

PARALLEL SPARSE AND BANDED MATRIX – MULTIPLE VECTORS
MULTIPLICATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEFTUN CİNCİOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2014

Approval of the thesis:

**PARALLEL SPARSE AND BANDED MATRIX – MULTIPLE VECTORS
MULTIPLICATION**

submitted by **MEFTUN CİNCİOĞLU** in partial fulfillment of the requirements for
the degree of **Master of Science in Computer Engineering Department, Middle
East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural And Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Murat Manguoğlu
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Murat Manguoğlu
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Ertan Onur
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

Dr. Murat Cenk
Institute of Applied Mathematics, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Meftun Cincioğlu

Signature :

ABSTRACT

PARALLEL SPARSE AND BANDED MATRIX – MULTIPLE VECTORS MULTIPLICATION

Cincioğlu, Meftun
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Murat Manguoğlu
August 2014, 72 pages

In this thesis, performance of two important primitives, namely sparse and banded matrix – multiple vectors multiplication are studied.

Sparse matrix – multiple vectors multiplication (SpMM) is one of the basic and most time consuming operations in many problems in science and engineering. Hence, any improvement in the performance of SpMM operations has a great impact on the wide spectrum of problems. One of the objectives of this thesis is to improve the performance of parallel SpMM operation by reducing indirect memory access, improving communication pattern, and load balancing. For this purpose, partitioning tools and permutation algorithms are used.

Banded matrix – multiple vectors multiplication is used as a primitive operation in iterative solution of banded linear systems or in other applications. An improved method is presented that has an advantage especially for banded matrices having small bandwidth and multiplied by large number of vectors.

All these numerical experiments are performed in two different computing platforms.

Keywords: Banded matrix, sparse matrix, multiple, vector, multiplication

ÖZ

PARALEL SEYREK VE BANT MATRİS – ÇOKLU VEKTÖR ÇARPIMI

Cinciođlu, Meftun
Yüksek Lisans, Bilgisayar Mühendisliđi Bölümü
Tez Yöneticisi: Doç. Dr. Murat Manguođlu
Ađustos 2014, 72 sayfa

Bu tezde iki önemli işlemin, seyrek ve bant matris – çoklu vektör çarpımının, performansı incelenmiştir.

Seyrek matris – çoklu vektör çarpımı (SpMM), bilimde ve mühendislikteki çođu problem için temel ve çok zaman alan işlemlerden biridir. Dolayısıyla, SpMM işleminin performansını etkileyecek herhangi bir iyileştirme, çok çeşitli alanlardaki problemlerin çözümünde büyük etki yaratmaktadır. Bu tezin amaçlarından biri, dolaylı bellek erişimini azaltarak, iletişim örüntülerini geliştirerek ve yük dengeleyerek paralel SpMM işleminin performansını arttırmaktır. Bu yüzden bölümlendirme araçları ve yer deđiştirme algoritmaları kullanılmıştır.

Bant matris – çoklu vektör çarpımı, bantlı çizgisel sistemlerin dolaylı yöntemler ile çözümünde veya diđer uygulamalarda temel işlem olarak kullanılmaktadır. Özellikle bant genişliđi düşük bant matrislerin, çok sayıda vektör ile çarpılmasında avantajları olan yeni bir yöntem sunulmuştur.

Tüm bu sayısal deneyler, iki farklı bilgisayar ortamında gerçekleştirilmiştir.

Anahtar Kelimeler: Bant matris, seyrek matris, çoklu, vektör, çarpma

To My Family...

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Assoc. Prof. Dr. Murat Manguođlu for his valuable guidance, suggestions and encouragement throughout this study. It is an honor for me to share his knowledge and wisdom.

I am very grateful to my commanders Nazım Yazır, Hakan Kurt, Bora Araz, Erkan Bulut, Utku Demir and all my colleagues, for their support and insight throughout this thesis.

The Masters of Science education of the author was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) with Program No: 2210.

The numerical calculations reported in this thesis were performed at TUBITAK ULAKBİM, High Performance and Grid Computing Center (TRUBA Resources) and Department of Computer Engineering, Middle East Technical University.

Finally, I am very thankful to my family for believing in me and supporting me during this study. I am greatly grateful to the sacrifice they made so that I could have the time to finish the thesis.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xii
LIST OF FIGURES	xiv
LIST OF ABBREVIATIONS	xvi
CHAPTERS	
1 INTRODUCTION	1
2 BACKGROUND AND RELATED WORK.....	3
2.1 Banded Matrix - Multiple Vectors Multiplication	3
2.2 Sparse Matrix - Multiple Vectors Multiplication.....	4
3 METHODS AND MOTIVATION	9
3.1 Banded Matrix - Multiple Vectors Multiplication	9
3.1.1 DGBMV	9
3.1.2 DSBMM.....	10
3.1.3 DSBMM2.....	12
3.2 Sparse Matrix - Multiple Vectors Multiplication.....	14
3.2.1 Reading Matrix.....	15
3.2.2 Partitioning.....	16

3.2.2.1	METIS	16
3.2.2.2	PATOH.....	17
3.2.3	Diagonal Block.....	20
3.2.3.1	Permutation	20
3.2.3.1.1	HSL MC73	21
3.2.3.1.2	RCM	22
3.2.4	Off-Diagonal Blocks	23
4	NUMERICAL EXPERIMENTS.....	25
4.1	Computing Platform.....	25
4.2	Programming Environment.....	26
4.3	Experiments and Results	27
4.3.1	Banded Matrix - Multiple Vectors Multiplication.....	27
4.3.2	Sparse Matrix - Multiple Vectors Multiplication	32
4.3.2.1	Matrix Collection	33
4.3.2.2	Effect of Permutation on Sequential Sparse Matrix - Multiple Vectors Multiplication.....	35
4.3.2.3	Parallel Scalability.....	39
4.3.2.3.1	METIS vs. PATOH	39
4.3.2.3.2	Parallel Scalability.....	43
5	CONCLUSION AND FUTURE WORK.....	47
	REFERENCES.....	49
	APPENDICES	
A	RESULTS OF BANDED MATRIX - MULTIPLE VECTORS MULTIPLICATION	53

B	RESULTS OF SPARSE MATRIX - MULTIPLE VECTORS MULTIPLICATION.....	59
----------	----------------------------------------------------------------------------	-----------

LIST OF TABLES

TABLES

Table 4.1 Platform specifications.....	25
Table 4.2 Programming environment specifications.....	26
Table 4.3 Sequential multiplication time using DGBMV and speedup of banded matrix - multiple vectors multiplication using DSBMM and DSBMM2 on NAR	27
Table 4.4 Sequential multiplication time using DGBMV and speedup of banded matrix - multiple vectors multiplication using DSBMM and DSBMM2 on MERCAN.....	30
Table 4.5 Matrices used for performance testing.....	33
Table 4.6 Sequential multiplication time of matrix in its original form and the times for obtaining the permutation using MC73 and RCM.....	36
Table 4.7 Sequential multiplication time and partitioning time of METIS and PATOH	40
Table A.1 Multiplication time of banded matrices, having 5, 10 and 20 lower bandwidth, with multiple vectors on NAR.....	55
Table A.2 Multiplication time of banded matrices, having 50, 100 and 200 lower bandwidth, with multiple vectors on NAR.....	56
Table A.3 Multiplication time of banded matrices, having 5, 10 and 20 lower bandwidth, with multiple vectors on MERCAN.....	57
Table A.4 Multiplication time of banded matrices, having 50, 100 and 200 lower bandwidth, with multiple vectors on MERCAN.....	58
Table B.1 Matrix reading time; Sequential multiplication time; Partitioning and parallel multiplication time of matrix partitioned using METIS on NAR61	
Table B.2 Partitioning and parallel multiplication time of matrix partitioned using PATOH on NAR.....	62

Table B.3 Matrix reading time; Sequential multiplication time; Partitioning and parallel multiplication time of matrix partitioned using METIS on MERCAN.....	63
Table B.4 Partitioning and parallel multiplication time of matrix partitioned using PATOH on MERCAN	64
Table B.5 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using MC73 on NAR	65
Table B.6 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using MC73 on NAR	66
Table B.7 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using MC73 on MERCAN.....	67
Table B.8 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using MC73 on MERCAN	68
Table B.9 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using RCM on NAR.....	69
Table B.10 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using RCM on NAR.....	70
Table B.11 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using RCM on MERCAN	71
Table B.12 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using RCM on MERCAN72	

LIST OF FIGURES

FIGURES

Figure 2.1 A 9×9 square and banded matrix	3
Figure 3.1 A 9×9 square and banded matrix divided into blocks using DSBMM. 10	
Figure 3.2 A 9×9 square and banded matrix divided into blocks using DSBMM212	
Figure 3.3 Sparsity structure of webbase-1M	15
Figure 3.4 Matrix webbase-1M is permuted with partition vector generated using METIS	17
Figure 3.5 Matrix webbase-1M is permuted with partition vector generated using PATOH.....	18
Figure 3.6 Matrix elements of master process after partitioning	19
Figure 3.7 Diagonal block of matrix of master process	19
Figure 3.8 Off-diagonal block of matrix of master process	19
Figure 3.9 Diagonal block of matrix of master process (empty off-diagonal part is removed).....	20
Figure 3.10 Diagonal block permuted with permutation vector generated using MC73	21
Figure 3.11 Diagonal block permuted with permutation vector generated using RCM	23
Figure 4.1 Speedup chart of banded matrix - multiple vectors multiplication for matrix having 100,000 rows on NAR.....	28
Figure 4.2 Speedup chart of banded matrix - multiple vectors multiplication for matrix having 1,500,000 rows on NAR.....	29

Figure 4.3 Speedup chart of banded matrix - multiple vectors multiplication for matrix having 100,000 rows on MERCAN	31
Figure 4.4 Speedup chart of banded matrix - multiple vectors multiplication for matrix having 1,500,000 rows on MERCAN	31
Figure 4.5 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on NAR (part 1 of 2)	37
Figure 4.6 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on NAR (part 2 of 2)	37
Figure 4.7 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on MERCAN (part 1 of 2)	38
Figure 4.8 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on MERCAN (part 2 of 2)	38
Figure 4.9 Speedup comparison chart of partitioning tools on NAR (part 1 of 2)	41
Figure 4.10 Speedup comparison chart of partitioning tools on NAR (part 2 of 2) ..	41
Figure 4.11 Speedup comparison chart of partitioning tools on MERCAN (part 1 of 2)	42
Figure 4.12 Speedup comparison chart of partitioning tools on MERCAN (part 2 of 2)	42
Figure 4.13 Speedup comparison chart of matrices on NAR (part 1 of 2)	44
Figure 4.14 Speedup comparison chart of matrices on NAR (part 2 of 2)	44
Figure 4.15 Speedup comparison chart of matrices on MERCAN (part 1 of 2).....	45
Figure 4.16 Speedup comparison chart of matrices on MERCAN (part 2 of 2).....	45

LIST OF ABBREVIATIONS

SpMV	:	Sparse Matrix – Vector Multiplication
SpMM	:	Sparse Matrix – Matrix Multiplication
ISDA	:	Invariant Subspace Decomposition Algorithm
BLAS	:	Basic Linear Algebra Subprograms
MKL	:	Math Kernel Library
RCM	:	Reverse Cuthill McKee
MPI	:	Message Passing Interface

CHAPTER 1

INTRODUCTION

In iterative solution of banded linear systems, banded matrix – vector multiplication is a crucial primitive. In the first part of this thesis, an improved algorithm is presented that has advantage especially for banded matrices that are dense within the band with a small bandwidth and multiplied by large number of vectors.

Sparse matrix – vector multiplication (SpMV) and sparse matrix – multiple vectors multiplication (SpMM) are another two important primitives, largely used in iterative linear system solvers and sparse eigenvalue solvers. Therefore, parallel scalabilities of SpMV and SpMM operations are crucial.

SpMV operation is defined as $y \leftarrow \alpha A x + \beta y$, where A is a sparse matrix, x and y are dense vectors. For simplicity α is assumed one and β is assumed zero. For each nonzero in matrix A , a_{ij} accessed only once, on the other hand, elements of x and y vectors are accessed multiple times. Thus, optimization of reusability of vector elements has a significant role in improving the cache utilization.

In distributed memory environments, SpMV operation can be performed concurrently within processes by distributing rows or columns (or both) of a sparse matrix A , input vector x and output vector y to processes. Each process multiplies nonzeros in A with input vector x and partially obtains the result in y . In order to provide efficient parallelization and to reduce communication cost between processes, distribution of nonzero elements has a significant role. Another important objective is balancing the computation load. In the second part of the thesis, partitioning tools and permutation algorithms are compared within different computing platforms and the matrices from different application areas.

The remaining of this thesis is arranged as follows. In Chapter 2, background information is given and related work is reviewed. In Chapter 3, the methods used in banded matrix – multiple vectors multiplication and sparse matrix – multiple vectors multiplication are proposed and compared with highly optimized routines in existing libraries. In Chapter 4, computing platform and programming environment is given. Moreover, numerical experiments are presented. Conclusion and future work is stated in Chapter 5. The complete measurements of numerical experiments are presented in the Appendices.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Banded Matrix – Multiple Vectors Multiplication

A banded matrix is a matrix in which the non-zero elements are located around the main diagonal, i.e., for all elements outside a banded area are zero. In a formal way, take an $n \times n$ matrix A , a_{ij} is the element of i^{th} row and j^{th} column; $a_{ij} = 0$ if $i > j + ml$ or $j > i + mu$, where $ml, mu \geq 0$. The lower and upper bandwidth are denoted by ml and mu , respectively. In Figure 2.1, a 9×9 square and banded matrix having the same lower and upper bandwidth ($ml = mu = 2$) is given. The bandwidth of the matrix is $ml + mu + 1$; for the sample matrix below, it has a value of five.

$$\begin{array}{c}
 \begin{array}{c} \leftarrow mu \rightarrow \\ \uparrow ml \\ \downarrow \end{array} \\
 A = \begin{bmatrix}
 a_{11} & a_{12} & a_{13} & 0 & . & . & . & . & 0 \\
 a_{21} & a_{22} & a_{23} & a_{24} & 0 & . & . & . & . \\
 a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 & . & . & . \\
 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & 0 & . & . \\
 . & 0 & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & 0 & . \\
 . & . & 0 & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & 0 \\
 . & . & . & 0 & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} \\
 . & . & . & . & 0 & a_{86} & a_{87} & a_{88} & a_{89} \\
 0 & . & . & . & . & 0 & a_{97} & a_{98} & a_{99}
 \end{bmatrix}
 \end{array}$$

Figure 2.1 A 9×9 square and banded matrix

Banded matrix multiplication is needed in Invariant Subspace Decomposition Algorithm (ISDA) [1]. Even though, ISDA uses multiplication of banded matrices, it can be formed as banded matrix – multiple vectors multiplication. Banded matrix – vector multiplication is used as a primitive in solving banded linear systems and eigenvalue problems. These systems arise in the discretization of several partial and ordinary differential equations [2]–[4], computational fluid dynamics [5] and computational nano-electronics [6].

Tsao and Turnbull compared several methods for multiplying banded matrices. They stated that good results could be achieved for matrices having bandwidth smaller than the order of matrix [7]. Remon and Quintana-Orti stated that in existing Basic Linear Algebra Subprograms (BLAS) libraries, there is no BLAS Level 3 routines for banded matrices. In fact, BLAS Level 2 routines do not sufficiently optimize the operations on banded matrices [8]. There are two routines in Intel Math Kernel Library (MKL) [9] that can be used for banded matrix – multiple vectors multiplication. The first one is xCSRMM, a sparse BLAS Level 3 routine, suits for SpMM. The second one is xGBMV, a BLAS Level 2 routine, suits for banded matrix – vector multiplication. Using xGBMV multiple times for each vector appears to be more appropriate than using former since the matrix is banded and not sparse. Remon and Quintana-Orti also stated that the performance of the banded BLAS Level 2 routine is highly dependent on the bandwidth and matrix size [8].

Spike [10], a parallel environment for solving banded linear systems, has banded primitives. Polizzi splits banded matrix into square dense blocks and triangular matrices. The method used for banded matrix – matrix multiplication in Spike is DSBMM. In this thesis, an improved version of it, named DSBMM2 is developed. The algorithm and results are presented in Section 3.1.

2.2 Sparse Matrix – Multiple Vectors Multiplication

SpMV operation is defined as $y \leftarrow \alpha A x + \beta y$, where A is an $n \times n$ sparse matrix, x and y are dense vectors of length n . For simplicity α is assumed one and β is assumed zero. For each nonzero in A , a_{ij} accessed only once, on the other hand, elements of x

and y are accessed multiple times. Moreover, elements of A and y are accessed contiguously but there is a random access to elements of vector x . Thus, reusability of vector elements should be optimized to increase utilization. This optimization has been well studied before, see [11]–[14] for example.

Sparse matrix – multiple vectors multiplication (can also be defined as sparse matrix – general matrix multiplication) computes a set of dense output vectors Y as a product of a sparse matrix A and a set of dense input vectors X , can be shown as,

$$Y \leftarrow \alpha A X + \beta Y \quad (1)$$

For simplicity, it is considered that α is one and β is zero. SpMM is used when solving a blocked linear system with multiple right-hand sides [15], and in blocked eigenvalue algorithms, such as block Lanczos and block Arnoldi methods [16]–[18].

If matrix A is multiplied with only one vector, elements of A are accessed only once. In this thesis, there are multiple vectors, eventually, reusability of matrix elements are needed to be well considered as well. Im, Yelick and Vuduck showed that in sparse matrix – multiple vectors multiplication, reuse of matrix elements in cache, which is not possible with a single vector, provides large opportunities for performance gains, such as storing i^{th} row of all vectors contiguously [19]. This optimization proposed in [19] was implemented by looping across the fixed number of vectors as fully unrolled. In this thesis, it is implemented without unrolling and looping. The i^{th} row of all vectors are multiplied with nonzeros in matrix as a block. Block size is equal to the number of vectors.

Sparse algebra kernels have low processor utilization, typically in the range of 10 - 20% of processor peak [20]. One of the reason for the low utilization is that the amount of computational power is increasing with a higher rate than the rate of increase of memory bandwidth [21]. This difference is stated as “a memory bandwidth starved multicore world” by Williams [22].

The low utilization could be improved by lowering indexing overheads of matrix A . In order to lower the overheads associated with storing and accessing elements of sparse

matrices, Kannan introduced a blocked sparse format, called mapped blocked row sparse format that can be used in sequential SpMV [21]. Pinar and Heath proposed packing all the nonzeros in contiguous locations into a block, named blocked compressed row storage format and compared with 1×2 blocks [14].

Another option to lowering indexing overhead is register blocking. Geus and Röllin used fixed size small dense blocks in sequential SpMV [12]. Toledo proposed handling the 1×2 blocks of a matrix separately [13]. Im, Yelick and Vuduck used rectangular register blocking in different platforms. They conclude that a small change in block size can make a large difference in performance. However the reason of it is not clear, they stated that compiler and memory structure are important factors [19]. Generally, in all previous studies about register blocking optimization, it is the common outcome that machine specific tuning is worthy.

To balance the workload of parallel SpMM operation, matrix partitioning is used in general. It means that subset of matrix A and corresponding input vectors X are distributed according to partition vector. Hence, each process multiplies and obtains output vectors Y that it owns.

There are two types of one-dimensional decomposition of sparse matrices, which are graph partitioning and hypergraph partitioning. Graph partitioning approximates the volume of nonzero elements in off-diagonal matrix blocks. But hypergraph partitioning reflects the actual communication volume by making nonzero elements in off-diagonal matrix blocks to be column aligned [23]. In this thesis, both a graph partitioning tool METIS [24] and a hypergraph partitioning tool PATOH [25] are used.

Another option to increase low utilization is reducing irregular memory accesses. It can be done by reordering the matrix to access input vector elements contiguously and to have a high cache reuse. Pinar and Heath stated that noteworthy improvement in SpMV performance could be achieved by reordering the matrix, which they showed to be NP-Complete. They also stated that the cost of reordering is often amortized over repeated SpMV operations with the same matrix, which also means SpMM operation. They proposed traveling salesman problem ordering and compared with Reverse

Cuthill McKee (RCM) ordering [14]. Toledo explored Cuthill McKee ordering yields an excellent results on a variety of matrices [13]. In this thesis, RCM [26] and HSL MC73 [27] algorithms are investigated for improving the cache utilization in parallel sparse matrix – multiple vectors multiplication.

