Noname manuscript No.
(will be inserted by the editor)

Layer Based Partition for Matrix Multiplication on Heterogeneous Processor Platforms

Yang Liu · Li Shi · Junwei Zhang · Thomas G. Robertazzi

Received: date / Accepted: date

Abstract While many approaches have been proposed to analyze the problem of matrix multiplication parallel computing, few of them address the problem on heterogeneous processor platforms. It still remains an open question on heterogeneous processor platforms to find the optimal schedule that balance the load within the heterogeneous processor set while minimizing the amount of communication. A great many studies are based on rectangular partition, whereas the optimality of rectangular partition as the basis has not been well justified.

In this paper, we propose a new method that schedules matrix multiplication on heterogeneous processor platforms with the mixed co-design goal of minimizing the total communication volume and the multiplication completion time. We first present the schema of our *layer based partition*(LBP) method. Subsequently, we demonstrate that our approach guarantees minimal communication volume, which is smaller than what rectangular partition can reach. We further analyze the problem of minimizing the task completion time, with network topologies taken into account. We solve this problem in both single-neighbor network case and multiple-neighbor network case. In single-neighbor network cases, we propose an equality based method to solve

Yang Liu
Stony Brook University
E-mail: yangliu89415@gmail.com

Li Shi
Stony Brook University
E-mail: lishi.pub@gmail.com

Junwei Zhang
Stony Brook University
E-mail: junwei.zhang@stonybrook.edu

Thomas G. Robertazzi
ECE Department, Stony Brook University
E-mail: Thomas.Robertazzi@stonybrook.edu

LBP, and simulation shows that the total communication volume is reduced by 75% from the lower bound of rectangular partition. In multiple neighbor network cases, we formulate LBP as a Mixed Integer Programming problem, and reduce the total communication volume by 81% through simulation. To summarize, this is a promising perspective of tackling matrix multiplication problems on heterogeneous processor platforms.

Keywords Matrix Multiplication · Heterogeneous processing · Optimization · Load balancing · Communication overhead

1 Introduction

Matrix multiplication has been widely performed in a variety of areas. For example, in image processing, a multiplication of projection matrix and system coefficient matrix is used to reconstruct the original images from the projections[1]. In signal processing, the discrete Fourier transform of a signal is calculated by multiplying the N-by-N DFT matrix with the signal matrix[2]. Many other applications include cryptography, computer graphics, economics, physics, electronics, etc, which all involve large scale data processing.

On homogeneous processor platforms, the problem of scheduling matrix multiplication load for parallel processing has been extensively studied, such as Canon's algorithm[3], SUMMA[5] and Solomonik's 2.5D algorithm[6], etc. However, these approaches generally ignore the heterogeneity of processors and links, as well as network topologies. Thus, in distributed computing and heterogeneous platforms, those algorithms fail to apply because they can't guarantee load balance in those scenarios. To minimize the task completion time without considering the heterogeneity of the system is simply impossible. Therefore, for distributed computing and heterogeneous systems, additional factors need to be taken into account, such as heterogeneous processor speeds, heterogeneous link speeds, network topologies, distributed storage, etc.

To schedule matrix multiplication on heterogeneous processor platforms, researchers usually consider the following questions.

1. How to allocate the computing load to minimize the total communication volume?
2. How to minimize the task completion time?

To optimize the total communication volume, many previous research works apply *rectangular partition* on the result matrix [13]-[26]. Rectangular partition, which adopts the well-known *divide and conquer* strategy, divides the result matrix into multiple sub-rectangles, and assigns each sub-rectangle's computing load to different processors respectively. However, approaches based on *rectangular partition* generally have the following drawbacks:

1. The restriction of each division's shape being rectangle brings difficulty to find the optimal partition that minimizes the total communication volume.
2. The best communication volume of *rectangular partition* may not be global optimal.

There have been perspectives using *non-rectangular partition*[27][28]. However, these approaches only allows one division of the partition to be in random shape, while keeping the majority rest still in the shape of rectangle. Thus, these approaches do not completely resolve the problem brought by the *rectangular partition*.

Motivated by this, we proposes another approach called *layer based partition*(LBP). Rather than assigning a certain area of rectangular sub-matrix to a specific processor, our algorithm assigns each processor one layer of the result matrix's computing load. Each layer is of the same shape with the result matrix. We will show that this method guarantees the optimality of the total communication volume.

We further study the problem of scheduling matrix multiplication on heterogeneous processor networks with the goal of minimizing the multiplication completion time. While several approaches have been proposed[22]-[29], none of them have addressed problem in the context of a specific network topology, like a heterogeneous mesh. Besides, most previous works consider the problem in real number domain such that it is allowed for a processor to get 0.3 rows, for instance. Comparatively, we consider the problem in integer domain which is more applicable in real practice. In our paper, we study the problem in two specific networks: star network and mesh, and propose different strategies to minimize their overall finishing time.

1.1 Assumption and Constraints

In this paper we only consider acquiring the distributed results of multiplication process, whereas the aggregation of those distributed results is out of the scope of this discussion. This is because we can exploit distributed storage to store those results in a distributed manner, rather than aggregate them together immediately. Since addition processes are of much lighter weight than multiplication processes, we assume that we are at a decent state once all the $O(N^3)$ multiplication procedures are done. Then, we store these multiplication results distributedly in the memory or hard disk of each processor. The aggregation process can be done asynchronously or only when necessary. In the concept of *rectangular partition*, that means we consider task completion once each sub-rectangle result is acquired, while the combination of these sub-rectangles are out of the scope of this discussion. In the concept of our *layer based partition*, that means we regard task as completed once each layer of the result matrix has been calculated and stored, whereas the summation process of these layers are left with asynchronous process. In either way, once all the distributed multiplication results are acquired, we mark that as the completion of the task. This is an important assumption for the comparison between *layer based partition* and *rectangular partition* in the latter part of this paper. We apply this assumption when calculating the total communication volume and task completion time.

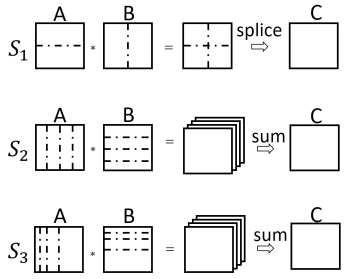
Memory limit can be a tight constraint here, especially for the case of *layer based partition* when the size of the matrix gets enormously large. But those super-large matrices are usually not read intensive. With distributed storage system we can store each layer of the result matrix on the hard disk of each processor. We only need to read from the disk and add up each layer when there's an infrequent read request.

1.2 An Example

To better illustrate this matrix multiplication scheduling problem, an example is shown in figure 1. Consider a task of multiplying two $8*8$ matrices(A and B) using four processors P_1, P_2, P_3 and P_4. Suppose the computing power of P_1, P_2, P_3 and P_4 are 1, 1, 2 and 4 respectively. How to schedule the computing load onto these four processors to optimize communication volume and multiplication finishing time? Figure 1 provides three scheduling schemes S_1, S_2 and S_3. S_1 represents a *rectangular partition* of the result matrix where each of the processor is assigned calculating one of the rectangular sub-matrices. S_2 and S_3 represent a *layer based partition* scheme. Specifically, S_2 is an evenly divided scheme, in which each processor takes an equal number of rows and columns and computes one layer of the result matrix. Compared to S_1, S_2 remains unchanged in overall finishing time, but substantially reduces the total communication volume. S_3 also partitions the matrix multiplication load based on layers, but make the number of columns taken by each processor being proportional to its computing power. As a result, S_3 keeps a low communication volume like S_2, while optimizing the multiplication finishing time. Note that according to our previous assumption in the introduction part, we only consider the process of getting the distributed results, whereas aggregating the distributed results can be done asynchronously and is out of the scope of this discussion.

The reason that S_2 and S_3 has much less communication volume compared to S_1 is that entries of the matrices are sent only once in S_2 and S_3 whereas they are sent twice in S_1. For example, in S_1, the first row of matrix A is sent to both P_1 and P_2 in order to calculate the upper left sub-matrix and upper right sub-matrix respectively. In contrast, in S_2 and S_3, each entry of matrix A and matrix B is sent only once. Therefore, the total communication volume of S_2 and S_3 is greatly less than S_1.

We can see that when performing matrix multiplication, *layer based partition* generates much less communication volume compared to *rectangular partition*. With correct distribution of multiplication loads to processor, we can further optimize the multiplication completion time. However, in practice, the problem of scheduling matrix multiplication load is more sophisticated, when we consider this problem in a specific network environment, in which factors like network topology, communication mode, etc have to be taken into account. It's even more complex, if we consider the heterogeneity of the system. This paper explores these cases in details.



		P_1	P_2	P_3	P_4	Total
Volume	S_1	64	64	64	64	256
	S_2	32	32	32	32	128
	S_3	16	16	32	64	128
Time	S_1	128	128	64	32	128
	S_2	128	128	64	32	128
	S_3	64	64	64	64	64

Fig. 1 Examples: scheduling matrix multiplication load to 4 processors.

1.3 Our Contributions

Our main contributions are:

1. We analyze the disadvantages of *rectangular partition*. First, we show it's essentially difficult to obtain a communication-optimal partition. Second, we show that the best communication volume achieved by *rectangular partition* is not globally optimal. We propose the lower bound of matrix multiplication's communication volume, and we prove *rectangular partition* consumes a total communication volume larger than the lower bound.
2. In contrast, we propose *layer based partition*(LBP) scheme. We show that our scheme is superior to *rectangular partition* because: A). It is easy to obtain a communication-optimal partition. B). It can reach the lower bound of total communication volume.
3. We study the load scheduling problem to minimize overall finishing time in star networks. We propose an equality based strategy, and apply this strategy under four communication modes respectively: SCSS, SCCS, PCCS, PCSS.
4. We study the load scheduling problem to minimize overall finishing time in mesh networks. We formulate this problem as a mixed integer programming problem called *MFT-LBP*. We propose an algorithms called *PMFT-LBP* to solve it, which contains 3 phases. We also provide a heuristic to reduce time complexity.

