High-Level Data Partitioning for Parallel Computing on Heterogeneous Hierarchical Computational Platforms

by

Brett A. Becker

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

Doctor of Philosophy
in
Computer Science

November 2010

College of Engineering, Mathematical and Physical Sciences
School of Computer Science and Informatics

Professor Joe Carthy, Ph.D. (Head of School)

Under the supervision of

Alexey Lastovetsky, Ph.D., Sc.D.

CONTENTS

DECLARATION

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.

Brett Arthur Becker,
Baile Átha Cliath, Éire, 23 Samhain, 2010
Dublin, Ireland, November 23, 2010

LIST OF TABLES

LIST OF FIGURES

NOTATION AND ABBREVIATIONS

$AlgorithmX_{y \rightarrow z}$	TVC for Algorithm X from partition y to partition z
DES	Discrete Event Simulation
HSCP-PC	Hybrid Square-Corner Paritioning using Parallel Communications
HSCP-SC	Hybrid Square-Corner Paritioning using Serial Communications
HSLP-PC	Hybrid Straight-Line Paritioning using Parallel Communications
HSLP-SC	Hybrid Straight-Line Paritioning using Serial Communications
LB	Lower Bound (Sum of Half Perimeters)
MDEM	Matrix Discrete Event Model
MMM	Matrix Matrix Multiplication
MPA	Max-Plus Algebra
N	Size of $N \times N$ matrix (number of rows and columns)
p	Number of partitions in a partitioning or number of computing entities (processors/clusters/etc.) mapped to partitions in a one-to-one mapping
ρ	When $p = 2$, the ratio of fastest entity to slowest entity, proportional to area of largest partition to smallest partition
$s_1, s_2, \ldots s_p$	Individual partition labels (or partition areas or speeds which are proportional to one another) of a particular partitioning, in non-increasing order
SCP	Square-Corner Partitioning
SHP ($\hat{C}$)	Sum of Half Perimeters

SLP	Straight-Line Partitioning
TVC	Total Volume of Communication
TVC_{AlgX}	Total Volume of Communication for Algorithm X

THEOREMS/PROPOSITIONS AND PROOFS

LIST OF EQUATIONS

PROBLEMS

ALGORITHMS

DEFINITIONS

Dedicated to Catherine, the love of my life... and more...

ABSTRACT

The current state and foreseeable future of high performance scientific computing (HPC) can be described in three words: heterogeneous, parallel and distributed. These three simple words have a great impact on the architecture and design of HPC platforms and the creation and execution of efficient algorithms and programs designed to run on them. As a result of the inherent heterogeneity, parallelism and distribution which promises to continue to pervade scientific computing in the coming years, the issue of data distribution and therefore data partitioning is unavoidable.

This data distribution and partitioning is due to the inherent parallelism of almost all scientific computing platforms. Cluster computing has become all but ubiquitous with the development of clusters of clusters and grids becoming increasingly popular. Even at a lower level, high performance symmetric multiprocessor (SMP) machines, General Purpose Graphical Processing Unit (GPGPU) computing, and multiprocessor parallel machines play an important role. At a very low level, multicore technology is now widespread, increasing in heterogeneity, and promises to be omnipresent in the near future.

Scientific computing is undergoing a paradigm shift like none other before. Only a decade ago most high performance scientific architectures were homogeneous in design and heterogeneity was seen as a difficult and somewhat limiting feature of some architectures. However this past decade has seen the rapid development of architectures designed not only to exploit heterogeneity but architectures *designed* to be heterogeneous. Grid and massively distributed computing has led the way on this front. The current shift is moving from these to architectures that are not heterogeneous by definition, but heterogeneous by necessity. Cloud and exascale computing architectures and platforms are not designed to be heterogeneous as much as they are heterogeneous *by definition*.

Indeed such architectures *cannot* be homogeneous on any large (and useful) scale. In fact more and more researchers see heterogeneity as the natural state of computing.

Further to hardware advances, scientific problems have become so large that the use of more than one of any of the above platforms in parallel has become necessary, if not unavoidable. Problems such as climatology and projects including the Large Hadron Collider necessitate the use of extreme-scale parallel platforms, often encompassing more than one geographically central supercomputer or cluster. Even at the core level large amounts of information must be shared efficiently.

One of the greatest difficulties in solving problems on such architectures is the distribution of data between the different components in a way that optimizes runtime. There have been numerous algorithms developed to do so over the years. Most seek to optimize runtime by reducing the total volume of communication between processing entities. Much research has been conducted to do so between distinct processors or nodes, less so between distributed clusters.

This thesis presents new data partitioning algorithms for matrix and linear algebra operations (these algorithms would in fact work for any application with similar communication patterns). In practice these partitionings distribute data between a small number of computing entities, each of which can have great computational power themselves. These partitionings may also be deployed in a hierarchical manner, which allows the flexibility to be employed in a great range of problem domains and computational platforms. These partitionings, in hybrid form, working together with more traditional partitionings, minimize the total volume of communication between entities in a manner proven to be optimal. This is done regardless of the power ratio that exists between the entities, thus minimizing execution time. There is also no restriction on the algorithms or methods employed on the clusters themselves locally, thus maximizing flexibility.

Finally, most heterogeneous algorithms and partitionings are designed by modifying existing homogeneous ones. With this in mind the ultimate goal of this thesis is to demonstrate that non-traditional and perhaps unintuitive algorithms and partitionings *designed with heterogeneity in mind from the start* can result in better, and in many cases optimal, algorithms and partitionings for heterogeneous platforms. The importance of this given the current outlook for, and trends in, the future of high performance scientific computing is obvious.

ACKNOWLEDGEMENTS

First I would like to most genuinely thank my advisor Dr. Alexey Lastovetsky whom I would like to nominate as the most capable, caring, and understanding supervisor ever to walk the face of planet Earth. I also owe many thanks to Prof. Joe Carthy, head of Computer Science and Informatics at UCD, for his help and understanding. Without Alexey and Joe, this thesis would not have ever reached fruition.

I would like to thank Dr. Olivier Beaumont and Prof. M. Tahar Kechadi for agreeing to be my external and internal examiners respectively and taking the time to read my thesis and attend my viva, especially Olivier for travelling from France to Dublin.

From the bottom of my heart I thank my mother Sharon—for everything. For understanding. For being there no matter what. For what we share. To say more would be tarnishing and dishonorable.

I also specially thank my father Art, and stepmother Joann for everything they have given me, taught me, and for believing in me. I am forever indebted and grateful. Thank you.

Thanks to my brother Wes for always being himself and keeping it real, and my sister-in-law Allison for the same. To my grandfather Art "Popa" who is the reason why I still scour parking lots for nails and screws so people don't get flat tires, and my grandmother Joan "Nana" for always reminding me who to look out for.

Thanks to my stepsisters Kristin and Andrea—I wish I saw you guys more.

Thanks to my Uncle Bob and Caroline for knowing what it's like and "what it is".

I would like to express a special thanks to my aunt Carol for too much to state, but especially for instilling the inspiration to travel in me, for without which I wouldn't be here writing this.

I would also like to thank all of the Beckers and the entire Finch Clan.

It goes without saying that I thank my best friend, confidant, comrade, brother in arms, and partner in crime, Patrolman Tommy McAnney. His acknowledgement is mentioned for completeness and not out of necessity. Without Tommy I would not be even a close semblance to my present self, for better or worse, and whom I hope continues to impact my life until the day I die. I wish him and Sandra the best on their new baby Noelle, my niece—no matter how far away I am.

I would also like to thank (in no particular order as always) my friends Carrie Hamblin, Donal Holohan, Dale Parsons, Eddie Pritsch, Marc Maggio, Tom Danza, Dave Marshall, Justin Smyth, Liam Power MSc. and his wife Gillian, Dr. John Regan and his wife Marylee, Dr. Mike Townsend and the soon to be Dr. Emma Howard, Dr. Mark "The Doctor" Dukes, and Dr. Fergus Toolan.

Thanks to UCD staff who have helped me along the way: Patricia Geoghegan, Clare Comerford, Paul Martin, Tony O'Gara and Gerry Dunnion. I would also like to thank Dr. Gianluca Polastri, Dr. Joe Kiniry, and Dr. Nicola Stokes for their support over the years.

A warm thanks to my UCD HCL colleagues Dr. Ravi Reddy, Dr. Thomas Brady, Dr. Michele Guidolin, Dr. Rob Higgins, Dr. Vladimir Rychov, Dr. Maureen O'Flynn, Dr. Xin Zuo, David Clarke and Kiril Dichev.

I am greatly appreciative for the help of Rob, Vladimir, Dave and Kiril for taking over the administration of my "baby", the HCL Cluster, and for initiating HCL Cluster 2.0—the first major upgrade since I built her.

A very special thanks to my GCD friends and colleagues: Eamonn Nolan, Dr. Waseem Akhtar, Orla Butler, Dr. Faheem Bukhatwa, Dr. Kevin Casey, Eoin Carroll, Ruairi Murphy, Dr. Kevin Hely, once more Dr. Fergus Toolan, and the rest of the GCD Computing Science lecturers and staff.

Thank you to the staff of the Drew University Department of Mathematics and Computer Science, Prof. Steve Kass and Prof. Christopher Van Wyk. And of course, the Drew University Physics Department, Prof. Robert Fenstermacher (Dr. F), Prof. James Supplee, and Prof. David McGee.

Thank you to the ever changing but constantly vigilant crew of the F/V Mary M. and many people in the microcosms that are the Atlantic City Surf Clam

Fleet, and the Great Bay/Tuckerton Bay/Little Egg Harbor Baymen.

Another very special thanks to Catherine's patient, understanding, and loving family: Dr. David and Margaret Mooney, Ruth and Darbs, Deborah and Angus, and their beautiful families.

I also would like to thank a very special group of people, some mentioned above—others not—who all went out of their way to help me in time of need. Thanks again—you know who you are.

Finally I would like to—and will do—more than thank Dr. Catherine Mooney for her love, trust, perseverance, faith, and not least of all for saving my life.

Sincerely yours, to all. - brett

Official Acknowledgements:

This work was supported by the University College Dublin School of Computer Science and Informatics.

This work was supported by Science Foundation Ireland.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

The author would like to thank Dr. Mark Dukes of the University of Iceland (at time of writing), now of the University of Strathclyde, for useful suggestions and emails (often not replied to) which always kept me going, and for getting me wrapped up in this Ph.D. business in the first place.

The author would like to thank Ms. Sharon Becker and Dr. Fergus Toolan for proofreading and Dr. Catherine Mooney for proofreading and many useful insights.

CHAPTER

ONE

INTRODUCTION

"I would never have got to know this remote and beautiful island otherwise."
– Erwin Schrödinger, November, 1960[1]

1.1 Motivation

Two general areas have provided the motivation for this work—those which are fundamental to this research, and the current state-of-the-art of high performance scientific computing. Areas which fall into the fundamental area include:

> What happens when we want to solve a well-established homogeneous problem on heterogeneous platforms?
>
> How can the performance of these problems be improved?
>
> Why have these problems, running on heterogeneous platforms, been largely ignored by research groups?
>
> Where is scientific computing headed in the future?

Platforms and architectures which are central to the state-of-the-art and future of scientific computing include:

- Super Computing
- Grid Computing

[1]Schrödinger, E. (1967). *What is life? With mind and matter and autobiographical sketches.* Cambridge University Press.

- Cloud Computing
- Cluster Computing
- GPGPU Computing
- Multicore Computing

1.1.1 Fundamentals

This work started with a simple question. Take two heterogeneous processing elements—how can they work together to best solve a specific problem? This question immediately raised many more. What happens if we add another element to make three? What about four? Is there something specific about the problems we want to solve that can be exploited to improve performance? How will the data partitioning and distribution impact the communication and execution times? How will the communication network affect the communication times? How will this affect execution times?

Regardless of the answers to these questions, two things are certain. It is desired for these elements to balance the computational load between themselves optimally, and to communicate data necessary for computations optimally. Unfortunately optimality is not always possible. These two tasks often turn out to be surprisingly difficult on heterogeneous platforms. Indeed solutions to problems that prove to be optimal on heterogeneous platforms are rare. Often some of the most simple tasks on homogeneous platforms turn out to be NP-Complete when attempted on heterogeneous ones (Beaumont *et al.*, 2002a). Sometimes approximation algorithms are found, often heuristics and problem restrictions are resorted to, and in some cases not even theoretical results exist. In the latter case researchers have deemed it necessary to resort to experimental approaches for reproducibility and comparison studies. Tools to facilitate such have already been developed (Canon and Jeannot, 2006).

To answer our questions we choose as a testbed the problem of matrix matrix multiplication (MMM). Matrices are probably the most widely used mathematical objects in scientific computing and their multiplication (or problems reducible to MMM) appear very frequently in all facets of scientific computing (Dongarra and Lastovetsky, 2006). Indeed, MMM is the prototype of tightly-coupled kernels with a high spatial locality that need to be implemented efficiently on distributed and heterogeneous platforms (Beaumont *et al.*, 2001b). Moreover most data partitioning studies mainly deal with matrix partitioning.

Why would we want to extend MMM to heterogeneous platforms? As stated in Beaumont *et al.* (2001b), the future of computing platforms is best described by the keywords *distributed* and *heterogeneous*.

Our fundamental motivation stems from two sources. First, there exist many general heterogeneous MMM algorithms which work well for several, dozens, hundreds, or even thousands of nodes, but *all currently known algorithms* result in simple, perhaps naïve partitionings when applied to the architecture of a small number of interconnected heterogeneous computing entities (two, three, etc.). Examples of these methods are explored in Beaumont *et al.* (2001b, 2002b); Dovolnov *et al.* (2003); Kalinov and Lastovetsky (1999); Lastovetsky (2007). As stated earlier we intentionally set out to investigate the particular case of a small number of computing entities to see what is happening in what is sometimes perceived to be a "degenerate" case. Despite its existence for at least 30 years, parallel MMM research has almost completely ignored this area. Early work is presented in Becker and Lastovetsky (2006, 2007), and some early application results in Becker and Lastovetsky (2010).

Second, we at the Heterogeneous Computing Laboratory[2] are keenly aware of the parallel, distributed and especially the heterogeneous nature of computing platforms which are omnipresent in scientific computing, and that most parallel and distributed algorithms in existence today are designed for, and only work efficiently on homogeneous platforms. After discussing the aforementioned load balancing and communication issues, we will survey modern scientific computing platforms, and where parallelism, distribution and heterogeneity impact them.

1.1.1.1 Load Balancing

The issue of load balancing is well studied and well understood, but not without its challenges. For a detailed study see Boulet *et al.* (1999). Neglecting the obvious such as the nature of the problem itself, failures and fluctuating capability due to other outside influences or factors, the issue can be reduced to a knowledge of the problem and the computing elements themselves. Suppose there is an amount of work W to do. If element A is capable of doing a work x in time t_1 and element B is capable of doing a work y in time t_2, the problem can be statically partitioned quite easily. We know that A works at a speed $s_1 = \frac{x}{t_1}$ and B works at a speed $s_2 = \frac{y}{t_2}$. If we normalize the speeds so

[2] `hcl.ucd.ie`

that $s_1 + s_2 = 1$, element A is to receive an amount of work equal to $W \times s_1$ and element B is to receive an amount of work equal to $W \times s_2$. Theoretically this would result in A and B finishing their work partitions in the same time, thus being optimal from a load balancing point of view.

For homogeneous computing elements such a problem scales well. In fact homogeneous systems are a standard platform for many supercomputers today (See 1.1.2). Current supercomputers utilize thousands of homogeneous elements (normally nodes or processors) working in parallel. At the time of writing the fourth fastest computer on Earth is "Kraken", a Cray CT5 Family, XT5-HE System Model, with 16,488 AMD x86_64 Opteron Six Core processors running at 2,600 MHz (a total of 98,928 cores) at the National Institute for Computational Sciences/University of Tennessee in Tennessee, USA.[3] Kraken is capable of 1,028,851 GFlops. For reference the laptop I am writing on now is an Intel Core Duo T5500 running at 1.66Ghz with 2GB memory, and has a peak performance of around 1Gflop, depending on the benchmark. An embarrassingly parallel problem—that is a problem that can be cut into any number of pieces as small as one wants with little or no communication between processes—could be theoretically balance load by partitioning the problem into 98,928 partitions and have each core solve one of the partitions. This would theoretically solve the whole problem in about one millionth of the time it would take my laptop. This is of course neglecting an multitude of factors such as data distribution and re-collection times, architectural differences, and vast memory and storage issues.

