REORDERING METHODS FOR EXPLOITING SPATIAL AND TEMPORAL LOCALITIES IN PARALLEL SPARSE MATRIX-VECTOR MULTIPLICATION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Nabil AbuBaker

August 2016

Reordering Methods for Exploiting Spatial and Temporal Localities in Parallel Sparse Matrix-Vector Multiplication
By Nabil AbuBaker
August 2016

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Cevdet Aykanat(Advisor)

Mustafa Özdal

Murat Manguoğlu

Approved for the Graduate School of Engineering and Science:

Levent Onural
Director of the Graduate School

ABSTRACT

REORDERING METHODS FOR EXPLOITING SPATIAL AND TEMPORAL LOCALITIES IN PARALLEL SPARSE MATRIX-VECTOR MULTIPLICATION

Nabil AbuBaker
M.S. in Computer Engineering
Advisor: Cevdet Aykanat
August 2016

Sparse Matrix-Vector multiplication (SpMV) is a very important kernel operation for many scientific applications. For irregular sparse matrices, the SpMV operation suffers from poor cache performance due to the irregular accesses of the input vector entries. In this work, we propose row and column reordering methods based on Graph partitioning (GP) and Hypergraph partitioning (HP) in order to exploit spatial and temporal localities in accessing input vector entries by clustering rows/columns with a similar sparsity pattern close to each other. The proposed methods exploit spatial and temporal localities separately (using either rows or columns of the matrix in a GP or HP method), simultaneously (using both rows and column) and in a two-phased manner(using either rows or columns in each phase). We evaluate the validity of the proposed models on a 60-core Xeon Phi co-processor for a large set of sparse matrices arising from different applications. The performance results confirm the validity and the effectiveness of the proposed methods and models.

Keywords: Sparse matrix-vector multiplication, graph model, hypergraph model, spatiotemporal, spatial locality, temporal locality, xeon phi, matrix reordering, parallel SpMV.

ÖZET

PARALEL SEYREK MATRİS VEKTÖR ÇARPIMINDA UZAYSAL VE ZAMANSAL YERELLİĞİ KULLANMAK İÇİN SIRALMA YÖNTEMLERİ

Nabil AbuBaker
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Cevdet Aykanat
Ağustos 2016

Seyrek matris-vektör çarpımı (SyMV), bilimsel uygulamalarda yaygın olarak kullanılan önemli bir çekirdek işlemdir. Düzensiz seyrek matrislerde SyMV işlemi, girdi vektör elemanlarına düzensiz erişim gerçekleştirdiği için önbellek başarımının düşük olmasına neden olmaktadır. Bu çalışmada, SyMv işleminde yüksek başarım elde etmek amacıyla, girdi-vektör elemanlarının erişiminde uzaysal ve zamansal yerellikten yararlanmak için, çizge ve hiperçizge bölümleme yöntemlerine dayanan matris satır ve sütun sıralama yöntemleri önerilmektedir. Bu yöntemler, seyreklik örüntüsü benzer olan matris satır ve sütunlarını gruplayarak birbirlerine yakın sıralamaktadırlar. Önerilen çizge ve hiperçizge bölümleme tabanlı uzaysal ve zamansal yerellik sağlama yöntemleri, tek aşamada ayrı ayrı kullanılabileceği gibi, tek veya iki aşamada birlikte de kullanılabilmektedirler. Önerilen yöntemlerin başarımı, 60 çekirdekli Xeon-Phi işlemcide, çeşitli uygulamalarda ortaya çıkan geniş bir matris kümesi kullanılarak denenmiştir. Deney sonuçları, önerilen yöntemlerin geçerlilik ve etkinliğini doğrulamaktadırlar.

Anahtar sözcükler: Seyrek matris-vektör çarpımı, çizge modeli, hiperçizge modeli, uzaysal yerellik, zamansal yerellik, xeon phi, matris sıralama, paralel SyMV .

Acknowledgement

I would like to thank to my supervisor Prof. Cevdet Ayakant for his support, guidance, patience and lovely spirit during our work on research. I would also like to thank him for his novel idea which led to to the development of this thesis.

A special thanks also goes to my friend Kadir Akbudak, who helped me throughout my research and was always patient for my endless questions.

Finally, I would like to express my deepest gratitude to my parents, my sisters and my lovely fiancé for their endless love, support and patience for my absence.

To the greatest women I've ever known, my mother ...

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Sparse Matrix-Vector Multiplication (SpMV) is vital kernel operation for many scientific computing applications. It is formulated as $y = Ax$, where y is the output vector, A is the sparse matrix and x is the input vector. SpMV has always suffered from two main issues: irregular data accesses to the input vector and limited bandwidth. The first issue often prevents the SpMV operation from utilizing the modern hierarchical memory structure, even when parallelizing the SpMV operation the difficulty might increase on shared-memory NUMA[1]architectures and distributed memory architecture.

In this thesis, we propose row/column reordering methods in order to exploit spatial and temporal localities in accessing input vector entries by clustering rows/columns that use the same or similar nonzeros/nonzero pattern close to each other. We propose Graph partitioning (GP) and Hypergraph partitioning (HP) based methods along with heuristic-based methods in order to exploit spatial and temporal localities separately (using either rows or columns of the matrix in a GP or HP method), simultaneously (using both rows and column) and in a two-phased manner(using either rows or columns in each phase).

[1]Non-Uniform Memory Access Architecture

The rest of this thesis is organized as follows: Chapter 2 explores some preliminaries useful to understand the scope of this thesis. Earlier and related work are presented in Chapter 3. Chapter 4 presents the proposed 1D methods, consisting of GP and HP based methods. We continue in Chapter 5 presenting the 2D methods, including simultaneous, heuristic-based and two-phase methods. The experiments for validating and evaluating the proposed methods are shown in Chapter 6. Finally, a conclusion and recommendations for future work are given in Chapter 7.

Chapter 2

Background

2.1 Matrix-Vector Multiplication

In this thesis, we consider the matrix-vector multiply of the form $y = A.x$, where A is a sparse matrix, x is the input vector and y is the output vector, that can be computed serially as:

$$\forall i : 0 \leq i < m : y_i = \sum_{j=0}^{n-1} a_{i,j} x_j$$

where m and n, respectively, denote the number of rows and columns of the matrix A.

2.2 Sparse Matrices

A matrix is said to be sparse if it is mostly populated by zero elements. In a sparse matrix A, the ratio of the nonzeros to the total size of the matrix A ($\frac{nnz(A)}{m.n}$) is much smaller than the ratio of the zero elements to $m.n$.

Figure 2.1 shows a sample sparse matrix of the size 22400×47185 and have

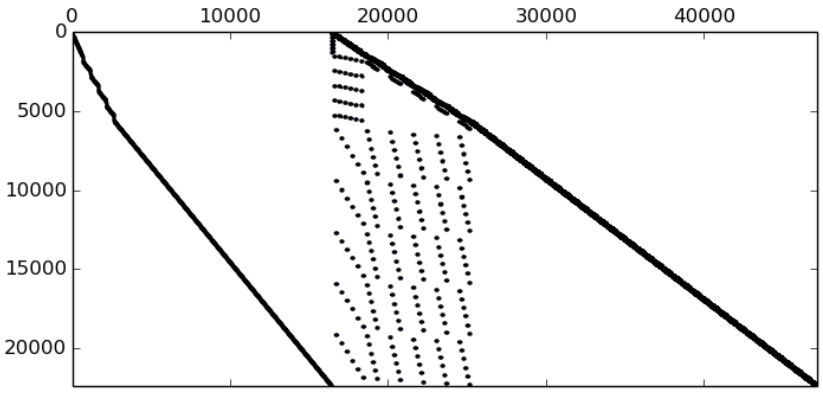


Figure 2.1: Sample sparse matrix fxm4_6

265442 nonzero elements.

In case of sparse matrix-vector multiplication, many running time and memory access optimizations are possible due to the fact that we need to perform multiplication and addition operations only on the nonzero elements of the sparse matrix.

2.3 Storage Schemes for Sparse Matrices

In this section, we will briefly review some storage schemes used to store the sparse matrix and have a role in efficient SpMV operation. The following sparse matrix is used to explain the storage schemes in this section.

$$M = \begin{bmatrix} 1 & & 0.7 & 5 \\ 0.5 & 0.6 & 3 & 0.22 \\ 0.1 & 0.05 & 0.13 & \end{bmatrix}$$

2.3.1 Coordinate Scheme (COO)

This format is simple, it stores the sparse matrix using three arrays i, j, v each of which has the length of number of nonzeros in the sparse matrix (nnz). The

i array stores the row indices of the nonzeros in v, while the j array stores the column indices. For example, consider the COO representation of the sample matrix M in Figure 2.2, the nonzero value 0.22 in the 6^{th} index of v has row index $i[6] = 2$ and column index $j[6] = 4$

Figure 2.2: COO storage scheme for the sample matrix M

i	1	1	1	2	2	2	2	3	3	3
j	1	3	4	1	2	3	4	1	2	3
v	1	0.7	5	0.5	0.6	3	0.22	0.1	0.05	0.13

The computation of SpMV of the form $y = Ax$ using COO format is straight forward as in algorithm 1. Note that in this scheme is not efficient during SpMV operation since it incurs irregular accesses to both input and output vector entries.

Algorithm 1 COO-Based SpMV

Require: Sparse matrix stored in COO format, input vector x, output vector y
  $r_{index} \leftarrow i[k]$
  $c_{index} \leftarrow j[k]$
  for $k = 0$ **to** $k = nnz - 1$ **do**
    $y[r_{index}] = y[c_{index}] + v[k].x[c_{index}]$
  end for

2.3.2 Compressed Row Storage (CRS)

CRS is the most common storage scheme for sparse matrices. It stores the sparse matrix M in three arrays, the nonzeros array v which stores all nonzeros in the sparse matrix in row-major order, col_{index} array which stores the column indices for each nonzero, and the row pointer row_{ptr} array which stores the indices of the start and end of each row in the nonzeros (and columns) arrays.

Figure 2.4 shows how to store the sample matrix M in CRS scheme. The CRS scheme saves more storage than COO scheme, COO scheme requires $3 \times nnz$ storage units while CRS requires $2nnz + R + 1$, where R is the number of rows in the sparse matrix.

Figure 2.3: CRS storage scheme for the sample matrix M

row_{ptr}	0	3	7	10						
col_{index}	1	3	4	1	2	3	4	1	2	3
v	1	0.7	5	0.5	0.6	3	0.22	0.1	0.05	0.13

Algorithm 2 shows how to perform SpMV using CRS storage scheme. The sparse matrix is traversed in row-major order, which allows the exploitation of data locality and increase the memory efficiency of SpMV. More details on data locality during CRS-based SpMV are dicussed in section 2.6 .

Algorithm 2 CRS-Based SpMV

Require: CRS storage of sparse matrix, number of rows in sparse matrix(R), input vector x, output vector y

```
for k = 0 to k = R − 1 do
    tmp ← 0
    for l = row_ptr[k] to l = row_ptr[k + 1] − 1 do
        tmp ← tmp + v[l].x[col_index[l]]
    end for
    y[k] ← tmp
end for
```

2.3.3 Compressed Column Storage (CCS)

Similar to CRS, the CCS scheme stores the sparse matrix M in three arrays, the nonzeros array v, row_{index} which stores the row indices for each nonzero, and the column pointer col_{ptr} array which stores the indices of the start and end of each columns in the nonzeros (and rows) arrays.

The difference between CRS and CCS is that the latter stores the sparse matrix as if it is traversed in column-major order.

Figure 2.4 shows how to store the sample matrix M in CCS scheme.

Similar to CRS, the storage amount required to store CCS is less than COO. It require $2nnz + C + 1$ storage units.

Figure 2.4: CCS storage scheme for the sample matrix M

col_{ptr}	0	3	5	8	10

row_{index}	1	2	3	2	3	1	2	3	1	2

v	1	0.5	0.1	0.6	0.05	0.7	3	0.13	5	0.22

Algorithm 3 shows how to perform SpMV using CCS storage scheme. The sparse matrix is traversed in column-major order, which allows the exploitation of data locality and increase the memory efficiency of SpMV.

