Optimal Partitioning for Parallel Matrix Computation on a Small Number of Abstract Heterogeneous Processors

Ashley DeFlumere
UCD Student Number: 10286438

This thesis is submitted to University College Dublin in fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science

School of Computer Science and Informatics

Head of School : Pádraig Cunningham

Research Supervisor: Alexey Lastovetsky

September 2014

Contents

1 Introduction to Heterogeneous Computing and Data Partitioning **1**
- 1.1 Heterogeneous Computing . . . 2
  - 1.1.1 Types of Heterogeneous Systems . . . 2
  - 1.1.2 Benefits of Heterogeneity . . . 3
  - 1.1.3 Issues in Heterogeneous Computing . . . 3
- 1.2 Dense Linear Algebra and Matrix Computation . . . 4
- 1.3 Data Partitioning . . . 5
- 1.4 The Optimal Data Partition Shape Problem . . . 6
  - 1.4.1 Defining Optimality . . . 6
  - 1.4.2 Problem Formulation . . . 7
  - 1.4.3 Roadmap to the Optimal Shape Solution . . . 7
- 1.5 Contributions of this Thesis . . . 8

2 Background and Related Work **9**
- 2.1 A Brief Review of Matrix Multiplication . . . 9
  - 2.1.1 Basic Matrix Multiplication . . . 9
  - 2.1.2 Parallel Matrix Multiplication . . . 10
- 2.2 Rectangular Data Partitioning . . . 13
  - 2.2.1 One-Dimensional Partitioning . . . 13
  - 2.2.2 Two-Dimensional Partitioning . . . 14
  - 2.2.3 Performance Model Based Partitioning . . . 16
- 2.3 Non-Rectangular Data Partitioning . . . 16
  - 2.3.1 Two Processors . . . 16
  - 2.3.2 Three Processors . . . 17
- 2.4 Abstract Processors . . . 18

3 Modelling Matrix Computation on Abstract Processors **20**
- 3.1 Communication Modelling . . . 21
  - 3.1.1 Fully Connected Network Topology . . . 22
  - 3.1.2 Star Network Topology . . . 22

3.2 Computation Modelling 23
3.3 Memory Modelling 24
3.4 Algorithm Description 25
3.4.1 Bulk Communication with Barrier Algorithms 25
3.4.2 Communication/Computation Overlap Algorithms . . 27

4 The Push Technique 29
4.1 General Form 29
4.2 Using the Push Technique to solve the Optimal Data Partition Shape Problem 31
4.3 Application to Matrix Multiplication 32
4.4 Push Technique on a Two Processor System 32
4.4.1 Algorithmic Description 33
4.4.2 Push: Lowering Communication Time for all Algorithms 34
4.4.3 Two Processor Optimal Candidates 36

5 Two Processor Optimal Partition Shape 39
5.1 Optimal Candidates 39
5.2 Applying the Abstract Processor Model 41
5.3 Optimal Two Processor Data Partition 47
5.4 Experimental Results 52
5.4.1 Experimental Setup 52
5.4.2 Results by Algorithm 53

6 The Push Technique Revisited: Three Processors 59
6.1 Additional Push Constraints 59
6.2 Implementing the Push DFA 61
6.2.1 Motivation 61
6.2.2 Algorithmic Description 62
6.2.3 End Conditions 63
6.3 Experimental Results with the Push DFA 63
6.3.1 Experimental Setup 63
6.3.2 Description of Shape Archetypes 65
6.3.3 Reducing All Other Archetypes to Archetype A 67
6.4 Push Technique on Four or More Processors 72

7 Three Processor Optimal Partition Shape 74
7.1 Archetype A: Candidate Shapes 74
7.1.1 Formal Definition of Candidate Shapes 75
7.1.2 Finding the Canonical Version of each Shape 77
7.2 Network Topology Considerations 80

7.3 Fully Connected Network Topology 81
7.3.1 Pruning the Optimal Candidates 81
7.3.2 Optimal Three Processor FC Data Partition 84
7.3.3 Experimental Results 95
7.4 Star Network Topology 98
7.4.1 Pruning the Optimal Candidates 98
7.4.2 Applying the Abstract Processor Model 99
7.4.3 Optimal Three Processor ST Data Partition 105
7.5 Conclusion 114

8 Local Search Optimisations **115**
8.1 Description of Local Search 115
8.2 Simulated Annealing 115
8.2.1 Experimental Method for Matrix Partitioning 116
8.3 Conclusion 117

9 The Push Technique and LU Factorisation **119**
9.1 A Brief Review of LU Factorisation 119
9.1.1 Basic LU Factorisation 119
9.1.2 Parallel LU Factorisation 120
9.2 Background on Data Partitioning for LU Factorisation 121
9.2.1 One-Dimensional Partitioning 121
9.2.2 Two-Dimensional Partitioning 123
9.3 Applying Push to LU Factorisation 123
9.3.1 Special Considerations of LU Push 124
9.3.2 Methodology of LU Push 124
9.3.3 Two Processor Optimal Candidates 125

10 Further Work **127**
10.1 Arbitrary Number of Processors 127
10.2 Increasing Complexity of Abstract Processor 128
10.2.1 Non-Symmetric Network Topology 128
10.2.2 Computational Performance Model 128
10.2.3 Memory Model 129
10.3 Push Technique and Other Matrix Computations 129

11 Conclusions **130**
11.1 The Push Technique 130
11.2 Non-Rectangular Partitioning 131
11.2.1 The Square Corner Shape 131
11.2.2 The Square Rectangle Shape 132

11.3 Pushing the Boundaries: Wider Applicability 133

A Two Processor Push Algorithmic Description 134
A.1 Push Down 134
A.2 Push Up 135
A.3 Push Over 136
A.4 Push Back 136

B Push Lowers Communication Time: Two Processor Proofs 138
B.1 Serial Communication 138
B.2 Parallel Communication 139
B.3 Canonical Forms 141

C Push Lowers Execution Time: Three Processor Proofs 142
C.1 Serial Communication with Barrier 142
C.2 Parallel Communication with Barrier 143
C.3 Serial Communication with Bulk Overlap 143
C.4 Parallel Communication with Overlap 144
C.5 Parallel Interleaving Overlap 144

D Three Processor Fully Connected Optimal Shape Proofs 145
D.1 Serial Communication with Overlap 145
D.1.1 Square Corner Description 145
D.1.2 SCO Optimal Shape 150
D.2 Parallel Communication with Overlap 152
D.2.1 Square Corner 152
D.2.2 Square Rectangle 154
D.2.3 Block Rectangle 155
D.2.4 PCO Optimal Shape 155

List of Figures

2.1 Basic Matrix Multiplication 10
2.2 Parallel Matrix Multiplication 12
2.3 One Dimensional Partitions 14
2.4 Grid and Cartesian Partitioning 15
2.5 Column Based Partitioning 15
2.6 Previous Two Processor Non-Rectangular Work 17
2.7 Previous Three Processor Non-Rectangular Work 17
2.8 Example of Hybrid CPU/GPU System 19

3.1 Fully Connected Network Topology 23
3.2 Star Network Topology 23
3.3 Communication Pattern for Square Corner 25
3.4 Serial and Parallel Communication with Barrier Algorithm Description 26
3.5 Serial and Parallel Communication with Overlap Algorithm Description 27

4.1 Example of Enclosing Rectangles 30
4.2 Example of Two Processor Push in all Directions 34
4.3 Two Processor Push DFA Results 37

5.1 Two Processor MMM Optimal Candidates 40
5.2 Communication Pattern for Square Corner 41
5.3 Two Processor Square Corner portion for overlap 42
5.4 Two Processor SCO Square Corner constituent functions of T_{exe} max 44
5.5 Two Processor PCO Square Corner constituent functions of T_{exe} max 46
5.6 Two Processor SCB Theoretical Curves 53
5.7 Two Processor SCB Experimental Curves 54
5.8 Two Processor PCB Theoretical Curves 54
5.9 Two Processor PCB Experimental Curves 55

5.10 Two Processor SCO Experimental Curves $r < 3$ 55
5.11 Two Processor SCO Experimental Curves $r > 3$ 56
5.12 Two Processor PCO Experimental Curves $r < 3$ 57
5.13 Two Processor PCO Experimental Curves $r > 3$ 57
5.14 Two Processor PIO Experimental Curves - Computation dominant 58
5.15 Two Processor PIO Experimental Curves - Communication dominant 58

6.1 Push DFA State Diagram Example 64
6.2 Three Processor DFA Results - All Archetypes 66
6.3 Introduction to Corners 68
6.4 Taxonomy of Corner Labels 69
6.5 Three Processor Archetype B Shapes 70

7.1 Six Candidates of Archetype A - General Form 75
7.2 Two Versions of Type One Partition 78
7.3 Three Processor Candidate Shapes - Canonical Form 79
7.4 Three Processor Fully Connected Network Topology 80
7.5 Three Processor Star Network Topology 80
7.6 Three Processor Rectangle Corner and Block Rectangle Canonical Form 82
7.7 Three Processor L Rectangle Partition Shape Canonical Form 83
7.8 Three Processor SCB 3D Graph 87
7.9 Three Processor PCB 3D Graph 89
7.10 Three Processor SCO 3D Graph 92
7.11 Three Processor PCO 3D Graph 93
7.12 Three Processor SCB Experimental Results 96
7.13 Three Processor PCB Experimental Results 97
7.14 Three Processor Star Topology Candidate Shapes 99
7.15 Three Processor Star Topology Variant One 3D Graph Serial Communication 107
7.16 Three Processor Star Topology Variant One 3D Graph Parallel Communication 108
7.17 Three Processor Star Topology Variant Two 3D Graph Serial Communication 110
7.18 Three Processor Star Topology Variant Two 3D Graph Parallel Communication 111
7.19 Three Processor Star Topology Variant Three 3D Graph Serial Communication 112

7.20 Three Processor Star Topology Variant Three 3D Graph Parallel Communication 113

8.1 Simulated Annealing Results Unpermuted 117
8.2 Simulated Annealing Results Permuted 118

9.1 LU factorisation parallel algorithm 122
9.2 LU Factorisation One-Dimensional Partition Shape 123
9.3 LU Factorisation Two-Dimensional Partition Shape 124
9.4 Push for LU Factorisation 125
9.5 Push for LU Factorisation 126
9.6 Some LU Factorisation Candidates 126

11.1 Square Corner Optimal Shape 131
11.2 Square Rectangle Optimal Shape 132

D.1 Overlap Area of Square Corner for SCO and PCO Algorithms 146
D.2 Three Processor SCO Square Corner 149

Abstract

High Performance Computing (HPC) has grown to encompass many new architectures and algorithms. The Top500 list, which ranks the world's fastest supercomputers every six months, shows this trend towards a variety of heterogeneous architectures - particularly multicores and general purpose Graphical Processing Units (GPUs). Heterogeneity, whether it is in computational power or communication interconnect, provides new challenges in programming and algorithm development. The general trend has been to adapt algorithms used on homogeneous parallel systems for use in the new heterogeneous parallel systems. However, assumptions carried over from those homogeneous systems are not always applicable to heterogeneous systems.

Linear algebra matrix operations are widely used in scientific computing and are an area of significant HPC study. To parallelise matrix operations over many nodes in an HPC system, each processor is given a section of the matrix to compute. These sections are collectively called the data partition. Linear algebra operations, such as matrix matrix multiplication (MMM) and LU factorisation, use data partitioning based on the original homogeneous algorithms. Specifically, each processor is assigned a rectangular sub matrix. The primary motivation of this work is to question whether the rectangular data partitioning is optimal for heterogeneous systems.

This thesis will show the rectangular data partitioning is not universally optimal when applied to the heterogeneous case. The major contribution will be a new method for creating optimal data partitions, called the Push Technique. This method is used to make small, incremental changes to a data partition, while guaranteeing not to worsen it. The end result is a small number of potentially optimal data partitions, called candidates. These candidates are then analysed for differing numbers of processors and topologies. The validity of the Push Technique is verified analytically and experimentally.

The optimal data partition for matrix operations is found for systems of two and three heterogeneous processors, including differing communication topologies. A methodology is outlined for applying the Push Technique to matrix computations other than MMM, such as LU Factorisation, and for larger numbers of heterogeneous processors.

Statement of Original Authorship

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the Title Page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

Acknowledgements

First and foremost I must extend my utmost gratitude to my supervisor, Alexey Lastovetsky. I couldn't imagine a better research advisor. Thank you for teaching me so much about thinking and writing like a scientist, but most importantly, thank you for teaching me how to balance work and life to become a happier person.

Thank you to my examiners, Umit Catalyurek and Neil Hurley, for their helpful corrections and support.

This thesis is the result of research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN./I2054. Experiments were carried out on Grid'5000 developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

I am very grateful for the opportunity to travel to INRIA in Bordeaux, France to work with the wonderful Olivier Beaumont, which was funded by COST Action IC0805 "Open European Network for High Performance Computing on Complex Environments".

I owe a huge debt of gratitude to my eternal proof-reader and calculation-checker, my father, Michael DeFlumere.

Thank you to Sadaf Alam for introducing me to research at Oak Ridge, and encouraging me to pursue a graduate degree (and to consider doing so outside the USA!). I would also like to thank the Department of Computer Science at Mount Holyoke College, especially Barbara Lerner, Paul Dobosh, and Jim Teresco, for their unfailing encouragement during my time there.

Thank you to the members of the Heterogeneous Computing Lab, Vladimir Rychkov, David Clarke, Kiril Dichev, Ziming Zhong, Ken O'Brien, Tania Malik, Khalid Hasanov, Oleg Girko, and Amani Al Onazi. A very special thanks to Brett Becker for getting me started on the Square Corner.

To my family, my friends, and all my teammates at Ireland Lacrosse, thank you for supporting me and my crazy dreams.

And finally, I would like thank my cat for tolerating a transatlantic journey to be my constant companion and the keeper of my sanity.

For Dr. Richard Edward Mulcahy
(1913 - 2003)

Chapter 1

Introduction to Heterogeneous Computing and Data Partitioning

The world of High Performance Computing (HPC) has grown to encompass many new architectures, algorithms, models and tools. Over the past four decades, HPC systems, comprised of increasingly complex and numerous components, have grown by orders of magnitude in terms of computational power, as measured by the number of floating point operations per second (FLOPs).

As machines increased in size and parallelism, the teraFLOP barrier was reached in 1996 by the ASCI Red supercomputer [1]. Just over a decade later, the petaFLOP barrier was broken by the Roadrunner supercomputer in 2008 [2].

The Top500[1] list [3], which ranks the world's fastest computers every six months, clearly shows a trend towards a variety of heterogeneous architectures - particularly multicores and hardware accelerators such as GPUs (Graphical Processing Units)[4, 5]. As of the June 2014 list, 37 systems around the world had achieved petaFLOP speeds.

In the coming decade, a monumental effort will be made to achieve exascale (1000 petaFLOPs, or 10^{18} FLOPs) computing systems. The massively parallel nature of HPC has led to many new advancements in algorithms, models and tools, which strive to reach the exascale goal by utilising and working with the latest computer hardware [6]. The software stack, *e.g.* compilers, message passing, and operating systems, is constantly being updated to reflect the new capabilities in performance of both the computation,

[1] http://www.top500.org

and the communication, of new chipsets. In general, as machines become heterogeneous in computation and communication the homogeneous parallel algorithms are adapted for use in these newer systems. However, the important question remains, what if assumptions carried over from homogeneous systems are no longer applicable when using a heterogeneous system?

1.1 Heterogeneous Computing

> Heterogeneous (adj.) - "Diverse in character or content"
> -Oxford English Dictionary

In terms of computing, "heterogeneous" refers to non-uniformity in some aspect of the system, either by design or through ageing. In general terms, heterogeneity in computing manifests in three ways:

- differing amounts of computational power among subsystems
- differing amounts of bandwidth or latency within the communication interconnect
- some combination of the above two

This may occur naturally as systems age, and begin to be replaced piecemeal. This type of heterogeneity happens particularly in smaller clusters, where financial considerations might preclude an older system from being removed entirely. Heterogeneity is increasingly intentional in HPC, however, with systems that combine traditional multi-core processors with general purpose GPUs and other hardware accelerators and coprocessors.

1.1.1 Types of Heterogeneous Systems

The following is an incomplete list, but should serve as an example of the many types of heterogeneity in HPC.

- Compute nodes comprised of combinations of CPUs, GPUs, FPGAs (field programmable gate arrays), or coprocessors
- Compute nodes comprised of specialised cores, each tailored to a specific function
- Communication networks with differing bandwidth or latency between nodes, either symmetric or non-symmetric

- Clusters comprised of heterogeneous nodes, or several differing clusters of homogeneous nodes used in concert
- Groups of workstations, often of different computational power, clock frequency, and system software, used in concert

Whatever the form, heterogeneous systems share the common thread of non-uniformity, which has both benefits and drawbacks [7]. These are discussed in the following sections.

1.1.2 Benefits of Heterogeneity

Heterogeneous computing presents many benefits, the scale and scope of which depend largely on the type of heterogeneity present.

The primary benefit of concern in this thesis is increased computational power, increasing the data throughput over traditional CPU only systems. GPUs, for instance, were first used to render images in video games, and so, are plentiful and inexpensive compared to other technology available for HPC [8, 9]. The relative computational power of the GPU, specifically for highly data parallel computations, allows speedups in many scientific applications [10, 11].

Another example of increased computational power, is collection of networked workstations viewed as a single HPC system. These workstations groups are a network of vastly different computing resources being harnessed for use on a single problem. This has also been shown to provide computational speedup for linear algebra applications [12, 13].

Other types of heterogeneous systems, can be used to generate benefits other than increased computational power. Nodes with heterogeneous specialised cores, similar to what has long been used in mobile phones, can be used to increase power efficiency [14, 15]. A general purpose CPU must be capable of a wide variety of tasks, but is not optimised for any of them. Instead, the idea is to orchestrate a variety of specialised heterogeneous resources to accomplish the same tasks, in a faster and more efficient way [16].

1.1.3 Issues in Heterogeneous Computing

Despite the advantages, heterogeneous systems do present unique challenges in scalability, algorithm development, and programmability for parallel processing. A GPU, for instance, must be controlled by a CPU core, and has limited bandwidth for communication and memory access [17].

Algorithms which have been carefully tuned for homogeneous parallel systems must be rethought to provide the optimal performance on heterogeneous systems, often to take advantage of the increased computational power. A large number of proposals have been made for algorithms [18, 19], models [20], and tools [21, 22, 23], to make up this gap in knowledge for heterogeneous systems.

Despite all this prior work, however, HPC systems can be large, complex and difficult to use in an optimal way. This is a rich and open research area; how to optimally model and program for these diverse, large scale heterogeneous environments.

1.2 Dense Linear Algebra and Matrix Computation

Linear algebra operations are widely used in scientific computing and are an important part of HPC. Software packages such as High Performance LinPACK (HPL) [24] and ScaLAPACK [25] provide linear algebra implementations for HPC.

Dense linear algebra is essential to the computation of matrices used in a variety of scientific fields. Some fundamental linear algebra kernels, such as LU factorisation and its underlying parallel computation, matrix multiplication, are used to solve systems of linear equations with arbitrarily large numbers of variables. In that way, any scientific field which can approximate its applications by linear equations, such as astronomy, meteorology, physics, and chemistry, can use HPC systems and the ScaLAPACK package to vastly improve the execution time of the domain code.

These linear algebra kernels are computationally intense, but often this computational load may be parallelised over many compute nodes. However, this introduces the problem of communicating between the nodes to share data and synchronise calculations. As HPC continues to gain computational power, the effect of this communication time will grow proportionately [26].

Because of their usefulness in such a wide variety of applications both matrix multiplication and LU factorisation have been studied in great detail. Ways to model heterogeneous systems, and algorithms to improve utilisation of computation and communication resources, are being developed and improved upon. A full accounting of the advancements in parallel computation of matrix multiplication is given in Chapter 2.

1.3 Data Partitioning

A data partition defines the way in which the linear algebra matrices are divided amongst the available processing resources. Data partitions are generally designed in order to optimise some fundamental metric of the application, such as execution time, or power efficiency. This thesis will focus on the former; the overall execution time of the matrix computation.

Data partitions endeavour to optimally distribute the computational load of the problem matrices amongst the available processors. The speed at which an individual processor can perform basic operations, like add and multiply, determines the proportion of the overall problem it will be assigned. In this way, when computing in parallel, all processors will complete computation at the same time, and no processor will sit idle without work.

The other consideration of a data partition is its shape. The shape of a data partition is the location within the matrix of each processor's assigned portion. In the case of matrix multiplication, the shape of the data partition does *not* affect the volume of computation (although the cost may be adversely affected by physical constraints such as cache misses depending on the data types used). However the partition shape *does* directly affect the volume of communication. A processor may require data "owned" by a second processor in order to compute its assigned portion.

In the case of LU factorisation, the layout of processor data within the partition affects both computation and communication costs. As the factorisation proceeds, an increasing amount of the matrix is completed and no longer used. If the data partition assigns a processor to a section completed early on, then it will sit idle for the remainder of the execution time; this is clearly inefficient.

The communication time of a data partition is increasingly important as HPC systems become ever more computationally efficient [26]. A variety of advances have been made in minimising communication for matrix computations on heterogeneous systems [27]. The general shapes of these partitionings are, however, always rectangular. These are described in detail in Chapter 2.

However, these approaches are fundamentally limited by nature of the fact that they are based on algorithms and techniques developed for homogeneous systems. To find an optimal solution to the data partition shape problem, one must consider not only rectangular shapes, but all shapes, *i.e.* non-rectangular.

1.4 The Optimal Data Partition Shape Problem

Despite all the previous study on data partitioning discussed in Chapter 2, the problem of optimal data partitioning for heterogeneous processors has yet to be solved. Finding simply the optimal rectangular shape is *NP*-complete [28], and research has focused almost exclusively on approximating the optimal rectangular solution. The previous work, discussed in Section 2.3, studying non-traditional shapes was not concerned with optimality, but with a direct comparison between one type of rectangular partition and one type of non-rectangular partition.

The goal of this work is to define, and solve, the broader problem of optimality. It is this fundamental broadness that necessitates a focus on small numbers of heterogeneous processors. This is a natural starting place for the larger problem of optimal data partitioning on arbitrary numbers of heterogeneous processors. The remainder of this section sets out the research questions and aims, and provides a roadmap of how the optimal data partition shape problem will be solved.

1.4.1 Defining Optimality

> Optimal (adj.) - "Best or most favourable; optimum"
>
> - Oxford English Dictionary

For the purposes of this thesis, the concept of optimality will need to be addressed directly and with specific intent. It is the data partition shape, whatever it may look like, which will be said to be optimal or sub-optimal. This judgement must be made on the basis of an objective fact. This fact will be the execution time of the relevant matrix computation when using the data partition of that shape.

A great number of factors contribute to the execution time of a data partition, both in the communication and computational subsystems. Therefore, it is only for a given set of these factors (*i.e.* processor computational power ratios or communication algorithm) that a partition shape can be said to be optimal.

Finally, and most importantly, a shape cannot be said to be optimal unless it has been compared to *all* other possible shapes, including those shapes composed of random arrangements of elements among processors. Consider every possible way to distribute elements among processors randomly throughout the matrix; each permutation is a shape to be evaluated.

1.4.2 Problem Formulation

As non-rectangular partition shapes have never been seriously considered, it is possible that, even if only for certain systems, a non-rectangular shape could be superior to the rectangular shape, or indeed even optimal in the entire solution space. The question remains of how to determine that a shape is optimal if it must be compared to all possible data partitions, in order to confirm that it is indeed best (has the lowest execution time). It is necessary to create a method which can state that a more manageable subset of data partition shapes are guaranteed to be superior to all shapes which are not included in the subset.

Beyond that, optimality requires specific data (the execution time) in order to be deduced. Therefore, a processor, with computation and communication characteristics, must be defined in full; but what type of heterogeneous processor to use? As previously discussed, heterogeneous systems can be composed of processing elements of differing design and speed, at the system and node levels. The solution of the optimal partition shape should be applicable to as wide a variety of these classes of heterogeneous processors as practical. This will require defining the key performance metrics of some *abstract* heterogeneous processor.

Furthermore, there must be a way in which to model the performance of the dense linear algebra application. Specifically, what type of algorithm will the communication use? The linear algebra computation used also determines the necessary communication pattern characteristics of the model.

Finally, the fundamental question is, what is the optimal data partitioning shape for two or three heterogeneous processors? Is it non-rectangular?

1.4.3 Roadmap to the Optimal Shape Solution

The rest of this thesis is dedicated to answering these questions. First, the stage is set with a full mathematical model for an abstract processor, algorithms and performance metrics of a partition shape in Chapter 3. These will provide the necessary language in order to determine the optimality of a partition shape.

Next, the Push Technique is introduced, allowing *all* possible partition shapes to be considered. A deterministic finite automaton is described to achieve this, and the practical implementation is also discussed in Chapter 4. The technique produces several partition shapes, *candidates*, which will then be analysed in the remaining chapters in order to determine the optimal shape for each set of factors.

For two heterogeneous processors, in Chapter 5, it will be shown that the

non-rectangular shape, called Square Corner, is optimal for defined ratios of computational power. For three heterogeneous processors, in Chapter 7, it will be shown that there are two non-rectangular shapes, called Square Corner and Square Rectangle, that are each optimal, for different levels of heterogeneity in computational power ratios.

1.5 Contributions of this Thesis

In summary the major contributions of this thesis are,

- Proposal of the Push Technique for finding optimal data partitions
- Analysis of the Push Technique for two and three processors
- The optimal data partition shape for any power ratio of two and three processors
- The introduction of a novel optimal non-rectangular data partitioning shape, the "Square Rectangle"
- Methodology to apply the Push Technique to any matrix computation

This thesis will show the rectangular data partitioning is *not* universally optimal when applied to the heterogeneous case. The major contribution is the new method for analysing data partitions, called the Push Technique. This technique is shown to produce novel candidate shapes, which can be evaluated directly to determine optimality. The validity of the Push Technique is verified analytically and experimentally.

The optimal data partition for matrix computations is found for systems of two and three heterogeneous processors, including differing communication topologies. For both two and three processors, non-rectangular partitions are shown to be optimal for certain system characteristics.

One data partition, which will be referred to as the Square Rectangle due to its shape, has never before been considered, and is shown to be an optimal three processor shape.

A methodology is outlined for applying the Push Technique to matrix computations other than matrix multiplication, and for larger numbers of heterogeneous processors. Specifically, it is shown how to apply the Push Technique to LU factorisation, and some two processor candidate shapes are given.

Chapter 2

Background and Related Work

This chapter will explore the background of the concepts to be questioned and studied in this thesis. First, a review of state of the art algorithms used to compute parallel matrix multiplication is given. Then, data partitioning is described, along with algorithms used to create the various existing rectangular partition shapes. Finally, there is a review of the work to date in non-rectangular data partitioning.

2.1 A Brief Review of Matrix Multiplication

Matrix multiplication is a focus of this thesis, and it is described first as a serial linear algebra algorithm with advancements in the required volume of computation. Then, several algorithms used to compute matrix multiplication in parallel on multiple processors are explored, including the state of the art SUMMA algorithm [29].

2.1.1 Basic Matrix Multiplication

Matrix multiplication is a fundamental linear algebra operation. The general convention followed here is to name the square input matrices A and B, and the product matrix C. An element of matrix C is the product of the corresponding row of matrix A and column of matrix B. Matrix A must have N columns, and matrix B must have N rows. This calculation requires N^2 dot products, which each require N multiplications, thus, naively, matrix multiplication is said to require N^3 multiplications. Various algorithms have been shown to reduce this number, with the current minimum being $N^{2.3727}$ [30].

In the past, some of the simpler reductions, such as Strassen's algorithm

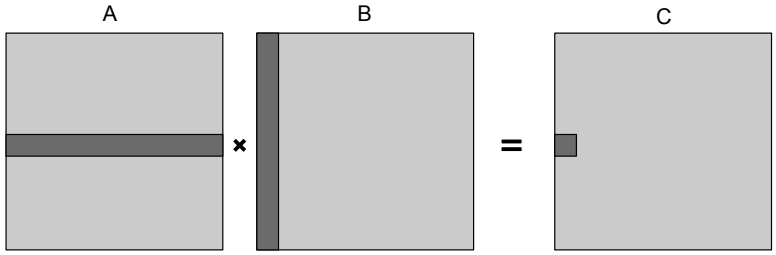


Figure 2.1: Basic Matrix Multiplication. The dark grey portion of matrix C is calculated using the data from matrices A and B, also shown in dark grey.

[31] ($N^{2.807}$), have been implemented as practical matrix multiplication techniques [32, 33, 34]. However, the newest algorithms developed to reduce the number of required multiplications can be awkward to implement in the heterogeneous case, and in scientific computing matrix multiplication is generally done in a straightforward N^3 way [35]. This is practical as HPC platforms, especially with future exascale computational speeds, will be increasingly bounded by limitations in communication and memory rather than computation speed. Computational speed is the metric by which HPC systems are judged, so in the future it is reasonable to expect that while the computation portion of a given algorithm is executed faster, the communication portion of that same algorithm won't experience the same speed up. For this reason, communication will become a larger portion of the overall execution time, relative to the computation.

2.1.2 Parallel Matrix Multiplication

When computing matrix multiplication on multiple machines in parallel, each processor is generally assigned some portion of the result matrix C to compute. Each processor must also store the necessary data from matrices A and B in order to complete these calculations.

For small matrices, it may be simple to naively give each processor a copy of the entirety of matrices A and B. However, for the large matrices used in scientific computing, there is quickly a memory bottleneck. Instead, only the area of these input matrices actually required for the computation is stored.

For larger matrices, if a processor does not have a copy of the required data of matrices A and B, it will be necessary to communicate that information from the processor which does have the data. Thus, it becomes necessary to devise a matrix multiplication data partitioning algorithm which will minimise the volume of communication among the processors. The following is a summary of the historical advancements in this area, and a description of

the current state of the art.

Cannon's Algorithm and Fox's Algorithm

Cannon's algorithm [36] was first suggested in 1969. The first efficient parallel matrix multiplication algorithm, it involves a circular shift and multiply approach. Similarly, Fox's algorithm [37] is the other classical example of parallel matrix multiplication algorithms. In both, processors are arranged in a 2-dimensional grid with block data distribution, and there must be a perfect square number of processors. These algorithms provide excellent communication performance, however, they are limited to perfect squares and are therefore inadequate for general purpose linear algebra applications.

3D Mesh

The 3D Mesh algorithm [38] arranges the p processors in a $p^{\frac{1}{3}} \times p^{\frac{1}{3}} \times p^{\frac{1}{3}}$ cube. The benefit of this 3D approach is a reduction in the communication volume. It requires $p^{\frac{1}{6}}$ less communication than a traditional 2D algorithm. However, the drawback is the additional memory required to store the extra copies of data. In all, the 3D Mesh requires an additional $p^{\frac{1}{3}}$ copies of the matrices, which is impractical for large problem sizes.

2.5D Algorithm

The 2.5D algorithm [39] is similar to the 3D Mesh algorithm, however it parameterises the third dimension. This allows some control over the amount of extra memory the algorithm requires, allowing customisation to the system. This has been shown to be communication optimal [40]. However, for large matrices on systems with low local memory (such as GPUs), it may not be possible to store any redundant matrix copies.

PUMMA

The PUMMA [41] (Parallel Universal Matrix Multiplication Algorithm) was created in an attempt to generalise Fox's algorithm to any 2-dimensional grid. This algorithm accomplishes this by using a block scattered data distribution. The major drawback is excessive memory usage for large matrices, making this algorithm scale poorly.

State of the Art: SUMMA

The SUMMA [29] (Scalable Universal Matrix Multiplication Algorithm) is an improvement of the PUMMA algorithm, looking to, as the name would suggest, make the PUMMA algorithm scalable. The SUMMA algorithm, although nearly two decades old, is still considered to be a state of the art algorithm. It is currently in use in popular linear algebra packages, such as ScaLAPACK [25]. For this reason, it is discussed in more detail here than the other algorithms.

In the SUMMA algorithm, the processors to be used in the computation are arranged in a 2-dimensional grid of dimensions $i \times j$ such that each processor is called $p(i, j)$. The multiplication is broken down into k equal steps, with the optimal size of a step generally being determined so as to fit in the cache memory available on the processors.

At every step, the data of that step k is broadcast to all processors requiring this information. Each column of matrix A is sent horizontally, and each row of matrix B is sent vertically, as seen in Figure 2.2. After this communication, the entire matrix C is updated with the equation $C[i, j] = C[i, j] + A[i, k] * B[k, j]$.

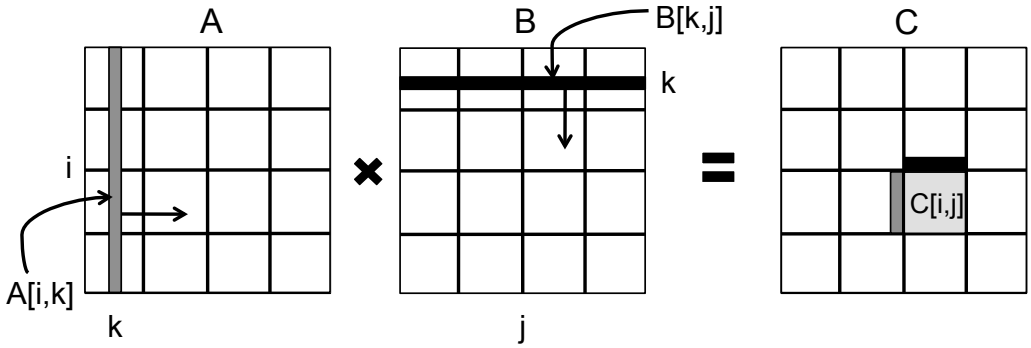


Figure 2.2: Parallel matrix multiplication, as computed by the SUMMA algorithm as shown on a grid of 16 processors. At each step, k, data from column k of matrix A is broadcast horizontally and data from row k of matrix B is broadcast vertically. At each step all elements of C are incrementally updated.

The efficiency of the communication is improved by pipelining, which is the formation of a logical ring out of each row and column, to pass the messages around. In this way, each processor need only communication with its neighbour in the grid, which is more efficient than a broadcast communication [29].

The overall cost of this algorithm is good for such a general solution. It is computationally balanced, and achieves within $\log p$ of the communication lower bound.

Recent Additions to SUMMA

Several algorithms have attempted to build on the SUMMA algorithmic design. The SRUMMA [42] (Shared and Remote-memory based Universal Matrix Multiplication Algorithm) has a complexity comparable to Cannon's algorithm, but uses shared memory and remote memory access instead of message passing. This makes it appropriate for clusters and shared memory systems.

The HSUMMA [19] (Hierarchical Scalable Universal Matrix Multiplication Algorithm) is another recent extension of the SUMMA algorithm. It adds a hierarchical, two dimensional decomposition to SUMMA in order to reduce the communication cost of the algorithm.

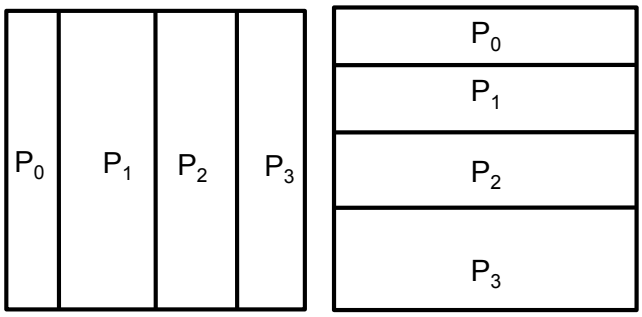
2.2 Rectangular Data Partitioning

This section includes a review of previous work in the area of data partitioning for linear algebra applications. The problem of optimally partitioning heterogeneous processors in a general way is NP-complete [28, 43]. However, a number of limited solutions have been created, and some common sub-optimal rectangular partitioning schemes are presented here.

In all cases, the matrices are assumed to be identically partitioned across matrices A, B, and C. As is found throughout the literature, especially those discussed below, a change in any partition shape will be reflected in the partition shape for all matrices. For this reason, the partition shape is displayed as a single matrix and it is understood to represent all three matrices.

2.2.1 One-Dimensional Partitioning

The simplest rectangular partition is a one-dimensional arrangement. Each processor is assigned an entire column, or an entire row, as shown in Figure 2.3. The rectangles are the entire length of the matrix, and of width relative to that processor's speed. However, these partitioning techniques have a large communication cost, as every processor must communicate with all the other processors.



(a) 1D Columns (b) 1D Rows

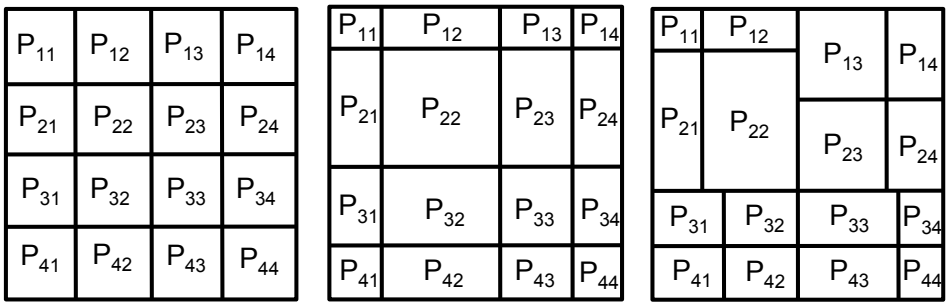
Figure 2.3: One-Dimensional data partitioning techniques for four heterogeneous processors.

2.2.2 Two-Dimensional Partitioning

Cartesian Partitioning. Cartesian partitioning is the most restrictive of the two-dimensional heterogeneous partitioning schemes. Each processor must align in its column and row with all other processors in that row and column, as seen in Figure 2.4. It is an obvious derivation of the traditional homogeneous partition. The benefit of this approach, for matrix multiplication, is that each processor communicates only with the processors directly in its row and column. As [44] points out, this attribute makes cartesian partitions highly scalable. However, the cartesian partition turns out to be too restrictive, as it cannot guarantee that the best cartesian partition will balance the computational load, given the number and relative speeds of processors.