CHAPTER 3

METHODS AND MOTIVATION

3.1 Banded Matrix – Multiple Vectors Multiplication

A banded matrix A with bandwidth ml , input vector x and output vector y are given. Banded matrix – vector multiplication is defined as $y \leftarrow \alpha A x + \beta y$. For simplicity, it is considered that α is one and β is zero. In Figure 2.1, a 9×9 square and banded matrix having the same lower and upper bandwidth with value of two is given.

In this chapter, serial implementation of banded matrix – multiple vectors multiplication primitives DGBMV and DSBMM are studied. Moreover, an improved variation of DSBMM, which is called DSBMM2, is proposed and implemented.

3.1.1 DGBMV

In Intel MKL library or in any other BLAS implementations, there is no BLAS Level 3 routine for banded matrix – multiple vectors multiplication. Instead, there is a BLAS Level 2 routine called DGBMV, double-precision type of GBMV. In this thesis, it is used for multiple vectors multiplication by calling it k times, where k is the number of vectors.

DGBMV implementation is used as a reference in banded matrix – multiple vectors multiplication as it is the only banded matrix – vector multiplication routine in existing optimized BLAS libraries.

3.1.2 DSBMM

DSBMM [28] is a method in Spike library [10], which multiplies banded matrices with multiple vectors. The method accepts only structurally symmetric matrices.

DSBMM divides matrix into square blocks of sizes ml and triangular blocks. Square blocks are multiplied with the block of vectors using DGEMM, a BLAS Level 3 routine used for general matrix – matrix multiplication. Triangular blocks are multiplied using DTRMM, another BLAS Level 3 routine used for triangular matrix – matrix multiplication. In Figure 3.1, square blocks are marked with gray background; triangular blocks are marked with polka dot background.

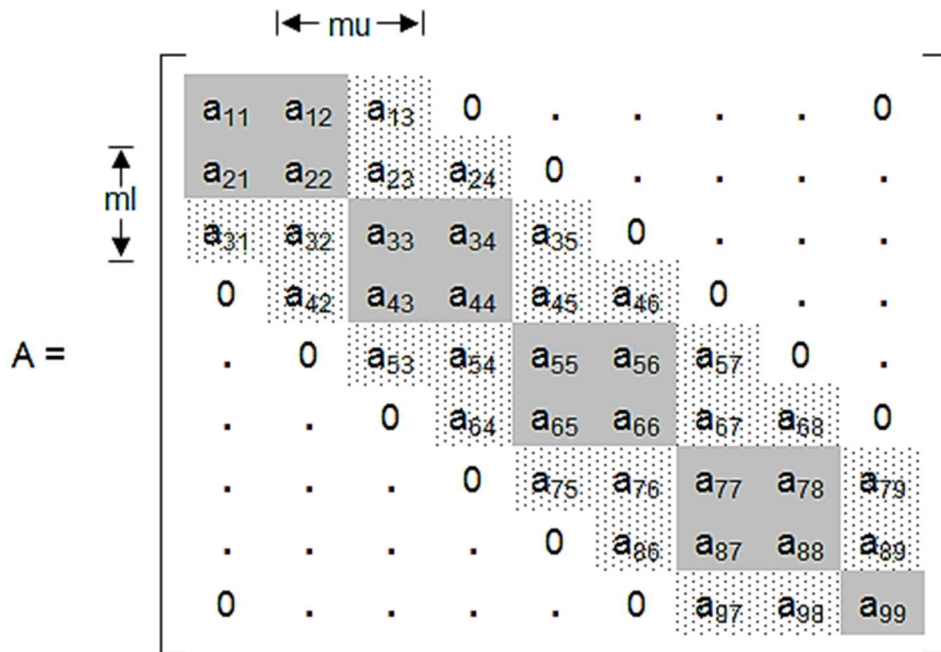


Figure 3.1 A 9×9 square and banded matrix divided into blocks using DSBMM

While FORTRAN programming language uses column-major order for array storage, C programming language uses row-major order. As DSBMM is implemented in FORTRAN, diagonals are given in columns so that the matrix elements are contiguous.

The pseudo-code of DSBMM for the given $n \times n$ matrix A , input vectors X , output vectors Y , number of rows of matrix and vectors n , number of vectors k and lower/upper bandwidth ml , is given in Algorithm 1.

Algorithm 1 Pseudo-code of DSBMM (A, X, Y, n, k, ml)

```
1:   $bl \leftarrow n / ml$  ▶ Number of square blocks
2:  for each square block with size  $ml$ 
3:      call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml$ )
4:  end for
5:   $last\_bl \leftarrow n - (bl * ml)$  ▶ Size of last square block
6:  if  $last\_bl > 0$  then
7:      call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, last\_bl$ )
8:  end if
9:  for each lower sub-diagonal (triangular) block do
10:      $Z (ml, k) \leftarrow X_{partial}$  ▶  $Z$  is a temporary matrix
11:     call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml, UPPER$ )
12:     Add  $Z$  to  $Y_{partial}$ 
13:  end for
14:  if  $last\_bl > 0$  then
15:      $Z (last\_bl, k) \leftarrow X_{partial}$ 
16:     call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, last\_bl, UPPER$ )
17:     Add  $Z$  to  $Y_{partial}$ 
18:     /* Rectangular part of the last lower sub-diagonal block */
19:     call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml - last\_bl$ )
20:  end if
21:  for each upper sub-diagonal (triangular) block do
22:      $Z (ml, k) \leftarrow X_{partial}$ 
23:     call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml, LOWER$ )
24:     Add  $Z$  to  $Y_{partial}$ 
25:  end for
26:  if  $last\_bl > 0$  then
```

Algorithm 1 Pseudo-code of DSBMM (A, X, Y, n, k, ml) (continued)

```

27:       $Z(last\_bl, k) \leftarrow X_{partial}$ 
28:      call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, last\_bl, LOWER$ )
29:      Add  $Z$  to  $Y_{partial}$ 
30:      /* Rectangular part of the last upper sub-diagonal block */
31:      call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml - last\_bl$ )
32:  end if

```

3.1.3 DSBMM2

In this thesis, an improved variation of DSBMM, which is called DSBMM2, is proposed and implemented. DSBMM2 has slightly larger square blocks with the size $(ml + 1)$. Eventually the number of triangular blocks become fewer, because dense square blocks are larger. Hence, a better utilization of the cache is obtained. An example partitioning using DSBMM2 for the same matrix in Figure 3.1 is given in Figure 3.2.

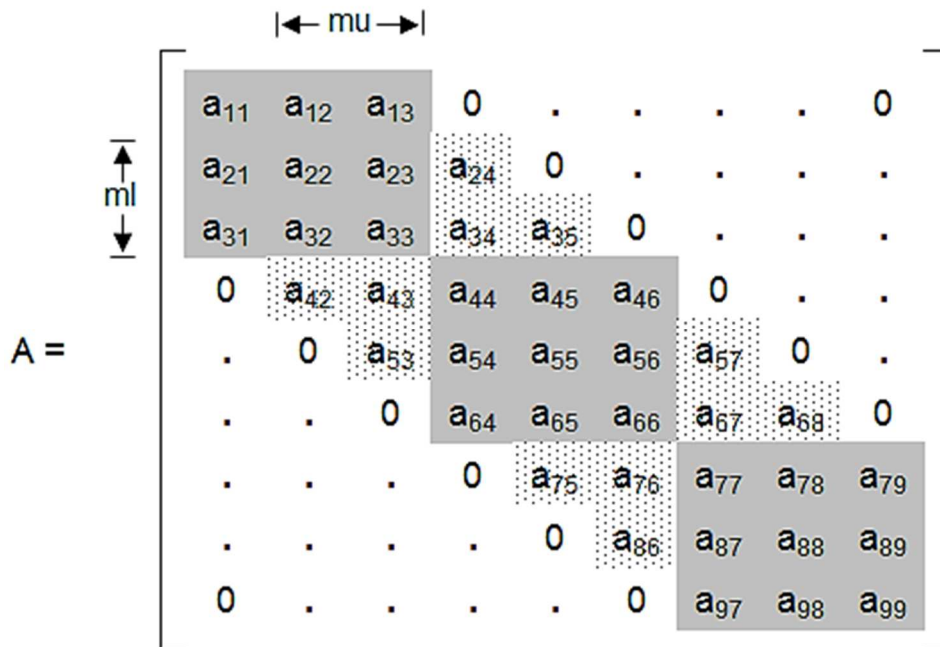


Figure 3.2 A 9×9 square and banded matrix divided into blocks using DSBMM2

DSBMM2 is implemented in C programming language, so diagonals are given in rows. The pseudo-code of DSBMM2 for the given $n \times n$ matrix A , input vectors X , output vectors Y , number of rows of matrix and vectors n , number of vectors k and lower/upper bandwidth ml , is given in Algorithm 2.

Algorithm 2 Pseudo-code of DSBMM2 (A, X, Y, n, k, ml)

```

1:   $bl \leftarrow n / (ml + 1)$  ▶ Number of square blocks
2:  for each square block with size  $ml$ 
3:      call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml + 1$ )
4:  end for
5:   $last\_bl \leftarrow n - (bl * (ml + 1))$  ▶ Size of last square block
6:  if  $last\_bl > 0$  then
7:      call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, last\_bl$ )
8:  end if
9:  for each lower sub-diagonal (triangular) block do
10:      $Z (ml, k) \leftarrow X_{partial}$  ▶  $Z$  is a temporary matrix
11:     call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml, UPPER$ )
12:     Add  $Z$  to  $Y_{partial}$ 
13:  end for
14:  if  $last\_bl > 0$  then
15:      $Z (last\_bl, k) \leftarrow X_{partial}$ 
16:     call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, last\_bl, UPPER$ )
17:     Add  $Z$  to  $Y_{partial}$ 
18:     /* Rectangular part of the last lower sub-diagonal block */
19:     call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml - last\_bl$ )
20:  end if
21:  for each upper sub-diagonal (triangular) block do
22:      $Z (ml, k) \leftarrow X_{partial}$ 

```

Algorithm 2 Pseudo-code of DSBMM2 (A, X, Y, n, k, ml) (continued)

```
23:      call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml, LOWER$ )
24:      Add  $Z$  to  $Y_{partial}$ 
25:  end for
26:  if  $last\_bl > 0$  then
27:       $Z (last\_bl, k) \leftarrow X_{partial}$ 
28:      call DTRMM ( $A_{partial}, X_{partial}, Y_{partial}, k, last\_bl, LOWER$ )
29:      Add  $Z$  to  $Y_{partial}$ 
30:      /* Rectangular part of the last upper sub-diagonal block */
31:      call DGEMM ( $A_{partial}, X_{partial}, Y_{partial}, k, ml - last\_bl$ )
32:  end if
```

In both DSBMM and DSBMM2, matrix is multiplied with input vectors not one by one but in blocks. Hence, they have better utilization than DGBMV for multiple vectors.

3.2 Sparse Matrix – Multiple Vectors Multiplication

In this chapter, the following techniques are used while performing sparse matrix – multiple vectors multiplication.

- Partitioning the matrix to move most nonzeros in the diagonal block.
- Permuting the matrix to move most nonzeros contiguously.
- Storing vectors to be able to do multiplication as a block operation.

Spy plots of a sample sparse matrix – multiple vectors multiplication have been given in each subsection of this section. As an example, “webbase-1M” matrix has been partitioned using METIS and has been multiplied with multiple vectors within four processes.

3.2.1 Reading Matrix

The matrices used in this section (given in Section 4.3.2.1) are all in Matrix Market format [29]. In Matrix Market format, general information (e.g. name, title, ID, kind, author, etc.) and structure summary (e.g. sparse or dense, real or pattern, symmetric or general) are given on the header of the file. MMIO library [30] is used when reading the header to check whether the matrix is suitable for the operations used in this thesis. Following the header lines, there are triplets (i, j, a_{ij}) for each nonzero, representing row index, column index and matrix value respectively.

The matrices used in this section are all sparse and square; entries of them are double-precision numbers. The matrices are read by master MPI process (process having rank of zero). Other processes get nonzeros via MPI communication routines. In Figure 3.3, there is a spy plot showing a sample matrix, webbase-1M, after reading it from file.

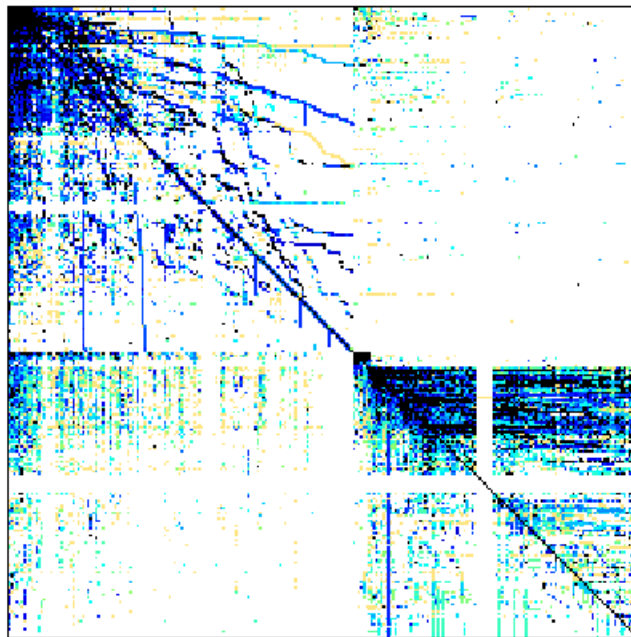


Figure 3.3 Sparsity structure of webbase-1M

3.2.2 Partitioning

One dimensional block distribution is distributing rows or columns of a matrix to different processes. In this thesis, row-wise distribution is used. Hence, for an $n \times n$ matrix and number of parts k , n / k rows are assigned to each process.

When multiplying sparse matrix with vector(s), the nonzeros in the diagonal blocks do not need to access vector elements on other processes. Partitioning tools are used to move most nonzeros of the matrix to the diagonal blocks.

Partitioning tools, such as METIS and PATOH, generate a permutation vector (which gives the permutation matrix P) to partition the matrix.

$$A_0 \leftarrow P A P^T \quad (2)$$

Rows and columns of the matrix are permuted with permutation vector and transpose of permutation vector respectively. After the symmetric permutation is applied, a permuted matrix A_0 is obtained. Equation of this operation is given above. Input and output vectors are permuted with the same permutation vector as well.

In this thesis, row-wise partitioning is used, so row i of the matrix, input and output vectors are all assigned to the same process. Calling partitioning method, getting the partitioning vector and permuting the matrix according to partition vector are all done by master MPI process. Number of partitions is equal to the number of running MPI processes. Hence, each process gets a part of matrix and vectors.

3.2.2.1 METIS

METIS is a sequential tool for partitioning graphs, partitioning finite element meshes and computing fill-reducing orderings of sparse matrices [24]. In addition, there is another tool called ParMETIS [31]. ParMETIS is an MPI-based parallel version of METIS.

There are two possible algorithms while partitioning graphs using METIS. They are multilevel recursive bisection and multilevel k-way partitioning.

In this thesis, k-way partitioning is used with the “minimizing total communication volume” parameter is set. The other parameter for objective type is edge-cut minimization, which is proven that it does not model the actual communication volume by Catalyurek and Aykanat [32], [33].

METIS requests input matrix to be symmetric, so for non-symmetric matrices, $|A| + |A|^T$ is given as the input. For the sample matrix in Figure 3.3, it is permuted with partition vector generated using METIS. The resulting reordered matrix using four partitions is given in Figure 3.4.

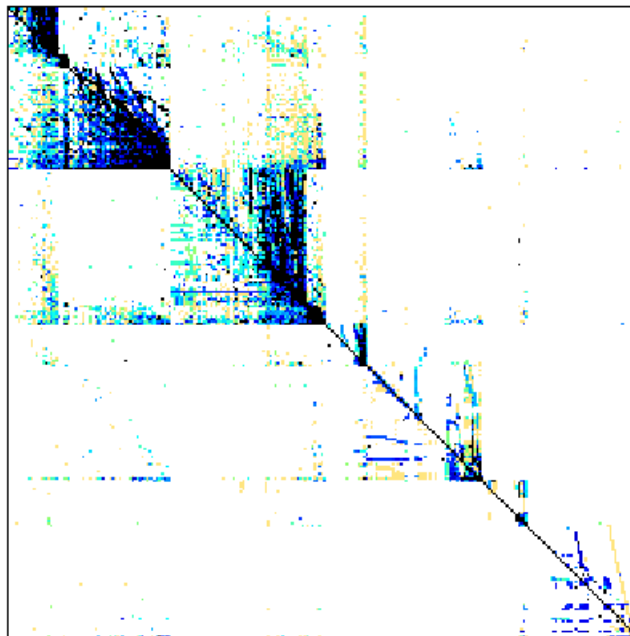


Figure 3.4 Matrix webbase-1M is permuted with partition vector generated using METIS

3.2.2.2 PATOH

PATOH is a sequential multi-level and multi-constraint hypergraph partitioning tool. It divides a hypergraph into two or more roughly equal sized parts such that a cost function on the hyperedges connecting vertices in different parts is minimized [25], [32].

Before calling partitioning method of PATOH, parameters are initialized to their default values and memory allocation is done. Afterwards, partitioning method is called with set as to use recursive multilevel hypergraph bisection algorithm. For the sample matrix in Figure 3.3, it is permuted with partition vector generated using PATOH. The resulting reordered matrix using four partitions is given in Figure 3.5.

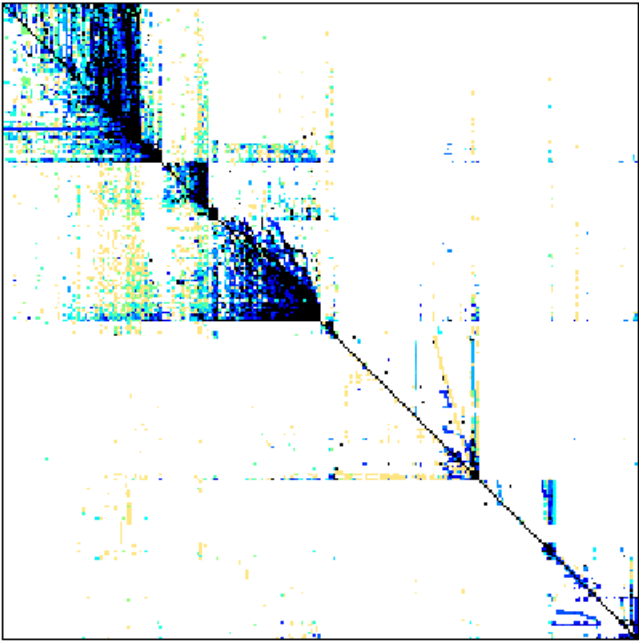


Figure 3.5 Matrix webbase-1M is permuted with partition vector generated using PATOH

Devine et al. stated that hypergraphs can model communication volume more precisely and can represent non-symmetric problems better [34]. In this thesis, both a graph partitioner, METIS and a hypergraph partitioner, PATOH are used. The results are given in Section 4.3.2.3.1.

Once the permutation of matrix and input vectors are finished, it is time to send them other MPI processes to do multiplication. Matrix elements of master process after it is partitioned using METIS, are given in Figure 3.6.