Algorithm 3 CCS-Based SpMV

Require: CCS storage of sparse matrix, number of columns in sparse matrix(C), input vector x, output vector y
  for $k = 0$ **to** $k = C - 1$ **do**
    $tmp \leftarrow 0$
    for $l = col_{ptr}[k]$ **to** $l = col_{ptr}[k+1] - 1$ **do**
      $tmp \leftarrow tmp + v[l].x[row_{index}[l]]$
    end for
    $y[k] \leftarrow tmp$
  end for

2.4 Graph Partitioning

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. An edge $e_{i,j} \in \mathcal{E}$ with weight $w(e_{i,j})$ connects two distinct vertices v_i and v_j. Vertex $v_i \in \mathcal{V}$ with weight $w(v_i)$ is said to have a degree d_i if it connects d number of vertices. The set of vertices connected by vertex v_i is called the adjacency list of v_i and is defined as $Adj(v_i) \in \mathcal{V}$. A k-way partition $\Pi = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_K\}$ of $\mathcal{G}$ is said to be correct if the following conditions hold: Each part $\mathcal{P}_k, 1 \leq k \leq K$ is a nonempty subset of $\mathcal{V}$, parts are pairwise disjoint, i.e., $\mathcal{P}_k \cap \mathcal{P}_l = \emptyset$ for all $1 \leq k < l \leq K$, and union of all K parts is equal to the vertices set $\mathcal{V}$. A partition Π of $\mathcal{V}$ is said to be balanced if each part satisfies the balance criterion:

$$W_k \leq W_{avg}(1 + \varepsilon), for\ k = 1, 2, ..., K, \tag{2.1}$$

where W_k is the sum of weights of all vertices in part $\mathcal{P}_k$, mathematically: $W_k = \sum_{v_i \in \mathcal{P}_k} w(v_i)$, the average weight $W_{avg} = (\sum_{v_i \in \mathcal{V}} w(v_i))/K$ denotes the

weight of each part under perfect load balance condition, and ε denotes the maximum imbalance ratio allowed in each part. In a partition Π of $\mathcal{G}$, an edge $e_{i,j}$ is called a cut edge (external) if connects two different parts, and its called uncut edge (internal) if both vertices v_i and v_j are in the same part. The set of external edges is denoted as $\mathcal{E}_E$. One way to define the cost function of a partition Π is the cutsize, as:

$$cost(\Pi) = \sum_{e_{i,j} \in \mathcal{E}_E} w(e_{i,j}) \tag{2.2}$$

The graph partitioning problem can be defined as dividing the graph into two or more parts with the goal of minimizing the cutsize in Eq. 2.2 while the load balance objective defined in Eq. 2.1 is maintained. The graph partitioning problem is known to be NP-hard. So, several heuristic-based multilevel algorithms [1, 2, 3] have been proposed to partition a graph that led to successful multilevel tools such as Chaco [4] and MeTiS [5].

2.5 Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets, also called hyperedges, $\mathcal{N}$ connecting those vertices. A net (hyperedge) $n_i \in \mathcal{N}$ can connect two or more vertices, so each net can be seen as a set of vertices which is a subset of the hypergraph's vertex set $\mathcal{V}$. Vertices connected by a net n_i are called its $Pins$ as $Pins(n_i)$. The nets connected by vertex v_i are called its $nets$ set as $nets[v_i]$.

In a k-way partitioning Π of $\mathcal{H}$, a net n_i that connects at least one vertex in a part is said to connect that part. The set of parts connected by a net n_i is called con set as $con(n_i)$. Connectivity λ_i of a net n_i denotes the number of connected parts by this net, i.e, $|con(n_i)|$. A net n_i is said to be an *uncut* net (internal net) if it only connect one part, i.e., $\lambda_i = 1$, and its said to be a *cut* net (external net) if it connects more than one part, i.e. $\lambda_i > 1$. The set of external nets of a k-way partition Π is denoted as $\mathcal{N}_E$. The partitioning objective of the hypergraph is often to minimize the cutsize cost of the partition Π defined over the cut nets.

One good cost function is the $Connectivity - 1$ function as:

$$cost(\Pi) = \sum_{n_i \in \mathcal{N}_E} (\lambda_i - 1). \tag{2.3}$$

Also, the load balancing criterion in a k-way partition Π of $\mathcal{H}$ is similar to the graph case as in 2.1. Similar to the graph partitioning problem, the hypergraph partitioning problem is known to be NP-hard.

The multilevel paradigms, which cited in section 2.4, are also used to develop hypergraph partitioning tools such as hMetis [6] and PaToH [7].

2.6 Data Locality in Parallel SpMV

In this section, we discuss the feasibility of data locality in CRS-based SpMV.

Spatial Locality Spatial locality is achieved normally on the input matrix because the nonzeros in CRS-based SpMV are stored and accessed consecutively. Likewise, it is normally achieved in accessing y-vector entries for the same reason. In terms of irregular accesses of the x-vector entries, spatial locality can be improved by reordering the columns of the input matrix.

Temporal Locality In accessing nonzeros of the input matrix, temporal locality is not achievable beacuse each nonzero, and its indices, are accessed only onces. In accessing y-vector entries, it can be achieved in the register-level since all partial product results are summed up consecutively during the consecutive row-wise access of the nonzeros. In accessing the x-vector entries, temporal locality is achievable and can be improved by reordering the rows of the input matrix.

Regarding spatial and temporal localities in accessing x-vector entries, reordering rows and columns for improving data locality should be done with a criteria that clusters rows and columns with similar sparsity patterns next to each other.

Chapter 3

Related Work

Existing work on improving sparse matrix-vector multiplication includes, but not limited to, techniques that use graph-theoretic models and methods to permute the rows and columns of the input sparse matrix, strategies that control processing order of nonzeros and adapting efficient and cache-friendly storage schemes for the sparse matrix. In this Chapter, we present previous works that uses graph and hypergraph based models and methods to exploite spatial and temporal localities.

3.1 General Irregular Applications

The terms "data reordering" and "iteration/computation reordering" have been used to express exploiting spatial and temporal localities, respectively. Al-Furaih and Ranka [8] used Graph Partitioning and BFS(Breadth-First-Search) to reorder data entries (spatial locality) for unstructured iterative applications. They also use a hybrid approach in which they use GP to partition the vertices and inside each part they traverse the nodes using BFS.

Strout and Hovland [10] proposed graph and hypergrah models for data and iteration reordering. They propose a spatial locality graph that corresponds to a hypergraph with degree 2, i.e., each net connects two vertices, in which they use

reordering heuristics like Consecutive Packing (CPACK) [13] and Breadth-First Search to traverse the graph. They also propose a temporal locality hypergraph in which they use reordering heuristics like modified CPACK, called CPACK-Iter, modified BFS (called BFSIter). Also they use Hypergraph partitioning to generate a reordering.

Han and Tseng [9] proposed a low-overhead graph partitioning algorithm (Gpart) for reordering data entires (spatial locality) and a space-filling curve algorithm called Z-Sort for iteration reordering (temporal locality).

3.2 SpMV

3.2.1 Serial Methods

In SpMV, previous methods of general irregular applications can be utilized on Compressed Row Storage scheme (CRS) by translating "data reordering" to columns of sparse matrix and "iteration reordering" to rows of sparse matrix.

Toledo [15] has used several bandwidth reduction techniques including Cuthill McKee (CM), Reversed CM (RCM) to reorder matrices in order to reduce cache misses in SpMV.

White and Sadayappan [16] also used GP tool MeTiS [5] to reorder sparse matrices.

Pinar and Heath [17] proposed a spatial graph model, which they reduce to a well known traveling salesman problem (TSP), to improve the cache utilization.

Yzelman and Bisseling [18], propose a row-net hypergraph model to exploit spatial locality primarily. They use the partitioned hypergraph to reorder columns of the matrix accordingly and they reorder the rows of the matrix according to the relation between nets of the hypergraph and the resulting partitions,

i.e., depending whether the net is a cut net or an internal net they apply a heuristic to reorder the corresponding column. The resulting permuted sparse matrix is called separated block diagonal (SBD) from.

Akbudak et. al [11] proposed a column-net model and two row-column-net hypergraph models to exploit temporal locality as a primary objective. In their 2D schemes, they use the relation between the nets and resulting partitions to permute the matrix into K-way doubly-bordered block diagonal (DB) form.

Yzelman and Bisseling [12] propose a row-column-net hypergraph model to permute the sparse matrix into doubly separated block diagonal (DSBD) form.

3.2.2 Parallel Methods

In [19], Yzelman and Roose combined several matrix reordering methods based on hypergraph partitioning and space filling curves to improve data locality on shared memory architectures. They report that row distributed parallel SpMV based on 1D reordering by Hilbert curve performs better than fully distributed 2D reordering based on 2D, fine-grain, hypergraph partitioning SBD form. Saule et. al. [20] have used RCM to reduce the bandwidth of the sparse input matrix for better cache performance during SpMV on Xeon Phi coprocessor. Karsavuran et al. [21] utilized 1D hypergraph models for exploiting temporal locality in Sparse matrix-vector and matrix-transpose-vector multiplication (SpMMTV), which contains two consecutive SpMVs.

Chapter 4

1D Locality Exploiting Methods

In this chapter, we propose graph and hypergraph models that encode exploiting either spatial or temporal locality in one dimensional and one phase methods.

In order to improve data locality, the models proposed in this chapter are partitioned using graph or hypergraph partitioning tools into K parts and this partitioning is used to reorder the rows or columns of the sparse matrix. The K-way partitioning on the vertex set $\mathcal{V}$ of the model, $\Pi(\mathcal{V}) = \{\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_K\}$, can be used to reorder one of the two dimensions of the sparse matrix, i.e., either rows or columns, as follows: The set of rows/ columns corresponding to the vertices in $\mathcal{V}_{k-1}$ are reordered before the set of rows/ columns corresponding to the vertices in $\mathcal{V}_k$ for $k = 1, 2, ..., K-1$, while rows/columns inside part $\mathcal{V}_k$'s boundaries can be reordered arbitrarily assuming that inside a part all rows/ columns have the same closeness to each other. Of course this assumption is not necessarily valid especially for large part sizes, so more improvements can be done here in future work/implementation. Note that the other dimension of the matrix that had not been targeted by the partitioning model will stay on its original ordering. Figure 4.1 shows a sample 6×8 sparse matrix to be used in the examples throughout the upcoming chapters.

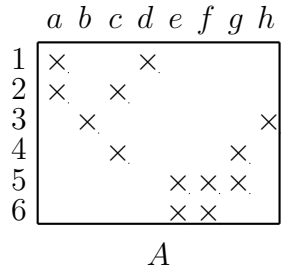
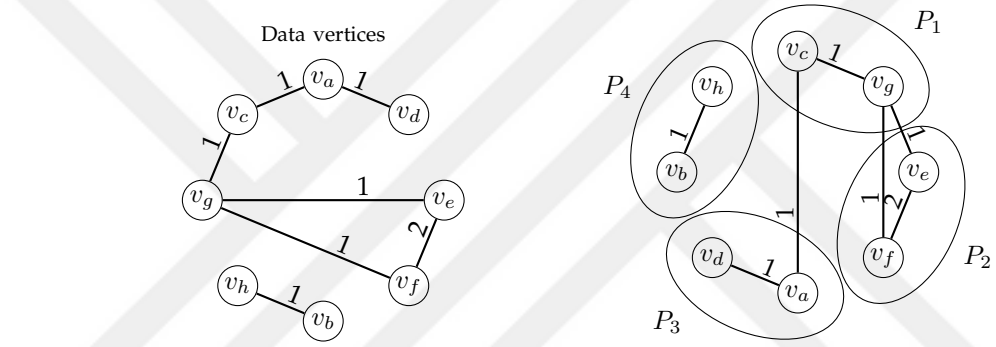


Figure 4.1: Sample matrix A.



(a) Spatial graph$\mathcal{G}^S(A)$. (b) 4-way partitioning of $\mathcal{G}^S(A)$.

Figure 4.2: Graph model $\mathcal{G}^S(A)$ of sample matrix A and its 4-way partitioning.

4.1 Exploiting Spatial Locality

In this section, we present graph and hypergraph partitioning models used in different methods in order to exploit spatial locality in parallel SpMV. In CRS-based SpMV, exploiting spatial locality requires the alignment of columns with similar sparsity pattern next to each other.

4.1.1 Spatial Graph (SG)

Model Definition

For a sparse matrix A, the access of x-vector entries by different inner-product tasks can be modeled as a graph $\mathcal{G}^S(A) = (\mathcal{V}^D, \mathcal{E})$ which can be called a similarity graph since it encodes the similarities between the sparsity patterns of the

corresponding columns. There exist a data vertex $v_i \in \mathcal{V}^D$ for each column c_i of A in $\mathcal{G}^S(A)$. Here and hereafter, calligraphic letters are used to denote sets, e.g., $\mathcal{V}$ and $\mathcal{E}$ denote the sets of vertices and edges, respectively. The superscripts "S" and "D" refer to spatial locality and data, respectively. Between two data vertices v_i and v_j in $\mathcal{G}^S(A)$, there exists an edge $e_{i,j} \in \mathcal{E}$ if columns c_i and c_j have a nonzero in at least one row in common. The weight $w(e_{i,j})$ associated with an edge $e_{i,j}$ is equal to the number of rows in common between columns c_i and c_j. That is,

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

Figure 4.2a shows the model $\mathcal{G}^S$ which is constructed from the sample matrix given in Figure 4.1.

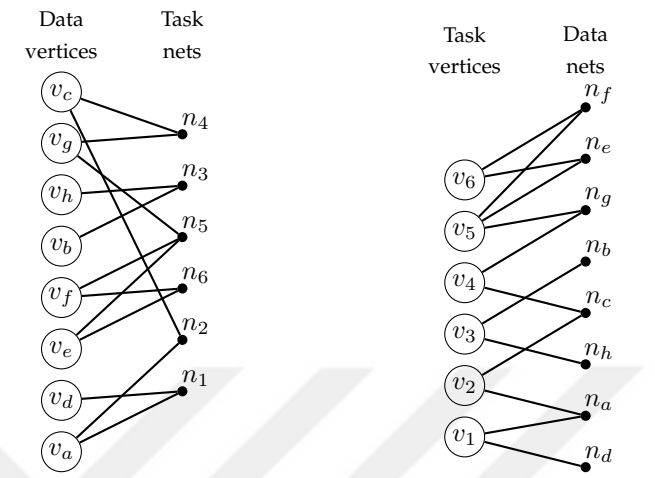
Validity Assessment

The cut edges in a K-way partition $\Pi(\mathcal{V}^D)$ of $\mathcal{G}^S$ can be used to model the number of cache misses that occur during the irregular access of x-vector due to loss of spatial locality, so minimizing the cutsize as in Eq. 2.2 corresponds to minimizing the number of cache misses.