1.1.1.2 Communication

It is the communication aspects of data partitioning and distribution which make designing such algorithms difficult. Again ignoring faults, other non-related network traffic, etc., does the communication component of data partitioning affect the time it takes to solve the problem? How is it affected? what are these effects? Are there possibilities of deadlocks, race conditions and other parallel communication issues? Most fundamentally, two simple questions arise:

- How does the way we partition the data affect the execution time?
- What is the best way to partition the data so that we minimize the com-

[3]`www.nics.tennessee.edu/computing-resources/kraken`

munication time, thus (hopefully) minimizing the execution time?

These questions can be quite difficult to answer—sometimes impossible to answer—but can have a significant effect on the overall execution time.

1.1.2 State-of-the-art Scientific Computing

1.1.2.1 The Top500

The website `www.top500.org` maintains a list of the fastest computers on Earth, updated bi-annually. The fastest computer on Earth at the time of writing is "Jaguar", a Cray XT5-HE At Oak Ridge National Laboratory in Tennessee, USA.[4] Jaguar is composed of two physical partitions (not to be confused with data partitions). The first partition is "XT5" with 37,376 Opteron 2435 (Istanbul) processors running at 2.6GHz, with 16GB of DDR2-800 memory, and a SeaStar 2+ router with a peak bandwidth of 57.6Gb/s. The resulting partition contains 224,256 processing cores, 300TB of memory, and a peak performance of 2.3 petaflop/s (2.3 quadrillion floating point operations per second). The second partition "XT4" has 7,832 quad-core AMD Opteron 1354 (Budapest) processors running at 2.1 GHz, with 8 GB of DDR2-800 memory (some nodes use DDR2-667 memory), and a SeaStar2 router with a peak bandwidth of 45.6Gb/s. The resulting partition contains 31,328 processing cores, more than 62 TB of memory, over 600 TB of disk space, and a peak performance of 263 teraflop/s (263 trillion floating point operations per second). The routers are connected in a 3D torus topology for high bandwidth, low latency, and high scalability. The combined top500 benchmarked performance is 2,331,000 GFlops.

For interest, 2,331,000 GFlops is 2.27 times faster than Kraken, and significantly over two million times faster than the computer I am using at the moment.

What makes Jaguar different to Kraken? Jaguar is heterogeneous. Note that this is not necessarily the reason that Jaguar is faster, it is just a fact. Actually, the second and third fastest, along with the sixth and seventh fastest computers on Earth are heterogeneous. That makes half of the ten fastest computers on Earth heterogeneous. Jaguar is heterogeneous in processor architecture, speed, number of cores per processor, memory, storage, and network communications.

[4] `www.nccs.gov/computing-resources/jaguar/`

We have seen that heterogeneity has pervaded the area of supercomputers, however there are several other cutting-edge technologies emerging that are inherently heterogeneous.

1.1.2.2 Grid Computing

Grid Computing has become very popular for high performance scientific computing in recent years (Foster and Kesselman, 2004). Compared to stand-alone clusters and supercomputers, grids tend to be more loosely coupled, geographically dispersed, and are inherently heterogeneous. Unlike some clusters, grids tend to be built with general purpose scientific computing in mind. In short, grids seek to combine the power of multiple clusters and/or sites to solve problems. Grid computing has been sought and promoted by organizations such as CERN[5] to analyze the vast amounts of data that such bodies produce.

A primary advantage of grid computing is that each constituent cluster or site can be built from off-the-shelf commodity hardware that is cheaper to purchase, upgrade and maintain. Additionally there has been a major effort to produce middleware—software which makes the management of resources and jobs easier and cheaper than a custom solution. For an example, see SmartGridRPC, a project between the HCL and the University of Tennessee (Brady *et al.*, 2010). The primary disadvantage is the geographic distribution of sites which combined with commodity network hardware makes inter-site communication much slower than the often custom-built, very expensive networks of supercomputers.

An example of an existing grid is Grid'5000 (Bolze *et al.*, 2006). Located in France, Grid'5000 is composed of nine sites. Porto Alegre, Brazil has just become the official tenth site, and Luxembourg is expected to join soon. There is also a connection available which extends to Japan.

Grid'5000 has 1,529 nodes from Altix, Bull, Carri, Dell, HP, IBM and SUN. A total of 2,890 processors with a total of 5,946 cores from both AMD and Intel. Local network connections are Myrinet, Infiniband, and Ethernet. All Grid'5000 sites in France are connected with a 10Gb/s dark fibre link provided by RENATER (The French National Telecommunication Network for Technology Education and Research)[6]. Figure 1.1 Shows the backbones of the Renater

[5] `public.web.cern.ch/public/`
[6] `www.renater.fr`

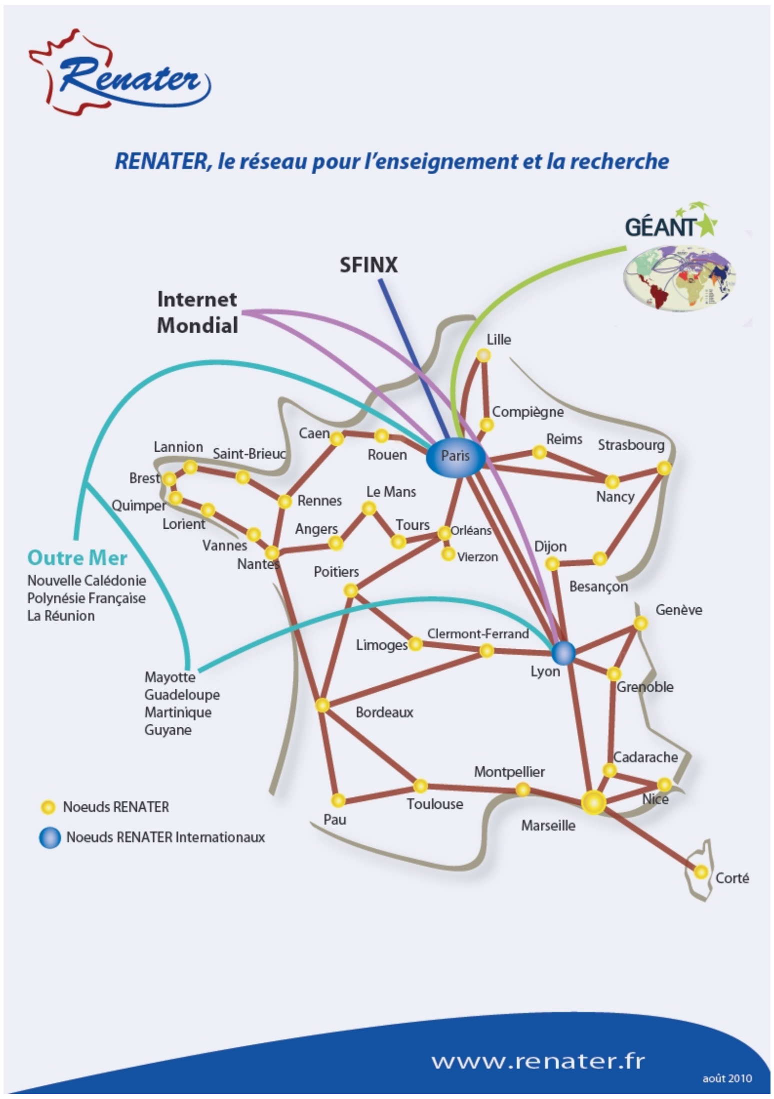


Figure 1.1: The Renater5 Network provides 10Gb/s dark fibre links that Grid'5000 utilizes for its inter-site communication network (Courtesy of www.renater.fr)

Network which connects the sites of Grid'5000. The important aspect of this figure is the architecture of the network connecting the various sites across France, and the connections to sites outside France.

In keeping with the decentralized nature of grid computing, Grid'5000 is funded by INRIA (The French National Institute for Research in Computer Science and Control)[7], CNRS (The French National Centre for Scientific Research)[8], the universities of all sites, and some regional councils. This highlights another advantage of grids—the cost of building and maintaining them can be shared amongst many different bodies easily.

[7] `www.inria.fr`
[8] `www.cnrs.fr`

A total of 606 experiments are listed on the Grid’5000 website as completed or in progress. A tiny sample of experiment areas include genetic algorithms, task scheduling, middleware testing, modeling physical phenomena, and linguistic processing. As an example of Grid’5000 performance, the Nancy site has a 92 node Intel Xeon cluster which achieves 7,360 GFlops, and a 120 node Intel Xeon cluster, which achieves 1,536 GFlops. As the Nancy site is average (actually a little lower than average) in size for Grid’5000, we can roughly calculate the power by dividing Nancy’s power by the number of nodes at Nancy then multiplying by the total number of nodes in Grid’5000. This roughly equals 65,000 Gflops, or 36 times slower than Jaguar. This of course is just a rough Gflop count, and does not take any specific parameters into account.

1.1.2.3 Cloud Computing

Cloud computing can have a different definition, depending on the source. Generally it is a form of computing where not only the details of who, what and where a user’s computations are being carried out are hidden from the user, but perhaps even the knowledge and details of how to calculate the computations. The general idea is that a user supplies data to a client program or interface, along with either a full program or program description, and the client program then selects the proper, available servers—which can be anywhere on the globe—and gets the work carried out. When the computation is complete, the results are delivered back to the user. For some applications where there are “canned” solutions available, all the user will have to do is supply the data and specify what solution is desired. In effect all the user needs to do is specify the problem and the solution will be delivered. In most definitions the “cloud” is a metaphor for the Internet, as one could view cloud computing as the computational (number crunching) equivalent of the Internet we know today. All the user knows is to open a web browser, supply information (what they’re looking for) and the results come back. The user doesn’t know from who, or where, and doesn’t care—it just comes.

1.1.2.4 Cluster Computing

In 1982 Sun Microsystems was founded upon the motto “The Network is the Computer”. This philosophy paved the way for the popularization of cluster computing, largely through their software products. At the time computer operating systems were designed only to run on and exploit the power of stand-alone computers. Sun's operating systems were revolutionary in that they were designed to harness the power of networks of computers.

Two or more such computers working together to achieve a common goal constitute a cluster. The topic itself, and much research focusing specifically on cluster computing as a pure subject is quite old, dating back 30 or more years. In fact such systems first started out as "the poor man's supercomputer" in academic departments where researchers would leave jobs running perhaps all night instead of purchasing expensive supercomputer time (Beaumont *et al.*, 2001b).

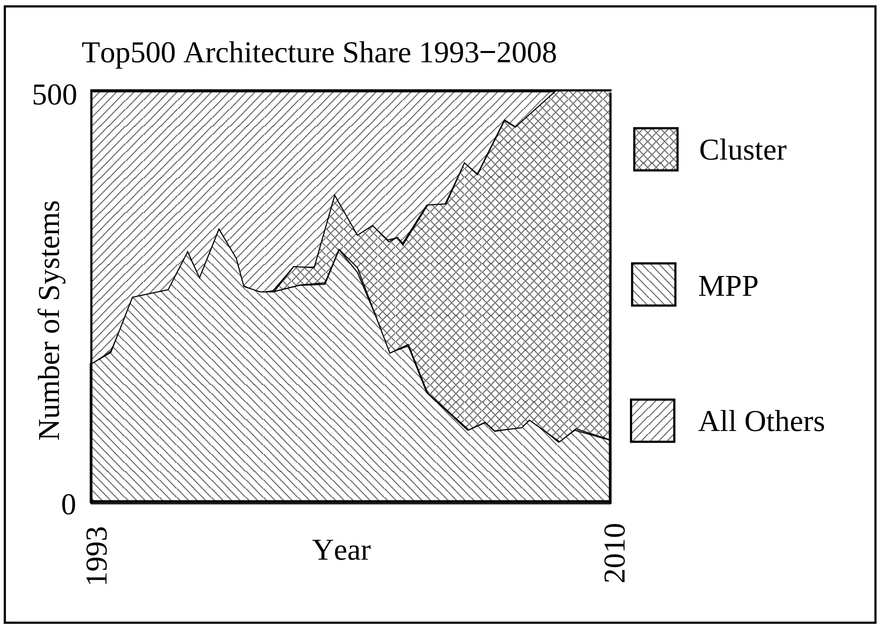


Figure 1.2: The architecture share of the top500 list from 1993 to 2010. (Courtesy of www.top500.org)

Cluster Computing has since become the dominant architecture for all scientific computing, including top500 supercomputers. Figure 1.2 shows the architectures of top500 computers from 1993 to 2010. In 1993 no top500 computers were clusters. They were MPPs, constellations, SMPs, and others—even single processor vector machines. It wasn't until the late 1990s that the first clusters joined the top500, but their popularity exploded, largely due to low cost and simple maintenance combined with great power. By 2007 about 80% of top500 machines were clusters and the number has grown to the point today where almost all top500 machines are clusters.

Let us demonstrate the prevalence and importance of clusters in the context of this section. Although not comprehensive in terms of state-of-the art scientific computing, this does provide a good overview:

- Top500 - Almost all computers in the top500 are based on cluster platforms
- Grid Computing - All grids are geographically distributed clusters or clusters of clusters.
- Cloud computing - As the name implies, how clusters fit in is slightly "fuzzy" but surely any cloud of even a moderate size would include clusters.
- GPGPU (General Purpose Graphical Processing Unit) computing is done on clusters of GPU machines.
- Multicore computing physically exists at the processor (single machine) level, but it is clusters of multicores which make up many top500 machines and grids.

Thus we have seen quite simply that cluster computing is actually the foundation of all other types of computing discussed here.

1.1.2.5 GPGPU (General Purpose Graphical Processing Unit) Computing

Another exciting area of high performance computing in which interest is gathering great pace is using Graphics Processing Units (GPUs) alongside traditional CPUs. Traditionally GPUs are used to take the burden of, and accelerate the performance of, rendering graphics (today often 3D graphics) to a display device. To this end, GPUs have evolved to become in most cases quite specialized in the operations necessary to do so, namely linear algebra operations. This makes them quite unintentionally well suited for many high performance scientific applications, as many of these rely heavily or exclusively on linear algebra operations. Examples of problems which have been explored with this approach include oil exploration, image processing and the pricing of stock options (Kruger and Westerman, 2005).

Beyond the confines of linear algebra, interest has also been gathering in so called General Purpose Computing on Graphics Processing Units or (GPGPU). This seeks to harness the computing power of GPUs to solve increasingly general problems. Recently nVidia and ATI (by far the two largest GPU manufacturers) have joined with Stanford University to build a dedicated GPU-based client for the Folding@home project which is one of the largest distributed computing projects in the world.

Briefly, Folding@home[9] harnesses (mostly) the unused CPU cycles of home computers across the globe to perform protein folding simulations and other molecular dynamics problems. A user downloads a client application and then when the user's computer is idle, packets of data from a server at Stanford are downloaded, and processed by the client program. Once the data has been processed using the client, the results are sent back to the server and the process repeated. At the time of writing the total number of active CPUs on the project is 286,723 with a total participation of 5,514,891 processing units, 343,843 of which are GPUs, and 1,003,463 are PlayStation 3s running the Cell Processor.[10] The total power of the Folding@home project is estimated to be 2,958,000 Gflops, theoretically 1.27 times faster than Jaguar. We must keep in mind however that if a problem with the complexity, memory, and data dependencies of those being solved on Jaguar was given to the Folding@home network, it would be incredibly—actually uselessly—slow, and very, very difficult to program.

Nonetheless, Folding@home is surely an example of extreme heterogeneity. Of course, mixed in those millions of computers are Linux, MAC, and Windows machines as well. The power of such a distributed, heterogeneous "system" can only be effectively harnessed due to the nature of the problems that are being solved. Although extremely large, the problems are embarrassingly parallel. In this case the key is that there are no data dependencies. No user computer needs information from, or needs to send information to, any other user computer. Further, the order in which data is sent back to the server does not matter. As long as all of the results eventually come back, they can be reconstructed back to the original order. If some results don't come back (which is inevitable), the data necessary to get the results are simply farmed out to another active user. Nonetheless we see a system with the power of a supercomputer, using a heterogeneous hierarchy at every level—client/server, system, processor and core.

For another similar project, see SETI@home[11], which distributes data from the Aricebo radio telescope in Puerto Rico to home users' computers, which then analyze the data for signs of extra-terrestrial life.

[9] `http://folding.stanford.edu`

[10] `fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats`

[11] `setiathome.ssl.berkeley.edu`

To wrap up GPU processing, nVidia has announced a new configuration using their video cards. Their PhysX physics engine can now be used on two heterogeneous nVidia GPUs in one machine.[12] A physics engine is software that computes and replicates the actual physics of events in real-time to make computer graphics more realistic such as shattering glass, trees bending in the wind, and flowing water. In this configuration the more powerful GPU renders graphics while the other is completely dedicated to running the PhysX engine.