Grid Partitioning. Grid based partitioning creates a two-dimensional grid divided into rectangles, one per processor, as seen in Figure 2.4. If an arbitrary horizontal or vertical line is drawn through the partition shape, it would always pass through the same number of processors. Unlike in the homogeneous and cartesian partitions, the processors in the grid partition need not be aligned into specified rows or columns. The major drawback of this partitioning is that it only minimises communication cost within its rectangular restrictions, and finding the optimal grid partition which minimises communication is NP-complete [43, 45].

Column-Based Partitioning. Column-Based partitioning, shown in Figure 2.5, was suggested in [46] and [47]. The idea is to divide the matrix into some number of columns, c, and to distribute processors within these



(a) Homogeneous (b) Cartesian (c) Grid

Figure 2.4: Data partitioning techniques. On the left, (a), is a 16 processor homogeneous partition for reference. Each processor is assigned a square of the matrix. In (b) is a heterogeneous cartesian partition over 16 processors of varying speeds. Similarly, (c) is a heterogeneous grid partition for 16 processors.

columns. The communication minimising algorithm presented in [28] extends the column based approach. Both the shape and location of the rectangles within the matrix are chosen to minimise the communication cost. As previously stated, the general solution to this problem was shown to be NP-complete in a non-trivial proof, however with the additional restriction that all rectangles within a column must have the same width, a polynomial time sub-optimal solution may be found[28].

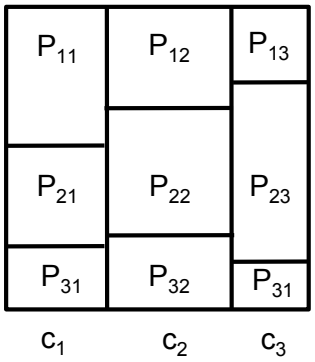


Figure 2.5: A column based partitioning with nine heterogeneous processors, and three columns, c_1, c_2, c_3.

2.2.3 Performance Model Based Partitioning

The previously discussed partitioning algorithms all assumed a constant performance model for the processors. Each processor is assigned a constant positive value to represent its speed in proportion to the rest of the processors. The benefit of this technique is its simplicity, as well as it's accuracy for small to medium, or constant, sized problem domains. However, for large or fluctuating problem sizes, it is possible that the performance will change leading to inaccuracy in the model, which can cause a degradation in the performance of the overall application.

Functional Performance Modelling. Functional performance modelling [48, 49, 50] takes into consideration the problem size when estimating the speed of a given processor. This allows for accurate modelling of processor performance if the processor throughput degrades with problem size. Most often, this occurs when some physical limit of the processor is met, such as the filling of cache memory, and a sudden and marked deterioration in performance occurs. With this clear physical limitation in mind, these functional performance models have been used to find data partitions for linear algebra applications such as parallel matrix multiplication [51].

2.3 Non-Rectangular Data Partitioning

This section provides a survey of previous work that examined non-rectangular partitioning of matrices for matrix multiplication. These results focus on comparing two shapes, rather than considering their optimality. They also consider only communication, not execution time, and shapes are not compared for a wide variety of performance model algorithms.

2.3.1 Two Processors

Previous work, [52], looking at the case of two heterogeneous processors considered two data partitions, one of which was non-rectangular. While this work did not consider the optimality of either of these shapes, it did show that in direct comparison a non-rectangular shape was superior to the traditional rectangular shape for ratios of computational power greater than three to one between the processors. These two shapes are shown in Figure 2.6. The non-rectangular shape has a lower volume of communication, and performs better in terms of execution time, at the indicated ratios.

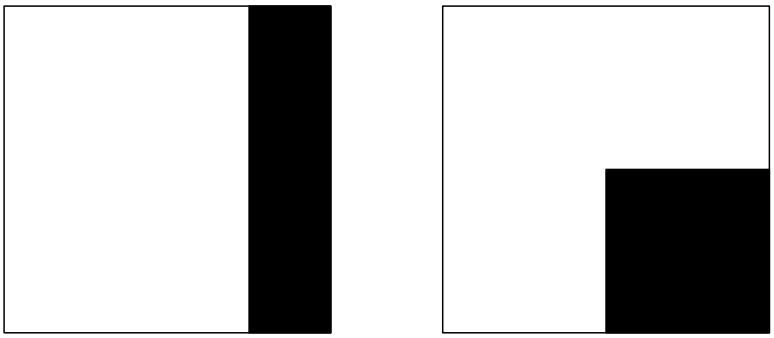


Figure 2.6: The two matrix data partition shapes considered in [52], partitioned between two processors (white and black). The shapes are Straight Line on the left, and Square Corner on the right. Analysis shows the Square Corner has the lower volume of communication when the computational power ratio between the processors is greater than $3:1$. (As shown, each data partition is of ratio $3:1$).

2.3.2 Three Processors

Previous work, [53], considered three heterogeneous processors. Again, this work does not consider the optimality of shapes, but directly compares one non-rectangular shape against a traditional rectangular partition shape. These shapes are detailed in Figure 2.7. This work finds that the non-rectangular partition shape can be superior to the rectangular shape for highly heterogeneous systems and non-fully connected network topologies.

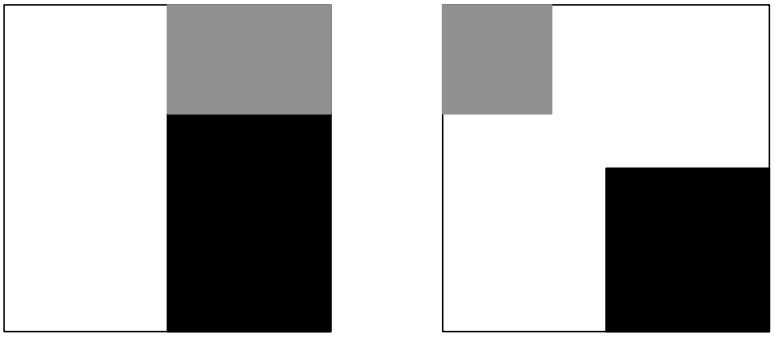


Figure 2.7: The two matrix data partition shapes considered in [53], partitioned between three processors (white, grey, and black). The shapes are Rectangular on the left, and Square Corner on the right.

Both of these works, and [54], motivate the idea that it is possible for a non-rectangular partition shape to be optimal. In some cases one non-

rectangular shape, the Square Corner, is superior to specific common rectangular shapes. However, these previous works fail to address the concept of optimality or matrix computations other than matrix multiplication.

2.4 Abstract Processors

The notion of an abstract processor has previously been used to represent a variety of different real world heterogeneous systems.

In [55], the authors used abstract processor models to encapsulate multicore processors. This approach was used to balance the computational load for matrix multiplication on multicore nodes of a heterogeneous multicore cluster.

In [56], the authors extend this abstract processor model approach to be applicable to both heterogeneous multicore and hybrid multicore CPU/GPU systems, an example of which can be seen in Figure 2.8. Using this approach, they were able to accurately model the performance of different heterogeneous configurations for scientific data parallel applications. These various heterogeneous components were often described as systems of two or three heterogeneous abstract processors. However, these works only considered traditional, rectangular data partitions for these systems. The types of novel, non-rectangular data partition shapes presented in this thesis have never been considered using this type of abstract processor model.

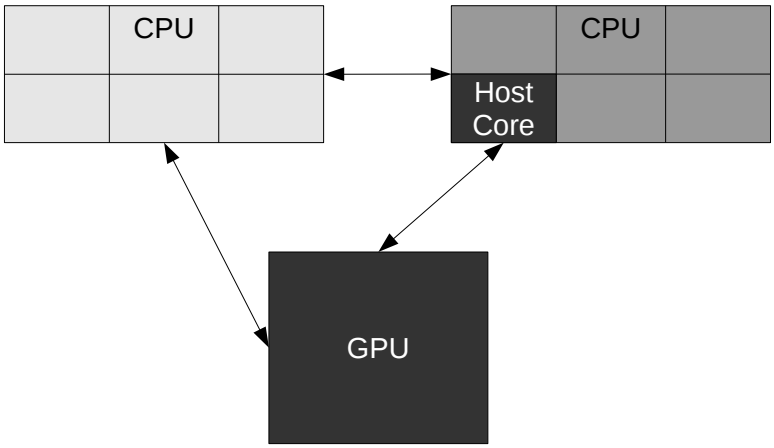


Figure 2.8: An example of the type of heterogeneous system addressed using abstract processor models in [55, 56]. As shown, a multicore CPU with 6 cores, a multicore CPU of 5 cores, and a GPU with its host core, are shown as three abstract processors.

Chapter 3

Modelling Matrix Computation on Abstract Processors

In this chapter, the complete performance model for the abstract heterogeneous processor is presented. An abstract processor is any unit of processing power which may receive, store, and compute data, and most importantly, is independent. An independent processing unit is not affected by the computational load placed on any other processing unit. For example, an independent processor is not affected by resource contention, as cores on the same die would be. Examples of an independent processing unit include single and multicore CPUs, a GPU and its host core, or an entire cluster.

The notion of an abstract processor, which focuses primarily on communication volume and computation volume, has been shown to accurately predict the experimental performance of a variety of processors and even entire clusters for matrix computations [52, 53, 54, 57, 58]. Below, the communication, computation, and memory modelling of an abstract processor is discussed further in the context of matrix computations.

Data Partition Metrics

In order to effectively model the matrix computations, several assumptions are made and partition metrics are devised.

- The matrices are square and of size $N \times N$ elements
- The matrices are identically partitioned among p processors
- The number of elements assigned to each processor is factorable, *i.e.* may be made into a rectangle

Formally, a partition is an arrangement of elements amongst processors such that,

$$\phi(i,j) = \begin{cases} 0 & \text{Element (i, j) assigned to } 1^{st} \text{ Processor} \\ 1 & \text{Element (i, j) assigned to } 2^{nd} \text{ Processor} \\ & \vdots \\ p-1 & \text{Element (i, j) assigned to } p^{th} \text{ Processor} \end{cases} \tag{3.1}$$

To determine whether a given row, i, contains elements that are assigned to a given processor, x,

$$r(\phi,x,i) = \begin{cases} 0 & \text{if } (i,\cdot) \text{ of } \phi \text{ is } \textit{not} \text{ assigned to Processor } x \\ 1 & \text{if some } (i,\cdot) \text{ of } \phi \text{ is assigned to Processor } x \end{cases} \tag{3.2}$$

To determine whether a given column, j, contains elements that are assigned to a given processor, x,

$$c(\phi,x,j) = \begin{cases} 0 & \text{if } (\cdot,j) \text{ of } \phi \text{ is } \textit{not} \text{ assigned to Processor } x \\ 1 & \text{if some } (\cdot,j) \text{ of } \phi \text{ is assigned to Processor } x \end{cases} \tag{3.3}$$

Processor Naming Convention

The equations provided below are written in such a way to be applicable to any number of processors p, and correspondingly the x^{th} processor is called p_x. However, for the small numbers of abstract processors studied in detail, it may be useful to call the processors by different letters, for clarity.

For two processors, the more powerful processor is known as Processor P, and the less powerful as Processor S. The ratio between the computational power of these two processors is called r, and is normalised to be $r : 1$.

For three processors, the processors are called, in descending order of computational power, Processor P, Processor R, and Processor S. The ratio between the computational power of these processors is $P_r : R_r : S_r$, and is normalised to $P_r : R_r : 1$.

3.1 Communication Modelling

The communication of matrix multiplication may be modelled in a variety of ways, depending on the level of specificity required. First is the question of topology. Does a link exist between all processors? Then consider, are all

links between processors symmetrical? Are there significant startup costs in sending an individual message?

When considering only small numbers of abstract processors, the possible topologies are limited. For two processors, for instance, the only option is fully connected, or no communication would be possible. When considering three processors, the options are fully connected or a star topology, so these are discussed in further detail below. Additionally, the focus is placed on symmetric communications, and latency will be ignored due to its lesser significance in the communication and computation volume in most of the algorithms studied (specifically those which use bulk communication).

3.1.1 Fully Connected Network Topology

In the fully connected topology, each processor has a direct communication link to all $p-1$ other processors, as seen in Figure 3.1. The simplest model of symmetric communication is the Hockney Model [59]. This states the time of communication is a factor of latency, bandwidth, and message size, and is given by,

$$T_{comm} = \alpha + \beta M \tag{3.4}$$

$\alpha = 0 =$ the latency, overhead of sending one message in seconds
(which is insignificant compared to βM in this model, so set to zero)
$\beta =$ the inverse of bandwidth, transfer time (in seconds) per element
(which for simplicity will be referred to as bandwidth throughout this thesis)
$M =$ the message size, the number of elements to be sent

3.1.2 Star Network Topology

The star topology puts a single processor at the centre, with all other processors communicating through it, as seen in Figure 3.2. For most of the partition shapes that will be found using the Push Technique, the data that an outer processor sends to the centre processor is not the same data (*i.e.* it comes from a different location or matrix) as it sends to some other outer processor. For this reason, the outer-to-outer term in the communication time equation is doubled. This may be possible to improve upon, depending on the partition shape, however this equation represents the worst case scenario.

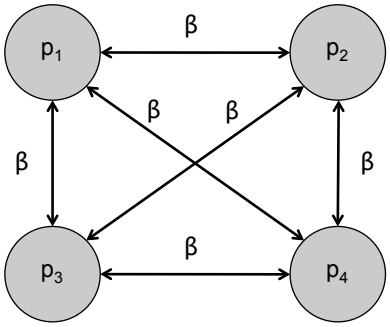


Figure 3.1: A fully connected network topology with four processors and symmetric communication bandwidth.

$$T_{comm} = (M_{co} + M_{oc} + 2 \times M_{oo})\beta \tag{3.5}$$

$M_{co} =$ message size sent from centre processor to outer processors
$M_{oc} =$ message size sent from outer processors to centre processor
$M_{oo} =$ message size sent from outer processors to other outer processors

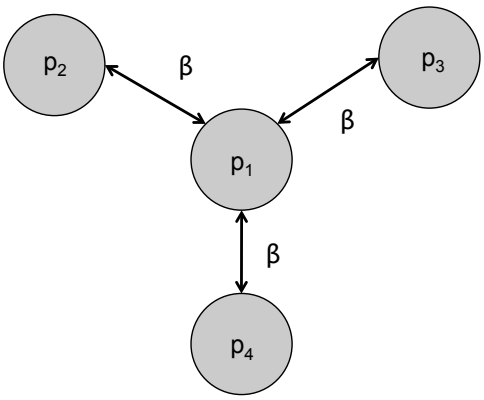


Figure 3.2: A star network topology with four processors and symmetric communication bandwidth.

3.2 Computation Modelling

The computation of matrix multiplication may be modelled in a straightforward way. Consider the most basic unit of computation to be the line SUMMA iterates over, $C[i,j] = A[i,k] * B[k,j] + C[i,j]$, one multiplication

and one addition. Each matrix element requires N of these units of computation to fully compute. The computation time in seconds, c_X, of Processor X is given by,

$$c_X = \frac{N * \#X}{S_X} \tag{3.6}$$

$$\begin{aligned} \#X &= \text{ number of elements assigned to Processor } X \\ N * \#X &= \text{ units of computation Processor } X \text{ is required to compute} \\ S_X &= \text{ units of computation per second achieved by Processor } X \end{aligned}$$

The value of S_X is quantifiable on all target systems using benchmarks designed to test system speed for linear algebra applications, such as High Performance LinPACK [24]. Many of the solutions found in this thesis will be independent of matrix problem size, so using a constant performance model is fully adequate.

Ratio of Computation to Communication speed

In some algorithms it will be useful to have some measure of computation speed of the system compared to overall communication speed. This is pegged to the fastest processor, known as P, and the communication speed β.

$$c = S_P * \beta \tag{3.7}$$

3.3 Memory Modelling

Matrix computations in scientific applications use large amounts of memory commensurate with the size of the matrix [60]. While some matrix computation algorithms such as the 3D Mesh use many redundant copies of the matrix to minimise communication, the state of the art SUMMA algorithm inherently uses less memory. As the SUMMA algorithm is presumed as the method of matrix computation, it is taken as an assumption that all abstract processors possess enough memory to store the necessary portions of matrices A, B, and C.

However, it is possible to imagine a real life processor, such as a GPU, which would have relatively little memory when compared to its high processing power. In this case, the portion of matrix C assigned to this processor can be divided into blocks, which are computed one at a time, and in an order which minimises extra communication.

3.4 Algorithm Description

There are many different ways to combine communication and computation time to create execution time. How this occurs is determined by the algorithm used to compute the matrix multiplication. The algorithm chosen directly alters the relative importance of communication and computation in determining execution time, and so will also effect the performance of each data partition shape. The five algorithms presented below attempt to encapsulate the characteristics of a wide variety of matrix multiplication algorithms in use, such as bulk communication, and interleaving communication and computation.

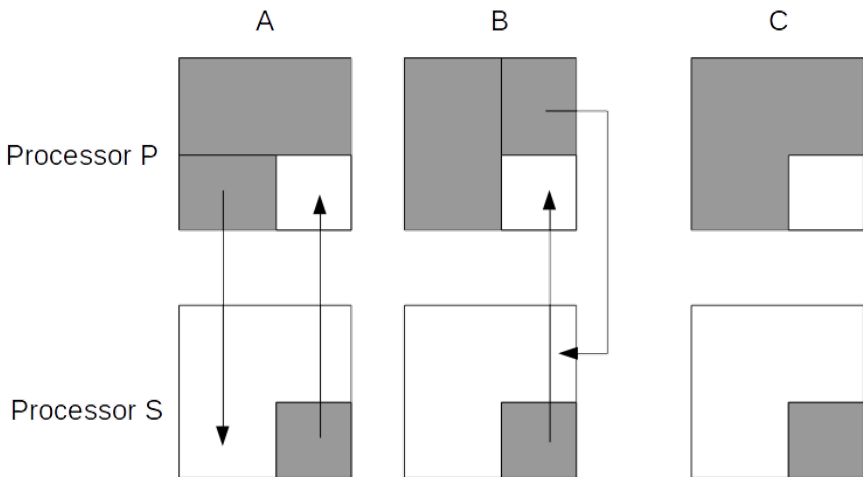


Figure 3.3: The communication pattern of the Square Corner two processor shape. Each processor requires data from the other processor, from both matrices A and B.

3.4.1 Bulk Communication with Barrier Algorithms

The first two algorithms are based on the idea of barriered bulk communications, meaning all processors send and receive all data before computation may begin.

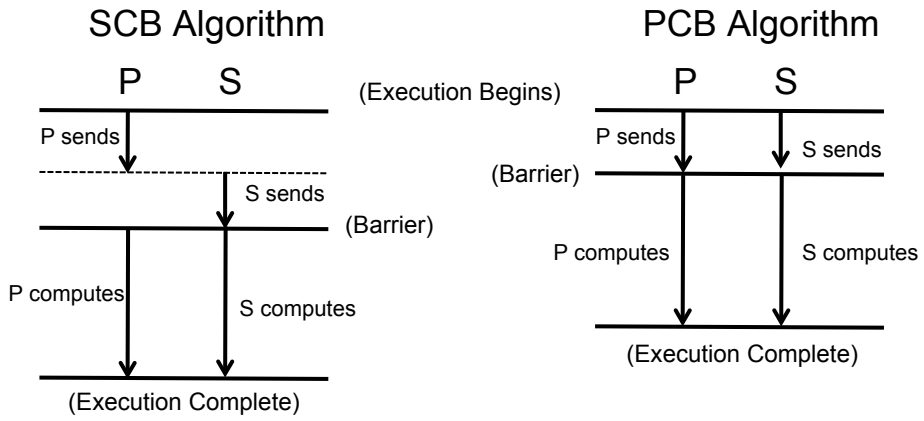


Figure 3.4: A depiction of the Serial Communication with Barrier (SCB) and Parallel Communication with Barrier (PCB) algorithms, for two processors, P and S. Time flows downward, with an arrow depicting when the given processor is active (sending data or computing), and no arrow indicating receiving data or an idle processor.

Serial Communication with Barrier

The first algorithm considered is the Serial Communication with Barrier (SCB). It is a simple matrix multiplication algorithm in which all data is sent by each processor *serially*, and only once communication completes among all processors does the computation proceed in *parallel* on each processor.

The execution time is given by,

$$T_{exe} = V\beta + \max(c_{P1}, c_{P2}, \cdots, c_{Pp}) \tag{3.8}$$

where V is the total volume of communication, and c_{Px} is the time taken to compute the assigned portion of the matrix on Processor X.

Parallel Communication with Barrier

The second algorithm considered is the Parallel Communication with Barrier. All data is sent among processors in *parallel*, and only once communication completes does the computation proceed in *parallel* on each processor.

$$T_{exe} = \max(v_{P1}, v_{P2}, \cdots, v_{Pp})\beta + \max(c_{P1}, c_{P2}, \cdots, c_{Pp}) \tag{3.9}$$

where v_Px is the volume of data elements which must be sent by Processor X.

3.4.2 Communication/Computation Overlap Algorithms

The final three algorithms attempt to overlap communication and computation where possible, in order to decrease execution time. The first two overlap algorithms describe partition shape specific overlap, which for some non-rectangular shapes, can allow computation to begin before communication is complete. The final algorithm will almost completely interleave communication and computation.

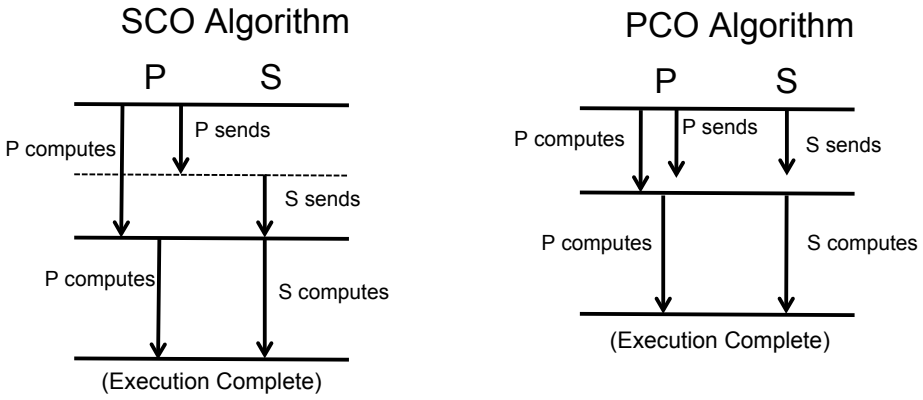


Figure 3.5: A depiction of the Serial Communication with Overlap (SCO) and Parallel Communication with Overlap (PCO) algorithms, for two processors, P and S. Processor P is shown to have a subsection of its matrix C which may be computed without communication. Time flows downward, with an arrow depicting when the given processor is active (sending data or computing), and no arrow indicating receiving data or an idle processor.

Serial Communication with Overlap

In the Serial Communication with Overlap (SCO) algorithm, all data is sent by each processor *serially*, while in *parallel* any elements that can be computed without communication are computed. Only once both communication and overlapped computation are complete does the remainder of the computation begin. The execution time is given by,

$$T_{exe} = \max\Big(\max(V\beta, o_{P1})+c_{P1}, \max(V\beta, o_{P2})+c_{P2}, \cdots, \max(V\beta, o_{Pp})+c_{Pp}\Big) \tag{3.10}$$

where o_{Px} is the number of seconds taken by Processor X to compute any elements not requiring communication, and c_{Px} is the number of seconds taken to compute the remainder of the elements assigned to Processor X.

Parallel Communication with Overlap

The Parallel Communication with Overlap (PCO) algorithm, completes all communication in *parallel*, while simultaneously computing any sections of matrix C which do not require interprocessor communication. Once these have finished, the remainder of the computation is carried out. The execution time is given by,

$$T_{exe} = \max\Big(\max(T_{comm}, o_{P1}) + c_{P1}, \max(T_{comm}, o_{P2}) + c_{P2}, \cdots , \max(T_{comm}, o_{Pp}) + c_{Pp}\Big) \quad (3.11)$$

where T_{comm} is the same as the PCB algorithm, o_{Px} is the number of seconds taken by Processor X to compute any elements not requiring communication, and c_{Px} is the number of seconds taken to compute the remainder of the elements assigned to Processor X.

Parallel Interleaving Overlap

The Parallel Interleaving Overlap (PIO) algorithm, unlike the previous algorithms described, does not use bulk communication. At each step data is sent, a row and a column (or k rows and columns) at a time, by the relevant processor(s) to all processor(s) requiring those elements, while, in *parallel*, all processors compute using the data sent in the previous step. The execution time for this algorithm is given by,

$$\begin{aligned} T_{exe} = {} & \text{Send } k + \\ & (N-1)\max\Big(V_k\beta, \max\left(k_{P1}, k_{P2}, \cdots , k_{Pp}\right)\Big) \\ & + \text{ Compute } (k+1) \end{aligned} \quad (3.12)$$

where V_k is the number of elements sent at step k, and k_X is the number of seconds to compute step k on Processor X.

Chapter 4

The Push Technique

The central contribution of this thesis is the introduction of the Push Technique. This novel method alters a matrix data partition, reassigning elements among processors, to lower the total volume of communication of the partition shape. The goal of using this technique is to prove that some random arrangement of elements is not the optimal shape. Instead, it allows the consideration of a few discrete partition shapes which are superior to all other data partitions (by virtue of the fact that no Push operation can be performed on them).

The Push Technique operates on an individual processor (although on a given Push any and all processors may be reassigned elements) and an individual row or column. That row or column, k, is determined by the direction of the Push (Up, Down, Back, or Over) and the enclosing rectangle of the processor being Pushed. An *enclosing rectangle* is an imaginary rectangle drawn around the elements of a given processor, which is strictly large enough to encompass all such elements, as seen in Figure 4.1. The edges of some Processor X's enclosing rectangle are known in clockwise order as $x_{top}, x_{right}, x_{bottom}, x_{left}$.

4.1 General Form

When applying the Push Technique to a data partition shape containing an arbitrary number of processors,

- Choose one processor, X, to be *Push*ed (must not be the most powerful processor)
- Choose the direction of Push, *i.e.* Up ($\uparrow$), Down ($\downarrow$), Back ($\leftarrow$), Over ($\rightarrow$)

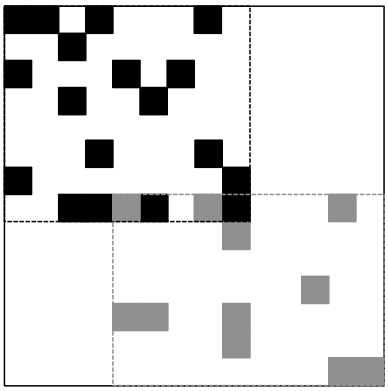


Figure 4.1: A matrix data partition among three processors, pictured in white, black, and grey. The enclosing rectangles for the black and grey processors are drawn as dotted lines. The enclosing rectangle of the white processor is the entire matrix.

- Determine the appropriate row or column k, the edge of the enclosing rectangle of X ($\uparrow$ acts on x_{bottom}, $\downarrow$ acts on x_{top}, $\leftarrow$ acts on x_{right}, and $\rightarrow$ acts on x_{left})
- For each element, x, assigned to Processor X in row or column k:
  1. Assign Processor X an element, z, within its enclosing rectangle
  2. Assign element x to the processor previously assigned z
- A valid Push may not increase the volume of communication, so select all z such that:
  1. Processor X is introduced to no more than one new row OR column
  2. No processor is assigned an element in k if k is outside that processor's enclosing rectangle
  3. A processor cannot be assigned an element in k, if it did not already own an element in k, unless doing so would also remove that processor from some other row or column

Note that this last item may also be achieved by considering a single Push as an atomic operation. If assigning several elements to the same new row or column results in all of those elements being removed from some other same row or column (thereby removing all elements of that processor), the volume of communication will be lowered or unchanged. This scenario is an acceptable Push.

4.2 Using the Push Technique to solve the Optimal Data Partition Shape Problem

The Push Technique may be applied iteratively to a data partition shape, incrementally improving it until a shape is reached on which no valid Push operations may be performed. Any such data partition, with no possible Push operations, is a candidate to be optimal and must be considered further.

In general, the Push Technique will be applied to some random starting partition. Push operations are performed until some local minima is found, where no further Push operations are possible. Those final states are the candidates considered throughout this thesis.

Consider a Deterministic Finite Automaton. This DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is the finite set of states, the possible data partitioning shapes
2. Σ is the finite set of the alphabet, the processors and the directions they can be Pushed
3. δ is $Q \times \Sigma \rightarrow Q$ the transition function, the Push operation
4. q_0, the start state, chosen at random
5. F is $F \subseteq Q$, the accept states, candidate partitions to be the optimum

The finite set of states, Q, is every possible permutation of the elements, assigned to each processor, within the $N \times N$ matrix. Therefore the number of states in the DFA is dependent on the size of the matrix, the number of processors and the relative processing speeds of those processors. The size of Q is given by

$$\frac{N^2!}{(\#P_1!) \times (\#P_2!) \times \cdots \times (\#P_p!)} \tag{4.1}$$

where,
$\#P_x$ is the number of elements assigned to Processor X

The finite set, Σ, called the alphabet, is the information processed by the transition function in order to move between states. Legal input symbols are the active processor being Pushed, Processor X, and the direction the elements of Processor X are to be moved, *i.e.* Up, Down, Over or Back.

The transition function, δ, is the Push operation. This function processes the input language Σ and moves the DFA from one state to the next, and

therefore the matrix from one partition shape to the next. The implementation of the transition function is discussed further in the next section. If the elements of the specified processor cannot be moved in the specified direction then the state is not changed, *i.e.* the transition arrow loops back onto the current state for that input.

The start state of the DFA, q_0, is chosen randomly.

Finally, the accept states F are those fixed points in which no Processor X may be Pushed in any direction. These states, and their corresponding partition shapes, must be studied further.

4.3 Application to Matrix Multiplication

The Push Technique, as described above for any number of processors, will also be detailed for specific applications of two and three processors, with further insight into how the Push can apply to four and more processors. This entire chapter is dedicated to the Push Technique as it applies to matrix multiplication, so it is worth noting now which parts are constant, and which change when applying to other matrix computations.

The Push Technique, so far as it is a methodology for incrementally altering a matrix partition for the better, is applicable to nearly any matrix computation. The difference, and what makes the remainder of this chapter specific to matrix multiply, is the performance model (volume of communication) the Push is operating under. The rules of the Push could, therefore, be altered to prejudice for some other metric(s) and create different matrix partitions for different computational needs.

Matrix multiplication is straightforward with easily parallelised computation and simple communication patterns, and as such is the best starting point for applying the Push Technique. However, any matrix computation which can be decomposed into some quantifiable incremental changes, can benefit from the Push Technique.

4.4 Push Technique on a Two Processor System

The two processor case is the simplest to which the Push Technique may be applied. This provides an excellent base case for describing the basics of the Push, before moving on to the more complex issues of three, four, and more processors.

This section first covers the algorithm used to carry out a single Push operation. The following section will show that continuous use of the Push, until no valid Push operations remain untaken, will result in a lower volume of communication, and thereby lower execution time, for all five matrix multiplication algorithms considered. Finally, the outcomes of these Push operations, the optimal candidates, are described.

4.4.1 Algorithmic Description

The algorithm used to accomplish the Push operation is given in more detail here. Each direction is deterministic, and moves element in a typewriter-like fashion, *i.e.* left to right, and top to bottom.

Elements assigned to S in row k, (set to s_{top}, in the example below), are reassigned to Processor P. Suitable elements are assigned to Processor S, from Processor P, from the rows below k and within the enclosing rectangle of Processor S. Assume S is the second processor, so $\phi(i,j) = 1$ if element (i,j) is assigned to Processor S.

Push Down

Formally, Push $\downarrow(\phi, k) = \phi_1$ where,

Initialise $\phi_1 \leftarrow \phi$
$(g,h) = (k+1, s_{left})$
for $j = s_{left} \rightarrow s_{right}$ **do**
  {If element belongs Processor S, reassign it.}
  if $\phi(k,j) == 1$ **then**
    $\phi_1(k,j) = 0$;
    $(g,h) =$ find(g, h); {Function defined below (finds new location).}
    $\phi_1(g,h) = 1$; {Assign new location to active processor.}
  end if
  $j \leftarrow j + 1$;
end for
find(g,h):
for $g \rightarrow s_{bottom}$ **do**
  for $h \rightarrow s_{right}$ **do**
    {If potential location belongs to other processor, hasn't been reassigned already, and is in a column already containing X.}
    if $\phi(g,h) == 0$ && $\phi_1(g,h) == 0$ && $c(\phi, S, h) == 1$ **then**
      return (g,h);
    end if

end for
$g \leftarrow g + 1$;
$h \leftarrow s_{left}$;
end for
return $\phi_1 = \phi$ {Could not find location, Push not possible in this direction.}

It is important to note that if no suitable $\phi(g, h)$ 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 $\downarrow (\phi, k) = \phi$.

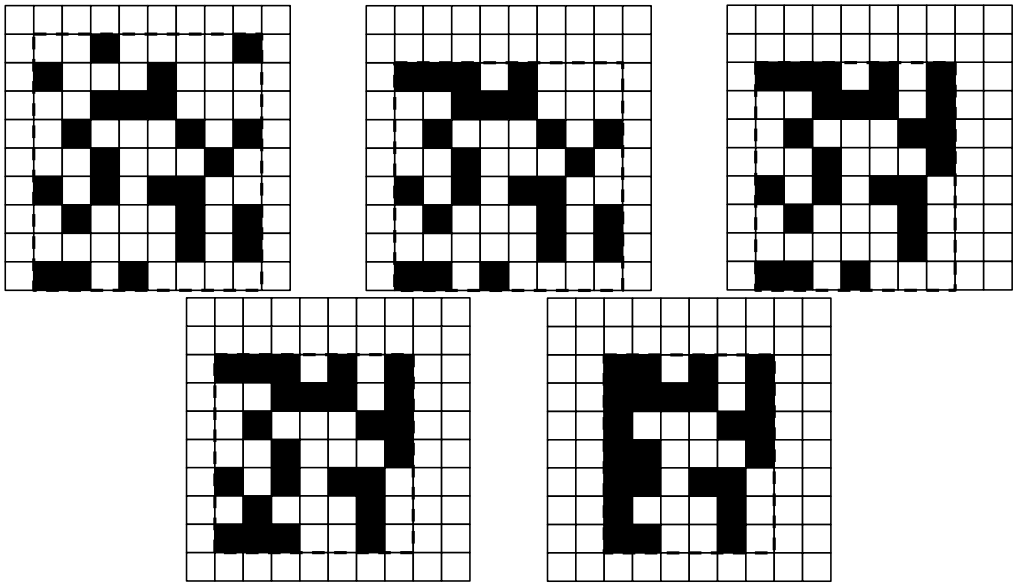


Figure 4.2: A 10×10 matrix partitioned between two processors, white and black. The first figure shows the starting partition. The second figure is after a Push Down has been performed. The third, after a Push Back. The fourth, after a Push Up. And the fifth and final figure shows after a Push Over has been performed.

The algorithmic descriptions for Push Up, Push Back, and Push Over, are similar in content and can be found in Appendix A.

4.4.2 Push: Lowering Communication Time for all Algorithms

The central idea of the Push Technique is that using it must not raise the execution time of any of the algorithms considered. After each Push step, the volume of communication must be lowered, or at least not increased. This

section demonstrates that the Push Technique will lower, or leave unchanged, the communication for each algorithm.

A data partition between two processors has a special metric,

$$\| \phi \|_x = \# \text{ of rows containing elements of } \textit{only one} \text{ processor in } \phi$$
$$\| \phi \|_y = \# \text{ of columns containing elements of } \textit{only one} \text{ processor in } \phi$$

Serial Communication.

Theorem 4.4.1 (Push)**.** *The Push Technique output partition, ϕ_1, will have lower, or at worst equal, communication time as the input partition, ϕ.*

Proof. First, observe several axioms related to the Push Technique.

Axiom 1. *Push $\downarrow$ and Push $\uparrow$, create a row, k, with no elements belonging to the Pushed Processor X, and may introduce Processor X to at most one row in ϕ_1 in which there were no elements of Processor X in ϕ. No more than one row can have elements of X introduced, as a row that was had no elements of X in ϕ will have enough suitable slots for all elements moved from the single row, k.*

Axiom 2. *Push $\downarrow$ and Push $\uparrow$ are defined to not add elements of Processor X to a column in ϕ_1 if there is no elements of X in that column of ϕ. However, these Push directions may create additional column(s) without X, if the row k being Pushed contains elements that are the only elements of X in their column, and there are sufficient suitable slots in other columns.*

Axiom 3. *Push $\rightarrow$ and Push $\leftarrow$ create a column, k, with no elements belonging to Processor X, and may create at most one column with X in ϕ_1 that did not contain X in ϕ.*

Axiom 4. *Push$\rightarrow$ and Push$\leftarrow$ will never add elements of X to a row in ϕ_1 that did not contain elements of X in ϕ, but may create additional row(s) without X.*

From (3.8) we observe as ($\| \phi \|_x + \| \phi \|_y$) increases, T_{comm} decreases.

Push $\downarrow$ or Push $\uparrow$ on ϕ create ϕ_1 such that:

For row k being pushed,
If there exists some row i that did *not* have elements of X, but now does:
$r(\phi, X, i) = 0$ and $r(\phi_1, X, i) = 1$
then by Axiom 1:
$$\| \phi_1 \|_x = \| \phi \|_x$$
else
$$\| \phi_1 \|_x = \| \phi \|_x + 1$$
and by Axiom 2:
$$\| \phi_1 \|_y \geq \| \phi \|_y$$

Push $\rightarrow$ or Push $\leftarrow$ on ϕ create ϕ_1 such that:

For column k being pushed,
If there exists some column j that did *not* have elements of X, but now does:,
$c(\phi, X, j) = 0$ and $c(\phi_1, X, j) = 1$
then by Axiom 3:
$$\| \phi_1 \|_y = \| \phi \|_y$$
else
$$\| \phi_1 \|_y = \| \phi \|_y + 1$$
and by Axiom 4:
$$\| \phi_1 \|_x \geq \| \phi \|_x$$

By these definitions of all Push operations we observe that for any Push operation, $(\| \phi_1 \|_x + \| \phi_1 \|_y) \geq (\| \phi \|_x + \| \phi \|_y)$. Therefore, it is concluded that all Push operations will either decrease communication time (3.8) or leave it unchanged. □

The proof for parallel communication is similar and may be found in Appendix B.