A proof can be given as follows: Under the following assumptions:

- Each cache line accommodates Wwords.
- Each part in $\Pi(\mathcal{V}^D)$ contains exactly Wvertices.
- Columns correspond to vertices in a part are reordered consecutively.

A cut edge $e_{i,j}$ will incur at most most $w(e_{i,j})$ extra cache misses because during each of the $w(e_{i,j})$ inner-products of rows in common between columns c_i and c_j, either the line that contains x_i or the line that contains x_j may not be reused due to loss of spatial locality. An uncut edge $e_{h,\ell}$ with weight $w(e_{h,\ell})$ means that x-vector entries x_i and x_j, which are on the same cache line, are accessed together during the $w(e_{h,\ell})$ inner-product tasks and hence the line contains them is reused due to exploiting spatial locality.



(a) Spatial hypergraph $\mathcal{H}^S(A)$.

(b) Temporal hypergraph $\mathcal{H}^T(A)$.

Figure 4.3: Spatial and temporal hypergraphs models of sparse matrix A.

4.1.2 Spatial Hypergraph (SH)

Model Definition

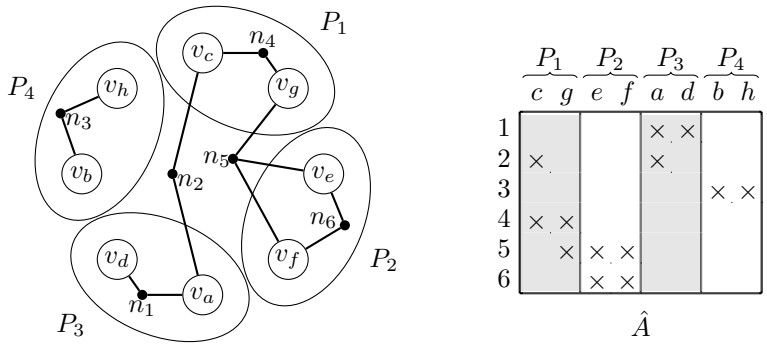
For a sparse matrix A, the access of x-vector entries by different inner-product tasks can be modeled as a hypergraph model $\mathcal{H}^S(A) = (\mathcal{V}^D, \mathcal{N}^T)$.

The vertex set of $\mathcal{H}^S$ consists of data vertices where each column c_j of A necessitates a vertex $v_j \in \mathcal{V}^D$. The nets set of $\mathcal{H}^S$ consists of task nets where each row r_j of A necessitates a net $n_i \in \mathcal{N}^T$. Net n_j connects data vertices corresponding to columns that have nonzeros in row r_i. That is,

$$Pins(n_i^T) = \{v_j^D : a_{i,j} \neq 0\}.$$

The nets and vertices in this model are associated with unit weights. This model is previously proposed in [18].

Figure 4.3a shows $\mathcal{H}^S$ model of the sample matrix A and taking net n_5 as an example, we can see that it 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.



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

(b) Reordered matrix $\hat{A}$.

Figure 4.4: Four-way partition $\Pi(\mathcal{V}^D)$ of the spatial hypergraph given in Figure 4.3a and matrix $\hat{A}$, which is obtained via reordering matrix A according to this partition.

Validity Assessment

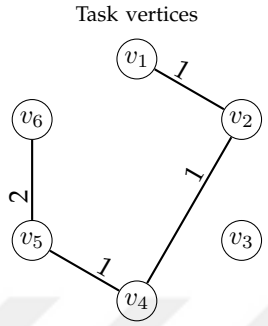
The cut nets in a K-way partition $\Pi(\mathcal{V}^D)$ of $\mathcal{H}^S$ can be used to model the number of cache misses that occur during the irregular access of x-vector due to loss of spatial locality, so minimizing the cutnets according to Eq. 2.3 corresponds to minimizing the number of cache misses.

A proof can be given as follows: Under the same assumptions mentioned in the proof of the SG model in the previous section, a cut net n_i means that the inner-product task corresponding to row i might not reuse the x-vector entries corresponding to vertices connected by n_i in different $|con(n_i)|$ parts.

An uncut net n_i tells us that the x-vector entries corresponding to vertices connected by net n_i will be reused during the inner-product task that corresponds to row r_i since they are reordered consecutively.

Figure 4.4 shows a four-way partition of $\mathcal{H}^S(A)$ model of the sample matrix given in Figure 4.1 and the permuted matrix $\hat{A}$. Note that row order of $\hat{A}$ matrix is kept as the original order.

Figure 4.5: Temporal graph $\mathcal{G}^T(A)$.



4.2 Exploiting Temporal Locality

In this section, we present graph and hypergraph partitioning models used in different methods in order to exploit temporal locality in parallel SpMV. Exploiting temporal locality in CSR-based SpMV requires the alignment of rows with similar sparsity pattern, which can be encoded as the pattern of accessing x-vector entires, next to each other.

4.2.1 Temporal Graph (TG)

Model Definition

For a matrix A, the similarities between inner product tasks (rows) in terms of their access pattern to the x-vector (number of shared columns) can be modeled as a graph $\mathcal{G}^T(A) = (\mathcal{V}^D, \mathcal{E})$ which can be called a similarity graph since it encodes the similarities between rows. Here an hereafter, the subscript "T" in the graph notion refers to temporal locality. The graph model $\mathcal{G}^T$ is constructed as follows, for each row r_i of the matrix A, there exists a "task" vertex $v_i \in \mathcal{V}$. There exists an edge $e_{i,j} \in \mathcal{E}$ between task vertices v_i and v_j if the rows corresponding to those vertices, i.e., r_i and r_j, have nonzeros in at least one column in common. The weight $w(e_{i,j})$ associated with an edge $e_{i,j}$ is equal to the number of shared

columns between between rows c_i and c_j. That is,

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

Figure 4.5 shows $\mathcal{G}^T$ model of the sample matrix given in Figure 4.1.

Validity Assessment

The cut edges in a K-way partition $\Pi(\mathcal{V}^D)$ of $\mathcal{G}^T$ can be used to model the number of cache misses that occur during the irregular access of x-vector due to loss of temporal locality, so minimizing the cutsize as in Eq. 2.2 corresponds to minimizing the number of cache misses. A proof can be given as follows: Under the following assumptions:

- The cache can fit only for x-vector entries that are used during processing tasks in the same part.
- The cache line accommodates only one word.
- The rows corresponding to the vertices in a part are reordered consecutively.

A cut edge $e_{i,j}$ will incur at most $w(e_{i,j})$ extra cache misses because during the inner-products corresponding to rows r_i and r_j, the $w(e_{i,j})$ x-vector entries used by tasks t_i and t_j may not be reused due to loss of temporal locality since the rows r_i and r_j are not reordered next to each other. An uncut edge $e_{h,\ell}$ with weight $w(e_{h,\ell})$ means that the $w(e_{i,j})$ x-vector entries used by both t_i and t_j will probably be reused since rows r_i and r_j are reordered consecutively.

4.2.2 Temporal Hypergraph (TH)

Model Definition

For a sparse matrix A, inner-product tasks (rows) requirement of different x-vector entries (columns) can be modeled as a hypergraph model

$\mathcal{H}^S(A) = (\mathcal{V}^D, \mathcal{N}^T)$.

The vertex set of $\mathcal{H}^S$ consists of task vertices where each row r_i of A necessitates a vertex $v_i \in \mathcal{V}^D$. The nets set of $\mathcal{H}^S$ consists of data nets where each column c_j of A necessitates a net $n_j \in \mathcal{N}^T$. Net n_j connects task vertices corresponding to rows that have nonzeros in column c_j. That is,

$$Pins(n_j^D) = \{v_i^T : a_{i,j} \neq 0\}.$$

In other words, net n_j encodes the fact that x-vector entry x_j corresponding to column c_j will be accessed during the inner products of $|Pins(n_j^D)|$ tasks in the task vertex set $Pins(n_j^D)$. In this model, we associate vertices and nets with unit weights. This model is previously proposed in [11]. Figure 4.3b shows $\mathcal{H}^T$ model of the sample matrix given in Figure 4.1.

Validity Assessment

The cut nets in a K-way partition $\Pi(\mathcal{V}^D)$ of $\mathcal{H}^T$ can be used to model the number of cache misses that occur during the irregular access of x-vector due to loss of temporal locality, so minimizing the cutnets according to Eq. 2.3 corresponds to minimizing the number of cache misses.

A proof can be given as follows: Under the same assumptions list used in the proof of $\mathcal{G}^T$ model, a cut net n_j will incur at most most $|Pins(n_j)|$ extra cache misses because during the inner-products in $Pins(n_j)$, the x-vector entry x_j used by those tasks may not be reused due to loss of temporal locality.

An uncut net n_j tells us that the x-vector entry x_j will be reused during the inner-product task in $Pins(n_j)$ since the rows corresponding to those tasks are reordered consecutively.

Chapter 5

2D Locality Exploiting Methods

In Chapter 4, we discuss the proposed methods for reordering either rows or columns to exploit spatial or temporal localities, respectively. In this chapter, we propose methods and models to exploit spatial and temporal localities by reordering rows and columns in two-dimensional manner. This chapter is organized as follows: first section presents heuristics-based reordering of rows/ columns which follows a 1D method from the previous chapter in order to reorder the dimension that has not been reordered by the 1D method. In section 2 we propose a spatiotemporal bipartite graph model and a hypergraph model which can be partitioned in one phase to obtain a reordering on both rows and columns. Finally in section 3 we present a two-phase method in which we reorder the sparse matrix to exploit spatial locality in the first phase and use the reordered matrix to exploit temporal locality in the second phase.

5.1 Using Heuristics with 1D Methods

Reordering methods in Chapter 4 target the exploitation of either spatial or temporal locality. In this section, we target the reordering of the dimension of the matrix (rows or columns) that had not been reordered after a 1D reordering

method from chapter 4 is applied.

Given a K-way partition Π on one dimension of the sparse matrix A, i.e., using either spatial or temporal models, in this section we show how to obtain a heuristic-based K-way partition $\hat{\Pi}$ on the other dimension of the sparse matrix in linear time.

Algorithm 4 Temporally-improved Spatial Reordering

Require: K-way partition vector on columns $CPart_{vec}$, Reordered Matrix $\hat{A}$
  $tmpBuf[K] \leftarrow \emptyset$
  $RPart_{vec} \leftarrow \emptyset$
  $ptmp \leftarrow 0$
  for each row r_i in $\hat{A}$ **do**
    for all column c_j have nonzero in r_i **do**
      $ptmp \leftarrow CPart_{vect}[cj]$
      $tmpBuf[ptmp] \leftarrow tmpBuf[ptmp] + 1$
    end for
    $RPart_{vec}[r_i] \leftarrow maxIndex(tmpBuf)$
    $tmpBuf[K] \leftarrow \emptyset$
  end for
  return $RPart_{vec}$

5.1.1 Temporally-Improved Spatial Methods (SG-H, SH-H)

Using a spatial method from Chapter 4 (Graph or Hypergraph based), we obtain a reordering on columns (data vertices) of sparse matrix A. From this reordering, we can derive a heuristic-based reordering for rows (task vertices) of sparse matrix A to further improve the temporal locality of the matrix.

Given a K-way partition Π on the columns of the sparse matrix A, a heuristic-based K-way partition $\hat{\Pi}$ on the rows of the sparse matrix is obtained, in linear time, as follows: for row r_i in a reordered matrix $\hat{A}$, assign r_i to the part $\hat{\mathcal{P}}_k$ of $\hat{\Pi}$ corresponds to part $\mathcal{P}_k$ of Π which minimizes the external connections of r_i to other parts, i.e., assign r_i to the part $\hat{\mathcal{P}}_k$ of $\hat{\Pi}$ corresponds to part $\mathcal{P}_k$ of Π that has maximum number of columns connected with r_i. Algorithm 4 shows

how to obtain heuristic-based partitioning vector on rows of matrix A given a partition on columns and a reordered matrix. In this algorithm, $RPart_{vec}$ with size equal to number of rows of matrix A is used to save the part number of row r_i as $RPart_{vec}[i]$, $CPart_{vec}$ with size equal to number of columns of matrix A is used for columns assignment to parts as in RP_{vec}, and $tmpBuf$ that has a size equal to number of parts K is used as a buffer during processing row r_i to know how many columns connected with r_i are in each part.

5.1.2 Spatially-Improved Temporal Methods (TG-H, TH-H)

Using a temporal method from Chapter 4 (Graph or Hypergraph based), we obtain a reordering on rows (task vertices) of sparse matrix A. Form this reordering, we can derive a heuristic-based reordering for columns (data vertices) of sparse matrix A to further improve the spatial locality of the matrix.