1.1.2.6 Multicore Computing

At a much lower level, multicore technology has become mainstream for most computing platforms from home through high-performance. Multicore processors have more than one core, which is the element of a processor that performs the reading and executing of an instruction. Originally processors were designed with a single core, however a multicore processor can be considered to be a single integrated circuit with more than one core, and can thus execute more than one instruction at any given time. Embarrassingly parallel problems can approach a speedup equal to the number of cores, but a number of limiting factors including the problem itself normally limits such realization. Currently most multicore processors have two, four, six or eight cores. The number of cores possible is limited however, and is generally accepted to be in the dozens. More cores would require more sophisticated communication systems to implement and are referred to as many-core processors.

The Cell processor is a joint venture between Sony Corporation, Sony Computer Entertainment, IBM, and Toshiba and has nine cores. One core is referred to as the "Power Processor Element" or PPE, and acts as the controller of the other eight "Synergistic Processing Elements" or SPEs. See Figure 1.3 for a basic schematic of the processing elements of the Cell processor. The PPE can execute two instructions per clock cycle due to its multithreading capability. It has a 32KB instruction and 32KB L1 cache, and a 512KB L2 cache. The PPE performance is 6.2 GFlops at 3.2GHz. Each SPE has 256KB embedded SRAM and can support up to 4GB of local memory. Each SPE is capable of a theoretical 20.8 GFlops at 3.2GHz. Recently IBM has shown that the SPEs can reach 98% of their theoretical peak performance using optimized parallel matrix matrix multiplication.[13] The elements are connected by an Element Interconnect Bus

[12]www.nvidia.com/object/physx_faq.html\#q4

[13]www.ibm.com/developerworks/power/library/pa-cellperf/

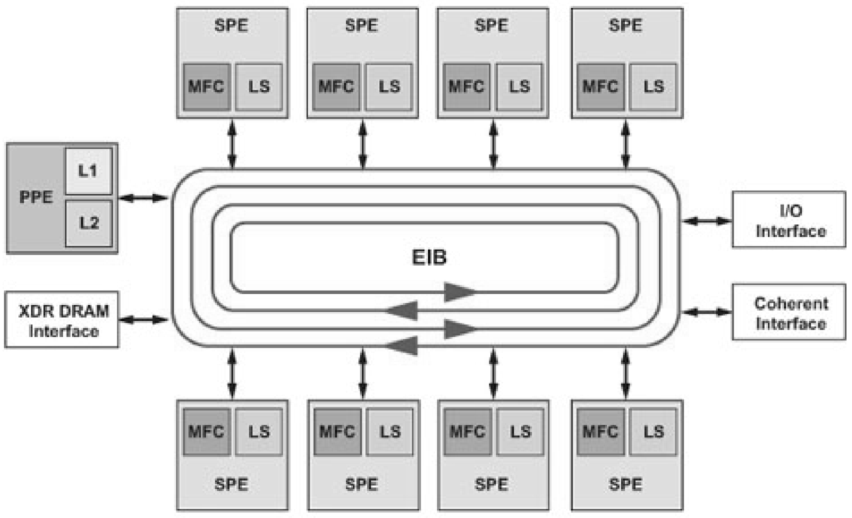


Figure 1.3: A basic schematic of the IBM Cell processor showing one Power Processing Element (PPE) and eight Synergistic Processing Elements (SPEs). (Figure from NASA High-End Computing, Courtesy of Mercury Computer Systems, Inc.)

(EIB), with a theoretical peak bandwidth of 204.8GB/s.

The Sony PlayStation 3 is an example of the Cell processor at work. To increase fabrication yields, Sony limited the number of operational SPEs to seven. One of the SPEs is reserved for operating system tasks, leaving the PPE and six SPEs for game programmers to use. Clearly this has utilized the Cell to create a more heterogeneous system. This is exemplary of a truly heterogeneous system in practice—functionality can be arranged as desired, and needed.

The Cell processor is used in the IBM "Roadrunner" supercomputer, which is a hybrid of AMD Opteron and Cell processors and is the third fastest computer on Earth (formerly number 1) at 13,752,776 GFlops. The PlayStation 3 "Gravity Grid" at the University of Massachusetts at Dartmouth Physics Department is a cluster of sixteen Playstation 3 consoles used to perform numerical simulations in the areas of black hole physics such as binary black hole coalescence using perturbation theory.[14]

Clearly the Cell processor is an example of parallel heterogeneous computing at a very low-level, with very diverse applications, and introduces a hierarchy with the PPE controlling the SPE's, while also maintaining some number crunching abilities itself.

The future of heterogeneous multicore architectures is expanding rapidly. The

[14]`arxiv.org/abs/1006.0663`

past month alone has seen two major developments. First, a research team at the University of Glasgow has announced what is effectively a 1000 core processor, although it differs from a traditional multicore chip as it is based on FPGA technology, which could easily lend itself to heterogeneous use. Second, the release of the first multicore mobile phones has been announced. The natural need for heterogeneity in such platforms is discussed in van Berkel (2009).

1.1.3 Heterogeneity

We have seen that heterogeneity and hierarchy have infiltrated every aspect of computing from supercomputers to GPUs, Cloud Computing to individual processors and cores. We have also seen that in many, many ways all of these technologies are interwoven and can join to form hybrid entities themselves.

To conclude it is fitting to state that homogeneity (even if explicitly designed) can be very difficult and expensive to maintain, and easy to break (Lastovetsky, 2003). Any distributed memory system will become heterogeneous if it allows several independent users to simultaneously run applications on the same system at the same time. In this case different processors will inevitably have different workloads at any given time and provide different performance levels at different times. The end result would be different performances for different runs of the same application.

Additionally, network usage and load, and therefore communication times will also be varied with the end result being different communication times for a given application, further interfering with the delivery of consistent performance for the same application being run more than once.

Component failure, aging, and replacement can all also impact homogeneity. Are identical replacement components available? Are they costly? Do all components deliver uniform and consistent performance with age? Even if these problems are managed, eventually when upgrading or replacement time comes, all upgrades and replacements must be made at the same time to maintain homogeneity.

We see now that heterogeneity is the *natural state* of parallel and distributed systems. Most interestingly, vendors are now starting to intentionally design and construct systems and platforms for high performance scientific computing which are heterogeneous from the outset. This is perhaps a natural progression as specialized hardware and software aimed at one particular class of problem is desired over more general-purpose approaches that yield poor performance.

1.2 Objectives

The main goal of this thesis is to present, validate, and experimentally demonstrate a new partitioning algorithm for high performance scientific computing on parallel hierarchal heterogeneous computing platforms. This partitioning could theoretically be deployed on any heterogeneous architecture including all those discussed in Section 1.1.2. It will also be shown that this partitioning can serve as the basis for other new partitionings. The following is a list of other goals that will and must be realized along the way. First the state-of-the-art will be reviewed, before the underlying mathematical principles of this new partitioning are explored. For the cases in which it applies the partitioning will be discussed and it's optimality proven. A hybrid algorithm will be discussed which is designed to be optimal in all cases for certain problem domains. The construction of a heterogeneous cluster hand-designed specifically for problems discussed in this thesis will be detailed. The partitioning will be compared to the state-of-the-art and both its benefits and deficits discussed. The partitioning will be modelled, simulated, and then experimentally verified before being shown to be beneficial to application areas indicative of those widely in use today. Then, future directions of research will be presented.

Finally, as stated, most heterogeneous algorithms and partitionings are designed by modifying existing homogeneous ones. The ultimate goal of this thesis is to demonstrate the concept that unintuitive, non-traditional algorithms and partitionings, designed with heterogeneity in mind from the start, can result in better—and optimal—algorithms and partitionings for high performance computing on heterogeneous platforms.

1.3 Outline

Chapter 2: Background and Related Work
In this section existing research in the area of heterogeneous parallel data partitioning and matrix matrix multiplication is explored. The state-of-the-art is clearly described and the benefits and drawbacks of current techniques are detailed.

Chapter 3: UCD Heterogeneous Computing Laboratory (HCL) Cluster
This chapter describes the design and construction of a heterogeneous cluster specifically for the simulation and testing of heterogeneous algorithms and partitionings. The cluster is unique in its ability to be configured in network parameters and topology which allows for testing on any number of heterogeneous scenarios.

Chapter 4: Partitioning a Matrix in Two – Geometric Approaches
This chapter presents and mathematically validates a new data partitioning method for matrix matrix multiplication on heterogeneous networks. A geometrical approach is used to present the design of the partitioning. The partitioning is shown to be optimal in most cases, and a hybrid partitioning is proposed which would be optimal in all cases.

Chapter 5: The Square-Corner Partitioning
This chapter defines the Square-Corner Partitioning and its application to matrix matrix multiplication. The optimality of the partitioning as well as other benefits such as overlapping computation and communication are explored. Experimental results are presented, first on two processors, then on small groups of clusters, then on two larger clusters.

Chapter 6: The Square-Corner Partitioning on Three Clusters
In this chapter the Square-Corner Partitioning is extended to three clusters. Both topologies possible (fully-connected and star) are explored. Experimental results are given for simulations on three processors and three clusters. Results of overlapping computation and communication are also explored.

Chapter 7: Max-Plus Algebra and Discrete Event Simulation on Parallel Hierarchal Heterogeneous Platforms
This chapter presents the results of applying the Square-Corner Partitioning on Max-Plus Algebra operations and a Discrete Event Simulation Application.

Chapter 8: Moving Ahead – Multiple Partitions and Rectangular Matrices
This chapter presents work on extending partitionings to more than three and to non-square matrices. The Square-Corner Partitioning is shown to be useful for the multiplication of rectangular matrices in particular.

Chapter 9: Conclusions and Future Work
In this chapter overall conclusions of this work are drawn, and indications of exciting areas of future work are detailed.

CHAPTER

TWO

BACKGROUND AND RELATED WORK

2.1 Data Distribution and Partitioning

In 1997, van de Geijn and Watts noted: "It seems somewhat strange to be writing a paper on parallel matrix multiplication almost two decades after commercial parallel systems first became available. One would think that by now we would be able to manage such an apparently straight forward task with simple, highly efficient implementations. Nonetheless, we appear to have gained a new insight into this problem" (van de Geijn and Watts, 1997).

It is now 2010 and researchers across the globe are still working with great ferocity on parallel matrix multiplication algorithms! This chapter presents a summary of parallel matrix multiplication, first and briefly on homogeneous algorithms, then heterogeneous algorithms. We start with the homogeneous case because most heterogeneous algorithms are modifications of their homogeneous counterparts. We will of course end with the current state-of-the-art in heterogeneous parallel MMM algorithms.

In order to discuss these algorithms however, we must discuss data distribution and therefore data partitioning, which is the focus of this thesis. The partitioning of data directly affects the communication between processing elements. These communications may be over a super-fast bus between cores or long-distance copper or fibre networks, but either way it is typically communications that are an order of magnitude or more slower than processing speed and therefore the most significant bottleneck in parallel and distributed algorithms.

2.2 Introduction

As stated in the summary, it is now 2010 and we are still working with great ferocity on parallel matrix multiplication algorithms. This is due to the common thread that ran through the motivation in Section 1.1.2, from cloud computing with theoretically millions or more machines at work, through supercomputers with hundreds of thousands of processors, down to multicore chips on single machines. This thread is *heterogeneity*. In fact there was another common thread, *distribution*, which is due to necessary physical separation of computing elements. Be they two computers on opposite sides of the Earth, or two cores separated by hundredths of a centimeter, computing elements need to be physically separated and therefore must communicate to work together.

This brings up an interesting question—Why must processing elements be physically separated? There are many answers to this question, the main reasons being:

(i) **Heat.** Too many processing elements in too small a space creates too much heat which increases cooling costs in the best case, and in the worst case leads to component failure.

(ii) **Cost.** Along with cooling cost, communication hardware costs are perhaps the largest cost factors. The ultra-fast communication buses between cores and processors and main-memory are simply too expensive to scale up. In fact such devices only barely scale out of the micro and into the macro scales. It is much, much cheaper to connect machines with commodity, off-the-shelf communication hardware, than two processors with an ultra-fast motherboard bus that is metres or more long.

(iii) **Convenience.** For economical, social and other reasons, it makes sense to distribute computing resources geographically. In the case of the Folding@home project introduced in the motivation, the utilized resources were already geographically distributed, and then exploited. We also saw that there are computing resources, particularly grids (for example Grid'5000 (Bolze *et al.*, 2006)), that are *intentionally* built to be geographically distributed. This is not done due to pure physical necessity, but to make cost, maintenance, upgrading and logistics more convenient.

(iv) **Modularity and Reliability.** Closely related to convenience is the modularity and reliability of the system itself. If a supercomputer was just a single, massive processor (if physically feasible), what would happen if it or a crucial component failed? Everything would grind to a halt. What happens if a chip, node, or even an entire cluster or even in the extreme an entire site in Grid'5000 goes down? All other processors, nodes, clusters, and sites go merrily on with their business. This gets even better if the middleware can handle fault tolerance. The user may not realize that a component failed, other than a possibly longer execution time, but not necessarily, and what counts most is the results will still be correct.

(v) **"That's just the way it is"** or **''That's just the way things evolve".** Everything from beings with exoskeletons to animals with internal skeletons, from dinghies to the most massive cargo ships, from subatomic particles to the most massive of stars have a physical size limit per entity. At some point the only way to generate more beings, carrying capacity, or energy, is to make *more* of them. And thanks to quantum physics no two things can occupy the same space at the same time, so more things means more space which necessitates physical distribution.

There are certainly more parameters, especially when individual cases are examined. Perhaps one more question could be asked, particularly in reference to (i) and (ii) above. Why can't a chip be manufactured which is just one big chip? Let's ignore heat, cost, reliability and other obvious answers and instead of answering the question directly just render the question itself a moot point. If it were possible to do so we would still be dealing with communications. Communications between individual registers or even in the extreme, transistors on the chip itself, still need to be done optimally or at least efficiently. Again, as in Section 1.1 we see that communication between different *entities* is an inherent fact of computing, no matter what scale we are dealing with and no matter how we look at or abstract the issue.

2.3 Parallel Computing for Matrix Matrix Multiplication

Currently parallel computing is undergoing a paradigm shift. It is spreading from supercomputer centers which have been established utilizing specialized, often custom-built and terribly expensive computers which are used and programmed by highly trained and specialized people to clusters of off-the-shelf commodity workstations that (with the proper libraries and some skill) can be used by "ordinary" people. By ordinary I mean people whose primary profession need not be programming supercomputers. Such clusters have already pervaded academia and industry, but promise to do so further, and are now becoming accessible to and "toys" of home users, who desire to use them. Indeed these clusters remain the poor man's supercomputer (Csikor *et al.*, 2000) as discussed in the motivation. Cloud Computing promises to carry this concept even further, with additional abstraction, and less expertise needed by the user.

From this point forward we will not consider heterogeneity (or homogeneity for that matter) and communication to be exclusive. From one follows the other. Any parallel architecture (homogeneous or heterogeneous) without *some* communication *somewhere* is useless. Let's just talk about parallel computing for a while. We will start with the simple case, homogeneous parallel computing.

In this thesis I exclusively address the linear algebra kernel of Matrix-Matrix Multiplication (MMM) as the prototype problem for High Performance Parallel Computing on Heterogeneous Networks (with the exception of some application areas explored later in Chapter 7). This is a common and justifiable decision as matrices are probably the most widely used mathematical objects in scientific computing (Lastovetsky, 2007). Further to that, MMM is the prototype for a group of tightly coupled kernels with a high special locality that should be implemented efficiently on parallel platforms [both homogeneous and heterogeneous] (Beaumont *et al.*, 2001b). Throughout this thesis, if only one matrix is being discussed, it may be thought of as the product C of two other matrices A and B, such that $C = A \times B$. In these cases, A and B are partitioned identically to C. This has become a standard in the field.

2.3.1 Data Partitioning for Matrix Matrix Multiplication on Homogeneous Networks

The problem of matrix partitioning on homogeneous networks has been well studied. It has perhaps been studied to exhaustion with the exception of the inclusion of application specific issues, or particular hardware/software considerations. In other words the theoretical underpinnings have been established and are very unlikely to undergo a significant change. For more see Kumar *et al.* (1994).

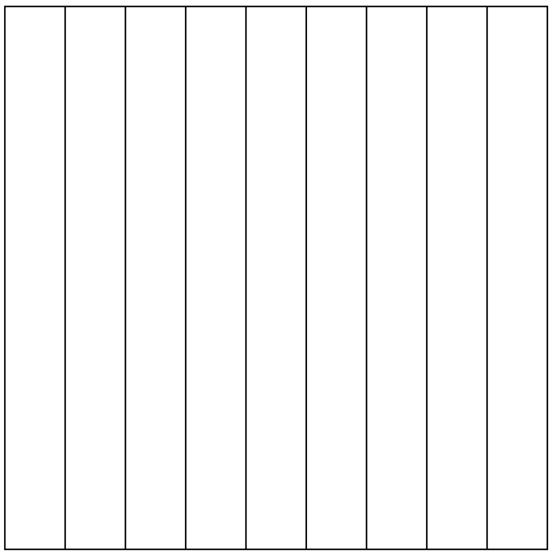


Figure 2.1: A one-dimensional homogeneous (column-based) partitioning of a matrix with a total of nine partitions.

