Partitioning for Parallel Matrix-Matrix Multiplication with Heterogeneous Processors: The Optimal Solution

Ashley DeFlumere, Alexey Lastovetsky, Brett A. Becker
School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Ireland
ashley.deflumere@ucdconnect.ie, {alexey.lastovetsky, brett.becker}@ucd.ie

Abstract—The problem of matrix partitioning for parallel matrix-matrix multiplication on heterogeneous processors has been extensively studied since the mid 1990s. During this time, previous research focused mainly on the design of efficient partitioning algorithms, optimally or sub-optimally partitioning matrices into rectangles. The optimality of the rectangular partitioning shape itself has never been studied or even seriously questioned. The accepted approach is that consideration of non-rectangular shapes will not significantly improve the optimality of the solution, but can significantly complicate the partitioning problem, which is already NP-complete even for the restricted case of rectangular shapes. There is no published research, however, supporting this approach. The shape of the globally optimal partitioning, and how the best rectangular partitioning compares with this global optimum, are still wide open problems. Solution of these problems will decide if new partitioning algorithms searching for truly optimal, and not necessarily rectangular, solutions are needed.

This paper presents the first results of our research on the problem of optimal partitioning shapes for parallel matrix-matrix multiplication on heterogeneous processors. Namely, the case of two interconnected processors is comprehensively studied. We prove that, depending on performance characteristics of the processors and the communication link, the globally optimal partitioning will have one of just two well-specified shapes, one of which is rectangular and the other is non-rectangular. The theoretical analysis is conducted using an original mathematical technique proposed in the paper. It is shown that the technique can also be applied in the case of arbitrary numbers of processors. While comprehensive analysis of the cases of three and more processors is more complicated and the subject for future work, the paper does prove the optimality of some particular non-rectangular partitioning shapes for some combinations of performance characteristics of heterogeneous processors and communication links. The paper also presents experimental results demonstrating that the optimal non-rectangular partitioning can significantly outperform the optimal rectangular one on real-life heterogeneous HPC platforms.

Keywords-Parallel Matrix Multiplication; Matrix Partitioning; Heterogeneous Computing; High Performance Computing

I. Introduction

Parallel Matrix-Matrix Multiplication (MMM) is a well studied problem on sets of homogenous processors. The optimal solutions and their corresponding data partition shapes are well known, and implemented in the form of mathematical software [1]. As heterogeneous systems have emerged as high performance computing platforms, the traditional homogenous algorithms have been adapted to these heterogeneous environments [2]. Although heterogeneous systems have been in use for some time, it remains an open problem of how to optimally partition data on heterogeneous processors to minimize computation, communication, and execution time.

Previous research has focused mainly on designing partitioning algorithms to find the optimal solution based on rectangles. Many solutions to accomplish this task efficiently have been proposed. Each of these solutions is based on different heuristics and models, but they all create partitions with a rectangular shape, *i.e.* one where each processor is assigned a rectangular portion of the matrix to compute [3] [4] [5] [6] [7]. However, finding the optimal rectangular partitioning efficiently is difficult. Indeed, to do so for an arbitrary number of processors has been shown to be NP-complete [8]. Despite all the research devoted to finding these optimal rectangular partitions, no body of research exists which compares these partitions to the global optimum. Indeed, the optimality of the rectangular partitioning shape itself has never been studied or even seriously questioned. The accepted thinking is that non-rectangular shapes will not significantly improve the solution, and that they can even significantly complicate the problem. How the optimal rectangle-based solution compares to the globally optimal solution, and the general shape of such a global optimum, remain wide open problems. These open problems, when solved, will decide if new partitioning algorithms are needed to search for globally optimal, and not necessarily rectangular, solutions.

As this is an initial foray into the subject, an open problem which to the best of our knowledge has not been previously studied, we begin with the fundamental case of two processors. We use this case as a test-bed to develop our novel mathematical technique, which is presented here as the central contribution of this paper. With this original method it is possible to analyze all partitions and mathematically

prove what are the optimal partition shapes. Central in developing this technique was the requirement that it be applicable beyond the two processor case. In the future, it will be used to comprehensively study three and more processors. In this work, the technique is employed to show that the optimal shape of a data partition for two processors may be non-rectangular, depending on the performance characteristics of the processors and the communication link. Specifically the optimal partition will take either one of two shapes, one of which is rectangular and the other non-rectangular. As part of this two processor work, the mathematical technique provided some low hanging fruit in the case of three and more processors. These initial results are discussed, proving for arbitrary numbers of processors the optimal partition shape is non-rectangular, for some performance characteristics of computation and communication.

For two processors, the only way to create a rectangular partition is the Straight-Line approach. The matrices are divided into two rectangles of areas proportional to processor speed. The other, non-rectangular, partition will be called the Square-Corner, in which a small square is assigned to the slower processor and the non-rectangular remainder of the matrix is assigned to the faster processor. These partition types are depicted in Figure 1.

The idea of a two processor partition composed of a small square and the non-rectangular remainder of the matrix is discussed previously in [9] and [10]. In this paper, however, we approach this not as a problem of whether the Square-Corner partition is superior to the rectangular under certain configurations, but what is the optimal partition shape. We also improve upon these works by considering a variety of algorithms to compute the MMM as opposed to one. The non-rectangular partition has also been extended to three processors in [11], and we will build on this and explore extending these results to arbitrary numbers of processors.

To validate and corroborate this theoretical work, included are a number of experimental results. These results, for both two and three processors, are obtained both on individual processors and clusters on the large geographically distributed grid platform Grid'5000. This work can also be easily adapted to other high performance heterogeneous systems. One common concept of heterogeneity today is a number of CPUs with a GPU accelerator, and the case of optimally partitioning between relatively slow processors (*i.e.* CPUs) and one faster processor (*i.e.* GPU) is discussed and solved for some performance values in this paper.

This paper is comprised of the comprehensive results of the initial study of two abstract processors. We analyze five different algorithms to compute MMM on two processors, beginning with the most simplistic and building up to the most realistic for use on todays systems. Even the basic algorithms are of importance, however, as they can be used to characterize other application types beyond simply parallel MMM.
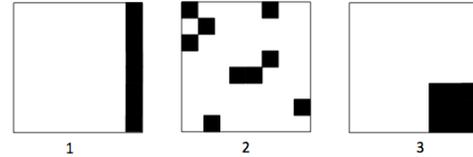
Figure 1. 1. A Straight-Line partition 2. An arbitrary partition 3. A Square-Corner partition

The first two algorithms place 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 use 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.

The contributions of this paper are three-fold. First, 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. Second, we show how this optimal solution may be extended to the case of an arbitrary number of processors. Last but not least, we develop original mathematical techniques in order to prove the optimality of the found solution.

The rest of this paper is outlined as follows: Theoretical results for each algorithm are contained in sections II - VII. Section IX presents experimental results. Section X presents our conclusions and future work.