The problem definition and details of the method are similar to the previous section with changing the roles of rows and columns.

Algorithm 5 shows how to obtain a heuristic-based partitioning vector on columns of matrix A given a partition on rows and a reordered matrix. In this algorithm, $CPart_{vec}$ with size equal to number of columns of matrix A is used to save the part number of column c_j as $CPart_{vec}[j]$, $RPart_{vec}$ with size equal to number of rows of matrix A is used for rows assignment to parts as in $CPart_{vec}$, and $tmpBuf$ that has a size equal to number of parts K is used as a buffer during processing column c_j to know how many rows connected with c_j are in each part. Note that this algorithm processes the nonezeros in column-major order, i.e. it uses CCS scheme to store the sparse matrix.

Algorithm 5 Spatially-improved temporal Reordering

Require: K-way partition vector on rows $RPart_{vec}$, Reordered Matrix $\hat{A}$
  $tmpBuf[K] \leftarrow \emptyset$
  $CPart_{vec} \leftarrow \emptyset$
  $ptmp \leftarrow 0$
  for each column c_j in $\hat{A}$ **do**
    for all row r_i have nonzero in c_j **do**
      $ptmp \leftarrow RPart_{vect}[r_i]$
      $tmpBuf[ptmp] \leftarrow tmpBuf[ptmp] + 1$
    end for
    $CPart_{vec}[c_j] \leftarrow maxIndex(tmpBuf)$
    $tmpBuf[K] \leftarrow \emptyset$
  end for
  return $CPart_{vec}$

5.2 Simultaneous Methods

In this section, we propose a bipartite graph partitioning model called STG and a hypergraph partitioning model called STH in order to encode spatial and temporal localities simultaneously.

5.2.1 Bipartite Spatiotemporal Graph (STG)

Model Definition

For a matrix A, the mutual requirement between the x-vector entries and the inner product tasks can be modeled as a bipartite graph $\mathcal{G}^{ST}(A) = (\mathcal{V} = \mathcal{V}^D \cup \mathcal{V}^T, \mathcal{E})$. The vertex set of $\mathcal{G}^{ST}$ consists of data vertices and task vertices as follows: each row r_i of the sample matrix necessitates a task vertex $v_i \in \mathcal{V}^T$ and each column c_j necessitates a data vertex $v_j \in \mathcal{V}^D$. There is an edge $e_{i,j}$ between vertex v_i and v_j if there is a nonezero $a_{i,j}$ between row r_i and column c_j. We associate unit weights with both edges and vertices.

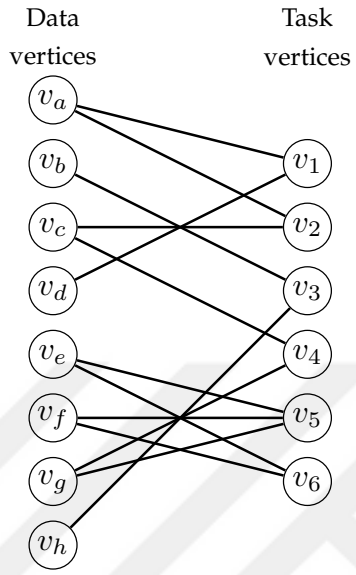


Figure 5.1: Bipartite Graph model $\mathcal{G}^{ST}(A)$ for exploiting spatial and temporal localities simultaneously.

Figure 5.1 shows the bipartite $\mathcal{G}^{ST}$ model of the sample matrix given in Figure 4.1. In the figure, the edges show the mutual requirements of the task vertices to the data vertices and vice versa.

Validity Assessment

The cut edges in a K-way partition $\Pi(\mathcal{V})$ of $\mathcal{G}^{ST}$ can be used to model the number of cache misses that occur during the irregular access of x-vector due to loss of spatial and temporal localities, so minimizing the cutsize as in Eq. 2.2 corresponds to minimizing the number of cache misses.

A proof can be given as follows: Under the following assumptions:

- Each cache line accommodates Wwords.
- Each part in $\Pi(\mathcal{V})$ contains exactly Wvertices.
- The cache can fit only for x-vector entries that are used during processing tasks in the same part.

- Columns correspond to vertices in a part are reordered consecutively.
- Rows correspond to vertices in a part are reordered consecutively.

A cut edge $e_{i,j}$ that connects task vertex v_i and data vertex v_j means two things:

1. The inner-product task t_i will not reuse the line contains data element x_j due to loss of spatial locality.
2. Data element x_j will not be reused by the inner-product task t_i due to loss of temporal locality.

While an uncut edge means the opposite, this proof does not cover the relation among tasks or among data entries. There is an indirect relation between task vertices that goes through data vertices as follows: minimizing the cut edges means clustering a task v_i^T with the data vertices it will use, i.e, the set $Adj(v_i^T)$, in the same part. This will encourage the task v_j^T with $\{Adj(v_i^T) \cap Adj(v_j^T)\} \neq \emptyset$ to be in the same part to decrease the cut edges during the K-way partitioning $\Pi(\mathcal{V})$.

5.2.2 Spatiotemporal Hypergraph (STH)

Model Definition

For a matrix A, the mutual requirement between the x-vector entries and the inner product tasks can be modeled as a hypergraph model $\mathcal{H}^{ST}(A) = (\mathcal{V} = \mathcal{V}^D \cup \mathcal{V}^T, \mathcal{N} = \mathcal{N}^T \cup \mathcal{N}^D)$.

The vertex set of $\mathcal{H}^{ST}$ consists of data vertices and task vertices as follows: each row r_i of the sample matrix necessitates a task vertex $v_i \in \mathcal{V}^T$ and each column c_j necessitates a data vertex $v_j \in \mathcal{V}^D$. The net set of $\mathcal{H}^{ST}$ consists of data nets and task nets as follows: each row r_i of the sample matrix necessitates a task net $n_i \in \mathcal{N}^T$ and each column c_j necessitates a data net $n_j \in \mathcal{N}^D$.

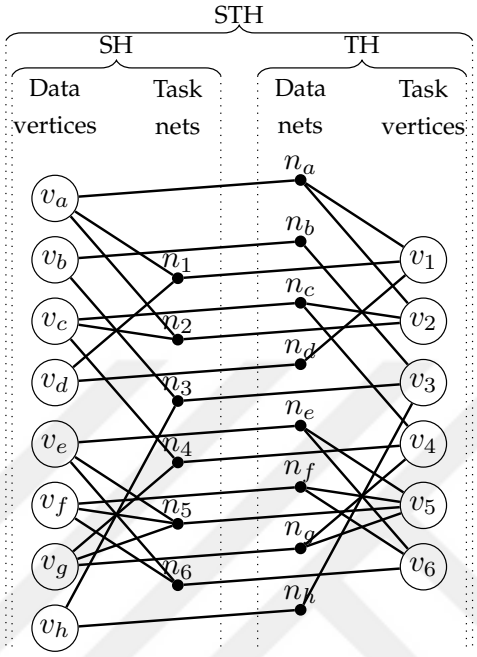


Figure 5.2: Hypergraph model $\mathcal{H}^{ST}(A)$ for exploiting spatial and temporal localities simultaneously.

$\mathcal{H}^{ST}(A)$ models the dependencies between task and data vertices as follows: a task net n_i^T connects data vertices corresponding to columns that have nonzeros in row r_i. It also has one more connection to vertex v_i which corresponds to row r_i. Data net n_j^D connects task vertices corresponding to rows that have nonzeros in column c_j. It also connects vertex v_j which corresponds to column c_j. That is,

$$\begin{aligned} Pins(n_j^D) &= \{v_i^T : a_{i,j} \neq 0\} \cup \{v_j^D\}. \\ Pins(n_i^T) &= \{v_j^D : a_{i,j} \neq 0\} \cup \{v_i^T\}. \end{aligned} \tag{5.1}$$

We associate unit weights with all nets and all vertices.

Figure 5.2 shows $\mathcal{H}^{ST}$ model of the sample matrix given in Figure 4.1. As seen in the figure, the $\mathcal{H}^{ST}$ model is build using both $\mathcal{H}^S$ and $\mathcal{H}^T$ models with the extra combining connections.

Validity Assessment

The cut nets in a K-way partition $\Pi(\mathcal{V})$ of $\mathcal{H}^{ST}$ can be used to model the number of cache misses that occur during the irregular access of x-vector due to loss of spatial and temporal localities, so minimizing the cutnets according to Eq. 2.3 corresponds to minimizing the number of cache misses.

A proof can be given as follows: Under the same assumptions used in the proof of the $\mathcal{G}^{ST}$ model in the previous section, a data cut net n_j^D means that x-vector entry x_j will not be re-used by, at most, $|con(n_j^D)| - 1$ tasks because the rows corresponding to the tasks are not ordered near each other. A data uncut net n_j^D means the opposite because the tasks are reordered consecutively. Similarly, a task cut net n_i^T means that the corresponding task t_i will not reuse the lines that contain x-vector entries that correspond to data vertices in $Pins(n_i)$. A task uncut net n_i^T means the opposite since the columns corresponding to data vertices are reordered consecutively.

5.2.3 Spatially-biased Spatiotemporal Hypergraph (S^{2}TH)

For a sample matrix A, the mutual requirement between the x-vector entries (columns) and the inner product tasks (rows) can be modeled as a hypergraph model $\mathcal{H}^{S^2T}(A) = (\mathcal{V} = \mathcal{V}^D \cup \mathcal{V}^T, \mathcal{N} = \mathcal{N}^T)$.

The vertex set of $\mathcal{H}^{S^2T}$ consists of data vertices and task vertices as follows: each row r_i of the sample matrix necessitates a task vertex $v_i \in \mathcal{V}^T$ and each column c_j necessitates a data vertex $v_j \in \mathcal{V}^D$. The net set of $\mathcal{H}^{ST}$ consists of task nets where each row r_i of the sample matrix necessitates a task net $n_i \in \mathcal{N}^T$.

$\mathcal{H}^{S^2T}(A)$ models the dependencies between task and data vertices as follows: a task net n_i^T connects data vertices corresponding to columns that have nonzeros in row r_i. It also has one more connection to vertex v_i which corresponds to row

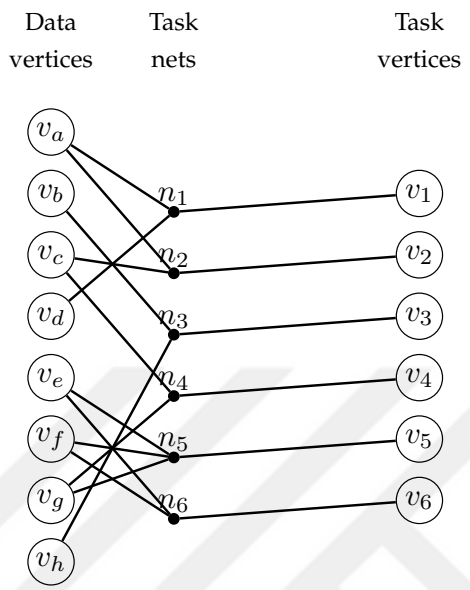


Figure 5.3: Hypergraph model $\mathcal{H}^{S^2T}(A)$ for exploiting spatial and temporal localities simultaneously.

r_i. That is,

$$Pins(n_i^T) \;=\; \{v_j^D : a_{i,j} \neq 0\} \cup \{v_i^T\}. \tag{5.2}$$

We associate unit weights with all nets and all vertices.

Figure 5.3 shows the $\mathcal{H}^{S^2T}$ model of the sample matrix given in Figure 4.1. In the figure, the 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.

This models can be seen as a variation of the $\mathcal{H}^{ST}$ model where the set of vertices to be partitioned is preserved while the number of nets is decreased by C, which is number of columns of matrix A, and the total number of *pins* is decreased by $nnz + R$, which is the number of nonzeros and the rows of the matrix A, respectively.

A task net n_i connecting vertices in the vertex set $Pins(n_i)$ encodes the requirement of x-vector entries in $\{x_k : v_k \in Pins(n_i) \setminus \{v_i\}\}$ by the inner product task t_i corresponding to row r_i. Hence, clustering vertices in $Pins(n_i) \setminus \{v_i\}$ into

common parts increases the possibility of exploiting spatial locality in accessing x-vector entries during the inner-product task t_i Since task vertex v_i is connected with task net n_i, minimizing the cut nets during the k-way partition of $\mathcal{H}^{S^2T}$ motivates v_i to be clustered in the part that its corresponding task net n_i connects most. This relation does not improve temporal locality with this direct definition, however there is an indirect relation between task vertices which states the following: if both nets $n_i, n_j \in \mathcal{N}^T$ have maximum number of *pins* in part $\mathcal{P}_k$, then clustering task vertices v_i and v_j to $\mathcal{P}_k$ will imporve the temporal locality of v_i and v_j in accessing x-vector entries corresponding to data vertices in part $\mathcal{P}_k$.

$\mathcal{H}^{S^2T}$ model encodes the exploitation of spatial locality as a primary partitioning goal, while it exploits temporal locality as a secondary goal. It is very similar to the heuristic-based Temporally-improved spatial hypergraph described in Section 5.1.1.