Let us start then with homogeneous matrix partitioning for three reasons:

1. Because it is well established
2. Because most heterogeneous algorithms are designed either from, or at least with, their homogeneous counterparts in mind
3. It is often the homogeneous counterparts that heterogeneous algorithms are compared to, particularly to address their effectiveness or lack thereof (Lastovetsky and Reddy, 2004)

When partitioning a matrix for distribution between homogeneous processors[1], the problem of load balancing is easy, as all processors have equal speed, and therefore each partition will have equal area. Thus the issue quickly turns to minimizing communications. The simplest homogeneous partitioning of a matrix is in one dimension, with the matrix partitioned into either rows *or* columns of equal area as in Figure 2.1. The other way to accomplish a homogeneous partitioning is to use a two-dimensional partitioning, which results in a grid of partitions as in Figure 2.2.

[1]We begin by talking about partitioning and distribution among individual processors and later will scale this up to individual clusters. *Entity* will sometimes be used to refer to any unit capable of processing be it a core, processor, cluster, etc.

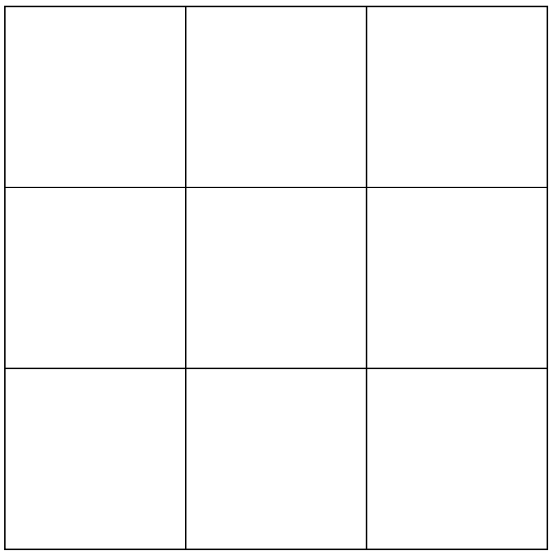


Figure 2.2: A two-dimensional homogeneous partitioning of a matrix with a total of nine partitions.

Let us start with the two-dimensional case. We have a matrix multiplication $C = A \times B$ and for simplicity we make each a square $n \times n$ matrix. Assume we have p homogeneous processors $P_1, P_2, \ldots, P_p$. Again for simplicity let us assume that the processors are arranged in a grid of size $p_1 \times p_2 = p$ such that $p_1 = p_2$. In this case the processors and the partitions of the matrix overlay each other exactly such that at each step k,

- Each processor $P_{i,k}$, $i \in \ldots p_1$ broadcasts horizontally $a_{i,k}$ to processors $P_{i,*}$
- Each processor $P_{k,j}$, $j \in \ldots p_2$ broadcasts vertically $b_{k,j}$ to processors $P_{*,j}$

This allows each processor $P_{i,j}$ to update its portion of C, using $c_{i,j} = c_{i,j} + a_{i,k} \times b_{k,j}$. In other words, at each step of the algorithm each processor does the following:

- Each processor broadcasts the part of the pivot column which it owns horizontally to the processors in the same processor row which the processor resides in
- Each processor broadcasts the part of the pivot row which it owns vertically to the processors in the same processor column which the processor resides in.

This is shown in Figure 2.3.

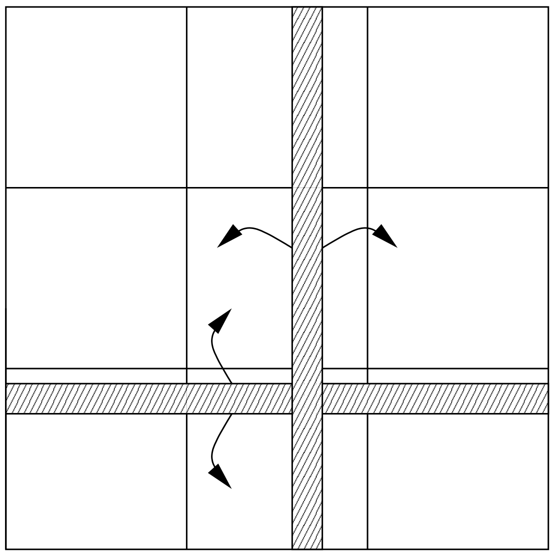


Figure 2.3: A two-dimensional homogeneous partitioning of a matrix with a total of nine partitions showing pivot rows and columns and the directions they are broadcast.

The popular ScaLAPACK library uses this technique, but uses a blocked version (Blackford *et al.*, 1997). In this case, each matrix element (say $C_{i,j}$, $A_{i,j}$, and $B_{i,j}$) is a square $r \times r$ block. For each processing entity there will be an optimal value of r, depending on memory and other architecture-specific details. Usually the number of blocks $\lceil (n/r) \rceil \times \lceil (n/r) \rceil$ is greater than the number of processors $p_1 \times p_2$. In this case the blocks are distributed in a cyclic fashion so that each processor is responsible for updating several blocks of the C matrix at each step k.

It is worth noting at this point that the total volume of communication (TVC) of the calculation described above is proportional to the sum of the half perimeters (SHP) of each partition. This can be viewed rather simply; at each step, each processor responsible for a square of $x \times x$ elements receives x elements from a row-mate and x elements from a column-mate for a total of $2x$ elements. Since the partitions are all equal in size and dimension (if $\sqrt{p}$ evenly divides n), the same will be true for all processors. Thus at each step each processor is receiving a number of elements proportional to $2 \times x$, the sum of each partition's half perimeter.

It is easy to show that the two-dimensional partitioning has a lower TVC than the one-dimensional partitioning. Given a fixed area, the rectangle covering that area and having the smallest perimeter is a rectangle that is square, and the fact that *any* other partitioning strategy would force at least one partition to be non-square, any other partitioning strategy (including the one-dimensional partitioning described above) would result in a greater SHP, and therefore a greater TVC.

There are other parallel algorithms for MMM, such as:

- Cannon's algorithm (Lee *et al.*, 1997)
- DNS (Kumar *et al.*, 1994)
- one-dimensional and two-dimensional Systolic (Golub and Van Loan, 1996)
- Broadcast Multiply Roll (Fox *et al.*, 1988)
- the Transpose Algorithm (Lin and Snyder, 1992)
- SUMMA (Scalable Universal Matrix Multiplication Algorithm (van de Geijn and Watts, 1997), which is used in PBLAS (the Parallel Basic Linear Algebra Subprograms) library (Choi *et al.*, 1996)

Each has its advantages and disadvantages and a discussion of each is beyond the scope of this thesis. They are mentioned for completeness.

2.3.2 Data Partitioning for Matrix Matrix Multiplication on Heterogeneous Networks

Now let us assume a heterogeneous network. We have p heterogeneous processors $P_1, P_2, \ldots, P_p$, each free to run at a speed unique to all other processors. We will express these speeds relatively as $s_1, s_2, \ldots, s_p$. Similar to the homogeneous algorithm above at each time step k, there will be a pivot row broadcast vertically and a pivot column broadcast horizontally. These rows and columns can be either made of individual matrix elements or blocks of elements as described above. If we are discussing blocks, they will remain square for efficiency on a processor-local level. We will now see how a modification of the homogeneous algorithm above can lead to heterogeneous ones.

The heterogeneity of processors means that we cannot necessarily partition the matrix into squares. We will generalize and partition the matrix into non-overlapping rectangles. Similar to the homogeneous algorithm, data should be partitioned so that the area of each rectangular partition is proportional to the speed of the processor who owns it. From a load-balancing point of view it is only the area of the rectangles that matter. The dimensions of each rectangle

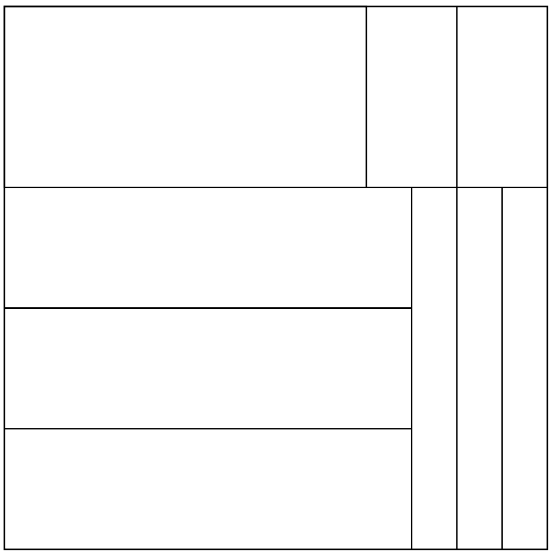


Figure 2.4: A two-dimensional heterogeneous partitioning consisting of nine partitions

are free for us to choose. Figure 2.4 shows a heterogeneous partitioning of nine rectangles.

The difficult question is this: What dimensions should each rectangle have to minimize the total inter-partition (and therefore inter-cluster, if each partition were owned by a cluster) volume of communication? Actually the word difficult turns out to be an understatement. This question (when formally defined) turns out to be NP-complete (Beaumont *et al.*, 2002b). There is no known polynomial time solution to this question. In other words, for even modestly sized problems, unreasonable (or impossible) lengths of time would be needed to achieve an optimal solution. We must therefore resort to approximation algorithms or heuristics. Generally these begin by imposing a restriction or restrictions on the proposed solution to allow it to run in polynomial time. The next three subsections will explore the state-of-the-art in attempts to do so.

Before we do it is worth mentioning (again for completeness) that we are and will be discussing *static* partitionings only. Static partitionings are determined based on input data, and after that the partitioning does not change throughout the course of the program. A *dynamic* partitioning is one that does (more properly has the ability to) change the initial partitioning before the program terminates. One way to do this is *use the past to predict the future* (Beaumont *et al.*, 2001b). This method determines how to best change the partitioning based on how the program is commencing during execution. There also exist dynamic master-slave techniques. These all may suffer from various drawbacks, not the least that they must be very general and therefore less likely to perform as well as static approaches which are tailored to a specific problem domain.

2.3.2.1 Restricting the General Approach: A Column Based Partitioning

In (Beaumont *et al.*, 2001b) the authors state the general matrix partitioning problem:

- Given p computing elements $P_1, P_2, \ldots, P_p$, the relative speed of which is characterized by a positive constant S_i, where $\sum_{i=1}^{p} S_i = 1$
- Partition a unit square into p rectangles so that
  - There is a one to one mapping of elements to rectangles
  - The area of the rectangle allocated to element P_i is equal to S_i where $i \in \{1, \ldots, p\}$.
  - The partitioning minimizes $\sum_{i=1}^{p}(w_i + h_i)$, where w_i is the width of the rectangle and h_i is the height of the rectangle assigned to element P_i.

Partitioning the unit square into rectangular partitions with areas proportional to the speed of the processing elements mapped to them is aimed at balancing the load of the work done by each processing element. As a rule there will be more than one partitioning satisfying this condition.

Minimizing the sum of half-perimeters of the partitions, $\sum_{i=1}^{p}(w_i + h_i)$ is aimed at minimizing the total volume of communication between the elements. This is possible because with the problem of MMM, at each step, each processing element that does not own the pivot row and column receives an amount of data proportional to the half-perimeter of the rectangular partition it owns. Therefore the amount of data communicated at each step between all processing elements will be proportional to the sum of the half-perimeters of all partitions, $\sum_{i=1}^{p}(w_i + h_i)$, less the sum of the heights of the partitions owning the pivot column, (one in the case of the unit square), and less the sum of the widths of the partitions owning the pivot row (also one for the unit square). Thus at each step the total data communicated will be $\sum_{i=1}^{p}(w_i + h_i) - 2$. Because the total amount of data communicated between the processing elements is the same at each step of the algorithm, the total amount of data communicated during the execution of the algorithm will also be proportional to $\sum_{i=1}^{p}(w_i + h_i) - 2$. Therefore minimization of $\sum_{i=1}^{p}(w_i + h_i) - 2$ will minimize the total volume of communication between all partitions. Clearly any solution minimizing $\sum_{i=1}^{p}(w_i + h_i)$ will also minimize $\sum_{i=1}^{p}(w_i + h_i) - 2$.

The authors of (Beaumont *et al.*, 2001b) have shown that this general partitioning problem is NP-Complete and therefore heuristics must be resorted to in order to find optimal solutions. Such a heuristic is presented by Beaumont *et al.* (2001b) which restricts the rectangles of the partitioning to forming columns such as in Figure 2.5. The algorithm follows:

Algorithm 2.1 (Beaumont *et al.*, 2001b): Optimal column-based partitioning of a unit square between p heterogeneous processors:

- First, the processing elements are re-indexed in the non-increasing order of their speeds, $s_1 \geq s_2 \geq \ldots \geq s_p$. The algorithm only considers partitionings where the i-th rectangle in the linear column-wise order is mapped to processor $P_i, i \in \{1, \ldots, p\}$.
- The algorithm iteratively builds the optimal c column partitioning $\beta(c, q)$ of a rectangle of height 1 and width $\sum_{j=1}^{q} s_j$ for all $c \in \{1, \ldots, p\}$ and $q \in \{c, \ldots, p\}$:
  - $\beta(1, q)$ is trivial.
  - For $c > 1, \beta(c, q)$ is built in two steps:
    - First, $(q - c + 1)$ candidate partitionings $\{\beta_j(c, q)\}(j \in \{1, \ldots, q - c + 1\})$ are constructed such that $\beta_j(c, q)$ is obtained by combining the partitioning $\beta(c - 1, q - j)$ with the straightforward partitioning of the last column (the column number c) of the width $\sum_{i=q-j+1}^{q} s_i$ into j rectangles of the corresponding areas $s_{q-j+1} \geq s_{q-j+2} \geq \ldots \geq s_q$.
    - Then, $\beta(c, q) = \beta_k(c, q)$ where $\beta_k(c, q) \in \{\beta_j(c, q)_{j=1}^{q-c+1}\}$ and minimizes the sum of the half-perimeters of the rectangles.

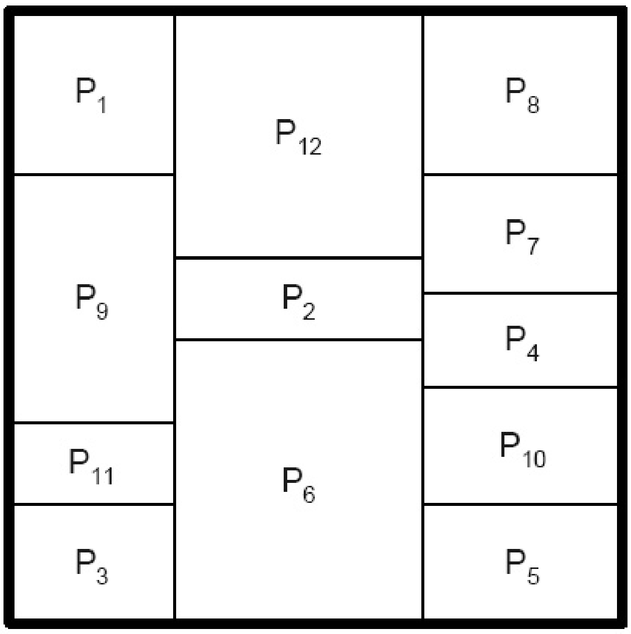


Figure 2.5: An example of a column-based partitioning of the unit square into 12 rectangles. The rectangles of the partitioning all make up columns, in this case three.

- The optimal column-based partitioning will be a partitioning from the set $\{\beta(c,p)_{c=1}^{p}\}$ that minimizes the sum of half-perimeters of rectangles.

Algorithm 2.1 runs in $O(p^3)$ time.

2.3.2.2 A More Restricted Column-Based Approach

Kalinov and Lastovetsky (2001), further restrict the column-based geometrical partitioning by assuming that the processing elements are already arranged in a set of columns (i.e. assuming that the number of columns c in the partitioning and the mappings of rectangles in each column to the processors are given). The algorithm is as follows.