II. THEORETICAL RESULTS

In this section we will prove that a non-rectangular partitioning is optimal compared to all other MMM partitioning shapes for

- all processor power ratios when bulk overlapping communication and computation
- 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 [1] to compute MMM, 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 Figure 2. For each iteration of k, every element of the result matrix C is updated, incrementally obtaining the final value. An element in the pivot column k of matrix A,
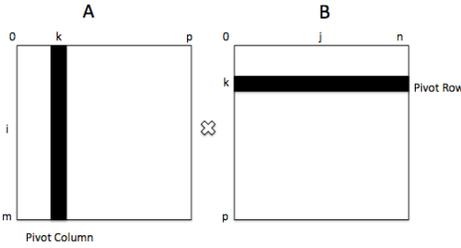
Figure 2. Pivot row and column, k, of the kij algorithm. Every element of C is updated before k is moved to the next row and column.

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,

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

C. 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.

$$\| \varphi \|_x = \text{\# of clean rows in } \varphi$$
$$\| \varphi \|_y = \text{\# of clean columns in } \varphi$$

$$r(\varphi, i) = \begin{cases} 0 & \text{if } (i, \cdot) \text{ of } \varphi \text{ is clean} \\ 1 & \text{if } (i, \cdot) \text{ of } \varphi \text{ is dirty} \end{cases}$$

$$c(\varphi, j) = \begin{cases} 0 & \text{if } (\cdot, j) \text{ of } \varphi \text{ is clean} \\ 1 & \text{if } (\cdot, j) \text{ of } \varphi \text{ is dirty} \end{cases}$$

III. SERIAL COMMUNICATION WITH BARRIER - SCB

In this first algorithm, communication is done serially; Processor P first sends data to Processor Q, and upon completion of receiving, Processor Q sends data to Processor P. Finally, each processor computes its portion of the matrix in parallel. Communication is modeled using the linear Hockney Model [12], and all computation is done with the kij algorithm. For all partitioning shapes,

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

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

$$Tcomp = \max(\#P \times Sp, \#Q \times Sq) \tag{3}$$

where $\#X =$ elements in processor X,

and $Sx =$ computation speed of processor X

Note that the computation time, (3), does not depend on the partitioning shape, φ, and therefore to minimize execution time, we must minimize communication time (2).

First we will show that no arbitrary partition's communication time is superior to either the Straight-Line or the Square-Corner communication time. The optimal partitioning shape, either Straight-Line or Square-Corner, depends on the ratio of computational processing power between the two processors. When this ratio, r, is less than $3 : 1$ the Straight-Line partitioning provides the minimum volume of communication. However, when r is greater than $3 : 1$, Square-Corner partitioning minimizes communication time. At $r = 3 : 1$, the two partitioning shapes are equivalent.

Theorem 3.1 (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.

This is the central theorem of this paper. Presented below are a number of component theorems and their proofs, which provide the basis of proof for this central theorem. First, we present the original mathematical technique, called a *push*. This technique takes any arbitrary starting partition and alters it in such a way that we may guarantee the communication time of the resulting shape is better, or at least not worse, than the starting partition. The basic component to the *push* technique is the *enclosing rectangle* placed around all elements of Processor Q, as seen in Figure 3. The enclosing rectangle is strictly large enough to contain

all elements belonging to Processor Q, and when a $push$ is done to a side of this rectangle, the other three sides are stationary. In this way, the enclosing rectangle condenses all the elements of Q into this smaller area, and, when applied repeatedly, eventually creates one of a defined set of resulting partition shapes. These resulting shapes may divided into three classes of shapes, one rectangular and two non-rectangular, and the canonical shape for each class is given. One non-rectangular shape, the Square-Corner, is shown to be better than the other non-rectangular shape in all circumstances. Analyzing the remaining two canonical shapes, Straight-Line and Square-Corner, we may determine for what performance characteristics each is optimal.

Here, we describe the $\downarrow$ direction of the $push$ operator. The $\uparrow, \leftarrow$ and $\rightarrow$ directions are similar, and a full description may be found in [13].
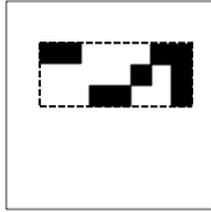
Figure 3. An arbitrary partition, φ, between Processor P (white) and Processor Q (black), with an enclosing rectangle (dashed line) around elements of Processor Q.

The $push \downarrow$ technique creates a new partition from the existing one by cleaning the top row of the enclosing rectangle, k_{top}, 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, is of P in the input partition φ, and is within the enclosing rectangle of Q. Consider the sides of the enclosing rectangle, in clockwise order, to be called $k_{top}, k_{right}, k_{bottom}$ and k_{left}. Formally, $\downarrow (\varphi) = \varphi_1$ where,

Initialize $\varphi_1 \leftarrow \varphi$
$(g, h) \leftarrow (k_{top} + 1, k_{left})$
for $j = k_{left} \rightarrow k_{right}$ **do**
  if $\varphi(k_{top}, j) = 1$ **then**
    $\varphi_1(k_{top}, j) \leftarrow 0$ {Element was dirty, clean it}
    $(g, h) \leftarrow$ find (g, h) {Function defined below}
    $\varphi_1(g, h) \leftarrow 1$ {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 $g \rightarrow k_{bottom}$ **do**
  for $h \rightarrow k_{right}$ **do**
    if $\varphi_1(g, h) = 0$ && $\varphi(g, h) = 0$ && $c(\varphi, h) = 1$
    then
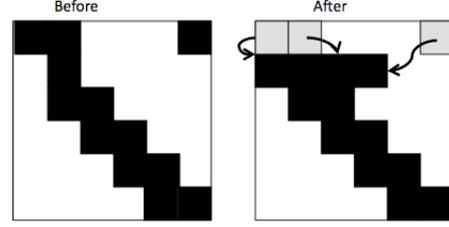
Figure 4. An arbitrary partition, φ, between Processor P (white) and Processor Q (black), and partition, φ_1, showing how the elements of row $k = 1$ have been pushed by the $\downarrow$ operation

      return (g, h)
    end if
    $h \leftarrow h + 1$
  end for
  $h \leftarrow k_{left}$
  $g \leftarrow g + 1$
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 $\downarrow (\varphi) = \varphi$.

Theorem 3.2 (push): The $push$ algorithm output partition, φ_1, will have lower, or at worst equal, communication time compared to the algorithm input partition, φ.

Proof: First 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 of 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.