4.4.3 Two Processor Optimal Candidates

Repeatedly applying Push operations will result in a discrete set of partition shapes. These fifteen shapes, shown in Figure 4.3, are formed using combinations of either one, two, three, or four directions of the Push operation.

The Push Technique consolidates the elements assigned to the less powerful processor into a rectangle, minimising the number of rows and columns containing elements of both processors, and thereby minimising the communication time. Of the possible dimensions for this rectangle, there exist two broad categories. First, rectangles with one dimension of N, and second, rectangles with both dimensions less than N.

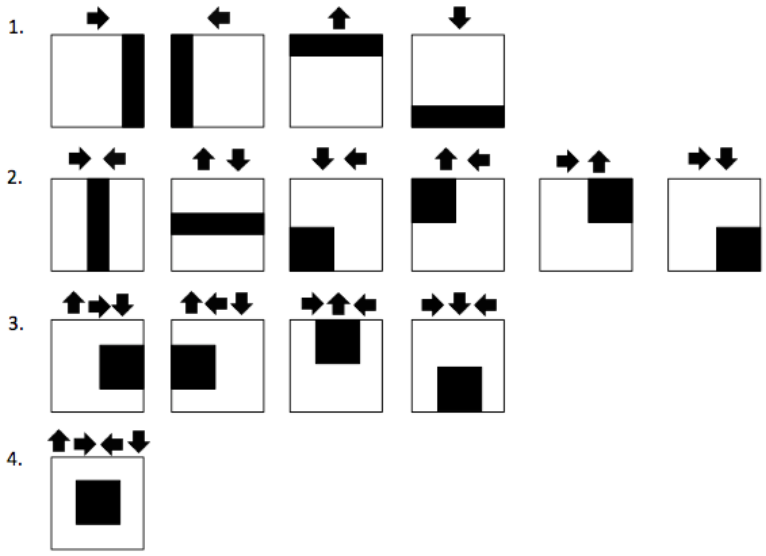


Figure 4.3: The result of applying operations $\downarrow$, $\uparrow$, $\leftarrow$, and $\rightarrow$, until the stopping point has been reached. Row 1 shows the result of applying just a single transformation. Row 2 shows the result of applying a combination of two transformations. Row 3 shows the possible results of applying three transformations, and Row 4 shows the result of applying all four transformations.

Within these broad categories of rectangle dimensions, the location of the rectangle within the matrix does not effect the volume of communication (because it does not increase the number of rows or columns containing elements of both processors). The portion assigned to Processor S may be moved within the matrix to create, from the original fifteen outputs, just two candidate shapes in canonical form.

More formally, this is stated as the following theorem, the proof of which is discussed in Appendix B.3.

Theorem 4.4.2 (Canonical Form)**.** *Any partition shape, for two processor matrix multiplication, in which Processor S has an enclosing rectangle of the same dimensions x, y has the same communication cost.*

The two canonical shapes are named as Straight Line and Square Corner respectively, and can be seen in Figure 5.1.

Chapter 5

Two Processor Optimal Partition Shape

The optimal partition shape for two abstract processors was a natural starting point for the Push Technique, which was used to create candidate optimal shapes in Chapter 4. The optimal candidates are known as Square Corner and Straight Line.

In this chapter, the candidates are analysed to determine which is the optimal shape. The Square Corner shape, which is non-rectangular, is shown to be optimal for a large range of ratios of processor computational power. Accordingly, the Straight Line shape, which is rectangular, is optimal for homogeneous systems, and small amounts of heterogeneity in processor speed.

In the final section of this chapter, these theoretical results are confirmed experimentally.

In order to determine which shape is optimal, first, the candidates are analysed to determine the volume of communication each requires. Then, using the model of an abstract processor discussed in Chapter 3, performance models are built for both shapes, for each algorithm. Finally, these performance models are directly compared to prove mathematically which candidate is the optimal shape.

5.1 Optimal Candidates

Applying the Push Technique results in two potential shapes with elements divided between Processors P and S. In the first candidate shape Processor S is assigned a rectangle of length N to compute (Straight Line), and in the second candidate shape Processor S is assigned a rectangle of less than length N to compute (Square Corner).

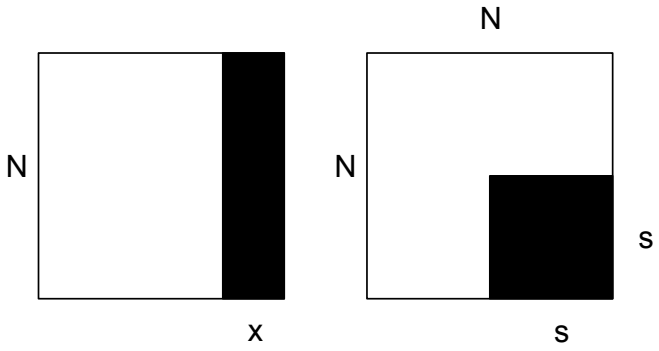


Figure 5.1: The optimal candidates found for two processors using the Push Technique, divided between Processor P (in white) and Processor S (in black). On the left is the Straight Line shape, on the right is the Square Corner shape.

In the second candidate shape, the Push Technique can create any size rectangle of length less than N. However, the optimal dimensions of this rectangle occur when its width equals its length, *i.e.* when it is square [57].

Volume of Communication

In the Straight Line shape, the necessary communication is,

- $P \rightarrow S$ a data volume of $N(N - x)$ elements
- $S \rightarrow P$ a data volume of Nx elements

In the Square Corner shape, the necessary communication is,

- $P \rightarrow S$ a data volume of $2s(N - s)$ elements
- $S \rightarrow P$ a data volume of $2s^2$ elements

The value of x and s

Unless otherwise stated, each algorithm, for all shapes, is assumed to begin computation on P and S at the same time. Therefore, the area of the matrix, the volume of elements, assigned to each processor is in proportion to its computational power.

$$x = \frac{1}{r+1} \tag{5.1}$$

$$s = \frac{1}{\sqrt{r+1}} \tag{5.2}$$

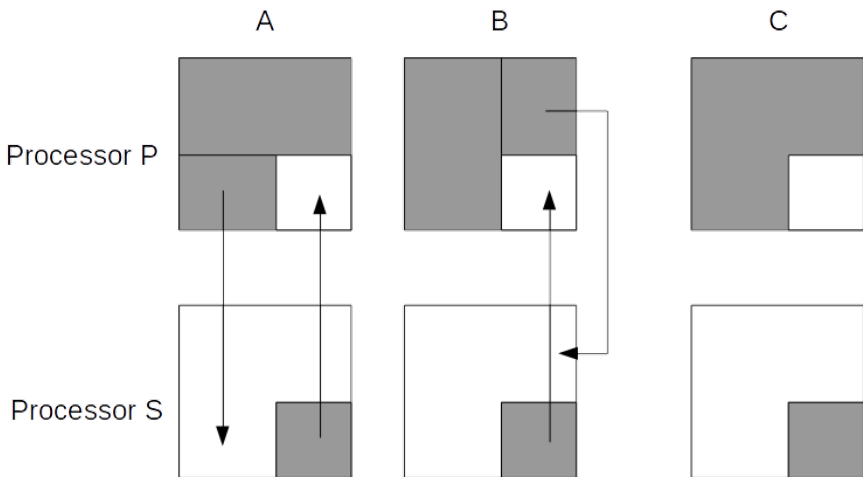


Figure 5.2: The communication pattern of the Square Corner two processor shape. Each processor requires data from the other processor, from both matrices A and B.

5.2 Applying the Abstract Processor Model

In the following sections, the models for both candidate shapes are derived for each algorithm.

Serial Communication with Barrier

The two processor Straight Line SCB shape execution time is given by,

$$T_{exe(SL)} = (N^2)\beta + \max(c_P, c_S) \tag{5.3}$$

The two processor Square Corner SCB shape execution time is given by,

$$T_{exe(SC)} = (2Ns)\beta + \max(c_P, c_S) \tag{5.4}$$

Parallel Communication with Barrier

The two processor Straight Line PCB shape execution time is given by,

$$T_{exe(SL)} = \max(N(N-x), Nx)\beta + \max(c_P, c_S) \tag{5.5}$$

The two processor Square Corner PCB shape execution time is given by,

$$T_{exe(SC)} = \max(2s(N-s), 2s^2)\beta + \max(c_P, c_S) \tag{5.6}$$

Serial Communication with Overlap

In the Serial Communication with Overlap algorithm, any portion of the result matrix C which can be computed without communication is done in parallel with the communication. The Straight Line shape does not have any such portion of the result matrix C, so the cost is the same with SCO as with SCB.

The Square Corner shape, however, does have a section that can be computed without communication, seen in Figure 5.3.

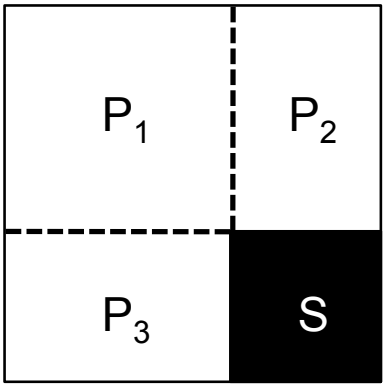


Figure 5.3: The Square Corner shape, divided into sections with dotted lines to show which portions do not require communication. Processor P owns all necessary data to compute section P_1. Computing sections P_2, P_3, and S, however, requires communication.

Recalling Equations 3.6 and 3.10, for computation time and the SCO algorithm respectively, the Straight Line SCO execution time can be written as,

$$T_{exe(SL)} = \max\left(\max\left(N^2\beta, 0\right) + \frac{N(N(N-x))}{S_P}, \max\left(N^2\beta, 0\right) + \frac{N(Nx)}{S_S}\right) \tag{5.7}$$

And the Square Corner execution time is given by,

$$T_{exe(SC)} = \max\left(\max\left((2Ns)\beta, \frac{N(N-s)^2}{S_P}\right) + \frac{2Ns(N-s)}{S_P}, \right. \\ \left. \max\left((2Ns)\beta, 0\right) + \frac{N(s^2)}{S_S}\right) \tag{5.8}$$

These equations will be simpler to analyse after removing the constant common factor $N^3\beta$, normalising s and x as a proportion of N so that $s = \frac{s}{N}$ and $x = \frac{x}{N}$, such that s is understood to be $0 < s < 1$ (and x is understood to be $0 < x < 1$), and introducing the value c (given in Equation 3.7).

The Straight Line execution time is now (understanding the optimal size of x to still be $\frac{1}{r+1}$),

$$\frac{T_{exe(SL)}}{N^3\beta} = \frac{1}{N} + \frac{1-x}{c} \tag{5.9}$$

The Square Corner execution is now,

$$\frac{T_{exe(SC)}}{N^3\beta} = \max\left(\max\left(\frac{2s}{N}, \frac{(1-s)^2}{c}\right) + \frac{2(s-s^2)}{c}, \quad \frac{2s}{N} + \frac{s^2 r}{c}\right) \tag{5.10}$$

The Optimal Size of s for SCO. The optimal value of s is the minimum of this $\frac{T_{exe(SC)}}{N^3\beta}$ on the interval of $\{0,\ 1\}$. However, since a value of $s = 1$ would indicate that Processor S has been assigned the entire matrix, the interval of possible s values can be made more specific. The largest s will be without overlap is when $r = 1:1$, and therefore $s = \frac{1}{\sqrt{2}}$. It has already been established that overlap algorithms will decrease the area assigned to S, so it can certainly be said that the optimal value of q is the minimum of $\frac{T_{exe(SC)}}{N^3\beta}$ on the interval $\{0, \frac{1}{\sqrt{2}}\}$.

There are 3 functions that comprise the $\frac{T_{exe(SC)}}{N^3\beta}$ equation. These functions and what they represent are as follows,

$$y = \frac{2s}{N} + 2\frac{s-s^2}{c}: \quad T_{comm} + (P_2 + P_3) \tag{5.11}$$

$$y = \frac{(1-s)^2}{c} + 2\frac{s-s^2}{c}: \quad P_1 + (P_2 + P_3) \tag{5.12}$$

$$y = \frac{2s}{N} + \frac{s^2 r}{c}: \quad T_{comm} + S \tag{5.13}$$

The first observation is that (5.11) is always less than (5.12) on the interval $\{0, \frac{1}{\sqrt{2}}\}$. Therefore, for possible values of s, it will never dominate the maximum function and can be safely ignored. Focusing on (5.12) and (5.13),

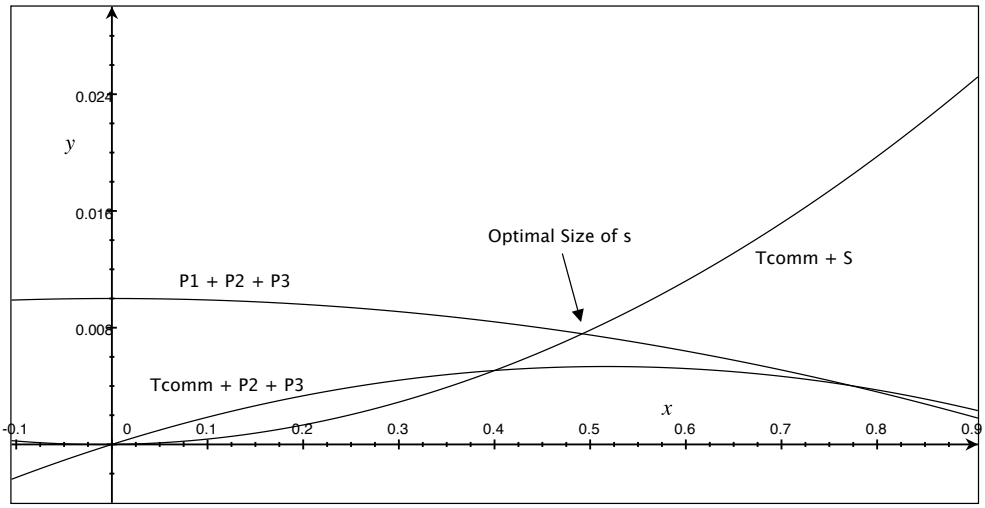


Figure 5.4: Graph of Equations 5.11, 5.12, and 5.13, the three possible functions to dominate the maximum function for execution time of the Square Corner shape using the Serial Communication with Overlap algorithm. In this example, problem size $N = 3000$, Processor ratio $r = 3$, and computation/communication ratio $c = 100$.

note that (5.12) is concave down and (5.13) is concave up, and therefore the minimum on the interval will be at the intersection of these two functions.

$$(5.12) \cap (5.13)$$

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

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

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

Parallel Communication with Overlap

The Parallel Communication with Overlap algorithm uses the same approach as the SCO algorithm, however the T_{comm} will be the same as the Parallel Communication with Barrier algorithm.

Recalling Equations 3.6 and 3.11, for computation time and the PCO algorithm respectively, the Straight Line PCO execution time can be written as,

$$T_{exe(SL)} = \max\Bigg(\max\big(\max(N(N-x), Nx)\beta, 0\big) + \frac{N(N(N-x))}{S_P}, \\ \max\big(\max(N(N-x), Nx)\beta, 0\big) + \frac{N(Nx)}{S_S}\Bigg) \quad (5.14)$$

And the Square Corner execution time is given by,

$$T_{exe(SC)} = \max\Bigg(\max\Big(\max\big(2s(N-s), 2s^2\big)\beta, \frac{N(N-s)^2}{S_P}\Big) + \frac{2Ns(N-s)}{S_P}, \\ \max\Big(\max\big(2s(N-s), 2s^2\big)\beta, 0\Big) + \frac{N(s^2)}{S_S}\Bigg) \quad (5.15)$$

As with the SCO algorithm, remove the constant common factor $N^3\beta$, normalise s and x as a proportion of N so that $s = \frac{s}{N}$ and $x = \frac{x}{N}$, and introduce the value c given in Equation 3.7.

The updated Straight Line execution time is given by,

$$\frac{T_{exe(SL)}}{N^3\beta} = \max\Big(\frac{1-x}{N}, \frac{x}{N}\Big) + \max\Big(\frac{1-x}{c}, \frac{rx}{c}\Big) \quad (5.16)$$

And the updated Square Corner execution time is given by,

$$\frac{T_{exe(SC)}}{N^3\beta} = \max\Bigg(\max\Big(\max\big(\frac{2s-2s^2}{N}, \frac{2s^2}{N}\big), \frac{(1-s)^2}{c}\Big) + \frac{2(s-s^2)}{c}, \\ \max\Big(\frac{2s-2s^2}{N}, \frac{2s^2}{N}\Big) + \frac{s^2 r}{c}\Bigg) \quad (5.17)$$

The Optimal Size of s for PCO. As with the SCO algorithm, the amount of the matrix assigned to Processor P in the Square Corner shape is increased to account for the "jumpstart" that Processor P gets on computing its portion of the matrix. The optimal size to set the square of Processor S, s, is found by examining the five constituent functions which make up the maximum execution time function for this shape.

$$y_1 = \frac{2s - 2s^2}{N} + 2\frac{(s - s^2)}{c} : \quad (v_P \to v_S) + (P_2 + P_3) \tag{5.18}$$

$$y_2 = \frac{2s^2}{N} + 2\frac{(s - s^2)}{c} : \quad (v_S \to v_P) + (P_2 + P_3) \tag{5.19}$$

$$y_3 = \frac{(1 - s)^2}{c} + 2\frac{(s - s^2)}{c} : \quad P_1 + (P_2 + P_3) \tag{5.20}$$

$$y_4 = \frac{2s - 2s^2}{N} + \frac{rs^2}{c} : \quad (v_P \to v_S) + S \tag{5.21}$$

$$y_5 = \frac{2s^2}{N} + \frac{rs^2}{c} : \quad (v_S \to v_P) + S \tag{5.22}$$

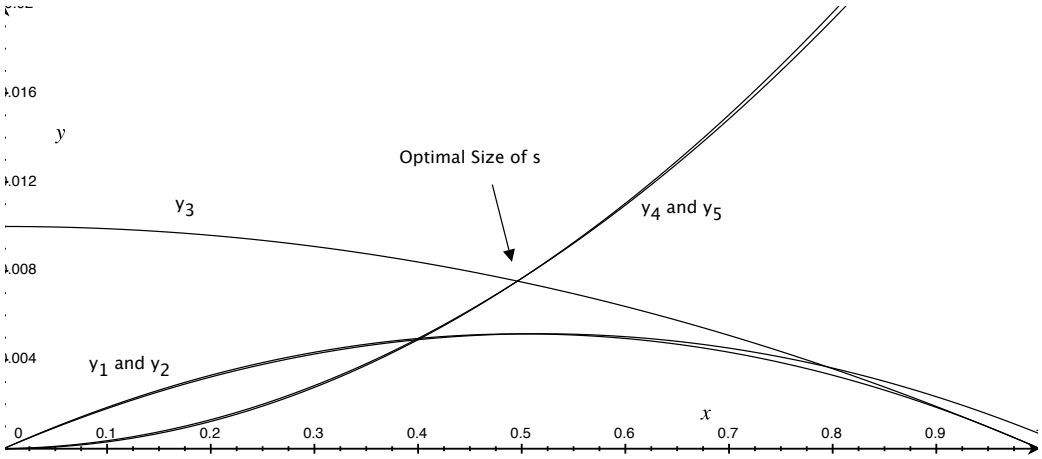


Figure 5.5: Graph of Equations 5.11, 5.12, and 5.13, the three possible functions to dominate the maximum function for execution time of the Square Corner shape using the Parallel Communication with Overlap algorithm. In this example, problem size $N = 3000$, Processor ratio $r = 3$, and computation/communication ratio $c = 100$.

Both (5.18) and (5.19) are less than (5.20) on the interval $\{0, \frac{1}{\sqrt{2}}\}$, and can be safely ignored. Of the remaining 3 equations, (5.20) is concave down and both (5.21) and (5.22) are concave up on the interval. The optimal value of s, the minimum, is therefore at the intersection of (5.20), and whichever other function dominates. For $s < \frac{1}{2}$, (5.21) dominates and for $s > \frac{1}{2}$ (5.22) dominates. It will be shown in the next section that the Square Corner shape is optimal for ratios greater than $2 : 1$ when using parallel communication. Ratios less than and equal to $2 : 1$, will have s values greater than $\frac{1}{2}$, so the optimal value of s for the comparison is at (5.20) $\cap$ (5.22), which will

give the Square Corner the optimal s value when facing the Straight Line on homogeneous systems.

$$(5.20) \cap (5.22)$$
$$\frac{(1-s)^2}{c} + 2\frac{(s-s^2)}{c} = \frac{2s^2}{N} + \frac{rs^2}{c}$$
$$s = \frac{1}{\sqrt{r+1+\frac{2c}{N}}}$$

Parallel Interleaving Overlap

For any given step k, the total amount of data being sent using this algorithm on a Square Corner partition will be s. The execution time of the Square Corner partition is given by,

$$T_{exe(SC)} = 2s\beta + (N-1) * \max\left(2s\beta, \frac{N^2-s^2}{S_P}, \frac{s^2}{S_S}\right) + \max\left(\frac{N^2-s^2}{S_P}, \frac{s^2}{S_S}\right) \tag{5.23}$$

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,

$$T_{exe(SL)} = N\beta + (N-1) * \max\left(N\beta, \frac{N(N-x)}{S_P}, \frac{Nx}{S_S}\right) + \max\left(\frac{N(N-x)}{S_P}, \frac{Nx}{S_S}\right) \tag{5.24}$$

5.3 Optimal Two Processor Data Partition

The optimal two processor matrix multiplication shape is the Square Corner when the processor power ratio is

- greater than three to one $(3:1)$ for SCB and PIO algorithms
- greater than two to one $(2:1)$ for PCB algorithm
- for all ratios for SCO and PCO algorithms

For all other ratios the Straight Line shape is the optimal. These results were previously published in [57]. The proofs for these claims follow.

Serial Communication with Barrier

Theorem 5.3.1 (2 Processor MMM - SCB)**.** *For matrix multiplication with two processors, using the Serial Communication with Barrier algorithm, the Square Corner partition shape is optimal for all computational power ratios, r, greater than $3:1$, and the 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 $2Ns$. We state that $2Ns < N^2$ subject to the conditions $N, s > 0$. The optimal value of s is given by $s = \frac{N}{\sqrt{r+1}}$. Substituting this in, yields:

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

Therefore, the Square Corner shape is optimal for all $r > 3:1$, and the Straight Line shape is optimal for all $r < 3:1$. □

Parallel Communication with Barrier

Theorem 5.3.2 (2 Processor MMM - PCB)**.** *For matrix multiplication with two processors, using the Parallel Communication with Barrier algorithm, the Square Corner partition shape is optimal for all computational power ratios, r, greater than $2:1$, and the Straight Line partition shape is optimal for all ratios less than $2:1$.*

Proof. For all power ratios, the communication volume for the Straight Line partition shape is $N^2 - Nx$, where x is the dimension of Processor S'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 S, $VP = 2Ns - 2s^2$ or Ss to P, $VS = 2s^2$, dominates. $VP > VS$ when $r > 3:1$. Therefore, we compare Square Corner's VS to Straight Line. For the

conditions $N, s, x > 0$:

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

□

Serial Communication with Overlap

Theorem 5.3.3 (2 Processor MMM - SCO)**.** *For matrix multiplication with two processors, using the Serial Communication with Overlap algorithm, the Square Corner partition shape is optimal, with a lower total execution time than the Straight Line partition shape, for all processor power ratios.*

Proof.

$$\begin{aligned} \text{Straight-Line Execution} &> \text{Square-Corner Execution} \\ \frac{1}{N} + \frac{1-x}{c} &> \frac{(1-s)^2}{c} + 2\frac{s-s^2}{c} \\ s^2 &> x - \frac{c}{N} \\ (\frac{\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1}}{r+1})^2 &> \frac{1}{r+1} - \frac{c}{N} \\ (\frac{-c}{N} + \sqrt{\frac{c^2}{N^2} + r + 1})^2 &> (r+1) - \frac{c}{N}(r+1)^2 \\ \frac{c^2}{N^2} - \frac{2c}{N}\sqrt{\frac{c^2}{N^2} + r + 1} + \frac{c^2}{N^2} + r + 1 &> \\ r + 1 - \frac{c}{N}(r+1)^2 & \\ \frac{2c}{N} + (r+1)^2 &> 2\sqrt{\frac{c^2}{N^2} + r + 1} \\ \frac{4c}{N}(r+1)^2 + (r+1)^4 &> 4(r+1) \\ \frac{4c}{N} + r^3 + 3r^2 + 3r &> 3 \end{aligned}$$

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

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

Therefore, by taking advantage of the overlap-ready layout of the Square Corner partition shape, the Square Corner shape is optimal for all processor power ratios for two processors. □

Parallel Communication with Overlap

Theorem 5.3.4 (PCB)**.** *For matrix multiplication with two processors, using the parallel communication with overlap algorithm, the Square Corner partition shape is optimal, having a lower total execution time than the Straight Line partition shape for all processor power ratios.*

Proof. The Straight Line partition shape has 4 functions which make up the execution time, of which only two dominate when $x < \frac{1}{2}$, which must always be true (as slower processor is always called S). 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 $x = \frac{1}{r+1}$.

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

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. □

Parallel Interleaving Overlap

Theorem 5.3.5 (PIO)**.** *For matrix multiplication with two processors, using the Parallel Interleaving Overlap algorithm the Square Corner partition shape is optimal for computational power ratios, r, greater than $3:1$, and the Straight Line shape is optimal for ratios less than $3:1$.*

Proof. These equations can be given the same treatment as previously in the SCO and PCO algorithms, removing the constant $N^3\beta$ and normalising x, s to $\frac{x}{N}$ and $\frac{s}{N}$ respectively. First we consider the values of c for which the communication time dominates. This occurs at $c > N(1-x)$ for the Straight Line shape and $c > \frac{N}{2}(\frac{1}{s} - s)$ for the Square Corner shape. When this occurs the execution times may be given by,

$$\frac{T_{exe(SC)}}{N^3\beta} = \frac{2s}{N} + \frac{1-s^2}{c} \tag{5.25}$$

$$\frac{T_{exe(SL)}}{N^3\beta} = \frac{1}{N} + \frac{1-x}{c} \tag{5.26}$$

Begin by stating that for the given optimal values of x and s, the Straight Line execution time is greater than the Square Corner,

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

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

However, when c is such that the computation time dominates the execution time, the formulas are,

$$\frac{T_{exe(SC)}}{N^3\beta} = \frac{2s}{N^2} + \frac{1-s^2}{c} \tag{5.27}$$

$$\frac{T_{exe(SL)}}{N^3\beta} = \frac{1}{N^2} + \frac{1-x}{c} \tag{5.28}$$

Stating that for the given optimal values of x and s, the Straight Line shape

has a greater execution time than the Square Corner shape,

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

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

5.4 Experimental Results

These theoretical claims of optimality can be verified and reinforced by experimental data. Due to the previous work by [52, 53] regarding the Square Corner partition shape for two processors, a wealth of experimental data exists to conclude that the optimal shape conforms to the theoretical prediction. Additional results, as first published in [57], for each of the matrix multiplication algorithms, are discussed here.

5.4.1 Experimental Setup

The Square Corner and Straight Line partition shapes were implemented using C and MPI, with local matrix multiplications completed using ATLAS [61]. All experiments were conducted on two identical processors, with heterogeneity created between the two nodes by using a program to limit available CPU time on a single node. This program, cpulimit [62], forces the relevant process to sleep when a specified percentage of CPU time has been reached using the /proc filesystem (the same information available to programs like top). The process is awoken when a suitable amount of time has passed, and runs normally. This tool provides a fine grained control over the CPU power available on each processor.

The results in this section 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 heterogeneity is achieved by lowering

the capacity of a single processor, the higher the ratio, the lower the overall computational power of the system.

5.4.2 Results by Algorithm

Serial Communication with Barrier

The expected result, when using the Serial Communication with Barrier algorithm, is that the Square Corner will be superior for processor ratios greater than $3 : 1$, while the Straight Line shape is superior for ratios less than $3 : 1$. The theoretical curves of both shapes, representing volume of communication, can be seen in Figure 5.6. The Straight Line shape has a constant volume of communication (for a given problem size) regardless of power ratio, while the Square Corner shape is decreasing as the power ratio increases.

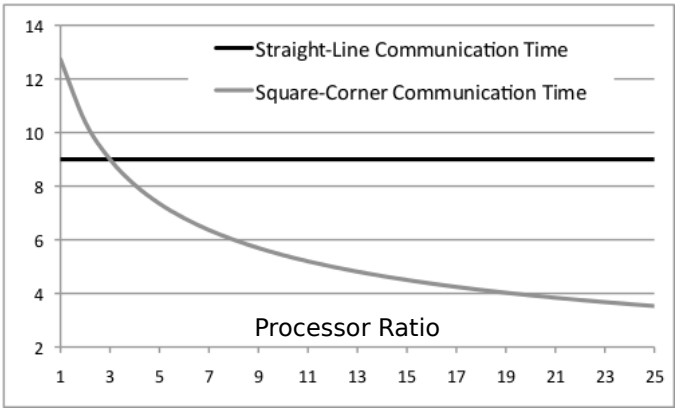


Figure 5.6: Straight Line and Square Corner theoretical communication time using the SCB algorithm.

Experimental results were obtained for both the Square Corner and Straight Line shapes for computational power ratios, r, from 1 to 25, as show in Figure 5.7. These results confirm the validity of the model, conforming well to the expected shape, with the communication times crossing at $r = 3$. Although experimental results for $N = 3000$ are shown, the ratio at which Square Corner is superior to Straight Line is not dependant on the size of the matrix. Additionally, note that as the level of heterogeneity increases (with the computational ratio headed towards $25 : 1$) the Square Corner continues to decrease it's necessary computation (as the size of the square decreases).

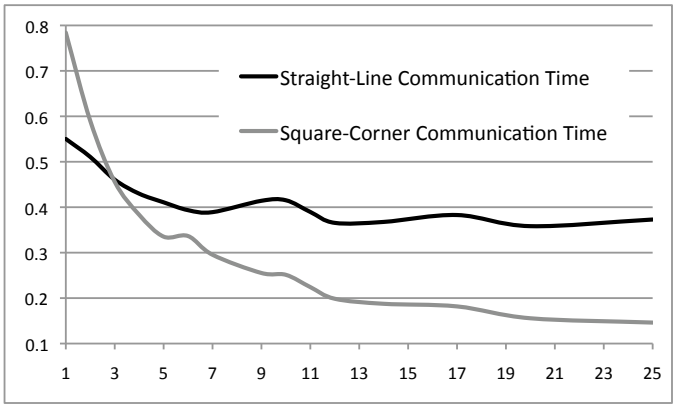


Figure 5.7: Experimental results of the Straight Line and Square Corner shape serial communication time, using the SCB algorithm, in seconds by computation power ratio. N=3000.

Parallel Communication with Barrier

For the Parallel Communication with Barrier algorithm, the theoretical results suggest that the Square Corner shape is optimal for power ratios greater than $2:1$, as shown in Figure 5.8.

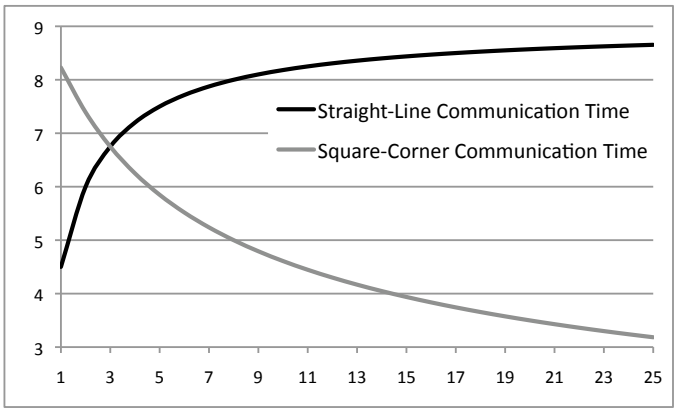


Figure 5.8: The theoretical prediction of Square Corner and Straight Line parallel communication times using the PCB algorithm.

The experimental results for ratios, r, from 1 to 25 are shown in Figure 5.9. As expected for small ratios of computational power, the Straight Line partition shape is optimal. As the ratio grows, the communication time drops significantly for the Square Corner shape, and it becomes the optimal solution.

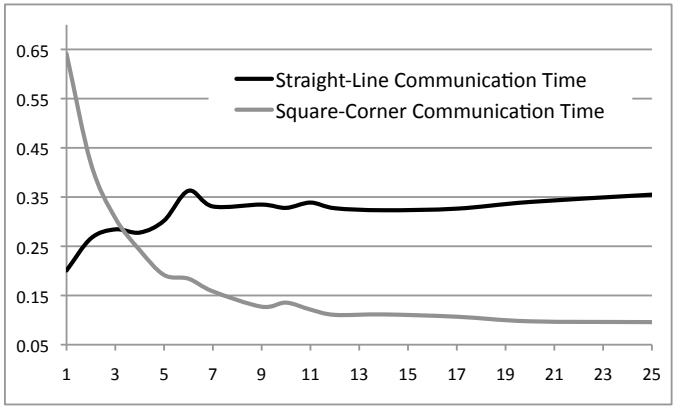


Figure 5.9: The experimental Square Corner and Straight Line parallel communication times, using the PCB algorithm, in seconds by computational power ratio. N=3000.

Serial Communication with Overlap

Under the Serial Communication with Overlap algorithm, the Square Corner is predicted to have a lower execution time for all computational power ratios, r, which is confirmed by the experiments. The focus should be on the ratios, $r < 3:1$, that were previously optimal for the Straight Line shape. However, allowing the Square Corner to take advantage of the portion available for computational overlap, makes it optimal for ratios of $r < 3:1$, as seen in Figure 5.10.

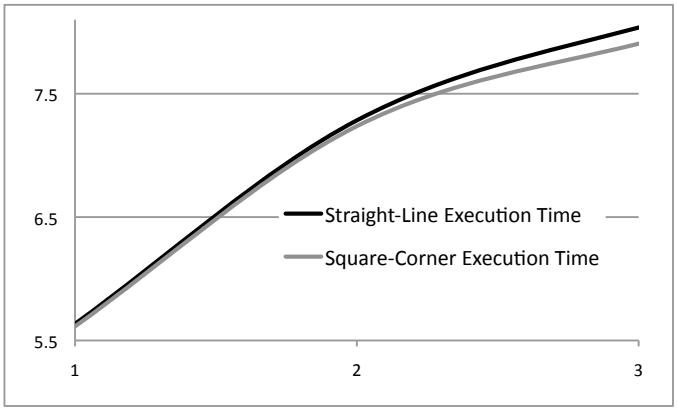


Figure 5.10: Experimental execution times for the Square Corner shape and the Straight Line shape, using the SCO algorithm, for small ratios $r < 3:1$, in seconds by computational power ratio. N=3000.

For those ratios, $r > 3:1$, in which the Square Corner was the optimal

for the SCB algorithm, the amount by which it is superior to the Straight Line shape increases with the SCO algorithm, as seen in Figure 5.11.

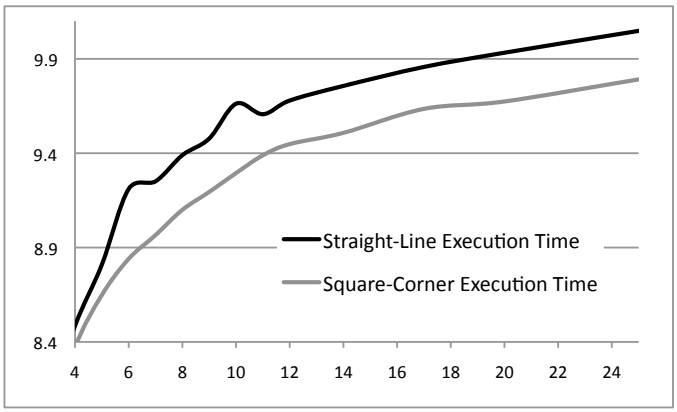


Figure 5.11: Experimental Square-Corner and Straight-Line execution times, using SCO, for large ratios $r > 3:1$, in seconds by power ratio. N=3000.

Parallel Communication with Overlap

The theoretical results predict that the Square Corner shape should be optimal for all power ratios when using the Parallel Communication with Overlap algorithm. Recall, when using PCB, the Straight Line is optimal for ratios two and under by a significant margin. Using PCB on the Square Corner partition shape has closed that gap. In Figure 5.12, 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 increases between the two processors, the benefit of overlapping communication and computation becomes more marked. In Figure 5.13, the results from $r \geq 4:1$ show the Square Corner shape outperforming the Straight Line as expected.

Parallel Interleaving Overlap

The experimental results for the Parallel Interleaving Overlap algorithm support the theoretical results which state that the Square Corner partition shape has a lower execution time for power ratios, $r > 3$. For ratios smaller than that, the Straight Line partition shape has the lower execution time.

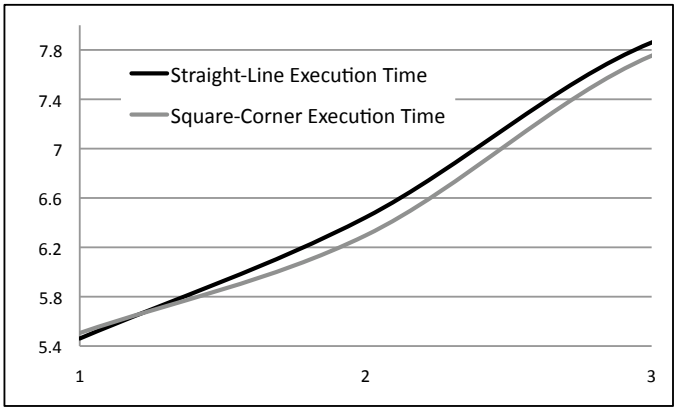


Figure 5.12: Experimental Square Corner and Straight Line shape execution times, using the PCO algorithm, for ratios $r \leq 3 : 1$ in seconds by computational power ratio. N=3000.