Algorithm 2.2: (Kalinov and Lastovetsky, 2001): An optimal partitioning of a unit square between p heterogeneous processors arranged into c columns, each of which is made of r_j processors where $j \in \{1,\ldots,c\}$:

- Let the relative speed of the i-th processor from the j-th column, $P_{i,j}$ be $s_{i,j}$ where $\sum_{j=1}^{c}\sum_{i=1}^{r_j} s_{i,j} = 1$.
- First, partition the unit square into c vertical rectangular slices so that the

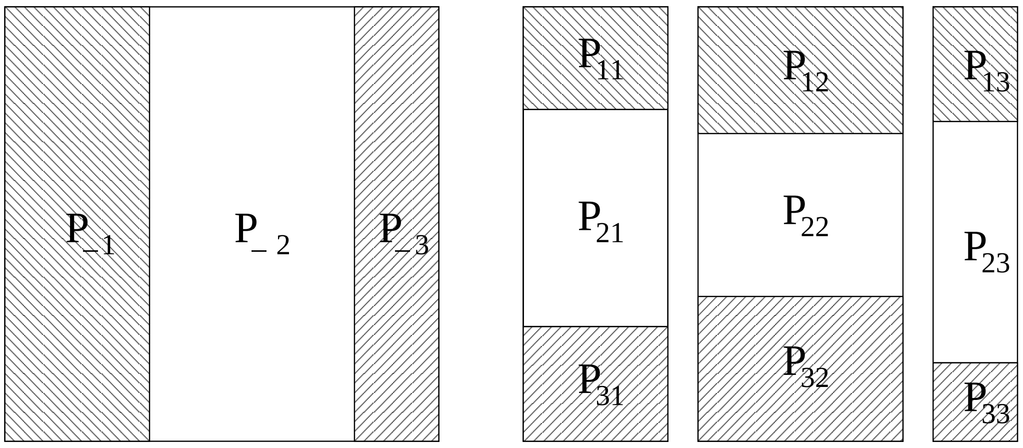


Figure 2.6: An example of Algorithm 2.2, a column-based partitioning for a 3×3 processor grid. Two steps are shown: Partitioning the unit square into columns then independently partitioning those columns into rectangles.

width of the j-th slice $w_j = \sum_{i=1}^{r_j} s_{i,j}$.

  - This partitioning makes the area of each vertical slice proportional to the sum of speeds of the processors in the corresponding column.

- Second, each vertical slice is partitioned independently into rectangles in proportion to the speeds of the processors in the corresponding column.

Algorithm 2.2 runs in linear time. Figure 2.6 shows this for a 3×3 processor grid.

2.3.2.3 A Grid Based Approach

Lastovetsky (2007) describes a partitioning which imposes a further restriction on the column-based approaches. The restriction is that the partitionings must form a grid as in Figure 2.7. This can be viewed from another approach. The grid based partitioning is one which partitions the unit square into rectangles so that there exist p and q such that any vertical line crossing the square will intersect exactly p rectangles and any horizontal line crossing the square will intersect exactly q rectangles, regardless of where these lines cross the unit square.

It is proposed and proven in (Lastovetsky, 2007) that in the case of a unit square partitioned into a grid of c columns, each of which is partitioned into r rows, the sum of half-perimeters of all partitions will be equal to (r+c). The corollaries which precipitate from this is that the optimal grid based partitioning

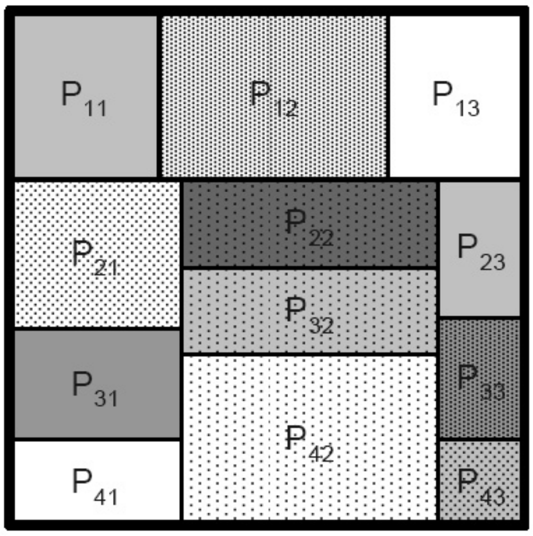


Figure 2.7: A grid based partitioning of the unit square into 12 partitions.

is one which minimizes r+c, and that the sum of half-perimeters of the optimal grid-based partitioning does not depend on the mapping of the processors onto the nodes of the grid. The algorithm for the optimal grid-based partitioning follows.

Algorithm 2.3: (Lastovetsky, 2007): Optimal grid-based partitioning of a unit square between p heterogeneous processors:

- Find the optimal shape $r \times c$ of the processor grid such that $p = r \times c$ and $(r + c)$ is minimized.
- Map the processors onto the nodes of the grid.
- Apply Algorithm 2.2 of the optimal partitioning of the unit square to this $r \times c$ arrangement of the p heterogeneous processors.

Step one finds the optimal shape of the processor grid that minimizes the sum of half-perimeters of any grid based partitioning for any mapping of the processors onto the nodes of the grid. Step two simply does the mapping (by any means). Step three then finds one such partitioning where the area of each rectangle is proportional to the speed of the processor owning it. It is noted that the solution is always column-based due to the nature of Algorithm 2.2.

The first step of Algorithm 2.3 is described by Algorithm 2.4.

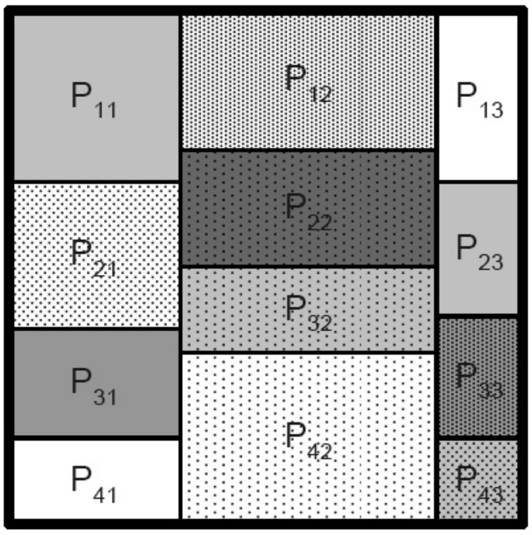


Figure 2.8: An optional grid based partitioning returned by Algorithm 2.3.

Algorithm 2.4 (Lastovetsky, 2007): Find r and c such that $p = r \times c$ and $(r + c)$ is minimal:

```
$r = \lfloor \sqrt{p} \rfloor$
**while** $(r > 1)$
 **if** $((p \bmod r) == 0)$
  **goto** stop;
 **else**
  $r - -;$
**stop:** $c = p/r$
```

Figure 2.8 shows an optimal grid based partitioning returned by algorithm 2.3.

In (Lastovetsky, 2007) the correctness of these algorithms is proven and the complexity of Algorithm 2.4 is shown to be $O(p^{3/2})$. Experimental results are also given which demonstrate the effectiveness of the grid based approach.

2.3.2.4 Cartesian Partitionings

The grid-based partitioning strategy is not the most restrictive partitioning problem that has been addressed. A *Cartesian* partitioning can be obtained from a column-based partitioning by imposing an additional restriction; namely that the rectangular partitionings also make up rows as seen in Figure 2.9. This results in any given partition having no more than four direct neighbors (up, down, left, right).

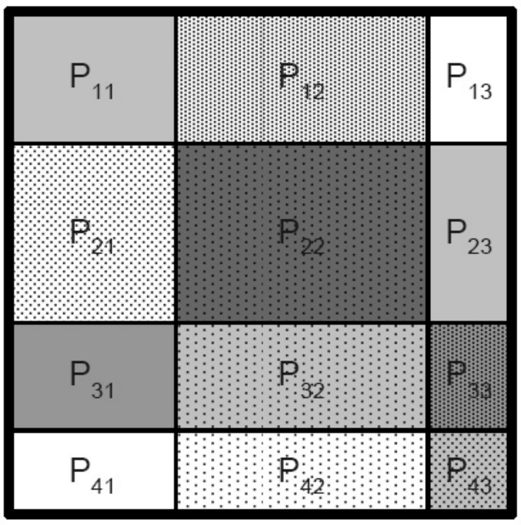


Figure 2.9: A cartesian partitioning of the unit square into 12 rectangular partitions. All partitions have no more than four direct neighbors: up, down, left, right.

Cartesian partitionings are important in heterogeneous algorithm design due to their scalability. This is due to no partition having more than one neighbor in any given direction. This lends itself to a scalable partitioning for algorithms which have communication patterns which only involve nearest neighbors, or for that matter, communications only between partitions in the same row and/or column.

Due to the additional restriction imposed by a Cartesian partitioning an optimal partitioning may not be achievable. The load between partitionings may not be perfectly balanced for some combinations of relative computing element speeds. This renders relative speeds unusable and the problem should be reformulated in terms of absolute speeds. The Cartesian partitioning problem can be formulated as follows:

- Given p processors, the speed of each of which is characterized by a given positive constant, find a Cartesian partitioning of a unit square such that:
  - There is a one to one mapping of partitions to computing elements.
  - The partitioning minimizes $\underset{i,j}{max}\left\{\frac{h_i \times w_j}{s_{ij}}\right\}$, where h_i is the height of partitions in the i-th row, w_j is the width of partitions in the j-th column, $s_{i,j}$ is the speed of the computing element owning the j-th partition in the i-th row, where $i \in \{1, \ldots, r\}$, $j \in \{1, \ldots, c\}$, and $p = r \times c$.

To the author's knowledge the Cartesian problem has not been studied as stated above in general form. An algorithm to solve this problem has to find an optimal shape $r \times c$ of the computing element grid, the mapping of the elements onto the grid, and the size of the partitions allocated to the elements. Simplified versions of the problem have been studied however (Beaumont *et al.*, 2001a; Dovolnov *et al.*, 2003). If the shape $r \times c$ of the partitioning is given the problem is proven to be NP-Complete (Beaumont *et al.*, 2001a). In addition it is not known if given both the shape $r \times c$ of the grid and the mapping of computing elements onto the grid are given, there exists a polynomial time solution.

An approximate algorithm of the simplified Cartesian problem where the shape $r \times c$ is given in Algorithm 2.5.

Algorithm 2.5 (Beaumont *et al.*, 2001a): Find a Cartesian partitioning of a unit square between p processors of the given shape $p = r \times c$:

- **Step 1.** Arrange the processors so that if linearly ordered row-wise, beginning from the top left corner, they will go in a nonincreasing order of their speeds.
- **Step 2.** For the processor arrangement, apply a known algorithm to find an approximate solution, $\{h_i\}_{i=1}^{r}, \{w_j\}_{j=1}^{c}$.
- **Step 3.**Calculate the areas h_i, w_i of the rectangles of the partitioning.
- **Step 4.** Rearrange the processors so that $\forall i, j, k, l : s_{ij} \geq s_{kl} \Leftrightarrow h_i \times w_j \geq h_k \times w_l$.
- **Step 5**. **If** Step 4 does not change the arrangement of the processors **then** return the current partitioning and stop the procedure **else** go to Step2.

2.4 Conclusion

In this section we saw that communication is unavoidable in parallel computing. Normally this is thought of in terms of computer networks: wires, switches, routers, etc. However it must be realized that communication occurs even at lower levels than two geographically distributed clusters. It happens between machines in the same rack, and even between cores on a single processor.

We then examined how the partitioning of data, specifically in the case of matrix matrix multiplication, can affect the amount of data that needs to be communicated. After briefly looking at elementary one and two-dimensional partitionings, several state-of-the-art methods were explored including general two-dimensional cases, more restricted column-based approaches, even more restricted strategies such as grid based, and finally very restricted two-dimensional approaches in cartesian cases. All of these partitioning strategies have two common threads. They all balance load and they all seek to minimize the sum of communication between partitions, normally through imposing restrictions. However this is not a hard rule, as in some cases relaxing restrictions can lead to better solutions.

CHAPTER

THREE

UCD HETEROGENEOUS COMPUTING LABORATORY (HCL) CLUSTER

3.1 Summary

The UCD Heterogeneous Computing Laboratory's main cluster was designed and hand built by myself to provide a controllable heterogeneous test platform. The cluster serves to develop and run HCL software and as a testbed for applications that are to be deployed on larger systems. The cluster is composed of 16 compute nodes and two switches, along with the necessary peripherals (server/UPS/access node/etc.) The cluster is unique in two ways: the bandwidth between each node can be controlled from 8kb/s up to 1Gb/s, and that the cluster itself can be split into two clusters in several configurations.

The compute nodes are heterogeneous in vendor, chipset, memory, operating system, number of cores, and many other ways. A complete host of scientific programming and monitoring system software is installed to allow HCL group members to easily develop and test applications. To date the cluster has been used in over 40 publications and six Ph.D. theses, with collaborators from many international universities. For a list see Appendix A. It has also been used for the development and testing of eight software packages. A list is included in Appendix B. A major upgrade (HCL Cluster 2.0) is currently under way in order to further serve the expanding needs of the Heterogeneous Computing Laboratory.

3.2 The UCD Heterogeneous Computing Laboratory Cluster

The UCD Heterogeneous Computing Laboratory's main cluster[1] is designed to provide a controllable hierarchal heterogeneous environment for research in high performance heterogeneous computing. The cluster was hand-built over the course of four months from purchasing to end of testing. It is completely stand-alone (that is to say it is not dependent on outside support, infrastructure, software, security, etc.).

The cluster is housed in a 42U server rack with power backup supplied by an APC Smart 1500 UPS. The main server is heterogeneous.ucd.ie (heterogeneous). Domain Name Service is provided by BIND (Berkley Internet Name Daemon). NAT (Network Address Translation) is implemented between the external network and the internal network (the compute nodes). Clonezilla is used for recovery issues. All compute nodes and access/server machines are kept synchronized through NTP (Network Time Protocol). DHCP (Dynamic Host Configuration Protocol) is used to configure IP addresses and hostnames. NFS (Network File System) is used to allow compute nodes access to server files. NIS (Network Information Service) is used to organize user names and passwords across the cluster. Smartmontools provides backup services. TORQUE/PBS manages jobs that users submit to the queues. There are four queues, with varying priority and preemption policies:

1. *normal*, where most jobs (which should not be interrupted) are placed. It has the highest priority.
2. *lowpri*, for running jobs which may run for extended periods but are interruptible. Should a job A be placed on the normal queue while a lowpri job B is running, B will be sent a kill signal, so it may shut down cleanly, and then it will be re-queued on the system, so it can resume running after A has finished.
3. *service*, which is for running service jobs such as home directory backups. This has lower priority than the above queues and jobs running on this queue are preemptable.
4. *volunteer*, which has the lowest priority, and is for executing volunteer computing jobs during otherwise idle periods. These jobs are also

[1] `hcl.ucd.ie/hardware`

preemptable.

For security the only incoming connections allowed are ssh except requests originating from inside the cluster that require incoming packets such as http, NTP, etc. which are also allowed. Incoming ssh connections are only allowed if they originate from designated IP addresses: currently some user machines and a central UCD School of Computer Science server (csserver.ucd.ie). The only other machine that is allowed is a gateway machine (hclgate.ucd.ie), which again is completely under group control. This machine is used to allow connections from users outside the UCD Intranet should csserver be down. Security on this machine is kept to the highest standard and updates as it is connected to the outside Internet.

3.3 Software

The HCL Cluster has been outfitted with all current packages necessary for the group to carry out parallel heterogeneous program and software development. Software packages currently available on the cluster include the following:

- autoconf
- automake
- ATLAS
- autotools
- BLAS
- Boost
- C\C++ (gnu)
- ChangeLog
- colorgcc
- Dia
- doxygen
- Eclipse
- evince
- fftw2
- GDB
- git
- gfortran
- gnuplot
- Graphviz
- gsl-dev
- LAPACK
- $\LaTeX$
- libboost-graph-dev
- libboost-serialization-dev

- libtool
- logp_mpi
- mc
- MPI (MPICH, OpenMPI)
- netperf
- octave3.2
- openmpi-bin
- openmpi-dev
- OProfile
- python
- qhull
- R
- r-cran-strucchange
- STL
- subversion
- valgrind
- vim

Software specifically for Heterogeneous Computing includes the following:

- Parallel extension of C:
  - mpC
- Extensions for MPI:
  - HeteroMPI
  - libELC
- Extensions for GridRPC:
  - SmartGridSolve
  - NI-Connect
- Computation benchmarking, modeling, dynamic load balancing:
  - FuPerMod
  - PMM
- Communication benchmarking, modeling, optimization:
  - CPM
  - MPIBlib

Software specifically for Mathematical Heterogeneous Computing includes the following:

- HeteroScaLAPACK
- Hydropad

Details on all of the above are available at:
`http://hcl.ucd.ie/wiki/index.php/Main\_Page`
and
`http://hcl.ucd.ie/wiki/index.php/HCL\_cluster`.