From (2) we observe 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 is now dirty:
$$r(\varphi, i) = 0 \text{ and } r(\varphi_1, i) = 1$$
then by Axiom 1:
$$\| \varphi_1 \|_x = \| \varphi \|_x$$
else
$$\| \varphi_1 \|_x = \| \varphi \|_x + 1$$
and 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 is now dirty:
$$c(\varphi, j) = 0 \text{ and } c(\varphi_1, j) = 1$$
then by Axiom 3:
$$\| \varphi_1 \|_y = \| \varphi \|_y$$
else
$$\| \varphi_1 \|_y = \| \varphi \|_y + 1$$
and 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 (2) or leave it unchanged. ∎

By repeatedly performing this operation, $push$, we incrementally lower the communication time, and each resulting output partition is better than the input. If we apply the $push$ until it can no longer alter the partition, 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 larger communication time and will certainly never lead to a smaller communication time.

Theorem 3.3 (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 Figure 5. ∎

The partition shapes fall into two main forms, rectangular and non-rectangular. The rectangular shapes have one dimension of the enclosing rectangle of Q equal to N, the
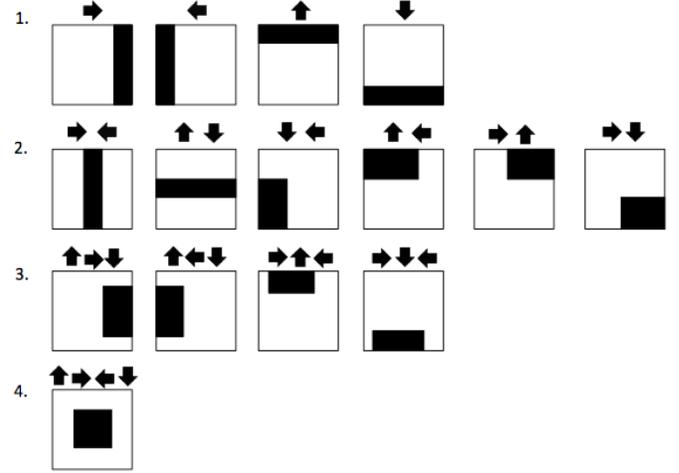
Figure 5. The result of applying operations $\downarrow$, $\uparrow$, $\leftarrow$, and $\rightarrow$, until the stopping point has been reached. Row 1 shows the result of applying just a single transformation. Row 2 shows the result of applying a combination of two transformations. Row 3 shows the possible results of applying three transformations, and Row 4 shows the result of applying all four transformations.

full length of the matrix. The non-rectangular shapes have an enclosing rectangle of Q in which both dimensions are less than N.

Theorem 3.4 (Canonical Forms): Partition shapes which have enclosing rectangles of Q of the same dimensions are equivalent regardless of the location of the enclosing rectangle within the overall matrix.

Proof: The location of the enclosing rectangle of Q within the overall matrix does not affect the total communication time necessary [13], and therefore is unimportant in minimizing execution time. We reduce these partition shapes to canonical forms, to allow for easy comparison between forms. These canonical forms are Straight-Line, Rectangle-Corner and Square-Corner, as shown in Figure 6.

Figure 6. Canonical forms of possible partitions resulting from the push operation. 1. Straight-Line 2. Rectangle-Corner 3. Square-Corner

∎

Theorem 3.5 (Square-Corner vs. Rectangle-Corner): Of all shapes with enclosing rectangles of dimensions less than N, the Square-Corner minimizes communication time.

Proof: Previous work has shown the optimal partition shape of a rectangle of width less than N, to minimize communication time, is a square [9]. ∎

From any arbitrary partition we have created a partition of the Square-Corner or Straight-Line shape. These are guaranteed to have lower, or equal, time of communication than any other possible partition. There exists no arbitrary partition with a lower communication time.

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

Theorem 3.6 (SCB): For serial communication with a barrier between communication and computation, Square-Corner partitioning is optimal for all computational power ratios, r, greater than $3 : 1$, and Straight-Line partitioning is optimal for all ratios less than $3 : 1$.

Proof: The Straight-Line partitioning shape has constant total volume of communication, always equal to N^2. The Square-Corner partitioning shape has a total volume of communication 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:

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

We conclude that the Square-Corner shape is optimal for all $r > 3 : 1$, and Straight-Line is optimal for all $r < 3 : 1$. ∎

IV. PARALLEL COMMUNICATION WITH BARRIER - PCB

This algorithm 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 less than $2 : 1$, the Straight-Line partitioning minimizes the communication time. However, when r is greater than $2 : 1$, Square-Corner partitioning minimizes the communication time.

$$Texe = Tcomm + Tcomp \qquad (4)$$
$$Tcomm = \max(P \rightarrow Q, Q \rightarrow P) \qquad (5)$$
$$Tcomp = \max(\#P \times Sp, \#Q \times Sq) \qquad (6)$$

Theorem 4.1 (Arbitrary Partition): For PCB, there exists no arbitrary partition with a lower communication time than the Straight-Line or Square-Corner partition.
The proof of this is similar to the proof for SCB and follows the same techniques. The full proof may be found in [13].

Theorem 4.2 (PCB): For parallel communication with a barrier between communication and computation, Square-Corner partitioning is optimal for all computational power ratios, r, greater than $2 : 1$, and Straight-Line partitioning is optimal for all ratios less than $2 : 1$.

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 $r > 3 : 1$. Therefore, we compare Square-Corner's VQ to Straight-Line. For the conditions $N, q, x > 0$:

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

∎

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 shape, the Square-Corner partition can be made to have a lower total execution time than the Straight-Line partitioning shape for all power ratios. Execution time using this algorithm is given by,

$$T_{exe} = \max(\max(Tcomm, P1) + (P2 + P3), (Tcomm + Q)) \qquad (7)$$

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

Theorem 5.1 (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}
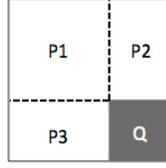
Figure 7. A partition divided between Processors P and Q. $P1$ is the subsection of P that does not need to be communicated to Processor Q. Both $P2$ and $P3$ require communication.

Figure 8. Graph of 3 possible functions of execution time for a sample N = 3000, Processor Ratio (r) = 5:1 and Communication/Computation ratio (c) = .05.