One thing to notice is that the matrices discussed in this paper are dense matrices with hundreds or thousands of rows/columns, which are very common in current parallel processing environment.

The rest of the paper is organized as follows. Section 2 reviews the related research. Section 3 discusses the difficulties of *rectangular partition*ing. Section 4 introduces *layer based partition* scheme, and analyzes its improvements over *rectangular partition*. Section 5 and 6 studies the load scheduling scheme of LBP in star network and mesh, respectively. Section 7 studies the performance of our algorithms through simulations. Finally, section 8 concludes the paper.

2 Related Work

A great amount of research effort has been devoted to the problem of matrix multiplication parallel/distributed processing. In this section, we categorize those most related works into the following parts:

Approaches on homogeneous platforms. Homogeneous platform assumes all the computing/comunication resources and environment are identical. Matrix multiplication scheduling on homogeneous platforms have been extensively studied in[3] - [10], among which, Canon introduces the first parallel algorithm on homogeneous grids[3]. Fox *et al.* extend the analysis on two-dimensional mesh and hypercubes[4]. SUMMA overcomes the shortcomings of Cannon's and Fox's, and becomes the most widely applied parallel matrix multiplication scheme[5]. Solomonik *et al.* [6] proposes an method which is known as the '2.5D Algorithm', which can achieve asymptotically less communication than Canon's algorithm and be faster in practise. Malysiak *et al.* [7] present a novel model of distributing matrix multiplication within homogeneous systems with multiple hierarchies. Scheduling of sparse-dense matrix multiplication has been studied in [8][9].

However, when computing scale gets larger and larger, heterogeneous computing is more suitable than homogeneous computing for super large distributed and parallel processing. An example is, CPU-GPU heterogeneous processing is inevitable a trend to achieve higher computing performance[11]. Heterogeneous System Architecture(HSA) is another example that uses multiple processor types on the same integrated circuit to provide the best overall performance[12].

Approaches on heterogeneous platforms. Efforts have been devoted to analyze matrix multiplication on heterogeneous platforms[13]-[21]. Some researchers try to extend those parallel processing algorithms from homogeneous platforms to heterogeneous platforms. Both Kalinov[13] and Quintin *et al.*[14] investigate the scalability to modify SUMMA[5], to fit into heterogeneous processor platforms. Ohtaki *et al.* [15] propose a scheme to apply the Strassen's algorithm on heterogeneous clusters.

However, these approaches only optimize communication volume, yet fail to concern whether multiplication completion time gets optimized as well.

In addition to approaches that try to apply homogeneous parallel algorithms, other researchers seek new perspectives. Alonso *et al.* [16] use two strategies to implement parallel solvers for dense linear algebra problems on heterogeneous clusters. Malik *et al.* [17] propose a topology-aware matrix multiplication algorithm, and they base their hierarchical communication model on irregular 2D rectangular partition. Zhong *et al.* [18] utilize functional performance model to balance load on heterogeneous networks of uni-processor computers. Demmel *et al.* [19] propose a communication-efficient algorithm for all dimensions of rectangular matrices, apart from square matrices. Beaumont *et al.* [20] compares static, dynamic and hybrid resource allocation strategies

for matrix multiplication, and analyse the benefit of introducing more static knowledge in runtime libraries.

The most utilized partition method in these approaches is *rectangular partition*. There exists a significant amount of research digging into the problem of *rectangular partition* on heterogeneous platforms, as discussed below.

Rectangular partition approaches on heterogeneous platforms. While so many approaches have been proposed based on *rectangular partition* of matrix, researchers start to concern the optimal *rectangular partition*[22][23][24]. However, even though the optimal *rectangular partition* problem attracts a lot of attention, the optimal partition that minimizes the total communication volume remains an open question. Researchers start to realize the problem is complex. Ballard *et al.* [25] study the lower bounds of communication volume, and the difficulty of finding communication optimal *rectangular partition*. Beaumont *et al.* [26] prove that given the area of each sub-rectangle, it is a NP-complete problem to find communication optimal *rectangular partition* of a specific matrix. In regard to this, DeFlumere *et al.* [27] question the optimality of *rectangular partition*, and propose a perspective of *non-rectangular partition*. Further, DeFlumere *et al.* [28] show that this non-rectangular scheme outperforms *rectangular partition* in terms of communication volume, and generalize this algorithm to three processors' scenario. Nagamochi *et al.* [29] propose a recursive partitioning algorithm that dissects a rectangle into rectangles with specified areas. Beaumont *et al.* [30] in addition, propose a new approximation algorithm for matrix partitioning by adopting the idea of non-rectangular partition, and the recursive partitioning algorithm proposed by Nagamochi *et al.* Fuenschuh *et al.* [31] present a polynomial time approximation algorithm that solves a soft rectangle packing problem, and derive an upper bound estimation on its approximation ratio.

In summary, researchers seemly have begun to realize the difficulty in finding the optimal *rectangular partition* that balance loads while minimize communication volume. Beaumont's work *et al.* [26] explicitly reveals that it is a NP-complete problem. Moreover, though alternative perspectives like *non-rectangular partition* have been proposed[27][28], those perspectives keep the majority of the partitions still in the shape of rectangle, which do not completely resolve the restriction brought by geometrical shapes. In contrast, in *layer based partition* scheme, we avoid the NP-complete problem and make it very easy to obtain a communication-optimal partition, which in the meanwhile, reaches the lower bound of total communication volume.

3 Layer Based Partition(LBP)

In this section we propose a new method - the *layer based partition*(LBP) scheme to tackle matrix multiplication on heterogeneous processor platforms.

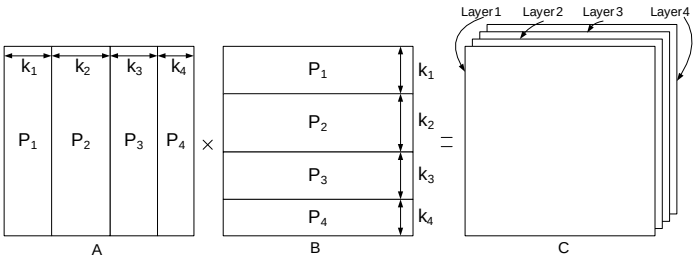


Fig. 2 Layer Based Partition Pattern.

3.1 Scheme Overview

While the goal remains as conducting two $N * N$ matrices' multiplication, the approach taken by LBP is different from *rectangular partition*. In LBP, each processor is responsible for calculating one layer of the output matrix. Each layer is of the same dimension of the output matrix, and the output matrix is the aggregation of all layers.

Figure 2 shows an example of this LBP scheme with four processors cooperating on two $N * N$ matrices' multiplication. Processor P_1 takes matrix A's leftmost k_1 columns and B's upmost k_1 rows, and then do multiplication. The result is still a $N * N$ matrix, which is actually the 1st layer of the final output matrix. The same method applies to the rest of the processors, with processor P_2 taking charge of 2nd layer, processor P_3 taking charge of 3rd layer, and processor P_4 taking charge of the last layer. The final output matrix is the sum of these four layers.

3.2 Improvements over Rectangular Partition

The important improvements of LBP scheme over *rectangular partition* include

- LBP avoids the NP-complete problem that partitions the matrix into rectangular sub-matrices. Instead, the work load each processor undertakes in LBP is one layer that is of the same size of the output matrix. We don't need to consider each sub-matrix's shape, and how to combine them into a matrix. We only need to consider how to divide matrix A based on columns and matrix B based on rows, which is comparatively much easier.
- LBP also takes less total volume of data sent by the source than *rectangular partition*. LBP essentially reaches the communication lower bound, as will be proven below.

In order to ensure an equal base of comparison, we assume in the following part of this paper that, all source nodes do not take part in computation.

Theorem 1 (Layer Based Partition Theorem) *LBP generates the minimal total communication volume in conducting two square matrices' multiplication.*

To prove the theorem, we firstly present the lower bound of communication volume.

Lemma 1 *For all scheduling schemes for two $N * N$ matrices' multiplication, the lower bound of communication volume is $2N^2$, if source does not take part in processing.*

Proof When conducting two $N * N$ matrices' multiplication, both matrices have to be sent from the source to the computing node, because the source doesn't take part in computing. Each matrix contributes a communication volume of N^2, so two matrices together are $2N^2$. Since each entry of these two matrices has to be sent at least once, the total communication volume is always greater than or equal to $2N^2$. Thus the lower bound stands.

Back to Theorem 1. Suppose there are p processors, therefore according to LBP, the task is divided into p layers. Processor p_1 takes matrix A's leftmost k_1 columns and B's upmost k_1 rows, and thus the communication volume to transfer the necessary processing data from source to processor p_1 is $N * k_1 + N * k_1 = 2Nk_1$. Similarly, the communication volume of processor p_2 is $2Nk_2$, processor p_3 is $2Nk_3$, ...etc. The total communication volume is

$$C_{LBP} = \sum_{i=1}^{p} 2Nk_i = 2N \sum_{i=1}^{p} k_i = 2N^2$$

As shown above, the communication volume of *layer based partition* scheme reaches the lower bound. Therefore, *layer based partition* scheme is communication-optimal.