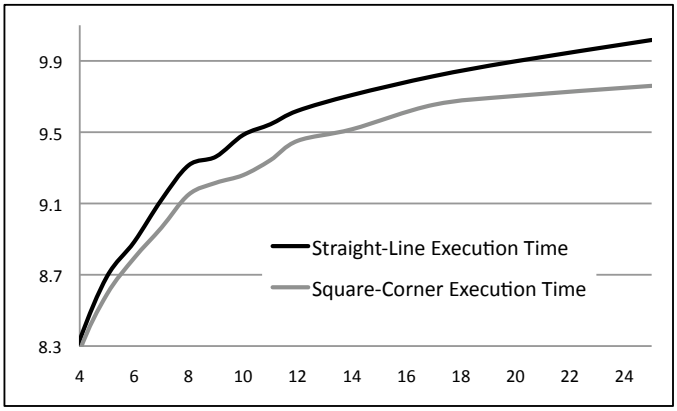


Figure 5.13: Experimental Square Corner and Straight Line shape execution times, using the PCO algorithm, for ratios $r \geq 4 : 1$ in seconds by computational power ratio. N=3000.

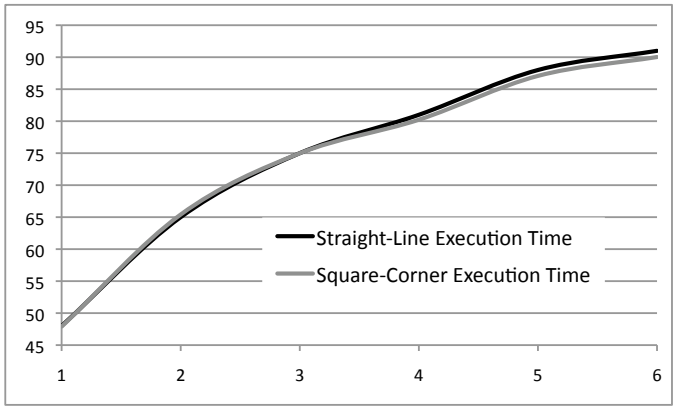


Figure 5.14: Experimental Square Corner and Straight Line shape execution times, using the PIO algorithm, when the communication-computation ratio, c, is such that computation dominates for all ratios, in seconds by computational power ratio. N=3000. The two partition shapes are equivalent.

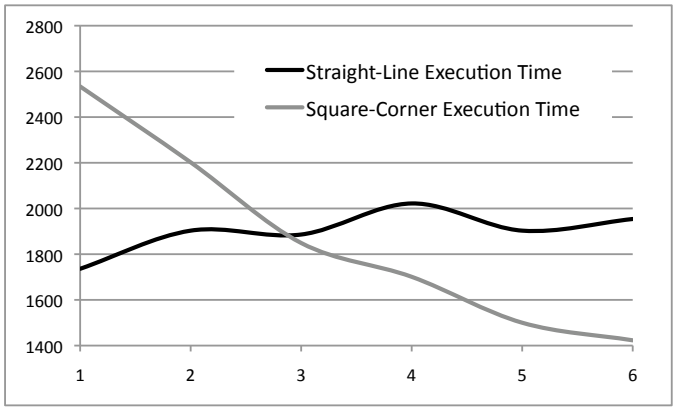


Figure 5.15: Experimental Square Corner and Straight Line shape execution times, using the PIO algorithm, when the communication-computation ratio, c, is such that communication dominates for all ratios, in seconds by computational power ratio. N=3000. The Straight Line partition shape is optimal for power ratios, r, less than $3:1$, and Square Corner is optimal for power ratios, r, greater than $3:1$.

Chapter 6

The Push Technique Revisited: Three Processors

Extending the Push Technique to three processors requires additional rules, as compared to two processors, which govern the Push in order to maintain the guarantee that the volume of communication, and thereby the time of execution, will not be increased. This section will describe these additional constraints on the three processor Push, and describe the software tool necessary to carry it out.

6.1 Additional Push Constraints

In a three processor Push operation, the movement of the inactive (*i.e.* not being Pushed) processors must be considered. An inactive processor may not be assigned an element outside its enclosing rectangle, or in a row and column which does not already contain elements of that processor. There exists six distinct ways this may occur for Processors P, R, and S. Recall that the volume of communication of any data partition shape q is given by,

$$\text{VoC} = \sum_{i=1}^{N} N(c_i - 1) + \sum_{j=1}^{N} N(c_j - 1) \tag{6.1}$$

$c_i -$ # of processors assigned elements in row i of q
$c_j -$ # of processors assigned elements in column j of q

For clarity, these sections will refer to removing a processor entirely from some row or column, leaving the processor assigned to no elements in that row or column, as *clean*ing that row or column of that processor. The addition of a processor to a row or column in which it did not previously own an element is referred to as *dirty*ing that row or column with that processor.

These descriptions assume the chosen input to the Push DFA is a Push Down ($\downarrow$) on Processor R, but are similar for other directions and Push operations on Processor S.

Type One - Decreases Volume of Communication

For each element assigned to Processor R in r_{top}, Processor R is assigned an element in the below rows and columns already containing elements of Processor R.

For each element which has been reassigned to R, the Processor previously assigned that element is given some unassigned element (r_{top}, j). Prior to the Push, this Processor must have already had an element in row r_{top} and in column j.

Type Two - Decreases Volume of Communication

For each element assigned to Processor R in r_{top}, Processor R is assigned an element in the rows below. Elements may go to some number, l, of rows and columns which did not already contain elements of Processor R, *dirty*ing those rows and columns, if l or more rows and columns are also *clean*ed of R.

For each element which has been reassigned to R, the Processor previously assigned that element is given some unassigned element (r_{top}, j). Prior to the Push, this processor must already have had an element in row r_{top} and in column j.

Type Three - Decreases Volume of Communication

For each element assigned to Processor R in r_{top}, Processor R is assigned an element in the rows and columns below that already contain elements of Processor R.

For each element which has been reassigned to R, the Processor previously assigned that element is given some unassigned element (r_{top}, j). Prior to the Push, it is not necessary for this Processor to have had an element in r_{top} or j, provided the number of rows and columns dirtied, l, is less than the number of rows and columns cleaned.

Type Four - Decreases Volume of Communication

For each element assigned to Processor R in r_{top}, Processor R is assigned an element in the rows below. Elements may go to some number of rows and columns, l, which did not already contain elements of Processor R, dirtying

those rows and columns, if l or more rows and columns are also cleaned of R.

For each element which has been reassigned to R, the Processor previously assigned that element is given some unassigned element (r_{top}, j). Prior to the Push, it is not necessary that this Processor have already had an element in r_{top} or j, provided the number of rows and columns dirtied, l, is less than the number of rows and columns cleaned.

Type Five - Unchanged Volume of Communication

For each element assigned to Processor R in r_{top}, Processor R is assigned an element in the rows below. A single row or column not containing elements of Processor R may be dirtied.

For each element which has been reassigned to R, the Processor previously assigned that element is given some unassigned element (r_{top}, j). Prior to the Push, this Processor must have been assigned an element in row r_{top} and in column j.

Type Six- Unchanged/Decrease Volume of Communication

For each element assigned to Processor R in r_{top}, Processor R is assigned an element in the rows below. A single row or column not containing elements of Processor R may be dirtied.

For each element which has been reassigned to R, the Processor previously assigned that element is given some unassigned element (r_{top}, j). Prior to the Push, it is not necessary that this Processor have had an element in r_{top} or j, provided the number of rows and columns dirtied, l, is less than or equal to the number of rows and columns cleaned.

6.2 Implementing the Push DFA

In order to show that the three processor Push always converges on some small set of candidates, the DFA was implemented as a software tool. This section describes the design and implementation of the tool, as well as the practical outcomes of its use.

6.2.1 Motivation

Generally, when proving mathematical concepts, the grand idea is already fixed in mind, and equations are tools used to convince others of the veracity of the big idea. The more complex nature of the three processor Push does

not lend itself to a simple mathematical proof for the simple reason that the final result, the candidate shapes, are too numerous and varied to be easily guessed ahead of time. Despite much consideration, the number of permutations and possibilities for the direction and order of the Push operations performed, are too difficult to categorise effectively by hand. However, the output, the candidate shapes, must still be found! Moreover, these shapes are only considered the full set of candidate shapes if it can be shown that no other partition shapes are output. The DFA is designed to allow us to consider ourselves certain that all possible candidates have been found.

6.2.2 Algorithmic Description

As an example, this portion will discuss a Push Down ($\downarrow$) on active Processor R, but the other directions are similar.

Formally, Push Down $(\downarrow)\phi(R) = \phi_1$ where,

Initialise $\phi_1 \leftarrow \phi$
$(g, h) \leftarrow (r_{top} + 1, r_{left})$
for $j = r_{left} \rightarrow r_{right}$ **do**
  if $\phi(r_{top}, j) = 0$ **then**
    {Element is dirty, clean it}
    $(g, h) \leftarrow$ find (g, h) {Function defined below}
    if $\phi(g, h) = 1$ **then**
      $\phi_1(r_{top}, j) \leftarrow 1$ {Cleaned element assigned to S}
    end if
    if $\phi(g, h) = 2$ **then**
      $\phi_1(r_{top}, j) \leftarrow 2$ {Cleaned element assigned to P}
    end if
    $\phi_1(g, h) \leftarrow 0$ {Put displaced element in new spot}
  end if
  $j \leftarrow j + 1$
end for

The function find(g,h) searches for a suitable swap of elements according to defined Push Types. This is the algorithm for finding a Type One Push, the other Types are similar.

findTypeOne(g, h) {Look for a suitable slot to put element}
for $g \rightarrow r_{bottom}$ **do**
  for $h \rightarrow r_{right}$ **do**
    if $\phi_1(g, h) \neq 0$ && $(row(\phi, r_{top}, (\phi_1(g, h)) = 1 \parallel col(\phi, j, (\phi_1(g, h)) = 1)$ && $(row(\phi, g, R) = 1 \parallel col(\phi, h, R) = 1$ **then**
      return (g, h)

end if
$h \leftarrow h + 1$
end for
$h \leftarrow k_{left}$
$g \leftarrow g + 1$
end for
return $\phi_1 = \phi$ {No Type One Push$\downarrow$ $\phi(R)$ possible}

6.2.3 End Conditions

In order to implement the Push, there must exist strictly defined conditions under which a partition is considered fully Pushed. In the theoretical Push, a partition is considered fully Pushed, or condensed, when the elements of no processors, except the largest processor, may be legally moved in any Push direction.

The implementation of the DFA program first determines the valid directions of Push for a given processor in a given run. A partition is fully condensed if there are no available Push operations in those predetermined directions.

6.3 Experimental Results with the Push DFA

This section describes the use of the Push DFA to collect all possible candidate partition shapes for further analysis. The results of these experiments were previously published in [63].

6.3.1 Experimental Setup

The size of the matrix chosen must be large enough to possess the granularity of elements required to form a variety of shapes, and be considered representative of any value of N. However, the larger the matrix size N, the larger the set of possible states, Q, and therefore more experimental runs are necessary to appropriately cover them all. To balance these two requirements, $N = 1000$ was chosen.

The processor ratios chosen for study were 2:1:1, 3:1:1, 4:1:1, 5:1:1, 10:1:1, 10:5:1, 10:8:1 2:2:1, 3:2:1, 4:2:1, 5:2:1, 5:3:1, 5:4:1. For each ratio, the DFA implementation was run approximately 10,000 times. The DFA is not a simulation of actual matrix multiplication on parallel processors, but searching for partition shapes which cannot be improved using the Push operation, so it is designed to be run on a single processor. Multiple instances of the

program were run on multiple processors to increase the speed at which data was collected. The Push DFA was run on a small cluster of Dell Poweredge 750 machines with 3.4 Xeon processors, 256 MB to 1 GB of RAM, and 1 MB of L2 cache.

Thoroughness of the Push DFA

As stated in Equation 4.1, the number of possible states for the DFA to pass through is quite large. However, it is not necessary for the DFA to pass through every state. It is sufficient for the DFA to pass through a subset of states, way stations, which may be reached from all states via valid Push operations. More formally, a state, q, has been "considered" if it is:

- a q_0 for an experiment
- some state, q_x, which an experiment passes through
- any state with a path of legal Push transition arrows leading to either q_0 or q_x

This is shown in Figure 6.1.

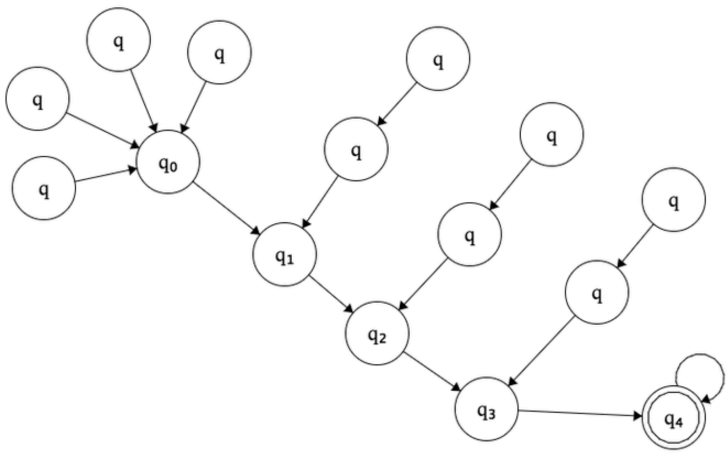


Figure 6.1: The Push DFA drawn as a state diagram. The program begins execution at q_0, passes through states using the transition arrows (Push operations) to reach a final accept state, q_4. Each state q which is not on this path, but which leads to this path via valid Push transitions, is also considered.

Randomising Push Direction

Part of the difficulty of searching for potentially optimal data partition shapes is ensuring all possible shapes are considered. Preconceived notions about what is likely to be optimal should not determine how the program searches for these potentially optimal shapes. For each new starting state, the DFA program selects a random number of directions (1, 2, 3 or 4) to Push the active processor. The Push directions are then randomly selected. For example, if 2 is selected as the number of directions, then Up and Left, or Down and Left, and so forth, might be selected as Push directions. Finally, the order of Push operations is randomly selected. In this way, quite disparate cases are accounted for, such as one Push direction only, two Push directions in which one direction is exhausted before the other begins, or four Push directions where each Push direction is interleaved with the others.

Randomising q_0

Before beginning the program, elements must be randomly dispersed (in correct proportion) amongst Processors P, R, and S. This is accomplished by the following procedure,

1. All elements are assigned to Processor P
2. Until the correct number of elements have been reassigned to R do:
    - Pick random integer value for row, i
    - Pick random integer value for column, j
    - If (i, j) is assigned to R, do nothing and pick again
    - If (i, j) is assigned to P, reassign to R
3. Until the correct number of elements have been reassigned to S do:
    - Pick random integer value for row, i
    - Pick random integer value for column, j
    - If (i, j) is assigned to R or S, do nothing and pick again
    - If (i, j) is assigned to P, reassign to S

6.3.2 Description of Shape Archetypes

The result of the experiments with the Push DFA was the creation of over one hundred thousand possible candidate partition shapes. These results

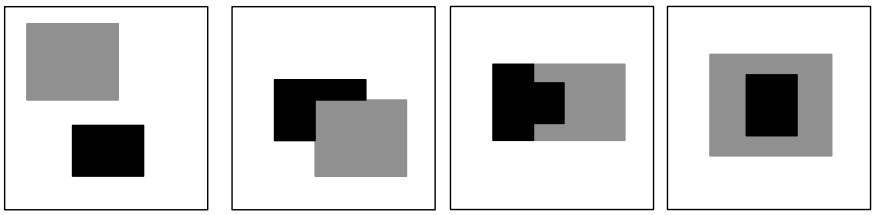


Figure 6.2: The general partition shape archetypes found experimentally by the Push DFA. Each shape archetype includes all possible shapes with the same characteristics of overlap of enclosing rectangles, and number of corners present.

were parsed and categorised according to two distinct metrics, the relative position of the enclosing rectangles, and the number of corners present in each processor's shape. This resulted in four broad categories, called Shape Archetypes, in which all shapes output by the DFA fit. These archetypes, called A, B, C, and D, are shown in Figure 6.2.

Archetype A - No Overlap, Minimum Corners

In Archetype A partitions, the enclosing rectangles of Processors R and S do *not* overlap. Processors R and S are each rectangular, possessing the minimum number of corners (four). Processor P is assigned the remainder of the matrix. Depending on the dimension and location of Processors R and S, the matrix remainder assigned to Processor P may be either rectangular or non-rectangular.

If Processor P is rectangular, the entire partition shape, q, is rectangular. Otherwise, q is a non-traditional, non-rectangular shape. It is important to note that although these two partition shape types seem disparate at first glance, they are similar in their description of enclosing rectangles and corners, and so are grouped together.

Archetype B - Overlap, L Shape

In Archetype B partitions, the enclosing rectangles of Processors R and S partially overlap. One processor, as shown in Figure 6.2, is rectangular, having four corners. The other processor has six corners, and is arranged in an "L" shape adjacent to the rectangle shape of the first Processor. Processor P is assigned the remainder of the matrix.

Archetype C - Overlap, Interlock

In Archetype C partitions, the enclosing rectangles of Processors R and S partially overlap. Neither processor has a rectangular shape. Each processor has a minimum of six corners. Processor P is assigned the remainder of the matrix, which may be rectangular or non-rectangular.

We note that in all experimentally found examples of Archetype C, if the shapes of Processors R and S were viewed as one processor, they would be rectangular.

Archetype D - Overlap, Surround

In Archetype D partitions, the enclosing rectangle of one processor, shown in Figure 6.2 as Processor S, is entirely overlapped, or surrounded, by Processor R's enclosing rectangle. Processor S has four corners, while Processor R has eight corners. Processor P is assigned the remainder of the matrix, which may be rectangular or non-rectangular.

6.3.3 Reducing All Other Archetypes to Archetype A

This section proves that all Archetypes may be reduced to Archetype A without increasing the volume of communication of any shapes within that Archetype. To do this, first, the idea of *corners* previously alluded to will be formally defined.

Defining Corners. A single processor, in partition shape q, has a corner at some point (x, y) if both:

- the constant coordinate (x or y), along that edge, changes after (x, y)
- the variable coordinate (x or y), along that edge, becomes constant after (x, y)

Each shape has four edges to consider, even if parts of each edge lie on different rows or columns. Consider the example illustrated in Figure 6.3. The sides of the matrix, beginning at the bottom and moving clockwise are named x, y, z, and w.

In this example, the location of each corner A through J is given in coordinate form by,

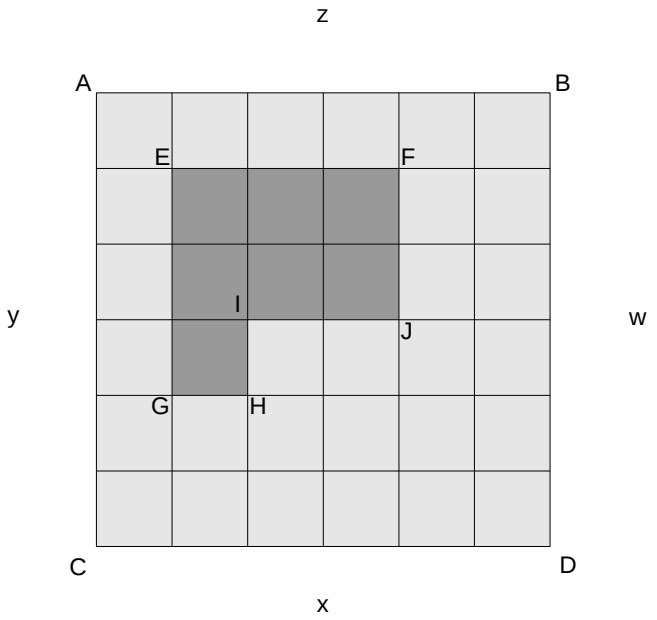


Figure 6.3: Introduction to corner terminology on a simple 6×6 grid between two processors. The various corners are labelled A through J, and edges labelled w, x, y and z.

$$\begin{array}{llll} A = (0,0) & B = (6,0) & C = (0,6) & D = (6,6) \\ E = (1,1) & F = (4,1) & G = (1,4) & H = (2,4) \\ & I = (2,3) & J = (4,3) & \end{array}$$

In this example, consider the Processors to be P and S, with P in light grey, and S in dark grey. The lines composing the edges, (x, y, z, w), are then given by,

$$\begin{array}{llll} P_z = AB & P_w = BD & P_x = CD & P_y = AC \\ S_z = EF & S_w = FJ + IH & S_x = GH + IJ & S_y = EG \end{array}$$

In this example, a corner must exist (on the right (w) or bottom (x) edge, depending on your viewpoint. Specifically, Corner I exists because $H \neq J$.

Now this will be applied to a more general notation for all processors, rather than continue to use arbitrary letters to denote corners.

Each processor, P, R and S, has a minimum of four corners. Each edge is denoted using the notation P_{x1}, P_{x2}, P_{y1}, P_{y2} and so on. This is shown in Figure 6.4. When a shape has the minimum number of corners, then each corner may be referred to by either of the two notations, *i.e.* $P_{y1} = P_{z1}, P_{z2} = P_{w1}, P_{w2} = P_{x2}$ and $P_{x1} = P_{y2}$. Note that for vertical edges, y and w, points are given top to bottom, and for horizontal edges, x and z, are given left to right.

If, for a given processor, the edge points are not equal to their corresponding adjacent edge points, then at least one extra corner must exist along those two edges.

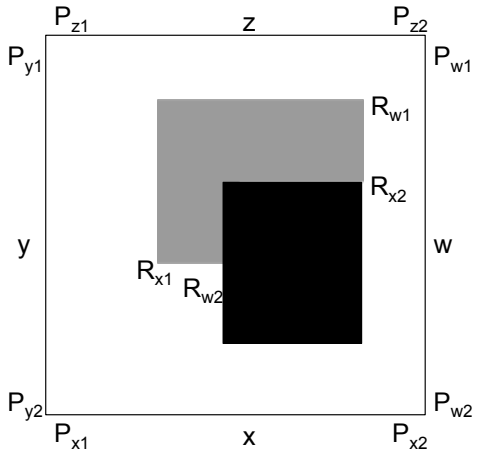


Figure 6.4: An Archetype B partition shape shown with the corner notation for Processor R (in grey). Not all points are labeled, but all points follow the pattern shown by points labeled for Processor P. Notice that $R_{w2} \neq R_{x2}$, so at least one corner must exist.

Location of Processors R and S. If the enclosing rectangles of Processors R and S are moved in a "drag and drop" fashion, communication time will not be increased, which will allow equivalency between different partition shapes.

Theorem 6.3.1. *In a partition among three heterogeneous processors, the position of the two smaller processor shapes, within the context of the larger matrix, does not affect the total volume of communication, if the position of the two smaller shapes do not change relative to each other.*

Proof. Consider the shapes of Processor R and S to be one continuous shape. Their position relative to each other will not change, so moving this combined

shape is analogous to moving a single small processor in a two processor data partition. This is known not to increase the volume of communication [57]. □

Reducing Archetype B to Archetype A

Theorem 6.3.2. *Any Archetype B partition shape, q, may be transformed into an Archetype A partition shape, q_1, without increasing the volume of communication of the shape.*

Proof. Within Archetype B, there exist two possible, distinct layouts of data. These are,

- combined width or height of the smaller processors is equal to N
- combined width and height of the smaller processors is less than N

These two possibilities are shown in Figure 6.5.

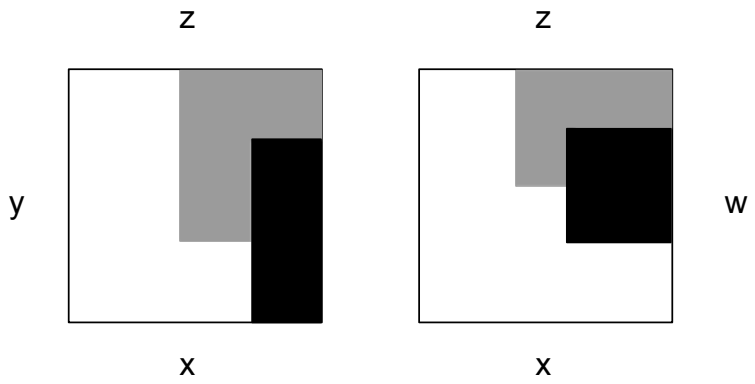


Figure 6.5: The two possible cases of an Archetype B partition shape. On the left, the combined length of the two shapes is N, the full length of the matrix. On the right, the combined length of the two shapes is less than N.

For both cases of Archetype B partitions, a Push-like transformation can be applied to Processor R, the "L" shape, along one of the planes with the extra corner. In Figure 6.5 this is either the x or w sides, so the elements of Processor R may be moved in either the Back ($\leftarrow$) or Up ($\uparrow$) directions. This is not strictly a Push operation, as the enclosing rectangle for R will be expanded in one direction. However, because the enclosing rectangle is also being diminished in another direction, the volume of communication does not increase.

In the first case, to move the elements of R, only one transformation direction is available because the length N of the combined rectangles does not allow room for additional swaps. In the example of Figure 6.5 the only available direction is Back ($\leftarrow$).

For each column transformed to remove elements of R, at most one column previously not containing R will have elements of R introduced. This is assured, by definition, by virtue of the existence of the corner:

$$R_{w1} \rightarrow R_{x2} < R_{y1} \rightarrow R_{y2} \tag{6.2}$$

where,

$$R_{w1} \rightarrow R_{x2} = \# \text{ of rows separating } R_{w1} \text{ and } R_{x2}$$
$$R_{y1} \rightarrow R_{y2} = \# \text{ of rows separating } R_{y1} \text{ and } R_{y2}$$

For the second case, the elements of R can be moved in either direction, as the combined length of both shapes is less than N, and therefore rows and columns exist in either direction into which elements of R can be moved. The direction of the Push-like transformation is decided by choosing that which requires the lower volume of elements to be moved. In example Figure 6.5, first use Theorem 6.3.1 to move the entire shape of Processors R and S down in the matrix so that $S_{x2} = P_{x2}$, opening rows above Processor R so elements may be moved in the Up ($\uparrow$) direction.

For each row transformed to remove elements of R, at most one row previously not containing R will have elements of R introduced. This is assured by definition, by virtue of the existence of the corner:

$$R_{x1} \rightarrow R_{w2} < R_{z1} \rightarrow R_{z2} \tag{6.3}$$

where,

$$R_{x1} \rightarrow R_{w2} = \# \text{ of rows separating } R_{x1} \text{ and } R_{w2}$$
$$R_{z1} \rightarrow R_{z2} = \# \text{ of rows separating } R_{z1} \text{ and } R_{z2}$$

For every row or column made dirty with R during these transformations, a row or column must have, by definition been made clean of R, so volume of communication is constant or decreasing. □

Archetype C to Archetype A

Theorem 6.3.3. *Any Archetype C partition shape, q, may be transformed into a Archetype A partition shape, q_1, without increasing the volume of communication of the shape by applying the Push operation.*

Proof. By definition of this shape, valid Push operations remain, which if applied will result in an Archetype A partition. □

Archetype C is the only archetype formed by the DFA program on which there are valid Push operations remaining. These form as a result of the randomised Push direction algorithm, and is a necessary downside to truly considering every possible partition shape without preconceived notions of the final shape. Transforming partition shapes of this archetype is a simple matter of applying the Push operation in the direction not selected by the program.

In the program, these cases are handled by a "beautify" function to return rectangular or asymptotically rectangular shapes, however Archetype C is included here for comprehensiveness.

Archetype D to Archetype A

Theorem 6.3.4. *Any Archetype D partition shape, q, may be transformed into an Archetype A partition shape, q_1, without increasing the volume of communication of the shape*

Proof. In [57] it was proven that for two processors, the location of the smaller processor within the context of the larger matrix, does not effect the total volume of communication.

Consider the surrounding processor, in figures Processor R, and the inner processor, Processor S, to be a two processor partition in a matrix the size of Processor R's enclosing rectangle.

By [57] Theorem 3.4 Canonical Forms, move Processor S so that $R_{x2} = S_{x2}$.

An Archetype B partition has now been created from an Archetype D, without increasing its volume of communication. By Theorem 6.3.2, it may be further reduced to Archetype A. □

6.4 Push Technique on Four or More Processors

The additional conditions introduced for Push on three processors are necessary and sufficient for using the Push Technique on four or more processors. Each inactive processor, including any arbitrarily large numbers of processors, must not,

- have its enclosing rectangle enlarged

- make *dirty* a number of rows and columns, without *clean*ing an greater or equal amount of rows and columns

However, as the number of processors increases, so does the likelihood that some arbitrary shape will be found that cannot be improved upon by the Push Technique. If the Push Technique were to regularly run into arbitrary arrangements of elements which are not true candidates, but which are dead ends beyond which is cannot progress, would significantly hamper its usefulness.

To accommodate this problem, a larger matrix size could be used to increase the granularity of the experimental runs. In this way, it is foreseeable that the Push Technique can be experimentally expanded to six, eight, or even ten abstract processors. And as an abstract processor can actually represent myriad computing resources, if the Push Technique is used at varying, hierarchical levels, there should be no limit to the number of processors it can be applied to. While this is outside the scope of this thesis, it does present exciting possibilities which are discussed further in Chapter 10.

Chapter 7

Three Processor Optimal Partition Shape

The next step in using the Push Technique is to find the optimal data partition for three processors, building on the work done with two processors. The Push DFA provides motivation for this postulate:

Postulate 1 (Three Processor Push)**.** *There exists no arrangement of elements among three heterogeneous processors in an $N \times N$ matrix which cannot be improved with the Push operation, except those arrangements of shapes defined as Archetypes A, B, C, and D.*

As it was shown in Chapter 4, the optimal shapes must lie within Archetype A, as all other Archetypes may be reduced to it without increasing communication volume. This chapter will evaluate all the possible shapes within the Archetype A to determine the optimal partition shape for all computational power ratios, four different network topologies, and the five matrix multiplication algorithms.

7.1 Archetype A: Candidate Shapes

A wide variety of partition shapes exist within Archetype A. All of these shapes may be further categorised into six partition shape types. The candidate partition types included under Archetype A, found experimentally by the Push DFA, are seen in Figure 7.1. In all six, Processors R and S are assigned rectangular portions of the matrix to compute. These rectangles vary in length from, at the longest, N to, at the shortest, $\sqrt{\#X}$, *i.e.* the side of a square containing all the elements of Processor X. Processor P is assigned a rectangular portion in three of the shape types, and non-rectangular portion in the remaining three types.

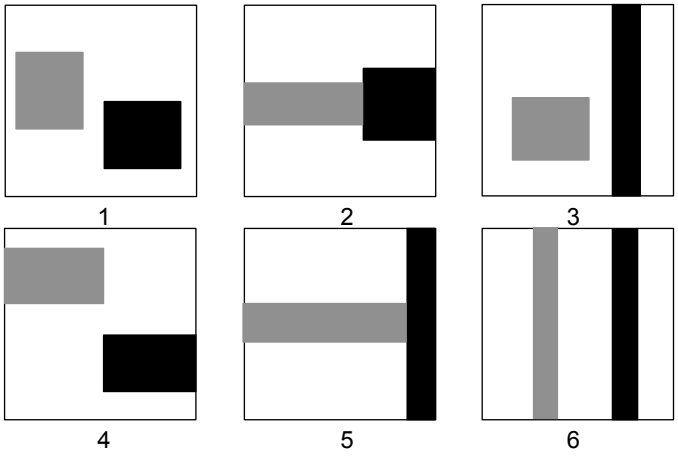


Figure 7.1: The six candidate partition types found under Archetype A. Confirmed experimentally using the Push DFA.

Each shape type is representative of all shapes matching the same general description (in terms of enclosing rectangles and corners). The location within the matrix for each rectangle assigned to Processors R and S may be different than shown, and is a factor to consider when determining the *canonical* form of each partition shape type.

In the following sections each of the partition shape types is formally defined to describe which parts are fixed and which may be changed to create a valid partition shape of the same type. However, it is important to note that it is *not* true to assert that all valid partition shapes of the same type are necessarily equivalent. Indeed, the next section defines the canonical, "best version", of each candidate type.

7.1.1 Formal Definition of Candidate Shapes

Listed here are the fixed points in the definition of each type. If a dimension or a relative location is left unspecified, that dimension may have any value from zero to the size of the matrix, N. A shape is considered rectangular if all processors are assigned a single rectangular portion of the matrix to compute. Shapes in which a single processor is assigned two or more rectangles to compute are non-rectangular.

Processors R and S will be referred to as having the dimensions R_{width}, R_{length} and S_{width}, S_{length} respectively. The value of R_{width} is derived from the distance between points R_{x1} and R_{x2} described in Section 6.3.3. The value of R_{length} is the distance between points R_{y1} and R_{y2}. The values S_{width} and S_{length} are derived in the same manner.

A partition shape falls under the given type if it fulfils the listed criteria or can be rotated to meet the criteria. For all types R and S are rectangular so, $R_{width} \times R_{length} = \#R$ and $S_{width} \times S_{length} = \#S$.

Type One

$R_{width} + S_{width} < N$
$R_{length} < N$
$S_{length} < N$

Type Two

$R_{width} + S_{width} = N$
$R_{length} < N$
$S_{length} < N$
$R_{length} \neq S_{length}$

Type Three

$R_{width} + S_{width} < 1$
$R_{length} < N$
$S_{length} = N$

Type Four

$R_{width} + S_{width} = N$
$R_{length} < N$
$S_{length} < N$
$R_{length} = S_{length}$

Type Five

$R_{width} + S_{width} = N$
$R_{length} < N$
$S_{length} = N$

Type Six

$R_{width} + S_{width} < N$
$R_{length} = N$
$S_{length} = N$

7.1.2 Finding the Canonical Version of each Shape

Each of the candidate partition types has some leeway for difference either in the dimensions of rectangles R and S or in their location within the matrix. The best, canonical version of each shape type will be one which minimises communication time, within the given constraints of that shape. To accomplish this, the combined perimeters of rectangles assigned to Processors R and S is minimised, as the sum of half perimeters is a metric of volume of communication [28, 52, 54].

For the following proofs the size of the matrix is normalised to $N = 1$, and the total computational power of the three processor system is defined by,

$$T = P_r + R_r + S_r \tag{7.1}$$

Splitting Type One (Square Corner and Rectangle Corner)

The Type One candidate partition shape is composed of two rectangles, each of width and length less than 1. The minimum perimeter of a rectangle of fixed area occurs when width and height are equal, *i.e.* when the rectangle is a square. However, it may not always be possible to form two non-overlapping squares in an 1×1 matrix, even if the locations are fixed such that: $R_{y1} = P_{y1}$ and $S_{x2} = P_{x2}$ (*i.e.* when the two squares are pulled to opposite corners).

Theorem 7.1.1. *The rectangles formed by Processors R and S may both be squares when $P_r > 2\sqrt{R_r}$.*

Proof. The volume of elements assigned to each processor is equal to the Processors' ratio divided by the sum of the ratios, and multiplied by the total volume of elements in the matrix. The volume of elements assigned to each of the Processors, P, R and S, are $\frac{P_r}{T}$, $\frac{R_r}{T}$, and $\frac{1}{T}$, respectively. Assuming that both Processors R and S are squares, then the length of their sides will be $\sqrt{\frac{R_r}{T}}$ and $\sqrt{\frac{1}{T}}$ respectively. In order for the squares to fit in the 1×1 matrix without overlapping,

$$\begin{aligned} \sqrt{\frac{R_r}{T}} + \sqrt{\frac{S_r}{T}} &< 1 \\ R_r + 2\sqrt{R_r S_r} + S_r &< T \\ 2\sqrt{R_r} &< P_r \end{aligned}$$

□

For those ratios where $P_r < 2\sqrt{R_r}$, and two squares may not be formed, the optimal shape which still conforms to the Type One criteria must be found. This requires minimising the following function,

$$f(x,y) = 2\Big(\frac{R_r}{Tx} + x + \frac{S_r}{Ty} + y\Big) \tag{7.2}$$

under the constraints,

$$\begin{aligned} 0 < \frac{R_r}{xT} < 1 \\ 0 < \frac{S_r}{yT} < 1 \\ x + y < 1 \end{aligned}$$

The slope of the surface bounded by these constraints is increasing with x and y. Indeed the derivative of Equation 7.2 is positive, indicating it is increasing. To find the minimum of (7.2) then, it is necessary to search along the lower bound, where $x + y \approx 1$.

Knowing the solution lies along this bound, Equation (7.2) may be rewritten as a function of x, and its derivative set equal to zero to solve for x. This gives $x = -\frac{\sqrt{R}-R}{R-1}$. Therefore, for ratios such that $P_r < 2\sqrt{R_r}$, the optimal shape is two non-square rectangles R and S of combined width of approximately 1, but less than 1 by definition. The two optimal versions of Type One partitions can be seen in Figure 7.2.

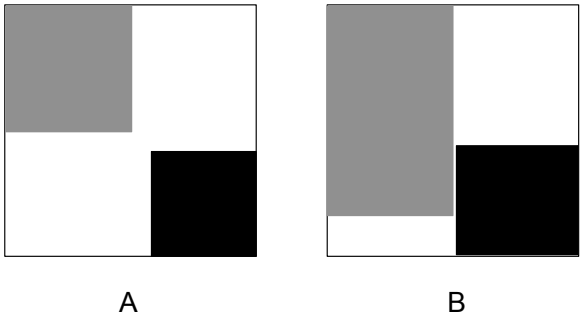


Figure 7.2: On the left is a Type 1A partition shape, the Square-Corner, with Processors R and S each formed into a square. On the right is a Type 1B partition shape, the Rectangle-Corner, showing two non-square rectangles.