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 = Speed of Processor P, units of computation/ second

Sq = Speed of Processor Q, units of computation/ second

y = units of computation/ matrix element = N

β = 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,

$$T_{exe} = \max(\max\left(2Nq\beta, \frac{N(N-q)^2}{Sp}\right)$$
$$+ 2\frac{Nq(N-q)}{Sp}, 2Nq\beta + \frac{Nq^2}{Sq}) \quad (8)$$

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 $0 \le q \le 1$. Also introduced is the variable c, given by $c = Sp*\beta$, which represents a ratio between computation and communication speeds.

$$\frac{T_{exe}}{N^3\beta} = \max(\max\left(\frac{2}{N}q, \frac{(1-q)^2}{c}\right)$$
$$+ 2\frac{(q-q^2)}{c}, \frac{2}{N}q + \frac{q^2}{\frac{c}{r}}) \quad (9)$$

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 values can be made more specific. The largest q will be without overlap is when $r = 1 : 1$, 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,

$$y = \frac{2}{N}q + 2\frac{q-q^2}{c} : Tcomm + (P2 + P3) \quad (10)$$
$$y = \frac{(1-q)^2}{c} + 2\frac{q-q^2}{c} : P1 + (P2 + P3) \quad (11)$$
$$y = \frac{2}{N}q + \frac{q^2r}{c} : Tcomm + Q \quad (12)$$

The first observation to make is that (10) is always less than (11) 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 (11) and (12), we note that (11) is concave down and (12) is concave up, and therefore the minimum on the interval will be at the intersection of these two functions.

$$11 \cap 12$$
$$\frac{(1-q)^2}{c} + 2\frac{q-q^2}{c} = \frac{2}{N}q + \frac{q^2r}{c}$$
$$0 = q^2(r+1) + q(\frac{2c}{N}) - 1$$
$$q = \frac{\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1}}{r+1}$$

Theorem 5.2 (SCO): For serial communication with a bulk overlap between communication and $P1$ computation, the Square-Corner partitioning shape is optimal, with a lower total execution time than the Straight-Line partitioning shape 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 the Straight-Line shape does not allow for this type of

easy overlap, its optimal x is still given by $x = \frac{1}{r+1}$.

Straight-Line Execution > Square-Corner Execution

$$\frac{1}{N} + \frac{1-x}{c} > \frac{(1-q)^2}{c} + 2\frac{q-q^2}{c}$$

$$q^2 > x - \frac{c}{N}$$

$$(\frac{\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1}}{r+1})^2 > \frac{1}{r+1} - \frac{c}{N}$$

$$(\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1})^2 > (r+1) - \frac{c}{N}(r+1)^2$$

$$\frac{c^2}{N^2} - \frac{2c}{N}\sqrt{\frac{c^2}{N^2} + r + 1} + \frac{c^2}{N^2} + r + 1 >$$
$$r + 1 - \frac{c}{N}(r+1)^2$$

$$\frac{2c}{N} + (r+1)^2 > 2\sqrt{\frac{c^2}{N^2} + r + 1}$$

$$\frac{4c}{N}(r+1)^2 + (r+1)^4 > 4(r+1)$$

$$\frac{4c}{N} + r^3 + 3r^2 + 3r > 3$$

(always positive for $c, N \ge 0$) $+ (> 3$ for $r \ge 1) > 3$

SL has a greater execution time for all $c, N \ge 0$ and $r \ge 1$

Therefore, by taking advantage of the overlap ready layout of the Square-Corner partitioning shape, 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 sub-section, $P1$, 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 with this algorithm is the same as (7) 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,

$$T_{exe} = \max(\max(\max(2Nq\beta - 2q^2\beta, 2q^2\beta), \frac{N(N-q)^2}{Sp})$$
$$+ 2\frac{Nq(N-q)}{Sp}, \max(2Nq\beta - 2q^2\beta, 2q^2\beta) + \frac{Nq^2}{Sq}) \quad (13)$$

Theorem 6.1 (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 similar to the proof for SCB and follows the same techniques. The full proof may be found in [13].

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
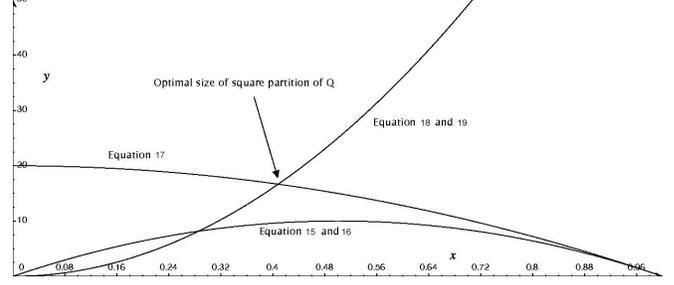
Figure 9. Graph of 5 possible parabolas for square corner partitioning with parallel communication and overlap. Equations 15 and 16, and 18 and 19 are nearly identical, respectively, and appear as a single curve. Given problem parameters N=3000, Processor Ratio = 5:1, and Communication/Computation Ratio = .05.

that q is understood to be $0 \le q \le 1$. Also introduced is the variable c, given by $c = Sp * \beta$, which represents the ratio between computation and communication speeds.

$$\frac{T_{exe}}{N^3\beta} = \max(\max(\max(\frac{2q}{N} - \frac{2q^2}{N}, \frac{2q^2}{N}), \frac{(1-q)^2}{c})$$
$$+ 2\frac{(q-q^2)}{c}, \max(\frac{2q}{N} - \frac{2q^2}{N}, \frac{2q^2}{N}) + \frac{rq^2}{c}) \quad (14)$$

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 (14). These functions and what they represent are as follows,

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

$$y = \frac{2q^2}{N} + 2\frac{(q-q^2)}{c} : VQ + (P2 + P3) \quad (16)$$

$$y = \frac{(1-q)^2}{c} + 2\frac{(q-q^2)}{c} : P1 + (P2 + P3) \quad (17)$$

$$y = \frac{2q}{N} - \frac{2q^2}{N} + \frac{rq^2}{c} : VP + Q \quad (18)$$

$$y = \frac{2q^2}{N} + \frac{rq^2}{c} : VQ + Q \quad (19)$$

Both (15) and (16) are less than (17) on the interval $\{0, \frac{1}{\sqrt{2}}\}$, and can be safely ignored. Of the remaining 3 equations, (17) is concave down and both (18) and (19) are concave up on the interval. The optimal value of q, the minimum, is therefore at the intersection of (17), and whichever other function dominates. For $q < \frac{1}{2}$, (18) dominates and for $q > \frac{1}{2}$ (19) dominates. We have already established that Square-Corner is optimal for ratios greater than $2 : 1$ using parallel communication. Ratios less than and equal to $2 : 1$, will have q values greater than $\frac{1}{2}$, so the

optimal value of q for the comparison is at $(17) \cap (19)$.

$$17 \cap 19$$
$$\frac{(1-q)^2}{c} + 2\frac{(q-q^2)}{c} = \frac{2q^2}{N} + \frac{rq^2}{c}$$
$$q = \frac{1}{\sqrt{r + 1 + \frac{2c}{N}}}$$

Theorem 6.2 (PCB): For parallel communication with a bulk overlap between communication and $P1$ computation, the Square-Corner partitioning shape is optimal, having 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 $x = \frac{1}{r+1}$.