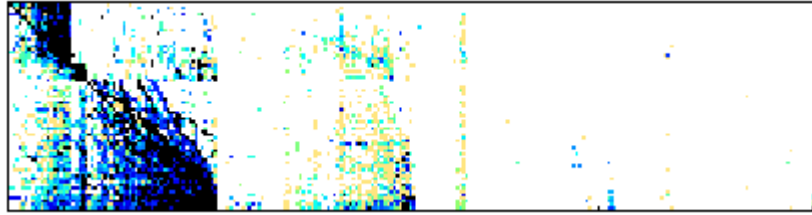


Figure 3.6 Matrix elements of master process after partitioning

After each MPI process obtains its block matrix, it is further partitioned into diagonal and off-diagonal blocks, shown in Figure 3.7 and Figure 3.8 respectively. This operation can be shown as;

$$D + R \leftarrow A \quad (3)$$



Figure 3.7 Diagonal block of matrix of master process

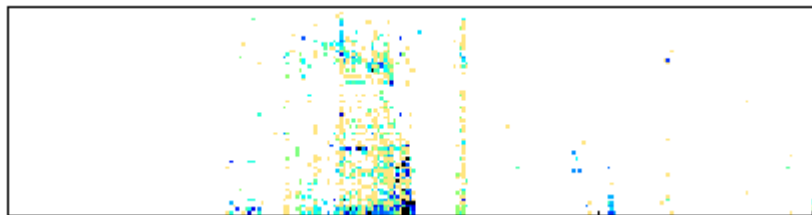


Figure 3.8 Off-diagonal block of matrix of master process

There is no need to communicate with other processes when multiplying nonzeros of D (diagonal block). On the contrary, communication is required when multiplying nonzeros of R (off-diagonal block). Because some elements of input vectors of other processes are required to do multiplication.

3.2.3 Diagonal Block

Diagonal block, D has much more nonzeros than off-diagonal block owing to partitioning tools. To illustrate, empty off-diagonal part in Figure 3.7 is removed and it becomes as in Figure 3.9.

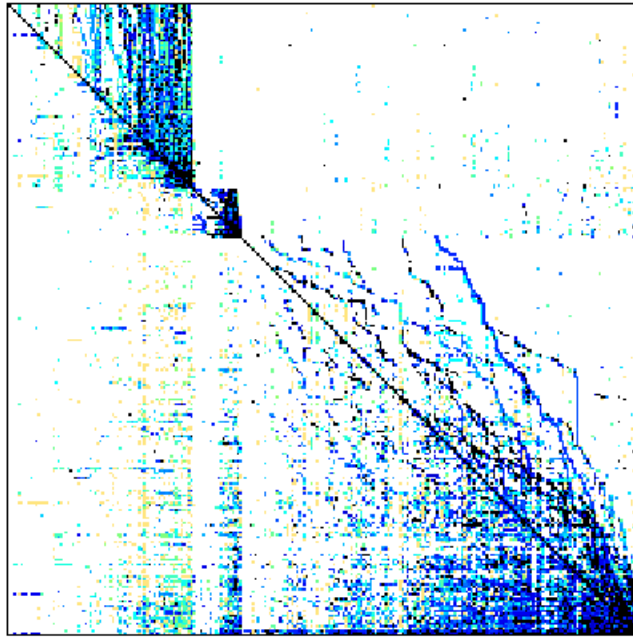


Figure 3.9 Diagonal block of matrix of master process (empty off-diagonal part is removed)

While multiplying sparse matrix with multiple vectors, DCSRMM, a sparse BLAS Level 3 routine is used. In order to improve cache-hit ratio, i^{th} rows of all vectors are stored contiguously. Hence, it provides usability of BLAS Level 3 routines rather than calling BLAS Level 2 routines many times for each vector.

3.2.3.1 Permutation

The objective of using permutation is to improve the speedup of sparse matrix – multiple vectors multiplication by moving most nonzeros contiguously as much as possible. To achieve that matrices are permuted with two different algorithms. After the permutation is done, the matrix is multiplied with multiple vectors using DCSRMM routine. Results of the experiments are given in Section 4.3.2.2.

Both permutation algorithms, HSL MC73 and RCM request input matrix to be symmetric, so for non-symmetric matrices, $|A| + |A|^T$ is given as the input matrix.

3.2.3.1.1 HSL MC73

HSL MC73 is a library that computes Fiedler vector of Laplacian matrix and computes a symmetric permutation that aims to reduce the profile and wavefront [35]. In this thesis, MC73 is used to move most nonzeros of matrix near the main diagonal.

Three different algorithms can be used while computing a symmetric permutation. They are multilevel Sloan, multilevel spectral ordering and hybrid ordering algorithms. In this thesis, multilevel spectral ordering algorithm is used. This algorithm computes the approximate Fiedler vector of the Laplacian of each component and then sorts the entries of this vector in non-decreasing order [27].

MC73 tries to move most all nonzeros near the main diagonal as it can be seen in figure below. For the sample diagonal block in Figure 3.9, it is permuted with the permutation vector generated using MC73. The resulting diagonal block is given in Figure 3.10.

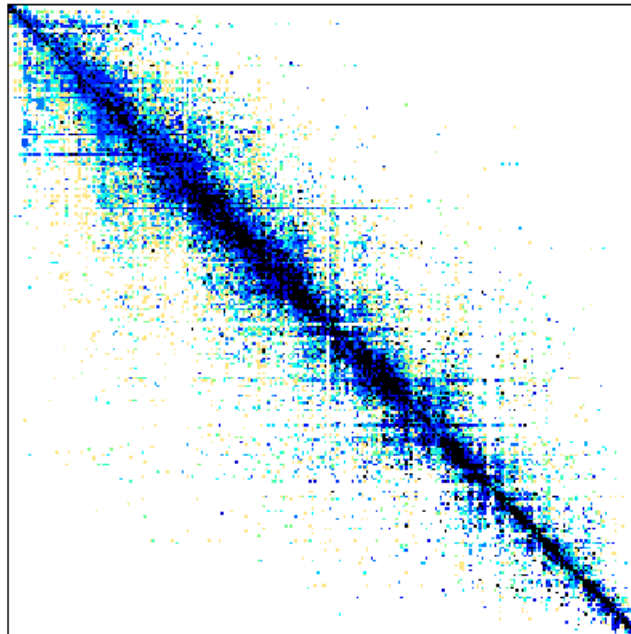


Figure 3.10 Diagonal block permuted with permutation vector generated using MC73

3.2.3.1.2 RCM

Reverse Cuthill McKee (RCM) algorithm [26] computes a symmetric permutation that reduces the bandwidth of sparse symmetric matrices. Algorithm of RCM for the given graph $G(n)$ is given below [36].

Algorithm 3 Algorithm of RCM

```
1:  $Q \leftarrow \{ \}$  ▶ Initialize an empty queue,  $Q$ 
2:  $R \leftarrow \{ \}$  ▶ Initialize an empty result array,  $R$ 
3: if there are unexplored nodes then
4:      $P \leftarrow$  The node with the lowest degree in  $G(n)$ 
5:     Add  $P$  in the first free position of  $R$ 
6:      $Q \leftarrow Q + \{$  All the nodes adjacent with  $P$  in the increasing order of
       their degree  $\}$ 
7:      $C \leftarrow$  The first node from the queue
8:      $Q \leftarrow Q - \{ C \}$ 
9:     while  $Q \neq \{ \}$  do
10:         if  $C$  has not previously been inserted in  $R$  then
11:             Add  $C$  in the first free position of  $R$ 
12:              $Q \leftarrow Q + \{$  All the neighbors of  $C$  that are not in  $R$  in
               the increasing order of their degree  $\}$ 
13:         end if
14:     end while
15: end if
16: Swap  $R[i]$  with  $R[n + 1 - i]$  ▶ Reverse the order of elements in  $R$ 
```

The vector elements accessed while multiplying i^{th} row of the sparse matrix are accessed again in the following row, owing to reduced bandwidth that RCM algorithm provides. In this thesis, RCM is used and advantage of reduced bandwidth is observed. Results of the experiments are given in Section 4.3.2.2.

For the sample diagonal block in Figure 3.9, it is permuted with the permutation vector generated using RCM. The resulting diagonal block is given in Figure 3.11.

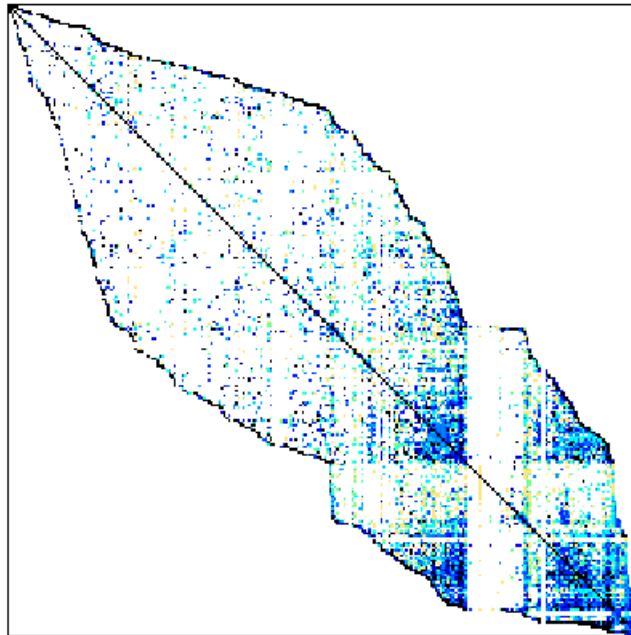


Figure 3.11 Diagonal block permuted with permutation vector generated using RCM

3.2.4 Off-Diagonal Blocks

Off-diagonal blocks, R , have much smaller number of nonzeros than the diagonal blocks. While multiplying nonzeros in R with input vectors, communication is required. Because vector elements that is going to be multiplied with nonzeros in off-diagonal block, are all owned by other processes. While sending these vector elements, non-blocking MPI send routines are used to have the possibility to continue computation. Upon vector elements are received, they are multiplied with the nonzeros in R .

Im, Yelick and Vuduck stated that for computations involving multiple vectors, reorganizing them to perform the entire set of multiplications as a single operation provides significant performance improvement [19]. In this thesis, i^{th} row of all vectors are stored contiguously. Hence, reuse of nonzeros in matrix R has been provided when multiplying.

Output vectors Y is formed after multiplying all nonzeros in D and R . If the matrix A is permuted with HSL MC73 or RCM, inverse permutation should be done to achieve final output vectors.

CHAPTER 4

NUMERICAL EXPERIMENTS

In the previous chapter, the methods used in implementation and the algorithm that is developed are described. In this chapter, the computing and programming environment are given and performance results are presented.

4.1 Computing Platform

The numerical computations reported in this thesis are performed using two different computing platforms. NAR is a cluster at Department of Computer Engineering, Middle East Technical University [37]. MERCAN is the other cluster at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA Resources) [38]. Specifications of two platforms are given below. Entire node is allocated while operating.

Table 4.1 Platform specifications

<i>Specification</i>	<i>NAR</i>	<i>MERCAN</i>
<i>Architecture</i>	Intel Xeon E5430	AMD Opteron 6176
<i>CPU Frequency</i>	2.66 GHz	2.3 GHz
<i>Number of CPUs</i> <i>Number of Cores</i>	2 CPU x 4 core	2 CPU x 12 core
<i>L1 Cache</i>	2 x 4 x 32 KB instruction	2 x 12 x 64 KB instruction
	2 x 4 x 32 KB write-back data	2 x 12 x 64 KB data
<i>L2 Cache</i>	2 x 2 x 6 MB	2 x 12 x 512 KB
<i>L3 Cache</i>	N / A	2 x 2 x 6 MB

Table 4.1 Platform specifications (continued)

<i>RAM</i>	8 x 2 GB 667 MHz (total 16 GB)	128 GB 1600 MHz
<i>Network</i>	20 Gbps Infiniband	40 Gbps QDR Infiniband

4.2 Programming Environment

Specifications of programming environment are stated in Table 4.2, followed by versions of other used libraries.

Table 4.2 Programming environment specifications

<i>Specification</i>	<i>NAR</i>	<i>MERCAN</i>
Operating System	Scientific Linux 5.2 64-bit	Scientific Linux 6.2 64-bit
Linux Kernel	2.6.18-92.1.17.el5 x86_64	2.6.32-220.23.1.el6 x86_64
MPI	MVAPICH2 v1.2p1	OpenMPI v1.4.3
MKL	Intel MKL v10.1	Intel MKL v11.1.073
Fortran Compiler	ifort v11.0	ifort v12.1.3
C Compiler	mpicc (icc v11)	mpicc (icc v12.1.3)

METIS version 5.1.0, a fill-reducing matrix reordering and a sequential graph partitioning algorithm; PATOH version 3.2, a multilevel hypergraph partitioning algorithm; HSL MC73 [35] version 1.2, a fast multilevel fiedler and profile reduction algorithm; RCM [39], an algorithm that reorders a sparse matrix into a band matrix with a small bandwidth; Sparsekit [40], a basic toolkit for sparse matrix operations; MMIO library [30], a library for files in Matrix Market format; CSPY, a MATLAB function in CSparse library [41] for spy plotting matrices, are used in the implementation of this thesis.

4.3 Experiments and Results

4.3.1 Banded Matrix – Multiple Vectors Multiplication

In this section, only square, banded and structurally symmetric (means having the same upper and lower bandwidths) matrices are used. Matrix entries are double-precision numbers and they are created randomly.

The results are evaluated in terms of speedup varying the number of rows, number of vectors, bandwidth, implementation type and computing platform. Multiplication time using DGBMV (in seconds) and speedup values using DSBMM and DSBMM2 are given in Table 4.3 and Table 4.4 for NAR and MERCAN platforms respectively. No threads are used in the implementation of this section and the implementation runs sequentially.

Table 4.3 Sequential multiplication time using DGBMV and speedup of banded matrix – multiple vectors multiplication using DSBMM and DSBMM2 on NAR

Number of Rows	Lower Bandwidth	Implementation Type	Number of Vectors					
			1		10		100	
			DGBMV Mult. Time(sec)	Speedup	DGBMV Mult. Time(sec)	Speedup	DGBMV Mult. Time(sec)	Speedup
100,000	10	DSBMM	0.01	0.25	0.07	0.91	0.71	1.55
100,000	10	DSBMM2		0.24		1.14		2.42
100,000	50	DSBMM	0.03	0.52	0.27	2.69	2.71	5.25
100,000	50	DSBMM2		0.34		2.10		5.52
100,000	200	DSBMM	0.09	0.70	0.92	3.89	9.23	8.18
100,000	200	DSBMM2		0.49		2.63		7.86
1,500,000	10	DSBMM	0.11	0.25	1.08	0.91	10.77	1.52
1,500,000	10	DSBMM2		0.24		1.15		2.44
1,500,000	50	DSBMM	0.41	0.52	4.07	2.70	40.71	5.26
1,500,000	50	DSBMM2		0.33		2.11		5.53
1,500,000	200	DSBMM	1.39	0.70	13.86	3.88	138.59	8.18
1,500,000	200	DSBMM2		0.49		2.62		7.86

The matrices used in test case have 100,000, 400,000, 1,500,000 and 3,000,000 rows. Only matrices having 100,000 and 1,500,000 rows are shown in this section. Remaining results are given in the Appendix A.

Eighteen vertical bars are shown in-group of six for varying number of vectors in each of Figure 4.1, Figure 4.2, Figure 4.3 and Figure 4.4. Each group represents speedup values of DSBMM and DSBMM2 relative to DGBMV for each bandwidth.

Chosen values for number of vectors are 1, 10 and 100; for lower bandwidth are 10, 50, and 200. Detailed test results can be found in Appendix A.

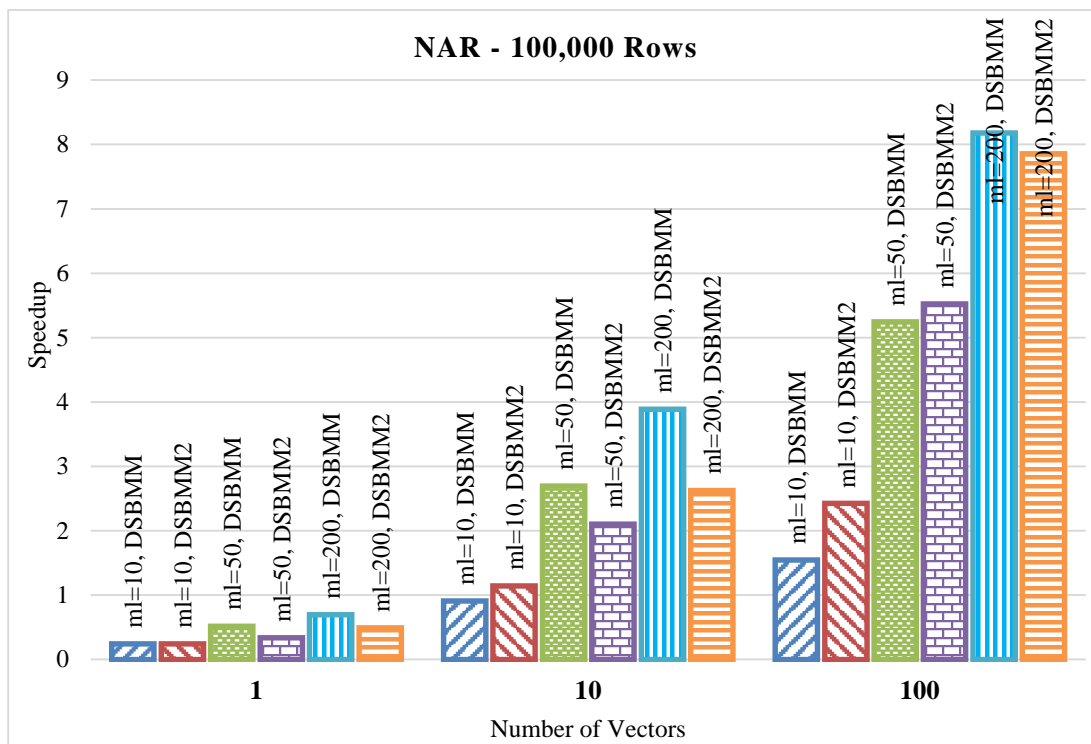


Figure 4.1 Speedup chart of banded matrix – multiple vectors multiplication for matrix having 100,000 rows on NAR

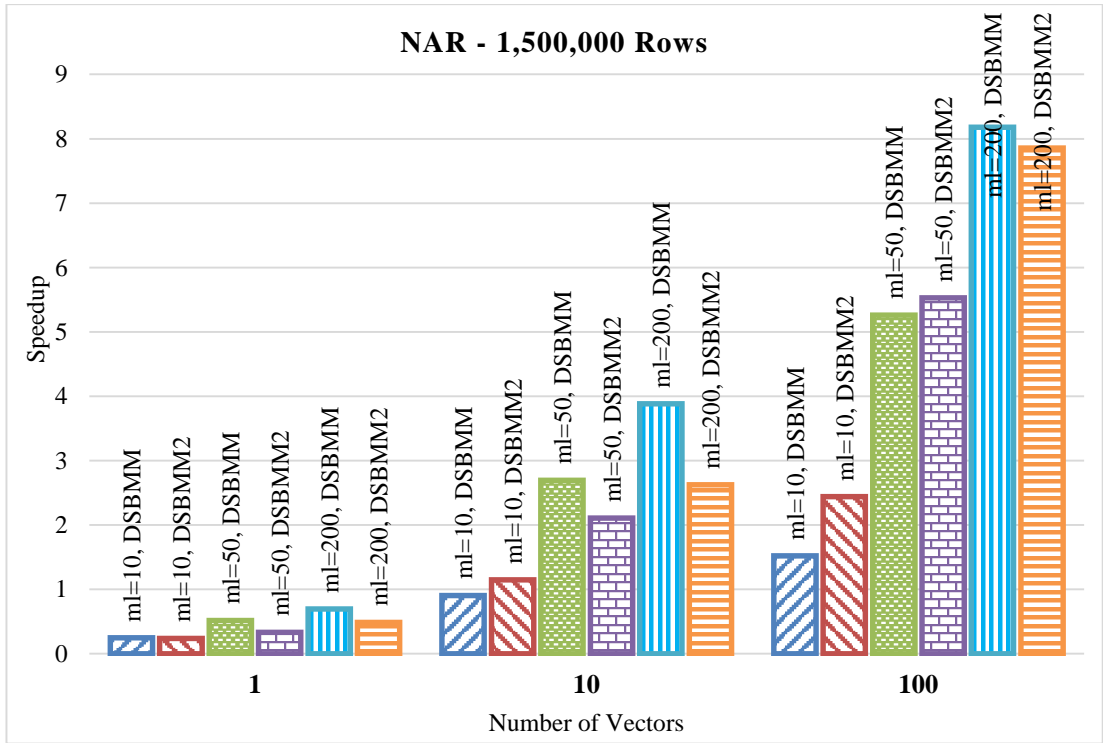


Figure 4.2 Speedup chart of banded matrix – multiple vectors multiplication for matrix having 1,500,000 rows on NAR

Figure 4.1 and Figure 4.2 show that DSBMM and DSBMM2 have a better speedup while number of vectors increases. The reason for this behavior is that when the number of vectors increases, more consecutive vector entries stay loaded in the cache and reusability of the values in cache is much higher than DGBMV. DSBMM2 is roughly 1.6 times faster than DSBMM if the bandwidth is 10 and the number of vectors is 100. For large number of vectors and bandwidth, however, DSBMM is slightly better.

