Matrix Multiplication on Two Interconnected Processors

Brett A. Becker
School of Computer Science and Informatics
University College Dublin (UCD), Belfield,
Dublin 4, Ireland
brett.becker@ucd.ie

Alexey Lastovetsky
School of Computer Science and Informatics
University College Dublin (UCD), Belfield,
Dublin 4, Ireland
alexey.lastovetsky@ucd.ie

Abstract

This paper presents a new partitioning algorithm to perform matrix multiplication on two interconnected heterogeneous processors. Data is partitioned in a way which minimizes the total volume of communication between the processors compared to more general partitionings, resulting in a lower total execution time whenever the power ratio between the processors is greater than 3:1. The algorithm has interesting and important applicability, particularly as the top-level partitioning in a hierarchal algorithm that is to perform matrix multiplication on two interconnected clusters of computers.

1. Introduction

This paper presents a new algorithm specifically designed to perform matrix multiplication on two interconnected heterogeneous processors. Data is partitioned in a way which minimizes the total volume of communication between the processors resulting in a lower execution time compared to more general partitionings, whenever the power ratio between the processors is greater than 3:1. A hybrid algorithm utilizing an existing partitioning algorithm for ratios less than 3:1 would guarantee performance equal to or better than known partitionings. Minimizing the total volume of communication is a natural goal as matrix multiplication involves substantial communication volumes and the link connecting the processors is a possible bottleneck. This partitioning strategy could also decrease the communication volume of other linear algebra kernels and applications which are communication intensive.

Our motivation stems from the fact that there are many general algorithms which work well for several, dozens, or even hundreds of nodes, but all result in a straight-forward partitioning when applied to the architecture of two interconnected processors. Examples of these methods are explored in [2][3][5][7]. The same partitioning occurs when any single or multidimensional partitioning strategy is applied to two processors. The result sees the matrix partitioned into two rectangular areas divided by a straight line, each proportional in area to the computational power of the processor which is to compute the partition. We refer to this as a 'straight-line' partitioning.

Our algorithm differs in that the matrix is not partitioned with a straight line. We create two partitions, one partition is a square located in a corner of the matrix, and the other partition is polygonal – the rest of the matrix with the square removed from a corner. This 'square-corner' partitioning has the effect of lowering the total volume of inter-processor communication and therefore the total execution time, whenever the power ratio between the processors is greater than 3:1.

As noted in [3], matrix multiplication is the prototype for a group of tightly-coupled kernels that should be efficiently solved on high performance computing architectures. To our knowledge no research has been conducted to optimize matrix multiplication for the specific architecture of two connected heterogeneous processors. The most related work is [3], which used this architecture to test a more general algorithm designed for any number of processors including two.

This lack of research may be due to the fact that developing special strategies to optimize particular algorithms for two processors has historically not been seen as worth the effort. Why run a parallel task on only two machines when powerful computers are relatively inexpensive and a general purpose parallel algorithm will run on dozens or hundreds of nodes? We point out that such "two-processor" algorithms can have useful applicability. One such application is performing the top-level partitioning between two connected clusters – a natural architecture for high

performance computing. With such an architecture, each cluster (containing any number of nodes internally) can be of considerable computational power and well worth using in parallel. At a high level such collections of clusters can be viewed as a collection of individual processors, and after an initial top-level partition utilizing this new algorithm, each cluster can then deal with its data partition using an algorithm that will most efficiently exploit its particular local architecture.

The rest of this paper is organized as follows. Section 2 introduces related research involving matrix multiplication on two processors. Section 3 introduces our 'square-corner' partitioning algorithm and its constraints, and proves its optimality amongst other variants of the algorithm. We then theoretically compare the square-corner algorithm with the 'straight-line' partitioning which results when more general algorithms are applied to the two-processor architecture. We then show that the square-corner algorithm minimizes the inter-processor volume of communication when the processor power ratio is greater than 3:1. We also compare this algorithm with the lower bound presented in [3] and show that the square-corner algorithm approaches that lower bound. In Section 4 we provide results of MPI Experiments running the square-corner algorithm on two processors of varying power ratios and link bandwidths. Our results demonstrate both a lower total volume of communication and total execution time than the more general straight-line algorithm in all cases where the power ratio is greater than 3:1. In Section 5 we give our concluding remarks and an indication of future work. Section 6 includes acknowledgements.

