires the x-vector entries x_e, x_f and x_g. Net n_5 also connects v_5. Data net n_f connects task vertices v_5 and v_6 because the inner product tasks associated with rows r_5 and r_6 require the x-vector entry x_f. Net n_f also connects v_f.

Comparison of (5) with (3) and (4) as well as comparison of Fig. 7 with Figs. 3c and 3d show that the STH model can be considered to be obtained by combining the SH and TH models into a composite hypergraph model whose partitioning will encode both spatial and temporal localities. The STH model contains both SH and TH hypergraphs, where these two hypergraphs are combined through making each data net n_i^d also connect the respective data vertex v_i^d as well as making each task net n_i^t also connect the respective task vertex v_i^t. Fig. 7 clearly shows how the composite hypergraph model STH is obtained from the constituent hypergraph models SH and TH. As seen in the figure, the left part contains SH, the right part contains TH and the middle part shows the data-net–to–vertex connections and task-net–to–vertex connections performed to realize the composition. In this way, the STH model encodes simultaneous clustering of rows and columns with similar sparsity patterns to the same part.

A task net n_i^t connecting vertices in $Pins(n_i^t)$ encodes the requirement of x-vector entries in $\{x_k : v_k^d \in Pins(n_i^t) \setminus \{v_i^t\}\}$ by the inner product $<r_i, x>$. Hence, assigning the vertices in $Pins(n_i^t) \setminus \{v_i^t\}$ into the same part increases the possibility of exploiting spatial locality in accessing x-vector entries during the inner product $< r_i, x >$. Similarly, a data net n_j^d connecting vertices in $Pins(n_j^d)$ encodes the access of the inner products consisting of rows in $\{r_k : v_k^t \in Pins(n_j^d) \setminus \{v_j^d\}\}$ to x-vector entry x_j. Hence, assigning the vertices in $Pins(n_j^d) \setminus \{v_j^d\}$ into the same part increases the possibility of exploiting temporal locality in accessing x_j. The STH method can simultaneously achieve these two aims while obtaining a single $\Pi(\mathcal{V})$ of $\mathcal{H}^{ST}$.

A brief insight on how the method works can be given as follows: A cut data-net n_j^d that connects one data vertex and $|Pins(n_j^d)| - 1$ task vertices means that x-vector entry x_j will not be re-used by at most $|con(n_j^d)| - 1$ tasks. An uncut net n_j^d means that x_j will be re-used by $|Pins(n_j^d)| - 1$ inner-product tasks. Similarly, a task cut net n_i^t that connects one task vertex and $|Pins(n_i^t)| - 1$ data vertices means that inner-product task $< r_i, x >$ will not access, at most, $|con(n_i^t)| - 1$ x-vector entries consecutively. An uncut net n_i^t means that task i will access $|Pins(n_i^t)| - 1$ x-vector entries consecutively, hence exploiting spatial locality. So, minimizing the cutsize according to (2) corresponds to minimizing the number of cache misses due to loss of both spatial and temporal localities.

TABLE 2: Sizes of graph and hypergraph models for an $nr \times nc$ matrix that contains nnz nonzeros

Graphs			
Method	model	# vertices	# edges
SG	$\mathcal{G}^S$	nc	$O(\frac{nc^2}{2})$
TG	$\mathcal{G}^T$	nr	$O(\frac{nr^2}{2})$
STBG	$\mathcal{G}^{ST}$	$nr+nc$	nnz

Hypergraphs				
Method	model	# vertices	# nets	# pins
SH	$\mathcal{H}^S$	nc	nr	nnz
TH	$\mathcal{H}^T$	nr	nc	nnz
STH	$\mathcal{H}^{ST}$	$nr+nc$	$nr+nc$	$nr+nc+2nnz$
TH$_{\text{line}}$ $\phi 1$	$\mathcal{H}^S$	nc	nr	nnz
TH$_{\text{line}}$ $\phi 2$	$\mathcal{H}^T_{\text{line}}$	nr	$\frac{nc}{L}$	$[\frac{nnz}{L}, nnz]$

6 Comparison of The Reordering Models

Table 2 compares the sizes of the graph and hypergraph models in terms of the number of rows (nr), columns (nc) and nonzeros (nnz) of a given matrix A. The similarity graph models for SG and TG may be quite dense when A contains dense columns and rows, respectively, as shown by the square upper-bound on their number of edges.

As seen in Table 2, the number of vertices and nets in SH and TH are determined by nr and nc in a dual manner, whereas the number of pins in both methods is determined by nnz. The number of vertices and nets of STH is determined by $nr + nc$, whereas its number of pins is determined by $nr + nc + 2nnz$. This is because the hypergraph model of STH is a composite model obtained from the hypergraph models of SH and TH.

In TH$_{\text{line}}$, phase one ($\phi 1$) uses the $\mathcal{H}^S$ model, so the number of vertices, nets and pins are identical to those of SH. In $\phi 2$, the number of nets is reduced to the number of columns of the resulting matrix of $\phi 1$, which is nc/L, where L is the line size. The number of pins is reduced to the number of nonzeros of the new matrix, which lies in the range $[nnz/L, nnz]$.

As seen in Table 2, the hypergraph models for SH and TH as well as the bipartite graph model for STBG are of the same size, whereas the size of the hypergraph model of STH is slightly more than twice the size of those models.

7 Experiments

7.1 Dataset

The empirical verifications of the proposed methods are performed on 60 irregular sparse matrices arising in a variety of applications. These matrices are obtained from the University of Florida Sparse Matrix Collection [9]. Table 3 shows the properties of these matrices, which are grouped into three categories: symmetric, square nonsymmetric and rectangular. In each category, matrices are sorted in alphabetical order of their names. In the table, "avg" and "max" respectively denote the average and maximum number of nonzeros per row/column. "cov" denotes the coefficient of variation of the number of nonzeros per row/column and is defined as the ratio of standard deviation to average. A "cov" value may be used to quantify the amount of irregularity in the sparsity pattern of a matrix. That is, larger "cov" values might refer to higher irregularity.

7.2 Experimental Framework

We evaluate the performances of the proposed methods on a single 60-core Xeon Phi 5110 co-processor in native mode. Each core of the Xeon Phi supports running up to four hardware threads, is clocked at 1.053 GHz and has a private 32 KB L1 cache. The Xeon Phi has 30 MB L2 cache.

We also experiment on a two-socket Xeon system, which has two E5-2643 processors having 8 cores in total. Each core supports running up to two hardware threads, is clocked at 3.30 GHz and has a private 32 KB L1 cache and a 256 KB L2 cache. Each Xeon processor has a 10 MB L3 cache.