Table 4.4 Sequential multiplication time using DGBMV and speedup of banded matrix – multiple vectors multiplication using DSBMM and DSBMM2 on MERCAN

Number of Rows	Lower Band-width	Implementation Type	Number of Vectors					
			1		10		100	
			DGBMV Mult. Time(sec)	Speedup	DGBMV Mult. Time(sec)	Speedup	DGBMV Mult. Time(sec)	Speedup
100,000	10	DSBMM	0.01	0.20	0.06	0.72	0.66	1.12
100,000	10	DSBMM2		0.13		0.86		2.04
100,000	50	DSBMM	0.02	0.43	0.18	1.81	1.78	2.90
100,000	50	DSBMM2		0.19		1.51		2.80
100,000	200	DSBMM	0.06	0.70	0.61	2.64	6.07	4.08
100,000	200	DSBMM2		0.28		2.33		4.08
1,500,000	10	DSBMM	0.10	0.20	0.97	0.70	9.76	0.91
1,500,000	10	DSBMM2		0.13		0.85		2.01
1,500,000	50	DSBMM	0.26	0.43	2.62	1.78	26.22	2.74
1,500,000	50	DSBMM2		0.19		1.49		2.75
1,500,000	200	DSBMM	0.91	0.70	9.09	2.62	90.90	4.05
1,500,000	200	DSBMM2		0.28		2.31		4.05

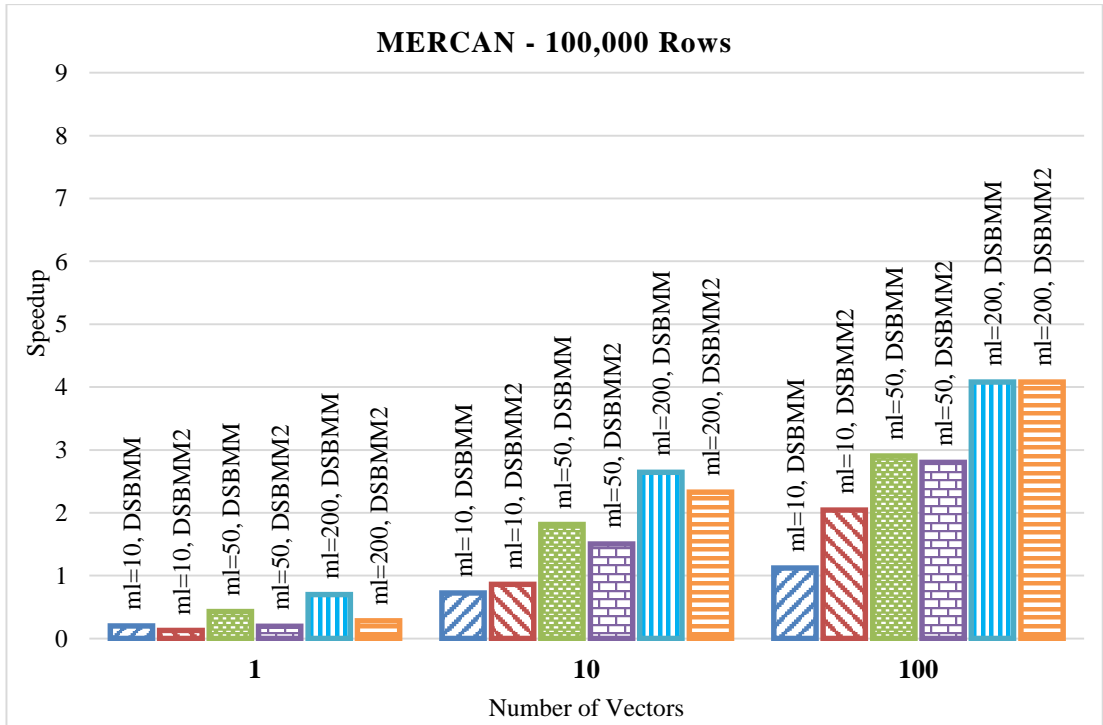


Figure 4.3 Speedup chart of banded matrix – multiple vectors multiplication for matrix having 100,000 rows on MERCAN

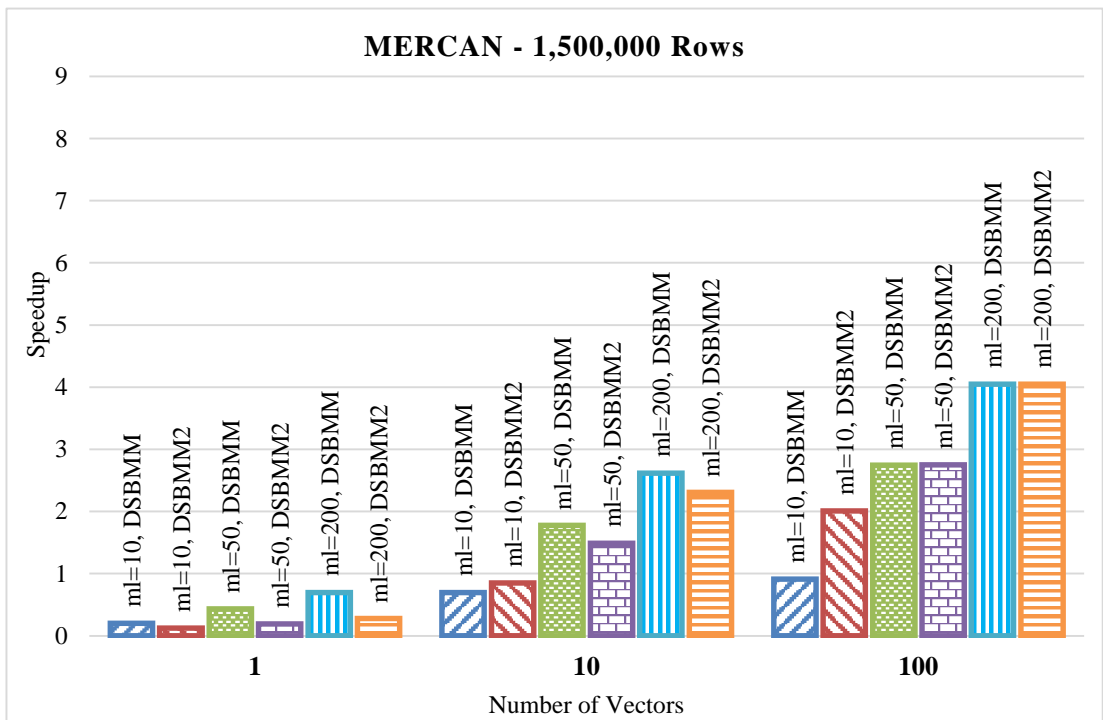


Figure 4.4 Speedup chart of banded matrix – multiple vectors multiplication for matrix having 1,500,000 rows on MERCAN

The four figures above indicate that DGBMV is better than DSBMM and DSBMM2 if banded matrix is multiplied with a single vector. Nevertheless, DSBMM and DSBMM2 provide a better speedup as the bandwidth and/or number of vectors increase in both platforms. DSBMM2 is roughly 1.8 times faster than DSBMM if the bandwidth is 10 and the number of vectors is 100 because of using square dense blocks more effectively and lowering the number of triangular blocks, seen in Figure 3.2.

As the bandwidth increases, DSBMM is faster for smaller number of vectors but they have almost equal performance if the number of vectors is 100. A possible reason for this behavior is that DSBMM is implemented in FORTRAN language whether DSBMM2 is implemented in C language.

The maximum speedup observed for DSBMM and DSBMM2 are 8.18 and 7.86 respectively. It is observed when the number of vectors is 100 and lower bandwidth is 200 on NAR platform. The results show that performance of the implementation grows linearly with the increasing number of processes, which means that they are suitable for multiprocessors.

Overall, considering either DSBMM or DSBMM2, it is observed that NAR has better speedup than MERCAN. Most probably, the reason for this behavior is that NAR has Intel processors and MERCAN has AMD processors. Intel MKL library is optimized for Intel platforms.

Matrices having 100,000 and 1,500,000 rows follow the same speedup trend. Other matrices having different number of rows follow the same trend as well.

4.3.2 Sparse Matrix – Multiple Vectors Multiplication

Three different set of experiments for sparse matrix – multiple vectors multiplication are performed. The first one is comparing matrix in its original form with matrix permuted using MC73 and RCM algorithms. After finding the best one, it will be used on the following set of experiments. The second one is comparing METIS and PATOH

partitioning tools. The best one will be used on the following experiment. The last one is comparing the parallel scalability.

No threads are used in the implementations of this section and all tests are done in two different computing platforms.

4.3.2.1 Matrix Collection

The matrices used in experiments in this section are all sparse and square; matrix entries are double precision numbers. They all are obtained from the University of Florida Sparse Matrix Collection [42]. Their basis properties are given in Table 4.5. All matrices are treated as general even if they are symmetric.

Table 4.5 Matrices used for performance testing

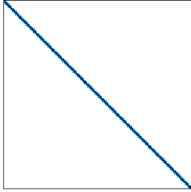
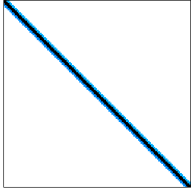
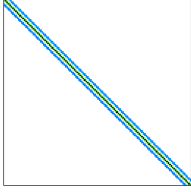
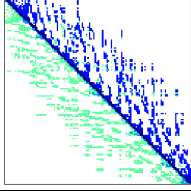
<i>Matrix Name</i>	<i>Spy Plot</i>	<i>Application</i>	<i>Sym- metric</i>	<i>Number of Rows and Columns</i>	<i>Number of Nonzeros (NNZ)</i>	<i>NNZ/ Row</i>
af_shell10		structural problem	Yes	1,508,065	52,259,885	34.65
atmosmodd		computational fluid dynamics	No	1,270,432	8,814,880	6.94
atmosmodl		computational fluid dynamics	No	1,489,752	10,319,760	6.93
cage14		directed weighted graph	No	1,505,785	27,130,349	18.02

Table 4.5 Matrices used for performance testing (continued)

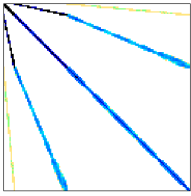
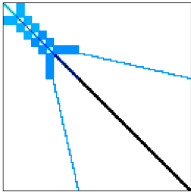
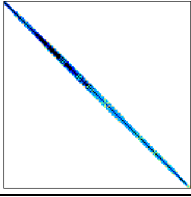
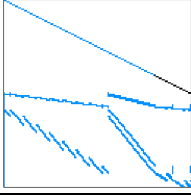
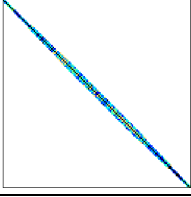
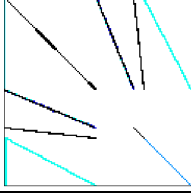
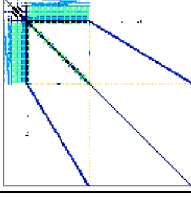
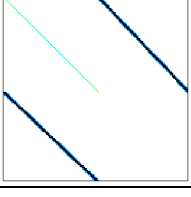
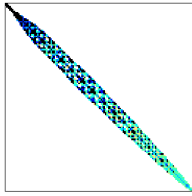
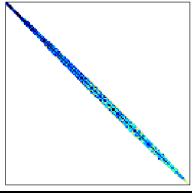
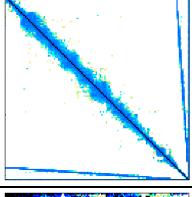
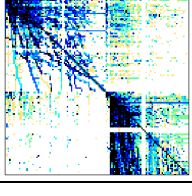
dielFilterV3real		electromagnetic	Yes	1,102,824	89,306,020	80.98
G3_circuit		circuit simulation	Yes	1,585,478	7,660,826	4.83
Geo_1438		structural problem	Yes	1,437,960	60,236,322	41.89
Hamrle3		circuit simulation	No	1,447,360	5,514,242	3.81
Hook_1498		structural problem	Yes	1,498,023	59,374,451	39.64
kkt_power		circuit optimization	Yes	2,063,494	12,771,361	6.19
memchip		circuit simulation	No	2,707,524	13,343,948	4.93
nlpkkt80		optimization	Yes	1,062,400	28,192,672	26.54

Table 4.5 Matrices used for performance testing (continued)

Serena		structural problem	Yes	1,391,349	64,131,971	46.09
StocF-1465		computational fluid dynamics	Yes	1,465,137	21,005,389	14.34
thermal2		thermal problem	Yes	1,228,045	8,580,313	6.99
webbase-1M		weighted directed graph	No	1,000,005	3,105,536	3.11

4.3.2.2 Effect of Permutation on Sequential Sparse Matrix – Multiple Vectors Multiplication

In this section, three different forms of matrix are compared. They are,

- Matrix in its original form
- Matrix permuted with permutation vector generated using MC73
- Matrix permuted with permutation vector generated using RCM

The matrix in all forms is multiplied with 100 vectors using DCSRMM, a sparse BLAS Level 3 routine. The multiplication time of matrix in the first form is used as a reference in this section. These operations are carried out sequentially, i.e., using a single core of a processor. The results are evaluated in terms of speedup varying matrix, reordering technique (stated above) and computing platform. Multiplication time of the original matrix, times for obtaining the permutation using MC73 and RCM for both platforms are given in Table 4.6. Values in the table are given in second.

Table 4.6 Sequential multiplication time of matrix in its original form and the times for obtaining the permutation using MC73 and RCM

<i>Matrix</i>	<i>NAR</i>			<i>MERCAN</i>		
	<i>Orig. Form Mult. Time (sec)</i>	<i>MC73 Time (sec)</i>	<i>RCM Time (sec)</i>	<i>Orig. Form Mult. Time (sec)</i>	<i>MC73 Time (sec)</i>	<i>RCM Time (sec)</i>
af_shell10	10.87	5.33	0.69	30.16	6.69	0.81
atmosmodd	2.60	2.51	0.45	15.65	2.80	0.47
atmosmodl	3.05	3.05	0.51	19.41	3.35	0.55
cage14	7.89	8.73	1.48	36.36	9.75	1.42
dielFilterV3real	19.26	11.02	1.27	54.83	13.08	1.63
G3_circuit	2.44	3.29	0.35	16.78	3.66	0.50
Geo_1438	13.64	7.82	0.78	39.62	9.42	0.94
Hamrle3	2.29	22.14	1.03	13.47	27.08	0.88
Hook_1498	12.74	7.97	0.80	39.07	9.55	0.96
kkt_power	5.82	31.73	3.41	31.80	32.73	2.62
memchip	5.58	9.36	1.30	29.65	10.34	1.35
nlpkkt80	5.54	6.90	0.62	22.50	8.51	0.73
Serena	13.48	8.18	0.76	40.42	9.76	0.93
StocF-1465	5.32	5.53	0.93	26.09	6.35	1.02
thermal2	2.93	3.78	0.55	15.45	4.16	0.58
webbase-1M	1.43	35.72	0.28	8.52	38.47	0.29

RCM requires at least one-sixth of MC73. Even though times to obtain the permutations on both platforms are nearly same, multiplication times are quite different. The reason for this difference is that NAR has Intel processors and MERCAN has AMD processors. Intel MKL library is optimized for Intel platforms.

Sequential performance comparison of sparse matrix – multiple vectors multiplication with matrix in its original form, matrix permuted using MC73 and RCM on NAR platform is given in Figure 4.5 and Figure 4.6. Figure 4.7 and Figure 4.8 gives the same comparison on MERCAN platform.

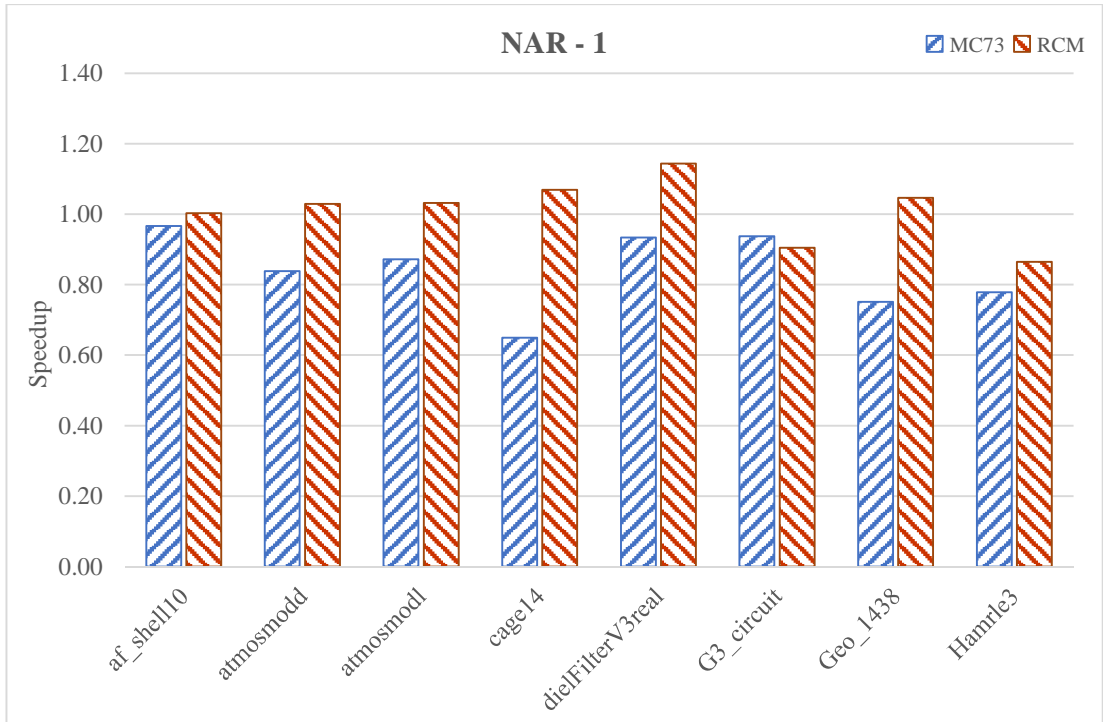


Figure 4.5 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on NAR (part 1 of 2)

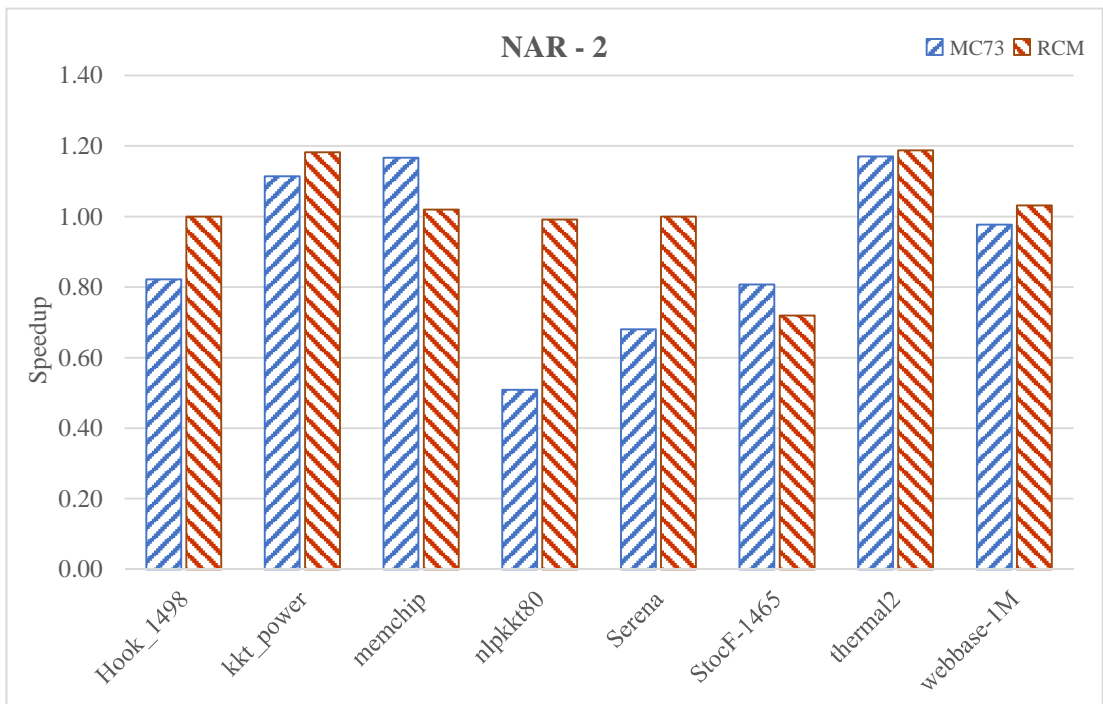


Figure 4.6 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on NAR (part 2 of 2)