Lemma 2 *Rectangular partition consumes a total communication volume greater than the lower bound.*

Proof In *rectangular partition*, suppose the output matrix is of size $N * N$, and each sub-rectangle's height is h_i, width is w_i, and area is s_i. The total communication volume according to [26] is:

$$C_{REC} = \sum_{i=1}^{p} (h_i + w_i) * N$$

Because $h_i + v_i \geq 2\sqrt{s_i}$, therefore we have:

$$C_{REC} \geq 2N * \sum_{i=1}^{p} \sqrt{s_i} \tag{1}$$

In the meanwhile, we have the sum of each sub-rectangle:

$$\sum_{i=1}^{p} s_i = N^2$$

$$\sum_{i=1}^{p} (\sqrt{s_i})^2 = N^2 \tag{2}$$

Since each s_i is positive and $i \geq 2$:

$$\sum_{i=1}^{p} (\sqrt{s_i})^2 < (\sum_{i=1}^{p} \sqrt{s_i})^2$$

Combined with equation (2), we get:

$$(\sum_{i=1}^{p} \sqrt{s_i})^2 > N^2$$

$$\sum_{i=1}^{p} \sqrt{s_i} > N \tag{3}$$

Take (3) back to (1),

$$C_{REC} > 2N^2 \tag{4}$$

Thus the total communication volume of LBP(C_{LBP}) is less than *rectangular partition*(C_{REC}).

3.3 Memory Limit

Memory limit may turn out to be a tight constraint for *layer based partition*, especially when the matrices are quite large. But since we can aggregate the distributed multiplication results asynchronously, or only when requested, we can store the results in a distributed manner on the hard disk of each processor. We only need to read from the disks and add up each layer when theres a read request, which shouldn't be very frequent for large matrices like this. After all, if a result matrix's size exceeds the memory limit, after aggregating all the layers, it has to be stored on a hard disk anyway. We change this centralized storage manner to a distributed manner, by storing each layer of the result matrix on the hard disk of each processor.

3.4 Summary

To summarize, *layer based partition* scheme avoids the NP-complete *rectangular partition* problem, and has been proven to be communication-optimal. The next step is to discuss how to apply it in different network topologies.

In the following section, we discuss the details of applying LBP scheme on heterogeneous processor networks. We focus on analyzing LBP's load balancing strategy, namely, how to schedule load distribution among different processors in the network. Since the LBP scheme already ensures communication volume to be minimal, the optimization goal of load balancing strategy

is to optimize the task completion time, which is another primary and widely discussed target of optimization on heterogeneous processor platforms. Again, as we mentioned in the introduction part, we mark it as the completion of the task once all the distributed multiplication results are acquired on each processor. The total time it takes to compute those multiplication results of all the layers is defined as the task completion time here.

4 LBP in Single-Neighbor Networks

To begin with, we discuss applying LBP in networks where each node can only receive loads from at most one of its neighbors. That node is allowed to further send loads to its other neighbors, except for the one where it obtains data from. We name this type of networks *single-neighbor networks* in the following parts of our paper. Typical examples of *single-neighbor networks* include star network, tree network, muti-level tree, etc. Here we use star network to discuss the load balancing strategy under four scenarios: Sequential Communication Simultaneous Start (SCSS), Sequential Communication Consecutive Start (SCCS), Parallel Communication Simultaneous Start (PCSS), Parallel Communication Consecutive Start (PCCS). Sequential Communication means each node can only send load to its neighbors sequentially, while, Parallel Communication allows the source to transmit simultaneously. Consecutive Start means each non-source processor can only start processing data after receiving all the data it needs, while, Simultaneous Start allows each node to start processing while receiving data.

Theorem 2 (Single Source Network Load Balancing Theorem) *For single-neighbor network, to reach load balancing is to have all the nodes finish working at the same time.*

This load balancing theorem comes from Bharadwaj etc.'s monograph[33]. The idea is if not all processors finish processing at the same time, loads can be reduced from busy processors, and assigned to idle processors to speed up the overall process. Consequently, the minimal finishing time is obtained only when all processors finishing working at the same time.

Table 1 Table of the Variables/Constants for Chapter 4

Variables/Constants	Meaning
k_i	The number of rows or columns assigned to ith processor
w_i	The inverse of the computing speed of ith processor
z_i	The inverse of the link speed of the ith link
T_{cp}	Computing intensity constant: Unit load on ith processor is processed in $w_i T_{cp}$ seconds
T_{cm}	Communication intensity constant: Unit load on ith link can be transmitted in $z_i T_{cm}$ seconds
T_f	The overall finishing time of the entire network

In practice, because k_i must be an integer, so it may be impossible to generate a set of integer $\{k_i\}$ that makes each processor finish exactly at the same time. Our approach is to relax k_i as real number first, solve the relaxed problem, then find the integer solution that is closest to the solution of the relaxed problem. We believe that by making the integer solution as close to the optimal solution of the relaxed problem (in real domain) as possible, it is more likely that we can get the optimal solution of the original problem in which each k_i is an integer.

4.1 Sequential Communication Simultaneous Start

In this subsection, we analyze the SCSS case where the source transmit loads to each processor sequentially, and in the mean time communication can overlap with computing. Figure 3(a) displays the time sequence of this case.

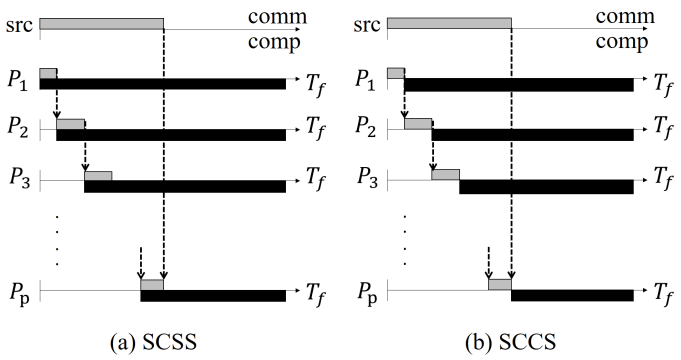


Fig. 3 SCSS mode and SCCS mode.

Each processor calculates one layer of the result matrix. Take processor i for instance. To work out i^{th} layer's data, processor i needs to get N rows $* k_i$ columns of A's data, plus k_i rows $* N$ columns of B's data. Thus its communication volume is $2k_iN$, and the corresponding communication time on i^{th} link is $2k_iNz_iT_{cm}$. Furthermore, for each entry of i^{th} layer, it needs k_i multiplications to get its value. There are as many as N^2 entries in this layer. Therefore the total number of multiplication is $k_i * N^2$. The corresponding computation time is $k_iN^2w_iT_{cp}$. Consider all the timing relationship shown in Figure 3(a), we have the following equations:

$$k_1N^2w_1T_{cp} = k_2N^2w_2T_{cp} + 2k_1Nz_1T_{cm} \tag{5}$$

$$k_2N^2w_2T_{cp} = k_3N^2w_3T_{cp} + 2k_2Nz_2T_{cm} \tag{6}$$

$$k_{i-1}N^2w_{i-1}T_{cp} = k_iN^2w_iT_{cp} + 2k_{i-1}Nz_{i-1}T_{cm} \tag{7}$$

$$\vdots$$

$$k_{p-1}N^2w_{p-1}T_{cp} = k_pN^2w_pT_{cp} + 2k_{p-1}Nz_{p-1}T_{cm} \tag{8}$$

$$k_1 + k_2 + k_3 + \cdots + k_p = N \tag{9}$$

The set of equations above consists of $p-1$ equations specifying the time sequence relationship between pairwise processors, and one equation specifying the normalization constraint. Meanwhile, we have p unknown variables $k_1, k_2, ..., k_p$. Therefore, the set of equations are solvable. Solving them, we get:

$$k_i = \prod_{j=2}^{i} \frac{Nw_{j-1}T_{cp} - 2z_{j-1}T_{cm}}{Nw_jT_{cp}} k_1, i = 2, 3, ...p \tag{10}$$

where k_1 is defined as:

$$k_1 = \frac{N}{1 + \sum_{i=2}^{p} \prod_{j=2}^{i} \frac{Nw_{j-1}T_{cp} - 2Z_{j-1}T_{cm}}{Nw_jT_{cp}}} \tag{11}$$

The overall finishing time of the whole network can be obtained through the following equation.

$$T_f = k_1N^2w_1T_{cp} \tag{12}$$

4.2 Sequential Communication Consecutive Start

In this section we discuss the case where the source sequentially assigns load to each processor, and communication can not overlap with computation. According to Figure 3(b), we have the following equations:

$$k_1N^2w_1T_{cp} = k_2N^2w_2T_{cp} + 2k_2Nz_2T_{cm} \tag{13}$$

$$k_2N^2w_2T_{cp} = k_3N^2w_3T_{cp} + 2k_3Nz_3T_{cm} \tag{14}$$

$$k_iN^2w_iT_{cp} = k_{i+1}N^2w_{i+1}T_{cp} + 2k_{i+1}Nz_{i+1}T_{cm} \tag{15}$$

$$\vdots$$

$$k_{p-1}N^2w_{p-1}T_{cp} = k_pN^2w_pT_{cp} + 2k_pNz_pT_{cm} \tag{16}$$

$$k_1 + k_2 + k_3 + \cdots + k_p = N \tag{17}$$

These p equations contains $k_1, k_2, ..., k_p$ unknown variables. The solution set is:

$$k_i = \prod_{j=2}^{i} \frac{Nw_{j-1}T_{cp}}{Nw_jT_{cp} + 2z_jT_{cm}} k_1, i = 2, 3, ...p \tag{18}$$

where k_1 is defined as:

$$k_1 = \frac{N}{1 + \sum_{i=2}^{p} \prod_{j=2}^{i} \frac{Nw_{j-1}T_{cp}}{Nw_jT_{cp} + 2z_jT_{cm}}} \tag{19}$$

The overall finishing time of the whole network is:

$$T_f = k_1N^2w_1T_{cp} + 2k_1Nz_1T_{cm} \tag{20}$$

4.3 Parallel Communication Consecutive Start

For the case where source communicates with each processor in a parallel manner, and communication can not overlap with computation, we have the following equations:

$$k_1 N^2 w_1 T_{cp} + 2k_1 N z_1 T_{cm} = k_2 N^2 w_2 T_{cp} + 2k_2 N z_2 T_{cm} \tag{21}$$

$$k_2 N^2 w_2 T_{cp} + 2k_2 N z_2 T_{cm} = k_3 N^2 w_3 T_{cp} + 2k_3 N z_3 T_{cm} \tag{22}$$

$$k_i N^2 w_i T_{cp} + 2k_i N z_i T_{cm} = k_{i+1} N^2 w_{i+1} T_{cp} + 2k_{i+1} N z_{i+1} T_{cm} \tag{23}$$

$$\vdots$$

$$k_{p-1} N^2 w_{p-1} T_{cp} + 2k_{p-1} N z_{p-1} T_{cm} = k_p N^2 w_p T_{cp} + 2k_p N z_p T_{cm} \tag{24}$$

$$k_1 + k_2 + k_3 + \cdots + k_p = N \tag{25}$$

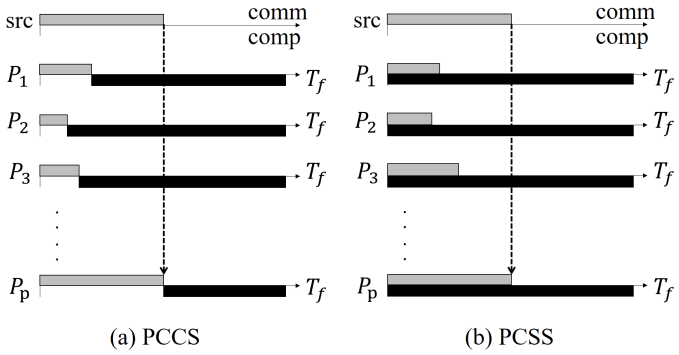


Fig. 4 PCCS mode and PCSS mode.