Straight-Line Execution > Square-Corner Execution

$$\frac{1}{N} - \frac{x}{N} + \frac{1-x}{c} > \frac{1-q^2}{c}$$
$$q^2 + \frac{c}{N} > x + \frac{cx}{N}$$
$$\frac{1}{r + 1 + \frac{2c}{N}} + \frac{c}{N} > \frac{1}{r+1} + \frac{c}{N(r+1)}$$
$$1 + \frac{c(r + 1 + \frac{2c}{N})}{N} > \frac{r + 1 + \frac{2c}{N}}{r+1} + \frac{c(r + 1 + \frac{2c}{N})}{N(r+1)}$$
$$\frac{cr^2}{N} + \frac{cr}{N} + \frac{2c^2r}{N^2} > \frac{2c}{N}$$
$$r + 1 - \frac{2}{r} > -\frac{2c}{N}$$

is ≥ 0 when $r \geq 1 >$ is < 0

Therefore, for all $c, N > 0$ and $r \geq 1$, the Square-Corner partitioning shape 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 entire 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,

$$T_{exe} = 2\beta q + (N-1) \times \max(2\beta q, \frac{N^2 - q^2}{Sp}, \frac{q^2}{Sq})$$
$$+ \max(\frac{N^2 - q^2}{Sp}, \frac{q^2}{Sq}) \quad (20)$$

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,

$$T_{exe} = N\beta + (N-1) \times \max(N\beta, \frac{N(N-x)}{Sp}, \frac{Nx}{Sq})$$
$$+ \max(\frac{N(N-x)}{Sp}, \frac{Nx}{Sq}) \quad (21)$$

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 7.1 (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. The proof of this is similar to the proof for SCB and follows the same techniques. The full proof may be found in [13].

Theorem 7.2 (PIO): For parallel interleaving overlap, Square-Corner is optimal for computational power ratios, r, greater than $3 : 1$, and Straight-Line is optimal for ratios less than $3 : 1$.

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,

$$T_{exe}, SC = \frac{2q}{N} + \frac{1 - q^2}{c} \quad (22)$$
$$T_{exe}, SL = \frac{1}{N} + \frac{1 - x}{c} \quad (23)$$

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

$$SL > SC$$

$$\frac{1}{N} + \frac{1-x}{c} > \frac{2q}{N} + \frac{1-q^2}{c}$$

$$\frac{1}{N} + \frac{1-(\frac{1}{r+1})}{c} > \frac{2(\frac{1}{\sqrt{r+1}})}{N} + \frac{1-(\frac{1}{\sqrt{r+1}})^2}{c}$$

$$1 > 2(\frac{1}{\sqrt{r+1}})$$

$$r+1 > 4$$

$$r > 3$$

Therefore, when c is such that communication dominates, Straight-Line is optimal for ratios less than $3 : 1$, and Square-Corner is optimal for ratios greater than $3 : 1$.

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

$$T_{exe}, SC = \frac{2q}{N^2} + \frac{1-q^2}{c} \tag{24}$$

$$T_{exe}, SL = \frac{1}{N^2} + \frac{1-x}{c} \tag{25}$$

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

$$SL > SC$$

$$\frac{1}{N^2} + \frac{1-x}{c} > \frac{2q}{N^2} + \frac{1-q^2}{c}$$

$$\frac{1}{N^2} + \frac{1-(\frac{1}{r+1})}{c} > \frac{2(\frac{1}{\sqrt{r+1}})}{N^2} + \frac{1-(\frac{1}{\sqrt{r+1}})^2}{c}$$

$$1 > 2(\frac{1}{\sqrt{r+1}})$$

$$r+1 > 4$$

$$r > 3$$

Therefore, when c is such that computation dominates, Straight-Line is optimal for ratios less than $3 : 1$ and Square-Corner is optimal for ratios greater than $3 : 1$. ∎

VIII. THREE AND MORE PROCESSORS

While a full study of our mathematical technique on three and more processors is outside the scope of this work, some quick and simple extensions to the two processor case will yield both three processor, and arbitrary number of processor, optimal partitions for some performance values. Although traditional algorithms partition data for arbitrary numbers of processors into rectangles, it is proved here that, in general, the optimal solution can be non-rectangular.

The optimal partition shape for three processors is non-rectangular for many performance characteristic values. Consider two processors, one fast and one slow, such that the ratio between their computational power, r, is greater than 3. It has already been shown for all 5 MMM algorithms

the Square-Corner is the optimal partition shape in this situation. Now consider a second slow processor, also having a ratio greater than 3 with the fast processor. The optimal, minimum, amount of communication is given by the Square-Corner partition if this second slow processor can be added to the two processors already partitioned without increasing communication above a Square-Corner partition with just the fast processor and the second slow processor. This means if we can guarantee that the two slower processors won't need to communicate, *i.e.* their ratios are such that they don't overlap as in Figure 10, we have optimal partitioned all three processors. The formal proof of this is presented below.
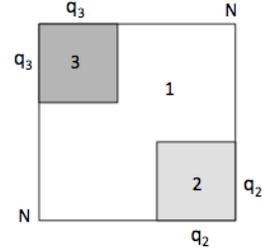
Figure 10.   Layout of 3 processors of ratios $r > 3$, optimally partitioned.

Theorem 8.1 (3 Processors): Square-corner partitioning minimizes communication time between 3 processors in partition φ, if both the ratio between processors 1 and 2, and the ratio processors 1 and 3, are greater than $3 : 1$.

Proof: Consider 3 processors in a fully connected topology. Their ratios are as follows:

$$r_{12} > 3 = \frac{S_1}{S_2} : \text{ ratio of processor 1 to processor 2}$$

$$r_{13} > 3 = \frac{S_1}{S_3} : \text{ ratio of processor 1 to processor 3}$$

We define the communication time between these processors as:

$$Tcomm, serial = T_{12} + T_{13} + T_{23}$$

$$Tcomm, parallel = max(T_{12}, T_{13}, T_{23})$$

$$\text{where, } T_{12} = \text{Comm time for Processors 1 and 2}$$

$$T_{13} = \text{Comm time for Processors 1 and 3}$$

$$T_{23} = \text{Comm time for Processors 2 and 3}$$

We minimize both $Tcomm, serial$ and $Tcomm, parallel$ by minimizing all 3 terms, T_{12}, T_{13}, T_{23}. We already know that using the Square-Corner partition will minimize T_{12}, T_{13}. So we simply define the ratios of Processor 2 and 3 such that they can fit in opposite corners without overlapping. Therefore, they do not need to communicate and $T_{23} = 0$, and is thereby minimized. The ratios to achieve

this are defined as follows:

$$r_2 = \frac{S_1 + S_3}{S_2} \quad : \text{ratio of Processor 2 to rest of the partition}$$

$$r_3 = \frac{S_1 + S_2}{S_3} \quad : \text{ratio of Processor 3 to rest of the partition}$$

$$q_2 = \frac{N}{\sqrt{r_2 + 1}} \quad : \text{size of side of square for Processor 2}$$

$$q_3 = \frac{N}{\sqrt{r_3 + 1}} \quad : \text{size of side of square for Processor 3}$$

Since $r_{12} > 3$ and $r_{13} > 3$,

then by definition $r_2 > 3$ and $r_3 > 3$

therefore, $q_2 < \dfrac{N}{2}$ and $q_3 < \dfrac{N}{2}$

and since $q_2 + q_3 < N$

there is a position in which 2 and 3 will not overlap

■

This concept also applies to n processors, so long as their ratios are such that the sum of their q lengths is $\leq N$, *i.e.* so long as the smaller partitions will not overlap with each other, as seen in Figure 11. For space considerations, the full proof is not presented here, but it is similar to the above 3 processor proof, showing that for the given ratios the optimal partition shape for any number of processors is non-rectangular.
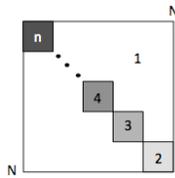
Figure 11.   Layout of n processors, optimally partitioned.

