

Research Repository UCD

Title	Theoretical Results on Optimal Partitoning for Matrix-Matrix Multiplication with Two Processors
Authors(s)	DeFlumere, Ashley, Lastovetsky, Alexey
Publication date	2011-09
Publication information	DeFlumere, Ashley, and Alexey Lastovetsky. Theoretical Results on Optimal Partitoning for Matrix-Matrix Multiplication with Two Processors. University College Dublin. School of Computer Science and Informatics, September, 2011.
Series	UCD CSI Technical Reports, ucd-csi-2011-09
Publisher	University College Dublin. School of Computer Science and Informatics
Item record/more information	http://hdl.handle.net/10197/12402

Downloaded 2024-03-28T04:02:09Z

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)



© Some rights reserved. For more information

Theoretical Results on Optimal Partitoning for Matrix-Matrix Multiplication with Two Processors

Ashley DeFlumere and Alexey Lastovetsky
School of Computer Science and Informatics
University College Dublin
ashley.deflumere@ucdconnect.ie, alexey.lastovetsky@ucd.ie

University College Dublin
Technical Report UCD-CSI-2011-09
September 2011

Abstract

In this report, we consider a simple but important linear algebra kernel, matrix-matrix multiplication. Building multi-core processors based on heterogeneous cores is an important current trend. In this context, it is of great interest to study optimal matrix partitioning algorithms for small cases (*i.e.* small number of cores). Indeed, the general case, with relatively high numbers of heterogeneous resources is now well understood, however the problem is in general NP-Complete when one aims at balancing the load while minimizing the communications. Nonetheless several approximation algorithms have been successfully designed. Nevertheless, negative complexity results do not apply for very few heterogeneous cores. Additionally, the case of a small number of processors is useful as a model for heterogeneous clusters and clusters of clusters. In this paper, we provide a complete study of 2 heterogeneous resources and we prove that in this case, the optimal partitioning is based on non-standard decomposition techniques.

I. Introduction

Matrix-Matrix Multiplication (MMM) is a simple but important linear algebra kernel used on multiple processors. Our goal is to minimize the execution time of MMM, which consists of two parts; the time taken to send the necessary sections of matrices between processors, i.e., the communication time, and the time taken to calculate the result matrix, i.e., the computation time. To minimize the execution time, we focus on minimizing the communication time, and exploring ways to complete the communication and computation in parallel, overlapping these two necessary parts to lower the total time. We look specifically at the case of two processors of differing computation speeds; while important in its own right, this is also a reasonable model for two clusters of processors. However, we also look to extend these results to three processors and beyond. Though we focus on heterogeneous processors with differing computation speeds, including ratios of 1:1 computation speeds also covers homogenous processors.

Traditionally data partitions made entirely of rectangles are thought to be the most efficient[1][2][3][4]. Our approach is to question whether a partitioning scheme made entirely of rectangles is optimal. Is it possible to create an arbitrary partition, meaning product elements mapped to processors in no recognizable shape, which will surpass the efficiency of the rectangular partition? Are there shapes other than rectangles that might provide an optimal solution? In this paper we do not wish to show that a given partition is

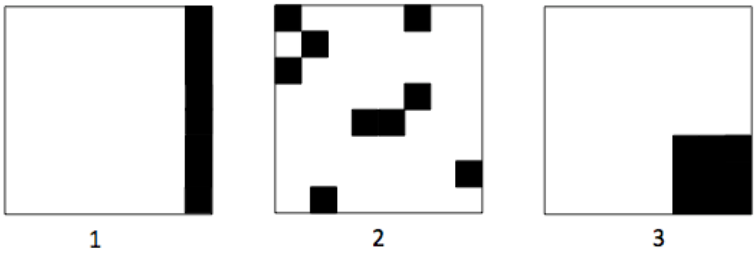


Fig. 1. 1. A Straight-Line partition 2. An arbitrary partition 3. A Square-Corner partition

simply better than the traditional rectangular approach, but that certain partitions are optimal, better than any other approach, given a particular algorithm used to compute the MMM.

We focus specifically on five different algorithms to accomplish MMM on two processors. The first two algorithms involve putting a barrier between communication and computation. Once the communication has completed, and both processors have all data required to compute their assigned portions of the result matrix, the computation proceeds. These algorithms are called Serial Communication with Barrier (SCB), and Parallel Communication with Barrier (PCB).

The final three algorithms involve methods of overlapping communication and computation. Serial Communication with Overlap (SCO) and Parallel Communication with Overlap (PCO) allow communication to occur while any subsections of the partition not requiring communication are computed. Finally, there is Parallel Interleaving Overlap (PIO), in which all communications and computations are overlapped, a single row and column at a time.

We will find for these five algorithms the optimal partition by considering all arbitrary partitions, meaning a partition with any possible shape. The rectangular partition is not optimal unconditionally and there is an alternative shape which is optimal under certain conditions, many of which are practically important when working with heterogeneous platforms.

For two processors, there is only one way in which to create an all rectangle partition. We call this scheme the Straight-Line approach. The matrices are divided into two rectangles with areas proportional to the processor's speed. Each processor requires data from the other in order to compute its portion of the matrix. Therefore the communication time is constant, every element of one matrix needing to be communicated.

With this in mind, if we could create a partition resulting in a lower communication time, it could outperform the Straight-Line, all rectangular, scheme. In this paper, we will begin with any arbitrary partition and alter it incrementally to lower its communication time. When we can no longer alter it to decrease communication volume, we will have created partition shapes that can not be improved. These shapes are candidates for the optimal shape. In [5], the non-rectangular Square-Corner partition was found to have a lower communication volume than Straight-Line partitions for processor computation power ratios greater than three when using the SCB algorithm. We will go further than this and show that not only is the Square-Corner better than Straight-Line in those cases, it is better than any other possible partition. We also examine the partitions using a variety of algorithms, as opposed to one.

In this report we do an exhaustive study of the problem of data partitioning for parallel MMM with two processors and find the optimal solution of this problem, which is different from the traditional one. We also develop original mathematical techniques in order to prove the optimality of the found solution.

II. Theoretical Results

In this section we will prove that the Square-Corner partitioning is optimal compared to all other MMM partitioning schemes for all processor power ratios when bulk overlapping communication and computation, and some processor power ratios when placing a barrier between them or using interleaving overlap.

A. Mathematical Model and Technique

Throughout, we will make several assumptions, as follows:

1) Matrices A, B and C are square, of size $N \times N$, and identically partitioned between processor P, represented in figures as white, and processor Q, represented in figures as black
2) Processor P computes faster than Processor Q by ratio, $r : 1$
3) The number of elements in the partition for Processor Q will be factorable, i.e. it will be possible to form a rectangle with them

B. MMM Computation

As the kij algorithm is well-known and used by such software as ScaLAPACK[6] to compute MMM, we will assume it is the manner of computation for all algorithms presented in this paper. The kij algorithm for MMM is a variant of the triply nested loop algorithms. The three for loops are nested and iterate over the line C[i, j] = A[i, k] * B[k, j] + C[i ,j]. The k variable represents a "pivot" row and column as shown in Fig.2. For each iteration of k, every element of the result matrix C is updated, incrementally obtaining the final value.