Ganglia is used to provide real-time cluster status to users and is available at: `http://heterogeneous.ucd.ie/ganglia`. Figure 3.1 shows overall cluster load, CPU, memory and network reports for the year April 2010 - March 2011. For node by node load, CPU, memory and network reports, see Appendix D.

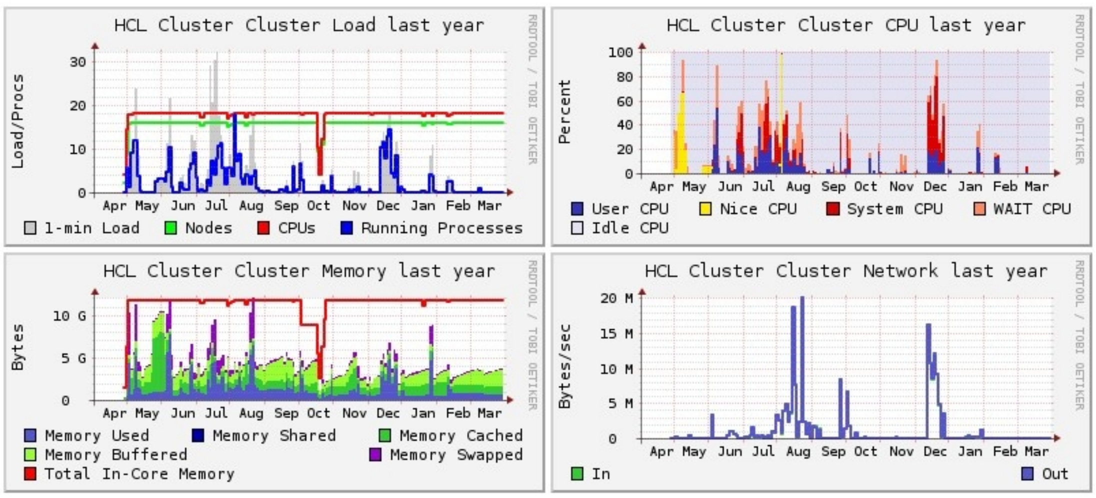


Figure 3.1: HCL Cluster load, CPU, memory and network profiles for the year April 2010 - March 2011.

3.4 Heterogeneity

The compute section of the cluster is comprised of 16 nodes (hcl01 - hcl16) from three different vendors (Dell, IBM, HP). It was designed to be heterogeneous in hardware, software, and network as follows.

3.4.1 Hardware

- Processor Architecture (Intel <Celeron, P4, Xeon>, AMD <Opteron Dual-Core>)

- Processor Speed (1.8 - 3.6 GHz)
- Ram (256 MB, 512 MB, 1GB)
- Main Storage (SCSI, SATA), (80 - 240GB)
- Front Side Bus (533, 800, 1k MHz)
- L2 Cache (1, 2 MB)

3.4.2 Operating Systems

- MS Windows (current option)
- Debian Linux "squeeze" kernel 2.6.32 (currently)
- Fedora Linux / Debian Linux (formerly)

3.4.3 Network

An overall schematic of the HCL Cluster network is given in Figure 3.2.

- 2 x Cisco 3560G 24+4 Switches (8Kb/s - 1Gb/s configurable bandwidth per port)

The HCL cluster is unique in two ways which make it specifically and particularly suited for research in heterogeneous computing. First, the ingress bandwidth of each link to every node can be controlled and specified from 8Kbp/s to 1Gb/s. This allows for a very large number of possible network configurations. Second, the cluster can be split to form two or three clusters of any possible configuration between the 16 nodes.

This is possible due to the following two reasons:

- The switches are connected by a 1GB/s SFP cable, also with an 8Kb/s - 1Gb/s configurable bandwidth.
- Each node has two network interface cards, the first of which (NIC1) is connected to Switch1 and the second (NIC2) is connected to Switch2.

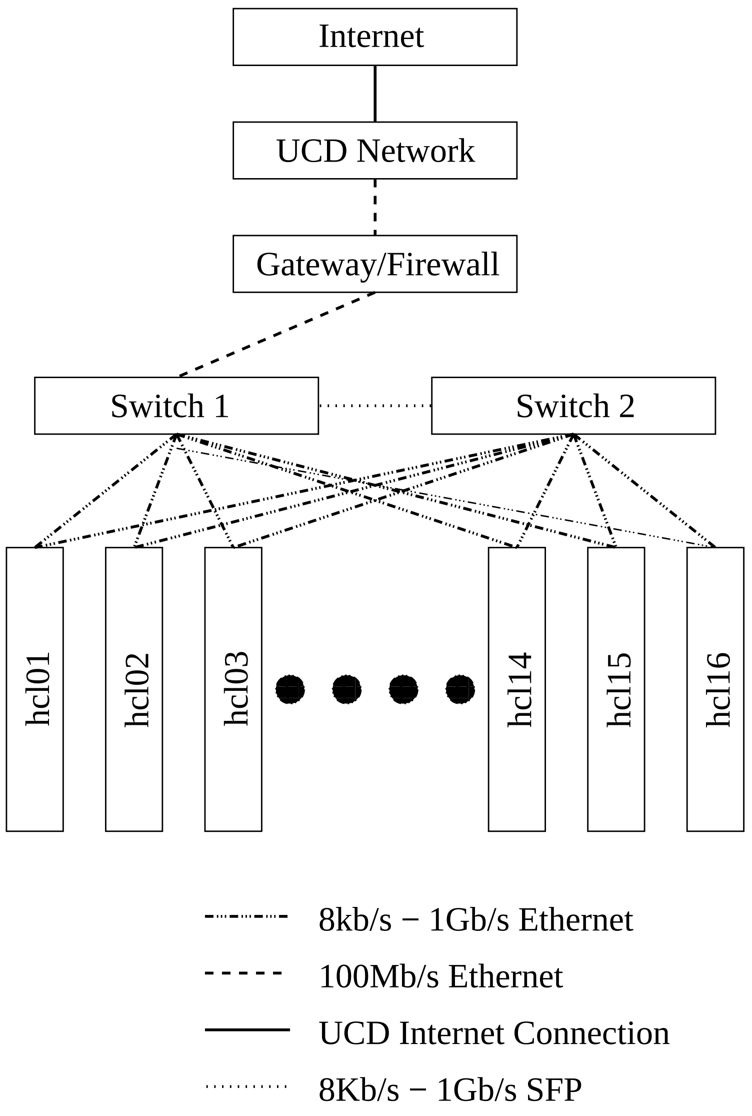


Figure 3.2: Schematic of the HCL Cluster Network.

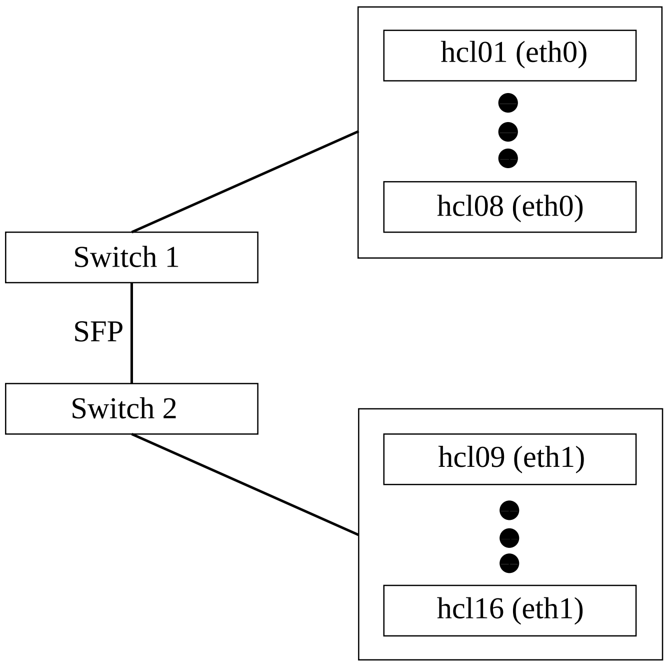


Figure 3.3: A two cluster configuration of the HCL Cluster. By bringing up NIC1 and bringing down NIC2 on hcl01 - hcl08, and the opposite for hcl09 - hcl16, and enabling the SFP connection between the switches, two connected clusters of eight nodes each are formed.

By enabling or disabling each node's particular NICs one can control which switch the node is currently connected to. For instance one can create a simple two cluster scenario by enabling NIC1 and disabling NIC2 on hcl01 - hcl08 and doing the opposite on hcl09 - hcl16. Due to the SFP connection between the switches this would create a two cluster scenario of eight nodes each, as depicted in Figure 3.3.

In fact, each node can be connected to both switches at the same time, allowing a group of hybrid clusters to be created. Think of nodes hcl01 and hcl02 being connected to both switches and then partitioning hcl03 - hcl09 so that they are connected to Switch1 and hcl10 - hcl16 so that they are connected to Switch2. We now have a hybrid cluster scenario as depicted in Figure 3.4.

Finally a hierarchy of clusters can be formed by disabling the SFP connection between the switches as shown in Figure 3.5. In this case the only way that Clusters 2 and 3 can communicate is through Cluster 1.

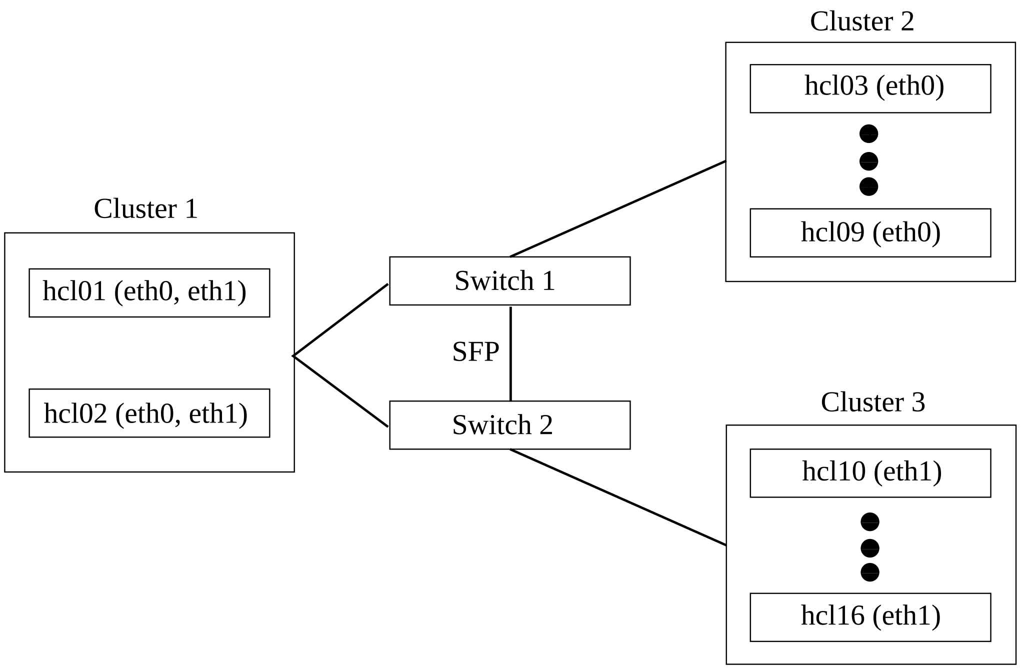


Figure 3.4: A hybrid configuration of the HCL Cluster. By bringing up NIC1 and NIC2 on hcl01 and hcl02, they form one cluster connected to both switches. By bringing up NIC1 and bringing down NIC2 for hcl03 - hcl09, and the opposite for hcl10 - hcl16, two more clusters are formed, one connected to Switch1 and one connected to Switch2.

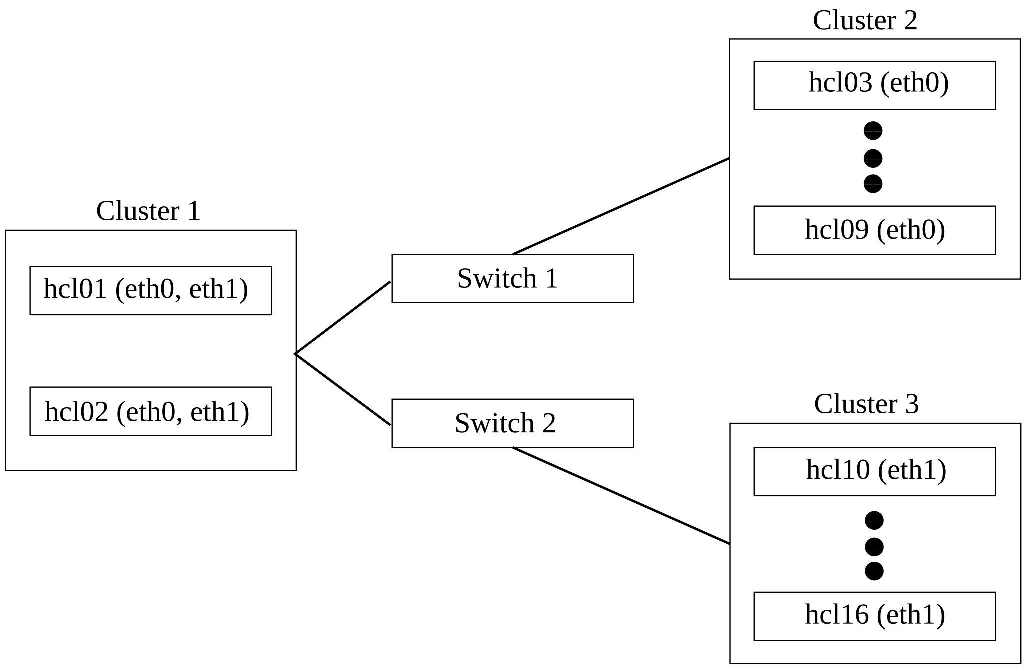


Figure 3.5: A hierarchal configuration of the HCL Cluster. By bringing up NIC1 and NIC2 on hcl01 and hcl02, they form one cluster. By bring up NIC1 and bringing down NIC2 for hcl03 - hcl09, and the opposite for hcl10 - hcl16, two more clusters are formed. The only way that Clusters 2 and 3 can communicate is through Cluster 1.

Node	Absolute Speed (MFlops)
hcl01	2171
hcl02	2099
hcl03	1761
hcl04	1787
hcl05	175
hcl06	1653
hcl07	1879
hcl08	1635
hcl09	3004
hcl10	2194
hcl11	4580
hcl12	1762
hcl13	4934
hcl14	4096
hcl15	2697
hcl16	4840
Total Cluster	**42,827**

Table 3.1: Performance of each node of the HCL Cluster as well as the aggregate performance in MFlops. All performance values were experimentally determined.

3.5 Performance and Specifications

Table 3.1 shows the performance of each node of the HCL Cluster as well as the aggregate performance in MFlops. All performance values were experimentally determined.

For results of the STREAM (McCalpin, 1995) benchmark on the HCL Cluster, see Appendix E. For results of the HPL (High Performance Linpack) benchmark[2] on the HCL Cluster, see Appendix F. The HCL Cluster has an absolute speed of 42.8 GFlops.

Table 3.5 shows the hardware and Operating System specifications for the HCL Cluster. As of May 2010 and the upgrade to HCL Cluster 2.0, all nodes are operating Debian "squeeze" kernel 2.6.32.

[2]`http://www.netlib.org/benchmark/hpl/`

3.6 Publications, Software Development and Releases

To date the HCL cluster has been used in over 40 publications and six Ph.D. theses (see Appendix A). It has also been used to develop the following eight software packages (also see Appendix B). Details, latest releases and patches are available at `http://hcl.ucd.ie`.

- ADL: Algorithm Definition Language, a new language and compiler that is designed to improve the performance of GridRPC/SmartGridRPC applications.
- CPM: Communication Performance Models of Heterogeneous Networks of Computers, a software tool that automates the estimation of the heterogeneous communication performance models of clusters based on a switched network.
- HeteroMPI: An extension of MPI for high performance heterogeneous computing.
- HeteroScaLAPACK: a linear algebra library for heterogeneous networks of computers.
- Hydropad: a grid enabled astrophysical application that simulates the evolution of clusters of galaxies in the universe
- MPIBlib: An MPI Benchmark library.
- NI-Connect: Non-intrusive and incremental evolution of grid programming systems.
- SmartGridSolve: High level programming system for high performance grid computing.

3.7 Conclusion

The Heterogeneous Computing Laboratory Cluster is a purpose built platform designed and built to suit the specific needs of the HCL group. It is a highly tunable and heterogeneous testbed unique in its flexibility and ability to take on a multitude of different configurations. It has been used in over 40 publications, six Ph.D. theses, and the development of eight software packages to date (See Appendices A and B). A major upgrade (HCL Cluster 2.0) is currently underway in order to further serve the expanding needs of the Heterogeneous Computing Laboratory in the future.