IX. EXPERIMENTAL RESULTS

To validate these theoretical results, we provide some simple experimental results. First experiments on a small cluster using individual processors for each MMM algorithm in the two processor case are described. Next to be presented are the results for three processors on the same small cluster using individual processors. Finally, are some larger scale experiments on Grid'5000, using an entire cluster as a "processor".

A. Experimental Setup

To corroborate the results, we implemented the Square-Corner and Straight-Line algorithms using MPI. The local matrix multiplications use ATLAS [14]. All experiments were carried out on two identical machines. The ratio of computational speeds between the two nodes was altered by slowing down a single node using a program for limiting CPU time available to a process. This program forces a given

process to sleep when a specified percentage of CPU time has been achieved, using the /proc filesystem — the same information available to a program like top. When enough time has passed, the process is woken and runs normally. This provides fine grained control over the CPU power available to each process. These results were achieved on two identical Dell Poweredge 750 machines with 3.4 Xeon processors, 1 MB L2 cache and 256 MB of RAM.

It is important to note that because varying processor power ratios was achieved by lowering the capacity of a single processor, the higher the ratio, the lower the overall computational power.

B. Serial Communication with Barrier

When running with a barrier between communication and computation, we focus on the communication time, as we expect the Square-Corner to have a lower total volume of communication for power ratios greater than $3 : 1$. In Figure 12 we present the theoretical curves for the communication time for both Square-Corner and Straight-Line, for comparison to the experimentally found results. We can see the constant volume of communication for Straight-Line, and the exponentially decreasing communication volume for Square-Corner.
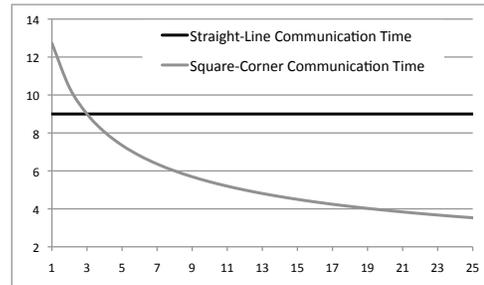
Figure 12.   Straight-Line and Square-Corner theoretical serial communication time using the SCB algorithm.

We ran both the Square-Corner and Straight-Line partitioning schemes for computational power ratios, r, from 1 to 25. In Figure 13 we show the comparison of the
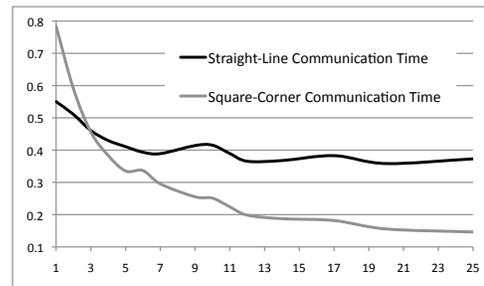
Figure 13.   Straight-Line and Square-Corner serial communication time, using SCB, in seconds by power ratio. N=3000.

communication times. The results follow the theory, with

the communication times crossing at $r = 3 : 1$, and with Square-Corner vastly improving on the Straight-Line communication time for the larger ratios.

C. Parallel Communication with Barrier

For PCB we also focus on the communication time. The theoretical results suggest that Square-Corner should best Straight-Line's communication time when $r > 2 : 1$. In Figure 14 we depict the theoretical curves for both Straight-Line and Square-Corner communication times.
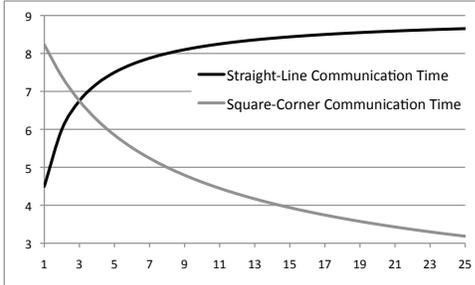
Figure 14. Square-Corner and Straight-Line theoretical parallel communication times using PCB.

The experimental results for ratios, r, from 1 to 25 can be seen in Figure 15. As expected for small ratios, Straight-Line is optimal. As the ratio grows, the communication time drops significantly for the Square-Corner, making it the optimal solution.
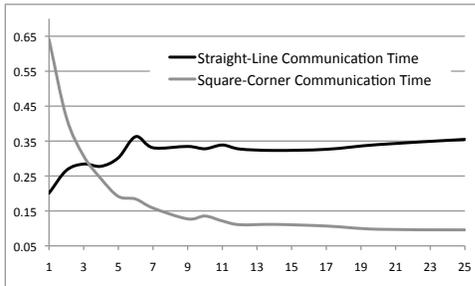
Figure 15. Square-Corner and Straight-Line parallel communication times, using PCB, in seconds by power ratio. N=3000.

D. Serial Communication with Bulk Overlap

Using SCO, the theory predicts that the Square-Corner should have a lower execution time for all power ratios, r. The experiments match this result. For small ratios, $r < 3$:!, where previously Straight-Line performed significantly better, the Square-Corner now has the better execution time. The overlap has allowed the Square-Corner partition to overtake the Straight-Line in those ratios where it originally performed worse, so it is optimal for all ratios, r. The SCO results for the small ratios, $r < 3 : 1$, can be seen in Figure 16. For larger ratios, where Square-Corner had
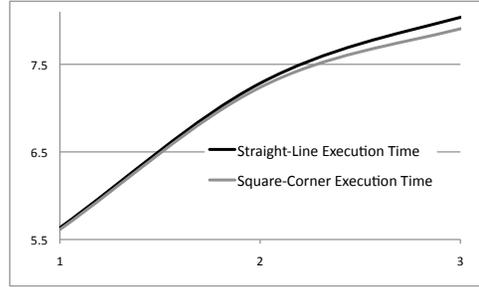
Figure 16. Square-Corner and Straight-Line execution times, using SCO, for small ratios $r < 3 : 1$, in seconds by power ratio. N=3000.

already outperformed the Straight-Line when using SCB, by using SCO the Square-Corner partition increases its lead over the Straight-Line partition. The experimental results for ratios larger than three can be found in Figure 17.

Figure 17. Square-Corner and Straight-Line execution times, using SCO, for large ratios $r > 3 : 1$, in seconds by power ratio. N=3000.