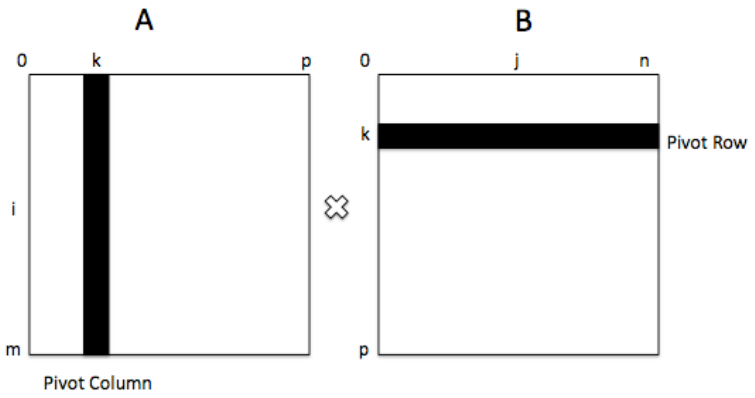


Fig. 2. Pivot row and column, k, of the kij algorithm. Every element of C is updated before k is moved to the next row and column.

An element in the pivot column k of matrix A, will be used to calculate every element in its row, i, in matrix C. Similarly, an element in the pivot row k of matrix B, will be used to calculate every element in its column, j, in matrix C. If the processor assigned to calculate these elements of matrix C has not also been assigned the element in the pivot column or row, that element must be communicated to the processor.

Here we define a partition to provide a basis for our performance models. Formally, each element of an $N \times N$ matrix is of the form (i, j) and a partition is a function, $\varphi(i, j)$, such that,

$$\varphi(i,j) = \begin{cases} 0 & \text{if } (i,j) \in P, \text{ the faster processor} \\ 1 & \text{if } (i,j) \in Q, \text{ the slower processor} \end{cases}$$

1) Partition Metrics: To measure a partitioning scheme, we use metrics of which parts of each matrix do *not* require communication. If a row of Matrix A, or a column of Matrix B, does not need to be communicated between the processors it is considered to be *clean*. A row or column is *dirty* if it contains elements belonging to both processors, and therefore requires communication. We need both the total number of clean rows and columns, and a way to determine if a given row or column is clean or dirty.

$$\begin{aligned}
&\| \varphi \|_x = \# \text{ of clean rows in } \varphi \\
&\| \varphi \|_y = \# \text{ of clean columns in } \varphi \\
&\| \varphi \|_x^0 = \# \| \varphi \|_x \text{ belonging to Processor P} \\
&\| \varphi \|_y^0 = \# \| \varphi \|_y \text{ belonging to Processor P} \\
&\| \varphi \|_x^1 = \# \| \varphi \|_x \text{ belonging to Processor Q} \\
&\| \varphi \|_y^1 = \# \| \varphi \|_y \text{ belonging to Processor Q} \\
&r(\varphi, i) = \begin{cases} 0 \text{ if } (i, \cdot) \text{ of } \varphi \text{ is clean} \\ 1 \text{ if } (i, \cdot) \text{ of } \varphi \text{ is dirty} \end{cases} \\
&c(\varphi, j) = \begin{cases} 0 \text{ if } (\cdot, j) \text{ of } \varphi \text{ is clean} \\ 1 \text{ if } (\cdot, j) \text{ of } \varphi \text{ is dirty} \end{cases}
\end{aligned}$$

C. The $push$ Algorithm

For each of our algorithms, SCB, PCB, SCO, PCO, and PIO, we will use the $push$ operations to start with any arbitrary partition, and end with a partition that has a regular shape. The $push$ algorithm, with $\uparrow, \downarrow, \leftarrow$, and $\rightarrow$ operations is described in full here.

1) $push \downarrow$ Algorithm: The $\downarrow$ operation creates a new partition from an existing one by cleaning a row, k, assigning elements $\in$ Q to the rows below. The reassigned elements are filled into the rows below in typewriter fashion, *i.e.* in the first available suitable slot from left to right and top to bottom. A slot is suitable if it is not in a clean column and is $\in$ P in the input partition φ.

Formally, $\downarrow (\varphi, k) = \varphi_1$ where,

Initialize $\varphi_1 \leftarrow \varphi$
$(g, h) = (k + 1, 1)$
for $j = 1 \rightarrow N$ **do**
  if $\varphi(k, j) == 1$ **then**
    $\varphi_1(k, j) = 0$; {If element was dirty, clean it}
    (g, h) = find(g, h);
    $\varphi_1(g, h) = 1$; {Put displaced element in new spot}
  end if
  $j \leftarrow j + 1$;
end for

find(g, h) {Look for a suitable slot to put element}
for $g \rightarrow N$ **do**

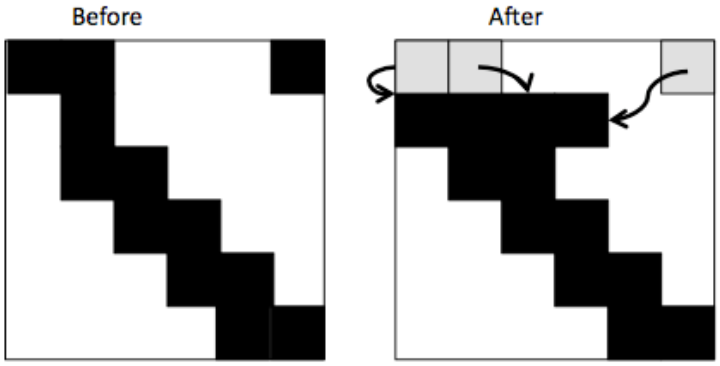


Fig. 3. An arbitrary partition, φ, between processor P (white) and processor Q (black), and partition, φ_1, showing how the elements of row $k = 1$ have been pushed by the $\downarrow$ operation

for $h \rightarrow N$ **do**
if $\varphi_1(g,h) == 0$ && $\varphi(g,h) == 0$ && $c(\varphi,h) == 1$ **then**
return (g,h);
end if
end for
$g \leftarrow g + 1$;
$h \leftarrow 1$;
end for
return $\varphi_1 = \varphi$ {No free slots, no more push possible in this direction}

It is important to note that if no suitable $\varphi(g,h)$ can be found for each element in the row being cleaned that requires rearrangement, then φ is considered fully condensed from the top and all further $\downarrow(\varphi,k) = \varphi$.

2) $push\uparrow$: The $\uparrow$ operation, similar to the $\downarrow$, creates a new partition, φ_1, by cleaning row k of partition φ by reassigning elements to the rows above row k. The algorithm is identical to the above for the $\downarrow$ operation, with the exception that possible slots for pushed elements are searched for in a left to right, bottom to top fashion.