5.2.4 Temporally-biased Spatiotemporal Hypergraph (ST^{2}H)

For a matrix A, the mutual requirement between the x-vector entries (columns) and the inner product tasks (rows) can be modeled as a hypergraph model $\mathcal{H}^{ST^2}(A) = (\mathcal{V} = \mathcal{V}^D \cup \mathcal{V}^T, \mathcal{N} = \mathcal{N}^D)$.

The vertex set of $\mathcal{H}^{ST^2}$ consists of data vertices and task vertices as follows: each row r_i of the sample matrix necessitates a task vertex $v_i \in \mathcal{V}^T$ and each column c_j necessitates a data vertex $v_j \in \mathcal{V}^D$. The net set of $\mathcal{H}^{ST^2}$ consists of data nets where each column c_j of the sample matrix necessitates a data net $n_j \in \mathcal{N}^D$.

$\mathcal{H}^{ST^2}(A)$ models the dependencies between task and data vertices as follows: a data net n_j^D connects task vertices corresponding to rows that have nonzeros in column c_j. It also connects vertex v_j which corresponds to column c_j. That is,

$$Pins(n_j^D) = \{v_i^T : a_{i,j} \neq 0\} \cup \{v_j^D\}. \tag{5.3}$$

We associate unit weights with all nets and all vertices. Figure 5.4 shows $\mathcal{H}^{ST^2}$

model of the sample matrix given in Figure 4.1.

This model can be seen as a variation of the $\mathcal{H}^{ST}$ model where the set of vertices to be partitioned is preserved while the number of nets is decreased by R, which is number of columns of matrix A, and the total number of *pins* is decreased by $nnz + C$, which are the number of nonzeros and the columns of the matrix A, respectively.

A data net n_j connecting vertices in the vertex set $Pins(n_j)$ encodes the access of the inner-products corresponding of rows in $\{r_k : v_k \in Pins(n_j) \setminus \{v_j\}\}$ to x-vector entry x_j. Hence, clustering vertices in $Pins(n_j) \setminus \{v_j\}$ into common parts increases the possibility of exploiting temporal locality in accessing x_j. Since data vertex v_j is connected with data net n_j, minimizing the cut nets during the k-way partition of $\mathcal{H}^{ST^2}$ motivates v_j to be clustered in the part that its corresponding data net n_j connects most. This relation does not improve spatial locality with this direct definition, however there is an indirect relation between data vertices which states the following: if both nets $n_i, n_j \in \mathcal{N}^D$ have maximum number of *pins* in part $\mathcal{P}_k$, then clustering data vertices v_i and v_j to $\mathcal{P}_k$ will improve the spatial locality of v_i and v_j since task vertices in part $\mathcal{P}_k$ will access them, probably, together.

$\mathcal{H}^{ST^2}$ model encodes the exploitation of temporal locality as a primary partitioning goal, while it exploits spatial locality as a secondary goal. It is very similar to the heuristic-based Spatially-improved temporal hypergraph described in Section 5.1.2.

5.3 Two-Phase Methods

The graph and hypergraph models proposed previously are based on the assumption that each cache miss brings one data word to the cache. Given the fact that each cache miss brings a cache line, consists of W words, we can observe that this fact is useful mostly in spatial locality and it can be utilized for further

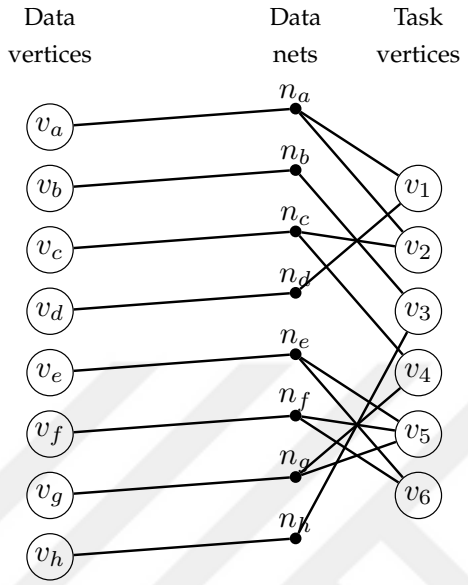
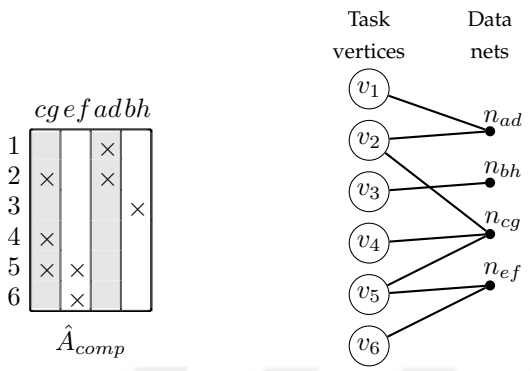


Figure 5.4: Hypergraph model $\mathcal{H}^{ST^2}(A)$ for exploiting spatial and temporal localities simultaneously.

improvement in building the models. In this section we propose two-phase methods that uses two 1D methods from Chapter 4 to exploit spatial and temporal localities. The two-phase methods has to exploit spatial locality in the first phase then temporal locality in the second. In the first phase, a spatial graph or hypergraph method is applied to the sparse matrix in order to reorder its columns and exploit spatial locality. This reordering, which corresponds to reordering x-vector entries, is used to determine the allocation of x-vector entries to blocks of size W as follows: the first W x-vector entries go to the first block, the second W entries go to the second block ... etc. In the second phase, we utilize a temporal method with column stripes, which correspond to x-vector blocks, treated as data vertices/ nets instead of a single column for each vertex/ net.



(a) Compressed matrix $\hat{A}_{comp}$.

(b) $\mathcal{H}^T(\hat{A}_{comp})model$

Figure 5.5: Compressed matrix $\hat{A}_{comp}$ after the first phase of the Two-phase method and the $\mathcal{H}^T$ model representation of $\hat{A}_{comp}$

5.3.1 Two-Phases Hypergraph method (SH→TH$_{line}$)

First Phase

For a matrix A, consider the 1D hypergraph model $\mathcal{H}^S(A) = (\mathcal{V}^D, \mathcal{N}^T)$ and its K-way partition $\Pi(\mathcal{V}^D)$. Using the K-way partitioning, we can reorder the columns of the matrix as describes in Section 4.1.2. The reordered matrix, i.e., output of the SH method, is referred to as $\hat{A}$. The same reordering vector used to reorder the columns of matrix A is used to reorder the entries of the x-vector. Following the reordering, we perform a compression on the columns of the reordered matrix $\hat{A}$ of size $m \times n$ as follows: take the first W columns and pack them in one column where in each row of this column, there exists a nonzero element if, for row r_i and column stripe α, $\{\bigcup_{j=\alpha.W+1}^{\alpha.W+W} a_{i,j}\} \neq \emptyset$. Then take the second, third ... etc. where the last set of columns might have a length less than W , that is, $(W - \delta) : 0 \leq \delta \leq W$. The resulting compressed matrix is called $\hat{A}_{comp}$ which has a size of $m \times \hat{n}$, where $\hat{n} = \lceil \frac{n}{W} \rceil$ is the actual number of cache lines will be brought to the cache during the inner-product task r_i.

Second Phase

For the compressed matrix $\hat{A}_{comp}$, we can use the temporal hypregraph model described in Section 4.2.2 to model the requirement of the inner-product tasks to the x-vector blocks. Figure 5.5 shows the compressed matrix $\hat{A}_{comp}$ and the temporal hypergraph model representation of $\hat{A}_{comp}$.

5.3.2 Two-Phases Graph method (SG→TG$_{line}$)

This method is similar to the previous method but it uses graph-based models instead of hypergraph-based models.

In the first phase, we use the spatial graph model $\mathcal{G}^S$ described in Section 2.1.1 to obtain a K-way partition $\Pi(\mathcal{V}^D)$ to reorder the columns of the sparse matrix A. Using the partition and following the same steps in the previous section, we obtain the compressed matrix $\hat{A}_{comp}$. In the second phase, we use the temporal graph model $\mathcal{G}^T$ based on the compressed matrix $\hat{A}_{comp}$ to model the requirement of the inner-product tasks to the x-vector blocks.

Chapter 6

Experiments

6.1 Data Set

We test the validity of the proposed methods on a set of sparse matrices with irregular sparsity patterns from different applications. All the matrices are obtained from University of Florida Sparse Matrix Collection [22]. Table 6.1 shows the properties of the test matrices, including "avg" and "max" which respectively means the average and maximum number of nonzeros per row/column. The "cov" means the coefficient of variation of the number of nonzeros per row/column. The matrices in the table are grouped into three categories: Symmetric, Non-symmetric Square and Rectangular.

6.2 Experimental Framework

The performances of the proposed methods are evaluated on a single 60-core Xeon Phi 5110 co-processor which runs 4 hardware threads per core and has 512KB L2 cache per core. We use the Sparse Library (SL) [23, 19, 24], which is highly optimized for SpMV operations on Xeon Phi co-processors, to perform our

tests. The SL library has many methods to perform SpMV, we use the method which reported to perform best [23] on Xeon Phi which is Row-Distributed Hilbert Scheme (RDHilbert). This method uses vectorized Bidirectional Incremental CRS (vecBICRS) as a data structure to store the sparse matrix and algorithm to perform the SpMV tasks. We use 240 threads, which is number of all threads in all cores, as suggested in [23, 24]. Yzelman has introduced the explicitly vectorized SpMV in [23] which shows a considerable improvement on the running time with Xeon Phi co-processors. The vectorization options, which yzelman also call blocking options, can be any combination of two numbers if multiplied their product is number of double precision operations that the Vector Processing Unit (VPU) can perform per cycle in the Xeon Phi, i.e., if the vector size is 8, the blocking options can be 1×8, 2×4, 4×2 and 8×1. There is also an additional blocking option 1×1 which means no vectorization. The aforementioned method in SL library has been slightly modified by removing the Space Filling Curve algorithm (Hilbert's Curves) since it might affect the reordering obtained by the proposed methods. We use this modified algorithm to test all methods and as a baseline algorithm, which is referred to throughout this chapter as BL.

To obtain a K-way partitioning on the graph and hypergraph models we use MeTiS [5] and PaToH [7, 25] respectively.

MeTiS is used as a multilevel K-way partitioner with the partitioning objective of minimizing the cut edge cost as in Eq. 2.2. PaToH is also used as a K-way partitioner with the partitioning objective of minimizing the "Connectivity-1" cost according to Eq. 2.3. MeTiS and PaToH are used with default parameters with the exception of the allowed imbalance ratio, which is set to 30% because we focus more on reordering than load balancing. Since these tools contain randomized algorithms, we repeat each partitioning instance three times and report the average results. While partitioning graph and hypergraph models, K is selected such that each part is sufficiently small to obtain cache-oblivious reorderings. That is, each matrix's storage size (in terms of KB), which is reported by the SL library, is divided by 32KB to obtain K.

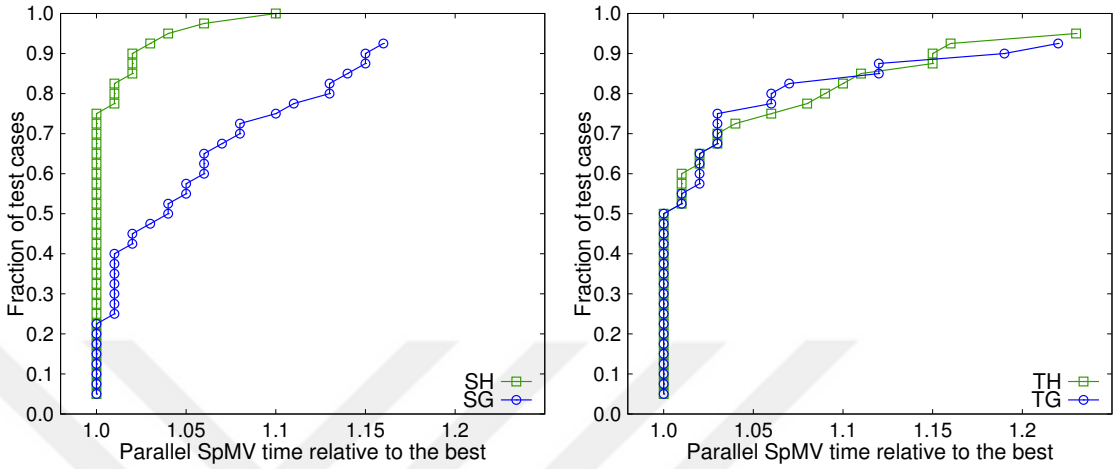
6.3 Performance Evaluation

In this section, we evaluate the performances of the proposed methods according to their performance speed in Gflop/s. The evaluation process consists of three main parts as follows: first we compare Graph vs Hypergraph partitioning methods, then we compare separate methods with simultaneous methods and afterwards we show the importance of exploiting locality and how it boosts up the effectiveness of vectorization. We use performance profiles to compare different methods. The performance profiles compare different methods by showing the fraction of test cases for how much each method is close to the best. In other words, if a method reaches 100% of test cases with full closeness to the best, i.e. 1, that means this method performs best in all test cases. Tables 6.3 is used to show the geometric means of the proposed methods for all possible vectorization options. In the table, "Best" means the geometric mean of all performances per matrix that performed best among all blocking options. Also, vec.Avg. means the geometric mean of all average performances among all blocking options, except for 1×1, per matrix.