2. Related Work

To date, very little research has been concentrated on matrix multiplication on a two processor architecture. In [3], Beaumont et al. target matrix multiplication on heterogeneous platforms. A column based partitioning based on that of [7] is introduced which balances the workload between processors of different speeds in an attempt to minimize the total volume of communication. First the matrix is partitioned into rectangles proportional in area to the speed of each processor. These rectangles are then arranged into columns in a defined manner. The total volume of communication is proportional to the sum of the half-perimeters s of each rectangle, given by (2.1), where p is the number of processors, and h_i and w_i are the height and width of the rectangle assigned to processor i, respectively.

$$s = \sum_{i=1}^{p} (h_i + w_i) \tag{2.1}$$

Since the perimeter of any rectangle enclosing a given area is minimized when that rectangle is a square, there is a natural lower bound l of (2.1), shown by (2.2), where a_i is the area of the partition belonging to processor i.

$$l = 2 \times \sum_{i=1}^{p} \sqrt{a_i} \tag{2.2}$$

The authors then carry out a simulation which takes a large number of randomly generated rectangular partition areas and compare their partitioning algorithm's sum of half-perimeters with the lower bound. They do this for a number of processors (and therefore rectangles) ranging from one to 40. Their partitioning algorithm performs well, with the worst average sum of half-perimeter to lower bound ratio being about 1.11, for the case of two processors.

The authors state that the lower bound can not always be met and use the case of two processors as an example. They ask the reader to consider the case of two processors with relative speeds such that processor 1 receives a rectangle of area $a_1 = 1 - \varepsilon$, and processor 2 receives a rectangle of area $a_2 = \varepsilon$, where $\varepsilon > 0$, is an arbitrarily small number. In order to partition the unit matrix into two rectangles, a line of length 1 must divide the matrix. Using (2.1), this results in a sum of half-perimeters equal to 3, but (2.2) shows that the lower bound can get arbitrarily close to 2. Substituting N^2 for 1 (generalizing on the unit square), we see that in the case of two processors as $\varepsilon \rightarrow 0$ the lower bound gets arbitrarily close to $2 \times N$, which is the half-perimeter of the matrix itself.

3. Matrix Multiplication on Two Heterogeneous Processors

3.1 Minimizing the Total Volume of Communication

The main objective of our algorithm is to minimize the total volume of communication between two connected processors that are to in parallel perform matrix multiplications. The simplest partitioning of a matrix multiplication $C = AB$ divides the C matrix in half. On two homogeneous processors, this will perfectly balance the load as each processor will receive equal amounts of work and should finish their respective jobs simultaneously. For two heterogeneous processors, the C matrix is divided into two rectangles, each having an area proportional to the speeds of the nodes. Again, this theoretically results in a perfect load

balance. In both cases the total volume of communication is N^2, as each node needs the entire data partition from the other node to compute its part of the C matrix.

Figure 1 shows a one-dimensional partitioning scheme that guarantees perfect load balancing between four homogeneous processors. In order to compute its partition of the C matrix, processor p1 needs to receive p2, p3, and p4's partitions of A. This results in a volume of communication equal to $3\times N^2/4$. All other nodes similarly need every other node's partition of A to compute their partition's product. Therefore the total volume of communication is equal to $3\times N^2$.

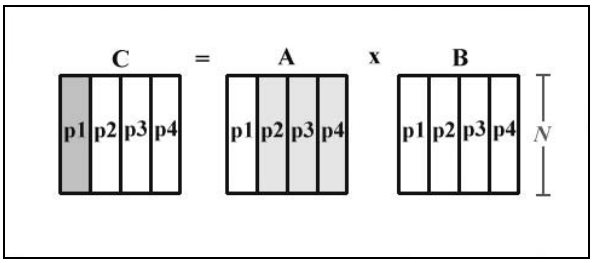

Figure 1. A one-dimensional homogeneous partitioning scheme.