Initialize $\varphi_1 \leftarrow \varphi$
$(g,h) = (k-1,1)$
for $j = 1 \rightarrow N$ **do**
if $\varphi(k,j) == 1$ **then**
$\varphi_1(k,j) = 0$; {If element was dirty, clean it}
(g,h) = find(g, h);
$\varphi_1(g,h) = 1$; {Put displaced element in new spot}
end if
$j \leftarrow j + 1$;
end for
find(g, h) {Look for a suitable slot to put element}
for $g \rightarrow 1$ **do**
for $h \rightarrow N$ **do**

if $\varphi_1(g,h) == 0$ && $\varphi(g,h) == 0$ && $c(\varphi,h) == 1$ **then**
return (g,h);
end if
end for
$g \leftarrow g-1$;
$h \leftarrow 1$;
end for
return $\varphi_1 = \varphi$ {No free slots, no more push possible in this direction}

3) push →: The → operation creates a new partition from an existing one by cleaning column, k, by reassigning elements ∈ Q to the columns to the right. It searches for a suitable slot from top to bottom, and then right to left. A slot is suitable if it is not in a clean row and is ∈ P in the partition ρ. Formally, $\rightarrow(\rho,k) = \rho_1$ where,

Initialize $\varphi_1 \leftarrow \varphi$
$(g,h) = (1,k+1)$
for $i = 1 \rightarrow N$ **do**
if $\varphi(i,k) == 1$ **then**
$\varphi_1(i,k) = 0$; {If element was dirty, clean it}
(g,h) = find(g, h);
$\varphi_1(g,h) = 1$; {Put displaced element in new spot}
end if
$i \leftarrow i+1$;
end for

find(g, h) {Look for a suitable slot to put element}
for $g \rightarrow N$ **do**
for $h \rightarrow N$ **do**
if $\varphi_1(g,h) == 0$ && $\varphi(g,h) == 0$ && $r(\varphi,h) == 1$ **then**
return (g,h);
end if
end for
$g \leftarrow 1$;
$h \leftarrow h+1$;
end for
return $\varphi_1 = \varphi$ {No free slots, no more push possible in this direction}

4) push ←: The ← operation, similar to the →, creates a new partition, ρ_1, by cleaning column k of partition ρ by reassigning elements to the columns to the left of column k. The algorithm is identical for the one above for →, with the exception that elements are placed in a top to bottom, right to left fashion.

Initialize $\varphi_1 \leftarrow \varphi$
$(g,h) = (1,k-1)$
for $i = 1 \rightarrow N$ **do**
if $\varphi(i,k) == 1$ **then**
$\varphi_1(i,k) = 0$; {If element was dirty, clean it}
(g,h) = find(g, h);
$\varphi_1(g,h) = 1$; {Put displaced element in new spot}
end if
$i \leftarrow i+1$;

end for
find(g, h) {Look for a suitable slot to put element}
for $g \rightarrow N$ **do**
  for $h \rightarrow 1$ **do**
    if $\varphi_1(g,h) == 0$ && $\varphi(g,h) == 0$ && $r(\varphi,h) == 1$ **then**
      return (g,h);
    end if
  end for
  $g \leftarrow 1$;
  $h \leftarrow h-1$;
end for
return $\varphi_1 = \varphi$ {No free slots, no more push possible in this direction}

We observe several axioms related to the $push$ algorithm.

Axiom 1: $\downarrow$ and $\uparrow$, create a clean white row, k, and may create at most one dirty row in φ_1 that was clean in φ. No more than one row can be made dirty, as a row that was clean will have enough suitable slots for all elements moved from the single row, k.

Axiom 2: $\downarrow$ and $\uparrow$ are defined to not create a dirty column, j, in φ_1 that was clean in φ. However, they may create additional clean column(s), if the row k being cleaned contains elements that are the only elements $\in Q$ in their column, and there are sufficient suitable slots in other columns.

Axiom 3: $\rightarrow$ and $\leftarrow$ create a clean column, k, and may create at most one dirty column in φ_1 that was clean in φ.

Axiom 4: $\rightarrow$ and $\leftarrow$ will never create a dirty row in φ_1 that was clean in φ, but may create additional clean rows.

III. Serial Communication with Barrier - SCB

In this algorithm communication is done serially, so Processor P first sends data to Processor Q. Once that has completed, communication from Processor Q to Processor P occurs. Finally, each processor computes its portion of the matrix in parallel. We model communication using the linear Hockey Model [7], and all computation is done with the kij algorithm. For all partitions,

$$Texecution = Tcomm + Tcomp \tag{1}$$

$$Tcomm = 2N^2 - N(\| \varphi \|_x + \| \varphi \|_y) \tag{2}$$

First we will show that no arbitrary partition is superior to either the Straight-Line or the Square-Corner partition. The optimal partitioning scheme depends on the ratio of computational processing power between the two processors. When this ratio, r, is less than 3 the Straight-Line partitioning provides the minimum volume of communication. However, when r is greater than 3, Square-Corner partitioning minimizes communication time. At $r = 3$, the two partitioning schemes are equivalent.

Theorem 1 (Arbitrary Partition): For SCB, there exists no arbitrary partition with a lower volume of communication than either the Straight-Line or the Square-Corner partition.

Proof: Consider any arbitrary partition, φ. Repeatedly apply algorithm $push$ in any direction.

Theorem 2 (push): The $push$ algorithm output partition, φ_1, will have lower, or at worst equal, communication volume to the algorithm input partition, φ.

Proof: From (2) we know that $Tcomm$ is given by $2N^2 - N(\| \varphi \|_x + \| \varphi \|_y)$. We observe,

Lemma 1: As $(\| \varphi \|_x + \| \varphi \|_y)$ increases, $Tcomm$ decreases.

$push \downarrow$ and $\uparrow$ on φ create φ_1 such that:

For row k being cleaned,
If there exists some row i that was clean, but now dirty:
$r(\varphi, i) = 0$ and $r(\varphi_1, i) = 1$
then by Axiom 1:
$\| \varphi_1 \|_x = \| \varphi \|_x$
else
$\| \varphi_1 \|_x = \| \varphi \|_x + 1$
By Axiom 2:
$\| \varphi_1 \|_y \geq \| \varphi \|_y$

$push \rightarrow$ and $\leftarrow$ on φ create φ_1 such that:

For column k being cleaned,
If there exists some column j that was clean, but now dirty:
$c(\varphi, j) = 0$ and $c(\varphi_1, j) = 1$
then by Axiom 3:
$\| \varphi_1 \|_y = \| \varphi \|_y$
else
$\| \varphi_1 \|_y = \| \varphi \|_y + 1$
By Axiom 4:
$\| \varphi_1 \|_x \geq \| \varphi \|_x$

By these definitions of all push operations we observe that for any push operation, $(\| \varphi_1 \|_x + \| \varphi_1 \|_y) \geq (\| \varphi \|_x + \| \varphi \|_y)$. Therefore, we conclude that all push operations will either decrease communication time or leave it unchanged. ■

By repeatedly performing this operation, $push$, we incrementally lower the volume of communication, and each resulting output partition is better than the input. If we apply the $push$ until it can longer be done anymore, we get the resulting partitions which minimize communication time. The optimal partition will have all of the possible $push$ operations performed, as leaving one unperformed may lead to a greater communication volume and will certainly never lead to a lower communication volume.