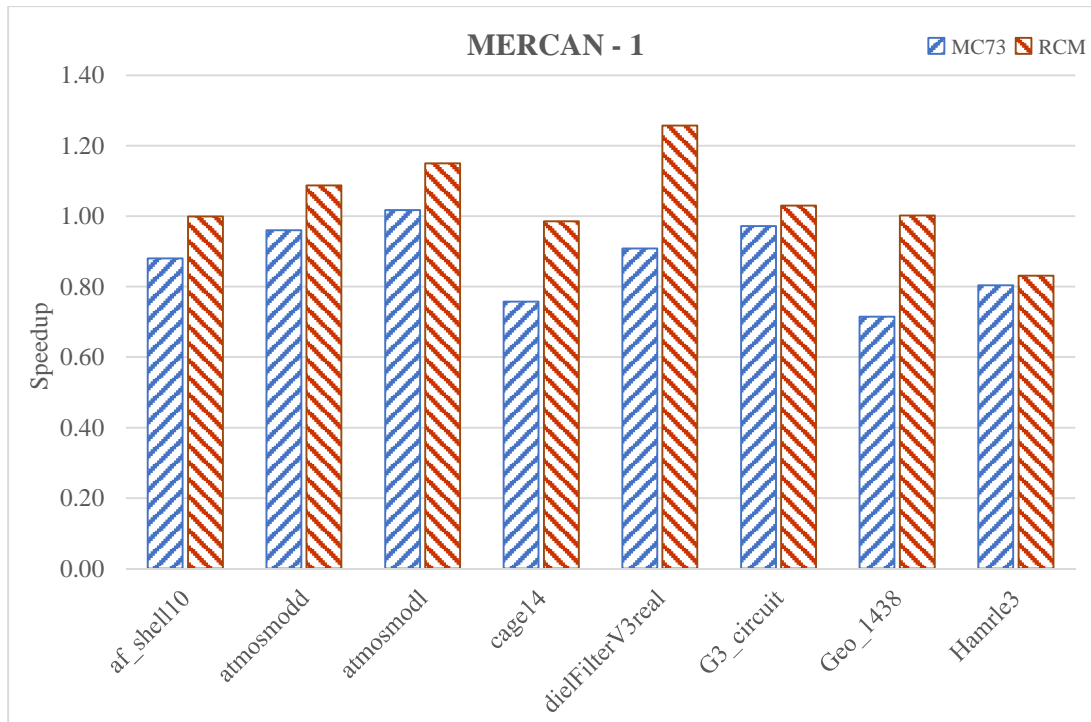


Figure 4.7 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on MERCAN (part 1 of 2)

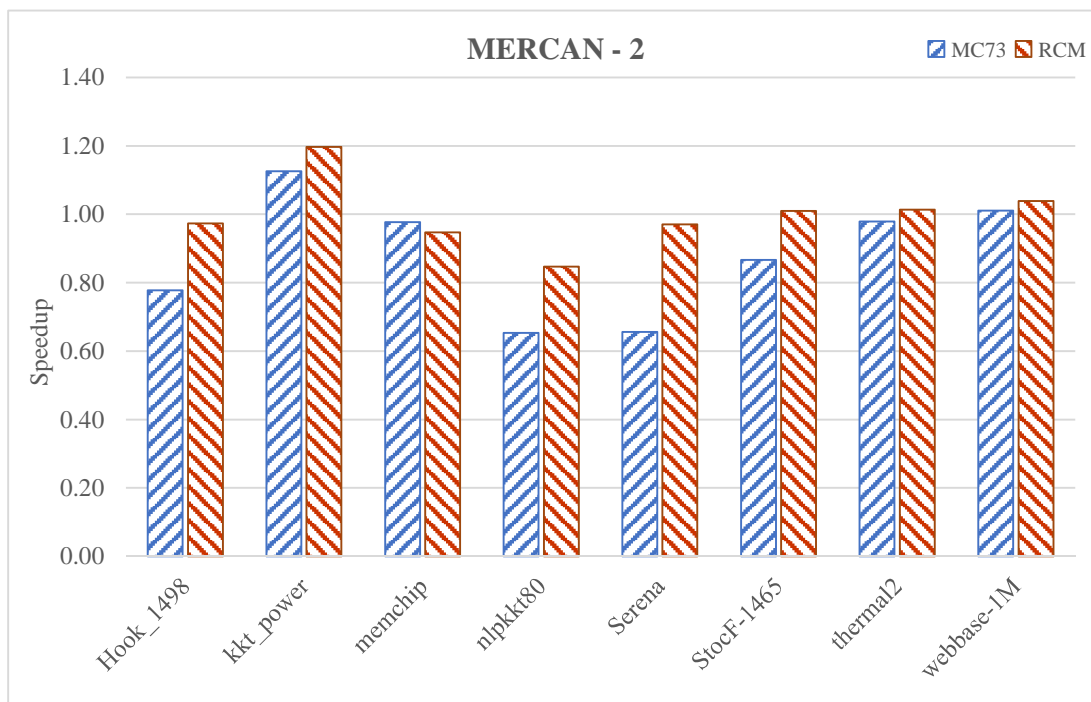


Figure 4.8 Sequential performance comparison chart of SpMM with matrix in its original form, matrix permuted using MC73 and RCM on MERCAN (part 2 of 2)

In both platforms, matrix permuted using MC73 shows no speedup except kkt_power, memchip and thermal2 matrices, which arise in circuit and thermal application areas. The reason is that MC73 dissolves any existing dense block substructure, particularly on cage14, Geo_1438, nlpkkt80 and Serena matrices, most of them represent a structural problem. MC73 shows a better speedup than RCM only on memchip matrix, which represents a circuit simulation problem.

In general, matrices permuted using RCM show a better speedup than the matrices permuted using MC73 and in its original form. Particularly on atmosmodl, dielFilterV3real, kkt_power and thermal2 matrices, RCM yields a good speedup. These matrices arise in computational fluid dynamics, electromagnetic, circuit optimization and thermal application areas respectively. It is roughly 1.16 times faster than reference multiplication for these matrices.

4.3.2.3 Parallel Scalability

In the previous section, the forms of matrix are studied and the matrix permuted using RCM shows a better speedup. Hence, in this section all operations are done with matrices permuted using RCM.

4.3.2.3.1 METIS vs. PATOH

In this section, a graph partitioning tool METIS and a hypergraph partitioning tool PATOH are compared. All the operations in this section are operated within 16 processes. Thus, matrix is partitioned into 16 parts by the partitioning tool. Then each process multiplies the permuted matrix with 100 vectors using DCSRMM, a sparse BLAS Level 3 routine.

The sequential multiplication time of matrix, permuted using RCM is used as a reference in this section. The results are evaluated in terms of speedup varying matrix, partitioning tool and computing platform. Sequential multiplication time, partitioning time of METIS and PATOH for both platforms are given in Table 4.7. Values in the table are given in second.

Table 4.7 Sequential multiplication time and partitioning time of METIS and PATOH

<i>Matrix</i>	<i>NAR</i>			<i>MERCAN</i>		
	<i>Seq. Mult. Time (RCM) (sec)</i>	<i>METIS Partitioning Time (sec)</i>	<i>PATOH Partitioning Time (sec)</i>	<i>Seq. Mult. Time (RCM) (sec)</i>	<i>METIS Partitioning Time (sec)</i>	<i>PATOH Partitioning Time (sec)</i>
af_shell10	10.85	4.23	90.80	30.19	2.59	65.83
atmosmodd	2.53	4.44	35.24	14.39	2.28	20.55
atmosmodl	2.96	4.03	43.10	16.88	2.36	24.64
cage14	7.39	51.06	236.83	36.91	24.09	127.80
dielFilterV3real	16.85	10.20	400.92	43.63	5.88	279.45
G3_circuit	2.70	2.85	24.99	16.29	1.68	14.23
Geo_1438	13.04	7.41	156.51	39.54	4.55	109.10
Hamrle3	2.65	132.74	18.70	16.22	63.08	10.90
Hook_1498	12.74	7.19	155.44	40.13	4.34	108.40
kkt_power	4.92	10.00	98.20	26.56	5.37	54.00
memchip	5.47	8.67	39.81	31.31	3.99	20.35
nlpkkt80	5.58	8.00	93.87	26.57	4.42	60.96
Serena	13.47	8.74	168.40	41.64	5.15	117.16
StocF-1465	5.29	5.83	73.01	25.83	3.35	43.89
thermal2	2.46	2.62	26.03	15.25	1.48	14.54
webbase-1M	1.38	3.70	18.45	8.20	2.33	12.23

Partitioning time of METIS is approximately one-seventh of PATOH partitioning time. Partitioning time of both tools on MERCAN is approximately two times better than on NAR due to higher memory capacity and computational power on MERCAN. On the contrary, sequential multiplication time of matrix on NAR is approximately one-fourth of multiplication time on MERCAN. The reason for this difference is that NAR has Intel processors and MERCAN has AMD processors. Intel MKL library is optimized for Intel platforms.

The speedup is measured using RCM and 16 processes compared to the sequential running time. Speedup plots using METIS and PATOH with RCM reordering for the diagonal block on NAR platform are given in Figure 4.9 and Figure 4.10. Figure 4.11 and Figure 4.12 gives the same comparison on MERCAN platform.

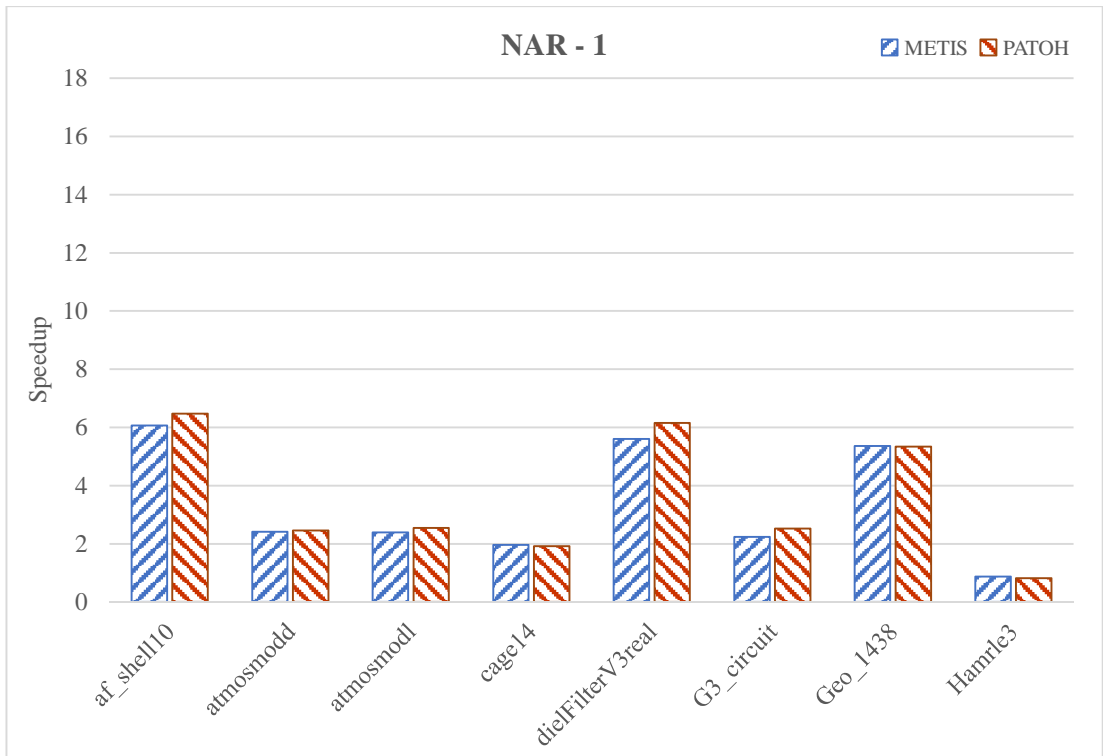


Figure 4.9 Speedup comparison chart of partitioning tools on NAR (part 1 of 2)

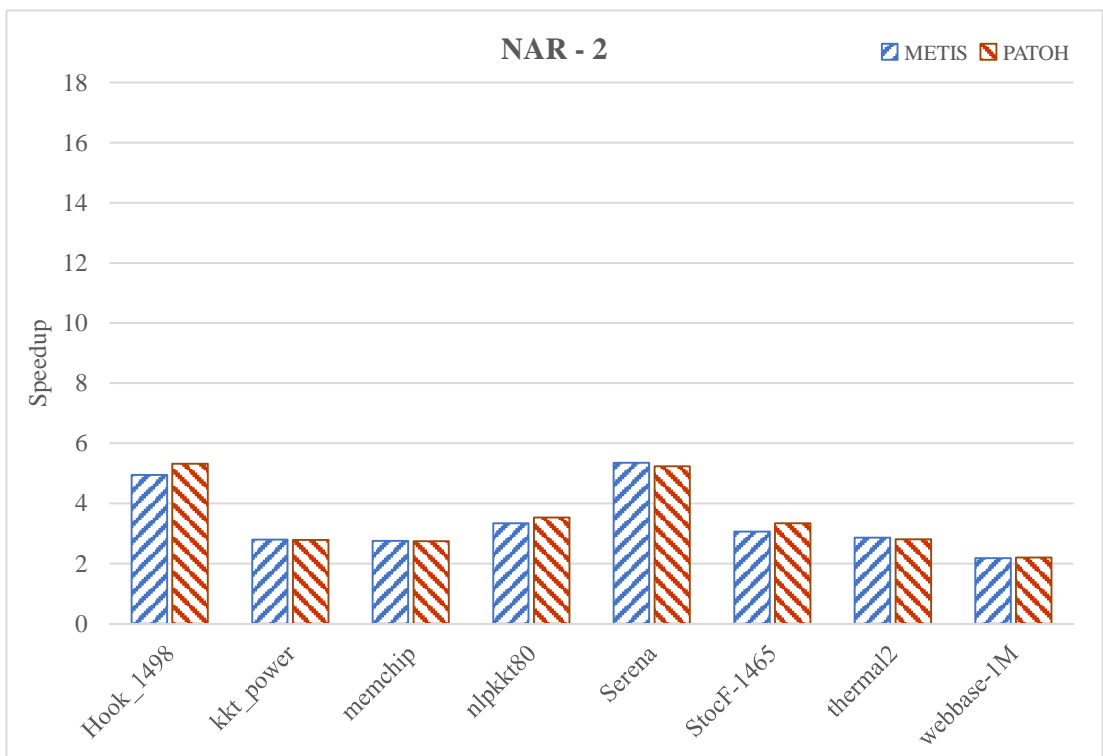


Figure 4.10 Speedup comparison chart of partitioning tools on NAR (part 2 of 2)

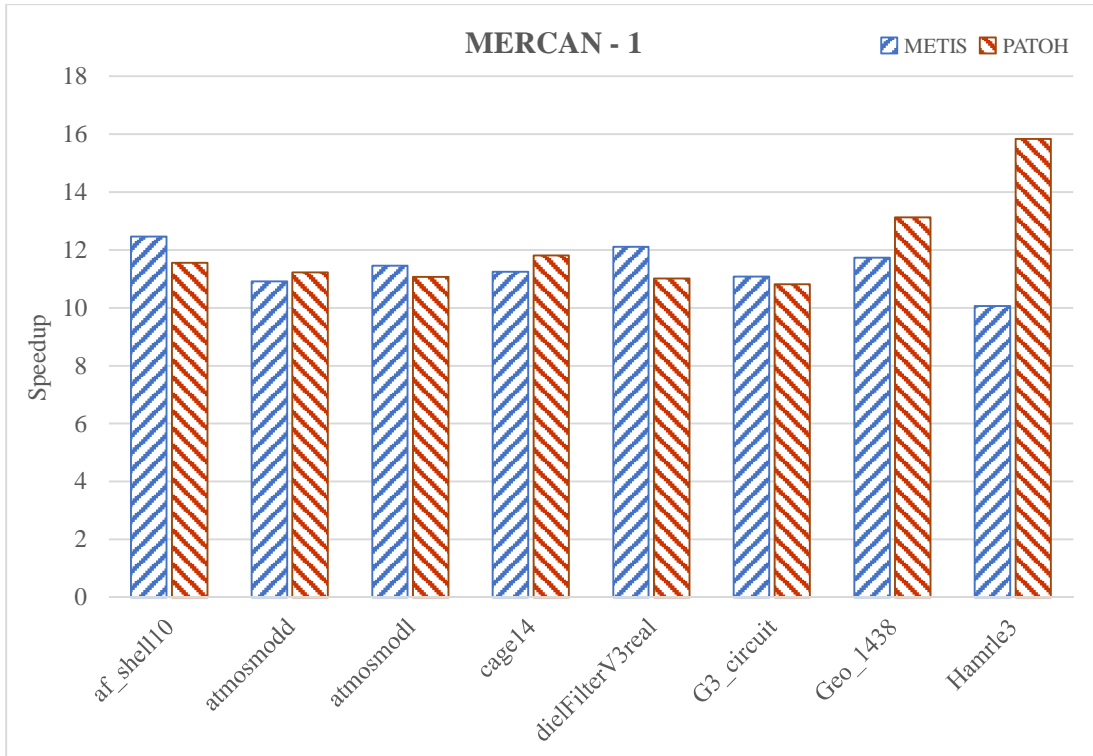


Figure 4.11 Speedup comparison chart of partitioning tools on MERCAN (part 1 of 2)

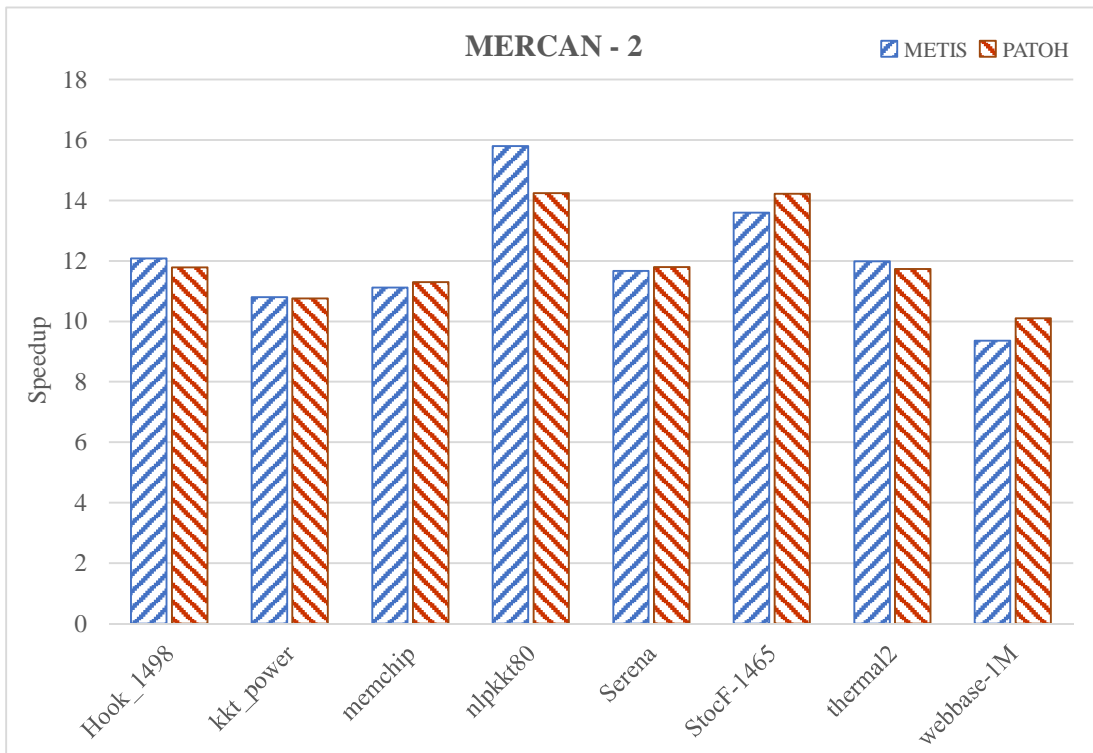


Figure 4.12 Speedup comparison chart of partitioning tools on MERCAN (part 2 of 2)

On both platforms, matrices partitioned using METIS and PATOH show nearly the same speedup. The only exception is Hamrle3 matrix, which is one of the most sparse matrix in the collection with 3.81 nonzeros per row. On NAR platform, due to smaller cache size, Hamrle3 matrix does not show speedup in both partitioning methods. On the contrary, the largest speedup difference between partitioning tools is observed with the same matrix on MERCAN platform.