Figure 2 shows a two-dimensional partitioning for the same problem and the same four homogeneous processors as Figure 1. Each processor has a partition of C proportional in area to its speed, but the two-dimensional partitioning results in a lower total volume of communication. In order to calculate its partition of C, processor p1 needs to receive p2's partition of A, and p3's partition of B. This is a volume of communication equal to $N^2/2$. Similarly, each other processor needs to receive the equivalent partitions from its neighboring processors, resulting in a total volume of communication equal to $2\times N^2$. Thus, the two-dimensional partitioning reduces the total volume of communication by 1/3.

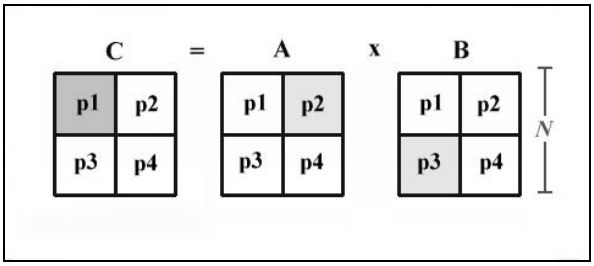

Figure 2. A two-dimensional homogeneous partitioning scheme.

The two-dimensional partitioning also reduces the total volume of communication on heterogeneous platforms, the difference being that the rectangles are not of equal size, as the area of each partition must be proportional to processor speed. For more information, see [1].

Clearly a traditional two-dimensional partitioning makes no sense on an architecture that is comprised of only two processors. This does not mean that a simple one-dimensional rectangular partitioning is optimal however. Figure 3 shows the partitioning scheme used by our 'square-corner' algorithm. The total volume of communication can be reduced from that of the more general straight-line partitioning by avoiding the use of a straight line to partition the matrix. Instead the slower processor is allocated a square partition of size $q\times q$, and the faster processor receives the balance of the matrix. The size of q is dictated by the ratio of processor powers r. We always normalize this ratio so that the speed of the slower processor is equal to 1, so a ratio of r is understood to be a ratio of r:1. Therefore q is given by (3.1).

$$q = \frac{N}{\sqrt{r+1}} \tag{3.1}$$

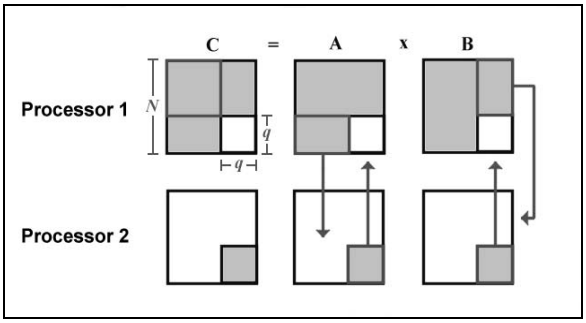

Figure 3. The square-corner partitioning and communication steps.

As shown in Figure 3, the necessary communications involve processor 1 sending two pieces of size $q\times(N-q)$ to processor 2, and processor 2 sending two pieces of size $q\times q$ to processor 1. This results in the following equation for the total volume of communication c.

$$c = 2\times N\times q \tag{3.2}$$

The straight-line partitioning always results in a total volume of communication equal to N^2, regardless of the power ratio.

The location of the square partition does not affect the total volume of communication. A corner is chosen to minimize the number of communication steps necessary. As shown in Figure 3 this number is four. Placing the square adjacent to an edge but not in a corner requires five communication steps. This is because the $q\times q$ square would interrupt either a set of

columns or rows of width or height q into two pieces, each of which need to be communicated. Placing the square inside the matrix, not adjacent to any edge requires six communication steps. This is due to the $q \times q$ square interrupting a set of columns of width q into two pieces, and likewise for a set of rows of height q.

Theorem: For all power ratios r greater than 3, the square-corner total volume of communication, equal to $2 \times N \times q$, will be less than that of the straight-line partitioning which is always equal to N^2.

Proof: We start by stating that the square-corner total volume of communication is less than that of the straight-line partitioning: $2 \times N \times q < N^2$ subject to the conditions $N, q > 0$. Substituting equation (3.1), yields $\frac{2 \times N^2}{\sqrt{r+1}} < N^2$ which simplifies to $2 < \sqrt{r+1}$, and $4 < r+1$, and finally $r > 3$. □

Similar proofs show that for the power ratio $r = 3$ (3:1), the square-corner total volume of communication is exactly equal to the straight-line total volume of communication, and for ratios where $r < 3$, the square-corner total volume of communication exceeds that of the straight-line partitioning.