Theorem 3 (Resulting Partitions): Applying the $push$ algorithm until all 4 directions return as output, φ_1, such that $\varphi_1 = \varphi$, the input results in one of 15 partitions.

Proof:

We have defined our problem to be limited only to numbers of elements which can be made to form a rectangle of some kind. The partitions are given in Fig.4 ■

Theorem 4 (Partition Location): The x, y coordinates of the smaller processor's section of a data partition does not affect the total volume of communication of that data partition.

Proof: The total volume of communication of a data partition is determined by the number of $dirty$ rows and columns. Two partitions with identical size and shape will create the same number of $dirty$ rows and columns, regardless of where they are placed. ■

Theorem 5 (Equivalent Partitions): All 15 resulting partitions are equivalent variations of the Straight-Line or Square-Corner partitioning schemes.

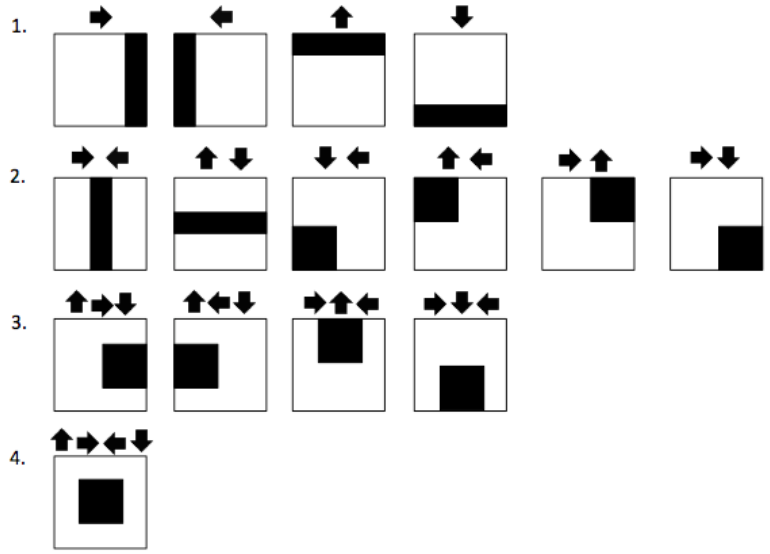


Fig. 4. The result of applying operations $\downarrow$, $\uparrow$, $\leftarrow$, and $\rightarrow$, until the stopping point has been reached. Row 1 shows the result of applying just a single transformation. Row 2 shows the result of applying a combination of two transformations. Row 3 shows the possible results of applying three transformations, and Row 4 shows the result of applying all four transformations.

Proof: The 6 Straight-Line partitions are equivalent to each other, as are the 9 Square-Corner partitions. The location of the Q portion of the partition with in the P portion does not affect the total volume of communication necessary, and therefore is unimportant. For simplicity we depict the Straight-Line as a rectangle with its N-length side adjacent to an edge of the partition, and the Square-Corner as adjacent to two edges of the partition. Previous work has also shown the ideal shape of rectangle of width less than N is a square[5]. So although some rectangles given by $push$ may be non-square, but not of length N, we will say they must conform to the Square-Corner scheme. ■

Therefore, from any arbitrary partition we have created a partition which is one of 15 possible outcomes, all of which are variations of the Square-Corner or Straight-Line partition. These are guaranteed to have lower, or equal, volumes of communication than any other possible partition. There exists no arbitrary partition with a lower volume of communication. ■

Now that we have eliminated partitioning schemes other than Straight-Line and Square-Corner, we focus on which of these is optimal in the given scenarios. The Straight-Line partition is understood to be $N \times x$ in dimension, and the Square-Corner partition is understood to be $q \times q$.

Theorem 6 (SCB): For serial communication with a barrier between communication and computation, Square-Corner partitioning is optimal for all computational power ratios, r, greater than 3, and Straight-Line partitioning is optimal for all ratios less than 3.

Proof: The Straight-Line partitioning total volume of communication is a constant, always equal to N^2. The Square-Corner partitioning total volume of communication is equal to $2Nq$. We state that $2Nq < N^2$ subject to the conditions $N, q > 0$. The optimal value of q is given by $q = \frac{N}{\sqrt{r+1}}$. Substituting

this in, yields:

$$\begin{aligned}\frac{2N^2}{\sqrt{r+1}} &< N^2\\ 2 &< \sqrt{r+1}\\ 4 &< r+1\\ r &> 3\end{aligned}$$

We conclude that Square-Corner is optimal for all $r > 3$, and Straight-Line is optimal for all $r < 3$. ■

IV. Parallel Communication with Barrier - PCB

This scenario takes place in two steps. First, communication is done, with Processor P sending to Processor Q, and Q sending to P, in parallel. Once the communication completes, both processors compute their portion of the MMM in parallel. Again, the optimal partitioning scheme depends on the computational power ratio, r, between the two processors. For ratios two and under, the Straight-Line partitioning minimizes the volume of communication. However, when r is greater than 2, Square-Corner partitioning minimizes the communication time.

$$Texecution = Tcomm + Tcomp \tag{3}$$

$$Tcomm = \max(P \rightarrow Q, Q \rightarrow P) \tag{4}$$

$$Tcomm = max[(2 \times \#P) - N(\parallel \rho \parallel_x^0 + \parallel \rho \parallel_y^0), (2 \times \#Q) - N(\parallel \rho \parallel_x^1 + \parallel \rho \parallel_y^1)]$$

Theorem 7 (Arbitrary Partition): For PCB, there exists no arbitrary partition with a lower volume of communication than the Straight-Line or Square-Corner partition.

Proof: Consider any arbitrary partition, φ. Repeatedly apply algorithm $push$ in any direction.

Theorem 8 (push): The $push$ algorithm output partition, φ_1, will have lower, or at worst equal, communication volume to the algorithm input partition, φ.