As a result, PATOH shows slightly a better speedup. The reason is that hypergraph based partitioning methods (e.g. PATOH) defines the communication problem better than graph based ones (e.g. METIS).

4.3.2.3.2 Parallel Scalability

In the last two sections, partitioning tools and permutation of matrix are studied. The matrix partitioned using PATOH and permuted using RCM shows a better speedup. Hence, in this section all operations are done with matrices partitioned using PATOH and then the diagonal blocks are permuted using RCM. The matrix is multiplied with a block of 100 vectors using DCSRMM, a sparse BLAS Level 3 routine.

In this section, parallel scalability for each matrix is given. The results are evaluated in terms of speedup varying matrix, number of processes and computing platform. The number of processes used for each test starts from one (also called sequentially) and continues with the power of two up to 16.

The sequential multiplication time of the matrix, permuted using RCM is used as a reference in this section. Sequential multiplication time is given in Table 4.7 above. Values in the table are given in second.

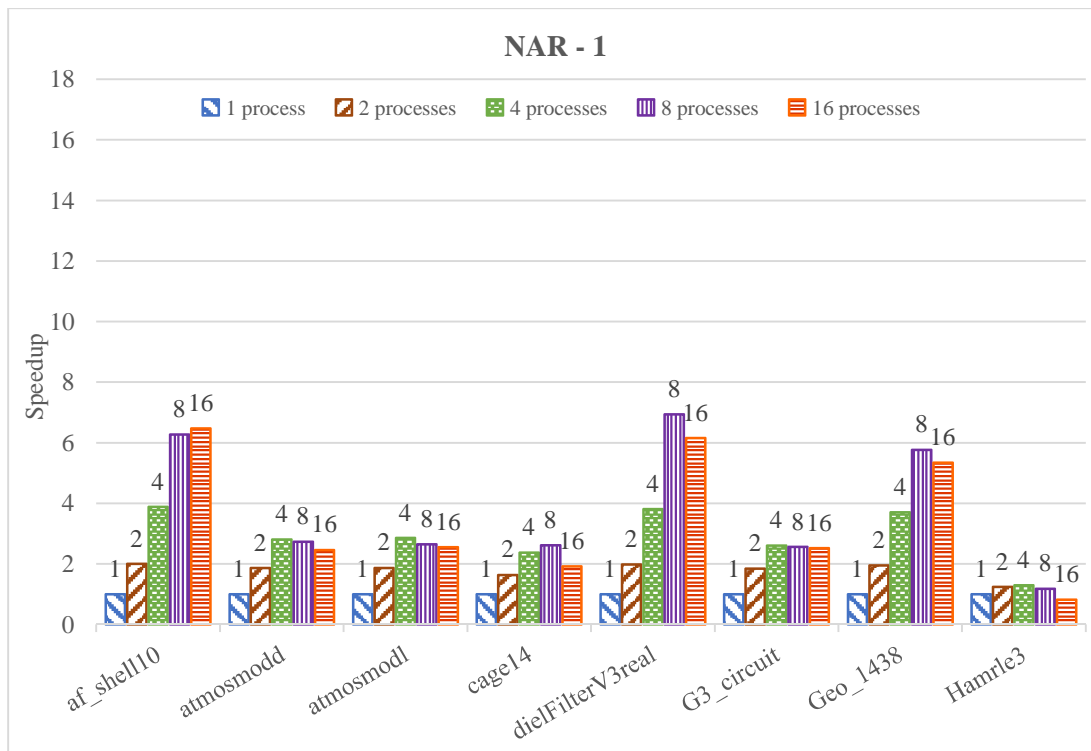


Figure 4.13 Speedup comparison chart of matrices on NAR (part 1 of 2)

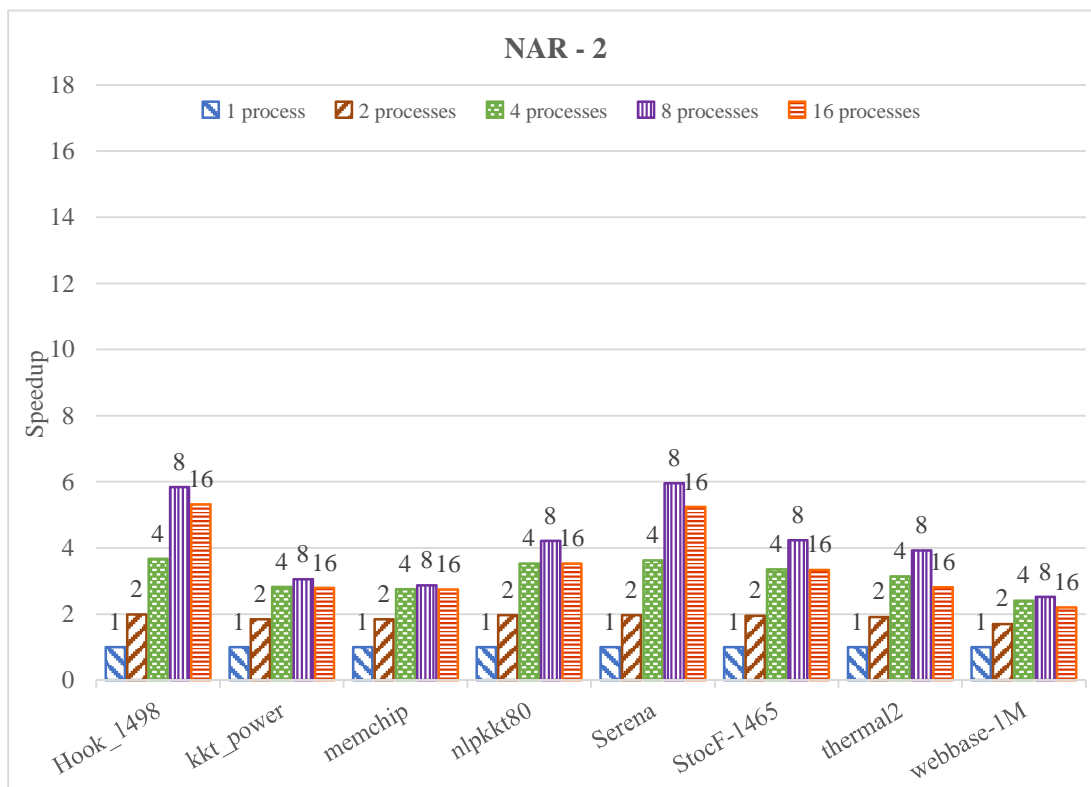


Figure 4.14 Speedup comparison chart of matrices on NAR (part 2 of 2)

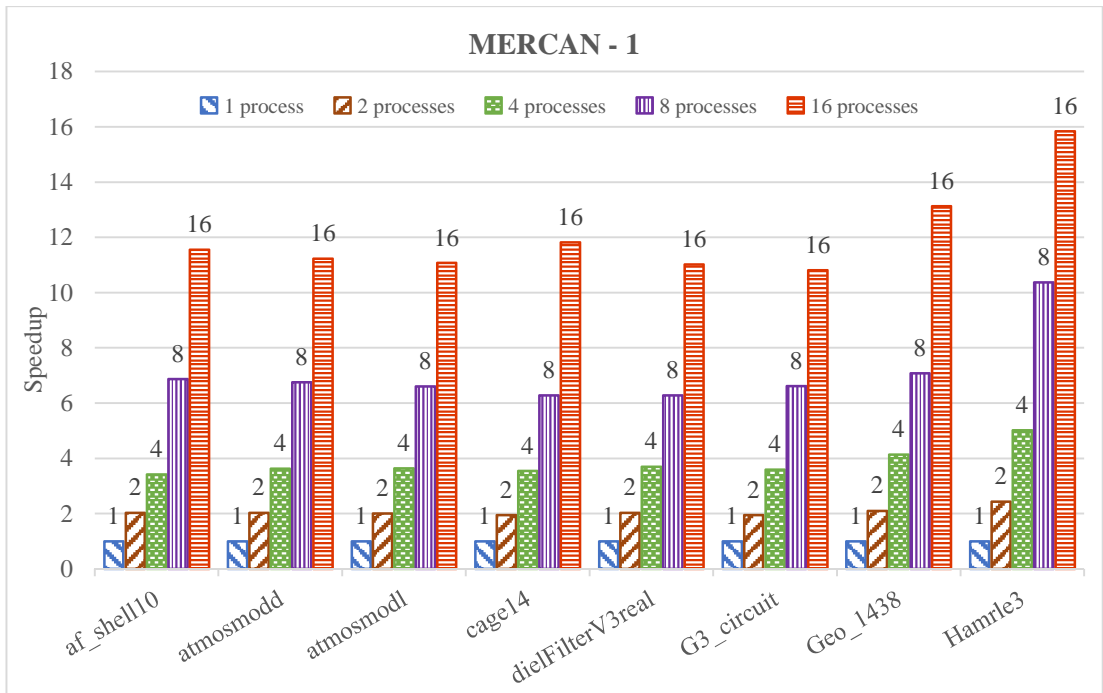


Figure 4.15 Speedup comparison chart of matrices on MERCAN (part 1 of 2)

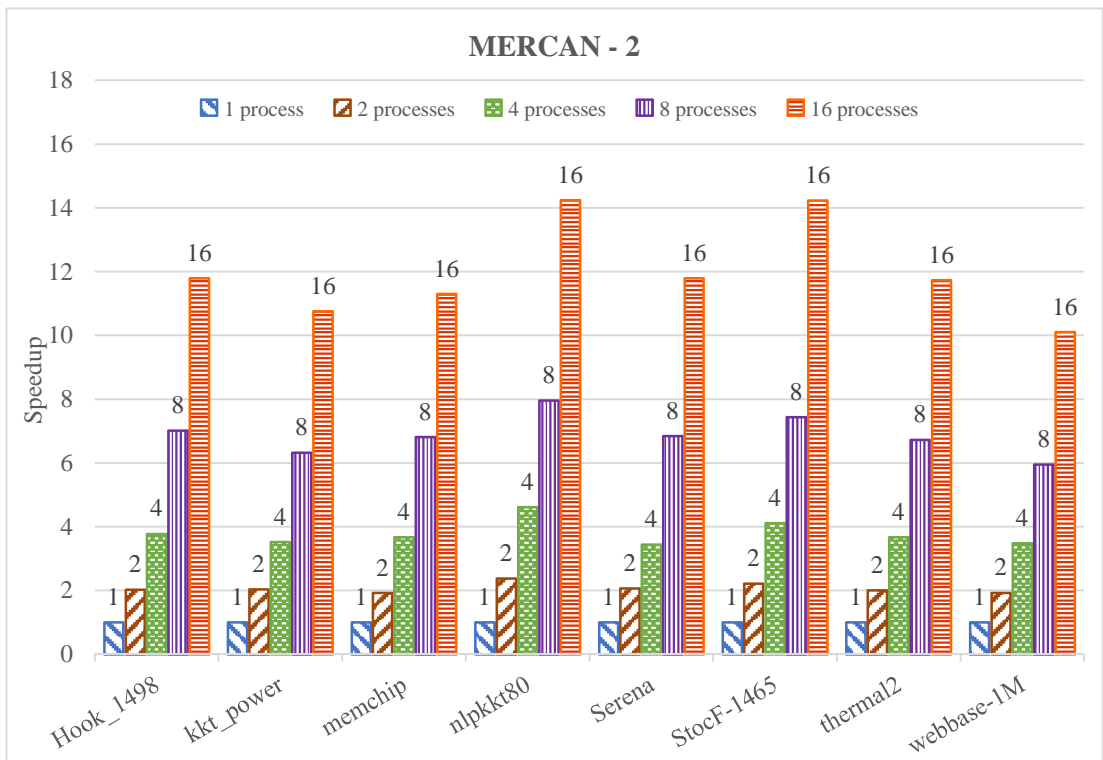


Figure 4.16 Speedup comparison chart of matrices on MERCAN (part 2 of 2)

On NAR platform, speedup continues to increase up to eight processes, and then there is a drop. The reason for the drop is that a node on NAR has eight cores and each process is mapped onto a single core. If the number of processes exceeds eight, more than one node are used and communication between nodes is required. Even it has an Infiniband switch between nodes, it is slower than on-chip communication.

Unlike NAR, speedup continues to increase on MERCAN, which has 24 cores in each node. Moreover, MERCAN shows approximately two times better speedup than NAR. The reason for this dissimilarity is that MERCAN has higher cache capacity. Namely, more vector entries can remain in the cache, reducing misses when the algorithm tries to access them again.

On MERCAN platform, Geo_1438, Hamrle3, nlpkkt80, StocF-1465 matrices, show a better speedup acceleration according to increasing number of processes. These matrices arise in structural problem, circuit simulation, optimization and computational fluid dynamics application areas respectively.

To provide an effective load balancing and a good parallel scalability, partitioning tools and permutation algorithms are used. The results show that performance of the implementation grows linearly with the increasing number of processes, which means that they are suitable for multiprocessors.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis presented and compared different techniques for improving multiplication of sparse and banded matrices with multiple vectors. For banded matrices, an improved method called DSBMM2 was presented that has advantage especially for banded matrices having small bandwidth and multiplied with large number of vectors. DSBMM2 is roughly 1.8 times faster than DSBMM if the bandwidth is 10 and the number of vectors is 100 on both platforms.

Whether DSBMM or DSBMM2 are used, machine specific tuning is worthy as the ideal size of square and triangular blocks, namely bandwidth, is dependent on cache size of platform.

For sparse matrices, it was shown that partitioning tools provide an effective load balancing and a good parallel scalability. To improve memory efficiency, each row of vectors were stored contiguously. Hence, reduced number of cache misses were observed while multiplying with multiple vectors, particularly on MERCAN platform, which has larger cache and memory size. Permuting the matrix with RCM was recommended rather than with MC73 in order to group nonzeros contiguously. Numerical experiments showed that matrices permuted using RCM yield approximately 10% speedup.

With the increasing multi-core environments, parallel scalability of primitive operations (such as SpMV, SpMM banded matrix – vector multiplication and banded matrix – multiple vectors multiplication) are becoming more important. For future work, block storage formats could be used to improve cache-hit ratio, platform dependent tuning (such as various block size, prefetching matrix and vector elements,

etc.) could be done to achieve better speedup. TRACEMIN-Fiedler method [43] could also be compared with RCM and MC73.

REFERENCES

- [1] C. Bischof, X. Sun, A. Tsao, and T. Turnbull, “A Study of the Invariant Subspace Decomposition Algorithm for Banded Symmetric Matrices,” *Proc. Fifth SIAM Conf. Appl. Linear Algebr.*, pp. 321–325, 1994.
- [2] L. M. Adams and R. J. Leveque, “Analysis of the SOR iteration for the 9-point Laplacian,” *SIAM J. Numer. Anal.*, vol. 25, pp. 1156–1180, 1998.
- [3] P. Amodio and F. Mazzia, “A parallel Gauss-Seidel method for block tridiagonal linear systems,” *SIAM J. Sci. Comput.*, vol. 16, no. 6, pp. 1451–1461, 1995.
- [4] J. M. Ortega, *Introduction to parallel and vector solution of linear systems*. New York: Plenum Press, 1988.
- [5] A. H. Sameh and V. Sarin, “Hybrid Parallel Linear System Solvers,” *Int. J. Comput. Fluid Dyn.*, vol. 12, pp. 213–223, 1998.
- [6] E. Polizzi and N. Ben Abdallah, “Subband decomposition approach for the simulation of quantum electron transport in nanostructures,” *J. Comput. Phys.*, vol. 202, no. 1, pp. 150–180, 2004.
- [7] A. Tsao and T. Turnbull, *A Comparison of Algorithms for Banded Matrix Multiplication*. Supercomputing Research Center, 1993, pp. 1–9.
- [8] A. Remon, E. S. Quintana-Orti, and G. Quintana-Orti, “The Implementation of BLAS for Band Matrices,” *Parallel Process. Appl. Math.*, vol. 4967, pp. 668–677, 2008.
- [9] “Intel Corporation, Intel Math Kernel Library (Intel MKL), 2014 (Version 11.1).” [Online]. Available: <https://software.intel.com/en-us/intel-mkl>. [Accessed: 01-Jul-2014].
- [10] E. Polizzi and A. Sameh, “SPIKE: A parallel environment for solving banded linear systems,” *Comput. Fluids*, vol. 36, no. 1, pp. 113–120, Jan. 2007.
- [11] R. Bisseling and W. Meesen, “Communication balancing in parallel sparse matrix-vector multiplication,” *Electron. Trans. Numer. ...*, vol. 21, pp. 47–65, 2005.

- [12] R. Geus and S. Röllin, “Towards a fast parallel sparse matrix-vector multiplication.,” *E.H. D’Hollander, J.R. Joubert, F.J. Peters, H. Sips (Editors), Proc. Int. Conf. Parallel Comput. (ParCo), Imp. Coll. Press*, pp. 308–315, 1999.
- [13] S. Toledo, “Improving the memory-system performance of sparse-matrix vector multiplication,” *IBM J. Res. Dev.*, vol. 41, no. 6, pp. 711–725, Nov. 1997.
- [14] A. Pinar and M. T. Heath, “Improving performance of sparse matrix-vector multiplication,” *Proc. 1999 ACM/IEEE Conf. ...*, 1999.
- [15] A. H. Baker, J. M. Dennis, and E. R. Jessup, “On Improving Linear Solver Performance: A Block Variant of GMRES,” *SIAM J. Sci. Comput.*, vol. 27, p. 2006, 2006.
- [16] R.-C. Li and L.-H. Zhang, “Convergence of the Block Lanczos Method for Eigenvalue Clusters Convergence of the Block Lanczos Method,” 2013.
- [17] G. H. Golub and R. Underwood, “The Block Lanczos Method for Computing Eigenvalues,” *J. R. Rice, Ed. Math. Softw. III, Acad. Press. New York*, pp. 361–377, 1977.
- [18] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. 2000, p. 316.
- [19] E.-J. Im, K. Yelick, and R. Vuduc, “Sparsity: Optimization Framework for Sparse Matrix Kernels,” *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 1, pp. 135–158, Feb. 2004.
- [20] M. Manguoglu, A. Sameh, and O. Schenk, “PSPIKE : A Parallel Hybrid Sparse Linear,” *Euro-Par 2009 Parallel Process.*, vol. 5704, pp. 797–808, 2009.
- [21] R. Kannan, “Efficient sparse matrix multiple-vector multiplication using a bitmapped format,” *High Perform. Comput. 2013*, pp. 286–294, 2012.
- [22] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, “Optimization of sparse matrix–vector multiplication on emerging multicore platforms,” *Parallel Comput.*, vol. 35, no. 3, pp. 178–194, Mar. 2009.
- [23] U. V. Catalyurek and C. Aykanat, “Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, pp. 673–693, 1999.
- [24] G. Karypis and V. Kumar, “Multilevel k-way Partitioning Scheme for Irregular Graphs,” *J. Parallel Distrib. Comput.*, vol. 48, pp. 96–129, 1998.