Figure 4 shows the total volume of communication of the square-corner partitioning compared to that of the straight-line partitioning. It is clear that for $r = 3$, the values are equal. For $r = 15$, the square-corner total volume of communication is exactly half that of the straight-line algorithm.

An additional advantage to the square-corner partitioning is that a large area of the matrix product can be computed by processor 1 without any communications taking place. This immediately calculable area of size $N\text{-}q \times N\text{-}q$ can be seen at the top left corner of processor 1's C matrix in Figure 3. On architectures with a dedicated communication subsystem, this property could be exploited to overlap some communication and computation which would reduce the total execution time further.

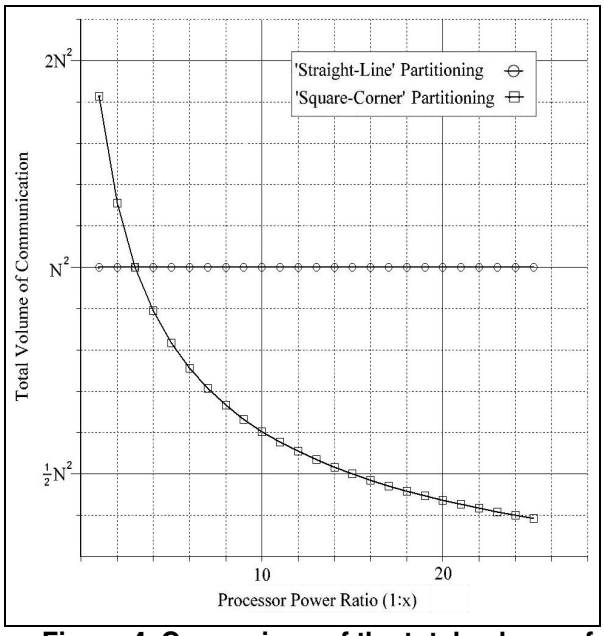

Figure 4. Comparison of the total volume of communication for the square-corner and straight-line partitionings.

3.2 Optimality of the Square-Corner Partitioning

In this section we prove that as presented, the square-corner algorithm minimizes the total volume of communication against variants of the algorithm. Possible variants include assigning non-square partitions to processor 2. This means relaxing the $q \times q$ square partition of area Q in Figure 3, to become a rectangle of width x, height y, and area Q.

We wish to minimize the total volume of communication, c, which is equal to (3.3).

$$c = x \times N + y \times N \tag{3.3}$$

With the restraints shown by (3.4).

$$x \times y = Q,\ 0 < x \leq N,\ 0 < y \leq N \tag{3.4}$$

Theorem: Subject to the constraints of (3.4), $c = x \times N + y \times N$ is minimized when $x = y$, and therefore when the partition is a square.

Proof: We substitute $y = Q/x$ from (3.4) into (3.3) to get $c = x \times N + \frac{Q \times N}{x}$. We then set $\frac{\partial c}{\partial x} = N - \frac{Q \times N}{x^2}$ equal to zero which results in $Q = x^2$, and therefore $x = y$, i.e. the partition is square. The second

derivative of c, $\frac{\partial^2 c}{\partial x^2} = N + 2 \times \frac{Q \times N}{x^3}$ is positive, indicating that the zero of the first derivative is a minimum and any other partition will result in an increased total volume of communication. □

Note that even in its most extreme variation where x or $y = N$, although the partitions of the square-corner algorithm and that of the straight-line algorithm are equal, the algorithms are still different. The square-corner algorithm always involves four communications (as long as the slower processor is allocated a partition in a corner of the matrix), and the straight-line algorithm always involves two. Thus one algorithm is not a special case of the other.