Combining Type Two and Four (Block Rectangle)

Partition shape Types Two and Four are similar, but Type Four is more rigid. In a Type Four partition, both dimensions of both rectangles R and S

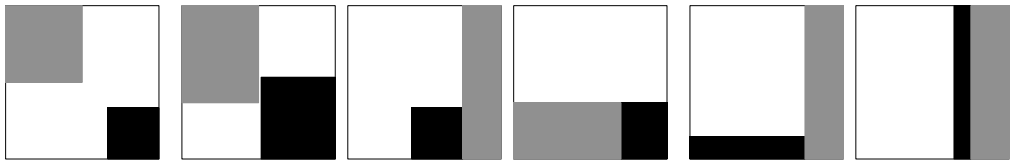


Figure 7.3: The candidate partition shapes identified as potentially optimal three processor shapes. Processors P, R, and S are in white, grey, and black, respectively. (1) Square Corner (2) Rectangle Corner (3) Square Rectangle (4) Block 2D Rectangular (5) L Rectangular (6) Traditional 1D Rectangular

are fixed, and only their relative location in the matrix may be altered. In Type Two, the total width of R and S is fixed, but the relative dimensions may change. A Type Two partition is improved, lowering the volume of communication, by transforming it into a Type Four partition by changing the relative widths so that $R_{height} = S_{height}$. The canonical form of the Type Four partition, the Block Rectangle, is shown in Figure 7.3, with $R_{y1} = P_{y2}$ and $S_{x1} = R_{x2}$.

As the Type Two is now obsolete, the Types 1A and 1B may be called Types 1 and 2, respectively.

Type Three (Square Rectangle) Canonical Form

The Type Three partition has a rectangle of height N, and therefore of fixed width. The second rectangle is unfixed in both dimensions, and as previously asserted, the optimal shape for a rectangle on length and width less than N is a square. It is possible to form a square and a rectangle for all ratios $P_r : R_r : 1$. The canonical form of the Type Three partition, the Square Rectangle, is shown in Figure 7.3, with $R_{x1} = S_{x2}$ and $S_{w1} = P_{x2}$.

Type Five (L-Rectangle) and Type Six (1D Rectangle) Canonical Form

In both Type Five and Type Six partitions, the height and width of both processors R and S are fixed, and only their relative position within the matrix may be changed. For Type Five, the L-Rectangle partition, the coordinates $S_{y1} = P_{y2}$ and $S_{x2} = R_{x1}$ are fixed. In Type Six partitions, the 1D Rectangle is set so $P_{x2} = S_{x1}$ and $S_{x2} = R_{x1}$, as seen in Figure 7.3.

7.2 Network Topology Considerations

There are two distinct network topologies in which three processors may be arranged, the fully connected ring or the star topology. In the fully connected topology each of the three processors has a direct communication link to all other processors, as seen in Figure 7.4.

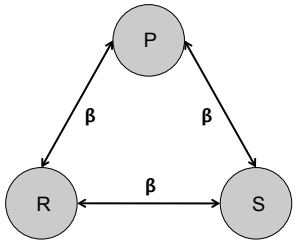


Figure 7.4: The Fully Connected Topology for three processors P, R, and S.

The star topology involves a single processor at the centre, with the other two processors with a direct communication link only to that processor. However, there are three distinct variations of the star topology, depending on which processor, P, R, or S, is the centre processor. These three variations of the star topology are seen in Figure 7.5.

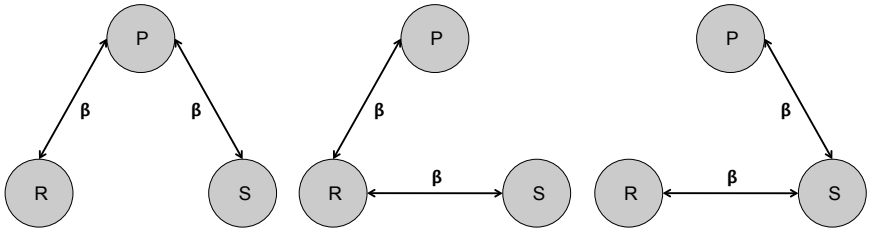


Figure 7.5: The Star Topology for three processors P, R, and S. In each variation, a different processor is at the centre of the star, giving different communication characteristics in the model.

As the fully connected topology has links between all processors, the parallel communication algorithm is straightforward. However, for the star topology it is worth discussing how this will be modelled. For example, consider star topology variant one, with P at the centre of the star. First consider all data flowing left to right, the volumes of which are called $R \to P$ and $P \to S$, for the volume of data R must send to P, and P must send to S, respectively. Any data, $R \to S$, that R must send to S, will be sent after $R \to P$, and only forwarded on by P after $P \to S$ is sent.

$$\begin{array}{l|l} (R \to P) + (R \to S) & \\ (P \to S) & +(R \to S) \end{array}$$

Processor P can only forward the message of size $(R \to S)$ to Processor S after it has been both received by P, and P has finished sending its own data to S. Similarly, data flows in parallel in the opposite direction such that,

$$\begin{array}{l|l} (S \to P) + (S \to R) & \\ (P \to R) & +(S \to R) \end{array}$$

This can be written as a function of maximums such that,

$$T_{comm} = \beta \max\Big(\max\big((R \to P) + (R \to S), (P \to S)\big) + (R \to S), \\ \max\big((S \to P) + (S \to R), (P \to R)\big) + (S \to R)\Big)$$

The other star topology variants are modelled in similar fashion.

7.3 Fully Connected Network Topology

The optimal partition shape must be among one of the six candidate partitions. First, this section will focus on eliminating those shapes which are never the optimal under any circumstances. Then the remaining candidates (Square Corner, Square Rectangle, and Block Rectangle) are evaluated to determine for what system characteristics each is the optimal three processor fully connected shape.

7.3.1 Pruning the Optimal Candidates

It is possible to eliminate three candidate shapes from contention for the optimal shape.

Theorem 7.3.1 (Three Candidates)**.** *The three partition shapes known as Rectangle Corner, L Rectangle and Traditional Rectangle, have a higher theoretical volume of communication than the Block Rectangle shape. The optimal shape must be among the remaining three candidate shapes, Block Rectangle, Square Rectangle and Square Corner.*

The following subsections will prove that Rectangle Corner, L Rectangle, and Traditional Rectangle each have a higher volume of communication than Block Rectangle.

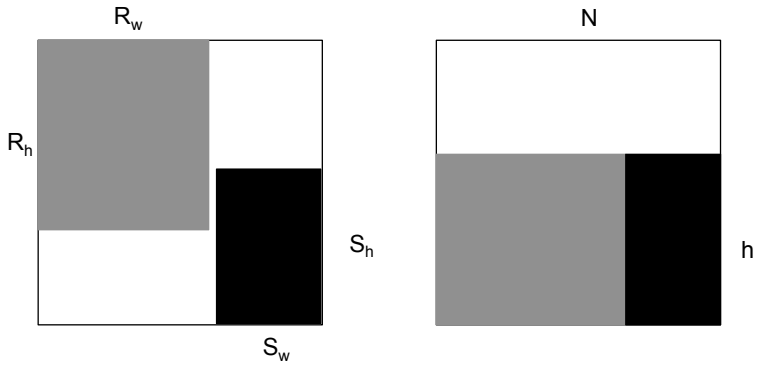


Figure 7.6: The Rectangle Corner, left, and the Block Rectangle, right, partition shapes shown in canonical form for processor ratio $2:2:1$.

Discarding Rectangle Corner

The Rectangle Corner shape is formed when the volume of elements assigned to Processors R and S is too large to form a Square Corner partition shape. In this case, when $P_r < 2\sqrt{R_r}$, the optimal size of these non-square rectangles is a combined width of N [63]. However, as a derivative of the Type One shape, the Rectangle Corner must be composed of two rectangles of width and height less than N as shown in Figure 7.6. By definition the Rectangle Corner has dimensions such that,

$$\begin{gathered} R_w < N \qquad R_h < N \qquad S_w < N \qquad S_h < N \\ R_w + S_w < N \qquad R_h + S_h > N \end{gathered}$$

As previously stated, the optimal size of R and S would be combined width N, which is not legal. Alternatively, set $R_w + S_w = (N-1)$, and with this system of equations create the following theorem,

Theorem 7.3.2 (Rectangle Corner)**.** *The Rectangle Corner partition shape has a larger volume of communication than the Block Rectangle partition shape for all ratios of computational power.*

Proof. Given the taxonomy in Figure 7.6, the volume of communication for Rectangle Corner and Block Rectangle shapes are given by,

$$V_{RC} = N(R_w + S_w) + N(R_h + S_h) \tag{7.3}$$
$$V_{BR} = N^2 + Nh \tag{7.4}$$

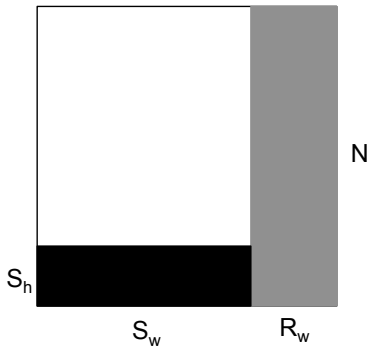


Figure 7.7: The canonical form of the L Rectangular partition shape with processor ratio $4:2:1$.

Substitute $R_w + S_w = (N-1)$, and create the inequality given by,

$$\begin{aligned} BR &< RC \\ N + h &< N - 1 + R_h + S_h \\ h + 1 &< R_h + S_h \\ h &\leq R_h + S_h \end{aligned}$$

By definition, $R_h + S_h > N$ and $h < N$, so this is true. □

Discarding L Rectangle

The L Rectangle partition shape, composed of an N height rectangle for Processor R and a rectangle for Processor S such that the combined width of S and R is N, can be seen in Figure 7.7. The width, S_w, of Processor S is given in terms of the known fixed width of Processor R.

$$\begin{aligned} V_{LR} &= N^2 + NS_w \\ S_w &= N - \frac{R_r N}{T} \end{aligned} \tag{7.5}$$

Theorem 7.3.3 (L Rectangle)**.** *The L Rectangle partition shape has a volume of communication larger than or equal to the Block Rectangle partition shape for all processor ratios.*

Proof. Note that h can be expressed as a function of the number of elements assigned to Processor P such that, $h = N - \frac{P_r N}{T}$. Begin by stating the

inequality,

$$\begin{aligned} BR &\leq LR \\ N^2 + Nh &\leq N^2 + NS_w \\ N\Big(N - \frac{P_r N}{T}\Big) &\leq N\Big(N - \frac{R_r N}{T}\Big) \\ -\frac{P_r}{T} &\leq -\frac{R_r}{T} \\ R_r &\leq P_r \end{aligned}$$

That $P_r \geq R_r$ is a problem constraint, so this must always be true. The L Rectangle has greater or equal volume of communication than the Block Rectangle for all valid processor ratios. □

Discarding Traditional Rectangle

Theorem 7.3.4 (Traditional 1D Rectangle)**.** *The Traditional 1D Rectangle partition shape has a larger volume of communication than the Block Rectangle partition shape for all processor ratios.*

Proof. The volume of communication for the Traditional 1D Rectangle shape is given by,

$$V_{TR} = 2N^2 \tag{7.6}$$

Setting the inequality with Equation 7.4 gives,

$$\begin{aligned} BR &< TR \\ N^2 + Nh &< 2N^2 \\ N + h &< 2N \\ h &< N \end{aligned}$$

By definition, $h < N$, so this is true. □

7.3.2 Optimal Three Processor FC Data Partition

The optimal data partition shape for three fully connected processors is always one of three shapes, the Square Corner, the Square Rectangle, or the Block Rectangle. Of these three, the first two are non-rectangular, but non-rectangular in different ways which make each suited to a different type of heterogeneous distribution of computational power.

In general these results show that,

- Square Corner is the optimal shape for heterogeneous systems with a single fast processor, and two relatively slow processors
- Square Rectangle is the optimal shape for heterogeneous systems with two fast processors, and a single relatively slow processor
- Block Rectangle is the optimal shape for heterogeneous systems with a fast, medium, and slow processor, as well as relatively homogeneous systems

A full summary of the optimality of each shape for all algorithms is given in Table 7.1.

These results are to be published in [58].

Serial Communication with Barrier

Applying the abstract processor models for the SCB algorithm to each of the three shapes - Square Corner (SC), Square Rectangle (SR), and Block Rectangle (BR) - gives,

$$T_{comm(SC)} = \left(2N\sqrt{\frac{R_r N^2}{T}} + 2N\sqrt{\frac{N^2}{T}}\right)\beta \tag{7.7}$$

$$T_{comm(SR)} = \left(N^2 + 2N\sqrt{\frac{N^2}{T}}\right)\beta \tag{7.8}$$

$$T_{comm(BR)} = \left(2N^2 - \frac{P_r N^2}{T}\right)\beta \tag{7.9}$$

The minimum of these three functions is the optimum shape. The three-dimensional graph of these shapes is shown in Figure 7.8

SCB Optimal Shape Proofs. This section contains the theorems and proofs which state for what system characteristics each candidate shape is optimal.

Theorem 7.3.5 (SCB Square Corner)**.** *For matrix multiplication on three heterogeneous processors, the Square Corner partition shape minimises execution time, i.e. is the optimum, using the SCB algorithm for all processor computational power ratios such that* $P_r < 2T - 2\sqrt{R_r T} - 2\sqrt{T}$.

Optimal Shape

Algorithm	Ratio	Shape
SCB	$P_r < 2T - 2\sqrt{R_r T} - 2\sqrt{T}$	Square Corner
	$P_r < T - 2\sqrt{T}$	Square Rectangle
	Remaining values of P_r	Block Rectangle
PCB	$P_r > 2(\sqrt{R_r T} - R_r + \sqrt{T} - 1)$	Square Corner
	$P_r < 2R_r + \frac{R_r}{\sqrt{T}} - 2\sqrt{T} - 1$ and $P_r > 5 + \frac{R_r - 2}{\sqrt{T}}$	Square Rectangle
	Remaining values of P_r	Block Rectangle
SCO	$P_r > \frac{\frac{2}{N}(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}) + \frac{2}{c}(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{2}{N}}{\frac{1}{Tc} - \frac{1}{TN}}$ and $P_r > \frac{2c}{N}(\sqrt{R_r T} + \sqrt{T}) + 2T(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{Tc}{N} - \frac{2c}{N}\sqrt{T}$	Square Corner
	$P_r < T - 2\sqrt{T}$	Square Rectangle
	Remaining values of P_r	Block Rectangle
PCO	Remaining values of P_r	Square Corner
	$P_r < \frac{1 + \frac{2}{\sqrt{T}} - \frac{R_r}{T} - \frac{R_r}{T\sqrt{T}} - \frac{3}{T} - 2r^2}{N(\frac{r^2}{cR_r} - \frac{1}{Tc})}$	Square Rectangle
PIO	Remaining values of P_r	Square Corner
	$P_r < 4\sqrt{T}$	Block Rectangle

Table 7.1: Summary of optimal shape results for three fully connected heterogeneous processors.

Proof. The graph of the Square Corner T_{comm} function shows the surface

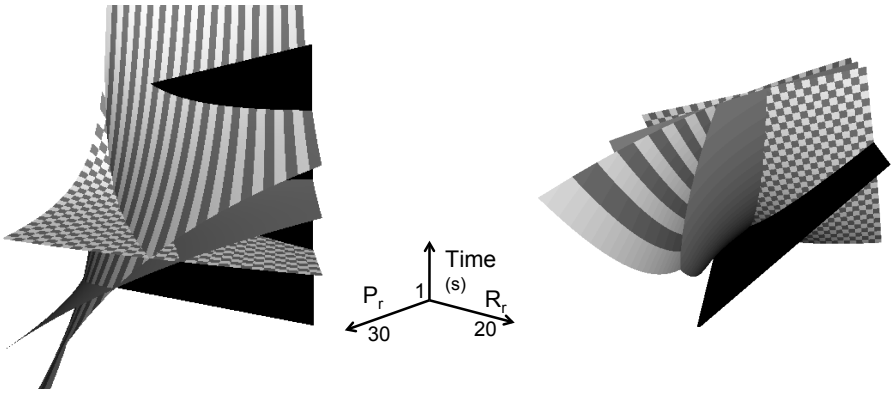


Figure 7.8: The SCB T_{comm} functions for the three candidate shapes, Square Corner (white and grey stripes), Block Rectangle (solid grey), and Square Rectangle (white and grey checkerboard). The x-axis is the relative computational power of P, P_r, from 1 to 30. The y-axis is the relative computational power of R, R_r, from 1 to 20. The z-axis is the communication time in seconds. The vertical black surface is the equation $x = y$, and represents the problem constraint $P_r \geq R_r$. On the left, viewed from the front, on the right, rotated so as to be viewed from underneath (the lowest function is optimal).

intersects with the Block Rectangle surface.

$$
\begin{aligned}
SC &< BR \\
2N(r+s) &< N^2 + Nh \\
2\left(\sqrt{\frac{R_r N^2}{T}} + \sqrt{\frac{N^2}{T}}\right) &< N + \left(N - \frac{P_r N}{T}\right) \\
2\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right) &< 2 - \frac{P_r}{T} \\
2\sqrt{R_r T} + 2\sqrt{T} &< 2T - P_r \\
P_r &< 2T - 2\sqrt{R_r T} - 2\sqrt{T}
\end{aligned}
$$

□

Theorem 7.3.6 (SCB Square Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Square Rectangle partition shape minimises execution time, i.e. is the optimum, using the SCB algorithm for all processor computational power ratios such that* $P_r < T - 2\sqrt{T}$.

Proof.

$$\begin{aligned} SR &< BR \\ N^2 + 2Ns &< N^2 + Nh \\ N + 2\left(\sqrt{\frac{N^2}{T}}\right) &< N + \left(N - \frac{P_r N}{T}\right) \\ 1 + 2\sqrt{\frac{1}{T}} &< 2 - \frac{P_r}{T} \\ P_r &< T - 2\sqrt{T} \end{aligned}$$

□

Corollary 7.3.7 (SCB Block Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Block Rectangle partition shape minimises execution time, i.e. is the optimum, for all processor computational power ratios except those specified in Theorems 7.3.5 and 7.3.6.*

Parallel Communication with Barrier

First, the abstract processor model for the Parallel Communication with Barrier algorithm is applied to the three candidate shapes - Square Corner (SC), Square Rectangle (SR), and Block Rectangle (BR) - to find the communication time for each shape.

$$T_{comm(SC)} = 2N^2\beta * \max\left(\sqrt{\frac{R_r}{T}} - \frac{R_r}{T} + \sqrt{\frac{1}{T}} - \frac{1}{T}, \frac{R_r}{T}, \frac{1}{T}\right) \tag{7.10}$$

$$T_{comm(SR)} = N^2\beta * \max\left(1 + \frac{2}{\sqrt{T}} - \frac{R_r}{T} - \frac{R_r}{T\sqrt{T}} - \frac{3}{T}, \frac{R_r}{T} + \frac{R_r}{T\sqrt{T}}, \frac{3}{T}\right) \tag{7.11}$$

$$T_{comm(BR)} = N^2\beta * \max\left(\frac{P_r}{T}, \frac{2R_r}{T}, \frac{2}{T}\right) \tag{7.12}$$

The optimum partition shape minimises T_{comm}. The graph of these three functions is found in Figure 7.9.

PCB Optimal Shape Proofs. This section contains the theorems and proofs which state for what system characteristics each candidate shape is optimal.

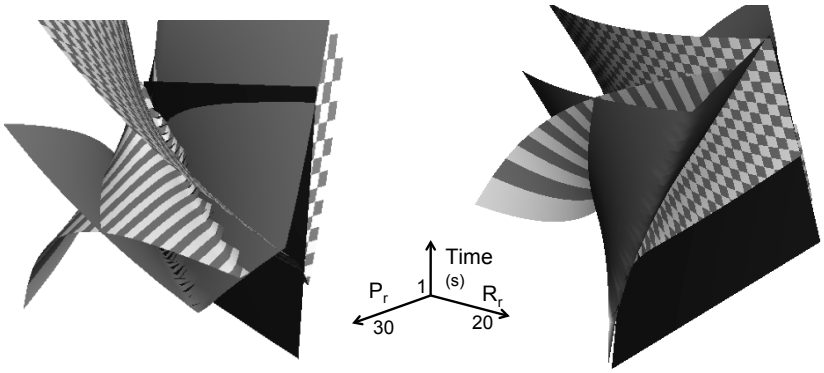


Figure 7.9: The PCB T_{comm} functions for the three candidate shapes, Square Corner (white and grey stripes), Block Rectangle (solid grey), and Square Rectangle (white and grey checkerboard). The vertical black surface is the equation $x = y$, and represents the problem constraint $P_r \geq R_r$. On the left, viewed from the front, on the right, rotated to be viewed from underneath (the lowest function is optimal). Notice the sharp change in slope of the Block Rectangle surface when the function dominating the max() changes.

Theorem 7.3.8 (PCB Square Corner)**.** *For matrix multiplication on three heterogeneous processors, the Square Corner partitioning shape minimises execution time, i.e. is the optimum shape, when using the PCB algorithm and the computational power ratios are such that* $P_r > 2(\sqrt{R_r T} - R_r + \sqrt{T} - 1)$.

Proof. Graphing the Square Corner shape for the PCB algorithm elucidates the values for which it minimises execution time. The range of these values lies at the border with the Block Rectangle partition shape. Begin by setting up the inequality and removing the common factors.

$$SC < BR$$

$$2\max(\sqrt{R_r T} - R_r + \sqrt{T} - 1, R_r, 1) < \max(P_r, 2R_r, 2)$$

Next, determine which portion of the max function on each side will dominate the equation within the relevant range of values, specifically when approximately $P_r > 2R_r$. Replacing the maximum functions with the dominate value gives,

$$2(\sqrt{R_r T} - R_r + \sqrt{T} - 1) < P_r$$

□

Theorem 7.3.9 (PCB Square Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Square Rectangle partitioning shape minimises*

execution time, i.e. is the optimum shape, when using the PCB algorithm and the computational power ratios are such that $P_r < 2R_r + \frac{R_r}{\sqrt{T}} - 2\sqrt{T} - 1$ *and* $P_r > 5 + \frac{R_r - 2}{\sqrt{T}}$.

Proof. Begin by stating the inequality and removing the common factors.

$$SR < BR$$

$$\max\left(T + 2\sqrt{T} - R_r - \frac{R_r}{\sqrt{T}} - 3, R_r + \frac{R_r}{\sqrt{T}}, 3\right) < \max(P_r, 2R_r, 2)$$

It is clear from the graph that the problem space of $2R_r > P_r$ should be examined more closely, to determine which functions dominate the maximum function for Square Rectangle and Block Rectangle for these ratios. Examining SR, the graphs show that $P_r > 5 + \frac{R_r - 2}{\sqrt{T}}$ must be true before the first term of the max function dominates the third term and the optimality inequality becomes,

$$T + 2\sqrt{T} - R_r - \frac{R_r}{\sqrt{T}} - 3 < 2R_r$$

$$P_r + R_r + 1 + 2\sqrt{T} - R_r - \frac{R_r}{\sqrt{T}} - 3 < 2R_r$$

$$P_r < 2R_r + \frac{R_r}{\sqrt{T}} - 2\sqrt{T} - 1$$

□

Corollary 7.3.10 (PCB Block Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Block Rectangle partition shape minimises execution time, i.e. is the optimum, for all processor computational power ratios except those specified in Theorems 7.3.8 and 7.3.9.*

Serial Communication with Overlap

The Square Corner shape is the only partition shape which has an area which can be computed without communication. For the Square Rectangle and Block Rectangle shapes, the SCO algorithm is identical to the Serial Communication with Barrier algorithm.

As with the two processor case for the Square Corner with the SCO algorithm, it is necessary not only to apply the abstract processor model, but to determine the optimal volume of data to assign to Processors R and S. As the faster processor, Processor P, has a "jumpstart" on computation, it should be assigned a larger portion of the matrix in order to optimally

distribute the computational load. The optimal size of the square assigned to R (with the size assigned to S being in proportion) is,

$$r = \frac{2 + \frac{2}{\sqrt{R_r}}}{\frac{P_r}{R_r} + \frac{2}{\sqrt{R_r}} + \frac{2}{R_r} + 2} \tag{7.13}$$

And for those ratios of computational power at which the Square Corner execution time intersects with the execution times of Square Rectangle and Block Rectangle (as seen in Figure 7.10), the execution time is given by,

$$\frac{T_{exe(SC)}}{N^3\beta} = \frac{2}{N}\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right) + \frac{2}{c}\left(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}\right) \tag{7.14}$$

More information on the derivation of these results may be found in Appendix D.

Applying the abstract processor model to Square Rectangle and Block Rectangle gives the execution times as,

$$\frac{T_{exe(SR)}}{N^3\beta} = \frac{1}{N} + \frac{2}{N}\sqrt{\frac{1}{T}} + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right)$$
$$\frac{T_{exe(BR)}}{N^3\beta} = \frac{2}{N} - \frac{P_r}{TN} + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right)$$

As with the two processor overlap algorithms, the constant factor of $N^3\beta$ has been removed to facilitate the analysis of these equations.

SCO Optimal Shape Theorems. The proofs of the included theorems can be found in Appendix D.

Theorem 7.3.11 (SCO Square Corner)**.** *For matrix multiplication on three heterogeneous processors, the Square Corner partition shape minimises execution time, i.e. is the optimum shape, when using the SCO algorithm for computational ratios such that* $P_r > \frac{\frac{2}{N}(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}})+\frac{2}{c}(r-r^2-\frac{r^2}{\sqrt{R_r}}+\frac{r}{\sqrt{R_r}}-\frac{r^2}{R_r})-\frac{2}{N}}{\frac{1}{Tc}-\frac{1}{TN}}$ *and* $P_r > \frac{2c}{N}(\sqrt{R_rT}+\sqrt{T})+2T(r-r^2-\frac{r^2}{\sqrt{R_r}}+\frac{r}{\sqrt{R_r}}-\frac{r^2}{R_r})-\frac{Tc}{N}-\frac{2c}{N}\sqrt{T}$*, where* r *is the optimal size of the square* R*, given in (7.13).*

Theorem 7.3.12 (SCO Square Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Square Rectangle partition shape minimises execution time, i.e. is the optimum shape, when using the SCO algorithm for computational ratios such that* $P_r < T - 2\sqrt{T}$ *and* $P_r < \frac{2c}{N}(\sqrt{R_rT}+\sqrt{T})+2T(r-r^2-\frac{r^2}{\sqrt{R_r}}+\frac{r}{\sqrt{R_r}}-\frac{r^2}{R_r})-\frac{Tc}{N}-\frac{2c}{N}\sqrt{T}$

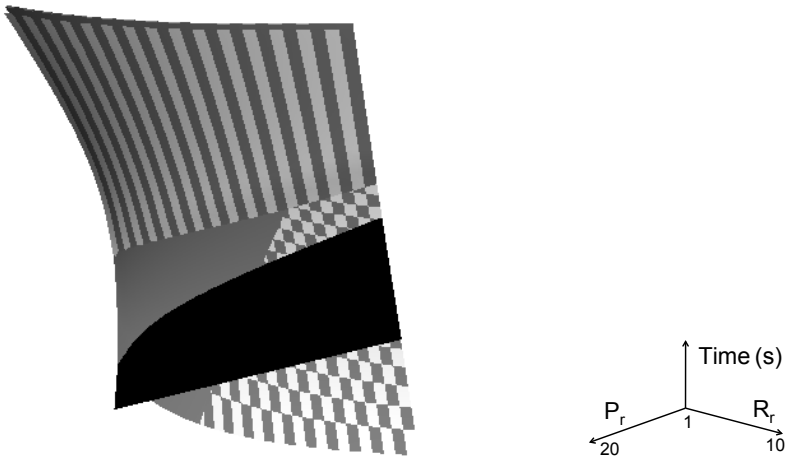


Figure 7.10: Using the SCO algorithm, the $\frac{T_{exe}}{N^3\beta}$ functions for the three candidate shapes, Square Corner (white and grey stripes), Block Rectangle (solid grey), and Square Rectangle (white and grey checkerboard). With the large constant $N^3\beta$ removed, each function appears closer than in reality so only the view from underneath is instructive. As shown, $c = 50, N = 3000$.

Corollary 7.3.13 (SCO Block Rectangle)**.** *The Block Rectangle partition shape minimises execution time, i.e. is the optimum shape, for all processor computational power ratios except those specified in Theorems 7.3.11 and 7.3.12.*

Parallel Communication with Overlap

As with the SCO algorithm, only the Square Corner shape has a portion which can be computed without communication. For this reason, the Square Rectangle and Block Rectangle shapes have the same execution time under PCO as under the PCB algorithm.

The derivation of the optimal size squares to assign to Processors R and S in the Square Corner shape, as well as the determination of which function dominates the maximum function, can be found in Appendix D.

The graphs of each of the three shapes for this algorithm can be found in Figure 7.11.

PCO Optimal Shape Proofs. The optimal shape under the PCO algorithm depends on the value of c. When examining all three shapes to determine the optimal, it is seen that as c decreases, all three equations converge.

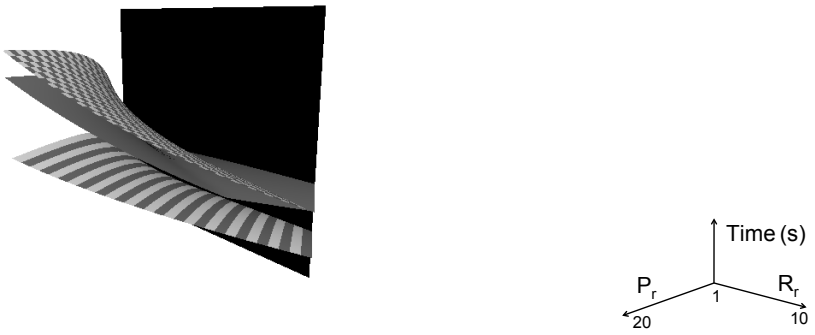


Figure 7.11: Uising the PCO algorithm, the $\frac{T_{exe}}{N^3\beta}$ functions for the three candidate shapes, Square Corner (white and grey stripes), Block Rectangle (solid grey), and Square Rectangle (white and grey checkerboard). Even with the large constant $N^3\beta$ removed, the Square Corner shape is clearly the minimum. As shown, $c = 50, N = 3000$.

However, for larger values of c, the Square Corner shape is optimal.

Theorem 7.3.14 (PCO Square Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Square Rectangle partition shape minimises execution time, i.e. is the optimum shape, when using the PCO algorithm for computational ratios such that* $P_r < \frac{1+\frac{2}{\sqrt{T}}-\frac{R_r}{T}-\frac{R_r}{T\sqrt{T}}-\frac{3}{T}-2r^2}{N(\frac{r^2}{cR_r}-\frac{1}{Tc})}$.

Proof. Examining the equations, notice the Square Corner shape equation is dominated by the communication and computation of R when the Square Rectangle shape is dominated by the communication and computation of P.

$$\frac{2}{N}r^2 + \frac{r^2P_r}{cR_r} < \frac{1}{N} + \frac{2}{N\sqrt{T}} - \frac{R_r}{NT} - \frac{R_r}{NT\sqrt{T}} - \frac{3}{NT} + \frac{P_r}{Tc}$$

$$P_r < \frac{1 + \frac{2}{\sqrt{T}} - \frac{R_r}{T} - \frac{R_r}{T\sqrt{T}} - \frac{3}{T} - 2r^2}{N(\frac{r^2}{cR_r} - \frac{1}{Tc})}$$

□

Corollary 7.3.15 (PCO Square Corner)**.** *The Square Corner partition shape minimises execution time, i.e. is the optimum shape, for all processor computational power ratios except those specified in Theorem 7.3.14.*

Parallel Interleaving Overlap

The Parallel Interleaving Overlap (PIO) algorithm, unlike the previous algorithms described, does not use bulk communication. At each step data is sent, a row and a column (or k rows and columns) at a time, by the relevant processor(s) to all processor(s) requiring those elements, while, in *parallel*, all processors compute using the data sent in the previous step.

In the case of the PIO algorithm, the processors compute at the same time, meaning the optimal distribution will be in proportion to their computational power. The optimal size of the r and s for the Square Corner is therefore $\sqrt{\frac{R_r N^2}{T}}$ and $\sqrt{\frac{N^2}{T}}$, respectively. In order to analyse the equations, the constant factor $N^4\beta$ is removed and the focus placed on the dominant middle term which is multiplied by $(N-1)$.

$$\frac{T_{exe(SC)}}{N^4\beta} = \max\left(\frac{2}{N^2}\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right), \frac{P_r}{Tc}\right) \tag{7.15}$$

$$\frac{T_{exe(SR)}}{N^4\beta} = \max\left(\frac{2}{N^2}\left(1 + 2\sqrt{\frac{1}{T}}\right), \frac{P_r}{Tc}\right) \tag{7.16}$$

$$\frac{T_{exe(BR)}}{N^4\beta} = \max\left(\frac{P_r}{N^2T}, \frac{P_r}{Tc}\right) \tag{7.17}$$

PIO Optimal Shape Proofs.

Theorem 7.3.16 (PIO Block Rectangle)**.** *For matrix multiplication on three heterogeneous processors, the Block Rectangle partition shape minimises execution time when using the PIO algorithm for computational power ratios such that* $P_r < 4\sqrt{T}$.

Proof. Either communication or computation will dominate the maximum function of all the potentially optimal partition shapes. If computation dominates, all shapes are equivalent. The communication will dominate the Block Rectangle partition shape when $c > N^2$. In this case, the inequalities may

be set as follows.

$$
\begin{aligned}
&BR < SC \\
&\frac{P_r}{N^2T} < \frac{2}{N^2}\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right) \\
&P_r < 2\sqrt{R_rT} + 2\sqrt{T} \\
&P_r < 4\sqrt{T}
\end{aligned}
$$

$$
\begin{aligned}
&BR < SR \\
&\frac{P_r}{N^2T} < \frac{2}{N^2}\left(1 + 2\sqrt{\frac{1}{T}}\right) \\
&P_r < 2T + 2\sqrt{T} \qquad \text{Always true as by definition } P_r < T
\end{aligned}
$$

□

Corollary 7.3.17 (PIO Square Corner)**.** *The Square Corner partition shape minimises execution time, i.e. is the optimum shape, for all processor computational power ratios except those specified in Theorem 7.3.16 when using the PIO algorithm.*

7.3.3 Experimental Results

To validate the theoretical results of this paper experiments were undertaken on Grid'5000 in France using the Edel cluster at the Grenoble site. Each algorithm was tested using three nodes, comprised of 2 Intel Xeon E5520 2.2 GHz CPUs per node, with 4 cores per CPU. The communication interconnect is MPI over gigabit ethernet, and the computations use ATLAS. Heterogeneity in processing power was achieved using the cpulimit [62] program, an open source code that limits the number of cycles a process may be active on the CPU to a percentage of the total.

Serial Communication with Barrier

The experimental results, for communication time, with the SCB algorithm can be found in Figure 7.12. Note it is not possible to form a Square Corner shape at ratio $1 : 1 : 1$. These experiments show that the theoretical optimum does indeed outperform the other possible shapes, which also perform in the expected order. We did find, that while the Square Corner and Square

Rectangle shapes are theoretically identical at the $14 : 5 : 1$ ratio, the Square Rectangle performed slightly better experimentally.

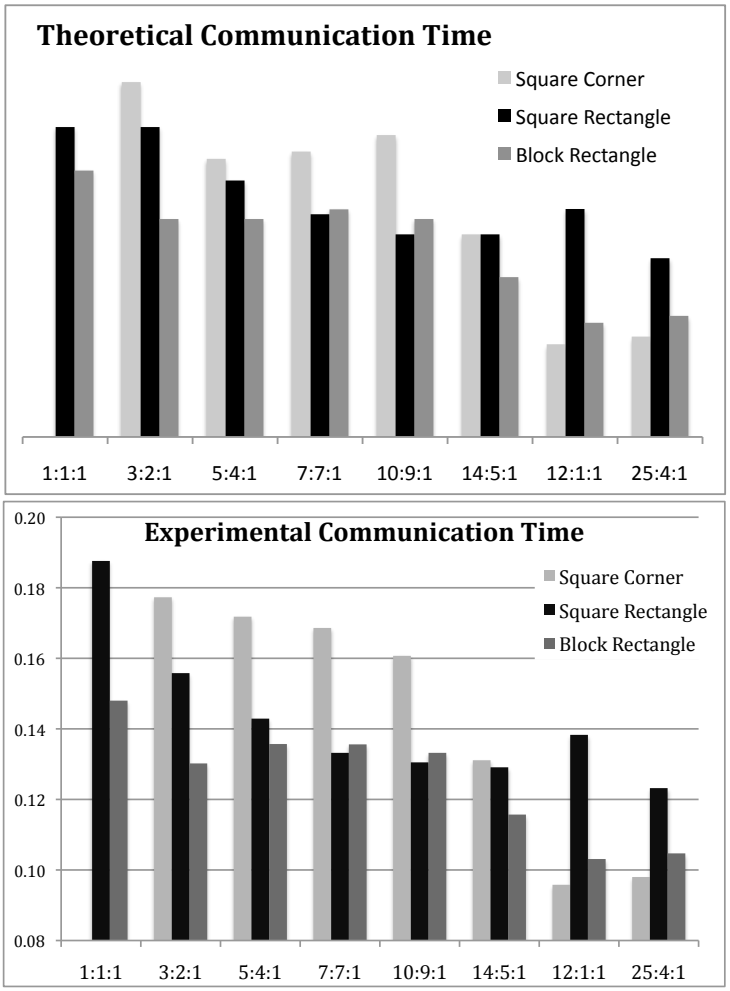


Figure 7.12: On top is the theoretical relative communication time for Square Corner, Square Rectangle and Block Rectangle when using the SCB algorithm. On bottom is the experimental communication time (in seconds) for given ratios of $P_r : R_r : 1$. The value of N is 5000.

Parallel Communication with Barrier

The experimental results, for communication time, with the PCB algorithm can be found in Figure 7.13. Note it is not possible to form a Square Corner shape at ratio $1 : 1 : 1$. The results conform to the theoretical predictions

with the optimum shape performing best, and the other two shapes performing in their predicted order.