For performing parallel SpMV operations, we use the Sparse Library (SL) [5], [6], [7], which is highly optimized for today's processors. For evaluations on the Xeon Phi, we use SL's row-parallel scheme based on vectorized Bidirectional Incremental CRS (vecBICRS), which is reported to perform the best in [5] for the Xeon Phi architecture. For evaluations on the Xeon processor, we use the SL's row-parallel scheme based on Incremental CRS (ICRS), which proceeds similar to CRS (Algorithm 1). The original row-parallel scheme of the SL library uses Hilbert-curve ordering, however we remove this ordering (for results on both Xeon Phi an Xeon) so that we can evaluate our reordering methods properly.

We use 240 threads for the Xeon Phi as suggested in [5], [7]. On the Xeon Phi, we test with non-vectorized code (referred to as 1×1), as well as all provided vectorization options, i.e., 1×8, 2×4, 4×2 and 8×1. We report the result of the best-performing vectorization option in all Xeon Phi related tables and figures except Table 5, in which results of all blocking options are reported separately. The SL library without using any reordering is selected as a baseline method, which is referred to as BaseLine (BL). The performance results are obtained by averaging 5000 SpMVs.

Construction of hypergraph and graph models takes linear time in the number of nonzeros of matrix A. However, construction of similarity graphs takes $O(|\mathcal{V}|^2)$ time, which is unacceptably high. The construction of similarity graphs corresponds to the construction of clique-net graph model of the respective hypergraph model. So we use the randomized clique-net graph model [8].

For partitioning the graph models, we use MeTiS [10] with the partitioning objective of minimizing the edge-cut metric given in (1). For partitioning the hypergraph models, we use PaToH [8], [11] with the partitioning objective of minimizing the "connectivity-1" cutsize metric. Note that minimizing the connectivity metric given in (2) directly corresponds to minimizing the "connectivity-1" metric because there is always a constant difference between them. MeTiS and PaToH are used with default parameters with the exception of the allowed imbalance ratio, which is set to 30%. Since these partitioning tools contain randomized algorithms, we repeat each partitioning instance three times and report the average results.

For each matrix, the K value given to MeTiS or PaToH is determined through dividing the storage size (in KB) reported by the SL library by the L1 cache size of 32 KB.

In the following figures, the performance results are displayed utilizing *performance profiles* [12] which is a generic tool for better comparing many methods over large test instances. In a performance profile, each method is plotted as the fraction of test cases that are x-magnitude worse than the best performing instance of the best performing method. For example, a point (abscissa = 1.10, ordinate = 0.40) on the performance plot of a given method means that for 40% of the test instances, the method performs within a factor of 1.10 of the best result. Hence, the method closest to the top-left corner is the best method.

7.3 Performance Evaluation on Xeon Phi

7.3.1 Comparison of GP- and HP-based Methods

We compare the performances of the GP-based and HP-based methods in each locality type of spatial, temporal and spatiotemporal. In Fig. 8, we present the performance profiles of the parallel SpMV times for spatial, temporal and spatiotemporal methods. As seen in Figs. 8a, 8b and 8c, the HP-based methods perform significantly better than their GP-based counterparts in each locality type of spatial, temporal and spatiotemporal. These findings can be attributed to the better capability of hypergraph models in representing data localities during SpMV operations.

7.3.2 Comparison of HP-based Spatiotemporal Methods

We compare the proposed spatiotemporal methods STH and $\mathrm{TH_{line}}$ against three HP-based spatiotemporal methods. The first two methods, which are described in [13] and [14], encode the spatial locality primarily and the temporal locality secondarily, and vice-versa, respectively. The former and latter methods are referred to as SBD and $\mathrm{sHP_{CN}}$. The SBD and $\mathrm{sHP_{CN}}$ methods can be respectively considered as the extensions of SH and TH methods, where the nets are considered as inducing partial orderings on the other dimension of the matrix (based on uncut- and cut-net categorization) to secondarily encode temporal and spatial locality respectively.

The reasoning behind the classification of exploiting spatial/temporal locality as primary and secondary can be explained as follows: In hypergraph models, the objective of minimizing the cutsize directly and closely relates to clustering vertices with similar net connectivity to the same parts. However, the partitioning objective of minimizing the cutsize relates indirectly and hence loosely to clustering nets with similar vertex connectivity to the same parts as uncut nets. A similar observation was reported in a different context in [2]. So, partial row or column reordering respectively induced by the net reordering of SBD or $\mathrm{sHP_{CN}}$ loosely relates to clustering columns or rows with similar sparsity pattern thus failing to address the secondary objective of exploiting spatial or temporal locality successfully.

The third method, referred to here as SH+TH, is developed for the sake of comparison. The SH+TH method consists of using SH for reordering columns and TH for