6.3.1 Graph vs Hypergraph Partitioning Methods

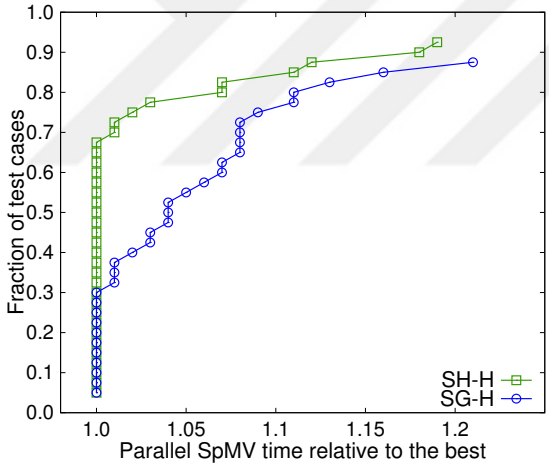
Here we compare the performances of graph-based and hypergraph-based methods.

Figure 6.1 shows the performance profiles of the SpMV times for all 1D and 2D methods, except for the hypergraph-based methods in Sections 5.2.3 and 5.2.4 because they do not have equivalent graph methods. As seen in the figure, the HP-based methods perform significantly better than the GP-based methods. However, in terms of temporal methods, the difference become smaller since temporal locality is less sensitive to the deficiencies of the graph models described in [25].

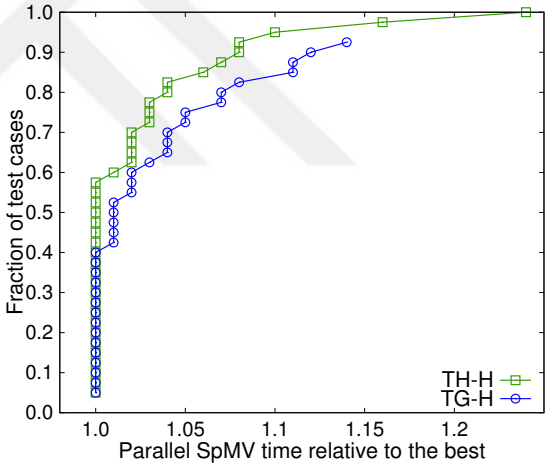


(a) Spatial Methods

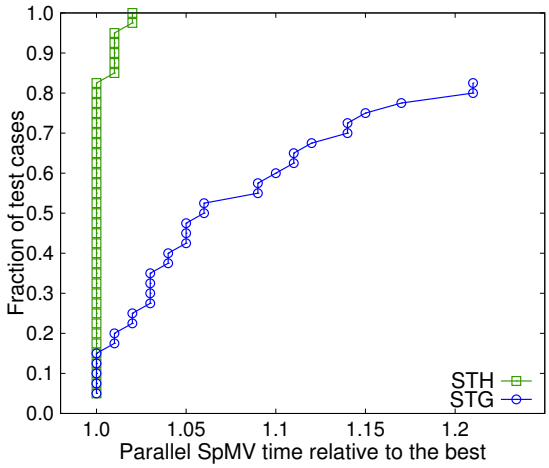
(b) Temporal Methods



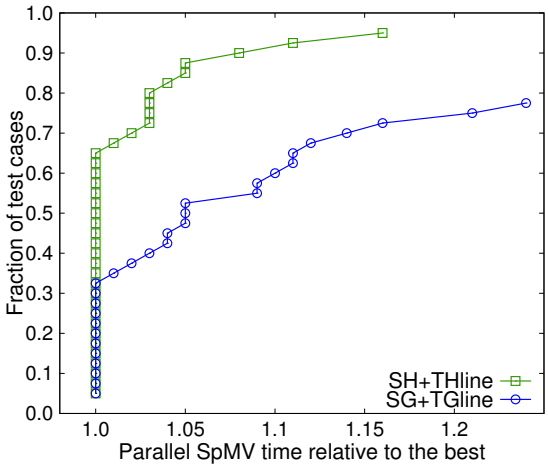
(c) Huristic-based Spatial Methods



(d) Huristic-based Temporal Methods



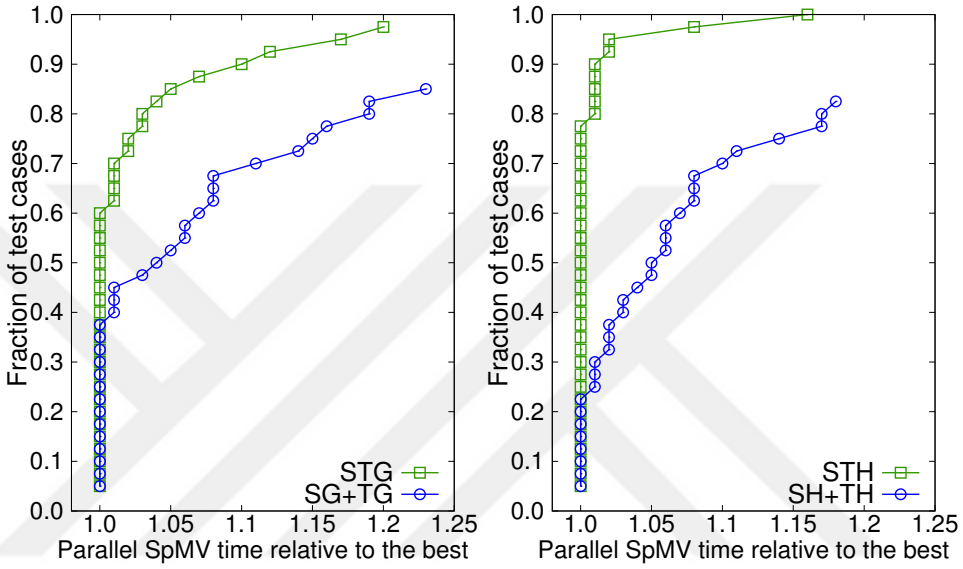
(e) Spatiotemporal Methods



(f) TwoPhase Methods

Figure 6.1: Performance profiles for HP-based methods vs GP-based methods.

6.3.2 Reordering with two 1D methods vs simultaneous methods



(a) GP Methods (b) HP Methods

Figure 6.2: Performance profiles for separate vs simultaneous graph and hypergraph partitioning methods.

A possible straight forward method to utilize 1D models in 2D reordering is to apply spatial method to reorder the columns of the sparse matrix and a temporal method to reorder the rows of the sparse matrix. Though we have four possible combinations by mixing graph and hypergraph models, we consider 1D graph models together and hypergraph models together. These methods are called SG+TG and SH+TH, respectively. Note that in SG+TG and SH+TH we apply each method separately and independently. In this section, we compare the above mentioned methods with 2D methods that target the exploitation of spatial and temporal localities simultaneously.

Figure 6.2 shows the performance profiles that compare the SpMV times of separate 2D reordering vs simultaneous 2D reordering. The figure shows how

simultaneous methods perform significantly better than separate methods. However, SG+TG and SH+TH can be useful if used to reorder symmetric matrices. Table 6.4 shows the geometric means of all matrices per category. In symmetric category, its obvious that SG+TG and SH+TH perform close to the simultaneous graph and hypergraph methods, respectively, while in other categories simultanious methods perform considerably better. The reason behind this is that in symmetric matrices both spatial and temporal models are the same so utilizing them to exploit both localities is nearly equal to using simultaneous methods.

6.3.3 Comparison of 2D Methods

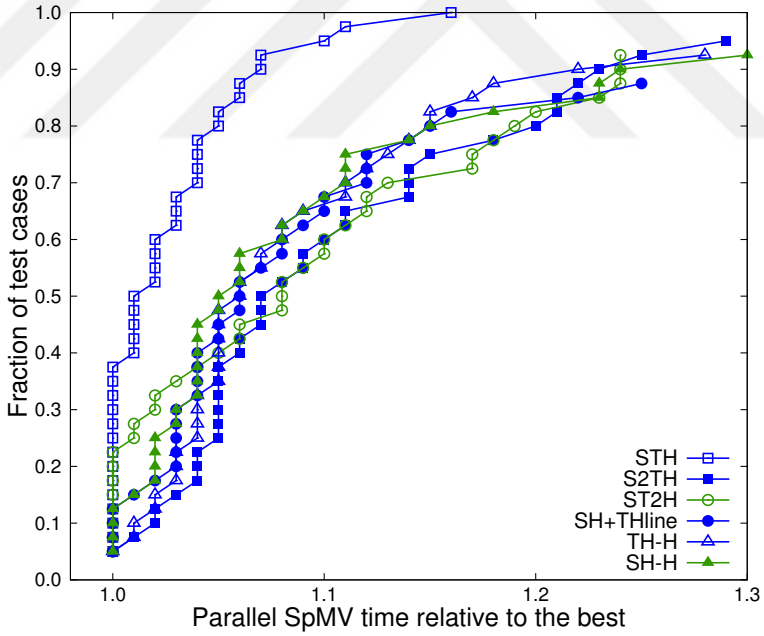


Figure 6.3: Performance profiles for comparing the performances of hypergraph-based 2D methods.

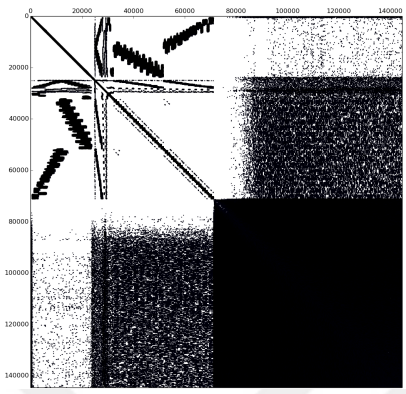
In this section, we compare the 2D methods and study the benefits of using each one of them. We exclude the graph-based methods from this section because we have already shown that hypergraph-based methods perform much better than the graph-based methods. Figure 6.3 shows the performance profiles that compare

the SpMV times of 2D hypergraph based methods. As seen in the figure, as well as in Table 6.3, STH outperform all other methods and it is the best fit for all vectorization options.

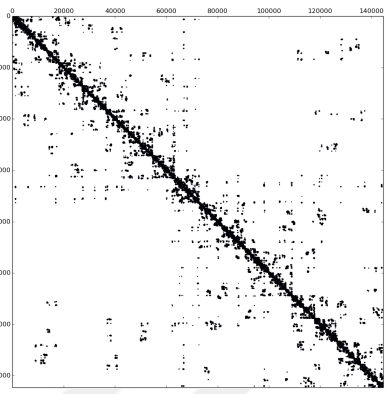
Figure 6.3 also shows that the other 2D hypergraph-based methods have a comparable performance. Table 6.3 shows that the geomeans of SH-H, TH-H, S^2TH, ST^2H and SH→TH_{line}methods are close to each other. We can distinguish between the 2D methods in Table 6.6 by their number of best instances. ST^2H method is the second best, after STH, with 13 instances out of 53. The methods ST^2H and TH-H both follow the same logic, in TH-H the temporal locality is improved first by reordering rows using TH method, then according to the reordered matrix we reorder the columns in a way that best fits the new temporally-improved matrix as discussed in Section 5.1.2. ST^2H does something similar during the partitioning of $\mathcal{H}^{ST^2}$ model, it encourages a data vertex to be clustered to the part that connects its corresponding data net most, in order to minimize the cut nets. The main difference between ST^2H and TH-H is the load balancing on the secondary exploiting objective. In ST^2H, clustering data vertices to parts is done during the partitioning of the model so the load balancing criteria holds. However in TH-H, the clustering of data vertices is done separately and it does not consider any load balancing criteria. Note that the same comparison terms are applicable on the comparison between SH-H and S^2TH. Table 6.5 shows the number of vertices, nets and pins in every hypergraph partitioning model. Number of pins in this context means the total number of connections between nets and vertices. Note that m is number of rows in the matrix and n is the number of columns.

Table 6.5: Number of vertices, nets and pins for the 2D models

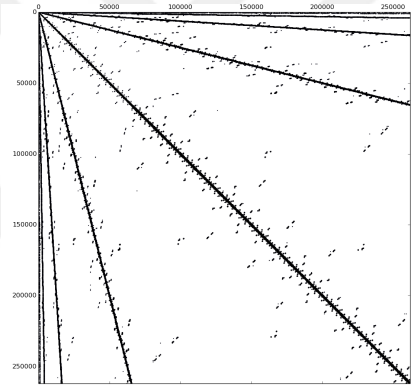
Method	# vertices	# nets	# pins
SH-H	n	m	nnz
TH-H	m	n	nnz
S^2TH	m+n	n	n + nnz
ST^2H	m+n	m	m + nnz
STH	m+n	m+n	m+n+2nnz
TH_{line} ϕ 1	n	m	nnz
TH_{line} ϕ 2	m	n	nnz



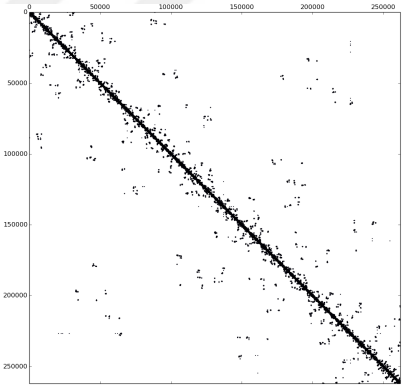
(a) 144



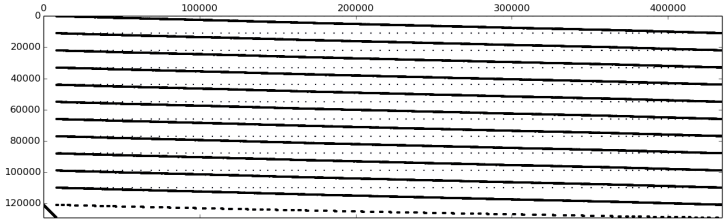
(b) 144 after STH



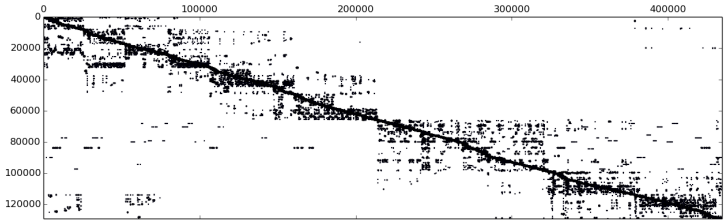
(c) delaunay_n18



(d) delaunay_n18 after STH



(e) pds-80



(f) pds-80 after STH

Figure 6.4: Plots of sample matrices before and after applying STH reordering method.