Name	Make/Model	Processor	FSB	L2 Cache
hclswitch1	Cisco Catalyst 3560G	N/A	N/A	N/A
hclswitch2	Cisco Catalyst 3560G	N/A	N/A	N/A
UPS	APC Smart UPS 1500	N/A	N/A	N/A
hcl01	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl02	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl03	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hl04	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl05	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl06	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl07	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl08	Dell Poweredge 750	3.4 Xeon	800MHz	1MB
hcl09	IBM E-server 326	1.8 Opteron	1GHz	1MB
hcl10	IBM E-server 326	1.8 Opteron	1GHz	1MB
hcl11	IBM X-Series 306	3.2 P4	800MHz	1MB
hcl12	HP Proliant DL 320 G3	3.4 P4	800MHz	1MB
hcl13	HP Proliant DL 320 G3	2.9 Celeron	533MHz	1MB
hcl14	HP Proliant DL 140 G2	3.4 Xeon	800MHz	1MB
hcl15	HP Proliant DL 140 G2	2.8 Xeon	800MHz	1MB
hcl16	HP Proliant DL 140 G2	3.6 Xeon	800MHz	2MB

Name	RAM	HDD 1	HDD 2	NIC	OS
hclswitch1	N/A	N/A	N/A	24 x 1Gb/s	N/A
hclswitch2	N/A	N/A	N/A	24 x 1Gb/s	N/A
UPS	N/A	N/A	N/A	N/A	N/A
hcl01	1GB	80GB SATA	250GB SATA	2 x 1Gb/s	FC4
hcl02	1GB	80GB SATA	250GB SATA	2 x 1Gb/s	FC4
hcl03	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl04	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl05	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl06	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl07	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl08	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl09	1GB	80GB SATA	N/A	2 x 1Gb/s	Debian
hcl10	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl11	512MB	80GB SATA	N/A	2 x 1Gb/s	Debian
hcl12	512MB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl13	1GB	80GB SATA	N/A	2 x 1Gb/s	FC4
hcl14	1GB	80GB SATA	N/A	2 x 1Gb/s	Debian
hcl15	1GB	80GB SATA	N/A	2 x 1Gb/s	Debian
hcl16	1GB	80GB SATA	N/A	2 x 1Gb/s	Debian

Table 3.2: Hardware and Operating System specifications for the HCL Cluster. As of May 2010 and the upgrade to HCL Cluster 2.0, all nodes are operating Debian "squeeze" kernel 2.6.32.

CHAPTER

FOUR

PARTITIONING A MATRIX IN TWO - GEOMETRIC APPROACHES

4.1 Summary

Chapter 2 concluded with a continuing thread which was common to a number of partitionings that improved performance by introducing restrictions to the general partitioning problem. It is however possible in some cases to improve performance by relaxing restrictions. In this chapter we explore one such case. That case is partitioning a matrix in two. The choice of two partitionings is not a restriction however, as future chapters will show that this approach is not restricted to just two partitions. It is how the partitions themselves are formed that is the relaxation of restriction.

The initial motivation which led to this work stems from two sources. First, there are many general heterogeneous matrix partitioning algorithms which work well for several, dozens, hundreds, or even thousands of computing entities. We will see that all result in a simple, and non-optimal partitioning when applied to architectures of small numbers of computing entities—in the case of this chapter, two. As stated earlier we intentionally set out to investigate the case of a small number of computing entities to see what is happening in what is this perceived (by some researchers) "degenerate" case. Despite its existence for at least 30 years, parallel MMM research has almost completely ignored this area. This is perhaps due to a lack of interest in a small number of processors or computers working together. Indeed in these cases the speedup benefits are obviously small. However with modern architectures, particularly cluster-based architectures and grids, small numbers of clusters or sites working together can clearly yield a very attractive benefit. As we will see, perhaps a small number of computing entities is not as "degenerate" as some researchers have believed.

Second, this was borne of a keen awareness of the parallel, distributed and especially the heterogeneous nature of computing platforms which are omnipresent in computing today. In addition, most parallel and distributed algorithms in existence today are designed for and only work well on homogeneous platforms. Finally, most heterogeneous algorithms are based on their homogeneous counterparts. It is now necessary for many new algorithms to be designed with heterogeneous platforms in mind from the start.

4.2 A Non-Rectangular Matrix Partitioning

All of the state-of-the-art matrix partitioning areas discussed thus far including the specific examples explored, had two common threads—they all balance computational load and they all seek to minimize communication between partitions. Both of these objectives are sought in order to minimize the overall execution time of matrix matrix multiplications.

It is true that there was another design feature common to all. This feature is that they are all rectangular. In all algorithms discussed, the solution sets contained only partitionings which result in nothing but rectangular partitions. Thus, it seems that the partitioning problem itself is somewhat restricted. Most researchers believe that allowing for non-rectangular partitions (relaxing this rectangular restriction) will not significantly improve the performance of the partitioning, and at the same time significantly complicate the solution of the partitioning problem.

This statement seems plausible, however in this chapter we introduce a non-rectangular partitioning solution that can significantly outperform counterpart rectangular partitionings. In some cases this non-rectangular partitioning can prove to be optimal amongst all partitionings. In addition, it does not significantly add to the complexity of the solution itself.

First we will define exactly what non-rectangular is meant in this context.

4.2.1 The Definition of Non-Rectangularity

In Figure 4.1 there are two partitions, s_1 and s_2. s_1 is non-rectangular and s_2 is rectangular. To precisely define what a non-rectangular partition is in context of the general partitioning problem, we start with the definition of a rectangular partitioning.

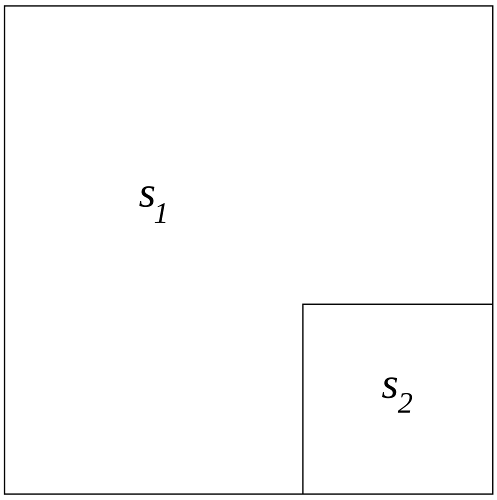


Figure 4.1: A heterogeneous non-rectangular partitioning consisting of two partitions, one rectangular and one a non-rectangular polygon.

Definition 4.1 A partitioning of any problem whose solution is to partition the unit square is *rectangular* if all of the following four rules apply:

1. The unit square is completely tiled by partitions
2. No partitions overlap
3. All partition boundaries are parallel to the boundaries of the unit square
4. All partitions are rectangular

It would seem at first that in order to define a *non-rectangular* partitioning, all that is needed is to eliminate Definition 4.1, Rule 4. This seems reasonable as non-rectangular partitions would now be allowed, and Definition 4.1, Rule 3 would eliminate circles, triangles, and other exotic partitions.

However, a definition is needed that makes non-rectangular partitionings such as that in Figure 4.1 and all purely rectangular partitionings mutually exclusive. Eliminating Definition 4.1, Rule 4 would not do so, as purely rectangular partitions would still fit into the new definition of non-rectangular partitions.

The solution is to replace Definition 4.1 Rule 4 with the following—*At least one partition is non-rectangular*. This gives us the following definition.

Definition 4.2 A partitioning of any problem whose solution is to partition the unit square is *non-rectangular* if all of the following four rules apply:

1. The unit square is completely tiled by partitions
2. No partitions overlap
3. All partition boundaries are parallel to the boundaries of the unit square
4. At least one partition is non-rectangular

We are now in a position where rectangular and non-rectangular partitionings are mutually exclusive, and we can differentiate between, and therefore compare, rectangular and non-rectangular partitionings.

4.2.2 A Non-Rectangular Partitioning - Two Partitions

In Section 2.3.2.1, the Matrix Partitioning Problem (Beaumont *et al.*, 2001b) was stated. It is restated more formally as Problem 4.1.

Problem 4.1: The Matrix Partitioning Problem

- Given p computing elements $P_1, P_2, \dots, P_p$, the relative speed of which is characterized by a positive constant s_i, where $\sum_{i=1}^{p} s_i = 1$
- Partition a unit square into p rectangles so that
  - There is a one to one mapping of elements to rectangles.
  - The area of the rectangle allocated to element P_i is equal to s_i where $i \in \{1, \dots, p\}$.
  - The partitioning minimizes $\sum_{i=1}^{p}(w_i + h_i)$, where w_i is the width of the rectangle and h_i is the height of the rectangle assigned to element P_i.

As Problem 4.1 is restricted to rectangles, we need to reformulate the problem to be more general in order to allow for non-rectangular partitions.

Problem 4.2: The General Matrix Partitioning Problem

- Given p computing elements $P_1, P_2, \ldots, P_p$, the relative speed of which is characterized by a positive constant s_i, where $\sum_{i=1}^{p} s_i = 1$
- Partition a unit square into p polygons so that
  - There is a one to one mapping of elements to polygons.
  - The area of the polygon allocated to element P_i is equal to s_i where $i \in \{1, \ldots, p\}$.
  - The partitioning minimizes the sum of half perimeters of all polygons.

In the case of a very small problem size, say ($p = 2$), *every* rectangular partitioning of Problem 4.1 or Problem 4.2 will result in one such as that in Figure 4.2. This is because there is no other way but to partition the unit square into two rectangles but to draw a straight line across the square, while ensuring that each rectangle is of the appropriate area as dictated by Problem 4.1 (or each partition is of the appropriate area as dictated by Problem 4.2).

As Problems 4.1 and 4.2 are equivalent, except that Problem 4.2 relaxes the restriction of rectangular solutions, clearly any rectangular partitioning algorithm can be applied to either, and generate the same solution.

A non-rectangular partitioning however can only be applied to Problem 4.2, but since Problem 4.2 is more general this is not a problem. Such an algorithm follows:

Algorithm 4.1: An algorithm to solve Problem 4.2, the General Matrix Partitioning Problem, for $p = 2$.

- **Step 1.** Re-index the processing elements P_i in the non-decreasing order of their speeds, $s_2 \leq s_1$.
- **Step 2.** Create a rectangular partition of size s_2 in the lower right corner of the unit square.
- **Step 3.** Map P_2 to the rectangular partition of size s_2, and map P_1 to the remaining non-rectangular balance of the unit square of size $1 - s_2 = s_1$.

The result of Algorithm 4.1 is of the form in Figure 4.3.

In each case the volume of communication is proportional to the sum of half perimeters of all partitions.

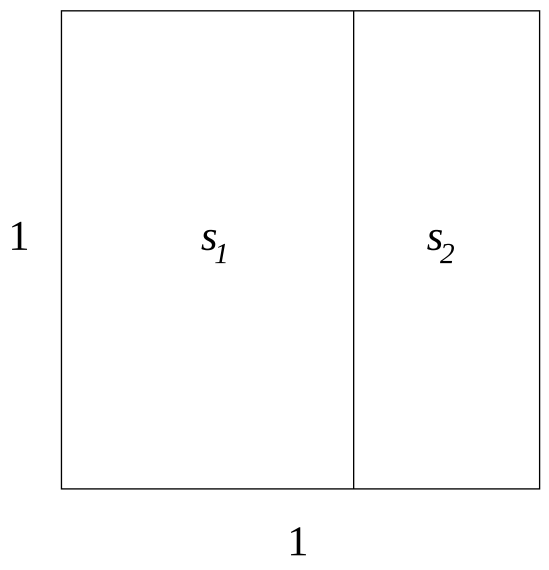


Figure 4.2: A rectangular (column-based) partitioning to solve Problem 4.2. This is the rectangular counterpart of the non-rectangular partitioning in Figure 4.3

.

- For the rectangular case this is equal to 3 .
- For the non-rectangular case in Figure 4.3 this is equal to Equation 4.1.

$$\left(\frac{1+1+(1-y)+(1-x)+x+y}{2}+x+y\right)=2+x+y \tag{4.1}$$

We have left the relative speeds s_i of the processors P_i as unknowns (but remember that their sum equals 1), and since both are non-zero, we can make the following observations:

- For the rectangular partitioning, the sum of half perimeters is equal to 3 regardless of the ratio between s_i and s_2. (Imagine the border between the two partitions in Figure 4.2 sliding left or right).
- For the non-rectangular partitioning, the sum of half perimeters is equal to $2+x+y$, and therefore depends on x and y, and the sizes of the partitions s_1 and s_2, and therefore the ratio between the partitions.

Thus a natural manipulation is to minimize the sum $(x+y)$, to drive down the sum of half perimeters for the non-rectangular case. Doing so will not affect the area of the partitions, as x and y form a rectangle of size s_2, and are free to be changed while keeping the area of s_2 (and therefore the area of s_1) constant.

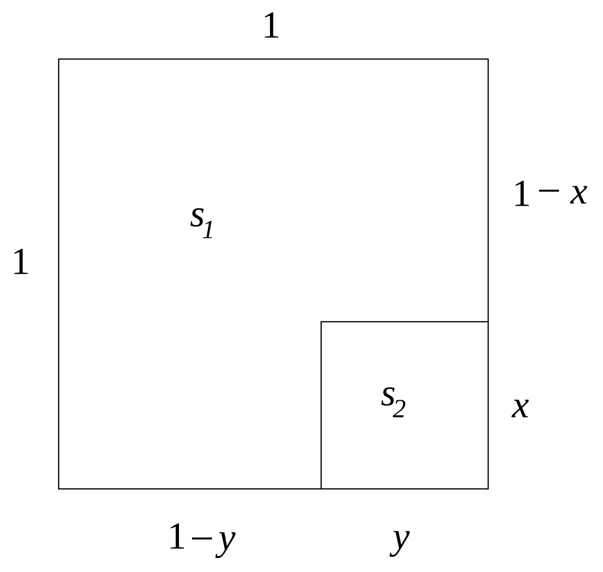


Figure 4.3: A non-rectangular partitioning to solve Problem 4.2.

Since $(x \times y)$ is a rectangle of fixed area, minimising its perimeter is proportional to minimising $(x + y)$, and occurs when the rectangle is a square. Thus Equation 4.1 becomes

$$2 + 2 \times \sqrt{s_2} \tag{4.2}$$

We can then answer the question: Is the non-rectangular partitioning's sum of half perimeters less than that of the rectangular partitioning? The answer is found by answering

$$2 + 2 \times \sqrt{s_2} <? \ 3. \tag{4.3}$$

Clearly, Inequality 4.3 is true for all $\sqrt{s_2} < \frac{1}{2}$. This corresponds to $s_2 < \frac{1}{4}$, which means that $s_1 > \frac{3}{4}$. We can therefore conclude that when $p = 2$, and $\frac{s_1}{s_2} > 3$, the non-rectangular partitioning has a lower sum of half perimeters than that of the rectangular. When $\frac{s_1}{s_2} = 3$, the sum of half perimeters of the two partitionings are equal.

In other words as long as the area of the larger partition is greater than three times the smaller, the non-rectangular partitioning will result in a lower sum of half perimeters and therefore a lower volume of communication. We can summarize with the following:

- For ratios $\rho : 1$ where $\rho \in [1, 3)$ the rectangular partitioning has a lower total volume of communication.
- For the ratio $\rho : 1$ where $\rho = 3$ the rectangular and non-rectangular partitionings are equivalent in total volume of communication.

- For ratios $\rho : 1$ where $\rho \in (3, \infty)$ the non-rectangular partitioning has a lower total volume of communication.

This of course depends on the sum of half perimeters $\hat{C}$ being proportional to the total volume of communication (TVC) for the non-rectangular solution. This will be explored in Section 5.5.2. We will then provide a short yet formal proof.

4.2.3 Ratio Notation

As in the previous section above, we will be discussing and utilizing the speed (or computational power) ratio between two computing entities (processors, clusters, etc.) often in the coming chapters. This ratio will be denoted ρ, where $\rho \overset{def}{=} \frac{s_1}{s_2}$, where s_1 and s_2 are the relative speeds or processing power of the entities s_1 and s_2. For simplicity, these ratios are always normalized so that $s_2 = 1$. Therefore the following identities apply:

$$\text{in fractional notation, } \rho = \frac{s_1}{s_2} = \frac{s_1}{1} = \frac{\rho}{1} \Rightarrow \rho = s_1$$

$$\text{and in ratio notation, } \rho = s_1 : s_2 = s_1 : 1 = \rho : 1 = \rho : s_2$$

At different times it will be more appropriate to use one over another.

Additionally, because we assume perfect load balancing, s_1 and s_2 are proportional to the partition areas assigned to entities P_1 and P_2 (or simply just entities 1 and 2) respectively.