TABLE 3: Properties of test matrices

		Number of			Nnz's in a row			Nnz's in a column		
MID	Matrix Name	rows	columns	nonzeros	avg	max	cov	avg	max	cov
	Symmetric matrices									
01	144	144,649	144,649	2,148,786	14.86	26	0.18	14.86	26	0.18
02	adaptive	6,815,744	6,815,744	27,248,640	4.00	4	0.01	4.00	4	0.01
03	AS365	3,799,275	3,799,275	22,736,152	5.98	14	0.14	5.98	14	0.14
04	bmw7st_1	141,347	141,347	7,339,667	51.93	435	0.25	51.93	435	0.25
05	ca2010	710,145	710,145	3,489,366	4.91	141	0.59	4.91	141	0.59
06	citationCiteseer	268,495	268,495	2,313,294	8.62	1318	1.89	8.62	1318	1.89
07	coAuthorsCiteseer	227,320	227,320	1,628,268	7.16	1372	1.48	7.16	1372	1.48
08	darcy003	389,874	389,874	2,101,242	5.39	7	0.36	5.39	7	0.36
09	delaunay_n18	262,144	262,144	1,572,792	6.00	21	0.22	6.00	21	0.22
10	delaunay_n19	524,288	524,288	3,145,646	6.00	21	0.22	6.00	21	0.22
11	F2	71,505	71,505	5,294,285	74.04	345	0.51	74.04	345	0.51
12	G3_circuit	1,585,478	1,585,478	7,660,826	4.83	6	0.13	4.83	6	0.13
13	gupta2	62,064	62,064	4,248,286	68.45	8413	5.20	68.45	8413	5.20
14	hugetrace-00020	16,002,413	16,002,413	47,997,626	3.00	3	0.01	3.00	3	0.01
15	hugetric-00020	7,122,792	7,122,792	21,361,554	3.00	3	0.01	3.00	3	0.01
16	m14b	214,765	214,765	3,358,036	15.64	40	0.20	15.64	40	0.20
17	NACA0015	1,039,183	1,039,183	6,229,636	5.99	10	0.14	5.99	10	0.14
18	netherlands_osm	2,216,688	2,216,688	4,882,476	2.20	7	0.26	2.20	7	0.26
19	NLR	4,163,763	4,163,763	24,975,952	6.00	20	0.14	6.00	20	0.14
20	ny2010	350,169	350,169	1,709,544	4.88	61	0.52	4.88	61	0.52
21	pattern1	19,242	19,242	9,323,432	484.54	6028	0.78	484.54	6028	0.78
22	pkustk12	94,653	94,653	7,512,317	79.37	4146	1.87	79.37	4146	1.87
23	vsp_bcsstk30_500sep_10in_1Kout	58,348	58,348	4,033,156	69.12	219	0.47	69.12	219	0.47
24	vsp_msc10848_300sep_100in_1Kout	21,996	21,996	2,442,056	111.02	722	0.44	111.02	722	0.44
	Square nonsymmetric matrices									
25	amazon0312	400,727	400,727	3,200,440	7.99	10	0.38	7.99	2747	1.89
26	amazon0505	410,236	410,236	3,356,824	8.18	10	0.38	8.18	2760	1.87
27	amazon0601	403,394	403,394	3,387,388	8.40	10	0.33	8.40	2751	1.82
28	av41092	41,092	41,092	1,683,902	40.98	2135	4.08	40.98	664	2.37
29	circuit5M_dc	3,523,317	3,523,317	19,194,193	5.45	27	0.38	5.45	25	0.23
30	flickr	820,878	820,878	9,837,214	11.98	10272	7.32	11.98	8549	5.97
31	heart1	3,557	3,557	1,387,773	390.15	1120	0.32	390.15	1120	0.32
32	laminar_duct3D	67,173	67,173	3,833,077	57.06	89	0.66	57.06	89	0.52
33	soc-Slashdot0811	77,360	77,360	905,468	11.70	2508	3.15	11.70	2540	3.18
34	soc-Slashdot0902	82,168	82,168	948,464	11.54	2511	3.20	11.54	2553	3.25
35	TSOPF_RS_b300_c1	14,538	14,538	1,474,325	101.41	209	1.01	101.41	6902	4.68
36	TSOPF_RS_b300_c3	42,138	42,138	4,413,449	104.74	209	0.98	104.74	20702	7.99
37	twotone	120,750	120,750	1,224,224	10.14	185	1.48	10.14	188	1.86
38	web-NotreDame	325,729	325,729	1,497,134	4.60	3445	4.67	4.60	10721	8.50
39	web-Stanford	281,903	281,903	2,312,497	8.20	255	1.38	8.20	38606	20.28
40	webbase-1M	1,000,005	1,000,005	3,105,536	3.11	4700	8.16	3.11	28685	11.88
41	wiki-Talk	2,394,385	2,394,385	5,021,410	2.10	100022	47.64	2.10	3311	5.82
	Rectangular matrices									
42	cont1_l	1,918,399	1,921,596	7,031,999	3.67	5	0.26	3.66	1279998	252.33
43	cont11_l	1,468,599	1,961,394	5,382,999	3.67	5	0.26	2.74	7	0.90
44	dbir2	18,906	45,877	1,158,159	61.26	4950	3.86	25.24	233	1.49
45	GL7d14	171,375	47,271	1,831,183	10.69	24	0.24	38.74	160	0.44
46	GL7d15	460,261	171,375	6,080,381	13.21	38	0.18	35.48	137	0.40
47	GL7d24	21,074	105,054	593,892	28.18	751	0.72	5.65	13	0.22
48	IMDB	428,440	896,308	3,782,463	8.83	1334	1.73	4.22	1590	3.09
49	mesh_deform	234,023	9,393	853,829	3.65	4	0.18	90.90	166	0.20
50	neos	479,119	515,905	1,526,794	3.19	29	0.16	2.96	16220	15.57
51	NotreDame_actors	392,400	127,823	1,470,404	3.75	646	2.75	11.50	294	1.02
52	nsct	23,003	37,563	697,738	30.33	848	2.54	18.58	628	3.32
53	pds-100	156,243	514,577	1,096,002	7.01	101	1.03	2.13	3	0.18
54	pds-80	129,181	434,580	927,826	7.18	96	0.99	2.13	3	0.18
55	pds-90	142,823	475,448	1,014,136	7.10	96	1.01	2.13	3	0.18
56	rel8	345,688	12,347	821,839	2.38	4	0.82	66.56	121	0.26
57	route	20,894	43,019	206,782	9.90	2781	7.06	4.81	44	1.01
58	sgpf5y6	246,077	312,540	831,976	3.38	61	2.21	2.66	12	0.74
59	Trec14	3,159	15,905	2,872,265	909.23	1837	0.39	180.59	2500	1.71
60	watson_1	201,155	386,992	1,055,093	5.25	93	2.45	2.73	9	0.47

reordering rows. The "+" notation in SH+TH refers to the fact that the partitioning in each of these two methods are performed independently so that they do not use the partitioning results of each other.

Fig. 9 shows the performance profiles that compare all HP-based spatiotemporal methods, where the detailed results are presented in Table A.1 in the supplemental material. As seen in the figure, STH is the clear winner, followed by TH_{line} and SH+TH which display comparable performance. The significantly better performance obtained by TH_{line} and SH+TH compared to SBD and sHP_{CN} is expected since both SBD and sHP_{CN} encode one type of locality primarily while encoding the other type of locality secondarily. This experimental finding also supports our claim for reordering based on vertex partitions encode much better locality than that on cut/uncut net categorization.

7.3.3 *Comparison of All Methods*

Table 4 displays the averages of parallel SpMV performances of the methods in terms of Gflop/s for 60 test matrices in three categories. Here, SG+TG refers to the graph counterpart of SH+TH. For each method, the table also

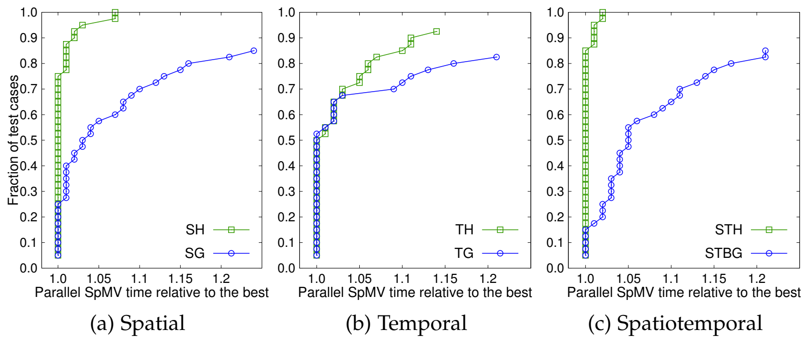


Fig. 8: Performance profiles for comparing hypergraph and graph models on Xeon Phi.