6.3.4 The 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 were selected from the data set, where the improvement in performance after reordering w.r.t the baseline is very high. Figure 6.4a shows the plot of the sparse matrix "144" which has a very irregular sparsity pattern. In Table 6.2, the matrix improved from 3.09 Gflop/s to 5.09 Gflop/s when exploiting spatial locality by using only SH method, to 7.58 Gflop/s when exploiting temporal locality by using only TH method, and to 16.78 Gflop/s when exploiting spatial and temporal localities with STH method. Figure 6.4b shows the reordered matrix after applying STH method.

The second case is the "delaunay_n18" matrix in Fig. 6.4c which has a good spatial locality properties, but a very bad sparsity pattern for temporal locality. As seen in Table 6.2, trying to improve spatial locality alone will not improve the performance. However, exploiting temporal locality alone has a high effectiveness on the performance, it is increased from 3.48 Gflop/s to 12.54 Gflop/s after reordering the rows with TH method. When targeting both localities, the performance increases to 16.34 when using STH method. Figure 6.4d shows the reordered matrix after applying STH method.

The last selected matrix is "pds-80" in Fig. 6.4e which suffers from poor spatial locality. Form Table 6.2, reordering using a temporal method will not improve the performance. However, reordering the columns to exploit spatial locality with 1D spatial method improves the performance significantly from 4.30 Gflop/s to 10.04 Gflop/s. Targeting both localities however improves the performance further to 12.35 Gflop/s when using TH_{line} method.

6.3.5 The impact of exploiting locality on vectorization

Vectorization might be very beneficial in improving the performance of SpMV. However, it is highly dependent on the sparsity pattern of the input matrix meaning that if the sparsity pattern does not fit the vectorization options or the matrix has a very irregular sparsity pattern, the vectorization might not have its suspected impact on performance. Our findings show that the reordering methods have an important role in utilizing the vectorized SpMV algorithm. Our data set contains matrices with highly irregular sparsity patterns. Table 6.3 shows the importance of exploiting data locality for utilizing vectorization. In the table, the baseline method preform worse in case of 1×8 compared to no-vectorization (1×1). If we look at the geomeans of the 1D methods, we can see that some blocking options might perform worse than no blocking 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.

In 2D methods, Table 6.3 shows that using any blocking option always perform better than 1×1 option. It also shows that the 2D methods that exploit spatial and temporal localities without giving a preference to one locality more than the other, e.g., STH and TH_{line}, performs good in terms of the average of all blocking options.

Table 6.1: Properties of test matrices

	Number of			Nnz's in a row			Nnz's in a column		
Matrix	rows	cols	nonezeros	avg	max	cov	avg	max	cov
Symmetric matrices									
144	144,649	144,649	2,148,786	14.86	26	0.18	14.86	26	0.18
bmw7st_1	141,347	141,347	7,339,667	51.93	435	0.25	51.93	435	0.25
ca2010	710,145	710,145	3,489,366	4.91	141	0.59	4.91	141	0.59
citationCiteseer	268,495	268,495	2,313,294	8.62	1,318	1.89	8.62	1,318	1.89
coAuthorsCiteseer	227,320	227,320	1,628,268	7.16	1,372	1.48	7.16	1,372	1.48
darcy003	389,874	389,874	2,101,242	5.39	7	0.36	5.39	7	0.36
delaunay_n18	262,144	262,144	1,572,792	6.00	21	0.22	6.00	21	0.22
delaunay_n19	524,288	524,288	3,145,646	6.00	21	0.22	6.00	21	0.22
F2	71,505	71,505	5,294,285	74.04	345	0.51	74.04	345	0.51
G3_circuit	1,585,478	1,585,478	7,660,826	4.83	6	0.13	4.83	6	0.13
gupta2	62,064	62,064	4,248,286	68.45	8,413	5.20	68.45	8,413	5.20
m14b	214,765	214,765	3,358,036	15.64	40	0.20	15.64	40	0.20
NACA0015	1,039,183	1,039,183	6,229,636	5.99	10	0.14	5.99	10	0.14
ny2010	350,169	350,169	1,709,544	4.88	61	0.52	4.88	61	0.52
pattern1	19,242	19,242	9,323,432	484.54	6,028	0.78	484.54	6,028	0.78
pkustk12	94,653	94,653	7,512,317	79.37	4,146	1.87	79.37	4,146	1.87
vsp_bcsstk30_500sep_10in_1Kout	58,348	58,348	4,033,156	69.12	219	0.47	69.12	219	0.47
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									
amazon0312	400,727	400,727	3,200,440	7.99	10	0.38	7.99	2,747	1.89
amazon0505	410,236	410,236	3,356,824	8.18	10	0.38	8.18	2,760	1.87
amazon0601	403,394	403,394	3,387,388	8.40	10	0.33	8.40	2,751	1.82
av41092	41,092	41,092	1,683,902	40.98	2,135	4.08	40.98	664	2.37
flickr	820,878	820,878	9,837,214	11.98	10,272	7.32	11.98	8,549	5.97
heart1	3,557	3,557	1,387,773	390.15	1,120	0.32	390.15	1,120	0.32
laminar_duct3D	67,173	67,173	3,833,077	57.06	89	0.66	57.06	89	0.52
soc-Slashdot0811	77,360	77,360	905,468	11.70	2,508	3.15	11.70	2,540	3.18
soc-Slashdot0902	82,168	82,168	948,464	11.54	2,511	3.20	11.54	2,553	3.25
TSOPF_RS_b300_c1	14,538	14,538	1,474,325	101.41	209	1.01	101.41	6,902	4.68
TSOPF_RS_b300_c3	42,138	42,138	4,413,449	104.74	209	0.98	104.74	20,702	7.99
twotone	120,750	120,750	1,224,224	10.14	185	1.48	10.14	188	1.86
web-NotreDame	325,729	325,729	1,497,134	4.60	3,445	4.67	4.60	10,721	8.50
web-Stanford	281,903	281,903	2,312,497	8.20	255	1.38	8.20	38,606	20.28
webbase-1M	1,000,005	1,000,005	3,105,536	3.11	4,700	8.16	3.11	28,685	11.88
wiki-Talk	2,394,385	2,394,385	5,021,410	2.10	100,022	47.64	2.10	3,311	5.82
Rectangular matrices									
cont1_l	1,918,399	1,921,596	7,031,999	3.67	5	0.26	3.66	1,279,998	252.33
cont11_l	1,468,599	1,961,394	5,382,999	3.67	5	0.26	2.74	7	0.90
dbir2	18,906	45,877	1,158,159	61.26	4,950	3.86	25.24	233	1.49
GL7d14	171,375	47,271	1,831,183	10.69	24	0.24	38.74	160	0.44
GL7d15	460,261	171,375	6,080,381	13.21	38	0.18	35.48	137	0.40
GL7d24	21,074	105,054	593,892	28.18	751	0.72	5.65	13	0.22
IMDB	428,440	896,308	3,782,463	8.83	1,334	1.73	4.22	1,590	3.09
mesh_deform	234,023	9,393	853,829	3.65	4	0.18	90.90	166	0.20
neos	479,119	515,905	1,526,794	3.19	29	0.16	2.96	16,220	15.57
NotreDame_actors	392,400	127,823	1,470,404	3.75	646	2.75	11.50	294	1.02
nsct	23,003	37,563	697,738	30.33	848	2.54	18.58	628	3.32
pds-100	156,243	514,577	1,096,002	7.01	101	1.03	2.13	3	0.18
pds-80	129,181	434,580	927,826	7.18	96	0.99	2.13	3	0.18
pds-90	142,823	475,448	1,014,136	7.10	96	1.01	2.13	3	0.18
rel8	345,688	12,347	821,839	2.38	4	0.82	66.56	121	0.26
route	20,894	43,019	206,782	9.90	2,781	7.06	4.81	44	1.01
sgpf5y6	246,077	312,540	831,976	3.38	61	2.21	2.66	12	0.74
Trec14	3,159	15,905	2,872,265	909.23	1,837	0.39	180.59	2,500	1.71
watson_1	201,155	386,992	1,055,093	5.25	93	2.45	2.73	9	0.47

Table 6.2: Performance results (in Gflop/s) for all test matrices

Matrix	BL	SG	TG	STG	SG+TG	SH	TH	STH	SH+TH	THline
Symmetric matrices										
144	3.09	5.18	7.96	16.15	16.88	5.09	7.58	16.78	**16.93**	16.09
bmw7st_1	21.17	23.13	23.39	24.46	25.11	22.66	23.37	**25.24**	24.83	25.21
ca2010	3.33	4.50	5.85	10.50	10.26	4.53	5.98	10.44	**10.52**	10.47
citationCiteseer	2.27	3.59	3.20	5.66	5.34	3.56	3.27	**5.73**	5.49	5.08
coAuthorsCiteseer	3.10	3.03	5.47	7.98	7.78	3.06	5.15	**8.38**	7.38	6.71
darcy003	2.82	2.81	8.40	**18.46**	18.45	2.76	8.44	18.10	18.05	17.40
delaunay_n18	3.48	3.43	12.73	16.17	16.22	3.47	12.54	**16.34**	16.08	16.27
delaunay_n19	2.56	2.82	11.01	13.49	13.49	2.78	10.98	13.47	**13.55**	13.45
F2	14.70	19.21	20.56	24.10	24.75	19.23	20.83	**24.82**	24.60	24.71
G3_circuit	5.54	5.39	10.35	12.93	13.04	5.44	10.06	13.04	12.77	**13.18**
gupta2	15.37	11.53	12.28	10.27	14.48	13.24	12.53	**15.98**	15.98	15.28
m14b	2.98	4.92	7.61	15.23	15.35	4.94	7.55	**15.67**	15.67	15.33
NACA0015	1.97	2.75	4.99	12.86	13.00	2.74	5.04	13.13	13.12	**13.26**
ny2010	5.29	6.58	7.81	11.91	11.89	6.36	8.21	11.87	**12.03**	11.68
pattern1	12.69	14.64	15.61	19.44	17.83	14.68	16.59	**20.29**	18.64	15.09
pkustk12	19.35	20.26	21.42	21.30	22.62	21.43	20.94	**23.63**	22.40	22.93
vsp_bcsstk30_500sep_10in_1Kout	6.70	15.27	11.76	23.64	22.64	15.37	10.78	24.61	23.52	**24.73**
vsp_msc10848_300sep_100in_1Kout	10.99	21.63	15.72	28.88	28.95	22.40	16.53	**30.96**	30.34	29.83
Square nonsymmetric matrices										
amazon0312	2.70	3.72	4.18	7.17	6.31	3.61	4.17	**7.31**	6.30	5.55
amazon0505	2.84	3.76	4.45	7.01	6.59	3.56	4.54	**7.27**	7.05	5.52
amazon0601	2.81	3.84	4.29	6.30	6.37	3.83	4.40	**7.13**	6.71	5.36
av41092	15.59	20.14	15.60	15.66	17.46	**21.72**	17.35	21.23	20.01	20.99
flickr	1.82	1.98	2.70	3.34	2.72	2.01	2.27	4.01	2.80	**4.32**
heart1	36.17	34.31	35.74	32.07	33.50	36.64	36.38	36.72	36.37	**37.09**
laminar_duct3D	21.58	14.05	22.84	23.69	23.65	14.73	23.16	24.86	**25.12**	24.62
soc-Slashdot0811	4.42	4.33	4.94	5.39	4.47	4.53	4.51	**6.14**	5.22	5.80
soc-Slashdot0902	4.82	4.05	4.86	5.19	4.88	4.45	4.25	**6.58**	4.80	5.72
TSOPF_RS_b300_c1	27.04	21.12	31.09	26.71	27.53	24.03	33.67	**34.44**	32.74	30.06
TSOPF_RS_b300_c3	19.98	10.60	26.22	24.61	22.27	16.73	26.69	26.71	**27.04**	26.16
twotone	15.16	14.51	13.30	15.25	12.88	15.34	16.15	**16.7**	15.55	16.32
web-NotreDame	6.81	5.85	6.48	**9.27**	5.21	6.85	5.31	9.26	6.20	8.36
web-Stanford	2.90	4.92	3.63	10.55	9.15	4.93	3.38	**10.65**	9.87	10.57
webbase-1M	4.56	4.45	2.76	6.44	2.69	4.49	4.89	**6.96**	5.22	6.62
wiki-Talk	1.17	0.85	1.15	1.25	0.79	1.54	1.04	**1.97**	1.50	1.62
Rectangular matrices										
cont1_l	9.47	6.03	8.58	10.03	9.51	6.81	8.42	**11.71**	10.73	10.84
cont11_l	9.11	5.98	7.51	12.48	10.94	6.16	7.29	**12.79**	10.94	11.51
dbir2	8.35	7.09	5.11	9.14	8.81	7.84	6.47	**14.14**	9.75	10.47
GL7d14	4.79	4.96	5.33	5.93	5.54	5.29	5.33	6.25	**6.26**	5.81
GL7d15	2.46	2.77	2.92	3.41	3.94	2.82	3.26	3.76	**4.36**	3.51
GL7d24	3.14	4.04	4.16	5.12	5.24	3.68	3.84	**5.65**	5.10	3.67
IMDB	2.03	2.41	1.72	3.25	2.35	2.71	1.14	**3.91**	2.63	3.50
mesh_deform	14.95	15.08	23.25	23.74	**23.84**	14.92	22.58	23.56	23.61	23.41
neos	12.54	7.05	3.47	12.50	3.20	8.01	7.97	**14.56**	9.98	12.56
NotreDame_actors	3.58	3.83	3.89	4.97	4.31	4.05	3.92	5.10	4.74	**5.31**
nsct	11.44	10.47	8.04	14.33	9.08	13.12	8.21	**17.83**	12.26	17.27
pds-100	3.56	8.88	3.84	10.68	9.96	9.82	3.33	11.20	10.57	**11.82**
pds-80	4.30	10.04	4.17	11.52	10.76	10.04	3.28	11.33	11.15	**12.35**
pds-90	3.96	9.43	3.87	11.31	10.73	10.17	3.38	11.27	11.29	**12.02**
rel8	15.61	15.94	15.74	15.26	16.66	16.27	15.84	**17.23**	16.91	16.03
route	4.86	9.68	3.54	7.80	9.29	10.12	4.18	9.47	**10.14**	9.87
sgpf5y6	8.90	7.82	6.70	10.10	8.49	9.02	8.60	**11.25**	9.86	11.15
Trec14	19.08	18.96	18.52	15.46	18.00	20.00	18.47	20.34	19.68	**20.36**
watson_1	10.82	9.95	11.05	**14.82**	10.97	11.43	11.04	14.69	13.88	13.92
Number of bests :	0	0	0	3	1	1	0	29	9	10