3.3 Comparison with Beaumont et al. and the Lower Bound

In section 2 we summarized the work of Beaumont et al. [3], which presented a column-based algorithm for partitioning a matrix into different sized rectangles to load-balance the problem for a heterogeneous architecture and minimize the total volume of communication. For two nodes, this algorithm results in a straight-line partitioning similar to Figure 1, with two rectangles each proportional in area to the relative powers of the nodes. The sum of half-perimeters s which is proportional to the total volume of communication was given by (2.1), and in the case of two nodes, is equal to (3.5).

$$s = \sum_{i=1}^{p} (h_i + w_i) = 3 \times N \tag{3.5}$$

The lower bound of the sum of half perimeters l is given by (2.2), and for the case of two processors is equal to (3.6), where a_i is the area of the partition belonging to processor i.

$$l = 2 \times \sum_{i=1}^{p} \sqrt{a_i} = 2 \times (\sqrt{a_1} + \sqrt{a_2}) \tag{3.6}$$

In the case of two nodes, the square-corner partitioning has a sum of half perimeters equal to (3.7).

$$s = 2 \times N + 2 \times q = 2 \times (N + q) \tag{3.7}$$

Equation (3.7) shows that for the square-corner partitioning, as $q \rightarrow 0$, $s \rightarrow 2 \times N$, which is equal to the lower bound that cannot be met by the straight-line partitioning.

To compare the square-corner sum of half-perimeters with that of the straight-line partitioning and the lower bound, we adopted the same approach as in [3]. We generated 2,000,000 random values for the partition areas a_1 and a_2, and calculated values for the sum of half-perimeters s and the lower bound l. Additionally, we restricted a_1 and a_2 so that $3 < a_1/a_2$. This is because we have already shown that the straight-line partitioning has a lower sum of half-perimeters (and therefore lower total volume of communication) for $a_1/a_2 < 3$. The average sum of half-perimeter to lower bound ratio for the straight-line partitioning is 1.176, while that of the square-corner partitioning is 1.054. Considering that 1.0 is the optimum value, this is an improvement of 69%. The minimum value for the sum of half-perimeter to lower bound ratio for the straight-line partitioning is 1.098076, while that of the square-corner partitioning is 1.000001, an improvement of well over 99%. This demonstrates that the square-corner partitioning does approach the lower bound which cannot be met by the straight-line partitioning.

In generating 2,000,000 random areas, there are bound to be many that are have very large ratios, making them computationally unrealistic. Surely nobody would use two processors in parallel if one of them is slower than the other by an order of hundreds or thousands or greater. We therefore impose the tighter but more realistic restriction of $3 < a_1/a_2 \leq 100$. Even with these much tighter restrictions, the average sum of half-perimeter to lower bound ratio for the straight-line partitioning is 1.169 while that of square-corner partitioning is 1.056, an improvement of 67%. The minimum is improved from 1.098 to 1.005, a gain of 95%.

3.4 Worst case analysis

Since the ratio r plays an important role in the cost of the worst case, Beaumont et al. analyzed the maximum value of the sum of half-perimeter to lower bound ratio for different values of r ranging from 2 to ∞, for two to 40 processors. This was done by generating 10,000 values of a_i and plotting the maximum value of the sum of half-perimeter to lower bound ratio. For two processors, this value ranged from just under 1.1 for $r = 2$, to the theoretical upper limit of 1.5 for $r = \infty$.

Figure 5 shows the maximum values of the sum of half-perimeter to lower bound ratio for values of r ranging from 1 to 25.

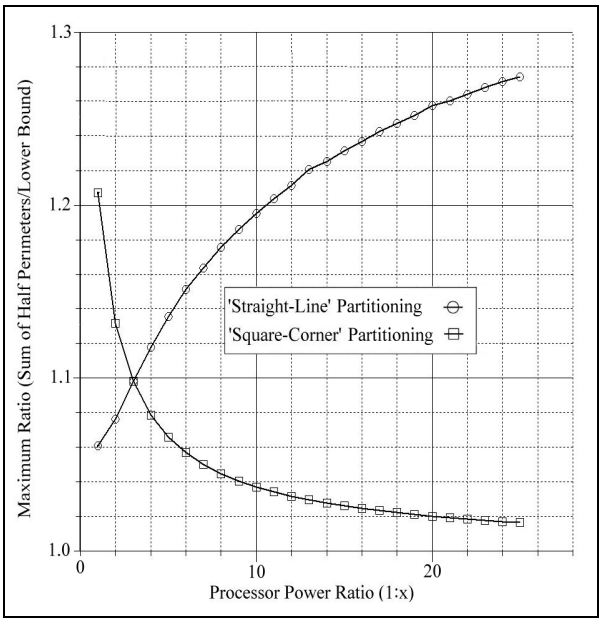

Figure 5. Comparison of the maximum s*um of half-perimeter / lower bound* values for the square-corner and straight-line partitionings.