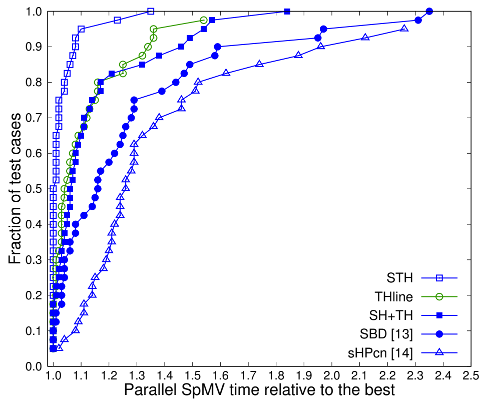


Fig. 9: Performance profiles for comparing HP-based spatiotemporal methods on Xeon Phi.

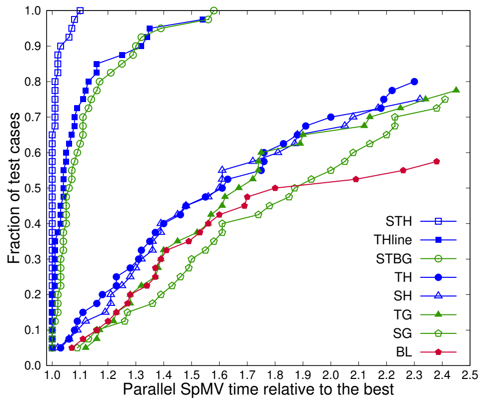


Fig. 10: Performance profiles of SpMV times on Xeon Phi.

contains the number of best results for per-category matrices and for all matrices in the bottom of the table. Table 4 shows that we obtain 13 instances that have more than 20 Gflop/s performance through the proposed reordering methods, whereas there are only 4 instances for the baseline method that have more than 20 Gflop/s. As also seen in Tables 4, the proposed SG+TG, STBG, SH+TH, $\mathrm{TH_{line}}$ and STH respectively perform the best in 1, 4, 10, 12, and 33 instances. These results show the superior performance of STH, which encodes spatial and temporal localities simultaneously.

Table 4 also displays SpMV performances of the methods averaged over matrices in each category. As seen in the table, the SH+TH method performs close to STH in the category of symmetric matrices. This is due to the nature of symmetric matrices, where SH and TH are exactly the same. On the other hand, in case of nonsymmetric square matrices and rectangular matrices, STH performs significantly better than other methods.

As seen in Table 4, the two-phase method $\mathrm{TH_{line}}$ does not outperform SH+TH in case of symmetric matrices. On the other hand, it performs slightly better in case of nonsymmetric square matrices and rectangular matrices.

Table 4 also shows the importance of using reordering methods as the STH method achieved 3.4x improvement for symmetric matrices and 1.6x improvement for other categories. The importance of simultaneous methods also arises from the fact that some matrices benefit more from exploiting spatial locality while other matrices benefit more from exploiting temporal locality. The simultaneous methods can benefit from exploiting both localities which leads to a better performance.

Fig. 10 shows the performance profiles of the SpMV times for all methods. As seen in the figure, STH, $\mathrm{TH_{line}}$ and STBG significantly outperform other methods. In the figure, STH is a clear winner, while $\mathrm{TH_{line}}$ is close to, but slightly outperforms the STBG method.

As seen in Table 4 as well as in Fig. 10, the STH method is the best method in terms of geometric means (overall and per-category) and the number of best instances.

7.3.4 Benefit of Exploiting Spatial and Temporal Localities

Exploiting locality in matrices with highly irregular sparsity patterns might have a huge impact on SpMV performance. To show this, three matrices are selected from the data set, where the improvement in performance after reordering with respect to the baseline is high. Fig. 11a shows the plot of the sparse matrix `144` which has a very irregular sparsity pattern. Table 4 shows that the performance on the matrix (MID=01) improves from 3.09 (by BL) to 5.09 Gflop/s by exploiting only spatial locality using the SH method, to 7.58 Gflop/s by exploiting only temporal locality using the TH method, and to 16.78 Gflop/s by exploiting both spatial and temporal localities using the STH method. Fig. 11b shows the reordered matrix after applying the STH method.

Fig. 11c shows the plot of the second matrix `delaunay_n18` (MID=09) which is experimentally found to have good spatial locality properties, but suffers from poor temporal locality. As seen in Table 4, trying to improve spatial locality alone using the SH method does not improve the performance. However, exploiting temporal locality alone has a high effectiveness on the performance, as it is increased from 3.48 Gflop/s to 12.54 Gflop/s after reordering the rows with the TH method. When targeting both localities, the performance increases to 16.34 by the STH method. Fig. 11d shows the reordered matrix after applying the STH method.

Fig. 11e shows the plot of the last selected matrix `pds-80` (MID=54) which is experimentally found to suffer from poor spatial locality. As seen in Table 4, reordering using a temporal method will not improve the performance. However, reordering the columns to exploit spatial locality with the SH method improves the performance significantly from 4.30 Gflop/s to 10.04 Gflop/s. On the other hand, targeting both localities improves the performance further to 12.35 Gflop/s when using $\mathrm{TH_{line}}$ method.

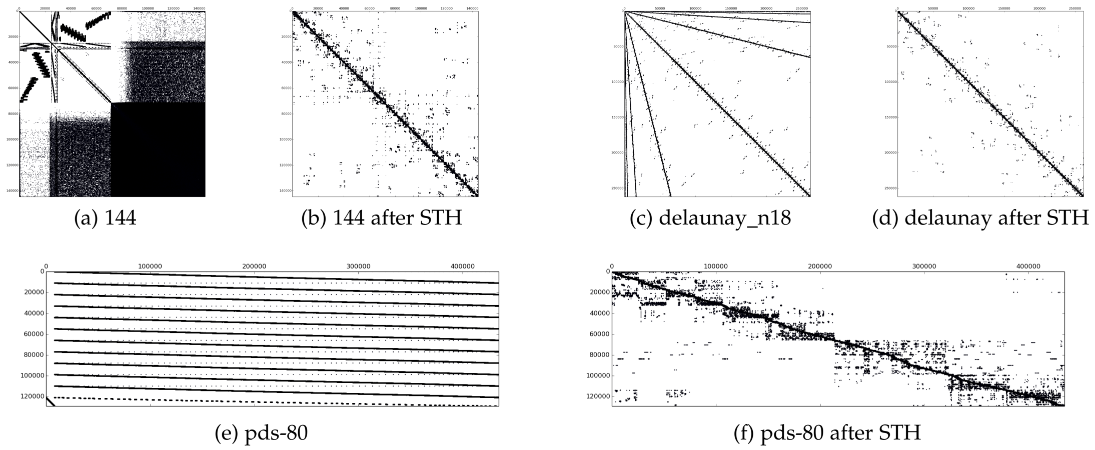

(a) 144 (b) 144 after STH (c) delaunay_n18 (d) delaunay after STH (e) pds-80 (f) pds-80 after STH

Fig. 11: Plots of sample matrices before and after applying the STH reordering method.