E. Parallel Communication with Bulk Overlap

The theoretical results predicted that Square-Corner should be optimal for all power ratios when using PCO. Recall when using PCB that Straight-Line was optimal for ratios two and under by a significant margin. Using PCB on the Square-Corner partition has closed that gap. In Figure 18, the result for ratios $r \leq 3 : 1$ are shown. The parallel communication aspect of the PCO algorithm gives less time to compute the overlapped portion, therefore it would be expected that less speedup can be gained while using PCO than SCO. However, using PCO the Square-Corner partition still manages to outperform the Straight-Line. As the ratio increased between the processors, the benefit of overlapping communication and computation became more marked. In Figure 19, we show the results from $r \geq 4 : 1$ as Square-Corner outperforms Straight-Line as expected..

F. Parallel Interleaving Overlap

The experimental results for PIO support the theoretical results that Square-Corner partitioning has a lower execution time for power ratios, r, greater than three. For ratios smaller than that, the Straight-Line partition has the lower execution time.
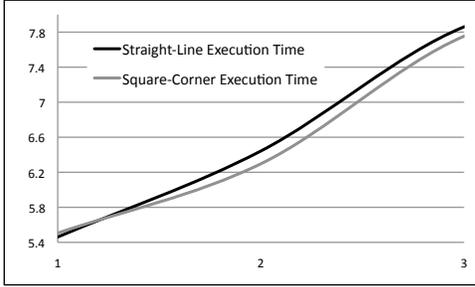
Figure 18. Square-Corner and Straight-Line execution times, using PCO, for ratios $r \leq 3 : 1$ in seconds by power ratio. N=3000.
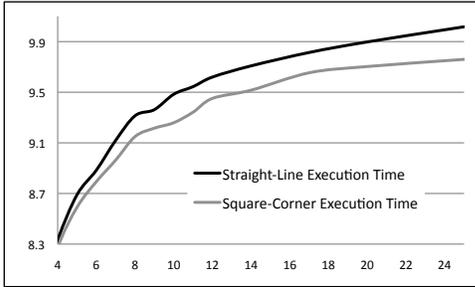
Figure 19. Square-Corner and Straight-Line execution times, using PCO, for ratios $r \geq 4 : 1$ in seconds by power ratio. N=3000.
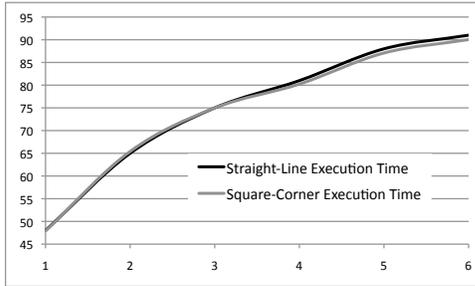
Figure 20. Square-Corner and Straight-Line execution times, using PIO, when the communication-compuation ratio, c, is such that computation dominates for all ratios in seconds by power ratio. N=3000. The two partitions are equivalent.

G. Three Processors - PCB and PCO

The three processor case has been explored for the PCB and PCO algorithms. These results were obtained on the same cluster as the previous two processor experiments. As displayed in the theory section, a fully connected network topology is used. For simplicity, the relative speeds of the two smaller processors, S2 and S3, are assumed to be equal. The sum of all speeds, S1, S2 and S3 is taken to be 100. The graphs are presented with relative speed of S2, S3 on the x-axis; the relative speeds range from 5 to 25. When the relative speed is 25, then S2 + S3 = 50 and since S1 = 100 - S2 + S3, this means a ratio of $50 : 50$ or $1 : 1$. We also note that the often important ratio of $3 : 1$ would be a relative speed, on these graphs, of 12.5. In other words,
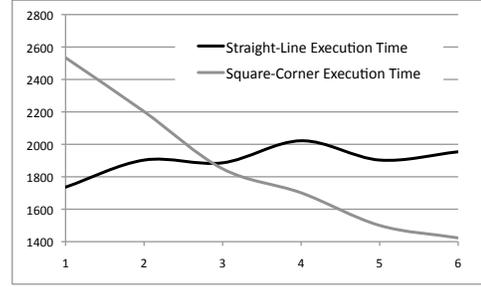
Figure 21. Square-Corner and Straight-Line execution times, using PIO, when the communication-compuation ratio, c, is such that communication dominates for all ratios in seconds by power ratio. N=3000. The Straight-Line partition is superior for power ratios, r, less than $3 : 1$, and Square-Corner partition is superior for power ratios, r, greater than $3 : 1$.

heterogeneity, the ratio of computational power, decreases moving along the graph left to right.
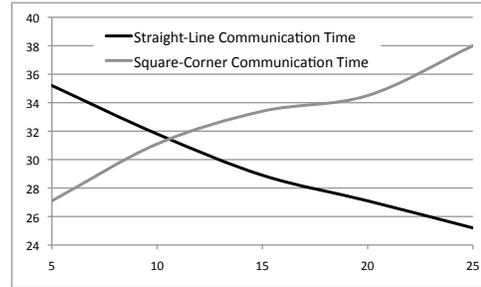
Figure 22. Square-Corner and Straight-Line communication times for 3 processors, using PCB, in seconds by relative speed of S2, S3. Note that for more heterogeneous environments, on the left of the graph, the Square-Corner has the lower communication time. The lines cross at approximately 12.5, which is equivalent to a $3 : 1$ ratio in the two processor case. N=5000.
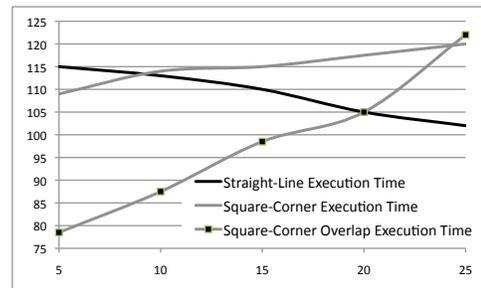
Figure 23. Square-Corner and Straight-Line execution times for 3 processors, using PCB, in seconds by relative speed of S2, S3. For the more heterogeneous environments, on the left of the graph, the Square-Corner is optimal. For less heterogeneous environments, the rectangular partition is optimal. Also, the Square-Corner execution time for 3 processors using PCO is optimal for a wider range of ratios than the regular Square-Corner using PCB. N=5000.