Solving the equations, we have:

$$k_i = \prod_{j=2}^{i} \frac{N w_{j-1} T_{cp} + 2 z_{j-1} T_{cm}}{N w_j T_{cp} + 2 z_j T_{cm}} k_1, i = 2, 3, ...p \tag{26}$$

where k_1 is defined as:

$$k_1 = \frac{N}{1 + \sum_{i=2}^{p} \prod_{j=2}^{i} \frac{N w_{j-1} T_{cp} + 2 z_{j-1} T_{cm}}{N w_j T_{cp} + 2 z_j T_{cm}}} \tag{27}$$

The overall finishing time of the whole network is:

$$T_f = k_1 N^2 w_1 T_{cp} + 2k_1 N z_1 T_{cm} \tag{28}$$

4.4 Parallel Communication Simultaneous Start

The case where source can transmit data to all processors at the same time, and communication can overlap with computation is shown in Figure 4(b). All processor start processing and end processing at the same time, and the size of load should be proportional to each processor's computing speed.

$$k_i N^2 w_i T_{cp} = k_{i-1} N^2 w_{i-1} T_{cp}, i = 2, 3..., p \tag{29}$$

$$k_1 + k_2 + k_3 + \cdots + k_p = N \tag{30}$$

Solving the equations, we have:

$$k_i = \prod_{j=2}^{i} \frac{w_{j-1}}{w_j} k_1, i = 2, 3, ..., p \tag{31}$$

$$k_1 = \frac{N}{1 + \sum_{i=2}^{p} \prod_{j=2}^{i} \frac{w_{j-1}}{w_j}} \tag{32}$$

$$T_f = k_1 N^2 w_1 T_{cp} \tag{33}$$

4.5 Integer Adjustment

By solving the above equations, we get a set of real numbers $\{k_i\}$ that generates the minimal task finishing time. The next step is to find the closest integer solution. We'll have a deeper discussion of how we obtain this closest integer solution from real number optimal solution, in the next chapter after we discuss *multi-neighbor networks*. Because we'll face the same problem there, too. Here we'll provide a simple heuristic as shown below.

We can round off the real number solution first, such that a processor gets the whole row/column assignment if it takes more than half of the fractional part of that row/column in the real number optimal solution. After the rounding off process, if the sum of each k_i fails to equal N, then we sort the processors in ascending order of their actual finish time $T_f(i)$. If the sum is less than N, then from the processor that has the smallest $T_f(i)$, we assign an extra row or column to that processor until the sum equals N. Otherwise, starting from the processor that has the largest $T_f(i)$, we remove one row/column from that processor until the sum equals N. After the process we obtain the actual integer assignment, we can further update the overall task finishing time.

5 LBP In Multi-Neighbor Networks

In this subsection, we discuss the load balance strategy for another type of network, in which each node is allowed to receive loads from more than one of its neighbors. We call this type of networks *multi-neighbor networks*, in contrast to the *single-neighbor networks* discussed previously. The target of load balancing strategy is still minimizing the task finishing time. The definition of variables and constants are listed in the following Table 2 and Table 3.

Table 2 Table of the Variables for Chapter 5

Variables	Meaning
$T_s(i)$	The start time of the ith node in the network
$T_f(i)$	The finish time of the ith node in the network
k_i	The number of columns or rows of the multiplier matrix that are assigned to ith node
$\phi(i,j)$	The volume of load transmitted from node i to node j

Table 3 Table of the Constants for Chapter 5

Constants	Meaning
$G(V,E)$	The mesh network with fixed topology
S	The set of source nodes
$z(i,j)$	The inverse of the link speed of the link connecting node i and j
T_{cm}	Communication intensity constant
$w(i)$	The inverse of the computing speed of ith processor
T_{cp}	Computing intensity constant
N	The size of both square multiplier matrices
p	The number of nodes in this mesh network
$\tau(i,j)$	specifies the position relationship of two nodes i and j in the network. $\tau(i,j)=1$ if i is in a position that should transmit to j, and $\tau(i,j)=0$ if otherwise

Note that in *multi-neighbor networks*, it is not applicable to apply the equal processing time load balancing strategy such as the one addressed in Theorem 2. Zhang et al. [34] analyze a similar problem. To summarise, if node i receives load from only one of its neighbors, says, j, we can obtain the following equation, $T_s(i) = T_s(j) + \phi(j,i)z(j,i)T_{cm}$, meaning that node i starts processing when it finishes receiving load from node j. However, if node i also receives load from another neighboring node, say j', the above equation may fail to stand, because the two neighbor nodes j and j' might not finish transmitting at the same time, i.e,

$$T_s(j) + \phi(j,i)z(j,i)T_{cm} \neq T_s(j') + \phi(j',i)z(j',i)T_{cm} \tag{34}$$

and we can not determine node i's start time.

Theorem 2 can optimize task finishing time for *single-neighbor networks* like tree, multi-level tree, etc. For *multi-neighbor networks* like multi-root tree, mesh, ring, torus, hypercube, since we can not apply Theorem 2, we formulate the load balancing problem as an optimization problem, called *Minimize Maximum Finishing Time in Layer Based Partition(MMFT-LBP)* problem.

Mesh is a typical *multi-neighbor network* where one single node can receive data from multiple neighbors. In the following part we discuss solving MMFT-LBP problem in mesh networks, to shed light on solving MMFT-LBP in other *multi-neighbor networks*.

5.1 MMFT-LBP In Mesh

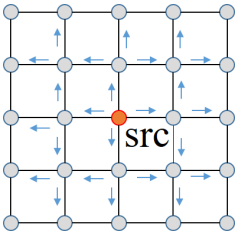


Fig. 5 data flow in mesh network.

In this subsection we focus on the *MMFT-LBP* problem in mesh network. We denote the mesh network as a graph $G(V, E)$, shown in figure 5. The graph contains p nodes which forms a dimension of $X*Y$, where $p = X*Y$. Generally speaking, the mesh has better performance in scheduling if the source is closer to geometric center. So we assume that the source is located at the center of this mesh network, and divides the mesh into four quadrants. The specific data flow pattern in each quadrant is shown in Figure 5.

For the limitation of scope of this paper, we only analyze PCCS mode, that is, data is forwarded in the network in the 'parallel communication and consecutive start' pattern. 'Parallel communication' allows each node to talk to its multiple neighbors at the same time. 'Consecutive start' means that each node has to wait until it receives its whole share of data before it can start computing. In short, we utilize PCCS mode as each node's communication and processing model in our *multi-neighbor networks*, and leave the other communication patterns for future study.

We present the *MMFT-LBP* problem in the following.

MMFT-LBP
Variables: $\{k_i\}$, $\{T_s(i)\}$, $\{T_f(i)\}$, $\{\phi(i,j)\}$, T_f
Objective:

$$Minimize: \max_{i \in G(V,E)} \{T_f(i)\} \tag{35}$$

Constraint:

$$T_s(i) = 0, \ \forall i \in S \tag{36}$$

$$T_s(i) = \max_{j \in G(V,E)} \{\tau(j,i)\big[T_s(j) + \phi(j,i) \cdot z(j,i) T_{cm}\big]\}, \ \forall i \notin S \tag{37}$$

$$T_f(i) = T_s(i) + k_i N^2 w(i) T_{cp},\ \forall i \in G(V,E) \tag{38}$$

$$2N^2 - \sum_{j \in G(V,E)} \phi(i,j) = 0,\ \forall i \in S \tag{39}$$

$$\sum_{j \in G(V,E)} \phi(j,i) - \sum_{j' \in G(V,E)} \phi(i,j') = 2k_i N,\ \forall i \notin S \tag{40}$$

$$\phi(i,j) \geq 0,\ \forall\ \tau(i,j) = 1 \tag{41}$$

$$\phi(i,j) = 0,\ \forall\ \tau(i,j) = 0 \tag{42}$$

$$k_i \in \mathbb{Z}_+,\ \forall i \in G(V,E) \tag{43}$$

$$k_i = 0,\ \forall i \in S \tag{44}$$

$$\sum_{i=1}^{p} k_i = N \tag{45}$$

Remarks:

- The objective of *MMFT-LBP*(35) is to minimize the maximum finishing time of the mesh network.
- Constraint (36) specifies the start time of the source. In this paper's case, there's only one node in set S.
- Constraint (37) specifies that the start time of those non-source nodes should be the maximum time that they finish receiving all loads from their adjacent neighbors.
- $\tau(i,j)$ is a constant once the mesh network is determined and fixed.
- Constraint (38) defines each node's finishing time. According to LBP, each node's computing load is the total number of multiplication it conducts $k_i * N^2$.
- Constraint (39) shows that because the source does not take part in processing, it sends out all of its load: the two multiplier matrices. And each entry of the matrices is sent only once.
- Constraint (40) defines the amount of load taken by those non-source nodes.
- Constraint (41) - (42) list different values of $\phi(i,j)$ in different cases. (41) is true if node i is in a position that should send load to node j. (42) applies when two nodes are not adjacent or node j is in a position that should send load to node i.
- The k_i in constraint (43) is the number of columns taken by each node, and should be integers.
- Constraint (44) shows that the source node does not take part in processing so its $k_i = 0$, and $T_f(i) = 0$ through constraint (38).
- Constraint (45) is the normalization constraint. The number of columns taken by each node should sum up to be the side length of the multiplier matrix.