Table 5 shows the geometric means of the Gflop/s performances obtained by all methods and all vectorization options for all matrices in the dataset. In the table, column "Best" means the per-instance result of best performed option among all vectorization options as well as no-vectorization option, while V-Avg. means the per-instance average of all vectorization options, not including 1×1. As seen in the table, STH outperforms other methods in all vectorization options. An important observation we can retrieve from Table 5 is the importance of the one-phase spatiotemporal methods in utilizing vectorization. As the table shows, with no reordering or reordering for only one type of locality (either spatial or temporal), non-vectorized runs might outperform other vectorization options. However, in one-phase spatiotemporal methods, vectorized runs always outperform non-vectorized runs. The impact of reordering on vectorization is discussed further in Section 7.3.5.

7.3.5 The Impact of Reordering on Vectorization

Vectorization might be very beneficial in improving the performance of SpMV on the Xeon Phi co-processor. However, it is highly dependent on the sparsity pattern of the input matrix, meaning that if the sparsity pattern is not favoured by any of the vectorization options or the matrix has a very irregular sparsity pattern, the vectorization might not improve or even degrade the performance of SpMV. Our findings show that the reordering methods have an important role in utilizing the vectorized SpMV algorithm.

We use Table 5 to show the importance of reordering for utilizing vectorization. In the table, the baseline method performs worse in case of 1×8 option compared to no-vectorization option, i.e., 1×1. If we look at the vectorization performances obtained by methods that target only one type of locality, i.e., SG, TG, SH and TH, we can see that some vectorization options might perform worse than the no-vectorization option as follows: If the method exploits spatial locality, then the blocking option that prefers extreme temporal locality (8×1 in this case) might perform worse than the 1×1 option. On the other hand, if the method exploits temporal locality, then the blocking option that prefers extreme spatial locality (1×8 in this case) might perform worse than the 1×1 option.

Regarding the spatiotemporal methods that target both localities (e.g., STBG, STH and TH$_{\text{line}}$), Table 5 shows that using any blocking option always performs better than the 1×1 option. It also shows that the spatiotemporal methods perform the best in terms of the average of all vectorization options (V-Avg. column).

7.4 Performance Evaluation on Xeon Processor

Fig. 12a displays the performance profiles of SpMV times of all methods on the Xeon processor. Table A.2 in the supplemental material shows the detailed performances results of all methods on the Xeon processor in terms of Gflop/s. Both Fig. 12a and Table A.2 show the superiority of STH and STBG methods over the others. Recall that the TH$_{\text{line}}$ performs better than the STBG method on the Xeon Phi. However, as no vectorization is used on the Xeon, TH$_{\text{line}}$ performs worse because it mostly utilizes vectorization to perform well as discussed in Section 7.3.5.

We also use `likwid` [15], which enables counting data transfers in a multi-threaded shared memory system, to measure L2 cache misses on the Xeon server during SpMV operation using all methods. Fig. 12b shows the performance profiles for cache misses. Table A.3 in the supplemental material contains detailed results of the number of cache misses normalized with respect to those of the BL method averaged over each category and at the bottom, for all matrices in the dataset. The comparison of Figs. 12b and 12a shows that the methods aiming at reducing cache misses effectively reduce the SpMV runtime.

7.5 Integration into Iterative Methods

Iterative symmetric linear solvers (e.g., CG) contain repeated SpMV operations that involve the same symmetric matrix. In such solvers, the input vector (x-vector) of the next iteration is obtained from the output vector (y-vector) of the current iteration via linear vector operations. Efficient implementation of these linear vector operations necessitates conformal partitioning/ordering of x- and y-vector entries. So for such solvers, after each SpMV operation, the y-vector entries should be reordered to the same order of the x-vector entries to establish conformal x-y ordering.

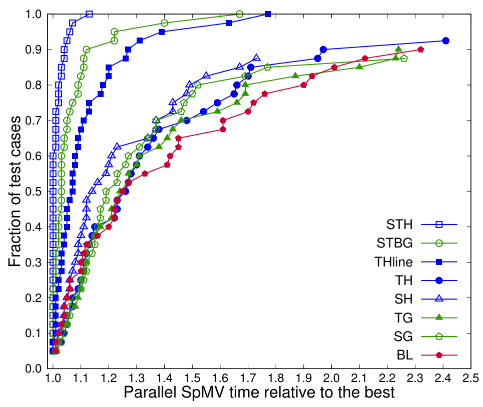
Iterative nonsymmetric linear solvers (e.g., CGNE, CGNR and QMR [16], [17], [18]) contain repeated SpMV and SpMTV (matrix-transpose-vector multiply) operations that involve the same nonsymmetric square matrix. In these