Proof:

From the definition of the maximum function, and the definition of $Tcomm, parallel$, we make the following observations:

1) $Tcomm, parallel$ decreases as VP decreases if $VP > VQ$
2) $Tcomm, parallel$ remains constant as VP decreases if $VQ > VP$
3) $Tcomm, parallel$ remains constant as VQ increases if $VP > VQ$

VP will decrease or remain constant for all push operations on partition ρ.

a) : A push is defined to keep constant the number of elements in $\in P$ and $\in Q$. From this fact and from the long form of the definition of $Tcomm, parallel$, we observe **Lemma 1:** VP decreases as $(\parallel \rho \parallel_x^0 + \parallel \rho \parallel_y^0)$ increases.

b) :

push $\uparrow$ and $\downarrow$ of ρ are defined to create ρ_1 such that:

For some k, $r(\rho, k) = 1$ and $r(\rho_1, k) = 0$

By axiom 1,

If there exists some i such that $r(\rho, i) = 0$ and $r(\rho_1, i) = 1$ then

$$\| \rho_1 \|_x^0 = \| \rho \|_x^0$$

Else,

$$\| \rho_1 \|_x^0 = \| \rho \|_x^0 + 1$$

By axiom 2,

$$\| \rho_1 \|_y^0 \geq \| \rho \|_y^0$$

push $\rightarrow$ and $\leftarrow$ of ρ are defined to create ρ_1 such that:

For some k, $c(\rho, k) = 1$ and $c(\rho_1, k) = 0$

By axiom 3,

If there exists some j such that $c(\rho, j) = 0$ and $c(\rho_1, j) = 1$ then

$$\| \rho_1 \|_y^0 = \| \rho \|_y^0$$

Else,

$$\| \rho_1 \|_y^0 = \| \rho \|_y^0 + 1$$

By axiom 4,

$$\| \rho_1 \|_x^0 \geq \| \rho \|_x^0$$

From these definitions we observe **Lemma 2:** For all push operations, $(\| \rho_1 \|_x^0 + \| \rho_1 \|_y^0) \geq (\| \rho \|_x^0 + \| \rho \|_y^0)$. By Lemmas 1 and 2, we conclude that VP decreases or remains constant for any push operation.

c) : ■

By repeatedly performing this operation, $push$, we incrementally lower the volume of communication, and each resulting output partition is better than the input. If we apply the $push$ until it can longer be done anymore, we get the resulting partitions which minimize communication time. The optimal partition will have all of the possible $push$ operations performed, as leaving one unperformed may lead to a greater communication volume and will certainly never lead to a lower communication volume.

Theorem 9 (Resulting Partitions): Applying the $push$ algorithm until all 4 directions return as output, φ_1, such that $\varphi_1 = \varphi$, the input results in one of 15 partitions.

Proof: We have defined our problem to be limited only to numbers of elements which can be made to form a rectangle of some kind. The partitions are given in Fig.4 ■

Theorem 10 (Partition Location): The x, y coordinates of the smaller processor's section of a data partition does not affect the total volume of communication of that data partition.

Proof: The total volume of communication of a data partition is determined by the number of $dirty$ rows and columns. Two partitions with identical size and shape will create the same number of $dirty$ rows and columns, regardless of where they are placed. ■

Theorem 11 (Equivalent Partitions): All 15 resulting partitions are equivalent variations of the Straight-Line or Square-Corner partitioning schemes.

Proof: The 6 Straight-Line partitions are equivalent to each other, as are the 9 Square-Corner partitions. The location of the Q portion of the partition with in the P portion does not affect the total volume of communication necessary, and therefore is unimportant. For simplicity we depict the Straight-Line as a rectangle with its N-length side adjacent to an edge of the partition, and the Square-Corner as adjacent to two edges of the partition. Previous work has also shown the ideal shape of rectangle of width less than N is a square[5]. So although some rectangles given by $push$ may be non-square, but not of length N, we will say they must conform to the Square-Corner scheme. ■

Therefore, from any arbitrary partition we have created a partition which is one of 15 possible outcomes, all of which are variations of the Square-Corner or Straight-Line partition. These are guaranteed to have lower, or equal, volumes of communication than any other possible partition. There exists no arbitrary partition with a lower volume of communication. ■

Now that we have eliminated partitioning schemes other than Straight-Line and Square-Corner, we focus on which of these is optimal in the given scenarios. The Straight-Line partition is understood to be $N \times x$ in dimension, and the Square-Corner partition is understood to be $q \times q$.

Theorem 12 (PCB): For parallel communication with a barrier between communication and computation, Square-Corner partitioning is optimal for all ratios greater than 2, and Straight-Line partitioning is optimal for all ratios less than 2.

Proof: For all power ratios, the communication volume for Straight-Line partitioning is $N^2 - Nx$, where x is the dimension of Processor Q's portion and is given by $x = \frac{N}{r+1}$. The total volume of communication for Square-Corner partitioning depends on whether communication from P to Q, $VP = 2Nq - 2q^2$ or Q to P, $VQ = 2q^2$, dominates. $VP > VQ$ when $r > 3$. Therefore, we compare Square-Corner's VQ to Straight-Line. For the conditions $N, q, x > 0$:

$$\begin{aligned} N^2 - Nx &< 2q^2 \\ N^2 - N\frac{N}{r+1} &< 2(\frac{N}{\sqrt{r+1}})^2 \\ N^2 - \frac{N^2}{r+1} &< 2\frac{N^2}{r+1} \\ r &< 2 \end{aligned}$$

■

V. Serial Communication with Bulk Overlap - SCO

We now consider the scenarios where we use overlap, meaning we do communication and some computation in parallel. Due to the layout of a Square-Corner partition, there is a section belonging to Processor P that does not require communication in order to compute. This section of P, of size $(N-q)^2$, will be computed while communication takes place first in one direction, and then the other. Only once the communication has completed does the computation begin on the other sections of P and on Processor Q.

By taking advantage of this feature of the Square-Corner partitioning, the Square-Corner partition can be made to have a lower total execution time than the Straight-Line partitioning for all power ratios. Execution time using this strategy is given by,

$$T_{exe} = \max(\max(Tcomm, P1) + (P2 + P3), (Tcomm + Q)) \tag{5}$$

where $P1, P2, P3, Q$ are the time taken to compute that section.

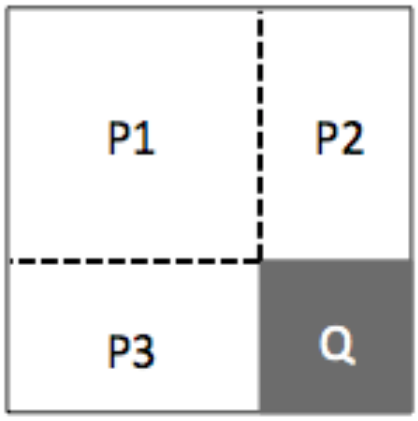


Fig. 5. A partition divided between Processors P and Q. P1 is the subsection of P that does not need to be communicated to Processor Q. Both P2 and P3 require communication.

Theorem 13 (Arbitrary Partition): For SCO, there exists no arbitrary partition with a lower volume of communication than the Straight-Line or Square-Corner partition.
This proof is the same as that for SCB, as they both use serial communication. See above.