Similar to Figure 4, Figure 5 shows that for $r = 3$, both the square-corner and straight-line partitionings are equivalent in communication volume. Considering that 1.0 is the optimal value, the square-corner maximum ratio is an average of 84% lower than that of the straight-line partitioning for the range $3 < r < 25$. As $r \to \infty$ the maximum ratio for the square-corner partitioning approaches the optimal value of 1.0, while that of the straight-line partitioning approaches the theoretical upper limit of 1.5.

4. MPI Experiments

To experimentally verify this new algorithm, we implemented the square-corner algorithm and the straight-line algorithm in Open-MPI [6]. Local matrix multiplications utilize ATLAS [8]. The experiments were carried out on two identical machines so that we could focus only on the partitioning without worrying about any contributions made by architectural differences. The machines were connected with a switch which allows the average bandwidth between the nodes to be specified. Both algorithms carry out all communications first, then all computations, hence there is no communication/computation overlap.

The ratio of speeds between the two nodes was varied by slowing down the CPU of one node using a CPU limiting program as proposed in [4]. This program supervises a specified processes and using the */proc* pseudo-filesystem, forces the process to sleep when it has used more than a specified fraction of CPU time. The process is then woken when enough idle CPU time has elapsed for the process to resume. Sampled frequently enough, this can provide a fine level of control over the fraction of CPU time used by the process. Comparison of the run-times of each node confirmed that this method does result in the desired ratios to within 2%.

4.1 Comparison of Communication Volumes

We ran the square-corner algorithm and the straight-line algorithm for power ratios ranging from 1:1 to 1:25 and for ten bandwidth values ranging from 50Mb/s to 400Mb/s. For all cases other than ratios of 1:1, 1:2, and 1:3, the total communication time for the square-corner algorithm was less than that of the straight-line partitioning.

Figure 6 shows a plot of the communication times for the square-corner partitioning and the straight-line partitioning. The average bandwidth is 80Mb/s and the power ratios range from 1:1 to 1:25. For ratios greater than 1:3, the average communication time for the square-corner partitioning is 45% lower than that of the straight-line partitioning.

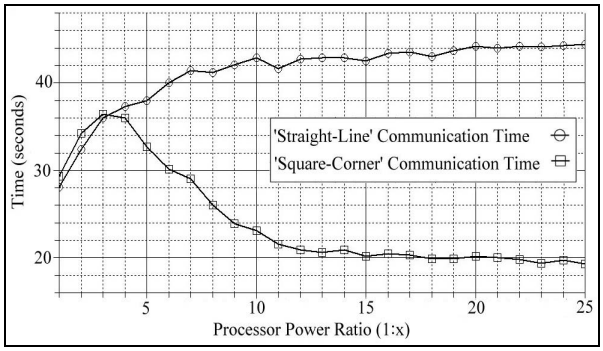

Figure 6. Comparison of the communication times for the square-corner and straight-line partitionings. The average bandwidth is 80Mb/s.

Figure 7 shows the same plot but with an average bandwidth of 380Mb/s. The average communication time for the square-corner partitioning is 44% less than the straight-line partitioning for ratios above 3:1.

The total communication time of the square-corner algorithm can be calculated with (4.1),

$$t_{comm} = \frac{2 \times N \times q \times m}{b} \qquad (4.1)$$

where m is the size in bits of the data type being communicated and b is the average bandwidth of the link in bits/second.

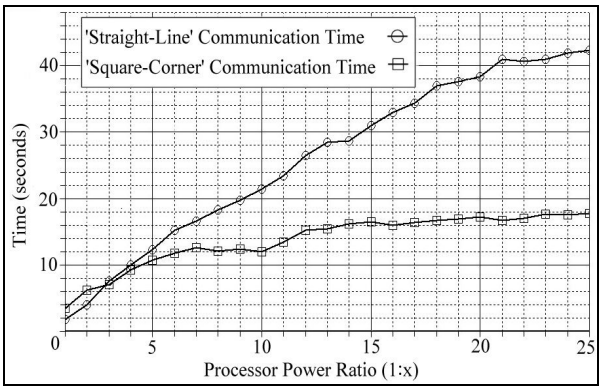

Figure 7. Comparison of the communication times for the square-corner and the straight-line partitionings. The average bandwidth is 380Mb/s.