Both the objective function and constraints contain maximum form of formulas. To solve it, we firstly reorganize the problem and remove those maximum forms of formulas, and show that they have the same optimal solution.

5.2 MFT-LBP In Mesh

The objective function of *MMFT-LBP* problem contains maximum form of formulas, which makes it hard to solve directly. To obtain the optimal solution of *MMFT-LBP* problem, the first step is to reorganize the objective function.

We introduce one additional unknown variable T_f, and a set of constraints $T_f \geq T_f(i), \forall i \in G(V,E)$, which ensure T_f to be no earlier than any node's finishing time $T_f(i)$. Then we transform the objective function from (35) to $Minimize : T_f$. The optimal solution to the original problem is still the optimal solution to the transformed problem, because when T_f is minimized, $T_f = \max_{i \in G(V,E)}\{T_f(i)\}$.

Similarly, we relax constraint (37) to be the following linear inequality.

$$T_s(i) \geq \tau(j,i)[T_s(j) + \phi(j,i) \cdot z(j,i)T_{cm}],\ \forall i \notin S \tag{46}$$

This inequality (46) implies processor i does not necessarily need to begin processing immediately when it finishes receiving all the data, it can choose to hold the data for a while and then start processing. Therefore, constraint (46) increases the solution space defined by constraint (37) to incorporate the following:

$$T_s(i) > \max_{j \in G(V,E)}\{\tau(j,i)\big[T_s(j) + \phi(j,i) \cdot z(j,i)T_{cm}\big]\},\ \forall i \notin S \tag{47}$$

Even if inequality (46) increases the feasible solution space, the optimal solution still remains the same. This is because the target of this problem is to minimize the overall finishing time, the minimal finishing time is achieved always when each node starts processing and forwarding once it completes receiving load from its neighbors, even if it is allowed to hold on for a while before it starts processing and forwarding.

In *MMFT-LBP* problem statements, if we use constraint (46) to replace constraint (37), we get a new problem called *Minimize Finish Time with Layer Based Partition(MFT-LBP)*. As we have discussed, we have the following theorem.

Theorem 3 (Maximization Relaxation Theorem) MMFT-LBP *problem has the same optimal solution with* MFT-LBP *problem.*

The full problem statement is presented as follows.
MFT-LBP
Variables: $\{k_i\}$, $\{T_s(i)\}$, $\{T_f(i)\}$, $\{\phi(i,j)\}$, T_f
Objective:

$$Minimize :\ T_f \tag{48}$$

Constraint:

$$T_s(i) = 0,\ \forall i \in S \tag{49}$$

$$T_s(i) \geq \tau(j,i)\big[T_s(j) + \phi(j,i) \cdot z(j,i)T_{cm}\big],\ \forall i \notin S \tag{50}$$

$$T_f(i) = T_s(i) + k_i N^2 w(i) T_{cp},\ \forall i \in G(V,E) \tag{51}$$

$$2N^2 - \sum_{j \in G(V,E)} \phi(i,j) = 0,\ \forall i \in S \tag{52}$$

$$\sum_{j \in G(V,E)} \phi(j,i) - \sum_{j' \in G(V,E)} \phi(i,j') = 2k_i N,\ \forall i \notin S \tag{53}$$

$$\phi(i,j) \geq 0,\ \forall\ \tau(i,j) = 1 \tag{54}$$

$$\phi(i,j) = 0,\ \forall\ \tau(i,j) = 0 \tag{55}$$

$$k_i \in \mathbb{Z}_+,\ \forall i \in G(V,E) \tag{56}$$

$$k_i = 0, \forall i \in S \tag{57}$$

$$\sum_{i=1}^{p} k_i = N \tag{58}$$

$$T_f \geq T_f(i), \forall i \in G(V,E) \tag{59}$$

Remarks: As constraint (56) shows, $\{k_i\}$ are positive integers, which make *MFT-LBP* problem a *Mixed Integer Non-linear Programming* problem. To solve it, we propose an algorithm called *Phased Minimization of Finish Time with Layer Based Partition*(PMFT-LBP).

5.3 PMFT-LBP

PMFT-LBP: The *PMFT-LBP* algorithm contains the following three phases. In Phase I, it relaxes integers $\{k_i\}$ to real numbers and solves the relaxed linear programming problem. In Phase II, based on the optimal real number solution obtained in Phase I, *PMFT-LBP* determines a feasible integer solution for the original *PMFT-LBP* problem, which is close to the optimal real number solution. In Phase III, starting from the feasible integer solution obtained in Phase II, *PMFT-LBP* conducts a "neighbor search" and seeks for local optimal feasible integer solution.

Phase I. In this phase, the *PMFT-LBP* algorithm relaxes condition (56) to be a positive real number

$$k_i \geq 0,\ \forall i \in G(V,E) \tag{60}$$

and solves a relaxed version of the *MFT-LBP* problem, called *MFT-LBP-relax*.

With this relaxation, all constraints are either linear equality or linear inequality, forming a convex polygon feasible region. Its objective function is also linear defined on this polyhedron. Hence, *MFT-LBP* is a LP problem and can be solved to get the optimal real number solution $Q^* = \{\{k_i\}, \{T_s(i)\}, \{T_f(i)\}, \{\phi(i,j)\}, T_f\}$.

Phase II. In this phase, *PMFT-LBP* calls an algorithm called *finds an integer feasible solution(FIFS)* based on the optimal real number solution Q^* obtained from phase I. we firstly round off each k_i to its closest integer. The intuition

Algorithm 1 PMFT-LBP

function PMFT-LBP()
Phase I:
    solve *MFT-LBP-relax*, get optimal real number solution $\{\{k_i\}, \{T_f(i)\}, \{T_s(i)\}, \{\phi(i,j)\}, T_f\}$.
Phase II:
    call *FIFS* to get $\{k'_i\}$, a feasible integer schedule.
Phase III:
    use current schedule as start point: $p_{cur} \Leftarrow \{k'_i\}$
    re-solve LP for p_{cur}.
    while true **do**
        choose node a with $T'_f(a) = \max\{T'_f(i)\}$
        $k''_a \leftarrow k'_a - 1$
        choose node b with $T'_f(b) = \min\{T'_f(i)\}$
        $k''_b \leftarrow k'_b + 1$
        construct $\{k''_i\}$ using k''_a, k''_b replacing k'_a, k'_b:
        $\{k''_i\} \leftarrow \{k'_1, k'_2, ...k''_a, ..., k''_b, ...k'_p\}$
        get neighbor: $p_{nb} \Leftarrow \{k''_i\}$
        re-solve LP for p_{nb}.
        if $T'_f\langle p_{cur}\rangle < T'_f\langle p_{nb}\rangle$ **then** break
        else $p_{cur} \leftarrow p_{nb}$
        end if
    end while
    return optimal schedule p_{cur}.
end function

here is that if the real number optimal solution assigns the larger portion of a column to one processor, then assigning that processor with the entire column will have a higher chance to get "closer" to optimality. It is vice versa that if the processor is assigned with the smaller portion of a column, we shall remove its share of this column.

However, the rounding off process alone may not guarantee a feasible integer solution, because it may result in the sum of all k_i fails to equal the multiplier matrix's side length N, constraint (58). To resolve this problem, we conduct a subtle adjustment. For cases in which the sum is greater than N, meaning that there exists duplicate assignments, we shall reduce work load from some processors. Since the processor with the longest finishing time is the bottleneck affecting the overall finishing time, it has the highest priority to reduce its share of loads. On the contrary, for cases in which the sum is less than N, meaning that some rows/columns haven't been assigned to any processor, processor currently with the shortest running time has the highest priority to take up the responsibility.

We don't make the adjustment to one processor all at once. Instead, we conduct the adjustment iteratively. Every iteration we only adjust one row/column, then we update each processor's $\{T_s(i)\}$, $\{T_f(i)\}$, $\{\phi(i,j)\}$ to determine the processor to conduct adjustment on for the next round of iteration. The processor with the longest processing time currently will be the one to remove a row/column from in the next round of iteration, and the processor with the

Algorithm 2 FIFS Algorithm

function FIFS ($\{k_i\}$, $\{T_s(i)\}$, $\{T_f(i)\}$, $\{\phi(i,j)\}$, T_f)
  for each $k_i \in \{k_i\}$ **do**
    $k'_i \leftarrow round(k_i)$
  end for
  $Sum \leftarrow \sum_i \{k'_i\}$
  while $Sum \neq N$ **do**
    use $\{k'_i\}$ as known variables, re-solve *MFT-LBP* problem, get updated $\{T'_s(i)\}$, $\{T'_f(i)\}$, $\{\phi'(i,j)\}$.
    if $Sum > N$ **then**
      choose node j with $T'_f(j) = \max\{T'_f(i)\}$
      $k'_j \leftarrow k'_j - 1$
      $Sum \leftarrow Sum - 1$
    end if
    if $Sum < N$ **then**
      choose node j with $T'_f(j) = \min\{T'_f(i)\}$
      $k'_j \leftarrow k'_j + 1$
      $Sum \leftarrow Sum + 1$
    end if
  end while
  return integer schedule $\{k'_i\}$.
end function

shortest processing time will be the one to take the extra load in the next iteration.