- [25] U. V. Catalyurek and C. Aykanat, “PaToH (Partitioning Tool for Hypergraphs),” *Encycl. Parallel Comput. Springer, Ed. D. Padua*, pp. 1479–1487, 2011.
- [26] E. Cuthill and J. Mckee, “Reducing the bandwidth of sparse symmetric matrices,” *Proc. 24th Natl. Conf. Assoc. Comput. Mach. New York*, pp. 157–172, 1969.
- [27] Y. Hu and J. Scott, “HSL MC73: a fast multilevel Fiedler and profile reduction code,” Rutherford Appleton Laboratory, Oxfordshire, UK, 2003.
- [28] E. Polizzi, “Subroutine DSBMM.” [Online]. Available: <https://github.com/certik/feast/blob/master/src/lbprim.f90>. [Accessed: 01-Jan-2014].
- [29] R. F. Boisvert, R. Pozo, K. Remington, R. F. Barrett, and J. J. Dongarra, “Matrix Market : A Web Resource for Test Matrix Collections,” *R. Boisvert, Ed. Qual. Numer. Softw. Assess. Enhanc.*, pp. 125–137, 1997.
- [30] N. I. of S. and T. Mathematical and Computational Sciences Division, Information Technology Laboratory, “ANSI C library for Matrix Market I/O.” [Online]. Available: <http://math.nist.gov/MatrixMarket/mmio-c.html>. [Accessed: 01-Jan-2014].
- [31] G. Karypis, “METIS and ParMETIS,” *Encycl. Parallel Comput. Springer, Ed. D. Padua*, pp. 1117–1124, 2011.
- [32] U. V. Catalyurek, “Hypergraph Models for Sparse Matrix Partitioning and Reordering. PhD Thesis,” Bilkent University, 1999.
- [33] U. V. Catalyurek and C. Aykanat, “Decomposing Irregularly Sparse Matrices for Parallel Matrix-Vector Multiplication,” *Lect. Notes Comput. Sci.*, vol. 1117, pp. 75–86, 1996.
- [34] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, “Parallel hypergraph partitioning for scientific computing,” *20th Int. Parallel Distrib. Process. Symp.*, 2006.
- [35] “HSL(2013). A collection of Fortran codes for large scale scientific computation.” [Online]. Available: <http://www.hsl.rl.ac.uk>. [Accessed: 01-Jan-2014].
- [36] C. Zavoianu, “Tutorial: Bandwidth reduction - The CutHill-McKee Algorithm.” [Online]. Available: <http://ciprian-zavoianu.blogspot.com.tr/2009/01/project-bandwidth-reduction.html>. [Accessed: 01-Jul-2014].

- [37] “Department of Computer Engineering, Middle East Technical University, High Performance Computing Facility.” [Online]. Available: <http://www.ceng.metu.edu.tr/hpc/index>. [Accessed: 01-Jul-2014].
- [38] “Turkish Academic Network and Information Center, Turkish Science e-Infrastructure, TRUBA.” [Online]. Available: <http://www.truba.gov.tr/eng/>. [Accessed: 01-Jul-2014].
- [39] J. Burkardt, “Reverse Cuthill McKee Ordering.” [Online]. Available: http://people.sc.fsu.edu/~jburkardt/cpp_src/rcm/rcm.html. [Accessed: 01-Jan-2014].
- [40] Y. Saad, “SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations,” Minneapolis, 1994.
- [41] T. A. Davis, *Direct Methods for Sparse Linear Systems*. Philadelphia: Part of the SIAM Book Series on the Fundamentals of Algorithms, 2006.
- [42] 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, 2011.
- [43] M. Manguoglu, E. Cox, F. Saied, and A. Sameh, “TRACEMIN-Fiedler: A Parallel Algorithm for Computing the Fiedler Vector,” *Lect. Notes Comput. Sci. (proceedings High Perform. Comput. Comput. Sci. – VECPAR10)*, vol. 6449, no. 1, pp. 449–455, 2011.

APPENDIX A

RESULTS OF BANDED MATRIX – MULTIPLE VECTORS MULTIPLICATION

Banded matrix – multiple vectors multiplication operations are done varying;

- Number of rows of the matrix
 - 100,000
 - 400,000
 - 1,500,000
 - 3,000,000
- Multiplication method
 - DGBMV
 - DSBMM
 - DSBMM2
- Lower/upper bandwidth size
 - 5
 - 10
 - 20
 - 50
 - 100
 - 200
- Number of vectors
 - 1
 - 10
 - 100
- Computing platform
 - NAR

- MERCAN

For NAR platform, the times are given in Table A.1 and Table A.2. For MERCAN platform, they are given in Table A.3 and Table A.4. Values in the tables are given in second.

Table A.1 Multiplication time of banded matrices, having 5, 10 and 20 lower bandwidth, with multiple vectors on NAR

Number of Rows	Implementation Type	<i>ml = 5</i>			<i>ml = 10</i>			<i>ml = 20</i>		
		<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>
100,000	DGBMV	0.00	0.03	0.34	0.01	0.07	0.71	0.01	0.12	1.18
100,000	DSBMM	0.03	0.09	0.54	0.03	0.08	0.46	0.03	0.06	0.34
100,000	DSBMM2	0.02	0.04	0.24	0.03	0.06	0.29	0.04	0.08	0.35
400,000	DGBMV	0.01	0.14	1.44	0.03	0.29	2.86	0.05	0.47	4.74
400,000	DSBMM	0.12	0.36	2.43	0.11	0.31	1.98	0.13	0.25	1.44
400,000	DSBMM2	0.07	0.18	0.96	0.12	0.25	1.18	0.17	0.31	1.39
1,500,000	DGBMV	0.05	0.54	5.44	0.11	1.08	10.77	0.18	1.78	17.78
1,500,000	DSBMM	0.47	1.42	8.34	0.43	1.19	7.08	0.49	0.95	5.20
1,500,000	DSBMM2	0.27	0.67	3.61	0.45	0.94	4.41	0.63	1.18	5.21
3,000,000	DGBMV	0.11	1.09	10.88	0.22	2.15	21.53	0.35	3.56	35.55
3,000,000	DSBMM	0.94	2.81	16.47	0.86	2.36	14.15	0.98	1.90	10.46
3,000,000	DSBMM2	0.55	1.33	7.21	0.89	1.87	8.82	1.26	2.35	10.41

Table A.2 Multiplication time of banded matrices, having 50, 100 and 200 lower bandwidth, with multiple vectors on NAR

Number of Rows	Implementation Type	<i>ml = 50</i>			<i>ml = 100</i>			<i>ml = 200</i>		
		<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>
100,000	DGBMV	0.03	0.27	2.71	0.05	0.54	5.42	0.09	0.92	9.23
100,000	DSBMM	0.05	0.10	0.52	0.08	0.14	0.69	0.13	0.24	1.13
100,000	DSBMM2	0.08	0.13	0.49	0.13	0.20	0.72	0.19	0.35	1.18
400,000	DGBMV	0.11	1.09	10.85	0.22	2.17	21.67	0.37	3.69	36.95
400,000	DSBMM	0.21	0.40	2.08	0.33	0.58	2.77	0.53	0.95	4.52
400,000	DSBMM2	0.33	0.51	1.96	0.52	0.82	2.90	0.76	1.41	4.70
1,500,000	DGBMV	0.41	4.07	40.71	0.81	8.13	81.27	1.39	13.86	138.59
1,500,000	DSBMM	0.79	1.51	7.74	1.22	2.17	10.38	1.99	3.57	16.95
1,500,000	DSBMM2	1.22	1.93	7.36	1.94	3.07	10.86	2.84	5.28	17.64
3,000,000	DGBMV	0.81	8.14	81.44	1.62	16.25	162.57	2.77	27.70	277.02
3,000,000	DSBMM	1.57	3.02	15.50	2.44	4.34	20.77	3.98	7.14	33.92
3,000,000	DSBMM2	2.45	3.86	14.72	3.89	6.14	21.73	5.67	10.56	35.29

Table A.3 Multiplication time of banded matrices, having 5, 10 and 20 lower bandwidth, with multiple vectors on MERCAN

Number of Rows	Implementation Type	<i>ml = 5</i>			<i>ml = 10</i>			<i>ml = 20</i>		
		<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>
100,000	DGBMV	0.00	0.03	0.35	0.01	0.06	0.66	0.01	0.09	0.94
100,000	DSBMM	0.04	0.12	0.78	0.03	0.09	0.59	0.03	0.07	0.48
100,000	DSBMM2	0.04	0.07	0.26	0.05	0.07	0.32	0.06	0.08	0.41
400,000	DGBMV	0.01	0.14	1.40	0.03	0.26	2.61	0.04	0.37	3.75
400,000	DSBMM	0.17	0.50	3.95	0.13	0.38	3.01	0.11	0.29	2.23
400,000	DSBMM2	0.17	0.28	1.04	0.20	0.30	1.29	0.25	0.33	1.65
1,500,000	DGBMV	0.05	0.52	5.18	0.10	0.97	9.76	0.14	1.40	14.03
1,500,000	DSBMM	0.64	1.81	14.03	0.47	1.40	10.67	0.41	1.07	8.13
1,500,000	DSBMM2	0.65	1.03	3.88	0.77	1.15	4.85	0.92	1.26	6.16
3,000,000	DGBMV	0.10	1.04	10.37	0.19	1.95	19.49	0.28	2.80	28.00
3,000,000	DSBMM	1.27	3.62	27.77	0.94	2.80	21.49	0.81	2.14	16.14
3,000,000	DSBMM2	1.31	2.07	7.79	1.53	2.29	9.68	1.84	2.53	12.38

Table A.4 Multiplication time of banded matrices, having 50, 100 and 200 lower bandwidth, with multiple vectors on MERCAN

Number of Rows	Implementation Type	<i>ml = 50</i>			<i>ml = 100</i>			<i>ml = 200</i>		
		<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>	<i>1 Vector (sec)</i>	<i>10 Vectors (sec)</i>	<i>100 Vectors (sec)</i>
100,000	DGBMV	0.02	0.18	1.78	0.03	0.32	3.19	0.06	0.61	6.07
100,000	DSBMM	0.04	0.10	0.61	0.06	0.14	0.95	0.09	0.23	1.49
100,000	DSBMM2	0.09	0.12	0.64	0.14	0.16	0.93	0.22	0.26	1.49
400,000	DGBMV	0.07	0.70	6.99	0.13	1.27	12.71	0.24	2.43	24.26
400,000	DSBMM	0.16	0.38	2.52	0.22	0.55	3.84	0.35	0.93	5.98
400,000	DSBMM2	0.36	0.46	2.55	0.55	0.65	3.72	0.87	1.05	5.96
1,500,000	DGBMV	0.26	2.62	26.22	0.48	4.77	47.56	0.91	9.09	90.90
1,500,000	DSBMM	0.60	1.48	9.56	0.83	2.09	14.49	1.31	3.48	22.44
1,500,000	DSBMM2	1.35	1.77	9.54	2.05	2.49	13.96	3.28	3.94	22.35
3,000,000	DGBMV	0.51	5.23	52.33	0.95	9.52	95.63	N/A	N/A	N/A
3,000,000	DSBMM	1.19	2.93	19.00	1.65	4.17	29.03	N/A	N/A	N/A
3,000,000	DSBMM2	2.71	3.53	19.26	4.11	4.99	27.92	N/A	N/A	N/A

On MERCAN, the tests with matrix having 3,000,000 rows and 200 lower bandwidth, give memory error. (N/A means not available)

APPENDIX B

RESULTS OF SPARSE MATRIX – MULTIPLE VECTORS MULTIPLICATION

Sparse matrix – multiple vectors multiplication operations are done varying;

- Form of the matrix
 - Matrix in its original form
 - Matrix permuted with permutation vector generated using MC73
 - Matrix permuted with permutation vector generated using RCM
- Partitioning tool
 - METIS
 - PATOH
- Number of processes
 - 1 (Sequential)
 - 2
 - 4
 - 8
 - 16
- Computing platform
 - NAR
 - MERCAN

The results of SpMM with matrix in its original form are given in Table B.1, Table B.2, Table B.3 and Table B.4.

The results of SpMM with matrix permuted with permutation vector generated using MC73 are given in Table B.5, Table B.6, Table B.7 and Table B.8.

The results of SpMM with matrix permuted with permutation vector generated using RCM are given in Table B.9, Table B.10, Table B.11 and Table B.12.

Sequential permutation and multiplication time of each form of the matrix are given in the first appearance of combination of matrix form and platform in the following tables. They are given in Table B.1 and Table B.3 for original form; Table B.5 and Table B.7 for MC73; Table B.9 and Table B.11 for RCM.

Matrix reading time for each platform are given only in tables having the results of matrix in its original form. They are given in Table B.1 and Table B.3 for NAR and MERCAN platforms, respectively.

Values in the tables are given in second.

Table B.1 Matrix reading time; Sequential multiplication time; Partitioning and parallel multiplication time of matrix partitioned using METIS on NAR

<i>Matrix Name</i>	<i>Reading Matrix (sec)</i>	<i>1 Process</i>	<i>2 Processes</i>		<i>4 Processes</i>		<i>8 Processes</i>		<i>16 Processes</i>	
		<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>
af_shell10	26.87	10.87	2.08	5.50	2.04	2.75	2.09	1.69	4.23	1.82
atmosmodd	7.77	2.60	1.66	1.38	1.73	0.97	1.98	1.02	4.64	1.31
atmosmodl	6.46	3.05	1.98	1.62	2.05	1.13	2.30	1.17	4.13	1.40
cake14	22.04	7.89	7.99	4.97	12.92	3.39	18.44	2.97	50.54	3.92
dielFilterV3real	66.02	19.26	3.41	9.85	3.47	5.23	3.95	3.18	9.99	3.38
G3_circuit	3.79	2.44	1.55	1.31	1.59	0.94	1.60	0.95	2.76	1.20
Geo_1438	31.71	13.64	3.06	6.79	3.14	3.57	3.35	2.18	7.39	2.42
Hamrle3	4.39	2.29	14.08	1.75	24.05	1.63	40.58	1.89	132.52	3.50
Hook_1498	32.77	12.74	3.06	6.46	3.19	3.46	3.37	2.10	7.09	2.32
kkt_power	8.36	5.82	3.71	2.70	4.16	1.69	4.51	1.56	10.00	1.90
memchip	13.95	5.58	4.18	2.86	4.23	2.04	4.27	1.98	8.08	2.16
nlpkkt80	12.40	5.54	2.58	2.87	2.90	1.68	3.32	1.33	8.01	1.63
Serena	31.65	13.48	3.19	6.93	3.36	3.69	3.59	2.22	8.64	2.49
StocF-1465	11.74	5.32	2.11	2.78	2.21	1.60	2.55	1.26	5.83	1.48
thermal2	4.97	2.93	1.36	1.50	1.39	0.95	1.41	0.94	2.52	1.01
webbase-1M	2.13	1.43	1.77	0.78	1.85	0.58	1.99	0.57	3.62	0.63

Table B.2 Partitioning and parallel multiplication time of matrix partitioned using PATOH on NAR

<i>Matrix Name</i>	<i>2 Processes</i>		<i>4 Processes</i>		<i>8 Processes</i>		<i>16 Processes</i>	
	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>
af_shell10	12.13	5.19	23.65	2.75	34.73	1.69	91.00	1.80
atmosmodd	6.31	1.39	10.50	0.98	14.45	1.03	35.41	1.20
atmosmodl	7.68	1.62	13.09	1.13	17.67	1.17	43.07	1.42
cage14	38.97	4.84	70.98	3.28	96.95	3.09	237.04	4.40
dielFilterV3real	53.30	9.87	104.66	5.23	154.10	3.15	401.59	3.31
G3_circuit	3.98	1.31	7.30	0.94	10.21	0.95	25.07	1.12
Geo_1438	23.17	6.73	42.18	3.56	60.97	2.16	157.35	2.50
Hamrle3	3.14	2.11	5.60	2.02	7.79	2.20	18.86	3.65
Hook_1498	21.90	6.53	41.50	3.48	60.03	2.08	155.57	2.37
kkt_power	18.01	2.77	32.56	1.75	41.91	1.56	98.20	1.73
memchip	6.14	2.87	11.35	2.04	16.11	1.97	39.85	2.11
nlpkkt80	14.44	2.82	26.17	1.59	37.28	1.28	94.20	2.09
Serena	24.05	6.85	45.23	3.69	65.26	2.22	169.02	2.54
StocF-1465	11.01	2.71	20.26	1.59	28.69	1.25	72.99	1.50
thermal2	4.01	1.48	7.39	0.95	10.40	0.95	25.77	1.03
webbase-1M	2.73	0.81	5.09	0.61	7.24	0.60	19.19	0.75

Table B.3 Matrix reading time; Sequential multiplication time; Partitioning and parallel multiplication time of matrix partitioned using METIS on MERCAN

<i>Matrix Name</i>	<i>Reading Matrix (sec)</i>	<i>1 Process</i>	<i>2 Processes</i>		<i>4 Processes</i>		<i>8 Processes</i>		<i>16 Processes</i>	
		<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>
af_shell10	23.97	30.16	2.59	15.13	2.50	7.81	2.55	4.50	2.59	2.69
atmosmodd	6.97	15.65	1.67	7.55	1.73	4.02	1.99	2.24	2.24	1.28
atmosmodl	5.91	19.41	1.97	9.54	2.04	5.03	2.25	2.73	2.34	1.62
cage14	20.77	36.36	8.47	19.50	12.54	10.37	17.45	5.67	23.90	3.50
dielFilterV3real	58.17	54.83	4.33	26.99	4.41	14.43	4.91	7.64	5.84	4.58
G3_circuit	3.58	16.78	1.58	8.45	1.60	4.69	1.64	2.77	1.67	1.50
Geo_1438	28.71	39.62	3.84	19.60	3.90	10.42	4.14	5.62	4.56	3.33
Hamrle3	3.88	13.47	13.51	7.38	22.11	4.01	37.79	2.39	62.52	1.45
Hook_1498	29.34	39.07	3.75	19.36	3.90	10.66	4.11	5.78	4.34	3.38
kkt_power	7.69	31.80	3.93	14.97	4.47	7.23	4.81	4.62	5.38	2.48
memchip	12.16	29.65	3.81	14.97	3.86	8.61	3.94	4.78	4.01	2.93
nlpkkt80	11.69	22.50	3.08	11.03	3.36	6.18	3.69	3.16	4.39	1.87
Serena	28.72	40.42	4.00	20.02	4.21	11.12	4.45	6.02	5.06	3.51
StocF-1465	10.66	26.09	2.48	12.69	2.61	6.84	2.97	3.71	3.33	2.23
thermal2	4.73	15.45	1.42	7.90	1.43	4.18	1.48	2.35	1.49	1.28
webbase-1M	1.98	8.52	1.80	4.44	1.92	2.31	2.04	1.39	2.32	0.81

Table B.4 Partitioning and parallel multiplication time of matrix partitioned using PATOH on MERCAN

<i>Matrix Name</i>	<i>2 Processes</i>		<i>4 Processes</i>		<i>8 Processes</i>		<i>16 Processes</i>	
	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Multiplication (sec)</i>
af_shell10	17.20	15.07	33.71	8.23	49.87	4.41	65.74	2.66
atmosmodd	6.28	7.56	11.38	4.05	16.09	2.24	20.57	1.28
atmosmodl	7.69	9.62	13.76	5.20	19.21	2.50	24.51	1.55
cage14	39.37	19.88	72.13	10.14	101.09	5.55	128.47	3.35
dielFilterV3real	72.52	26.18	142.89	12.91	212.64	7.80	279.57	4.50
G3_circuit	4.03	8.49	7.64	4.68	11.14	2.55	14.34	1.40
Geo_1438	30.67	19.41	56.96	10.41	83.15	5.70	109.34	3.32
Hamrle3	3.21	5.71	5.85	2.73	8.44	1.60	10.75	0.98
Hook_1498	29.46	19.30	56.39	10.78	82.74	5.32	108.47	3.32
kkt_power	16.95	15.15	31.11	7.93	42.34	4.17	53.58	2.44
memchip	5.63	15.20	10.87	8.40	15.82	4.81	20.43	2.98
nlpkkt80	17.40	10.27	32.46	5.75	47.00	3.39	60.78	1.79
Serena	32.29	19.87	61.17	11.15	89.75	5.51	117.50	3.24
StocF-1465	12.12	12.59	23.18	6.56	33.98	3.63	44.11	2.05
thermal2	4.17	7.87	7.81	4.36	11.34	2.31	14.57	1.35
webbase-1M	3.28	4.44	6.27	2.30	9.11	1.36	12.21	0.79