H. Grid'5000 Experimental Results

For further validation, we include more substantial experiments performed on Grid'5000, a large grid system

spread across nine sites in France [15]. These results are obtained on two types of machines in Bordeaux. The first are IBM x4355 dual-processor nodes with AMD Opteron 2218 2.6GHz Dual Core Processors with 2MB L2 Cache and 800MHz Front Side Bus. Each node has 4GB of RAM (2GB per processor, 1GB per core). The second are IBM eServer 325 dual-processor nodes with AMD Opteron 248 2.2GHz single core processors with 1MB L2 Cache. Each node has 2GB of Ram (1GB per processor). The problem size is $N = 15,000$. Unlike the earlier experiments, where total computational power decreased with higher ratios, these experiments attempt to have a similar computation power across all ratios. The computational power ratios are varied from $1:1$ to $1:8$ by varying the number of CPUs in each cluster.
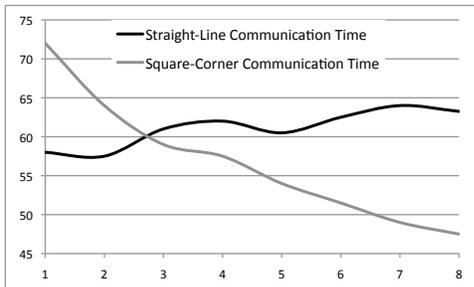
Figure 24. Square-Corner and Straight-Line communication times on Grid'5000 for computational power ratios, r, from $1:1$ to $1:8$. Network bandwidth is 1Gb/s and $N = 15000$.
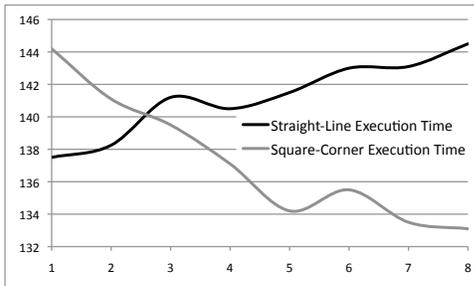
Figure 25. Square-Corner and Straight-Line execution times on Grid'5000 for computational power ratios, r, from $1:1$ to $1:8$. Network bandwidth is 1Gb/s and $N = 15000$.

X. CONCLUSIONS AND FUTURE WORK

We have found the general optimal solution for all matrix-matrix multiplication problems involving two processors. For two processors with a barrier between communication and computation, the Square-Corner partitioning is optimal for ratios greater than $3:1$ with serial communication and greater than $2:1$ for parallel communication. The Straight-Line partitioning is optimal for ratios less than $3:1$ for serial communication and less than $2:1$ for parallel communication. However, if we overlap the

computation that can be done immediately on a Square-Corner partition with the communication, the Square-Corner partitioning is optimal for all power ratios of processors and communication/computation ratios for both serial and parallel communication.

We have also found a solution for matrix-matrix multiplication with n processors under certain conditions of relative power ratios between all the processors. If the power ratios of each processor with the fastest processor, processor 1, are greater than $3:1$, and the other $n-1$ processors can be arranged as non-overlapping squares, then the square-corner partitioning will optimize communication time between the n processors.

Throughout this work we have made various assumptions that limit the complexity of our model. We use square matrices, instead of arbitrary dimensions. We also use a single bandwidth characteristic and ignore latency, not considering asymmetric communication. Finally, we assume that all processors have sufficient memory to store the assigned portions of matrix C and necessary portions of matrices A and B to compute. Further work should be done to expand the model to include these factors. For instance, in the case of limited memory, which is likely on a GPU machine, the algorithm would need to include the additional communication of many smaller sections of the assigned portion of the matrix C which will fit into local memory. The ramifications of this algorithmic change should be explored further.

Most importantly, however, we have found in this work that the optimal partition shape is often non-rectangular. For many heterogeneous systems, the current data partitioning algorithms have been solving the wrong problem by finding the best rectangular solution; the algorithms should be tuned to find the non-rectangular solution when it is appropriate. A portion of this open problem, the two processor case, has now been thoroughly studied, however substantial work remains to be done to solve the optimal partition shape problem in general. The mathematical technique developed in this paper will be applied to this case of an arbitrary number of processors to solve this problem in the future.

REFERENCES

[1] 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.

[2] A. Lastovetsky and J. Dongarra, *High-performance heterogeneous computing*. Wiley, 2009.

[3] 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 Europe'99)*, vol. 1593, 1999, pp. 191–200.

[4] R. A. Van De Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, pp. 225–274, 1997.

[5] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, "Heterogeneous matrix-matrix multiplication or partitioning a square into rectangles: Np-completeness and approximation algorithms," in *Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing (PDP 2001)*, 2001, pp. 298 –305.

[6] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 520–535, 2001.

[7] E. Dovolnov, A. Kalinov, and S. Klimov, "Natural block data decomposition for heterogeneous clusters," in *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, April 2003.

[8] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-matrix multiplication on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1033–1051, 2001.

[9] 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.

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

[11] B. A. Becker and A. Lastovetsky, "Towards data partitioning for parallel computing on three interconnected clusters," in *Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007)*, IEEE Computer Society. Hagenberg, Austria: IEEE Computer Society, 5-8 July 2007, pp. 285–292.

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

[13] A. DeFlumere and A. Lastovetsky, "Theoretical results on optimal partitioning for matrix-matrix multiplication with two processors," School of Computer Science and Informatics, University College Dublin, Tech. Rep. UCD-CSI-2011-09, September 2011.

[14] R. Whaley and J. Dongarra, "Automatically tuned linear algebra software," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (SC98)*. IEEE Computer Society, 1998, pp. 1–27.

[15] [Online]. Available: http://www.grid5000.fr

Ashley DeFlumere was born in Boston, Massachusetts, in 1986. She received the B.A. degree in computer science from Mount Holyoke College, M.A., U.S.A., in 2009. She is currently a PhD student in the School of Computer Science and Informatics at University College Dublin (UCD), Ireland.

She was previously an intern at the Oak Ridge National Laboratory in T.N., U.S.A., where she was involved in performance evaluation research on multi-core platforms. Her main research interests are models and algorithms for high performance heterogenous and parallel computing.

Alexey Lastovetsky received a PhD degree in 1986 and a Doctor of Science in 1997. His main research interests include algorithms, models and programming tools for high performance heterogeneous computing. He published over 100 technical papers in refereed journals, edited books and proceedings of international conferences. He authored the monographs "Parallel computing on heterogeneous networks" (Wiley, 2003) and "High performance heterogeneous computing" (with J. Dongarra, Wiley, 2009).

He is currently a senior lecturer in the School of Computer Science and Informatics at University College Dublin (UCD). At UCD, he is also the founding Director of the Heterogeneous Computing Laboratory (http://hcl.ucd.ie/).

Brett Becker was born in New Jersey in 1976. He received the B.A. degree in computer science and the B.A. degree in physics, both from Drew University, N.J., U.S.A., in 2003. He received the M.Sc. degree in computational science and the Ph.D. degree in computer science from University College Dublin, Ireland, in 2004 and 2011, respectively.

He lectured in computer science at Griffith College Dublin, Ireland, for six years where he is involved in pedagogical research. He has helped organize the International Conference on Engaging Pedagogy since 2008, having served as co-chair in 2011, and is currently a member of the steering committee. His main areas of research interest are science education and high performance heterogeneous and parallel computing.