TABLE 4: Performance results (in Gflop/s) on Xeon Phi

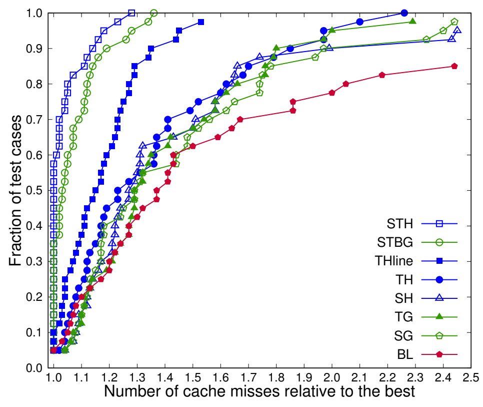
		Graphs				Hypergraphs				
MID	BL	SG	TG	SG+TG	STBG	SH	TH	SH+TH	TH_{line}	STH
					Symmetric matrices					
01	3.09	5.04	7.47	15.62	16.15	5.09	7.58	**16.93**	16.09	16.78
02	4.89	2.20	4.97	12.77	13.12	2.19	4.95	12.89	13.28	**13.73**
03	1.22	1.81	3.25	13.62	13.29	1.84	3.26	13.54	**13.87**	13.69
04	21.17	18.89	14.72	7.68	24.46	22.66	23.37	24.83	25.21	**25.24**
05	3.33	4.42	5.98	10.10	10.50	4.53	5.98	**10.52**	10.47	10.44
06	2.27	3.64	3.02	4.56	5.66	3.56	3.27	5.49	5.08	**5.73**
07	3.10	3.04	5.20	6.76	7.98	3.06	5.15	7.38	6.71	**8.38**
08	2.82	2.80	8.31	17.07	**18.46**	2.76	8.44	18.05	17.40	18.10
09	3.48	3.49	12.75	16.10	16.17	3.47	12.54	16.08	16.27	**16.34**
10	2.56	2.80	10.98	13.20	13.49	2.78	10.98	**13.55**	13.45	13.47
11	14.70	13.31	10.62	8.29	24.10	19.23	20.83	24.60	24.71	**24.82**
12	5.54	5.47	10.30	12.94	12.93	5.44	10.06	12.77	**13.18**	13.04
13	15.37	8.29	11.47	7.89	10.27	13.24	12.53	**15.98**	15.28	**15.98**
14	1.42	1.84	2.70	9.03	10.46	1.85	2.73	9.59	10.33	**10.64**
15	1.39	1.69	2.72	10.43	10.79	1.69	2.70	10.57	10.95	**11.02**
16	2.98	4.90	7.47	**15.78**	15.23	4.94	7.55	15.67	15.33	15.67
17	1.97	2.72	5.11	13.01	12.86	2.74	5.04	13.12	**13.26**	13.13
18	3.73	3.40	6.33	8.82	9.26	3.40	6.23	9.09	3.37	**9.6**
19	1.25	1.88	3.20	13.54	13.22	1.89	3.24	13.46	**13.8**	13.67
20	5.29	6.46	8.02	11.56	11.91	6.36	8.21	**12.03**	11.68	11.87
21	12.69	13.61	13.04	13.78	19.44	14.68	16.59	18.64	15.09	**20.29**
22	19.35	11.38	13.62	5.17	21.30	21.43	20.94	22.40	22.93	**23.63**
23	6.70	12.11	5.14	10.70	23.64	15.37	10.78	23.52	**24.73**	24.61
24	10.99	12.05	11.01	12.01	28.88	22.40	16.53	30.34	29.83	**30.96**
Avg.	4.29	4.67	6.84	10.68	14.10	5.21	7.80	14.33	13.60	**14.75**
# bests	0	0	0	1	1	0	0	5	5	13
					Square nonsymmetric matrices					
25	2.70	3.64	4.40	6.57	7.17	3.61	4.17	6.30	5.55	**7.31**
26	2.84	3.79	4.53	6.70	7.01	3.56	4.54	7.05	5.52	**7.27**
27	2.81	3.93	4.59	6.45	6.30	3.83	4.40	6.71	5.36	**7.13**
28	15.59	19.25	15.57	20.76	15.66	**21.72**	17.35	20.01	20.99	21.23
29	5.63	2.43	7.77	12.56	14.24	2.48	7.32	12.10	14.26	**14.62**
30	1.82	2.02	2.48	2.42	3.34	2.01	2.27	2.80	**4.32**	4.01
31	36.17	34.31	35.74	33.50	32.07	36.64	36.38	36.37	**37.09**	36.72
32	21.58	14.34	14.35	12.42	23.69	14.73	23.16	**25.12**	24.62	24.86
33	4.42	4.23	5.05	4.77	5.39	4.53	4.51	5.22	5.80	**6.14**
34	4.82	4.16	5.68	5.55	5.19	4.45	4.25	4.80	5.72	**6.58**
35	27.04	19.37	30.93	23.49	26.71	24.03	33.67	32.74	30.06	**34.44**
36	19.98	12.05	22.17	15.44	24.61	16.73	26.69	**27.04**	26.16	26.71
37	15.16	15.46	14.40	14.12	15.25	15.34	16.15	15.55	16.32	**16.7**
38	6.81	6.24	6.75	6.11	**9.27**	6.85	5.31	6.20	8.36	9.26
39	2.90	4.78	3.70	7.99	10.55	4.93	3.38	9.87	10.57	**10.65**
40	4.56	4.35	2.70	2.48	6.44	4.49	4.89	5.22	6.62	**6.96**
41	1.17	0.86	1.15	0.79	1.25	1.54	1.04	1.50	1.62	**1.97**
Avg.	6.39	6.02	7.12	7.61	9.47	6.59	7.33	9.33	9.86	**10.79**
# bests	0	0	0	0	1	1	0	2	2	11
					Rectangular matrices					
42	9.47	5.88	8.58	9.51	10.03	6.81	8.42	10.73	10.84	**11.71**
43	9.11	5.98	7.37	10.98	12.48	6.16	7.29	10.94	11.51	**12.79**
44	8.35	6.36	6.67	7.88	9.14	7.84	6.47	9.75	10.47	**14.14**
45	4.79	4.93	5.35	5.71	5.93	5.29	5.33	**6.26**	5.81	6.25
46	2.46	2.78	2.76	3.51	3.41	2.82	3.26	**4.36**	3.51	3.76
47	3.14	3.73	4.07	4.69	5.12	3.68	3.84	5.10	3.67	**5.65**
48	2.03	2.37	1.60	2.22	3.25	2.71	1.14	2.63	3.50	**3.91**
49	14.95	14.79	15.13	15.29	**23.74**	14.92	22.58	23.61	23.41	23.56
50	12.54	6.64	6.10	5.23	12.50	8.01	7.97	9.98	12.56	**14.56**
51	3.58	3.72	4.16	4.25	4.97	4.05	3.92	4.74	**5.31**	5.10
52	11.44	11.56	11.04	13.27	14.33	13.12	8.21	12.26	17.27	**17.83**
53	3.56	9.40	3.68	10.47	10.68	9.82	3.33	10.57	**11.82**	11.20
54	4.30	10.73	3.42	10.66	11.52	10.04	3.28	11.15	**12.35**	11.33
55	3.96	10.11	3.53	10.46	11.31	10.17	3.38	11.29	**12.02**	11.27
56	15.61	16.21	13.66	13.65	15.26	16.27	15.84	16.91	16.03	**17.23**
57	4.86	9.81	4.74	8.96	7.80	10.12	4.18	**10.14**	9.87	9.47
58	8.90	8.09	7.76	7.40	10.10	9.02	8.60	9.86	11.15	**11.25**
59	19.08	14.67	18.95	14.70	15.46	20.00	18.47	19.68	**20.36**	20.34
60	10.82	10.21	11.23	11.52	**14.82**	11.43	11.04	13.88	13.92	14.69
Avg.	6.58	7.26	6.09	7.98	9.41	7.87	6.07	9.47	9.89	**10.54**
# bests	0	0	0	0	2	0	0	3	5	9
					Overall					
Avg.	5.50	5.77	6.67	8.85	11.08	6.35	7.08	11.13	11.22	**12.13**
# bests	0	0	0	1	4	1	0	10	12	33