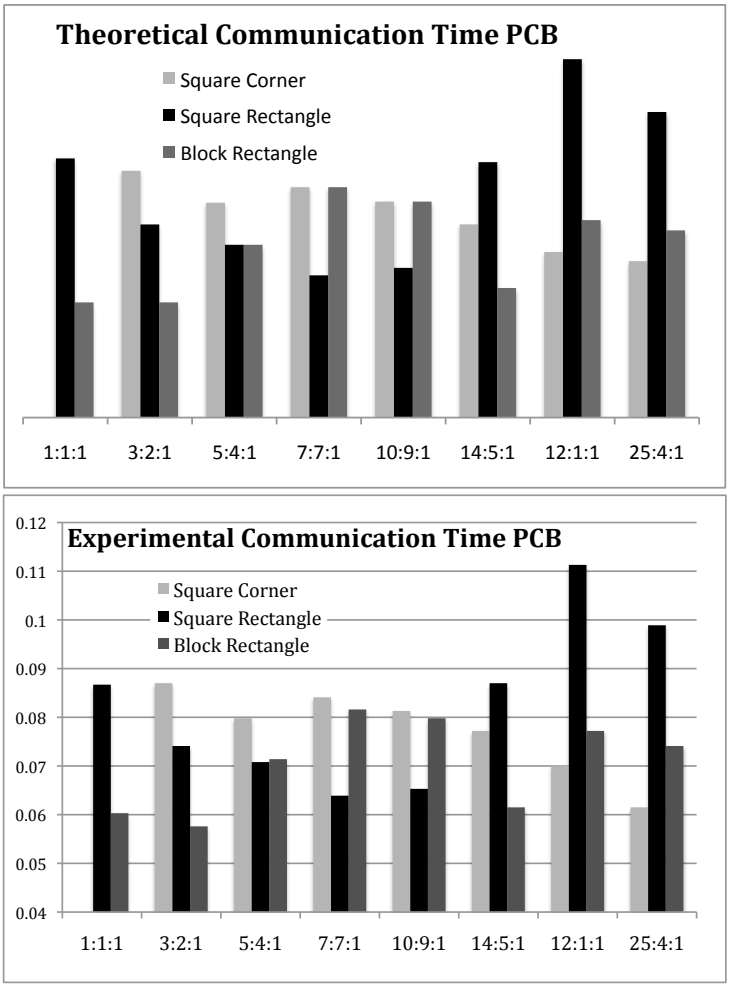


Figure 7.13: On top is the theoretical relative communication time for Square Corner, Square Rectangle and Block Rectangle partition shapes when using the PCB algorithm. On bottom is the experimental communication time (in seconds) for given ratios of $P_r : R_r : 1$. The value of N is 5000.

Overlap Algorithms

For both the Serial Communication with Overlap and the Parallel Communication with Overlap algorithms, two of the shapes (Square Rectangle and Block Rectangle) are identical to the respective barrier algorithm. These

experimental results are found in Figures 7.12 and 7.13. The Square Corner shape was implemented and provides modest speedups over the Square Corner results in previous experiments, in keeping with the theoretical predictions.

However, the experiments show more clearly than theoretical data at what ratios of computational power it becomes advantageous to not only decrease the amount of data given to the slower processors, but to eliminate them altogether. In other words, when a single fast processor, unencumbered by communication performs the entire computation. Particularly in these experiments, with computationally fast nodes and small(ish) problem sizes, this occurred around ratios of $25 : 1 : 1$. Further exploration on the bounds of useful heterogeneity would be an interesting exercise, but is not done here.

7.4 Star Network Topology

The star network topologies, of which there are three variants when using three processors, alter the necessary volume of communication and thereby the optimality of each shape. Beginning again with the six shapes under Archetype A of the Push DFA, the candidates are pruned to four potentially optimal shapes. These are Square Corner, Square Rectangle, Block Rectangle, and L Rectangle. Finally, the optimal shapes for all computational ratios, for each of the three topology variants, and for all five algorithms, are found.

7.4.1 Pruning the Optimal Candidates

The Rectangle Corner is transformed into the Block Rectangle in the same manner as for the fully connected topology. However, for the first star variant, minimising the communication between Processors R and S is advantageous, so the Rectangle Corner is the canonical form, modified so that the combined widths of Processors R and S are equal to N, rather than $N - 1$. Naturally, this is only relevant for those ratios where a Square Corner partition cannot be made. For parallel communication, Block Rectangle is the better shape for the majority of ratios, and neither shape is competitive to be the minimum at those ratios for which the Block Rectangle is not superior to Rectangle Corner. Therefore, the Rectangle Corner is superseded by the Block Rectangle shape for all but variant one serial communication.

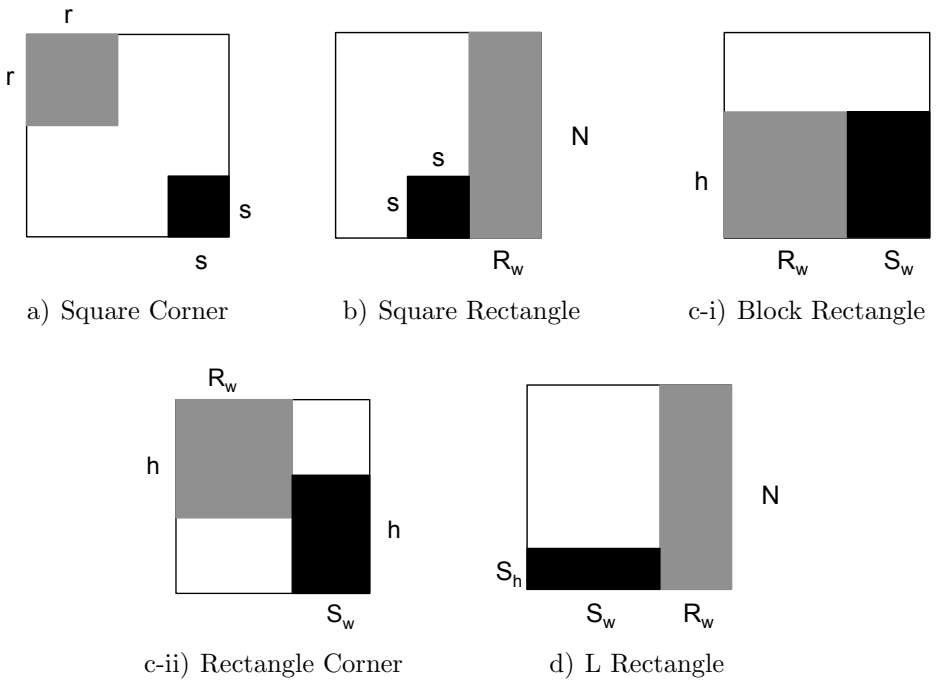
The 1D Rectangle can be shown to have a higher volume of communication than Block Rectangle for all three variants of the star network topology. This can be simply observed by noting that using the 1D rectangle partition

shape requires each processor to send its entire volume of data to each other processor. However, in the Block Rectangle, while Processors R and S do send their entire volume to all processors, Processor P must only send a portion of it's data to each other processor.

Therefore, the four candidate partition shapes to consider are Square Corner, Square Rectangle, Block Rectangle (Rectangle Corner for variant one serial), and L Rectangle.

7.4.2 Applying the Abstract Processor Model

To apply the abstract processor model for each algorithm, first the necessary volume of data in each direction must be quantified for all four shapes.



a) Square Corner b) Square Rectangle c-i) Block Rectangle

c-ii) Rectangle Corner d) L Rectangle

Figure 7.14: The four candidate shapes for three heterogeneous processors with a star network topology. The Block Rectangle canonical form is changed to Rectangle Corner for the first variant of the star topology only. The labels to the various dimensions allow the data volumes to be quantified for application of the abstract processor and algorithm models.

Volume of Communication

The total volume of communication, broken down by processor and direction is given for each candidate shape. However, note that while common naming conventions between shapes does not imply equal values, *i.e.* while S_w stands for the width of Processor S in multiple shapes, the actual width of S in those shapes is not necessarily equivalent. Also note that if the entire volume assigned to a given processor must be sent, it is given in the form of the ratio multiplied by the number of elements in the matrix, *e.g.* $\frac{P_r N^2}{T}$ for Processor P.

Square Corner Volume of Communication. The Square Corner volume of communication is derived using the naming scheme found in Figure 7.14a. Processors R and S do not share any rows or columns, and therefore do not need to communicate.

- $P \rightarrow R$ has data volume of $2r(N - r)$ elements
- $P \rightarrow S$ has data volume of $2s(N - s)$ elements
- $R \rightarrow P$ has data volume of $2r^2$ elements
- $R \rightarrow S$ has data volume of 0 elements, no communication necessary
- $S \rightarrow P$ has data volume of $2s^2$ elements
- $S \rightarrow R$ has data volume of 0 elements, no communication necessary

Square Rectangle Volume of Communication. The Square Rectangle volume of communication is derived from Figure 7.14b.

- $P \rightarrow R$ has data volume of $\frac{P_r N^2}{T}$ elements
- $P \rightarrow S$ has data volume of $2sN - 2s^2 - sR_w$ elements
- $R \rightarrow P$ has data volume of $\frac{R_r N^2}{T}$ elements
- $R \rightarrow S$ has data volume of sR_w elements
- $S \rightarrow P$ has data volume of $2s^2$ elements
- $S \rightarrow R$ has data volume of s^2 elements

Block Rectangle Volume of Communication. The Block Rectangle volume of communication is derived from Figure 7.14c-i. This is the canonical shape for star topology variants two and three (when Processor R and S are the centre of the star, respectively). It is also the canonical form for variant one type with parallel communication.

- $P \rightarrow R$ has data volume of $R_w(N - h)$ elements
- $P \rightarrow S$ has data volume of $S_w(N - h)$ elements
- $R \rightarrow P$ has data volume of $\frac{R_r N^2}{T}$ elements
- $R \rightarrow S$ has data volume of $\frac{R_r N^2}{T}$ elements
- $S \rightarrow P$ has data volume of $\frac{N^2}{T}$ elements
- $S \rightarrow R$ has data volume of $\frac{N^2}{T}$ elements

Rectangle Corner Volume of Communication. The Rectangle Corner volume of communication is derived from Figure 7.14c-ii. This is the canonical shape for star topology variant one, when Processor P is the centre of the star, for serial communication. This decreases the communication between Processors R and S, which is advantageous as no direct communication link exists between these processors for this variant. For parallel communication, there is a point at which the Block Rectangle is the better form, and Rectangle Corner is never the minimum due to better performance by other shapes.

- $P \rightarrow R$ has data volume of $\frac{P_r N^2}{T}$ elements
- $P \rightarrow S$ has data volume of $\frac{P_r N^2}{T}$ elements
- $R \rightarrow P$ has data volume of $\frac{R_r N^2}{T} + R_w(N - h)$ elements
- $R \rightarrow S$ has data volume of $R_w(2h - N)$ elements
- $S \rightarrow P$ has data volume of $\frac{N^2}{T} + S_w(N - h)$ elements
- $S \rightarrow R$ has data volume of $S_w(2h - N)$ elements

L Rectangle Volume of Communication.

- $P \to R$ has data volume of $\frac{P_r N^2}{T}$ elements
- $P \to S$ has data volume of $\frac{P_r N^2}{T}$ elements
- $R \to P$ has data volume of $R_w(N - S_h)$ elements
- $R \to S$ has data volume of $R_w S_h$ elements
- $S \to P$ has data volume of $\frac{N^2}{T}$ elements
- $S \to R$ has data volume of $\frac{N^2}{T}$ elements

Model Equations by Shape, Variant, and Communication Algorithm

For each of the three variants of the star network topology, depicted in Figure 7.5, the above communication volumes are translated into equations for both serial and parallel communication. Note, in some places the equations have been simplified to increase readability. Particularly in the case of maximum functions, any terms which may be mathematically shown to never dominate the maximum have been removed.

Variant One. The first variant of the star network topology puts Processor P at the centre of the star. Communication cost between Processor R and S is doubled, which is the worst case scenario.

The Variant One Serial Communication equations, for Square Corner, Square Rectangle, Rectangle Corner, and L Rectangle, are given respectively by,

$$\frac{T_{comm(SC)}}{N^2} = 2\sqrt{\frac{R_r}{T}} + 2\sqrt{\frac{1}{T}} \tag{7.18}$$

$$\frac{T_{comm(SR)}}{N^2} = 1 + 2\sqrt{\frac{1}{T}} + \frac{R_r}{T}\sqrt{\frac{1}{T}} + \frac{1}{T} \tag{7.19}$$

$$\frac{T_{comm(RC)}}{N^2} = \frac{2P_r}{T} + \frac{4R_r}{T} + \frac{4}{T} - \frac{R_r}{T - P_r} - \frac{1}{T - P_r} \tag{7.20}$$

$$\frac{T_{comm(LR)}}{N^2} = \frac{2P_r}{T} + \frac{3}{T} + \frac{R_r}{T} + \frac{R_r}{T(T - R_r)} \tag{7.21}$$

The Variant One Parallel Communication equations, for Square Corner, Square Rectangle, Block Rectangle, and L Rectangle, are given respectively

by,

$$\frac{T_{comm(SC)}}{N^2} = \max\Big(2\sqrt{\frac{R_r}{T}} - \frac{2R_r}{T}, 2\sqrt{\frac{1}{T}} - \frac{2}{T}, \frac{2R_r}{T}\Big) \tag{7.22}$$

$$\frac{T_{comm(SR)}}{N^2} = \max\Bigg(\max\Big(\frac{P_r}{T}, \frac{3}{T}\Big) + \frac{1}{T}, \max\Big(2\sqrt{\frac{1}{T}} - \frac{2}{T} - \frac{R_r}{T}\sqrt{\frac{1}{T}}, \frac{R_r}{T} + \frac{R_r}{T}\sqrt{\frac{1}{T}}\Big) + \frac{R_r}{T}\sqrt{\frac{1}{T}}\Big) \tag{7.23}$$

$$\frac{T_{comm(BR)}}{N^2} = \max\Bigg(\max\Big(\frac{R_r}{T-P_r} - \frac{R_r}{T}, \frac{2}{T}\Big) + \frac{1}{T}, \max\Big(\frac{1}{T-P_r} - \frac{1}{T}, \frac{2R_r}{T}\Big) + \frac{R_r}{T}\Big) \tag{7.24}$$

$$\frac{T_{comm(LR)}}{N^2} = \max\Bigg(\max\Big(\frac{P_r}{T}, \frac{2}{T}\Big) + \frac{1}{T}, \frac{P_r}{T} + \frac{R_r}{T(T-R_r)}\Big) \tag{7.25}$$

Variant Two. The second variant of the star network topology puts Processor R at the centre of the star. Communication cost between Processors P and S is doubled, which is the worst case scenario.

The Variant Two Serial Communication equations, for Square Corner, Square Rectangle, Block Rectangle, and L Rectangle, are given respectively by,

$$\frac{T_{comm(SC)}}{N^2} = 2\sqrt{\frac{R_r}{T}} + 4\sqrt{\frac{1}{T}} \tag{7.26}$$

$$\frac{T_{comm(SR)}}{N^2} = 1 - \frac{R_r}{T}\sqrt{\frac{1}{T}} + 4\sqrt{\frac{1}{T}} \tag{7.27}$$

$$\frac{T_{comm(BR)}}{N^2} = \frac{2R_r}{T} + \frac{3}{T} + \frac{R_r}{R_r+1}\Big(1 - \frac{R_r+1}{T}\Big) + \frac{2}{R_r+1}\Big(1 - \frac{R_r+1}{T}\Big) \tag{7.28}$$

$$\frac{T_{comm(LR)}}{N^2} = \frac{3P_r}{T} + \frac{3}{T} + \frac{R_r}{T} \tag{7.29}$$

The Variant Two Parallel Communication equations, for Square Corner, Square Rectangle, Block Rectangle, and L Rectangle, are given respectively

by,

$$\frac{T_{comm(SC)}}{N^2} = \max\Big(\frac{2R_r}{T} + \frac{2}{T}, 2\sqrt{\frac{R_r}{T}} - \frac{2R_r}{T} + 4\sqrt{\frac{1}{T}} - \frac{4}{T}\Big) \tag{7.30}$$

$$\begin{aligned}\frac{T_{comm(SR)}}{N^2} = \max\Bigg(&\max\Big(\frac{R_r}{T}, \frac{3}{T}\Big) + \frac{2}{T},\\ &\max\Big(\frac{P_r}{T} + 2\sqrt{\frac{1}{T}} - \frac{2}{T} - \frac{R_r}{T}\sqrt{\frac{1}{T}}, \frac{R_r}{T}\sqrt{\frac{1}{T}}\Big)\\ &+ 2\sqrt{\frac{1}{T}} - \frac{2}{T} - \frac{R_r}{T}\sqrt{\frac{1}{T}}\Bigg)\end{aligned} \tag{7.31}$$

$$\begin{aligned}\frac{T_{comm(BR)}}{N^2} = \max\Bigg(&\max\Big(\frac{R_r}{T}, \frac{2}{T}\Big) + \frac{1}{T},\\ &\max\Big(\frac{R_r}{T - P_r} - \frac{R_r}{T} + \frac{1}{T - P_r} - \frac{1}{T}, \frac{R_r}{T}\Big) + \frac{1}{T - P_r} - \frac{1}{T}\Bigg)\end{aligned} \tag{7.32}$$

$$\begin{aligned}\frac{T_{comm(LR)}}{N^2} = \max\Bigg(&\max\Big(\frac{R_r}{T} - \frac{R_r}{T(T - R_r)}, \frac{2}{T}\Big) + \frac{1}{T},\\ &\max\Big(\frac{2P_r}{T}, \frac{R_r}{T(T - R_r)}\Big) + \frac{P_r}{T}\Bigg)\end{aligned} \tag{7.33}$$

Variant Three. The third variant of the star network topology puts Processor S at the centre of the star. Communication cost between Processors P and R is doubled.

The Variant Three Serial Communication equations, for Square Corner, Square Rectangle, Block Rectangle, and L Rectangle, are given respectively by,

$$\frac{T_{comm(SC)}}{N^2} = 4\sqrt{\frac{R_r}{T}} + 2\sqrt{\frac{1}{T}} \tag{7.34}$$

$$\frac{T_{comm(SR)}}{N^2} = 2\frac{P_r}{T} + \frac{2R_r}{T} + 2\sqrt{\frac{1}{T}} + \frac{1}{T} \tag{7.35}$$

$$\frac{T_{comm(BR)}}{N^2} = \frac{P_r}{T} + \frac{3R_r}{T} + \frac{R_r}{R_r + 1}\Big(1 - \frac{R_r + 1}{T}\Big) + \frac{2}{T} \tag{7.36}$$

$$\frac{T_{comm(LR)}}{N^2} = \frac{3P_r}{T} + \frac{2}{T} + \frac{R_r}{T} + \frac{R_r}{T}\Big(1 - \frac{1}{T - R_r}\Big) \tag{7.37}$$

The Variant Three Parallel Communication equations, for Square Corner, Square Rectangle, Block Rectangle, and L Rectangle, are given respectively by,

$$\frac{T_{comm(SC)}}{N^2} = \max\left(\frac{4R_r}{T}, 2\sqrt{\frac{1}{T}} - \frac{2}{T} + 4\sqrt{\frac{R_r}{T}} - \frac{4R_r}{T}\right) \tag{7.38}$$

$$\frac{T_{comm(SR)}}{N^2} = \max\Bigg(\max\left(\frac{R_r}{T}\sqrt{\frac{1}{T}} + \frac{R_r}{T}, \frac{2}{T}\right) + \frac{R_r}{T}, \max\left(2\sqrt{\frac{1}{T}} - \frac{2}{T} - \frac{R_r}{T}\sqrt{\frac{1}{T}} + \frac{P_r}{T}, \frac{1}{T}\right) + \frac{P_r}{T}\Bigg) \tag{7.39}$$

$$\frac{T_{comm(BR)}}{N^2} = \max\left(\frac{3P_r}{T}, \max\left(\frac{R_r}{T-P_r} - \frac{R_r}{T} + \frac{1}{T-P_r} - \frac{1}{T}, \frac{1}{T}\right) + \frac{R_r}{T-P_r} - \frac{R_r}{T}\right) \tag{7.40}$$

$$\frac{T_{comm(LR)}}{N^2} = \frac{3P_r}{T} \tag{7.41}$$

7.4.3 Optimal Three Processor ST Data Partition

The exact system characteristics for which each candidate shape is optimal depends on the variant of star network topology used, however, some general observations can be made.

- Square Corner is optimal for systems with a single fast processor, and two relatively slower processors
- Square Corner is particularly effective for variant one topology, due to not requiring any communication along the non-existent link
- Square Rectangle is optimal for systems with two fast processors, and a single relatively slow processor
- Block Rectangle is optimal for systems with a fast, medium, and slow processor
- Block Rectangle is heavily affected by topology, depending on the variant used, either being optimal for most ratios, or no ratios at all.
- L Rectangle is optimal for relatively homogeneous systems

Variant One

The optimal data partition shape for variant one star network topology (P at centre, no link between R and S) is unsurprisingly favoured toward the

non-rectangular data partitions. The Square Corner shape especially has the advantage of not requiring any message to be forwarded through Processor P. The L Rectangle, which was discarded as inferior when considering the fully connected network topology, emerges as optimal for relatively homogeneous systems.

Serial Communication. For serial communication with variant one star topology, the optimal shape is always one of Square Corner, Square Rectangle, and L Rectangle, as seen in Figure 7.15. The intersection of these surfaces gives the ratios at which each is the optimal shape.

The following theorems are created by solving for these surface intersections.

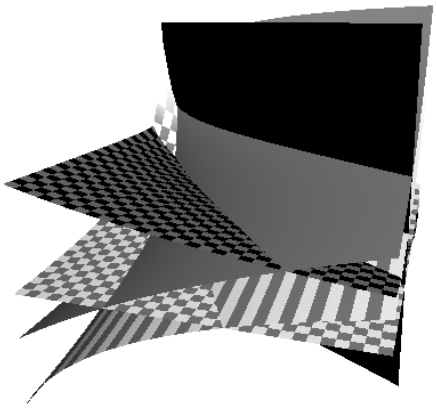
Theorem 7.4.1 (Square Corner Serial Variant One)**.** *For matrix multiplication on three heterogeneous processors, with a variant one star network topology, the Square Corner shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r > 2\sqrt{R_r T} - R_r - \frac{R_r}{\sqrt{T}} - 2$ *and* $P_r > T\sqrt{\frac{R_r}{T}} + T\sqrt{\frac{1}{T}} - \frac{R_r}{2(T-R_r)} - \frac{R_r}{2} - \frac{3}{2}$.

Theorem 7.4.2 (Square Rectangle Serial Variant One)**.** *For matrix multiplication on three heterogeneous processors, with a variant one star network topology, the Square Rectangle shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r < 2\sqrt{R_r T} - R_r - \frac{R_r}{\sqrt{T}} - 2$ *and* $P_r > 2\sqrt{T} + \frac{R_r}{\sqrt{T}} - \frac{R_r}{T-R_r} - 1$.

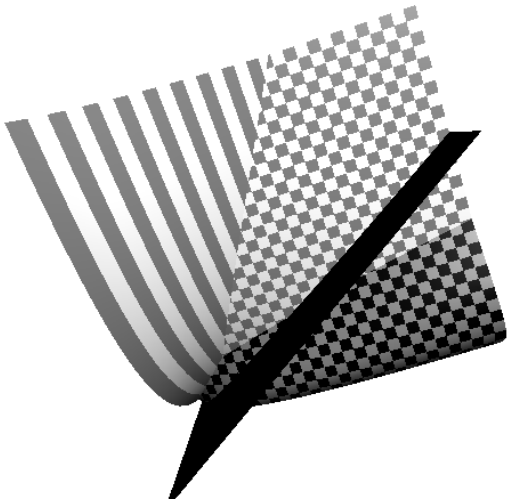
Corollary 7.4.3 (L Rectangle Serial Variant One)**.** *For matrix multiplication on three heterogeneous processors, with a variant one star network topology, the L Rectangle shape is the optimum, i.e. minimises execution time, for ratios not given in Theorems 7.4.1 and 7.4.2.*

Parallel Communication. Using parallel communication for the variant one star network topology yields only two optimal data partition shapes, Square Corner and L Rectangle. As seen in Figure 7.16, each shape is the optimal for approximately half of the problem domain, so to determine the exact ratio, we focus on the terms dominating the maximum function at approximately $P_r = 2R_r$.

It is worth noting that around this area, the Square Rectangle shape is also a very good shape and is asymptotically equal to the L Rectangle shape for that brief time.



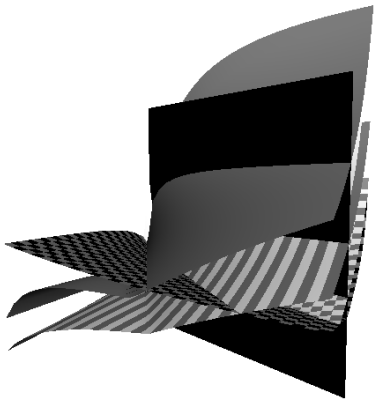
a) Front View



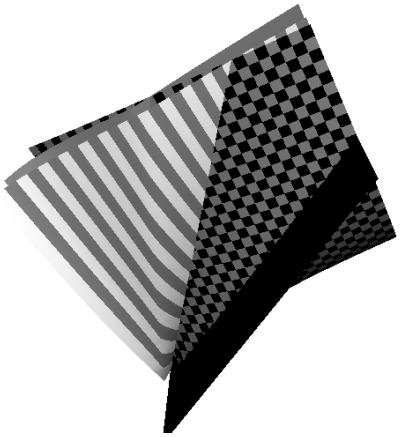
b) Bottom View

Figure 7.15: The four candidate shapes for three heterogeneous processors with a variant one star network topology for serial communication time. The shapes are Square Corner (white and grey stripes), Square Rectangle (white and grey checkerboard), Rectangle Corner (solid grey), and L Rectangle (black and grey checkerboard).

Theorem 7.4.4 (Square Corner Parallel Variant One)**.** *For matrix multiplication on three heterogeneous processors, with a variant one star network topology, the Square Rectangle shape is the optimum, i.e. minimises execu-*



a) Front View



b) Bottom View

Figure 7.16: The four candidate shapes for three heterogeneous processors with a variant one star network topology for parallel communication time. The shapes are Square Corner (white and grey stripes), Square Rectangle (white and grey checkerboard), Rectangle Corner (solid grey), and L Rectangle (black and grey checkerboard).

tion time, for ratios such that $P_r > 2R_r - 1$.

Corollary 7.4.5 (L Rectangle Parallel Variant One)**.** *For matrix multiplication on three heterogeneous processors, with a variant one star network topology, the L Rectangle shape is the optimum, i.e. minimises execution time, for ratios not given in Theorems 7.4.4.*

Variant Two

The optimal shape for variant two of the star network topology (with R at the centre of the star) is often dominated by the Block Rectangle partition shape, except in cases of extreme heterogeneity in processing speeds.

Serial Communication. For serial communication with variant two star topology, the optimal shape is always one of Square Corner, Square Rectangle, and Block Rectangle, with Block Rectangle being optimal for the largest range of ratios, as seen in Figure 7.17. The intersection of these surfaces gives the ratios at which each is the optimal shape.

The following theorems are created by solving for these surface intersections.

Theorem 7.4.6 (Square Corner Serial Variant Two)**.** *For matrix multiplication on three heterogeneous processors, with a variant two star network topology, the Square Corner shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r > 2\sqrt{R_r T} + 4\sqrt{T} - 2R_r + \frac{T}{R_r+1} - 2$.

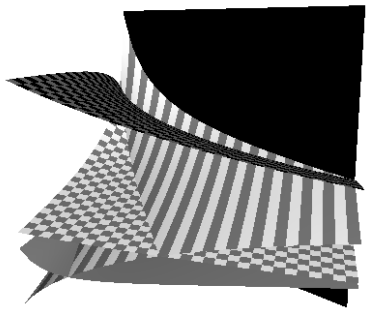
Theorem 7.4.7 (Square Rectangle Serial Variant Two)**.** *For matrix multiplication on three heterogeneous processors, with a variant two star network topology, the Square Rectangle shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r < T + \frac{T}{R_r+1} + \frac{R_r}{\sqrt{T}} - 4\sqrt{T}$.

Corollary 7.4.8 (Block Rectangle Serial Variant Two)**.** *For matrix multiplication on three heterogeneous processors, with a variant two star network topology, the Block Rectangle shape is the optimum, i.e. minimises execution time, for all ratios not specified in Theorems 7.4.6 and 7.4.7.*

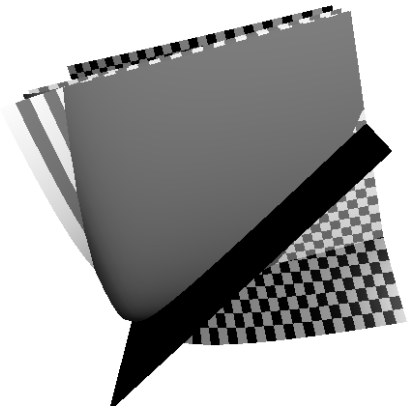
Parallel Communication. Parallel communication with the variant two star network topology yields only two optimal data partition shapes, Square Corner and Block Rectangle. As seen in Figure 7.18, the Block Rectangle is optimal for all but those highly heterogeneous ratios at which Processor P is very powerful.

Theorem 7.4.9 (Square Corner Parallel Variant Two)**.** *For matrix multiplication on three heterogeneous processors, with a variant two star network topology, the Square Rectangle shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r > T - \frac{R_r+2+\frac{R_r}{T}+\frac{2}{T}}{2\sqrt{\frac{R_r}{T}}+4\sqrt{\frac{1}{T}}}$.

Corollary 7.4.10 (Block Rectangle Parallel Variant Two)**.** *For matrix multiplication on three heterogeneous processors, with a variant two star network topology, the Block Rectangle shape is the optimum, i.e. minimises execution time, for ratios not given in Theorems 7.4.9.*



a) Front View



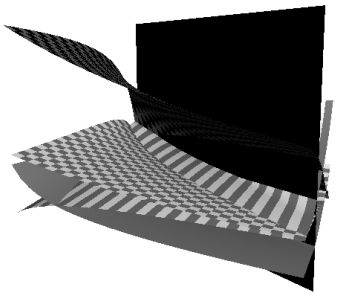
b) Bottom View

Figure 7.17: The four candidate shapes for three heterogeneous processors with a variant two star network topology for serial communication time. The shapes are Square Corner (white and grey stripes), Square Rectangle (white and grey checkerboard), Rectangle Corner (solid grey), and L Rectangle (black and grey checkerboard).

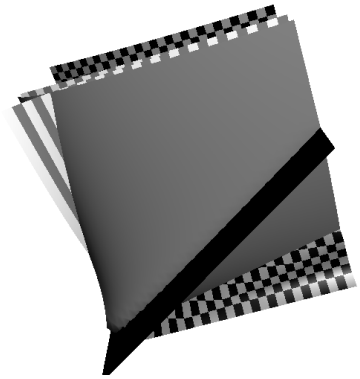
Variant Three

Serial Communication. For serial communication with variant three star topology, the optimal shape is always one of Square Corner, Square Rectangle, and Block Rectangle, as seen in Figure 7.19. The intersection of these surfaces gives the ratios at which each is the optimal shape.

The following theorems are created by solving for these surface intersections.



a) Front View



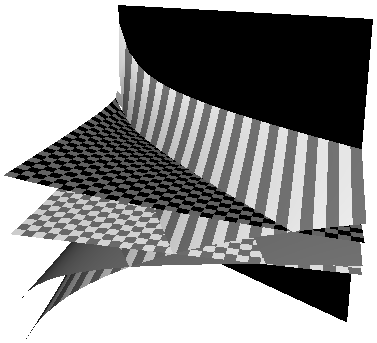
b) Bottom View

Figure 7.18: The four candidate shapes for three heterogeneous processors with a variant two star network topology for parallel communication time. The shapes are Square Corner (white and grey stripes), Square Rectangle (white and grey checkerboard), Rectangle Corner (solid grey), and L Rectangle (black and grey checkerboard).

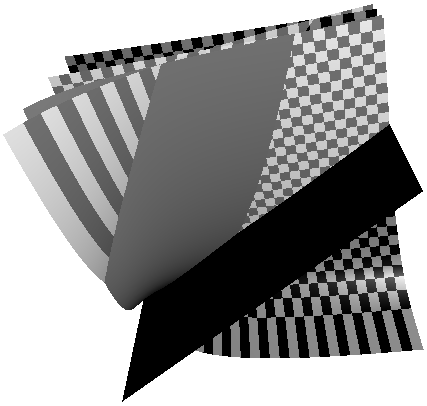
Theorem 7.4.11 (Square Corner Serial Variant Three)**.** *For matrix multiplication on three heterogeneous processors, with a variant three star network topology, the Square Corner shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r > 4\sqrt{R_r T} + 2\sqrt{T} - 2R_r + \frac{R_r T}{R_r+1} - 2$.

Theorem 7.4.12 (Square Rectangle Serial Variant Three)**.** *For matrix multiplication on three heterogeneous processors, with a variant three star network topology, the Square Rectangle shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r < 1 + \frac{R_r T}{R_r+1} - 2\sqrt{T}$.

Corollary 7.4.13 (Block Rectangle Serial Variant Three)**.** *For matrix multi-*



a) Front View



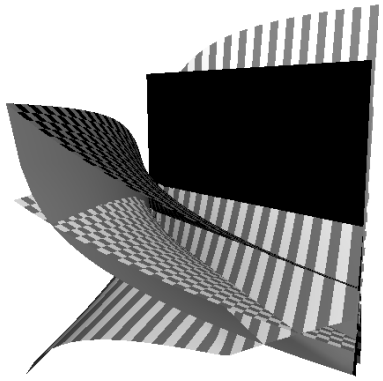
b) Bottom View

Figure 7.19: The four candidate shapes for three heterogeneous processors with a variant two star network topology for serial communication time. The shapes are Square Corner (white and grey stripes), Square Rectangle (white and grey checkerboard), Rectangle Corner (solid grey), and L Rectangle (black and grey checkerboard).

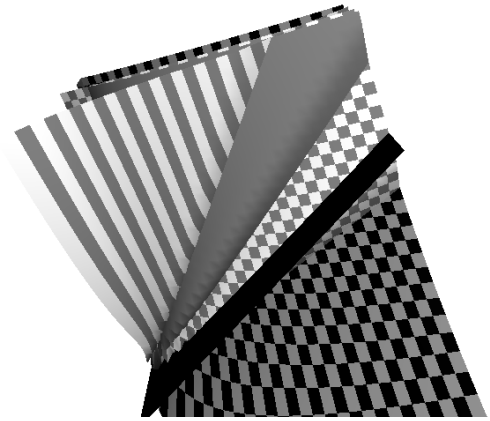
plication on three heterogeneous processors, with a variant three star network topology, the Block Rectangle shape is the optimum, i.e. minimises execution time, for all ratios not specified in Theorems 7.4.11 and 7.4.12.

Parallel Communication. Parallel communication with the variant three star network topology has three optimal data partition shapes, Square Cor-

ner, Square Rectangle, and Block Rectangle. As seen in Figure 7.20, the ratios for which these shapes are optimal are in the keeping with the pattern found throughout - Square Corner for fast P, Square Rectangle for $P_r \approx R_r$, and Block Rectangle for all those in between.



a) Front View



b) Bottom View

Figure 7.20: The four candidate shapes for three heterogeneous processors with a variant three star network topology for parallel communication time. The shapes are Square Corner (white and grey stripes), Square Rectangle (white and grey checkerboard), Rectangle Corner (solid grey), and L Rectangle (black and grey checkerboard).

Theorem 7.4.14 (Square Corner Parallel Variant Three)**.** *For matrix multiplication on three heterogeneous processors, with a variant three star network*

topology, the Square Corner shape is the optimum, i.e. minimises execution time, for ratios such that $P_r > T - \frac{(2R_r+1)T}{6R_r+1}$.

Theorem 7.4.15 (Square Rectangle Parallel Variant Three)**.** *For matrix multiplication on three heterogeneous processors, with a variant three star network topology, the Square Rectangle shape is the optimum, i.e. minimises execution time, for ratios such that* $P_r > \frac{3R_r}{2} + \frac{R_r}{2\sqrt{T}} - \sqrt{T} + 1$.

Corollary 7.4.16 (Block Rectangle Parallel Variant Three)**.** *For matrix multiplication on three heterogeneous processors, with a variant three star network topology, the Block Rectangle shape is the optimum, i.e. minimises execution time, for ratios not given in Theorems 7.4.14 and 7.4.15.*

7.5 Conclusion

The overwhelming conclusion when considering data partitioning shapes on three heterogeneous processors is that while the traditional two dimensional rectangular shape is the ubiquitous for good reason, the novel non-rectangular have significant advantages, especially as the level of heterogeneity increases.

In the fully connected network topology, the three optimal shapes formed the same pattern for all algorithms considered. The Square Corner is suited to a single fast processor, the Square Rectangle to two fast processors, and the Block Rectangle to everything in between.

More interestingly, the exploration of the star topologies gives insight into what happens when the complexity of the model is increased. Rather than favour the traditional 2D decomposition, new candidates began to vie for optimality. This should only reinforce the contention that a single blanket partition shape (Block Rectangle) is not adequate when considering the wide variability in system parameters in real life heterogeneous systems. Many other shapes are superior to this, and for some variants in topology the traditional Block Rectangle is not even a contender for the optimal shape. For each nuance in a given system, the Push Technique can be extended to provide insight into the optimum shape.

Chapter 8

Local Search Optimisations

This chapter will compare and contrast the Push Technique with other local search methods, with a focus on simulated annealing. The Push Technique is of similar computational complexity, and produces all possible output data partition shapes. Simulated annealing experimental results are given, which reinforce the completeness of the Push Technique results.

8.1 Description of Local Search

Local search is a method for solving computationally intensive optimisation problems. As previously discussed, the more limited problem of optimally partitioning heterogeneous rectangles is NP-Complete.

In general, in local search each possible solution has some associated cost, and the aim is to minimise this cost. Each solution is part of a local "neighbourhood", which is the area searched for the local minima [64]. The initial solution is repeatedly replaced with a solution in the neighbourhood with a lower cost. At some point, no neighbouring solution has a lower cost than the one found, and this is the local minimum.

A variety of algorithms operate in this manner such as Hill Climbing, Simulated Annealing, Tabu Search, and Reactive Search.