When the sum of all k_i equals N, we finally find an integer feasible solution $\{k'_i\}$, $\{T'_s(i)\}$, $\{T'_f(i)\}$, $\{\phi'(i,j)\}$.

Phase III. In this phase, *PMFT-LBP* conducts a so-called "Neighbor Search" process to optimize the feasible solution obtained in phase II. The reason why this feasible solution still need optimization is that the "Rounding Off" procedure in phase II might bring about bias and take our feasible solution "away" from optimal. Therefore, a further optimization is necessary.

The "Neighbor Search" runs in iterations. For the first iteration, it starts from the feasible integer schedule $\{k'_i\} = \{k'_1, k'_2, ...k'_a, ..., k'_b, ...k'_p\}$ obtained in phase II, and compares the current overall finishing time with its neighbors'. $\{k''_i\} = \{k''_1, k''_2, ...k''_a, ..., k''_b, ...k''_p\}$ is defined as one of $\{k'_i\}$'s neighbors if all $k''_i = k'_i$ except for only two dimensions k''_a and k''_b, where $k''_a = k'_a - 1$ and $k''_b = k'_b + 1$, such that $\sum_{i=1}^{p} k''_i = \sum_{i=1}^{p} k'_i = N$. If one neighbor $\{k''_i\}$ offers shorter overall finishing time than $\{k_i\}'$ and the rest of its neighbors, we change load schedule from $\{k'_i\}$ to $\{k''_i\}$. Then, "Neighbor Search" will use $\{k''_i\}$ as a new start point and begins a new iteration. If at some stage, no further optimization can be made towards minimizing overall finishing time, which means the schedule at that stage $\{k^*_i\}$ is a local minimal point that has the shortest overall finishing time among all of its neighbors, iteration stops and we say we find our optimal schedule.

Algorithm 3 MFT-LBP-heuristic

function MFT-LBP-HEURISTIC ()
  solve *MFT-LBP-relax*, get optimal real number solution $\{k_i\}$, $\{T_f(i)\}$.
  $\{k_i'\} \leftarrow round(\{k_i\})$
  use $\{k_i'\}$ as known variables, re-solve *MFT-LBP* problem, get updated $\{T_f'(i)\}$.
  $Sum \leftarrow \sum_i \{k_i'\}$
  $diff \leftarrow Sum - N$
  if $diff < 0$ **then**
    sort $\{T_f'(i)\}$ in ascending order
    corresponding $k_i \leftarrow k_i + 1$ till $diff = 0$.
  else sort $\{T_f'(i)\}$ in descending order
    corresponding $k_i \leftarrow k_i - 1$ till $diff = 0$.
  end if
end function

5.4 MFT-LBP-heuristic

Considering the high time complexity of *PMFT-LBP* algorithm because so many LP-based updates are involved, we propose a heuristic called *MFT-LBP-heuristic*. This heuristic calculates a good feasible integer solution that is close to the optimal solution with a time complexity which is substantially reduced from *PMFT-LBP*. The basic idea is that whether one single row/column should be assigned to one processor or another simply doesn't bring about much improvement of the result. However, simplify these steps saves time complexity substantially.

Based on this idea, *MFT-LBP-heuristic* only keeps phase I the same with *PMFT-LBP* algorithm. In phase II, after obtaining the optimal real-number schedule $\{k_i\}$, the heuristic rounds off each k_i to get $\{k_i'\}$ slightly differently. The difference is, if the sum of $\{k_i'\}$ doesn't equal N, the heuristic sorts all processors in ascending order of their finishing time $\{T_f(i)'\}$, which forms an array $Arr[p]$. If the sum of $\{k_i'\}$ is less than N, then from the first element in $Arr[p]$, the heuristic keeps adding 1 to each processor's k_i' until $\sum_i \{k_i'\} = N$. Otherwise if the sum of $\{k_i'\}$ is greater than N, the heuristic starts from the last element of $Arr[p]$ and minus 1 from each processor's k_i' one by one until $\sum_i \{k_i'\} = N$. The adjusting process continues in a circular manner so if it reaches one end of $Arr[p]$, it jumps to the the other end of the array and continue the process again.

In phase III, *PMFT-LBP* searches each neighbor of current schedule $\{k_i'\}$. The time complexity of searching process is $O(p^2)$, where p is the number of processors. Once p is big, the scalability of the algorithm is poor. In order to speed up the local search process, we apply the thought of 'gradient descent' here. At each iteration, we only look at the neighbor $\{k_i''\}$ that has the highest chance to decrease the overall finishing time from current schedule $\{k_i'\}$. We know that $\{k_i''\}$ differentiate from $\{k_i'\}$ by just two dimensions $k_a'' = k_a' - 1$ and $k_b'' = k_b' + 1$. If k_a' is the schedule of the processor that currently takes the longest processing time, while k_b' is the schedule from the processor that takes the shortest, $\{k_i''\}$ then stands the highest the chance to be the neighbor

that has shortest overall finishing time. We compare $\{k_i''\}$'s T_f with $\{k_i'\}$'s. If $\{k_i''\}$'s is shorter, we use $\{k_i''\}$ in the next iteration. Otherwise, since even $\{k_i''\}$ cannot further decrease the overall finishing time, the other neighbors probably cannot either. Therefore we take $\{k_i'\}$ as the optimal schedule we look for.

This heuristic only solves LP problems twice and reduces time complexity of the iterative LP-based update process in *PMFT-LBP*. The result of this may sacrifice the overall finishing time of the mesh network a little bit, but reduce the algorithm's time complexity substantially by reducing a lot of time-consuming LP iterative updating processes.

As will be seen in next section, the performance of our heuristic is extremely close, and even in cases equal to *PMFT-LBP* both in terms of communication volume and finish time.

6 Performance Evaluation

6.1 Performance Evaluation of Star Network

In this subsection, we study the performance of our *layer based partition* scheme in a star network, which is an instance of *single neighbor network.* In each iteration, we randomly generate two $N * N$ matrices, and a heterogeneous star network of processors to conduct the matrix multiplication.

Star Network. The star network contains one source and multiple children connected to the source. As mentioned previously, we assume the source node only transmits load, but does not take part in computing. In our simulation, we use a star network containing 16 children, where each child's unit processing time $wTcp$ is uniformly distributed in the range of (0.0002, 0.0007), and each link's unit transmission time $zTcm$ is uniformly distributed in the range of (0.0003, 0.0008).

Communication and Processing mode. In order to better compare our algorithm with some other related algorithms, we use PCCS mode as the communication and processing mode here, meaning the source can communicate with each processor in a parallel manner whereas communication cannot overlap with computation.

Matrices. The side length of the matrices in our simulation goes from 100 to 1000. When analyzing the performance of each algorithm over matrix size, each data point is an average of 10 independent experiments over 10 independently different star network.

6.1.1 Evaluation Metrics

Total Communication Volume. Total communication volume is defined as the sum of data volume transmitted on each link.

Task Finishing Time. The task finishing time in star network is defined as the time period from the source starts to send data till the last processor finishes working.

6.1.2 Comparison Algorithms

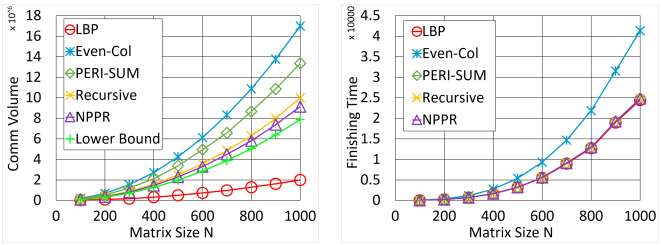
We compare our *layer based partition* algorithm to a couple of typical *rectangular partition* algorithms.
Even-Col. Even-Col is a naive *rectangular partition* algorithm that simply partitions the matrix into equivalent columns.
PERI-SUM. Beaumont *et al.* [26] deal with the geometric problem of partition the unit square into p rectangles of given area s_1, s_2, s_3, etc, so as to minimize the the sum of the perimeters(PERI-SUM) of the p rectangles, which is proportional to communication volume. Beaumont *et al.* further propose the communication lower bound, and introduce a 1.75-approximation algorithm. We use it in our comparison.
Recursive. Nagamochi *et al.* [29] introduce a recursive partitioning technique on the basis of **PERI-SUM**, and improve the approximation ratio from 1.75 to 1.25.
NRRP. Beaumont *et al.* [30] combine the idea of non-*rectangular partition*ing from DeFlumere[28] and recursive partitioning algorithm proposed by Nagamochi[29]. The combination of these two ingredients lead to an improvement of the approximation ratio down to $\frac{2}{\sqrt{3}} \simeq 1.15$.
Lower Bound. Ballard *et al.* [25] proposed the lower bound of communication volume in *rectangular partition* to be $2\sum_{i=1}^{p}(\sqrt{s_i})$. We use this lower bound to compare with our LBP's communication volume.

6.1.3 Evaluation Result on Star Network

Communication Volume with Increasing Matrix Size. As one of our biggest contributions, the simulation displays overwhelming superiority of our *layer based partition* algorithm over *rectangular partition* algorithms in total communication volume. Figure 6(a) compares the total communication volume of each algorithm. While the total communication volume generally increases along with the expansion of matrix size N, LBP generates the smallest total communication volume. Specifically, when the matrix size reaches 1000, the total communication volume of *layer based partition* reduces 75% from the lower bound of the *rectangular partition*. Meanwhile, as stated in [30], the lower bound of *rectangular partition* is too optimistic to reach in actual practice, and the real best *rectangular partition* result is obtained actually in NPPR[30], Recursive[29] and PERI-SUM[26]. Genearally, LBP presents a total communication volume that reduces 78% from NPPR, 79.7% from Recursive, and 85.1% from PERI-SUM, respectively. We observe that LBP keeps this ratio over *rectangular partition* algorithms along with the increase of matrix side length. Moreover, when the star network gets larger, this ratio gain gets

even bigger. We attribute these to the advantage of *layer based partition* over *rectangular partition* in communication volume.