As the faster processor, P, gets a jump start on computing its section of the matrix, we will want to adjust the proportion of the total matrix it receives, making it larger. Therefore, the optimal value for the size of Processor Q, q, will decrease. To determine this optimal value of q, the T_{exe} equation must be put in terms of q. A single unit of computation is considered to be $C[i,j] = A[i,k] * B[k,j] + C[i,j]$.

$$Tcomm = \#\text{ of elements} * \beta = ((2Nq - 2q^2) + 2q^2)\beta = 2Nq\beta$$
$$Sp = \text{Speed of Processor P, units of computation/ second}$$
$$Sq = \text{Speed of Processor Q, units of computation/ second}$$
$$y = \text{units of computation/ matrix element} = N$$
$$\beta = \text{transfer time / matrix element}$$

Each portion of the matrix is therefore equal to $\frac{V*y}{S}$, the volume of that section times N, divided by the speed of the processor. Substituting all these, we find T_{exe} is,

$$T_{exe} = \max(\max\left(2Nq\beta, \frac{N\left(N-q\right)^2}{Sp}\right) + 2\frac{Nq\left(N-q\right)}{Sp}, 2Nq\beta + \frac{Nq^2}{Sq}) \tag{6}$$

This equation will be easier to analyze and compare by factoring out the large constant, $N^3\beta$, and normalizing q as a proportion of N, $\frac{q}{N}$, so that q is understood to be $0 \leq q \leq 1$. Also introduced is the variable c, given by $c = Sp * \beta$, which represents a ratio between computation and communication speeds.

$$\frac{T_{exe}}{N^3\beta} = \max(\max\left(\frac{2}{N}q, \frac{\left(1-q\right)^2}{c}\right) + 2\frac{\left(q-q^2\right)}{c}, \frac{2}{N}q + \frac{q^2}{\frac{c}{r}}) \tag{7}$$

The optimal value of q is the minimum of this function on the interval of $\{0, 1\}$. However, since a value of $q = 1$ would indicate that Processor Q has been assigned the entire matrix, the interval of possible q

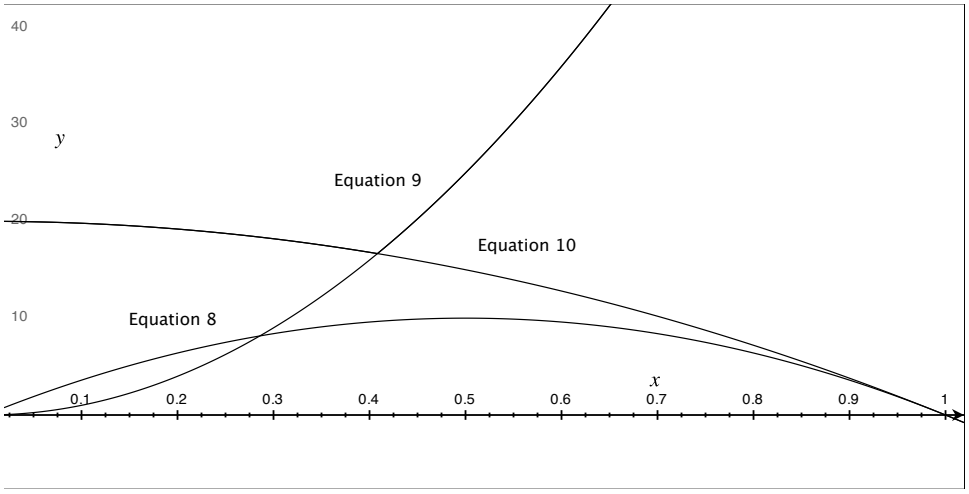


Fig. 6. Graph of 3 possible functions of execution time for a sample N = 3000, Processor Ratio (r) = 5:1 and Communication/Computation ratio (c) = .05.

values can be made more specific. The largest q will be without overlap is when $r = 1$, and therefore $q = \frac{1}{\sqrt{2}}$. We have already established that overlap will decrease the area assigned to Q, so it can certainly be said that the optimal value of q is the minimum of T_{exe} on the interval $\{0, \frac{1}{\sqrt{2}}\}$.

There are 3 functions that comprise the T_{exe} equation. These functions and what they represent are as follows,

$$y = \frac{2}{N}q + 2\frac{q - q^2}{c} : Tcomm + (P2 + P3) \tag{8}$$

$$y = \frac{(1-q)^2}{c} + 2\frac{q - q^2}{c} : P1 + (P2 + P3) \tag{9}$$

$$y = \frac{2}{N}q + \frac{q^2 r}{c} : Tcomm + Q \tag{10}$$

The first observation to make is that (8) is always less than (9) on the interval $\{0, \frac{1}{\sqrt{2}}\}$. Therefore for possible values of q, it will never dominate the max function and can be safely ignored. Focusing on (9) and (10), we note that (9) is concave down and (10) is concave up, and therefore the minimum on the interval will be at the intersection of these two functions.

$$9 \cap 10$$

$$\frac{(1-q)^2}{c} + 2\frac{q - q^2}{c} = \frac{2}{N}q + \frac{q^2 r}{c}$$

$$0 = q^2(r+1) + q(\frac{2c}{N}) - 1$$

$$q = \frac{\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1}}{r+1}$$

Theorem 14 (SCO): For serial communication with a bulk overlap between communication and P1 computation, Square-Corner partitioning is optimal with a lower total execution time than Straight-Line partitioning for all processor power ratios.

Proof: The Straight-Line partitioning has an execution time, once the constant $N^3\beta$ is removed and x is normalized, given by $T_{exe}, SL = \frac{1}{N} + \max(\frac{1-x}{c}, \frac{rx}{c})$. Because the layout of Straight-Line does not allow for this type of easy overlap, its optimal x is still given by $x = \frac{1}{r+1}$.

$$\text{Straight-Line Execution} > \text{Square-Corner Execution}$$
$$\frac{1}{N} + \frac{1-x}{c} > \frac{(1-q)^2}{c} + 2\frac{q-q^2}{c}$$
$$q^2 > x - \frac{c}{N}$$
$$(\frac{\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1}}{r+1})^2 > \frac{1}{r+1} - \frac{c}{N}$$
$$(\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1})^2 > (r+1) - \frac{c}{N}(r+1)^2$$
$$\frac{c^2}{N^2} - \frac{2c}{N}\sqrt{\frac{c^2}{N^2} + r + 1} + \frac{c^2}{N^2} + r + 1 > r + 1 - \frac{c}{N}(r+1)^2$$
$$\frac{2c}{N} + (r+1)^2 > 2\sqrt{\frac{c^2}{N^2} + r + 1}$$
$$\frac{4c}{N}(r+1)^2 + (r+1)^4 > 4(r+1)$$
$$\frac{4c}{N} + r^3 + 3r^2 + 3r > 3$$
$$(\text{always positive for } c, N \geq 0) + (> 3 \text{ for } r \geq 1) > 3$$
$$\text{SL has a greater execution time for all } c, N \geq 0 \text{ and } r \geq 1$$

Therefore, by taking advantage of the overlap ready layout of the Square-Corner partitioning scheme, the Square-Corner partitioning becomes optimal for all processor ratios. ■

VI. Parallel Communication with Bulk Overlap - PCO

In this algorithm, communication occurs in both directions in parallel while Processor P is also computing its subsection, P1, which does not require communication. Once the communication is complete, Processor P computes the remainder of its portion, while Processor Q computes in parallel. Square-Corner execution time using this strategy the same as (5) where $Tcomm = \max(VP, VQ)$. Again, computation of each portion of the matrix is $\frac{V*y}{S}$, the volume times N, divided by processor speed. Substituting these, total execution time is given by,

$$T_{exe} = \max(\max(\max(2Nq\beta - 2q^2\beta, 2q^2\beta), \frac{N(N-q)^2}{Sp}) + 2\frac{Nq(N-q)}{Sp}, \max(2Nq\beta - 2q^2\beta, 2q^2\beta) + \frac{Nq^2}{Sq}) \quad (11)$$

Theorem 15 (Arbitrary Partition): For PCO, there exists no arbitrary partition with a lower volume of communication than the Straight-Line or Square-Corner partition.
The proof of this is identical to the proof for PCB as they both use parallel communication. See above.