4.2 Comparison of Total Execution Times

The square-corner algorithm is designed to reduce the inter-processor communication time resulting in a lower total execution time. Both algorithms should have equal computation times as each processor receives the same amount of work regardless of which algorithm is used. The only difference is how the data is partitioned. Since the total execution time is dependent on communication and computation time, any savings in total execution time will be dependent on how dominant communication time is in the overall execution time. As bandwidth increases, communication time decreases, and the overall contribution of communication in the total execution time decreases.

Figure 8 shows a plot of the total execution times for the square-corner partitioning and the straight-line partitioning with an average bandwidth of 80Mb/s.

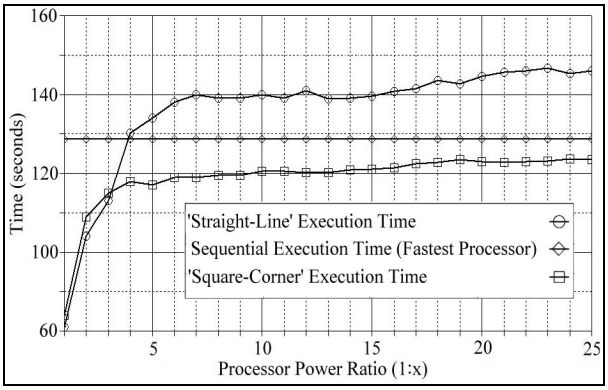

Figure 8. Comparison of the total execution times for the square-corner and the straight-line partitionings. The average bandwidth is 80Mb/s.

For all ratios above 3:1, the square-corner partitioning has a lower total execution time compared to the straight-line partitioning. On average, it is 14% faster for these ratios. It is notable that at this bandwidth, the straight-line partitioning results in a total execution time greater than the sequential execution time – the execution time achieved by straight-forward matrix multiplication of the same matrices on the faster of the two processors. However, the square-corner partitioning results in total execution times faster than the sequential.

Figure 9 shows a plot of the total execution times with an average bandwidth of 380Mb/s. On average, the total execution time for the square-corner algorithm is 10% less than the straight-line partitioning for ratios greater than 3:1. Additionally, the straight-line algorithm exceeds the sequential time for ratios above 1:15, while the square-corner algorithm's execution time is less than that of the sequential through all ratios tested.

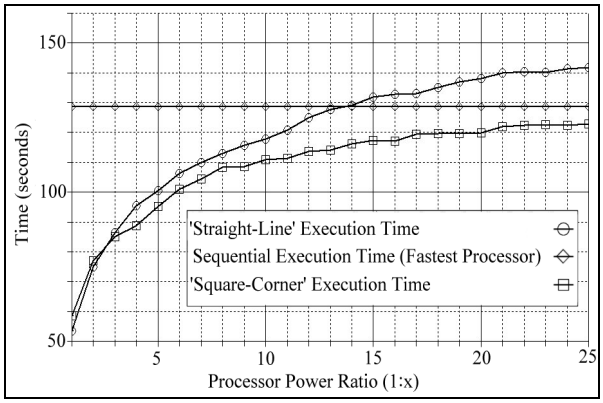

Figure 9. Comparison of the total execution times for the square-corner and the straight-line partitionings. The average bandwidth is 380Mb/s.