Finishing Time with Increasing Matrix Size. Figure 6(b) shows the overall finishing time of each algorithm. We observe that while all algorithm's finishing time increase with matrix size, LBP, PERI-SUM, Recursive, NPPR present similar curves, with their overall finishing time much smaller than that of Even-Col. Specifically, when the matrix size is 1000, the overall finishing time of those four algorithm is about 40% smaller than that of Even-Col. One reason for this is that those four algorithms, LBP, PERI-SUM, Recursive, NPPR all achieve load balance when scheduling the load. No matter dividing the matrix into layers or rectangles, each of the four algorithms makes sure that each share of load is proportional to that processor's computing ability.



(a) Communication Volume (b) Finishing Time

Fig. 6 Performance comparison on 16-node star network

Summary. In this subsection we compare our *layer based partition* algorithm with *rectangular partition* algorithms in terms of total communication volume and task finishing time. The simulation results proves that our *layer based partition* algorithm generates a total communication volume that is substantially reduced from the state-of-the-art *rectangular partition* algorithms. In the meanwhile, LBP also reaches load balance and generates a total overall finishing time that is as low as the other algorithms.

6.2 Performance Evaluation of Mesh

In this subsection, we study how the *layer based partition* scheme performs in mesh networks. In each single run of the simulation, we randomly generate a heterogeneous mesh network, and two square matrices conducting multiplication. Then our LBP algorithm and the other comparing algorithms are called to schedule the matrix multiplication load on the given mesh network.

Mesh Network. The mesh network is heterogeneous, with each link speed and processor speed independently generated. The unit processing time $wTcp$ of the processors is uniformly distributed in the range of $(0.0002, 0.0007)$, while the unit transmission time $zTcm$ of the links is uniformly distributed in the range of $(0.0003, 0.0008)$. In our simulation, we use use three square meshes,

which are of dimension 5*5, 7*7 and 9*9. For model simplicity and without loss of generality, we focus on studying the case for one quadrant-the lower right one-in figure 5, and the source node is located at the top left corner. The cases of the other three quadrants are similar. However, SUMMA is an exception, in which no single source exists, and each processor in the mesh takes one block of matrix data. So when evaluating the performance of SUMMA, we divide the matrix data into blocks and store it on corresponding processor.

Matrices. The matrices we analyze are large scale dense matrices. In our simulation, we randomly generate matrices with their side length N ranging from 1000 to 2000.

When analyzing the performance of each algorithm over matrix size, each data point in our simulation is an average of 10 independent experiments over 10 independently different mesh network.

6.2.1 Evaluation Metrics

Overall Communication Volume. Overall communication volume is defined as the sum of data volume transmitted on each link. Compared to the total data volume coming out of the source, overall communication volume provides a more direct view of data running on each link.

Task Finishing Time. The task finishing time in mesh network is defined as the time period from the source starts to send data till the last processor finishes working.

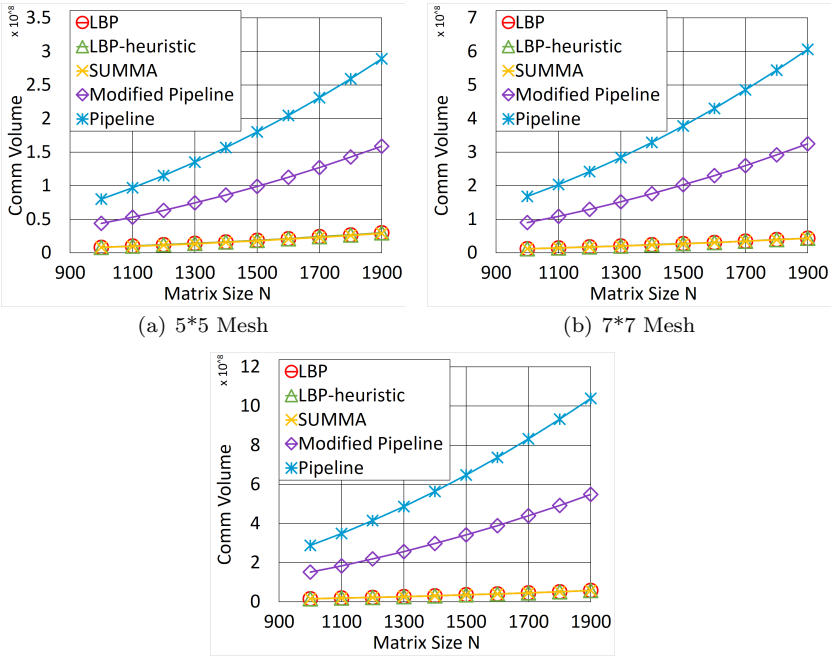
Total Number of Iterations to Solve LP. Since we rely on simplex algorithm to solve LP in our LBP and LBP-heuristic, we evaluate both algorithms' efficiency in terms of average total number of iterations taken by simplex algorithm to solve LP.

6.2.2 Comparison Algorithms

SUMMA. Geijn *et al.* [5] propose SUMMA, which is the most widely applied parallel matrix multiplication scheme on homogeneous grid. The algorithm allocates matrix blocks over the grids. In each step, the pivot column of blocks is communicated horizontally and the pivot row of blocks is communicated vertically. Each processor uses the pivot blocks it get to update its rectangle in each step. We apply this algorithm on our heterogeneous mesh network.

Pipeline. The Pipeline algorithm is a classic scheduling method. Starting from the source, each node forwards the entire copy of data along the grids to each of its neighbor in the mesh network. Duplicate copy may be sent to one node, however, it only keep the first received one. Once that node finishes receiving its first copy, it starts processing the data while forwarding. The whole system acts like a pipeline with communication overlaps with computing.

Modified Pipeline. Tan *et al.* [35] propose an improved pipeline broadcast scheme to achieve performance enhancement for distributed matrix multiplication. The non-blocking pipeline scheme takes advantage of tuned chunk size to



(a) 5*5 Mesh (b) 7*7 Mesh

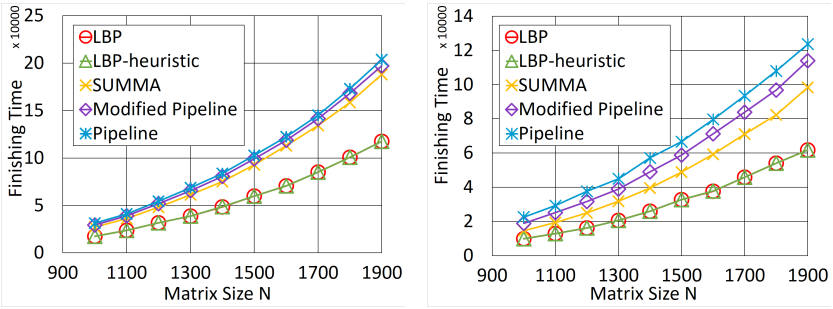
(c) 9*9 Mesh

Fig. 7 Communication volume comparison with increasing matrix size and network dimension.

boost communication performance. We apply the idea to heterogeneous mesh network.

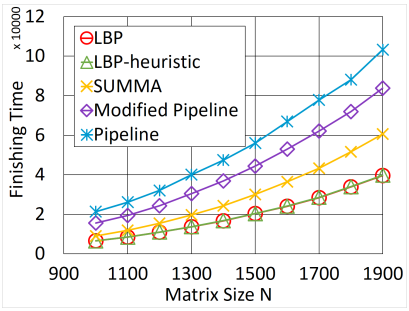
6.2.3 Evaluation Result on Mesh

Overall Communication Volume. Figure 7 displays the overall communication volume when conducting matrix multiplication of two $N * N$ matrices in (a) 5*5 mesh, (b) 7*7 mesh, and (c) 9*9 mesh respectively. The simulation result reveals that while all algorithms's overall communication volume goes up as matrix size increases, SUMMA, LBP and LBP-heuristic generate almost the same smallest overall communication volume, which is 81% smaller than that of Modified Pipeline and 90% smaller than that of Pipeline. SUMMA is well-known to be communication-optimal on homogeneous mesh network. Though applying SUMMA on heterogeneous mesh may affect its overall finishing time due to the change of processor speed and link speed, its data transmission pattern won't be affected. So SUMMA is still communication-optimal on heterogeneous mesh. LBP and LBP-heuristic generates almost the same communication volume as SUMMA, consequently, implies that LBP and LBP-heuristic are at least close to communication optimal on heterogeneous mesh. Moreover, we observe that LBP, LBP-heuristic, SUMMA are close to



(a) 5*5 Mesh

(b) 7*7 Mesh



(c) 9*9 Mesh

Fig. 8 Finishing time comparison with increasing matrix size and network dimension.

each other as network dimension increases, but their difference ratio with the other algorithms are getting larger and larger.

Task Finishing Time. Figure 8 shows the task finishing time of each algorithm on (a) 5*5 mesh, (b) 7*7 mesh, and (c) 9*9 mesh. Generally, LBP generates the smallest task finishing time than the rest of algorithms. LBP-heuristic gives a task finishing time that is slightly longer than that of LBP, which are 0.03% more in 5*5 mesh, 0.08% more in 7*7 mesh, and 0.18% more in 9*9 mesh, respectively. This tiny difference can entirely be ignored. SUMMA, since it can no longer reach load balance with link speed and processor speed vary, its task finishing time are, respectively, 56.4% more in 5*5 mesh, 52.9% more in 7*7 mesh, and 46.7% more in 9*9 mesh, than that of LBP. Moreover, Modified Pipeline are respectively 66.7% more in 5*5 mesh, 87.2% more in 7*7 mesh, and 121.1% more in 9*9 mesh. Pipeline are respectively 73.4% more in 5*5 mesh, 114% more in 7*7 mesh, and 185% more in 9*9 mesh. All in all, LBP and LBP-heuristic present the best performance in terms of task finishing time.