Table B.5 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using MC73 on NAR

<i>Matrix Name</i>	<i>1 Process</i>		<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	5.33	11.25	2.08	2.71	5.49	2.04	1.59	2.92	2.09	1.37	1.87	4.23	0.48	1.87
atmosmodd	2.51	3.10	1.66	1.40	1.65	1.73	0.72	1.03	1.98	0.50	1.23	3.96	0.15	1.24
atmosmodl	3.05	3.50	1.98	1.68	1.83	2.05	0.87	1.30	2.30	0.69	1.18	4.82	0.18	1.54
cage14	8.73	12.15	7.98	4.94	7.88	12.90	3.98	5.44	18.42	1.42	4.00	50.53	0.41	4.77
dielFilterV3real	11.02	20.63	3.36	5.63	10.82	3.46	3.22	5.51	3.94	3.10	3.39	10.01	2.05	3.27
G3_circuit	3.29	2.61	1.55	1.80	1.38	1.59	0.87	0.88	1.60	0.65	0.85	3.25	0.16	1.04
Geo_1438	7.82	18.17	3.06	3.85	8.55	3.14	2.16	4.32	3.36	1.71	3.18	7.40	0.76	2.84
Hamrle3	22.14	2.94	14.15	11.52	2.17	24.14	2.66	1.96	40.52	1.40	2.25	132.66	0.47	3.08
Hook_1498	7.97	15.49	3.06	3.91	7.73	3.20	2.18	4.11	3.39	1.70	2.85	7.20	0.83	2.93
kkt_power	31.73	5.22	3.73	20.10	2.77	4.17	6.23	1.69	4.51	3.70	1.68	10.00	1.33	1.84
memchip	9.36	4.78	4.18	4.04	2.52	4.23	2.26	1.62	4.27	1.69	1.44	8.08	0.38	1.45
nlpkkt80	6.90	10.87	2.58	3.59	6.27	2.91	1.69	3.34	3.33	1.10	2.71	8.01	0.45	2.58
Serena	8.18	19.81	3.20	4.12	9.26	3.37	2.30	4.91	3.60	1.82	3.11	8.79	1.07	3.02
StocF-1465	5.53	6.58	2.12	2.95	3.52	2.21	1.33	2.43	2.55	0.89	2.20	5.83	0.35	2.04
thermal2	3.78	2.50	1.36	1.68	1.31	1.39	1.07	0.80	1.41	0.61	0.66	2.47	0.18	1.04
webbase-1M	35.72	1.46	1.77	17.63	0.80	1.85	7.80	0.58	1.98	3.81	0.54	3.62	1.83	0.64

Table B.6 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using MC73 on NAR

<i>Matrix Name</i>	<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	12.09	2.73	5.49	23.59	1.59	2.89	34.63	1.31	1.84	90.77	0.48	1.89
atmosmodd	6.10	1.36	1.59	10.50	0.73	1.03	14.52	0.56	1.08	35.17	0.18	1.05
atmosmodl	7.55	1.60	1.79	13.03	0.86	1.16	17.64	0.66	1.26	43.09	0.19	1.24
cage14	38.98	3.59	7.38	69.62	1.85	4.92	96.84	0.93	4.07	237.37	0.43	4.86
dielFilterV3real	53.37	5.59	10.79	104.48	3.23	5.48	154.15	2.43	3.29	401.26	0.81	3.02
G3_circuit	4.00	1.73	1.39	7.35	0.93	0.87	10.25	0.61	0.85	25.25	0.14	1.12
Geo_1438	23.17	3.79	8.44	42.45	2.13	4.29	61.12	1.70	2.92	157.01	0.84	2.82
Hamrle3	3.13	4.97	2.38	5.59	1.56	2.06	7.77	0.19	2.33	18.73	0.02	3.17
Hook_1498	21.93	3.87	7.69	41.30	2.14	4.10	59.98	1.74	2.70	155.21	0.96	2.74
kkt_power	18.34	9.30	2.72	31.17	4.08	1.66	41.02	1.87	1.59	99.04	0.75	1.69
memchip	6.16	3.49	2.51	11.37	1.99	1.59	16.13	1.57	1.43	39.81	0.39	1.54
nlpkkt80	14.14	3.10	5.66	26.24	1.67	2.89	37.04	1.09	2.79	94.22	0.43	2.18
Serena	24.00	3.84	9.27	45.13	2.09	4.71	65.25	1.71	3.06	168.79	0.92	2.93
StocF-1465	10.92	2.86	3.30	20.19	1.34	2.21	28.86	1.05	2.21	73.00	0.36	1.87
thermal2	4.02	1.65	1.32	7.39	1.00	0.80	10.42	0.48	0.65	26.06	0.17	0.81
webbase-1M	2.73	8.73	0.83	5.08	5.51	0.56	7.28	0.86	0.55	18.36	0.74	0.60

Table B.7 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using MC73 on MERCAN

<i>Matrix Name</i>	<i>1 Process</i>		<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	6.69	34.28	2.60	3.36	16.53	2.51	1.64	8.57	2.58	0.81	4.80	2.61	0.44	2.52
atmosmodd	2.80	16.31	1.68	1.55	8.16	1.75	0.73	4.32	2.00	0.35	2.44	2.24	0.19	1.35
atmosmodl	3.35	19.08	1.99	1.82	9.68	2.02	0.89	5.19	2.26	0.46	2.82	2.38	0.23	1.63
cage14	9.75	47.99	8.50	5.62	25.75	12.55	4.29	13.23	17.51	0.95	6.92	24.00	0.48	3.54
dielFilterV3real	13.08	60.40	4.30	6.46	29.66	4.42	3.18	15.44	4.95	1.83	7.79	5.89	1.01	4.27
G3_circuit	3.66	17.27	1.58	1.97	8.67	1.64	0.87	4.84	1.62	0.43	2.53	1.65	0.22	1.59
Geo_1438	9.42	55.41	3.88	4.67	26.69	3.94	2.25	12.47	4.18	1.11	6.38	4.56	0.59	3.80
Hamrle3	27.08	16.76	13.53	12.48	8.84	22.15	2.90	4.76	38.06	1.21	2.76	61.99	0.56	1.49
Hook_1498	9.55	50.26	3.77	4.73	24.63	3.94	2.32	12.64	4.12	1.13	6.60	4.37	0.62	3.37
kkt_power	32.73	28.24	3.93	20.64	13.87	4.49	7.39	7.47	4.82	3.36	4.12	5.32	1.14	2.42
memchip	10.34	30.33	3.86	4.21	15.40	3.87	2.00	8.43	3.90	0.96	4.89	4.03	0.48	2.50
nlpkkt80	8.51	34.42	3.08	4.29	17.05	3.38	2.02	8.33	3.74	0.85	4.35	4.41	0.52	2.18
Serena	9.76	61.59	4.03	4.89	27.74	4.21	2.39	13.43	4.46	1.18	7.03	5.14	0.60	3.65
StocF-1465	6.35	30.12	2.50	3.34	14.53	2.62	1.45	8.25	2.99	0.75	4.08	3.35	0.37	2.14
thermal2	4.16	15.79	1.42	1.86	7.93	1.44	1.19	4.34	1.47	0.51	2.45	1.50	0.25	1.30
webbase-1M	38.47	8.42	1.82	19.23	4.78	1.91	9.47	3.12	2.06	4.39	1.45	2.33	2.00	0.88

Table B.8 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using MC73 on MERCAN

<i>Matrix Name</i>	<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	17.21	3.37	16.10	33.62	1.66	8.86	49.82	0.81	5.19	65.73	0.43	3.04
atmosmodd	6.32	1.46	8.11	11.38	0.71	4.49	16.20	0.38	2.35	20.63	0.20	1.37
atmosmodl	7.66	1.68	9.54	13.74	0.85	5.20	19.25	0.48	3.04	24.68	0.22	1.59
cage14	40.09	3.98	25.88	72.52	1.74	13.35	102.07	0.76	6.90	128.33	0.42	3.66
dielFilterV3real	72.53	6.39	29.75	143.22	3.16	14.54	213.00	1.67	7.52	280.02	0.88	4.10
G3_circuit	4.08	1.92	8.80	7.68	0.89	4.75	11.13	0.41	2.72	14.40	0.22	1.49
Geo_1438	30.64	4.60	25.02	57.11	2.22	12.67	83.48	1.13	7.02	109.33	0.60	3.48
Hamrle3	3.23	6.31	7.00	5.95	1.60	3.11	8.45	0.23	1.69	10.89	0.04	0.94
Hook_1498	29.55	4.67	24.80	56.24	2.28	12.53	82.76	1.13	6.87	108.27	0.61	3.54
kkt_power	16.74	10.76	14.11	30.54	4.82	7.34	42.95	2.09	4.01	54.03	0.95	2.33
memchip	5.59	3.72	15.75	10.86	1.79	8.96	15.79	0.86	5.00	20.43	0.47	2.67
nlpkkt80	17.47	3.95	16.82	32.47	1.77	8.35	46.83	1.01	4.26	60.95	0.43	2.14
Serena	32.58	4.65	27.09	61.51	2.17	13.82	90.17	1.11	6.54	117.20	0.55	3.66
StocF-1465	12.18	3.26	14.30	23.02	1.48	7.33	33.71	0.80	4.06	44.26	0.37	2.18
thermal2	4.14	1.83	7.90	7.81	0.92	4.26	11.38	0.45	2.29	14.79	0.23	1.31
webbase-1M	3.30	9.37	4.49	6.27	5.15	2.40	9.17	0.87	1.32	12.24	0.66	0.82

Table B.9 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using RCM on NAR

<i>Matrix Name</i>	<i>1 Process</i>		<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	0.69	10.85	2.08	0.35	5.42	2.04	0.19	2.78	2.09	0.14	1.78	4.23	0.04	1.79
atmosmodd	0.45	2.53	1.67	0.16	1.35	1.73	0.10	0.90	1.98	0.05	0.97	4.44	0.02	1.05
atmosmodl	0.51	2.96	1.98	0.20	1.58	2.04	0.14	1.05	2.30	0.07	1.13	4.03	0.01	1.24
cage14	1.48	7.39	7.95	0.68	4.79	12.82	0.38	3.42	18.31	0.26	2.81	51.06	0.03	3.77
dielFilterV3real	1.27	16.85	3.36	0.53	8.50	3.47	0.36	4.40	3.93	0.21	2.54	10.20	0.07	3.01
G3_circuit	0.35	2.70	1.55	0.20	1.48	1.59	0.08	1.02	1.60	0.05	1.10	2.85	0.02	1.21
Geo_1438	0.78	13.04	3.07	0.56	6.61	3.15	0.24	3.54	3.36	0.16	2.29	7.41	0.05	2.43
Hamrle3	1.03	2.65	14.01	0.42	1.93	23.90	0.19	1.77	40.33	0.11	2.04	132.74	0.02	3.03
Hook_1498	0.80	12.74	3.06	0.38	6.54	3.20	0.23	3.53	3.38	0.21	2.31	7.19	0.06	2.57
kkt_power	3.41	4.92	3.71	1.23	2.63	4.15	0.47	1.78	4.49	0.46	1.62	10.00	0.03	1.76
memchip	1.30	5.47	4.18	0.45	2.97	4.23	0.20	2.01	4.27	0.14	1.83	8.67	0.04	1.99
nlpkkt80	0.62	5.58	2.57	0.28	2.84	2.89	0.13	1.73	3.32	0.09	1.41	8.00	0.02	1.67
Serena	0.76	13.47	3.20	0.65	6.96	3.38	0.24	3.77	3.59	0.26	2.31	8.74	0.08	2.52
StocF-1465	0.93	5.29	2.12	0.34	2.77	2.21	0.16	1.57	2.56	0.12	1.25	5.83	0.03	1.73
thermal2	0.55	2.46	1.36	0.21	1.29	1.39	0.11	0.78	1.42	0.06	0.66	2.62	0.02	0.86
webbase-1M	0.28	1.38	1.77	0.12	0.77	1.85	0.05	0.59	1.98	0.03	0.57	3.70	0.01	0.64

Table B.10 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using RCM on NAR

<i>Matrix Name</i>	<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	12.09	0.27	5.43	23.60	0.17	2.80	34.67	0.10	1.73	90.80	0.04	1.68
atmosmodd	6.05	0.16	1.36	10.44	0.07	0.90	14.40	0.05	0.93	35.24	0.01	1.03
atmosmodl	7.71	0.19	1.59	12.96	0.10	1.04	17.68	0.07	1.12	43.10	0.01	1.16
cage14	39.10	0.67	4.52	70.88	0.34	3.11	96.88	0.22	2.82	236.83	0.03	3.86
dielFilterV3real	53.34	0.52	8.49	104.63	0.33	4.42	154.08	0.27	2.43	400.92	0.14	2.74
G3_circuit	3.99	0.17	1.46	7.31	0.08	1.04	10.25	0.05	1.06	24.99	0.01	1.07
Geo_1438	23.08	0.36	6.68	42.46	0.21	3.52	61.03	0.16	2.26	156.51	0.04	2.44
Hamrle3	3.13	2.04	2.14	5.59	0.04	2.05	7.79	0.02	2.24	18.70	0.01	3.22
Hook_1498	21.95	0.37	6.42	41.52	0.21	3.47	59.97	0.15	2.18	155.44	0.05	2.40
kkt_power	18.08	1.00	2.68	31.45	0.56	1.75	41.67	0.49	1.61	98.20	0.03	1.77
memchip	6.13	0.45	2.97	11.37	0.20	1.99	16.11	0.13	1.91	39.81	0.03	1.99
nlpkkt80	14.50	0.19	2.84	26.30	0.12	1.58	37.19	0.09	1.33	93.87	0.02	1.58
Serena	24.14	0.38	6.86	45.14	0.23	3.72	65.16	0.15	2.26	168.40	0.07	2.57
StocF-1465	10.96	0.25	2.72	20.25	0.14	1.58	28.86	0.09	1.25	73.01	0.03	1.59
thermal2	4.01	0.21	1.29	7.37	0.10	0.79	10.39	0.06	0.63	26.03	0.02	0.88
webbase-1M	2.73	0.12	0.82	5.07	0.03	0.58	7.36	0.02	0.55	18.45	0.00	0.63

Table B.11 Sequential permutation and multiplication time; Partitioning, parallel permutation and multiplication time of matrix partitioned using METIS and then permuted using RCM on MERCAN

<i>Matrix Name</i>	<i>1 Process</i>		<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>	<i>Parti- tioning (sec)</i>	<i>Permu- tation (sec)</i>	<i>Multipli- cation (sec)</i>
af_shell10	0.81	30.19	2.58	0.43	14.79	2.50	0.20	8.00	2.57	0.11	4.41	2.59	0.05	2.42
atmosmodd	0.47	14.39	1.70	0.23	7.20	1.77	0.12	3.89	2.03	0.05	2.27	2.28	0.02	1.32
atmosmodl	0.55	16.88	2.00	0.27	8.45	2.05	0.15	4.64	2.29	0.05	2.48	2.36	0.02	1.47
cage14	1.42	36.91	8.51	0.75	19.70	12.58	0.37	10.67	17.58	0.12	5.72	24.09	0.07	3.28
dielFilterV3real	1.63	43.63	4.34	0.64	21.41	4.47	0.36	10.76	4.99	0.18	6.29	5.88	0.09	3.60
G3_circuit	0.50	16.29	1.60	0.23	8.28	1.62	0.09	4.67	1.63	0.04	2.72	1.68	0.02	1.47
Geo_1438	0.94	39.54	3.86	0.62	18.97	3.91	0.23	10.47	4.14	0.11	6.16	4.55	0.06	3.37
Hamrle3	0.88	16.22	13.57	0.39	8.39	22.35	0.20	5.00	38.40	0.07	2.92	63.08	0.03	1.61
Hook_1498	0.96	40.13	3.73	0.45	19.56	3.89	0.22	10.54	4.09	0.14	5.88	4.34	0.08	3.32
kkt_power	2.62	26.56	3.91	0.99	13.32	4.51	0.42	8.37	4.87	0.16	4.01	5.37	0.09	2.46
memchip	1.35	31.31	3.77	0.51	15.98	3.86	0.24	8.55	3.93	0.12	5.01	3.99	0.06	2.82
nlpkkt80	0.73	26.57	3.06	0.36	11.99	3.37	0.15	5.98	3.73	0.07	3.23	4.42	0.04	1.68
Serena	0.93	41.64	4.03	0.75	20.50	4.21	0.24	11.21	4.44	0.17	5.58	5.15	0.08	3.57
StocF-1465	1.02	25.83	2.50	0.44	12.13	2.60	0.21	5.70	2.97	0.13	3.44	3.35	0.06	1.90
thermal2	0.58	15.25	1.40	0.24	7.68	1.43	0.11	4.13	1.46	0.05	2.32	1.48	0.02	1.27
webbase-1M	0.29	8.20	1.82	0.14	5.31	1.91	0.08	2.75	2.04	0.03	1.39	2.33	0.01	0.88

Table B.12 Partitioning, parallel permutation and multiplication time of matrix partitioned using PATOH and then permuted using RCM on MERCAN

<i>Matrix Name</i>	<i>2 Processes</i>			<i>4 Processes</i>			<i>8 Processes</i>			<i>16 Processes</i>		
	<i>Partitioning (sec)</i>	<i>Permutation (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Permutation (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Permutation (sec)</i>	<i>Multiplication (sec)</i>	<i>Partitioning (sec)</i>	<i>Permutation (sec)</i>	<i>Multiplication (sec)</i>
af_shell10	17.25	0.39	14.88	33.81	0.19	8.84	49.89	0.09	4.40	65.83	0.05	2.61
atmosmodd	6.43	0.23	7.09	11.26	0.10	3.96	16.15	0.04	2.13	20.55	0.02	1.28
atmosmodl	7.57	0.26	8.43	13.68	0.12	4.65	19.04	0.05	2.56	24.64	0.03	1.52
cage14	38.83	0.65	18.96	72.72	0.28	10.42	100.20	0.11	5.88	127.80	0.05	3.12
dielFilterV3real	72.33	0.64	21.51	142.71	0.32	11.80	212.66	0.18	6.95	279.45	0.09	3.96
G3_circuit	4.03	0.19	8.36	7.69	0.09	4.54	11.02	0.04	2.47	14.23	0.02	1.51
Geo_1438	30.48	0.44	18.82	57.11	0.22	9.57	83.21	0.12	5.59	109.10	0.06	3.01
Hamrle3	3.24	0.28	6.66	5.91	0.06	3.23	8.50	0.02	1.56	10.90	0.01	1.02
Hook_1498	29.55	0.46	19.79	56.31	0.23	10.64	82.54	0.12	5.73	108.40	0.06	3.40
kkt_power	16.57	0.88	13.05	30.47	0.44	7.56	42.60	0.16	4.20	54.00	0.08	2.47
memchip	5.63	0.51	16.31	10.79	0.23	8.52	15.83	0.12	4.60	20.35	0.05	2.77
nlpkkt80	17.26	0.29	11.19	32.55	0.15	5.77	47.12	0.07	3.34	60.96	0.03	1.87
Serena	32.47	0.46	20.17	61.38	0.22	12.11	89.90	0.11	6.09	117.16	0.06	3.53
StocF-1465	12.22	0.34	11.67	23.11	0.18	6.29	33.55	0.07	3.48	43.89	0.04	1.82
thermal2	4.18	0.23	7.60	7.89	0.11	4.15	11.27	0.05	2.27	14.54	0.02	1.30
webbase-1M	3.28	0.08	4.26	6.27	0.03	2.35	9.10	0.02	1.38	12.23	0.01	0.81