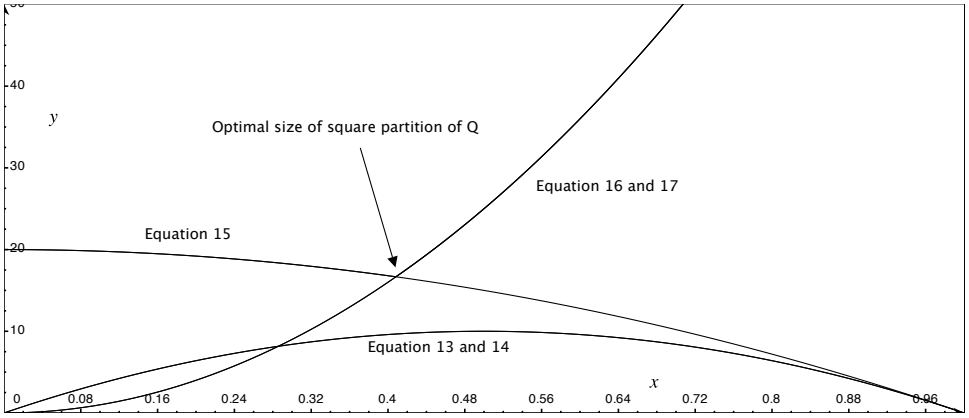


Fig. 7. Graph of 5 possible parabolas for square corner partitioning with parallel communication and overlap. Equations 13 and 14, and 16 and 17 are nearly identical, respectively, and appear as a single curve. Given problem parameters N=3000, Processor Ratio = 5:1, and Communication/Computation Ratio = .05.

Again, for analysis and comparison we factor out the large constant, $N^3\beta$, and normalize q as a proportion of N, $\frac{q}{N}$, so that q is understood to be $0 \leq q \leq 1$. Also introduced is the variable c, given by $c = Sp * \beta$, which represents the ratio between computation and communication speeds.

$$\frac{T_{exe}}{N^3\beta} = \max(\max(\max(\frac{2q}{N} - \frac{2q^2}{N}, \frac{2q^2}{N}), \frac{(1-q)^2}{c}) + 2\frac{(q-q^2)}{c}, \max(\frac{2q}{N} - \frac{2q^2}{N}, \frac{2q^2}{N}) + \frac{rq^2}{c}) \quad (12)$$

In order to compare this with Straight-Line partitioning, the optimal value of q must be found on the interval $\{0, \frac{1}{\sqrt{2}}\}$. There are 5 functions that comprise (12). These functions and what they represent are as follows,

$$y = \frac{2q}{N} - \frac{2q^2}{N} + 2\frac{(q-q^2)}{c} : VP + (P2 + P3) \quad (13)$$

$$y = \frac{2q^2}{N} + 2\frac{(q-q^2)}{c} : VQ + (P2 + P3) \quad (14)$$

$$y = \frac{(1-q)^2}{c} + 2\frac{(q-q^2)}{c} : P1 + (P2 + P3) \quad (15)$$

$$y = \frac{2q}{N} - \frac{2q^2}{N} + \frac{rq^2}{c} : VP + Q \quad (16)$$

$$y = \frac{2q^2}{N} + \frac{rq^2}{c} : VQ + Q \quad (17)$$

Both (13) and (14) are less than (15) on the interval $\{0, \frac{1}{\sqrt{2}}\}$, and can be safely ignored. Of the remaining 3 equations, (15) is concave down and both (16) and (17) are concave up on the interval. The optimal value of q, the minimum, is therefore at the intersection of (15), and whichever other function dominates. For $q < \frac{1}{2}$, (16) dominates and for $q > \frac{1}{2}$ (17) dominates. We have already established that Square-Corner is superior for ratios greater than 2 using parallel communication. Ratios less than and equal to 2, will

have q values greater than $\frac{1}{2}$, so the optimal value of q for the comparison is at $(15) \cap (17)$.

$$15 \cap 17$$
$$\frac{(1-q)^2}{c} + 2\frac{(q-q^2)}{c} = \frac{2q^2}{N} + \frac{rq^2}{c}$$
$$q = \frac{1}{\sqrt{r+1+\frac{2c}{N}}}$$

Theorem 16 (PCB): For parallel communication with a bulk overlap between communication and P1 computation, Square-Corner partitioning is optimal with a lower total execution time than Straight-Line partitioning for all processor power ratios.

Proof: The Straight-Line partitioning has an execution time, once the constant $N^3\beta$ is removed and x is normalized, given by $T_{exe}, SL = \max(\frac{1}{N} - \frac{x}{N}, \frac{x}{N}) + \max(\frac{1-x}{c}, \frac{rx}{c})$. Of the 4 functions which comprise this equation, only two dominate when $x < \frac{1}{2}$, which must always be true for Straight-Line partitioning. Of these two functions, one is of negative slope, and the other of positive slope, so the minimum on the interval is at their intersection. Again, this intersection is at $x = \frac{1}{r+1}$.

$$\text{Straight-Line Execution} > \text{Square-Corner Execution}$$
$$\frac{1}{N} - \frac{x}{N} + \frac{1-x}{c} > \frac{1-q^2}{c}$$
$$q^2 + \frac{c}{N} > x + \frac{cx}{N}$$
$$\frac{1}{r+1+\frac{2c}{N}} + \frac{c}{N} > \frac{1}{r+1} + \frac{c}{N(r+1)}$$
$$1 + \frac{c(r+1+\frac{2c}{N})}{N} > \frac{r+1+\frac{2c}{N}}{r+1} + \frac{c(r+1+\frac{2c}{N})}{N(r+1)}$$
$$\frac{cr^2}{N} + \frac{cr}{N} + \frac{2c^2r}{N^2} > \frac{2c}{N}$$
$$r + 1 - \frac{2}{r} > -\frac{2c}{N}$$
$$\text{is } \geq 0 \text{ when } r \geq 1 > \text{ is } < 0$$

Therefore, for all $c, N > 0$ and $r \geq 1$, Square-Corner partitioning is optimal when taking advantage of the communication/computation overlap on the faster processor. ■

VII. Parallel Interleaving Overlap - PIO

The bulk overlap operation is not the only way in which to overlap communication and computation. The parallel kij algorithm we use to compute the matrices allows each processor to incrementally update the result matrix as it receives the necessary data. We will refer to this as interleaving overlap. It occurs as described in the following algorithm.