8.2 Simulated Annealing

In the literature, simulated annealing is a popular choice for local search algorithms. It allows the discovery of a variety of local minima within a large search space. At the start of the process, an initial solution is swapped with neighbouring solutions, even if those solutions provide an increased cost. As the local search progresses, or "cools", the swaps with neighbouring solutions

must be strictly beneficial in minimizing the cost. This allows the solution to move through the search space initially, exploring it fully, before settling down and finding the local minimum.

In order to function, the simulated annealing algorithm has two important parts; the objective function, and the neighbour function. The first is used to evaluate a given solution, while the second determines how solutions are swapped.

8.2.1 Experimental Method for Matrix Partitioning

To use simulated annealing to solve the optimal data partition shape problem, the first step is to define the objective function. This is understood to be the volume of communication required in a given solution. This is calculated by considering the number of processors that own elements in each row and each column of the matrix.

The next step is to define the neighbourhood function. The simplest approach would be to pick two random elements of the matrix (which are assigned to different processors) and swap their assigned processors. However, this simple function does *not* result in a tractable simulated annealing algorithm. A single swap of two random elements, in the scheme of larger partition shape problem, is very unlikely to improve the objective function.

To create a tractable algorithm, a fairly sophisticated neighbourhood function is required. Designing this function leads to swaps which are very similar to the *Push* operations discussed in this thesis. The implemented strategy is described here.

1. A row or column of the matrix is chosen at random. Assume a row is chosen. The move is to chose a batch of assignment swaps in order to lower the number of processors assigned to the selected row.

2. Specifically, a processor assigned to the chosen row is selected at random, with preference given to the processor assigned to the fewest elements in the row. Say that there are s elements in the row assigned to this processor.

3. Then s elements from the remainder of the matrix (excluding the row) are selected at random, such that these elements are assigned to one of the other processors assigned to the chosen row. Preference is given to elements with highest *score*, where the score of an element is the sum of the number of processors assigned to the row and column corresponding to that element *i.e.* the elements selected to move into the chosen row are elements from high-cost rows and columns of the matrix.

The effect of this neighbourhood function is to propose moves that definitely improve a selected row or column and which are relatively unlikely to worsen other rows and columns. The simulated annealing algorithm proceeds by iteratively proposing a move and accepting this move if it improves the objective, or with some random chance even if it worsens the move. An exploration of the simulated annealing parameters suggests that the Kirkpatrick annealing strategy with the temperature updated using $T_{k+1} = \alpha T_k, \alpha = 0.8$ works well [65].

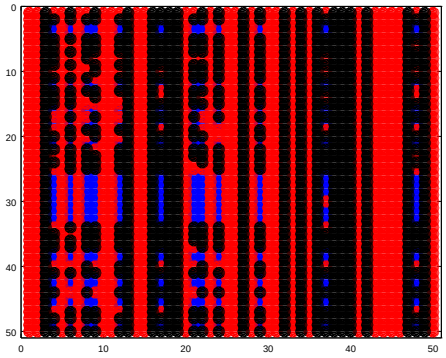


Figure 8.1: The partition shape solution found on a 50×50 matrix by Simulated Annealing, with the three processors shown in red, blue and black. This is the raw result, before permutations to create an equivalent partition are done.

8.3 Conclusion

The simulated annealing algorithm, in order to be tractable, became quite similar to the Push operation. The results of the simulated annealing give a variety of partition shapes, all of which had been previously found using the

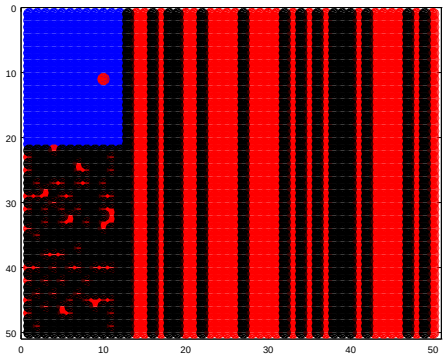


Figure 8.2: The solution found on a 50×50 matrix by Simulated Annealing. This is the final result, after permutations to create an equivalent partition are done. All rows and columns containing three processors are moved to the far left/top, then all rows and columns containing two processors, and so on.

Push. For example, in Figures 8.1 and 8.2, the results of a single run of the simulated annealing are given. At first, the results look unfamiliar, as the Push technique has a natural tendency towards a processor being assigned contiguous elements. However, a simple permutation of the matrix gives a shape that is recognisable as being in the same archetype as the Square Rectangle. This permutation involves swapping rows or columns, and does not affect the cost of the solution found.

The Push Technique is similar to local search. It starts with a random solution, and moves between solutions according to a predetermined function to arrive at some final result which is a local minimum. The Push Technique finds all possible local minima, also called candidates.

Chapter 9

The Push Technique and LU Factorisation

The Push Technique can be adapted to create optimal matrix partition shapes for a variety of matrix computations. This section will lay out the methodology for a Push on LU factorisation, showing that it satisfies the requirement for not increasing communication time of the algorithm. Finally, the Push is applied to create some two processor partition shapes.

9.1 A Brief Review of LU Factorisation

This section gives a basic review of LU factorisation, both for a single processor and in parallel on multiple processors.

9.1.1 Basic LU Factorisation

LU factorisation (also called decomposition) is a common linear algebra kernel based on Gaussian Elimination. The matrix A is factorised to become the product of two (or three) matrices. The factors L and U are triangular matrices, and a permutation matrix P may be used if row exchanges in matrix A are necessary to complete the elimination. For simplicity here, the row exchanges are ignored. The factorisation $A = LU$ is shown as,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Gaussian Elimination. The cost of performing Gaussian elimination is about $\frac{1}{3}N^3$ multiplications and $\frac{1}{3}N^3$ subtractions [35]. The main goal of the Gaussian elimination is to create a matrix in which back substitution can be used to solve the linear equation $A = xb$.

For this description, we assume that the elimination will be successful.

The elimination multiplies a row by some number (called a *multiplier*), and subtracts it from the next row. The purpose of this is to remove (eliminate) some variable from that second row. This is continued for each row, until a zero is in the first column for all but the first row. This first row, first column element is called the pivot.

The elimination then starts on the second row, second column, which is now the pivot. For each of the subsequent rows a multiplier is found and the second row is subtracted.

A simple example from [35] is repeated here, (pivots are marked with [ ] and multipliers are listed below the relevant matrix),

$$\begin{bmatrix} [1] & 2 & 1 \\ 3 & 8 & 1 \\ 0 & 4 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & [2] & -2 \\ 0 & 4 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & [5] \end{bmatrix}$$

Multipliers: 3,0 Multiplier: 2

The final matrix shown is the U of LU factorisation.

9.1.2 Parallel LU Factorisation

Computing LU factorisation in parallel on multiple processors requires an algorithm which will balance computational load, while minimising communication cost. The parallel LU factorisation algorithm used by the popular software package ScaLAPACK, as described in [66], is explained here.

In general, as pointed out by [67], this parallel algorithm can be divided into two broad parts. First is the *panel factorisation*, in which the multipliers for the gaussian elimination are calculated, and all this information is accumulated for the entire step. The second part is the *trailing sub matrix update*, in which all the changes accumulated during the panel factorisation are applied to the trailing sub matrix in a single matrix multiply step. Matrix multiplication is one of the most efficient parallel BLAS (Basic Linear Algebra Subroutines)[68] computations. For this reason, the parallel algorithm waits to update the trailing matrix with matrix multiply, rather than executing as a series of rank-1 updates.

These two parts, and their sub steps, are discussed further below.

Algorithm Description. Figures in 9.1 are adapted from James Demmel's Lecture Notes[1] on Dense Linear Algebra.

Divide the matrix A into k steps, each of width b columns. The simple example shown in Figure 9.1a has just four steps. For each row, i, within a given step k, determine if pivoting it needed. The row with the largest value in column i is swapped with row i, as shown in Figure 9.1b. Next, each row is divided by the pivot value in parallel on all processors to determine the multipliers. Subtract multiples of row i from the new sections of L and U. The completed panel is shown in Figure 9.1c.

When those steps have been repeated for each row in step k, all the work done so far (row swaps and calculation of LL) is broadcast horizontally among the processors, as shown in Figure 9.1d. The final steps are a triangular solve, and the matrix multiplication to update the trailing matrix as shown in Figure 9.1e.

9.2 Background on Data Partitioning for LU Factorisation

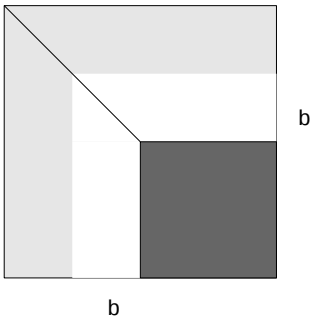
As with matrix multiplication several methods of data partitioning exist for LU factorisation, each with benefits and drawbacks. For both one and two-dimensional partition shapes, the computational load must remain balanced while minimising communication.

9.2.1 One-Dimensional Partitioning

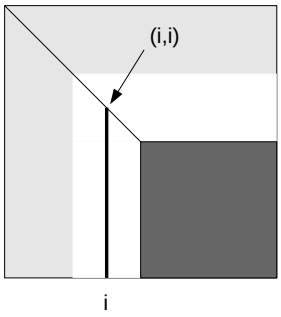
The simplest partition is the one-dimension shape. Like the one-dimensional matrix multiplication shapes, the LU shape can either be row or column oriented. However, due to the communication and computation needs of the panel factorisation stage, column oriented is more commonly used [13].

In order to balance computation, each processor receives multiple columns, in round robin or cyclic fashion, unlike the one-dimensional matrix multiplication partition which assigns one column per processor. The sum of all the widths of all the columns assigned to a given processor is in proportion to the speed of that processor. A one-dimensional partition shape is shown in Figure 9.2. This type of shape is an example of the *generalised block* partition discussed in [47] and [26]. The generalised block, static one-dimensional, partition is also used in [13] and [69].

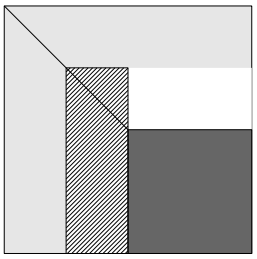
[1] http://cs.berkeley.edu/~demmel/cs267_spr14



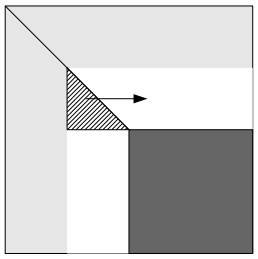
a) $k = 4$ steps



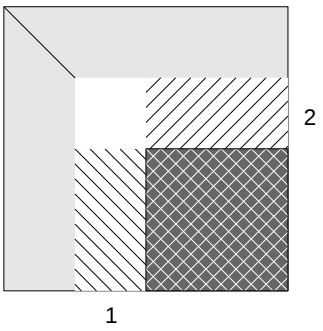
b) Search column for pivot and swap



c) Compute multipliers and subtract



d) Send LL horizontally and do triangular solve



e) Matrix Multiply (Submatrix = Submatrix - (Matrx 1 * Matrix 2)

Figure 9.1: LU factorisation parallel algorithm. White is the current step, light grey the completed factorisation, and dark grey the trailing submatrix to be updated/factorised.

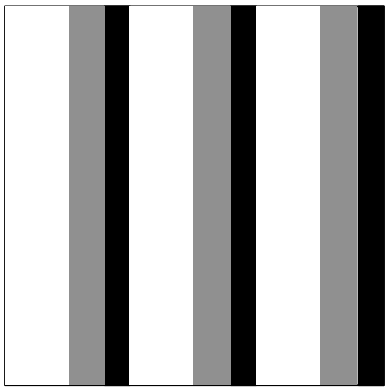


Figure 9.2: A one dimensional, column oriented, partition shape for LU factorisation on three heterogeneous processors (shown in white, grey, and black).

In addition to the generalised block, two other algorithms for one-dimensional LU partitions exist. First is the Dynamic Programming [45, 70] method, which distributes column panels to processors for execution. This algorithm returns an optimal solution, but does not scale well for non-constant performance models. The second is the Reverse Algorithm [71, 72]. This algorithm also provides an optimal solution, at somewhat worse complexity, with the benefit of an easier extension to non-constant performance models of processor speed.

9.2.2 Two-Dimensional Partitioning

Two-dimensional partitioning for LU factorisation is not common among two and three processor systems, due to the considerations of computational load. However, the same two-dimensional partitions used for four and more processors could be used to partition three heterogeneous processors.

A common one of these is the two-dimensional cyclic layout as shown in Figure 9.3.

9.3 Applying Push to LU Factorisation

This section will show how the Push Technique may be applied to LU factorisation, specifically considering the issues of maintaining a balanced computational load while minimising communication. To use the Push under these constraints involves a modified methodology, which is also described. Finally,

1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4

Figure 9.3: A two dimensional, cyclic, partition shape for LU factorisation on four processors.

the Push Technique is used to find some two processor optimal candidates for LU factorisation.

9.3.1 Special Considerations of LU Push

Data partitioning for LU factorisation is fundamentally different from matrix multiplication due to the serialised nature of the computation, as described in Section 9.1.2. As the computation progresses, the factorisation is finalised on an increasingly large portion of the matrix. Therefore, if a processor is assigned a square of the matrix in the top left hand corner of size $r \times r$ (as with the Square Corner for matrix multiply), then after the first r steps that processor is idle for the remainder of the factorisation.

Data partitions for LU factorisation must evenly distribute the computation of each step, and of each trailing sub matrix. As a large portion of the computation and communication load takes places in the update to the trailing sub matrix (in the form of matrix multiplication), this is the area to which the Push must be applied. Consideration must be given to both maintaining the proper distribution of computational load within each trailing sub matrix (and thereby also in each step), and to minimising communication that occurs at each step.

9.3.2 Methodology of LU Push

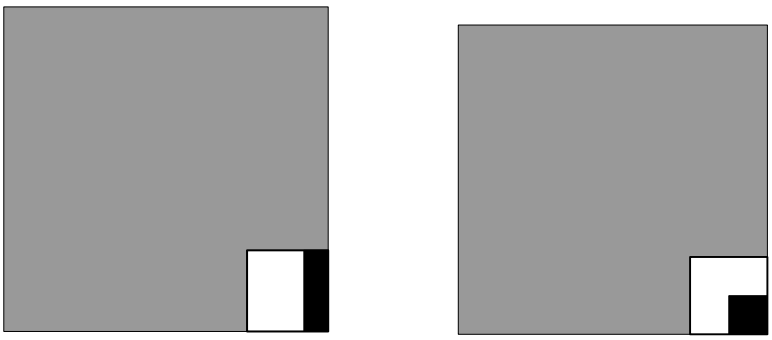
The two processor Push on a matrix partition shape for LU factorisation must accomplish two things,

- Maintain the correct ratio of data volume for each processor, in each

trailing sub matrix

- Minimise necessary communication between preceding steps and the current trailing sub matrix

Begin by considering the smallest trailing matrix, with elements randomly distributed to two Processors in proper proportion, and execute the Push Technique. The boundaries of the Push are the edges of the submatrix; no elements may move through this hard boundary. The simplest results to predict are those of the Push Technique for matrix multiplication on two processors, Straight Line and Square Corner, as shown in Figure 9.4.



a) Straight Line b) Square Corner

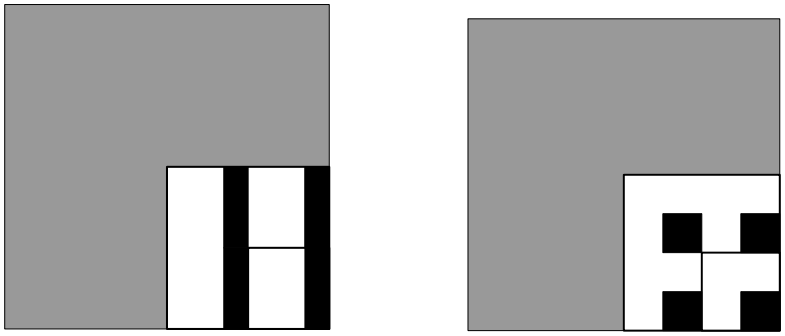
Figure 9.4: Perform Push on the smallest submatrix, to obtain partition shape for step k, partitioned between two processors (white and black). Grey represents the unPushed area of the matrix which has not yet been addressed.

This process continues considering the next smallest trailing submatrix, and Pushing the previously unPushed elements within it. The aim is to minimise the number of rows and columns containing elements of both processors. This could look like the partitions in Figure 9.5.

It is important to note that in order for the partition to remain computationally balanced, elements may *not* be Pushed across the boundaries of a step. Each step, and each trailing submatrix, must contain the proportional number of elements belonging to a given processor.

9.3.3 Two Processor Optimal Candidates

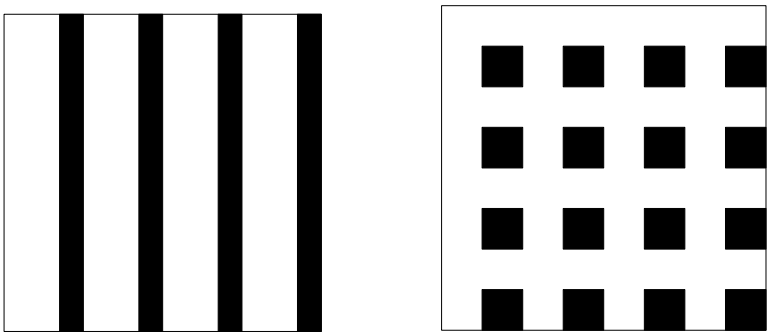
This section is *not* intended to display all possible optimal candidates for two processor LU factorisation. It is possible to imagine many dimensions and



a) Straight Line b) Square Corner

Figure 9.5: Add the previous step to the submatrix, and perform Push on the new submatrix.

locations of the rectangle assigned to the smaller processor in the smallest trailing submatrix. This rectangle will affect the ending partitions which are possible to make. Further investigation is required before other rectangle dimensions and locations can be eliminated. However, the two partitions shown in Figure 9.6 are certainly among the candidates, and are based on the Straight Line and Square Corner partitions, respectively.



a) Straight Line b) Square Corner

Figure 9.6: Some potential candidates for the optimal LU factorisation shape with two heterogeneous processors, depicted between a fast (white) processor and slow (black) processor.

Chapter 10

Further Work

While the work of the previous chapters has made significant progress determining optimal data partitioning shapes for matrix computations, there are a variety of ways this work could be further expanded. First, the most obvious addition to this work is to increase the number of processors to which it is applicable. A general roadmap to using the Push Technique on four and more processors has been provided, however, this section further expands on the possibilities and challenges in this future work.

The concept of the abstract processor was strictly defined in a simple manner, to provide a base case that might be applicable to the largest number systems possible. By increasing the number of metrics used to describe an abstract processor in terms of computation, communication, and memory, the boundaries at which each candidate becomes the optimal shape could be further refined for particular system types.

Finally, the Push Technique itself is a powerful tool that, with some changes to the performance model of the optimisation metric, can be applied to matrix computations in general.

10.1 Arbitrary Number of Processors

This thesis does not exhaust the number of processors for which the Push Technique is a useful tool. Using the Push DFA to implement an extension to the software tool for four and more processors is underway. However, it is not expected that the Push Technique be a feasible tool beyond some number of processors, perhaps eight or ten.

To consider the question of an arbitrary number of processors, a formal method for using the Push Technique in a hierarchical way would need to be developed. This method would describe the interaction of multiple layers of

processing power, and how a Push at one level in the hierarchy would effect the optimal partition shape at other levels. For example, if at the highest level several clusters were used (and due to their computational speeds) the Square Corner was chosen to distribute the data amongst them, any change in their underlying computational speed (due to change in the optimal data partition at a lower, node level) could affect the distribution at the higher level. This process could be defined in fashion similar to that of the LU factorisation Push, which starts at the base level and works up to some complete partition shape.

This leads naturally to the question of how would non-rectangular partitions be divided at the lower levels. The Push Technique should be extended for use on non-square matrices, so a non-rectangular matrix could potentially be considered two or more non-square rectangular matrices.

10.2 Increasing Complexity of Abstract Processor

There are a variety of ways the abstract processor model could be extended to increase the sensitivity of the model to diverse types of heterogeneous processors and networks.

10.2.1 Non-Symmetric Network Topology

The communication interconnect among processors can be widely varied in actual technology (Ethernet, InfiniBand), and in latency and bandwidth as a result of overall network congestion. Throughout this work, it has been assumed that communication is symmetric and can be represented by a single constant bandwidth, β. In future work, non-symmetric links could be represented using a unique constant for each link (including a constant of zero for non-existent links). If p processors are connected by $n\frac{n-1}{2}$ links, then there exist $n(n-1)$ bandwidth variables labelled $p_{i\rightarrow j}$ for communication travelling from Processor p_i to Processor p_j.

As with the optimal rectangle tiling problem, the optimal distribution of data on a heterogeneous network has also been shown to have cases of NP-completeness [73].

10.2.2 Computational Performance Model

The abstract processor is currently modelled computationally as a single integer number, expressed in the form of a ratio in proportion to the total

power of the system. However, this constant performance model could be replaced with a more complex functional performance model, based on the overall problem size (matrix size). A functional model would account for the changes that occur, as the matrix size grows, in the performance of different processors. While the addition of a functional performance model would not change the candidate shapes, it could alter the boundaries at which each candidate is the optimum.

Other options for growth in computational modelling include further parameterising of the basic unit of computation, such as is necessary for other matrix computations.

10.2.3 Memory Model

Related to the computational model, as it is often the reason for varying computational performance, is the memory model. It is well known that different processor types possess significantly different amounts of memory, especially in comparison to their respective throughput in FLOPs.

10.3 Push Technique and Other Matrix Computations

The Push Technique has been thoroughly investigated for matrix multiplication, and a methodology for use with LU factorisation has been given. However, other matrix computations pose some difficult challenges.

The key idea will be to identify those performance metrics of a given matrix computation which can be incrementally altered to achieve an optimal ending shape. Some possibilities for extension lie in other common linear algebra kernels in ScaLAPACK such as QR factorisation and other decomposition problems.

However, it is worth mentioning that the work done in this thesis with matrix multiplication could give insights into any matrix computation which uses matrix multiply in its parallel algorithm for the BLAS-3 speedup (as LU factorisation does).

Chapter 11

Conclusions

This thesis has outlined a solution to a particularly thorny problem - the optimal data partitioning for matrix computation for heterogeneous processors. Candidates for the optimal shape were found using the new Push Technique, which included unconventional non-rectangular shapes. Several of these shapes are indeed optimal for certain system characteristics in regards to ratios of computational power and communication speed.

11.1 The Push Technique

The Push Technique is a simple but powerful method of obtaining partition shapes. Using small, incremental changes to a data partition (each of which improves the partition metrics of communication time) the end result of the Push DFA is a finite set of candidate partition shapes. These candidate shapes may be compared directly, according to their performance models and algorithms, to determine which of the candidates is indeed the optimum.

The Push has several distinct advantages over typical methods for finding the partition shapes. The Push Technique:

- finds all candidate shapes, especially unexpected or novel shapes (such as Square Rectangle)
- ensures that candidate shapes are potentially optimally (as opposed to sub-optimal by definition)
- needs only to be thoroughly used once for each number of processors (to find all possible candidates)
- complexity of the DFA scales well with problem size

11.2 Non-Rectangular Partitioning

A fundamental assumption of much of the prior study of data partitioning for matrix computation on heterogeneous processors is that the partition shape should be rectangular. A rectangular partition is composed of rectangles (of any dimension, but of fixed area relative to their speed) assigned to each processor; no processor may have a non-rectangular portion of the matrix to compute. This problem, of optimally tiling rectangles of varying dimensions onto a unit square to minimise communication, is provably NP-complete [28]. The NP-complete result has driven the search for data partitions away from the realm of optimality. Instead, most work in this area has focused on various algorithms and heuristics to approximate a solution, which is always assumed to be rectangular.

The results of this thesis show that these approximate rectangular solutions are provably suboptimal for a variety of heterogeneous computational power ratios, under several different matrix multiplication algorithms. The larger goal of this thesis is to provide a framework for finding the optimal partition shape under myriad conditions, without assumptions about the rectangular/non-rectangular nature of those shapes.

11.2.1 The Square Corner Shape

The Square Corner shape, which was first described in [52] and [53], inspired this new search for optimality in data partitioning. For certain types of heterogeneous systems, this shape minimises the sum of the half perimeters, a metric of the total volume of communication required for matrix multiplication. In Figure 11.1, the Square Corner is shown for two and three processors, and is drawn to show example ratios for which it is optimal.

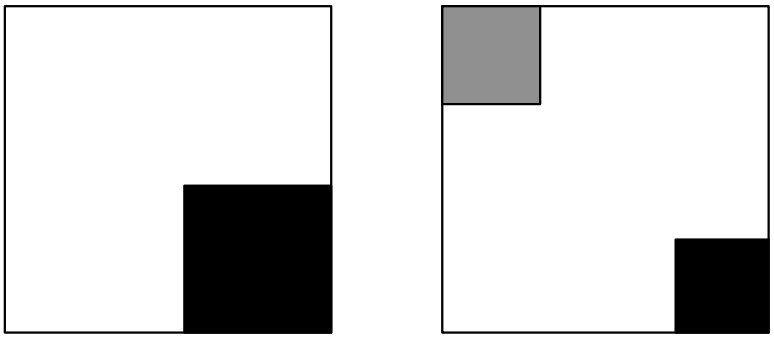


Figure 11.1: Examples of the Square Corner shape for two and three processors. Drawn to approximate ratios for which it is optimal.

In general, the Square Corner shape is optimal for two abstract processors when the ratio between the computational power of the two processors is greater than 3 : 1. However, the arrangement of elements in the Square Corner shape allows for a large swath of elements of the faster Processor P to be computed without communication. Overlapping this computation with the communication increases the ratios for which the Square Corner is optimal to ratios greater than 2 : 1.

For three abstract processor systems, the Square Corner shape is generally optimal when there is a single fast processor, and two slower processors. As with two processors, it is possible for a part of Processor P to be computed in parallel with communication, which increases the range for which the Square Corner is optimal.

11.2.2 The Square Rectangle Shape

The Square Rectangle shape is a completely new shape introduced in this thesis. The Square Rectangle was discovered using the Push Technique, and despite its unconventional non-rectangular appearance, it is optimal for a variety of heterogeneous systems. Figure 11.2 shows the Square Rectangle shape drawn to approximate ratios for which it is the optimum.

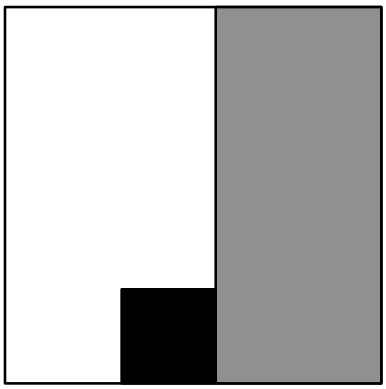


Figure 11.2: The Square Rectangle shape for three processors. Each processor is drawn to approximate ratios for which it is optimal.

In general, the Square Rectangle is optimal for three abstract processors when the ratios of computational power are such that there exists two fast processors and one slow processor. The Square Rectangle can be seen as a combination of the best attributes of both of the two processor shapes, Square Corner and Straight Line. For two relatively equal, nearly homogeneous, processors the Straight Line is optimal. Added to this is a small third

processor, in the shape of a square to minimise communication with the rest of the matrix.

11.3 Pushing the Boundaries: Wider Applicability

While Chapter 10 outlined the limitations of this thesis in terms of number of processors used (two or three rather than many) and abstract processor model (basic rather than complex), the overall theme of this thesis is one of generality. The Push Technique, incrementally improving a partition shape according to some metric, was applied to both matrix multiplication and LU factorisation with good results. To increase the complexity of the abstract processor would not change this fundamental result.

Similarly, the three processor case shows the wider applicability of the Push Technique to any arbitrary number of processors. It appears that for any reasonable number of processors, say less than ten, the Push Technique could create candidate partitions for sufficiently large enough problem sizes (to allow greater granularity in element movement). Furthermore, the candidates created using the Push Technique can be utilised in a hierarchical way, so as to minimise communication for any number of processors.

Appendix A

Two Processor Push Algorithmic Description

This section contains the algorithm description for the Push Up, Push Over, and Push Back, operations.

A.1 Push Down

Formally, Push $\downarrow(\phi, k) = \phi_1$ where,

Initialise $\phi_1 \leftarrow \phi$
$(g, h) = (k + 1, x_{left})$
for $j = x_{left} \rightarrow x_{right}$ **do**
{If element belongs Processor X, reassign it.}
if $\phi(k, j) == 1$ **then**
$\phi_1(k, j) = 0$;
$(g, h) =$ find(g, h); {Function defined below (finds new location).}
$\phi_1(g, h) = 1$; {Assign new location to active processor.}
end if
$j \leftarrow j + 1$;
end for

find(g, h):

for $g \rightarrow x_{bottom}$ **do**
for $h \rightarrow x_{right}$ **do**
{If potential location belongs to other processor, hasn't been reassigned already, and is in a column already containing X.}
if $\phi(g, h) == 0$ && $\phi_1(g, h) == 0$ && $c(\phi, X, h) == 1$ **then**
return (g, h);
end if

end for
$g \leftarrow g + 1$;
$h \leftarrow x_{left}$;
end for
return $\phi_1 = \phi$ {Could not find location, Push not possible in this direction.}

It is important to note that if no suitable $\phi(g, h)$ 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 $\downarrow(\phi, k) = \phi$.

A.2 Push Up

The Push Up operation is the opposite of the Push Down. Elements are reassigned from the bottom edge of the enclosing rectangle to the rows above.

Formally, Push $\uparrow(\phi, k) = \phi_1$ where,

Initialise $\phi_1 \leftarrow \phi$
$(g, h) = (k - 1, x_{left})$
for $j = x_{left} \rightarrow x_{right}$ **do**
  if $\phi(k, j) == 1$ **then**
    $\phi_1(k, j) = 0$;
    $(g, h) =$ find(g, h);
    $\phi_1(g, h) = 1$;
  end if
  $j \leftarrow j + 1$;
end for
find(g, h)
for $g \rightarrow x_{top}$ **do**
  for $h \rightarrow x_{right}$ **do**
    if $\phi(g, h) == 0$ && $\phi_1(g, h) == 0$ && $c(\phi, X, h) == 1$ **then**
      return (g, h);
    end if
  end for
  $g \leftarrow g - 1$;
  $h \leftarrow x_{left}$;
end for
return $\phi_1 = \phi$

A.3 Push Over

The Push Over operation reassigns elements from the left edge of the enclosing rectangle to the columns to the right.

Formally, Push $\rightarrow (\phi, k) = \phi_1$ where,

Initialise $\phi_1 \leftarrow \phi$
$(g, h) = (x_{left}, k + 1)$
for $i = x_{top} \rightarrow x_{bottom}$ **do**
  if $\phi(i, k) == 1$ **then**
    $\phi_1(i, k) = 0$;
    $(g, h) =$ find(g, h);
    $\phi_1(g, h) = 1$;
  end if
  $i \leftarrow i + 1$;
end for
find(g, h)
for $g \rightarrow x_{bottom}$ **do**
  for $h \rightarrow x_{right}$ **do**
    if $\phi(g, h) == 0$ && $\phi_1(g, h) == 0$ && $r(\phi, X, h) == 1$ **then**
      return (g, h);
    end if
  end for
  $g \leftarrow x_{top}$;
  $h \leftarrow h + 1$;
end for
return $\phi_1 = \phi$

A.4 Push Back

The Push Back operation, the opposite of the Push Over, reassigns elements from the right edge of the enclosing rectangle to the columns to the left.

Formally, Push $\leftarrow (\phi, k) = \phi_1$ where,

Initialise $\phi_1 \leftarrow \phi$
$(g, h) = (x_{top}, k - 1)$
for $i = x_{top} \rightarrow x_{bottom}$ **do**
  if $\phi(i, k) == 1$ **then**
    $\phi_1(i, k) = 0$;
    $(g, h) =$ find(g, h);
    $\phi_1(g, h) = 1$;
  end if

```
  i ← i + 1;
end for
find(g, h)
for g → x_bottom do
  for h → x_left do
    if φ(g, h) == 0 && φ_1(g, h) == 0 && r(φ, X, h) == 1 then
      return (g, h);
    end if
  end for
  g ← x_top;
  h ← h − 1;
end for
return φ_1 = φ
```

Appendix B

Push Lowers Communication Time: Two Processor Proofs

This appendix provides the full proofs for each of the five algorithms. These show that the Push Technique, when applied repeatedly, lowers the communication volume of the partition shape. These results were previously given in technical report [74].

B.1 Serial Communication

Theorem B.1.1 (Push)**.** *The Push Technique output partition, ϕ_1, will have lower, or at worst equal, communication time as the input partition, ϕ.*

Proof. First, observe several axioms related to the Push Technique.

Axiom 5. *Push $\downarrow$ and Push $\uparrow$, create a row, k, with no elements belonging to the Pushed Processor X, and may introduce Processor X to at most one row in ϕ_1 in which there were no elements of Processor X in ϕ. No more than one row can have elements of X introduced, as a row that was had no elements of X in ϕ will have enough suitable slots for all elements moved from the single row, k.*

Axiom 6. *Push $\downarrow$ and Push $\uparrow$ are defined to not add elements of Processor X to a column in ϕ_1 if there is no elements of X in that column of ϕ. However, these Push directions may create additional column(s) without X, if the row k being Pushed contains elements that are the only elements of X in their column, and there are sufficient suitable slots in other columns.*

Axiom 7. *Push $\rightarrow$ and Push $\leftarrow$ create a column, k, with no elements belonging to Processor X, and may create at most one column with X in ϕ_1 that did not contain X in ϕ.*

Axiom 8. *Push→ and Push← will never add elements of X to a row in ϕ_1 that did not contain elements of X in ϕ, but may create additional row(s) without X.*

From (3.8) we observe as ($\| \phi \|_x + \| \phi \|_y$) increases, T_{comm} decreases.
Push $\downarrow$ or Push $\uparrow$ on ϕ create ϕ_1 such that:

For row k being pushed,

If there exists some row i that did *not* have elements of X, but now does:

$r(\phi, X, i) = 0$ and $r(\phi_1, X, i) = 1$

then by Axiom 1:

$$\| \phi_1 \|_x = \| \phi \|_x$$

else

$$\| \phi_1 \|_x = \| \phi \|_x + 1$$

and by Axiom 2:

$$\| \phi_1 \|_y \geq \| \phi \|_y$$

Push $\rightarrow$ or Push $\leftarrow$ on ϕ create ϕ_1 such that:

For column k being pushed,

If there exists some column j that did *not* have elements of X, but now does:,

$c(\phi, X, j) = 0$ and $c(\phi_1, X, j) = 1$

then by Axiom 3:

$$\| \phi_1 \|_y = \| \phi \|_y$$

else

$$\| \phi_1 \|_y = \| \phi \|_y + 1$$

and by Axiom 4:

$$\| \phi_1 \|_x \geq \| \phi \|_x$$

By these definitions of all Push operations we observe that for any Push operation, ($\| \phi_1 \|_x + \| \phi_1 \|_y$) $\geq$ ($\| \phi \|_x + \| \phi \|_y$). Therefore, we conclude that all Push operations will either decrease communication time (3.8) or leave it unchanged. □

B.2 Parallel Communication

Theorem B.2.1 (Push)**.** *The Push Technique output partition, ϕ_1, will have lower, or at worst equal, communication volume to the input partition, ϕ.*

Proof. From the definition of the maximum function of execution time, and the definition of T_{comm}, for the PCB algorithm, the following observations may be made:

1. T_{comm} decreases as the volume of v_P decreases, if $v_P > v_S$
2. T_{comm} remains constant as VP decreases if $VQ > VP$
3. T_{comm} remains constant as VQ increases if $VP > VQ$

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

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

push $\uparrow$ and $\downarrow$ of ϕ are defined to create ϕ_1 such that:

For some k, $r(\phi, k) = 1$ and $r(\phi_1, k) = 0$

By axiom 1,

If there exists some i such that $r(\phi, i) = 0$ and $r(\phi_1, i) = 1$ then

$$\| \phi_1 \|_x^0 = \| \phi \|_x^0$$

Else,

$$\| \phi_1 \|_x^0 = \| \phi \|_x^0 + 1$$

By axiom 2,

$$\| \phi_1 \|_y^0 \geq \| \phi \|_y^0$$

push $\rightarrow$ and $\leftarrow$ of ϕ are defined to create ϕ_1 such that:

For some k, $c(\phi, k) = 1$ and $c(\phi_1, k) = 0$

By axiom 3,

If there exists some j such that $c(\phi, j) = 0$ and $c(\phi_1, j) = 1$ then

$$\| \phi_1 \|_y^0 = \| \phi \|_y^0$$

Else,

$$\| \phi_1 \|_y^0 = \| \phi \|_y^0 + 1$$

By axiom 4,

$$\| \phi_1 \|_x^0 \geq \| \phi \|_x^0$$

From these definitions we observe **Lemma 2:** For all push operations, $(\| \phi_1 \|_x^0 + \| \phi_1 \|_y^0) \geq (\| \phi \|_x^0 + \| \phi \|_y^0)$. By Lemmas 1 and 2, we conclude that VP decreases or remains constant for any push operation.

□

B.3 Canonical Forms

The location within the larger matrix of the rectangle assigned to Processor S does not affect the overall communication time because it does not change the number of rows and columns containing elements of both Processors P and S.

Theorem B.3.1. *Any partition shape, for two processor matrix multiplication, with an enclosing rectangle of Processor S of some dimensions x, y has the same communication cost.*

Proof. Consider a partition shape, q, divided between two Processors P and S. Processor S is of equal or lesser computational power of P and is assigned a rectangle of some dimensions x, y to compute. Processor P may be assigned a rectangular or non-rectangular portion to compute.