TABLE 5: Average performance results (in Gflop/s) for all methods and all possible blocking sizes on Xeon Phi

Method	1x1	1x8	2x4	4x2	8x1	Best	Vec-Avg
BL	4.12	3.70	4.57	4.70	4.36	5.50	4.38
Graph Methods							
SG	4.55	4.69	5.16	5.01	4.46	5.77	4.88
TG	5.11	4.82	5.74	5.86	5.37	6.67	5.50
SG+TG	6.26	7.10	8.15	7.90	7.09	8.85	7.63
STG	6.85	8.45	9.92	9.67	8.36	11.08	9.26
Hypergraph Methods							
SH	4.79	5.14	5.62	5.42	4.75	6.35	5.30
TH	5.06	4.77	6.02	6.41	6.01	7.08	5.85
SH+TH	6.84	8.61	10.02	9.95	8.73	11.13	9.46
SH+THline	6.85	8.61	10.28	9.98	8.58	11.22	9.57
STH	7.11	9.30	11.00	10.85	9.14	12.13	10.19



(a) SpMV times.



(b) Cache misses.

Fig. 12: Performance profiles for the Xeon processor.

methods, there is no computational dependency between the input and output vectors of the individual SpMV operations. This feature naturally holds for repeated SpMV operations that involve the same rectangular matrix in several applications (e.g., LSQR method [19] used for solving the least squares problem and interior point method used for the LP problems via iterative solution of normal equations) since input and output vectors are of different dimensions. So, for the methods that involve repeated SpMV of nonsymmetric and rectangular matrices, there is no need for reordering the y-vector entries after the SpMV.

In accordance with the above discussion, we consider the y-to-x-vector reordering only for the symmetric matrices. All our methods require y-to-x reordering for conformal x-y

ordering except SG+TG and SH+TH. This is because, for symmetric matrices, the similarity graph models in the SG and TG are exactly the same and the hypergraph models in SH and TH are also exactly the same. So, we can easily apply the conformal x-y (row-column) reordering by utilizing the partition obtained by either SG or TG in SG+TG and by either SH or TH in SH+TH. Here, we propose and develop a scheme for reducing the y-to-x reordering overhead in the powerful STH method, whereas the same scheme can easily be utilized in STBG. Recall that in the row/column reorderings obtained by the discussed partitioning methods, the rows/columns corresponding to vertices in a part are reordered arbitrarily. In the proposed scheme, in each part of a partition, for each pair of vertices that represent the row and column with the same index, the respective y- and x-vector entries are ordered conformably, so only non-conformal y-vector entries need to be reordered.

Experimental results on the symmetric matrix dataset show that, the STH method that utilizes the proposed y-to-x reordering scheme performs much better (22% on average) than the STH method that utilizes the arbitrary y-to-x reordering scheme. Despite this performance improvement, STH utilizing the proposed reordering scheme performs worse than SH+TH (13.28 vs 15.48 Gflop/s on average), whereas it performs better than SH+TH only for very sparse matrices (e.g., `ca2010`, `delaunay_18`, and `ny2010`). We refer the reader to the supplemental material for more detailed discussion and results.

8 Related Work

Reordering is used in the literature to improve data locality for irregular applications such as molecular dynamics [20], [21], [22] and sparse linear algebra [13], [14], [23], [27], [28], [29], [31]. Al-Furaih and Ranka [20] use MeTiS and a breadth-first-search (BFS) based reordering algorithm to reorder data elements for unstructured iterative applications. Han and Tseng [21] propose a low-overhead graph partitioning algorithm (Gpart) for data reordering (spatial locality).

For irregular applications, Strout and Hovland [22] propose graph and hypergraph models for data and iteration reordering. They use different reordering heuristics to traverse the graph or hypergraph models including Consecutive Packing (CPACK) [24] and Breadth-First Search. They also use Gpart [25] to partition the graph models and PaToH [11] to partition the hypergraph models.

On exploiting locality in sequential SpMV, Temam and Jalby [26] investigate effects of metrics based on matrix properties, cache size and line size on the number of cache misses. They propose a probabilistic model to estimate the number of cache misses and hits. They conclude that data hit ratio is the lowest while accessing x-vector entries and can be increased via reordering techniques. In our work, we also target reducing misses due to accessing x-vector entries and we report the actual number of cache misses (Table A.3 in the supplemental material) instead of estimations.

For sequential SpMV, Toledo [27] uses several bandwidth reduction techniques including Cuthill McKee (CM), Reversed CM (RCM) and top-down graph partitioning for reordering matrices to reduce cache misses. White and Sadayappan [28] also use the top-down graph partitioning tool MeTiS [10] to reorder sparse matrices. Pinar and Heath [29] use a spatial graph model and a formulation of traveling salesperson problem (TSP) for obtaining 1×2 blocks to halve the indexing overhead. Yzelman and Bisseling [13] propose a row-net hypergraph model to exploit spatial locality primarily and temporal locality secondarily. Akbudak et al. [14] propose a column-net hypergraph model to exploit temporal locality primarily and spatial locality secondarily. One-dimensional matrix partitioning is used in all of the above-mentioned reordering methods that are based on graph and hypergraph partitioning. Yzelman and Bisseling [23] and Akbudak et al. [14] propose reordering methods based on two-dimensional matrix partitioning. The DSBD method proposed in [23] permutes the matrix into doubly separated block diagonal form, whereas the $\mathrm{sHP}_{\mathrm{eRCN}}$ method proposed in [14] permutes the matrix into doubly-bordered block diagonal (DB) form, both through partitioning a fine-grain hypergraph model of which size is significantly larger than our proposed models.

On exploiting locality in parallel SpMV on shared-memory architectures, Williams et al. [30] propose 17 different optimizations in three categories (i.e., code, data structure and parallelism) for three different CPU architectures. These optimizations include but are not limited to cache blocking, using SIMD instructions, software prefetching, auto-tuning and exploiting process & memory affinity.

Yzelman and Roose in [6] combine several matrix reordering methods based on hypergraph partitioning and space filling curves to improve locality on shared memory architectures. RCM is used in [31] for bandwidth reduction of sparse matrix A on the Xeon Phi coprocessor. For sparse matrix-vector and matrix-transpose-vector multiplication (SpMMTV), which contains two consecutive SpMVs, Karsavuran et al. [32] utilize hypergraph models for exploiting temporal locality on Xeon Phi.