Table 6.3: Performance geomeans (in Gflop/s) of all test matrices for all methods and all possible blocking sizes

Method	1x1	1x8	2x4	4x2	8x1	Best	vec.Avg.
BL	4.49	4.07	5.02	5.27	4.79	6.18	4.89
	1D Methods						
SG	5.16	5.42	6.06	5.82	5.06	6.90	5.70
TG	5.37	5.20	6.48	6.79	6.20	7.51	6.21
SH	5.33	5.75	6.44	6.18	5.34	7.33	6.04
TH	5.27	5.13	6.41	6.89	6.35	7.62	6.29
	2D Methods						
SG-H	6.63	7.93	9.42	9.22	7.98	10.48	8.78
TG-H	6.75	8.30	9.79	9.63	8.18	10.85	9.11
SG+TG	6.42	7.84	9.09	8.80	7.70	9.96	8.47
TG_{line}	6.47	7.98	9.04	8.56	7.29	9.97	8.35
STG	6.80	8.48	9.88	9.48	7.98	10.97	9.11
SH-H	6.86	8.58	10.11	9.91	8.32	11.33	9.40
TH-H	6.83	8.59	10.24	10.12	8.64	11.29	9.53
SH+TH	6.79	8.66	10.00	9.83	8.43	11.08	9.36
TH_{line}	6.87	8.98	10.41	9.98	8.36	11.33	9.57
S^2TH	6.89	8.94	10.08	9.28	7.57	11.25	9.13
ST^2H	6.96	8.36	10.10	10.04	8.53	11.30	9.42
STH	**7.07**	**9.39**	**11.06**	**10.75**	**8.80**	**12.11**	**10.11**

Table 6.4: Performance geomeans (in Gflop/s) catagorized per matrix type for all methods

Method	Symmetric	Nonsymmetric square	Rectangular
BL	5.57	6.44	6.58
SG	7.01	6.31	7.34
TG	10.04	7.21	5.90
SG-H	15.02	7.72	9.65
TG-H	15.34	8.90	9.23
STG	15.06	9.23	9.41
SG+TG	15.31	7.76	8.18
TG_{line}	15.29	7.65	8.31
SH	7.07	7.01	7.87
TH	10.03	7.33	6.07
SH-H	15.24	9.18	10.22
TH-H	15.19	9.80	9.61
STH	**15.83**	**10.58**	**10.54**
ST^2H	15.49	9.35	9.82
S^2TH	14.94	9.73	9.71
SH+TH	15.48	9.18	9.47
TH_{line}	15.10	9.63	9.89

Table 6.6: Performance results (in Gflop/s) for 2D HP methods

Matrix	SH-H	TH-H	STH	ST^{2}H	S^{2}TH	TH$_{line}$
Symmetric matrices						
144	16.41	**16.96**	16.78	15.43	15.39	16.09
bmw7st_1	24.98	24.69	25.24	**25.4**	24.62	25.21
ca2010	10.45	10.45	10.44	**10.88**	10.25	10.47
citationCiteseer	5.44	5.36	**5.73**	5.46	4.77	5.08
coAuthorsCiteseer	7.36	7.60	**8.38**	7.54	6.95	6.71
darcy003	18.02	18.35	18.10	**18.43**	17.71	17.40
delaunay_n18	16.39	16.15	16.34	**17.33**	16.21	16.27
delaunay_n19	13.49	13.54	13.47	**13.95**	13.33	13.45
F2	24.53	24.52	**24.82**	23.91	23.90	24.71
G3_circuit	12.78	12.84	13.04	**13.48**	12.67	13.18
gupta2	13.55	12.12	**15.98**	12.99	13.03	15.28
m14b	15.36	15.57	**15.67**	15.54	15.11	15.33
NACA0015	13.11	13.11	13.13	**13.54**	12.78	13.26
ny2010	12.03	12.06	11.87	**12.55**	12.03	11.68
pattern1	17.61	17.97	20.29	**21.83**	21.70	15.09
pkustk12	22.92	22.61	**23.63**	21.14	20.73	22.93
vsp_bcsstk30_500sep_10in_1Kout	22.46	22.73	24.61	23.07	23.53	**24.73**
vsp_msc10848_300sep_100in_1Kout	30.70	29.77	**30.96**	28.55	28.76	29.83
Square nonsymmetric matrices						
amazon0312	5.60	6.39	**7.31**	6.55	6.20	5.55
amazon0505	6.91	7.03	**7.27**	6.20	5.82	5.52
amazon0601	5.81	6.64	**7.13**	5.99	5.63	5.36
av41092	18.86	19.46	21.23	19.18	**21.58**	20.99
flickr	**4.44**	4.03	4.01	3.73	4.31	4.32
heart1	36.39	36.99	36.72	31.36	36.10	**37.09**
laminar_duct3D	**25.0**	24.38	24.86	23.71	23.50	24.62
soc-Slashdot0811	4.73	5.26	**6.14**	4.99	5.87	5.80
soc-Slashdot0902	5.41	5.12	**6.58**	4.89	5.73	5.72
TSOPF_RS_b300_c1	23.83	33.59	**34.44**	29.61	30.44	30.06
TSOPF_RS_b300_c3	15.21	**27.25**	26.71	25.32	23.93	26.16
twotone	16.10	16.31	**16.7**	16.46	15.89	16.32
web-NotreDame	8.78	8.98	9.26	8.03	**9.45**	8.36
web-Stanford	10.20	**11.14**	10.65	10.56	10.97	10.57
webbase-1M	6.91	6.75	6.96	**7.13**	5.53	6.62
wiki-Talk	1.61	1.26	**1.97**	1.45	1.82	1.62
Rectangular matrices						
cont1_l	10.46	11.32	11.71	**11.8**	11.58	10.84
cont11_l	12.63	12.20	**12.79**	12.42	12.65	11.51
dbir2	12.90	13.22	**14.14**	12.95	13.03	10.47
GL7d14	6.23	5.44	**6.25**	5.67	6.05	5.81
GL7d15	**3.85**	3.43	3.76	3.50	3.51	3.51
GL7d24	4.03	4.92	**5.65**	4.61	4.15	3.67
IMDB	3.60	3.06	**3.91**	2.75	3.45	3.50
mesh_deform	23.54	23.81	23.56	**24.44**	23.45	23.41
neos	13.15	12.86	**14.56**	12.53	13.40	12.56
NotreDame_actors	**5.31**	4.53	5.10	4.96	4.84	5.31
nsct	**17.87**	17.64	17.83	17.00	12.76	17.27
pds-100	11.00	11.24	11.20	11.39	9.75	**11.82**
pds-80	11.52	11.79	11.33	12.28	10.37	**12.35**
pds-90	11.50	11.45	11.27	11.80	9.96	**12.02**
rel8	**17.81**	15.66	17.23	16.60	16.11	16.03
route	**10.97**	5.54	9.47	7.48	10.47	9.87
sgpf5y6	11.03	10.64	11.25	11.28	**11.47**	11.15
Trec14	19.99	18.20	20.34	16.51	19.41	**20.36**
watson_1	13.71	14.66	14.69	**15.19**	13.32	13.92
Number of bests :	7	3	21	13	3	6

Chapter 7

Conclusion and Future Work

In this thesis, models and methods for exploiting spatial and temporal localities were proposed. The proposed methods consist of 1D methods that target the reordering of either rows or column of the input matrix using a Graph-based or Hypergraph-based partitioning model, 2D simultaneous methods that target the reordering of both rows and columns using one Graph or Hypergraph based partitioning model that targets both dimensions, and 2D hybrid methods that utilize a GP or HP model to reorder one dimension and a heuristic-based algorithm to reorder the other. The validity of the proposed methods was confirmed by conducting excessive tests on a wide set of irregular sparse matrices revealing a significant improvements in SpMV performance. The results show all methods that are based on hypergraph models perform generally better than their graph-based counterparts in terms of SpMV performance. The results also report that the spatiotemporal hypergraph method (STH) is best performed method among all proposed methods.

For future work, developing a smart, but fast, algorithm to reorder further the vertices inside a part after a K-way partitioning might be beneficial for the performance. Another future research direction is to develop a criteria to select the best fit method to be used to reorder a matrix depending on its sparsity pattern or depending on the application.

Bibliography

[1] T. Bui and C. Jones, *A heuristic for reducing fill-in in sparse matrix factorization*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (United States), Dec 1993.

[2] B. Hendrickson and R. W. Leland, "A multi-level algorithm for partitioning graphs.," *SC*, vol. 95, p. 28, 1995.

[3] 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.

[4] B. Hendrickson and R. Leland, "The chaco users guide: Version 2.0," tech. rep.

[5] 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.

[6] G. Karypis and V. Kumar, "hmetis 1.5: A hypergraph partitioning package," tech. rep., Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL http://www. cs. umn. edu/metis, 1998.

[7] 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.

[8] I. Al-Furaih and S. Ranka, "Memory hierarchy management for iterative graph structures," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pp. 298–302, Mar 1998.

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

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

[11] 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.

[12] "Two-dimensional cache-oblivious sparse matrixvector multiplication," *Parallel Computing*, vol. 37, no. 12, pp. 806 – 819, 2011. 6th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'10).

[13] 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*, PLDI '99, pp. 229–241, 1999.

[14] 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*, ch. A Comparison of Locality Transformations for Irregular Codes, pp. 70–84. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[15] S. Toledo, "Improving the memory-system performance of sparse-matrix vector multiplication," *IBM Journal of Research and Development*, vol. 41, pp. 711–725, Nov 1997.

[16] J. B. White and P. Sadayappan, "On improving the performance of sparse matrix-vector multiplication," in *High-Performance Computing, 1997. Proceedings. Fourth International Conference on*, pp. 66–71, Dec 1997.

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

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

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

[20] E. Saule, K. Kaya, and Ü. V. Çatalyürek, *Parallel Processing and Applied Mathematics: 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I*, ch. Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi, pp. 559–570. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[21] M. O. Karsavuran, K. Akbudak, and C. Aykanat, "Locality-aware parallel sparse matrix-vector and matrix-transpose-vector multiplication on many-core processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2015.

[22] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, pp. 1:1–1:25, Dec. 2011.

[23] "Chapter 27 - sparse matrix-vector multiplication: Parallelization and vectorization," in *High Performance Parallelism Pearls* (J. Reinders and J. Jeffers, eds.), pp. 457 – 476, Boston: Morgan Kaufmann, 2015.

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

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