Total Number of Iterations to Solve LP. As mentioned previously, we use the simplex algorithm to solve LP in our LBP and LBP-heuristic algorithm. Each time solving the LP costs a certain number of iterations by the simplex algorithm. And according to our algorithm, we may re-solve LP a couple of times due to 1.find real number solution 2. find integer solution 3. local search, etc. Therefore, the total number of iterations to solve LP is a good indication

of the efficiency of our algorithms. Figure 9 counts the average total number of iterations in solving LP by LBP and LBP-heuristic on 5*5 mesh, 7*7 mesh, 9*9 mesh. Each point is an average of 10 identical independent experiments. The solid lines represent LBP whereas the dashed lines represent LBP-heuristic. We have the following observations: 1. The solid lines vary dramatically due to the uncertain number of times to re-solve LP by LBP, while the dashed lines are comparatively stable, because LBP-heuristic only re-solves LP either two or three times. 2. The total number iterations show no correlation with respect to matrix size, a good evidence indicating that both algorithms are suitable for large scale matrix scheduling. 3. The total number of iterations does show positive correlation with respect to mesh size. 4. For the same mesh size, dashed lines are generally far below solid line, which indicates that LBP-heuristic generally requires much less total number of iterations to find a solution than LBP. In other words, LBP-heuristic is more efficient.

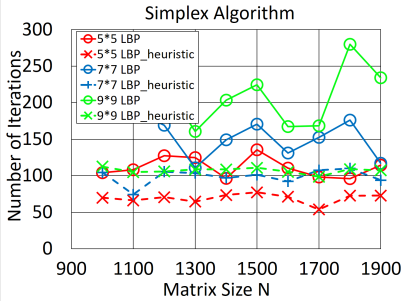


Fig. 9 Number of iterations by the simplex method in solving LP.

Summary. LBP-heuristic is significantly more efficient than LBP while maintaining almost equally good performance, which makes it widely applicable. Additionally, both algorithms outperforms the other heterogeneous mesh scheduling algorithms substantially.

7 Conclusion

In this paper, we focus on the problem of scheduling matrix multiplication on heterogeneous processor platforms. We first address two drawbacks of traditional *rectangular partition*: 1. difficult to determine partition shapes. 2. communication volume is not optimal. Alternatively, we present a novel scheduling method: *layer based partition*. We demonstrate that *layer based partition* scheme is easy to find a partition, and generates a total communication volume that is optimal, smaller than the lower bound of *rectangular partition*. In the following part, we study how to minimize task finishing time using LBP. In *single-neighbor network*, we propose an equality based theory. In *multi-neighbor network*, we formulate the problem as a Mix Integer Programming

problem, which we provide a 3-Phase algorithm to solve. Considering the high time complexity, a heuristic algorithm is also proposed.

Simulation results show that *layer based partition* outperforms the other comparing algorithms both in single and multiple neighbor networks. It generates a total communication volume that is substantially reduced from the state-of-the-art *rectangular partition* algorithms, and maintain load balancing so that its task finishing time is minimized as well. Results also shows that in mesh network, LBP-heuristic achieves close, and even equal performance to *layer based partition*, with its total number of iterations significantly reduced. Hence, we believe LBP-heuristic is very applicable in real world practice.

References

1. G. L. Zeng, *Medical Image Reconstruction A Conceptual Tutorial.* Springer, 2009.
2. R. G. Lyons, *Understanding digital signal processing.* Prentice Hall, 2010.
3. L. Canon, "A cellular computer to implement the kalman filter algorithm," in *Ph.D. dissertation of Montana State University*, 1969.
4. G. Fox, S. Otto, and A. Hey, "Matrix algorithms on a hypercube i: Matrix multiplication," *Parallel Computing*, vol. 4, no. 1, pp. 17–31, 1987.
5. R. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency - Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1994.
6. E. Solomonik, J. Demmel, "Communication-optimal Parallel 2.5 D Matrix Multiplication and LU factorization Algorithms ," in *Euro-Par 2011 Parallel Processing*, Springer, 2011, pp. 90–109.
7. D. Malysiak, and T. Kopinski, "A generic and adaptive approach for workload distribution in multi-tier cluster systems with an application to distributed matrix multiplication," in *Computational Intelligence and Informatics (CINTI), 2015 16th IEEE International Symposium on*, IEEE, Nov 2015, pp. 255–266.
8. A. N. Yzelman, and D. Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix-Vector Multiplication," *IEEE Transactions on Parallel and Distribute Systems*, vol. 25, no. 1, pp. 116–125, 2014.
9. P. Koanantakool, A. Azad, A. Bulu? D. Morozov, S. Oh, L. Oliker, K. Yelick, "Communication-Avoiding Parallel Sparse-Dense Matrix-Matrix Multiplication," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*, IEEE, May 2016, pp. 842–853.
10. B. S. Samantray, D. Kanhar, " Implementation of dense matrix multiplication on 2D mesh," in *High Performance Computing and Applications (ICHPCA), 2014 International Conference on*, IEEE, Dec 2014, pp. 1–5.
11. S. Mittal, and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, 2015.
12. W. W. Hwu, "Heterogeneous System Architecture: A New Compute Platform Infrastructure" *Morgan Kaufmann*, Dec 2015.
13. A. Kalinov, "Scalability analysis of matrix-matrix multiplication on heterogeneous clusters," in *Parallel and Distributed Computing, International Symposium on*, ser. ISDPC '04. IEEE Computer Society, Jul 2004, pp. 303–309.
14. J. Quintin, K. Hasanov, and A. Lastovetsky, "Hierarchical parallel matrix multiplication on large-scale distributed memory platforms," in *IEEE International Conference on Parallel Processing*, ser. ICPP '13. IEEE, Jun 2013, pp. 754–762.
15. Y. Ohtaki, D. Takahashi, T. Boku, and M. Sato, "Parallel implementation of strassens matrix multiplication algorithm for heterogeneous clusters," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International.* IEEE, Apr 2004.
16. P. Alonso, R. Reddy, and A. Lastovetsky, "Experimental study of six different implementations of parallel matrix multiplication on heterogeneous computational clusters of multicore processors," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on.* IEEE, Feb 2010, pp. 263–270.

17. T. Malik, V. Rychkov, A. Lastovetsky, and J. Quintin, "Topology-aware optimization of communications for parallel matrix multiplication on hierarchical heterogeneous hpc platforms," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, May 2014, pp. 39–47.
18. Z. Zhong and A. Lastovetsky, "Data partitioning on multicore and multi-gpu platforms using functional performance models," *Computers, IEEE Transactions on*, vol. 64, no. 9, pp. 2506–2518, Dec 2014.
19. J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, "Communication-optimal parallel recursive rectangular matrix multiplication," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed*, ser. IPDPS '13. IEEE Computer Society, 2013, pp. 261–272.
20. O. Beaumont, L. E. Dubois, A. Guermouche, T. Lambert, " Comparison of Static and Runtime Resource Allocation Strategies for Matrix Multiplication," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*, IEEE, Oct 2015, pp. 170–177.
21. J. Yang, X. Meng, M. W. Mahoney, "Implementing Randomized Matrix Algorithms in Parallel and Distributed Environments," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 58–92, Dec 2016.
22. A. Lastovetsky, " On Grid-based Matrix Partitioning for Heterogeneous Processors," in *Parallel and Distributed Computing Sixth International Symposium on*, ser. ISPDC '07. IEEE, Jul 2007, pp. 51–58.
23. A. Lastovetsky, R. Reddy" Two-dimensional Matrix Partitioning for Parallel Computing on Heterogeneous Processors Based on their Functional Performance Models ," in *Euro-Par 2009 ?Parallel Processing Workshops*, Springer, 2010, pp. 91–101.
24. D. Clarke, A. Lastovetsky, V. Rychkov" Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models ," in *Euro-Par 2011 ?Parallel Processing Workshops*, Springer, 2012, pp. 450–459.
25. G. Ballard, J. Demmel, and A. Gearhart, "Communication bounds for heterogeneous architectures," in *23rd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA 2011. ACM, Feb 2011.
26. O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix multiplication on heterogeneous platforms," *IEEE Transactions on Parallel and Distribute Systems*, vol. 12, no. 10, pp. 1033–1051, 2001.
27. A. DeFlumere and A. Lastovetsky, "Searching for the optimal data partitioning shape for parallel matrix matrix multiplication on 3 heterogeneous processors," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, May 2014, pp. 17–28.
28. A. DeFlumere, "Partitioning for parallel matrix-matrix multiplication with heterogeneous processors: The optimal solution," in *IEEE International Parallel and Distributed Processing Symposium Workshops and PhD Forum*. IEEE, May 2012, pp. 125–139.
29. H. Nagamochi and Y. Abe, "An approximation algorithm for dissecting a rectangle into rectangles with specified areas ," *Discrete Applied Mathematics*, vol. 155, no. 4, pp. 523–537, 2007.
30. O. Beaumont, L. E. Dubois, A. Guermouche, T. Lambert, " A New Approximation Algorithm for Matrix Partitioning in Presence of Strongly Heterogeneous Processors," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, May 2016, pp. 474–483.
31. A. Füenschuh, K. Junosza-Szaniawski, and Z. Lonc, "Exact and approximation algorithms for a soft rectangle packing problem ," *Optimization*, vol. 63, no. 11, pp. 1637–1663, 2014.
32. R. Korf, "Multi-Way Number Partitioning," *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, Jul 2009.
33. V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed System*. IEEE Computer Society, 1996.
34. Z. Zhang and T. G. Robertazzi, "Scheduling divisible loads in gaussian, mesh and torus network of processors," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3249–3264, Jan 2015.

35. L. Tan, "Improving performance and energy efficiency of matrix multiplication via pipeline broadcast," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, IEEE, Sep 2013, pp. 1–5.