$k \leftarrow 1$
Send data corresponding to row and column k
for $k = 1 \rightarrow (N-1)$ **do**
  In Parallel:
  Send data corresponding to row and column $k+1$
  Processor P updates C with data from row and column k

Processor Q updates C with data from row and column k

end for

Processors P and Q update C with data from row and column N

For any given step k, the total amount of data being sent using this algorithm on a Square-Corner partition will be q. We define the execution time of the Square-Corner partitioning to be given by,

$$T_{exe} = 2\beta q + (N-1) \times \max(2\beta q, \frac{N^2-q^2}{Sp}, \frac{q^2}{Sq}) + \max(\frac{N^2-q^2}{Sp}, \frac{q^2}{Sq}) \tag{18}$$

Similarly, we may use this algorithm for the Straight-Line partitioning, where the amount of data sent at each step k will be N. We define the execution time of the Straight-Line partitioning to be given by,

$$T_{exe} = N\beta + (N-1) \times \max(N\beta, \frac{N(N-x)}{Sp}, \frac{Nx}{Sq}) + \max(\frac{N(N-x)}{Sp}, \frac{Nx}{Sq}) \tag{19}$$

Because there is no bulk overlap, the optimal size for the smaller partition is the same as for SCB and PCB, $\frac{1}{r+1}$ for Straight-Line and $\frac{1}{\sqrt{r+1}}$ for Square-Corner.

Theorem 17 (Arbitrary Partition): For PIO, there exists no arbitrary partition with a lower volume of communication than either the Straight-Line or the Square-Corner partition.

Proof: Note first that PIO completely serializes the communication for Square-Corner and Straight-Line partitions. The communication takes place in one direction, and then the other. Therefore, communication time is the same as for any serial algorithm. See SCB above. ■

Intuitively, when communication dominates the PIO algorithm we expect the partition with the lower volume of communication to be optimal. We have already shown that Square-Corner is optimal for computational power ratios greater than 3, and Straight-Line for ratios less than 3. When c, the ratio of communication and computation time, is such that computation dominates we expect the two partitions to be equivalent, because we have optimally balanced the volume of computation between the two processors. This is true if we look at the dominating term of each equation, the max function times $(N-1)$. However, because of the addition of communication time as the first term of the equations, whichever partition has the lower volume communication volume will be a small amount more efficient for these values of c. Since we are trying to find the optimal partition under all circumstances with this algorithm, we do not ignore these terms.

Theorem 18 (PIO): For parallel interleaving overlap, Square-Corner is optimal for computational power ratios, r, greater than 3, and Straight-Line is optimal for ratios less than 3.

Proof: We begin by giving these equations the same treatment as previously, removing the constant $N^3\beta$ and normalizing x, q to $\frac{x}{N}$ and $\frac{q}{N}$ respectively. First we consider the values of c where communication dominates. This occurs at $c > N(1-x)$ for Straight-Line and $c > \frac{N}{2}(\frac{1}{q}-q)$ for Square-Corner. Practically, these are large values of c which would indicate a relatively small communication bandwidth compared to the computational resources. When communication dominates our function, the formulas are,

$$T_{exe}, SC = \frac{2q}{N} + \frac{1-q^2}{c} \tag{20}$$

$$T_{exe}, SL = \frac{1}{N} + \frac{1-x}{c} \tag{21}$$

We begin by stating that for the given optimal values of x and q, Straight-Line is greater than Square-Corner,

$$\begin{aligned} SL &> SC \\ \frac{1}{N} + \frac{1-x}{c} &> \frac{2q}{N} + \frac{1-q^2}{c} \\ \frac{1}{N} + \frac{1-(\frac{1}{r+1})}{c} &> \frac{2(\frac{1}{\sqrt{r+1}})}{N} + \frac{1-(\frac{1}{\sqrt{r+1}})^2}{c} \\ 1 &> 2(\frac{1}{\sqrt{r+1}}) \\ r+1 &> 4 \\ r &> 3 \end{aligned}$$

Therefore, when c is such that communication dominates, Straight-Line is optimal for ratios less than 3, and Square-Corner is optimal for ratios greater than 3.

When c is such that computation dominates, the formulas are,

$$T_{exe}, SC = \frac{2q}{N^2} + \frac{1-q^2}{c} \tag{22}$$

$$T_{exe}, SL = \frac{1}{N^2} + \frac{1-x}{c} \tag{23}$$

We state that for the given optimal values of x and q, Straight-Line is greater than Square-Corner,

$$\begin{aligned} SL &> SC \\ \frac{1}{N^2} + \frac{1-x}{c} &> \frac{2q}{N^2} + \frac{1-q^2}{c} \\ \frac{1}{N^2} + \frac{1-(\frac{1}{r+1})}{c} &> \frac{2(\frac{1}{\sqrt{r+1}})}{N^2} + \frac{1-(\frac{1}{\sqrt{r+1}})^2}{c} \\ 1 &> 2(\frac{1}{\sqrt{r+1}}) \\ r+1 &> 4 \\ r &> 3 \end{aligned}$$

Therefore, when c is such that computation dominates, Straight-Line is optimal for ratios less than 3, and Square-Corner is optimal for ratios greater than 3. ■

VIII. Summary of Results

We have found the general optimal solution for all matrix-matrix multiplication problems involving two processors. For 2 processors with a barrier between communication and computation, Square-Corner partitioning is optimal for ratios greater than 3 in serial communication and greater than 2 in parallel communication, and Straight-Line partitioning is optimal for ratios less than 3 in serial communication and less than 2 for parallel communication. However, if we overlap the computation that can be done immediately on a square-corner partition with the communication, square-corner partitioning is optimal for all power ratios of processors and communication/computation ratios for both serial and parallel communication.

REFERENCES

[1] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers," in *Proceedings of the 7th International Conference on High Performance Computing and Networking Europe (HPCN'99)*, 1999.

[2] O. Beaumont, V. Boudet, F. Fabrice, and Y. Robert, "Matrix-matrix multiplication on heterogeneous platforms," in *IEEE Transactions on Parallel and Distributed Systems*, 5-8 July 2001, Vol. 12, No 10, pp. 1033–1051.

[3] A. Lastovetsky and J. Dongarra, *High-performance heterogeneous computing*, ser. Wiley series on parallel and distributed computing. Wiley, 2009. [Online]. Available: http://books.google.com/books?id=wfC7LPTcRmYC

[4] B. A. Becker, "High-level data partitioning for parallel computing on heterogeneous hierarchical computational platforms," PhD Thesis, University College Dublin, Dublin, Ireland, April 2011.

[5] B. A. Becker and A. Lastovetsky, "Matrix multiplication on two interconnected processors," in *Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006)*, IEEE Computer Society. Barcelona, Spain: IEEE Computer Society, 25-28 Sept 2006, cd-rom/Abstracts Proceedings.

[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[7] R. Hockney, "The communication challenge for mpp: Intel paragon and meiko cs-2," *Parallel Computing*, vol. 20, no. 3, pp. 389–398, 1994.