9 Conclusion

We proposed bipartite and hypergraph partitioning based methods that aim at exploiting spatial and temporal localities simultaneously for efficient parallelization of SpMV operations on many-core architectures. The experimental results on the Xeon Phi and Xeon processors showed that the proposed spatiotemporal methods for simultaneous row and column reordering significantly performed better than the methods that exploit either spatial or temporal locality. The experimental results also showed that the proposed spatiotemporal methods significantly benefit more from vectorization compared to the other methods. Among the proposed methods, hypergraph-based methods were found to produce better SpMV performance with respect to their bipartite graph counterparts.

Acknowledgment

This work was partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant EEEAG-115E212.

References

[1] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao *et al.*, "The Sunway Taihulight supercom-

puter: system and applications," *Science China Information Sciences*, vol. 59, no. 7, p. 072001, 2016.

[2] K. Akbudak and C. Aykanat, "Exploiting locality in sparse matrix-matrix multiplication on many-core architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 8, pp. 2258–2271, 2017.

[3] D. R. Liu and S. Shekhar, "Partitioning similarity graphs: A framework for declustering problems," *Information Systems*, vol. 21, no. 6, pp. 475 – 496, 1996.

[4] C. Konstantopoulos, B. Mamalis, G. Pantziou, and D. Gavalas, "Efficient parallel text retrieval techniques on bulk synchronous parallel (BSP)/coarse grained multicomputers (CGM)," *The Journal of Supercomputing*, vol. 48, no. 3, pp. 286–318, 2009.

[5] A. N. Yzelman, D. Roose, and K. Meerbergen, "Chapter 27 - Sparse Matrix-Vector Multiplication: Parallelization and Vectorization," in *High Performance Parallelism Pearls*, J. Reinders and J. Jeffers, Eds. Morgan Kaufmann, 2015, pp. 457–476.

[6] A. N. Yzelman and D. Roose, "High-level strategies for parallel shared-memory sparse matrix-vector multiplication," *IEEE Trans. Parallel Distributed Systems*, vol. 25, no. 1, pp. 116–125, 2014.

[7] A. N. Yzelman, "Generalised vectorization for sparse matrix-vector multiplication," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, ser. IA3 '15, 2015, pp. 6:1–6:8.

[8] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. Parallel Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.

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

[10] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[11] U. V. Catalyürek and C. Aykanat, "PaToH: a multilevel hypergraph partitioning tool, version 3.0," *Bilkent University, Department of Computer Engineering, Ankara*, vol. 6533, 1999.

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

[13] A. N. Yzelman and R. H. Bisseling, "Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 3128–3154, 2009.

[14] K. Akbudak, E. Kayaaslan, and C. Aykanat, "Hypergraph partitioning based models and methods for exploiting cache locality in sparse matrix-vector multiplication," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. C237–C262, 2013.

[15] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010.

[16] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[17] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003, vol. 82.

[18] R. W. Freund and N. M. Nachtigal, "QMR: a quasi-minimal residual method for non-hermitian linear systems," *Numerische Mathematik*, vol. 60, no. 1, pp. 315–339, Dec 1991.

[19] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM transactions on mathematical software*, vol. 8, no. 1, pp. 43–71, 1982.

[20] I. Al-Furaih and S. Ranka, "Memory hierarchy management for iterative graph structures," in *Parallel Processing Symposium, IPPS/SPDP*, Mar 1998, pp. 298–302.

[21] H. Han and C.-W. Tseng, "Exploiting locality for irregular scientific codes," *IEEE Trans. Parallel Distributed Systems*, vol. 17, no. 7, pp. 606–618, 2006.

[22] M. M. Strout and P. D. Hovland, "Metrics and models for reordering transformations," in *Proceedings of the 2004 workshop on Memory system performance*. ACM, 2004, pp. 23–34.

[23] A. N. Yzelman and R. H. Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication," *Parallel Computing*, vol. 37, no. 12, pp. 806 – 819, 2011, 6th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'10).

[24] C. Ding and K. Kennedy, "Improving cache performance in dynamic applications through data and computation reorganization at run time," in *Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation*, ser. PLDI '99, 1999, pp. 229–241.

[25] H. Han and C.-W. Tseng, *Languages, Compilers, and Run-Time Systems for Scalable Computers: 5th International Workshop, LCR 2000 Rochester, NY, USA, May 25–27, 2000 Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, ch. A Comparison of Locality Transformations for Irregular Codes, pp. 70–84.

[26] O. Temam and W. Jalby, "Characterizing the behavior of sparse algorithms on caches," in *Proceedings Supercomputing'92*. Minn., MN: IEEE, Nov. 1992, pp. 578–587.

[27] S. Toledo, "Improving the memory-system performance of sparse-matrix vector multiplication," *IBM Journal of Research and Development*, vol. 41, no. 6, pp. 711–725, Nov 1997.

[28] J. B. White and P. Sadayappan, "On improving the performance of sparse matrix-vector multiplication," in *Proc. Int. Conf. High Perform. Comput.*, Dec 1997, pp. 66–71.

[29] A. Pinar and M. T. Heath, "Improving performance of sparse matrix-vector multiplication," in *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, ser. SC '99, 1999.

[30] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178 – 194, 2009, revolutionary Technologies for Acceleration of Emerging Petascale Applications.

[31] E. Saule, K. Kaya, and U. V. Çatalyürek, *Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 559–570.

[32] M. O. Karsavuran, K. Akbudak, and C. Aykanat, "Locality-aware parallel sparse matrix-vector and matrix-transpose-vector multiplication on many-core processors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1713–1726, June 2016.

Nabil Abubaker received the BS degree in computer engineering from An-Najah National University, Nablus, Palestine, and the MS degree from Bilkent University, Ankara, Turkey where he is currently pursuing his PhD degree, both in computer engineering. His research interests include parallel computing, high-performance computing and locality exploiting methods for scientific irregular applications.

Kadir Akbudak received the BS degree from Hacettepe University, Ankara, Turkey, and the MS and PhD degrees from Bilkent University, Ankara, Turkey, all in computer science. He is currently working as a postdoctoral researcher in the Extreme Computing Research Center at KAUST.

Cevdet Aykanat received the BS and MS degrees from Middle East Technical University, Turkey, both in electrical engineering, and the PhD degree from Ohio State University, Columbus, in electrical and computer engineering. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Turkey, where he is currently a professor. He has served as an Associate Editor of IEEE Transactions of Parallel and Distributed Systems between 2009 and 2013.