Volume of communication, the communication cost, is given by

$$\text{VoC} = \sum_{i=1}^{N} N(c_i - 1) + \sum_{j=1}^{N} N(c_j - 1) \tag{B.1}$$

$c_i -$ # of processors assigned elements in row i of q
$c_j -$ # of processors assigned elements in column j of q

Regardless of the location, the rectangle assigned to S may only occupy y rows and x columns. So all shapes with identical dimensions of Processor S are equivalent. □

Appendix C

Push Lowers Execution Time: Three Processor Proofs

This section contains the performance models of each of the five algorithms, specific to three processors. It is asserted that each of these models will decrease the execution time, or at least not increase it, for each of these algorithms. These results were previously published in technical report [75].

C.1 Serial Communication with Barrier

$$T_{exe} = T_{comm} + T_{comp} \tag{C.1}$$

$$T_{comm} = \Big(\sum_{i=1}^{N} N(p_i - 1) + \sum_{j=1}^{N} N(p_j - 1) \Big) \times T_{send} \tag{C.2}$$

where,

$$\begin{aligned} p_i &= \#\text{ of processors assigned elements in row } i \\ p_j &= \#\text{ of processors assigned elements in column } j \\ T_{send} &= \#\text{ of seconds to send one element of data} \end{aligned}$$

As Push is designed to clean a row or column of a given processor, it will decrease p_i and p_j, lowering communication time, and thereby execution time. At worst, it will leave p_i and p_j unchanged.

C.2 Parallel Communication with Barrier

$$T_{exe} = T_{comm} + T_{comp} \tag{C.3}$$
$$T_{comm} = \max(d_P, d_R, d_S) \tag{C.4}$$

Here, d_X is the time taken, in seconds, to send all data by Processor X and is formally defined below.

$$d_X = \big((N \times i_X + N \times j_X) - \#X\big) \times T_{send} \tag{C.5}$$

where,

$$\begin{aligned} i_X &= \#\text{ of rows containing elements of Processor } X \\ j_X &= \#\text{ of columns containing elements of Processor } X \\ \#X &= \#\text{ of elements assigned to Processor } X \end{aligned}$$

Each Push operation is guaranteed, by definition, to decrease or leave unchanged, either i_X or j_X, or both.

C.3 Serial Communication with Bulk Overlap

$$T_{exe} = \max\big(d_P + d_R + d_S, \max(o_P, o_R, o_S)\big) + \max(c_P, c_R, c_S) \tag{C.6}$$

where,

$$\begin{aligned} o_X =& \#\text{ of seconds to compute the overlapped computation} \\ & \text{on Processor } X \\ c_X =& \#\text{ of seconds to compute the remainder of the data on} \\ & \text{Processor } X \end{aligned}$$

Each Push operation is guaranteed, by definition, to decrease or leave unchanged, i_X and j_X of d_X for the active processor. It also, by definition, will not increase d_X for either inactive processor.

C.4 Parallel Communication with Overlap

$$T_{exe} = \max\Big(\max(d_X, d_R, d_S), \max(o_P, o_R, o_S)\Big) + \max(c_P, c_R, c_S) \quad \text{(C.7)}$$

Each Push operation is guaranteed, by definition, to decrease or leave unchanged, i_X and j_X of d_X for the active processor. It also, by definition, will not increase d_X for either inactive processor.

C.5 Parallel Interleaving Overlap

$$T_{exe} = \text{Send } k + \sum_{i,j,k=1}^{N} \max\Bigg(\Big(N \times \big((p_i - 1) + (p_j - 1)\big)\Big) \times T_{send}, \max(k_P, k_R, k_S)\Bigg) + \text{Compute}(k+1) \quad \text{(C.8)}$$

where,

$$\begin{aligned} p_i &= \#\text{ of processors assigned elements in row } i \\ p_j &= \#\text{ of processors assigned elements in column } j \\ k_X &= \#\text{ of seconds to compute step } k \text{ on Processor } X \end{aligned}$$

Again, Push is designed to clean a row or column of a given processor, it will decrease p_i and p_j, lowering communication time, and thereby execution time. At worst, it will leave p_i and p_j unchanged.

Appendix D

Three Processor Fully Connected Optimal Shape Proofs

D.1 Serial Communication with Overlap

In the Serial Communication with Bulk Overlap (SCO) algorithm, all data is sent by each processor *serially*, while in *parallel* any elements that can be computed without communication are computed. Only once both communication and overlapped computation are complete does the remainder of the computation begin. The execution time is given by,

$$T_{exe} = \max\Big(\max(T_{comm}, o_P)+c_P, \max(T_{comm}, o_R)+c_R, \max(T_{comm}, o_S)+c_S\Big)$$

where T_{comm} is the same as that of the SCB algorithm, o_X is the number of seconds taken by Processor X to compute any elements not requiring communication, and c_X is the number of seconds taken to compute the remainder of the elements assigned to Processor X.

D.1.1 Square Corner Description

Of the three candidate partitions, only the Square Corner has an o_X term which is not equal to zero, *i.e.* it contains elements which may be computed without any communication amongst processors. The overlap-able area may be seen Figure D.1. The addition of the non-zero o_P term implies that c_P will no longer be equal to c_R and c_S if we continue to naively assign the volume of elements as $\frac{N^2 P_r}{T}$. As Processor P is getting a head start on its

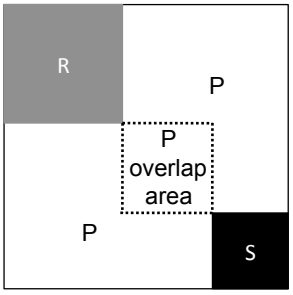


Figure D.1: The area of Processor P which does not require communication in the Square Corner partition shape is enclosed in dotted lines. The existence of this area is taken advantage of by the SCO and PCO algorithms.

computation, it should be assigned a larger portion of the matrix to compute than suggested by P_r.

Execution Time

To determine this optimal size, we first assume that the volumes (and thereby the size of the squares) assigned to Processors R and S should decrease in proportion to each other, so their computation times remain equal ($c_R = c_S$). The size of a side of the square R, r, and a side of the square S, s, being set at $s = \sqrt{\frac{r^2}{R_r}}$ ensures that computation on Processors R and S will complete at the same time. We may safely ignore the third term of the SCO max function, as it is redundant in this case. Execution time may now be given by,

$$T_{exe} = \max\Big(\max(T_{comm}, o_P) + c_P, T_{comm} + c_R\Big)$$

There are three possible functions that may dominate this execution time. Those are communication time and computation of P ($T_{comm} + c_P$), all computation of P ($o_P + c_P$) or communication time and computation of R, ($T_{comm} + c_R$). These functions are given by,

$$T_{exe(1)}(T_{comm} + c_P) = 2N^2\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right)\beta + \frac{2N}{Sp}\left(Nr - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{Nr}{\sqrt{R_r}} - \frac{r^2}{R_r}\right)$$

$$T_{exe(2)}(o_P + c_P) = \frac{N}{Sp}\left(N^2 - \frac{r^2}{R_r} - r^2\right)$$

$$T_{exe(3)}(T_{comm} + c_R) = 2N^2\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right)\beta + r^2\frac{N}{Sr}$$

where $\frac{Sp}{N}$ is the number of elements computed per second by Processor P, and $\frac{Sr}{N}$ is the number of elements computed per second by Processor R.

In order to make the execution time equations easier to analyse, the constant factor $N^3\beta$ has been removed. This introduces a new variable $c = Sp\beta$, which represents a ratio between computation and communication speeds. The size of N and r have been normalised, so that $\frac{r}{N}$ becomes r, and r is understood to be $0 \leq r < 1$.

$$\frac{T_{exe(1)}}{N^3\beta}(T_{comm} + c_P) = \frac{2}{N}\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right) + \frac{2}{c}\left(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}\right)$$

$$\frac{T_{exe(2)}}{N^3\beta}(o_P + c_P) = \frac{(1 - \frac{r^2}{R_r} - r^2)}{c}$$

$$\frac{T_{exe(3)}}{N^3\beta}(T_{comm} + c_R) = \frac{2}{N}\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right) + \frac{r^2P_r}{cR_r}$$

Lemma 1 (SCO $T_{exe(1)}$)**.** *The completion time of the communication and the computation of the remainder of Processor P's elements will dominate the maximum function and determine the execution time of the Square Corner partition shape for the SCO algorithm when* $T - R_r - \frac{2c}{N}(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}) + r(r + \frac{2r-2}{\sqrt{R_r}} + \frac{r}{R_r} - 2) < P_r < \frac{2R_r}{r} - 2R_r - 2\sqrt{R_r} + \frac{2\sqrt{R_r}}{r} - 2$.

Proof. Directly compare each of the T_{exe} functions, beginning with,

$$T_{exe(1)} > T_{exe(2)}$$

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

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

$$\frac{2c}{N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right)+2r-r^2-\frac{2r^2}{\sqrt{R_r}}+\frac{2r}{\sqrt{R_r}}-\frac{r^2}{R_r}-T > 1-P_r-R_r-S_r$$

$$T-R_r-\frac{2c}{N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right)+r\left(r+\frac{2r-2}{\sqrt{R_r}}+\frac{r}{R_r}-2\right) < P_r$$

And then comparing,

$$T_{exe(1)} > T_{exe(3)}$$

$$\frac{2}{N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right)+\frac{2}{c}\left(r-r^2-\frac{r^2}{\sqrt{R_r}}+\frac{r}{\sqrt{R_r}}-\frac{r^2}{R_r}\right) > \frac{2}{N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right)+\frac{r^2P_r}{cR_r}$$

$$\frac{2}{c}\left(r-r^2-\frac{r^2}{\sqrt{R_r}}+\frac{r}{\sqrt{R_r}}-\frac{r^2}{R_r}\right) > \frac{r^2P_r}{cR_r}$$

$$\frac{2R_r}{r}-2R_r-2\sqrt{R_r}+\frac{2\sqrt{R_r}}{r}-2 > P_r$$

□

Lemma 2 (SCO $T_{exe(2)}$)**.** *The computation of all elements assigned to Processor P will dominate the maximum function and determine the execution time of the Square Corner partition shape for the SCO algorithm when* $P_r < T-R_r-\frac{2c}{N}(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}})+r(r+\frac{2r-2}{\sqrt{R_r}}+\frac{r}{R_r}-2)$ *and when* $P_r < \frac{R_r}{r^2}-1-R_r-\frac{2cR_r}{r^2N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right)$.

Proof. The comparison of $T_{exe(1)}$ and $T_{exe(2)}$ may be found in the proof of Lemma 1. The second comparison is given by,

$$Texe(2) > T_{exe(3)}$$

$$\frac{(1-\frac{r^2}{R_r}-r^2)}{c} > \frac{2}{N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right)+\frac{r^2P_r}{cR_r}$$

$$1-\frac{r^2}{R_r}-r^2-\frac{2c}{N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right) > \frac{r^2P_r}{R_r}$$

$$\frac{R_r}{r^2}-1-R_r-\frac{2cR_r}{r^2N}\left(\sqrt{\frac{R_r}{T}}+\sqrt{\frac{1}{T}}\right) > P_r$$

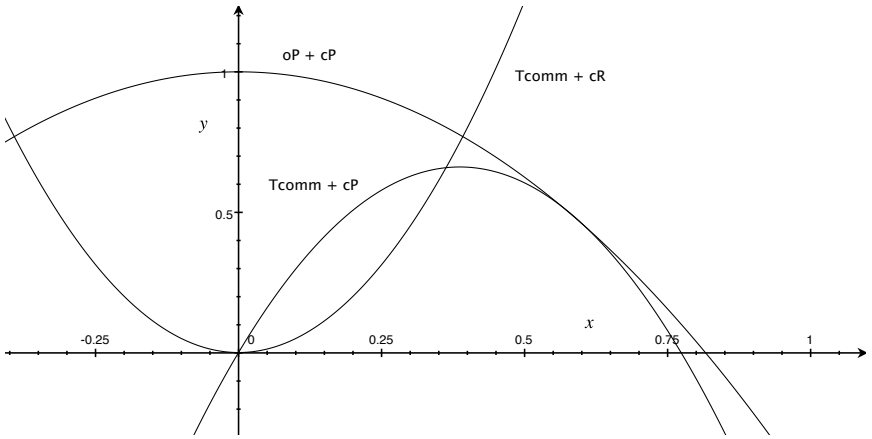


Figure D.2: Graph of the three constituent functions of execution time for the Square Corner shape under the SCO algorithm. N is normalised to 1, so valid values of x, which represents r, are $0 < x < 1$. As shown, $c = 1$, $R_r = 2$ and $P_r = 10$.

□

Lemma 3 (SCO $T_{exe(3)}$)**.** *The completion time of the communication and the computation of all elements assigned to Processor R will dominate the maximum function and determine the execution time of the Square Corner partition shape for the SCO algorithm when* $P_r < \frac{2R_r}{r} - 2R_r - 2\sqrt{R_r} + \frac{2\sqrt{R_r}}{r} - 2$ *and when* $P_r < \frac{R_r}{r^2} - 1 - R_r - \frac{2cR_r}{r^2N}\left(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}\right)$.

Proof. The comparison of $T_{exe(3)}$ and $T_{exe(1)}$, and the comparison of $T_{exe(3)}$ and $T_{exe(2)}$, may be found in Lemma 1 and Lemma 2, respectively. □

Optimal size of R and S

The optimal size of r depends on which function dominates the maximum function of the execution time. To determine the optimal sizes of r, we examine the graphs of these three functions. The graph is shown in Figure D.2. The functions are all parabolas, with the first two ($T_{comm} + c_P$ and $o_P + c_P$) are concave down while the third ($T_{comm} + c_R$) is concave up. If the size of r is zero, neither Processor R or S are assigned any part of the matrix to compute.

$$(1 \cap 2) \qquad r = \frac{R_r + \sqrt{R_r} - \sqrt{(\frac{2cR_r}{N})(1 + R_r + 2\sqrt{R_r})(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}})}}{1 + R_r + 2\sqrt{R_r}} \tag{D.1}$$

$$(2 \cap 3) \qquad r = \frac{\sqrt{-(\frac{P_r}{R_r} + 1 + \frac{1}{R_r})(\frac{2c}{N}\sqrt{\frac{R_r}{T}} + \frac{2c}{N}\sqrt{\frac{1}{T}} - 1)}}{(\frac{P_r}{R_r} + 1 + \frac{1}{R_r})} \tag{D.2}$$

$$(1 \cap 3) \qquad r = \frac{2 + \frac{2}{\sqrt{R_r}}}{\frac{P_r}{R_r} + \frac{2}{\sqrt{R_r}} + \frac{2}{R_r} + 2} \tag{D.3}$$

D.1.2 SCO Optimal Shape

Theorem D.1.1 (SCO Square Corner)**.** *The Square Corner partition shape minimizes execution time, i.e. is the optimum shape, when using the SCO MMM algorithm for computational ratios such that* $P_r > \frac{\frac{2}{N}(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}) + \frac{2}{c}(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{2}{N}}{\frac{1}{Tc} - \frac{1}{TN}}$ *and* $P_r > \frac{2c}{N}(\sqrt{R_r T} + \sqrt{T}) + 2T(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{Tc}{N} - \frac{2c}{N}\sqrt{T}$*, where* r *is the optimal size of the square* R*, given in (D.3).*

Proof. The Square Corner shape, as seen in graph 7.10, minimises execution time along the highly heterogeneous processor ratios, intersecting with the Block Rectangle and the Square Rectangle shape surfaces as the ratio approaches $P_r = R_r$. To compare the Square Corner shape, we first determine which constituent function of the max will dominate the execution time at those ratios. As heterogeneity decreases, and the intersections with Square Rectangle and Block Rectangle shapes are approached, it becomes increasingly vital to performance that powerful processors (R and S) not be left idle. In these cases, for the ratios at which each function of the maximum intersects with these other shapes, by Lemma 1, the maximum function will be dominated by the first function, $T_{exe(1)} = T_{comm} + c_P$.

It is now possible to compare each shape directly.

$$SC < BR$$

$$\frac{2}{N}(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}) + \frac{2}{c}(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) < \frac{2}{N} - \frac{P_r}{TN} + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right)$$

$$\frac{\frac{2}{N}(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}) + \frac{2}{c}(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{2}{N}}{\frac{1}{Tc} - \frac{1}{TN}} < P_r$$

$$SC < SR$$

$$\frac{2}{N}(\sqrt{\frac{R_r}{T}} + \sqrt{\frac{1}{T}}) + \frac{2}{c}(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) < \frac{1}{N} - \frac{2}{N}\sqrt{\frac{1}{T}} + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right)$$

$$\frac{2c}{N}(\sqrt{R_r T} + \sqrt{T}) + 2T(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{Tc}{N} - \frac{2c}{N}\sqrt{T} < P_r$$

□

Theorem D.1.2 (SCO Square Rectangle)**.** *The Square Rectangle partition shape minimises execution time, i.e. is the optimum shape, when using the SCO MMM algorithm for computational ratios such that* $P_r < T - 2\sqrt{T}$ *and* $P_r < \frac{2c}{N}(\sqrt{R_r T} + \sqrt{T}) + 2T(r - r^2 - \frac{r^2}{\sqrt{R_r}} + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}) - \frac{Tc}{N} - \frac{2c}{N}\sqrt{T}$

Proof. The ratios of intersection of the Square Rectangle and the Square Corner shapes is found in the proof of Theorem 7.3.11 above. The comparison of the Square Rectangle and Block Rectangles shapes gives,

$$SR < BR$$

$$\frac{1}{N} - \frac{2}{N}\sqrt{\frac{1}{T}} + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right) < \frac{2}{N} - \frac{P_r}{TN} + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right)$$

$$\frac{1}{N} - \frac{2}{N}\sqrt{\frac{1}{T}} < \frac{2}{N} - \frac{P_r}{TN}$$

$$P_r < T - 2\sqrt{T}$$

□

Corollary D.1.3 (SCO Block Rectangle)**.** *The Block Rectangle partition shape minimises execution time, i.e. is the optimum shape, for all processor computational power ratios except those specified in Theorems D.1.1 and D.1.2.*

D.2 Parallel Communication with Overlap

In the Parallel Communication with Bulk Overlap (PCO) algorithm all data is sent among processors in *parallel*, while in *parallel* any elements that can be computed without communication are computed. Once both communication and overlapped computation are complete, the remainder of the computation begins. The execution time for this algorithm is given by,

$$T_{exe} = \max\Big(\max(T_{comm}, o_P)+c_P, \max(T_{comm}, o_R)+c_R, \max(T_{comm}, o_S)+c_S\Big)$$

where T_{comm} is the same as that of the PCB algorithm, o_X is the number of seconds taken to compute any elements not requiring communication, by Processor X, and c_X is the number of seconds taken to compute the remainder of the elements assigned to Processor X.

D.2.1 Square Corner

In the PCO algorithm, the Square Corner shape has a portion assigned to Processor P, o_P, which may be overlapped with the communication. Processor P begins computation before Processors R and S, meaning that Processor P should be assigned more elements to compute. This will decrease the size of the squares assigned to Processor R and S, which we assert should be decreased in proportion to each other such that $c_R = c_S$. In this section, we determine the optimal size of square assigned to Processors R and S. We fix the size of the square of Processor S to be $s = \frac{r}{\sqrt{R_r}}$. The general form of the execution time equation will then be,

$$T_{exe} = \max\Big(\max(T_{comm}, o_P) + c_P, T_{comm} + c_R\Big)$$

Execution Time

There exist three functions which may dominate the execution time function for this partition shape, $T_{comm} + c_P$, $o_P + c_P$, and $T_{comm} + c_R$. These are given by,

$$
\begin{aligned}
T_{exe(1)}(T_{comm} + c_P) &= \max(2rN - 2r^2 + 2sN - 2s^2, 2r^2, 2s^2)\beta \\
&\quad + \frac{2r(N-r) + 2s(N-r-s)}{\frac{Sp}{N}} \\
T_{exe(2)}(o_P + c_P) &= \frac{N^2 - r^2 - s^2}{\frac{Sp}{N}} \\
T_{exe(3)}(T_{comm} + c_R) &= \max(2rN - 2r^2 + 2sN - 2s^2, 2r^2, 2s^2)\beta + \frac{r^2}{\frac{Sr}{N}}
\end{aligned}
$$

To simplify these equations, we substitute s, remove the constant $N^3\beta$ and normalize $\frac{r}{N}$ to r. As r and s are no longer independent variables, the third term of T_{comm} is redundant, and is ignored. The execution time functions may now be given by,

$$
\begin{aligned}
\frac{T_{exe(1)}}{N^3\beta}(T_{comm} + c_P) &= \frac{2}{N}\max(r - r^2 + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}, r^2) \\
&\quad + 2\frac{r - r^2 + \frac{r}{\sqrt{R_r}} - \frac{r^2}{\sqrt{R_r}} - \frac{r^2}{R_r}}{c} \\
\frac{T_{exe(2)}}{N^3\beta}(o_P + c_P) &= \frac{1 - r^2 - \frac{r^2}{R_r}}{c} \\
\frac{T_{exe(3)}}{N^3\beta}(T_{comm} + c_R) &= \frac{2}{N}\max(r - r^2 + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r}, r^2) + \frac{r^2 P_r}{cR_r}
\end{aligned}
$$

Value of T_{comm}

The T_{comm} function may be dominated by either the communication time of the data sent by Processor P or the data sent by Processor R.

Theorem D.2.1 (PCO SC T_{comm})**.** *The T_{comm} function of the Square Corner shape, under the PCO algorithm, will be dominated by the communication time of Processor P (the first term) when $P_r < T - 1 + \sqrt{R_r} - 2rR_r - r$, and by the communication time of Processor R otherwise*

Proof. Begin by stating,

$$\begin{aligned} V_P &> V_R \\ r - r^2 + \frac{r}{\sqrt{R_r}} - \frac{r^2}{R_r} &> r^2 \\ 1 - r + \frac{1}{\sqrt{R_r}} - \frac{r}{R_r} &> r \\ R_r + \sqrt{R_r} &> 2rR_r + r \\ T + \sqrt{R_r} - 1 + r(2R_r + 1) &< P_r \end{aligned}$$

□

Optimal Size of R and S

The optimal size of r is found by examining the graphs of the constituent functions $T_{exe(SC)}$. The minimum will be found at the intersection of the concave up and concave down parabolas, such that

$$r = \frac{-(\frac{2c}{N} + \frac{2c}{N\sqrt{R_r}}) + \sqrt{(\frac{2c}{N} + \frac{2c}{N\sqrt{R_r}})^2 + 4(\frac{P_r}{R_r} - \frac{2c}{NR_r} - \frac{2c}{N} + \frac{1}{R_r} + 1)}}{2(\frac{P_r}{R_r} - \frac{2c}{NR_r} - \frac{2c}{N} + \frac{1}{R_r} + 1)} \tag{D.4}$$

D.2.2 Square Rectangle

As with the SCO algorithm, the Square Rectangle shape does not have a portion which may be overlapped with communication under PCO. The time of execution, as with PCB model, is given by,

$$\begin{aligned} \frac{T_{exe(SR)}}{N^3\beta} = \max\left(\frac{1}{N} + \frac{2}{N\sqrt{T}} - \frac{R_r}{NT} - \frac{R_r}{NT\sqrt{T}} - \frac{3}{NT}, \frac{R_r}{NT} + \frac{R_r}{NT\sqrt{T}}, \frac{3}{NT}\right) \\ + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right) \end{aligned}$$

Value of T_{comm}

The T_{comm} function may be dominated by the communication time of the data sent by Processor P or S.

Theorem D.2.2 (PCO SR T_{comm})**.** *The T_{comm} function of the Rectangle Corner shape, under the PCO algorithm, will be dominated by the communication time of Processor P (the first term) when $P_r > 5 - 2\sqrt{T} + \frac{R_r}{\sqrt{T}}$, and by the communication time of Processor S otherwise*

Proof. First, the intersection of the equations of communication time for Processors P and R lies at $P_r = R_r + \frac{2R_r}{\sqrt{T}} - 2\sqrt{T} + 2$. However, this intersection may be show to lie outside the problem domain of $P_r \geq R_r$. Instead, we consider the intersection of Processor P and Processor S's communication time. Begin by stating,

$$\frac{1}{N} + \frac{2}{N\sqrt{T}} - \frac{R_r}{NT} - \frac{R_r}{NT\sqrt{T}} - \frac{3}{NT} > \frac{3}{NT}$$

$$P_r > 5 - 2\sqrt{T} + \frac{R_r}{\sqrt{T}}$$

□

D.2.3 Block Rectangle

$$\frac{T_{exe(BR)}}{N^3\beta} = \max\left(\frac{P_r}{NT}, \frac{2R_r}{NT}, \frac{2}{NT}\right) + \max\left(\frac{P_r}{Tc}, \frac{P_r}{Tc}, \frac{P_r}{Tc}\right)$$

Value of T_{comm}

The T_{comm} function may be dominated by either the communication time of the data sent by Processor P or the data sent by Processor R.

Theorem D.2.3 (PCO SC T_{comm})**.** *The T_{comm} function of the Block Rectangle shape, under the PCO algorithm, will be dominated by the communication time of Processor P (the first term) when $P_r > 2R_r$, and by the communication time of Processor R otherwise*

Proof. The communication time of Processor R will always be equal to or larger than the communication time of Processor S, $2R_r \geq 2$, as by definition $R_r \geq 1$. Therefore, when communication of Processor P does not dominate, that of Processor R will. □

D.2.4 PCO Optimal Shape

The optimal shape under the PCO algorithm depends on the value of c. When examining all three shapes to determine the optimal, we see that as c decreases, all three equations converge. However, for larger values of c, the Square Corner shape is optimal.

Theorem D.2.4 (PCO Square Rectangle)**.** *The Square Rectangle partition shape minimises execution time, i.e. is the optimum shape, when using the*

PCO MMM algorithm for computational ratios such that $P_r < \frac{1+\frac{2}{\sqrt{T}}-\frac{R_r}{T}-\frac{R_r}{T\sqrt{T}}-\frac{3}{T}-2r^2}{N(\frac{r^2}{cR_r}-\frac{1}{Tc})}$.

Proof. Examining the equations, we see that the Square Corner shape equation is dominated by the communication and computation of R when the Square Rectangle shape is dominated by the communication and computation of P.

$$\frac{2}{N}r^2 + \frac{r^2 P_r}{cR_r} < \frac{1}{N} + \frac{2}{N\sqrt{T}} - \frac{R_r}{NT} - \frac{R_r}{NT\sqrt{T}} - \frac{3}{NT} + \frac{P_r}{Tc}$$

$$P_r < \frac{1+\frac{2}{\sqrt{T}}-\frac{R_r}{T}-\frac{R_r}{T\sqrt{T}}-\frac{3}{T}-2r^2}{N(\frac{r^2}{cR_r}-\frac{1}{Tc})}$$

□

Corollary D.2.5 (PCO Square Corner)**.** *The Square Corner partition shape minimises execution time, i.e. is the optimum shape, for all processor computational power ratios except those specified in TheoremD.2.4.*

Bibliography

[1] T. G. Mattson, D. Scott, and S. Wheat, "A teraflop supercomputer in 1996: the asci tflop system," in *Parallel Processing Symposium, 1996., Proceedings of IPPS'96, The 10th International*, pp. 84–93, IEEE, 1996.

[2] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the petaflop era: the architecture and performance of roadrunner," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, p. 1, IEEE Press, 2008.

[3] J. J. Dongarra, H. W. Meuer, and E. Strohmaier, "Top500 supercomputer sites," *Supercomputer*, vol. 13, pp. 89–111, 1997.

[4] H. W. Meuer, "The top500 project. looking back over 15 years of supercomputing experience," *PIK-Praxis der Informationsverarbeitung und Kommunikation*, vol. 31, no. 2, pp. 122–132, 2008.

[5] E. Strohmaier, H. Simon, J. Dongarra, and M. Meuer, "Highlights of the 43^{rd} top500 list." http://www.top500.org/lists/2014/06/highlights/, June 2014.

[6] G. Shainer, T. Wilde, P. Lui, T. Liu, M. Kagan, M. Dubman, Y. Shahar, R. Graham, P. Shamis, and S. Poole, "The co-design architecture for exascale systems, a novel approach for scalable designs," *Computer Science-Research and Development*, vol. 28, no. 2-3, pp. 119–125, 2013.

[7] H. J. Siegel, L. Wang, V. P. Roychowdhury, and M. Tan, "Computing with heterogeneous parallel machines: advantages and challenges," in *Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on*, pp. 368–374, IEEE, 1996.

[8] C. J. Thompson, S. Hahn, and M. Oskin, "Using modern graphics architectures for general-purpose computing: a framework and analysis," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pp. 306–317, IEEE Computer Society Press, 2002.

[9] X. Xue, A. Cheryauka, and D. Tubbs, "Acceleration of fluoro-ct reconstruction for a mobile c-arm on gpu and fpga hardware: a simulation study," in *Medical Imaging*, pp. 61424L–61424L, International Society for Optics and Photonics, 2006.

[10] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, *et al.*, "Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu," in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 451–460, ACM, 2010.

[11] F. Song and J. Dongarra, "A scalable approach to solving dense linear algebra problems on hybrid cpu-gpu systems," *Concurrency and Computation: Practice and Experience*, pp. 2–35, 2014.

[12] F. Tinetti, A. Quijano, A. De Giusti, and E. Luque, "Heterogeneous networks of workstations and the parallel matrix multiplication," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 296–303, Springer, 2001.

[13] J. Barbosa, J. Tavares, and A. J. Padilha, "Linear algebra algorithms in a heterogeneous cluster of personal computers," in *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pp. 147–159, IEEE, 2000.

[14] Y. Wu, S. Hu, E. Borin, and C. Wang, "A hw/sw co-designed heterogeneous multi-core virtual machine for energy-efficient general purpose computing," in *Code Generation and Optimization (CGO), 2011 9th Annual IEEE/ACM International Symposium on*, pp. 236–245, IEEE, 2011.

[15] C. Van Berkel, "Multi-core for mobile phones," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1260–1265, European Design and Automation Association, 2009.

[16] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pp. 23–32, ACM, 2006.

[17] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in heterogeneous computing," *Scientific Programming*, vol. 18, no. 1, pp. 1–33, 2010.

[18] V. Boudet, F. Rastello, and Y. Robert, "Algorithmic issues for (distributed) heterogeneous computing platforms," in *In Rajkumar Buyya and Toni Cortes, editors, Cluster Computing Technologies, Environments, and Applications (CC-TEA'99). CSREA*, Citeseer, 1999.

[19] J.-N. Quintin, K. Hasanov, and A. Lastovetsky, "Hierarchical parallel matrix multiplication on large-scale distributed memory platforms," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, pp. 754–762, IEEE, 2013.

[20] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, "Fupermod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous hpc platforms," in *Parallel Computing Technologies*, pp. 182–196, Springer, 2013.

[21] L.-C. Canon, E. Jeannot, *et al.*, "Wrekavoc: a tool for emulating heterogeneity.," in *IPDPS*, 2006.

[22] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[23] F. Song and J. Dongarra, "A scalable framework for heterogeneous gpu-based clusters," in *Proceedinbgs of the 24th ACM symposium on Parallelism in algorithms and architectures*, pp. 91–100, ACM, 2012.

[24] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, *HPL - A Portable Implementation of the High-performance Linpack Benchmark For Distributed-memory Computers*. http://www.netlib.org/benchmark/hpl/, 2008.

[25] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, *et al.*, *ScaLAPACK users' guide*, vol. 4. siam, 1997.

[26] J. Dongarra and A. Lastovetsky, *High performance heterogeneous computing*, vol. 78. John Wiley & Sons, 2009.

[27] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, "Heterogeneity considered harmful to algorithm designers," in *2000 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 403–403, IEEE Computer Society, 2000.

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

[29] R. A. Van De Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency-Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[30] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 887–898, ACM, 2012.

[31] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[32] B. Grayson and R. Van De Geijn, "A high performance parallel strassen implementation," *Parallel Processing Letters*, vol. 6, no. 01, pp. 3–12, 1996.

[33] Y. Ohtaki, D. Takahashi, T. Boku, and M. Sato, "Parallel implementation of strassen's matrix multiplication algorithm for heterogeneous clusters," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 112, IEEE, 2004.

[34] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-optimal parallel algorithm for strassen's matrix multiplication," in *Proceedinbgs of the 24th ACM symposium on Parallelism in algorithms and architectures*, pp. 193–204, ACM, 2012.

[35] G. Strang, "Introduction to linear algebra," *Cambridge Publication*, 2009.

[36] L. E. Cannon, "A cellular computer to implement the kalman filter algorithm.," tech. rep., DTIC Document, 1969.

[37] G. C. Fox, S. W. Otto, and A. J. Hey, "Matrix algorithms on a hypercube i: Matrix multiplication," *Parallel computing*, vol. 4, no. 1, pp. 17–31, 1987.

[38] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM Journal of Research and Development*, vol. 39, no. 5, pp. 575–582, 1995.

[39] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5 d matrix multiplication and lu factorization algorithms," in *Euro-Par 2011 Parallel Processing*, pp. 90–109, Springer, 2011.

[40] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1017–1026, 2004.

[41] J. Choi, D. W. Walker, and J. J. Dongarra, "Pumma: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers," *Concurrency: Practice and Experience*, vol. 6, no. 7, pp. 543–570, 1994.

[42] M. Krishnan and J. Nieplocha, "Srumma: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 70, IEEE, 2004.

[43] 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 *Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on*, pp. 298–305, IEEE, 2001.

[44] A. L. Lastovetsky, "On grid-based matrix partitioning for heterogeneous processors.," in *ISPDC*, pp. 383–390, 2007.

[45] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A proposal for a heterogeneous cluster scalapack (dense linear solvers)," *Computers, IEEE Transactions on*, vol. 50, no. 10, pp. 1052–1070, 2001.

[46] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," in *High-Performance Computing and Networking*, pp. 189–200, Springer, 1999.

[47] 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, no. 4, pp. 520–535, 2001.

[48] A. Lastovetsky and R. Reddy, "Data partitioning with a functional performance model of heterogeneous processors," *International Journal of*

High Performance Computing Applications, vol. 21, no. 1, pp. 76–90, 2007.

[49] D. Clarke, A. Lastovetsky, and V. Rychkov, "Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models," in *Euro-Par 2011: Parallel Processing Workshops*, pp. 450–459, Springer, 2012.

[50] D. Clarke, *Design and Implementation of Parallel Algorithms for Modern Heterogeneous Platforms Based on the Functional Performance Model*. PhD thesis, University College Dublin, 2014.

[51] D. Clarke, A. Ilic, A. Lastovetsky, V. Rychkov, L. Sousa, and Z. Zhong, "Design and optimization of scientific applications for highly heterogeneous and hierarchical hpc platforms using functional computation performance models," *High-Performance Computing on Complex Environments*, pp. 235–260, 2013.

[52] B. A. Becker and A. Lastovetsky, "Matrix multiplication on two interconnected processors," in *Cluster Computing, 2006 IEEE International Conference on*, pp. 1–9, IEEE, 2006.

[53] B. A. Becker and A. Lastovetsky, "Towards data partitioning for parallel computing on three interconnected clusters," in *Parallel and Distributed Computing, 2007. ISPDC'07. Sixth International Symposium on*, pp. 39–39, IEEE, 2007.

[54] B. A. Becker, *High-level data partitioning for parallel computing on heterogeneous hierarchical computational platforms*. PhD thesis, University College Dublin, 2010.

[55] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on heterogeneous multicore platforms," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pp. 580–584, IEEE, 2011.

[56] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on heterogeneous multicore and multi-gpu systems using functional performance models of data-parallel applications," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pp. 191–199, IEEE, 2012.

[57] A. DeFlumere, A. Lastovetsky, and B. A. Becker, "Partitioning for parallel matrix-matrix multiplication with heterogeneous processors: The optimal solution," in *Parallel and Distributed Processing Symposium*

Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pp. 125–139, IEEE, 2012.

[58] A. DeFlumere and A. Lastovetsky, "Optimal data partitioning shape for matrix multiplication on three fully connected heterogeneous processors," in *Euro-Par 2014WS, HeteroPar 2014 - Twelfth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms*, 2014, in press.

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

[60] A. Lastovetsky and R. Reddy, "Data partitioning for multiprocessors with memory heterogeneity and memory constraints," *Scientific Programming*, vol. 13, no. 2, pp. 93–112, 2005.

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

[62] A. Marletta, "Cpulimit." https://github.com/opsengine/cpulimit, 2012.

[63] A. DeFlumere and A. Lastovetsky, "Searching for the optimal data partitioning shape for parallel matrix matrix multiplication on 3 heterogenous processors," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2014 IEEE 28th International*, IEEE Computer Society, 2014.

[64] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?," *Journal of computer and system sciences*, vol. 37, no. 1, pp. 79–100, 1988.

[65] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, "Optimization by simmulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[66] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, "Design and implementation of the scalapack lu, qr, and cholesky factorization routines," *Scientific Programming*, vol. 5, no. 3, pp. 173–184, 1996.

[67] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, no. 1, pp. 38–53, 2009.

[68] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley, "A proposal for a set of parallel basic linear algebra subprograms," in *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science*, pp. 107–114, Springer, 1996.

[69] J. Barbosa, C. Morais, and A. J. Padilha, "Simulation of data distribution strategies for lu factorization on heterogeneous machines," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pp. 8–pp, IEEE, 2003.

[70] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic issues on heterogeneous computing platforms," *Parallel processing letters*, vol. 9, no. 02, pp. 197–213, 1999.

[71] A. Lastovetsky and R. Reddy, "A novel algorithm of optimal matrix partitioning for parallel dense factorization on heterogeneous processors," in *Parallel Computing Technologies*, pp. 261–275, Springer, 2007.

[72] A. Lastovetsky and R. Reddy, "Data distribution for dense factorization on computers with memory heterogeneity," *Parallel Computing*, vol. 33, no. 12, pp. 757–779, 2007.

[73] O. Beaumont, A. Legrand, Y. Robert, *et al.*, "Data allocation strategies for dense linear algebra on two-dimensional grids with heterogeneous communication links," Tech. Rep. 4165, INRIA, 2001.

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

[75] A. DeFlumere and A. Lastovetsky, "Theoretical results for data partitioning for parallel matrix multiplication on three fully connected heterogeneous processors," Tech. Rep. UCD-CSI-2014-01, School of Computer Science and Informatics, University College Dublin, 2014.