The computation time for the faster processor can be calculated by finding the size of a square matrix equal in area to the polygonal partition and comparing the work required to multiply two of these matrices to that required to multiply two test matrices of size $M \times M$. This is given by equation (4.2).

$$t_{comp} = \frac{t_1 \times (\sqrt{N^2 - q^2})^3}{M^3} \qquad (4.2)$$

Where t_1 is the time taken for the faster processor to calculate the product of two test matrices of size $M \times M$. We found M = 1000 to be suitable. For the slower processor, (4.2) reduces to equation (4.3).

$$t_{comp} = \frac{t_2 \times q^3}{M^3} \quad (4.3)$$

Where t_2 is the time taken for the slower processor to calculate the product of two $M \times M$ test matrices. Combining equations (4.1), (4.2), and (4.3), the total execution time is equal to (4.4).

$$t_{exe} = \frac{2 \times N \times q \times m}{b} + \max\left(\frac{t_1 \times (\sqrt{N^2 - q^2})^3}{M^3}, \frac{t_2 \times q^3}{M^3} \right) \quad (4.4)$$

4.3 "Real-World" Applicability

It has been proven that the square-corner algorithm has a lower total volume of communication for all power ratios greater than 1:3 compared to the straight-line algorithm. Further, our research has shown that this reduction in the total volume of communication results in lower execution times compared to the straight-line algorithm for the same ratios. A hybrid algorithm which utilizes the straight-line algorithm for power ratios less than 3:1, and the new square-corner algorithm for greater ratios would always be equivalent to or faster than the straight-line algorithm in all cases.

Additionally our research shows that for power ratios greater than 3:1 and matrix sizes in the range of N = 6500, two processors or clusters connected by a single communication link would benefit from parallelization utilizing this hybrid algorithm provided the bandwidth is 60Mb/s or greater. Such an architecture would not benefit from parallelizing the same problem using just the straight-line algorithm until the bandwidth reached 120Mb/s. Thus the square-corner algorithm not only reduces the communication volume and execution time, but reduces the bandwidth necessary for parallelization to be profitable.

5. Conclusion and Future Work

This paper presented a new data partitioning algorithm for matrix multiplication on two heterogeneous interconnected processors. Compared to more general algorithms which result in simple 'straight-line' rectangular partitions on a two-processor architecture, this 'square-corner' algorithm is proven to reduce the total volume of inter-processor communication when the power ratio of the two processors is greater than 3:1. This results in a lower execution time for architectures with these ratios. MPI experiments show average reductions in the total communication time to be on the order of 45%.

This partitioning algorithm can be utilized as the top-level partitioning of a hierarchal algorithm that is to multiply matrices across two connected clusters. A hybrid algorithm utilizing this new algorithm for power ratios equal to or greater than 3:1, and the existing straight-line partitioning for ratios of less than 3:1 guarantees that the total volume of communication will be equal to or less than previously existing algorithms for all ratios.

Our future work will include exploiting the possibility of overlapping some communication and computation as described in section 3.1, taking a functional model of processor performance into account when determining the size of the partitions, and deploying the square-corner algorithm on two heterogeneous clusters. We are also investigating the possibility of extending the square-corner algorithm to the architecture of three connected heterogeneous processors.

6. Acknowledgements

This work was supported by the Science Foundation Ireland

References

[1] Jorge G. Barbosa, João Tavares and Armando J. Padilha, "Linear Algebra Algorithms in Heterogeneous Cluster of Personal Computers", *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)*, 2000.

[2] Olivier Beaumont, Vincent Boudet, Fabrice Rastello and Yves Robert, "Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms", *Algorithmica*, 2002, Vol.34, No.3, pp.217-239.

[3] Olivier Beaumont, Vincent Boudet, Fabrice Rastello and Yves Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms", *IEEE Transactions on Parallel and Distributed Systems*, 2001, Vol.12, No.10, pp.1033-1051.

[4] Louis-Claude Canon and Emmanuel Jeannot, "Wrekavoc: a Tool for Emulating Heterogeneity", *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2006),* 2006.

[5] Egor Dovolnov, Alexey Kalinov and Sergey Klimov, "Natural Block Data Decomposition for Heterogeneous Clusters", *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003),* 2003.

[6] Edgar Gabriel et al., "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation", *Proceedings of the 11th European PVM/MPI Users' Group Meeting,(Euro PVM/MPI 2004)* 2004.

[7] Alexey Kalinov and Alexey Lastovetsky, "Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers", *Proceedings of the 7th International Conference on High Performance Computing and Networking Europe (HPCN`99)*, 1999.

[8] R. Clint Whaley and Jack Dongarra, “Automatically Tuned Linear Algebra Software”, *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.