4.2.4 Optimization

We have seen that for partitioning the unit square into two partitions, a non-rectangular partitioning can have a lower sum of half perimeters than a rectangular one, given the ratio between the two partition areas is $\rho : 1$ where $\rho \in (3, \infty)$.

What can be said about the lower bound of the sum of half perimeters? How close to this lower bound is the non-rectangular partitioning? In Beaumont *et al.* (2001b) the authors give a lower bound for the sum of half perimeters $\hat{C}$

to be LB in Equation 4.4

$$LB = 2 \times \sum_{i=1}^{p} \sqrt{s_i} \tag{4.4}$$

where p is the number of partitions and s_i is the area of the ith partition. This is because the half perimeter of any partition is at a minimum when that partition is a square.

We then consider the following case. We have $p = 2$, $s_1 = 1 - \epsilon$ and $s_2 = \epsilon$ where ϵ is some arbitrarily small but positive number. *Any* rectangular partitioning will require two partitions, formed by drawing a line of length 1 (See Figure 4.2). This will result in a sum of half perimeters, $\hat{C} = 3$, however LB gets arbitrarily close to 2.

Now we consider the non-rectangular partitioning of Figure 4.3. As $\epsilon \to 0$, the sum of half perimeters $\hat{C} \to 2$ and $LB \to 2$, which happens to be the half perimeter of the unit square itself, and therefore obviously optimal. Thus the non-rectangular partitioning can be optimal.

In (Beaumont *et al.*, 2001b), the authors make an experimental comparison of the sum of half perimeters and theoretical lower bound of the column-based rectangular partitioning discussed in Section 2.3.2.1. The comparison between $\hat{C}$ and LB is made in the following manner:

- Two curves are generated for a number of processors (entities) from 1 to 40.
- The first curve reflects the mean value $\frac{\hat{C}}{LB}$ for 2,000,000 randomly generated partition areas (s_i).
- The second curve reflects the minimum value generated for the same.

Clearly the average values give a good idea of how the partitioning performs overall, while the minimum value (with such a large set of randomly generated areas) should reflect on how optimal the partitioning can be for each number of processors (entities). (Note that the number of processors is equal to the number of partitions as there is a one-to-one mapping of processors to partition areas). Figure 4.4 shows the results of these experiments (adopted from (Beaumont *et al.*, 2001b)).

Figure 4.4 shows a number of interesting characteristics:

- The average values show:

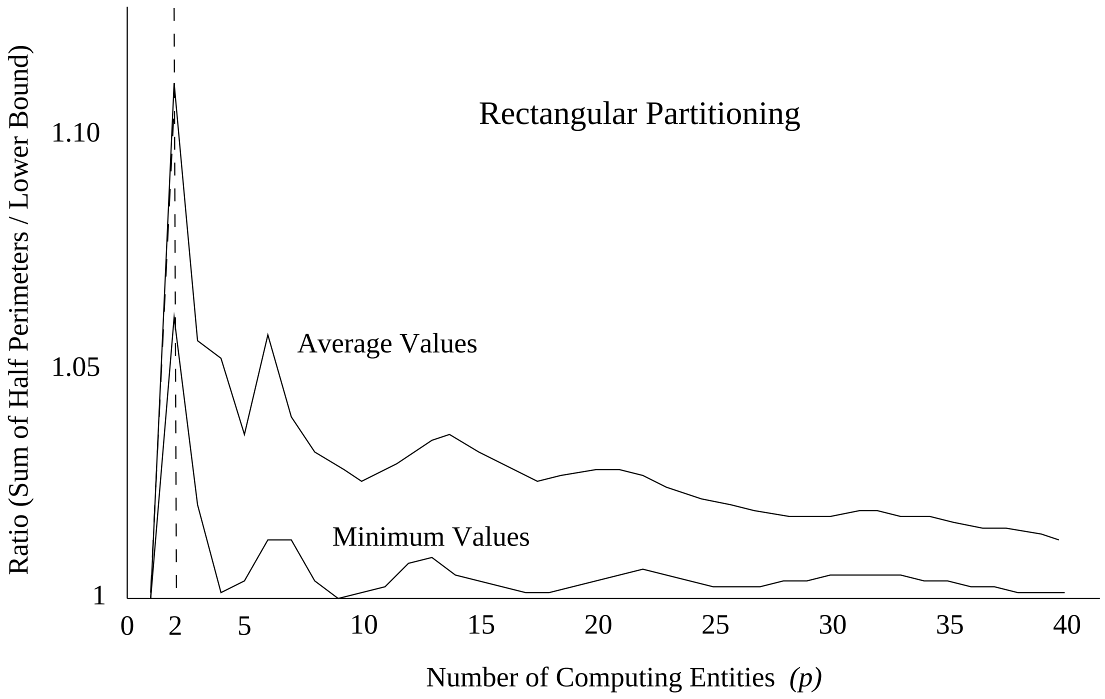


Figure 4.4: A plot of the average and minimum $\frac{\hat{C}}{LB}$ values for a number of processors (entities) p from 1 to 40 for the column based rectangular partitioning of Beaumont *et al.* (2001b). For each value p, 2,000,000 partition values s_i are randomly generated.

  - By far the worst performance is for $p = 2$, by a factor of approximately 2 over that of the second worst performing case, $p = 6$.

- The minimum values show:
  - "Magic" numbers where the minimum $\frac{\hat{C}}{LB}$ values are at or near 1, indicating optimal or near optimal solutions. These are numbers where it is theoretically possible to tile the unit square with a grid of x squares of size $(\sqrt{x} \times \sqrt{x})$. Of course these numbers are the perfect squares $(1, 4, 9, 16, 25, 36, ...)$. The average results also show better performance near these numbers.
  - By far the worst performance is for $p = 2$, by a factor of approximately 5, over that of the second worst performing case, $p = 6$. (Note that $p = 3$ just out performs $p = 6$ and is the third worst performing case, as we will discuss this case in Chapter 6.)

It is important to note that in the case of $p = 2$, *any and all* rectangular partitionings should have the same performance, as they all result in equivalent partitionings in the form of that in Figure 4.2.

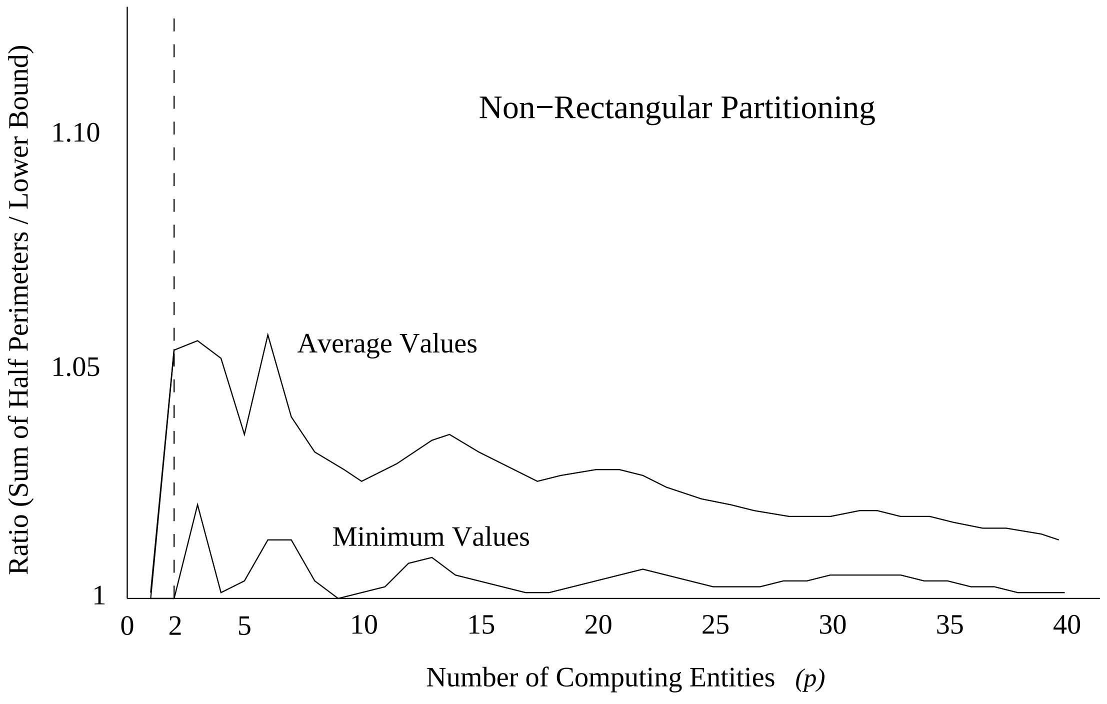


Figure 4.5: A plot of the average and minimum $\frac{\hat{C}}{LB}$ values for a number of processors (entities) p from 1 to 40 except $p = 2$ using the rectangular partitioning, and for $p = 2$, the non-rectangular partitioning. For each value p, 2,000,000 partition values s_i are randomly generated.

Figure 4.5 Shows the same plot as Figure 4.4 but with the non-rectangular partitioning data for $p = 2$. In other words for $p = 2$ the non-rectangular partitioning of Figure 4.3 is used (with $s_2 = \sqrt{s_2} \times \sqrt{s_2}$) but for all other p, the partitioning values of Figure 4.4 have been retained. In addition, for $p = 2$, the restriction $\frac{s_1}{s_2} \geq 3.0$ has been added since it is known that in cases where $\frac{s_1}{s_2} < 3.0$, the rectangular partitioning should be used instead.

Figure 4.5 Shows a number of interesting characteristics:

- The average values show
  - For the case of $p = 2$, the average value has dropped from 1.105 to 1.054, an improvement of almost 49%.
  - $p = 2$ has gone from the worst average value to third worst, now better than $p = 3$ and $p = 6$.
  - The worst performing average case is now $p = 6$.
- The minimum values show
  - The worst performing case is now $p = 3$ instead of $p = 2$

- For the case of $p = 2$ (which was the worst performing case at 1.061), the performance is now optimal (actual value 1.000001). This concludes that p need not be a "magic" number in order for the minimum ratio to be at (or very very near) 1 (optimal).

It is interesting that for the simple case of $p = 2$ a non-rectangular partitioning has out performed a rectangular one in all but a very small practical range of ratios. One more glance at Figures 4.4 and 4.5 begs the question, can the non-rectangular partitioning be extended to $p > 2$?

What has been established, is that in the case of $p = 2$, a hybrid partitioning employing *any* rectangular partitioning for $\frac{s_1}{s_2} < 3.0$, and using the non-rectangular partitioning discussed in this section for all others will give the best theoretical performance known. Further, the only room for improvement lies in the region $\frac{s_1}{s_2} < 3.0$, as above that ratio the non-rectangular partitioning provides an optimal solution to the General Matrix Partitioning Problem (Problem 4.2).

4.3 Conclusion

This chapter presented a geometric approach to partitioning a matrix into two partitions. All rectangle-based partitionings reduce to equivalent partitionings when applied to the General Matrix Partitioning Problem (Problem 4.2), yielding solutions with sum of half perimeters $\hat{C} = 3$, and an accordingly proportional total volume of communication. Despite common belief that a non-rectangular approach would not yield a better result, and significantly complicate the solution (Lastovetsky and Dongarra, 2009), a non-rectangular solution was presented which does yield better performance at little-to-no complication. This performance comes in a lower sum of half perimeters than all rectangular partitionings provided the ratio between the areas of the two partitions is greater than $3 : 1$. This non-rectangular solution was shown to be optimal as the sum of half perimeter approaches the theoretically optimal sum of half perimeters $\hat{C} = 2$ as the ratio between partition areas grows. A hybrid algorithm utilizing the non-rectangular algorithm for ratios $\geq 3 : 1$, and any rectangular algorithm for ratios $< 3 : 1$ would yield better overall results compared to all known algorithms.

Further, this chapter has laid the foundation for Chapter 5, which will show that an architecture requiring a data partitioning amongst "only" two computing entities is not a degenerate case, as with modern scientific computing platforms each entity/cluster/site/etc. can be of great computational power locally. This concept is extended to more than two processors and useful application areas in Chapters 6, 7, and 8, providing theoretical and experimental evidence to show that a partitioning amongst small numbers of computing entities can be quite advantageous.

CHAPTER

FIVE

THE SQUARE-CORNER PARTITIONING

5.1 Summary

This chapter introduces the "Square-Corner Partitioning", whose geometrical version was introduced in Chapter 4. The Square-Corner Partitioning is a top-level, non-rectangular partitioning for matrix matrix multiplication between two clusters or other computing entities. After communication details and theoretical performance are examined, experimental results of simulations using two processors, then experimental results using two different sets of two clusters are presented. The Square-Corner Partitioning is compared to rectangular partitionings in both communication and execution times.

The Square-Corner Partitioning is based on Algorithm 4.1, and accompanying geometrical analysis in Chapter 4. The solution partitioning is similar to Figure 4.3. Results of experiments on two clusters correlate well with both the simulations presented here and theoretical performances. In the case of two clusters, each of which may have great computational power, this partitioning proves to be an important asset to anyone performing large matrix matrix multiplications, or any problem with a communication schedule similar to MMM.

5.2 The Square-Corner Partitioning

In Chapter 4 a non-rectangular partitioning solving the General Matrix Partitioning Problem (Problem 4.2) for $p = 2$ was presented. This partitioning was shown to have a lower sum of half perimeters (SHP) than any rectangular partitioning solving the same problem, provided the ratio between the two clusters is greater than 3 : 1. More specifically:

- For ratios $\rho : 1$ where $\rho \in [1, 3)$ the rectangular partitioning has a lower total volume of communication.
- For the ratio $\rho : 1$ where $\rho = 3$ the rectangular and non-rectangular partitionings are equivalent in total volume of communication.
- For ratios $\rho : 1$ where $\rho \in (3, \infty)$ the non-rectangular partitioning has a lower total volume of communication.

This partitioning can be now be more specifically defined as The Square-Corner Partitioning, described by Algorithm 5.1.

Algorithm 5.1: The Square-Corner Partitioning, a solution to Problem 4.2, the General Matrix Partitioning Problem, for $p = 2$.

- **Step 1.** Re-index the processing elements P_i in the non-decreasing order of their speeds, $s_2 \leq s_1$.
- **Step 2.** Create a *square* partition of size s_2 in *any* corner of the matrix to be partitioned.
- **Step 3.** Map P_2 to the square partition of size s_2, and map P_1 to the remaining balance of the matrix of size $s_1 = N^2 - s_2$, where N is the size of the matrix.
- **Step 4.** Do the same for all matrices.

As Figure 5.1 shows, the rectangular Straight Line Partitioning always results in a TVC equal to N^2, in two communication steps regardless of the power ratio ρ.

The Square-Corner Partitioning has a TVC equal to Equation 5.1,

$$\text{TVC} = 2 \times N \times \sqrt{s_2} \tag{5.1}$$

where N is the matrix dimension and $\sqrt{s_2} \times \sqrt{s_2}$ is the dimension of Cluster 2's square partitions, shown in Figure 5.2.

As we have now formally defined the Square-Corner Partitioning, we will formally, yet simply prove that it has a lower TVC than the Straight-Line Partitioning for $\rho > 3$.

Theorem 5.1: For all power ratios ρ greater than 3, the Square-Corner Partitioning has a lower TVC than that of the Straight-Line Partitioning, and therefore all known partitionings, when $p = 2$.
Proof:

$$\begin{aligned}
TVC_{SCP} &< TVC_{SLP} \\
2 \times N \times \sqrt{s_2} &< N^2 \\
\frac{2 \times N^2}{\sqrt{\rho + 1}} &< N^2 \\
2 &< \sqrt{\rho + 1} \\
4 &< \rho + 1 \\
3 &< \rho
\end{aligned}$$

Q.E.D

Similar proofs show that for the power ratio $\rho = 3$, the Square-Corner TVC is exactly equal to the Straight-Line TVC, and for ratios $\rho < 3$, the Square-Corner TVC exceeds that of the Straight-Line Partitioning.

We can also simply prove that as defined, the Square-Corner Partitioning minimizes the total volume of communication against variants of the algorithm. Possible variants include assigning non-square partitions to Cluster 2. This means relaxing the $\sqrt{s_2} \times \sqrt{s_2}$ square partition to become a rectangle of width α, height β, and area s_2. We wish to minimize the total volume of communication, which more generally than Equation 5.1 can be given by Equation 5.2.

$$\text{TVC} = \alpha \times N + \beta \times N \tag{5.2}$$

With the restraints:

$$\alpha \times \beta = s_2, \quad 0 < \alpha \leq N, \quad 0 < \beta \leq N \tag{5.3}$$

Theorem 5.2: The TVC of the Square-Corner Partitioning, C is minimized only when the Square-Corner Partitioning assigns a square partition to the smaller partition area $\alpha \times \beta = s_2$.

Proof: The first derivative of C is set equal to zero and it is shown that this

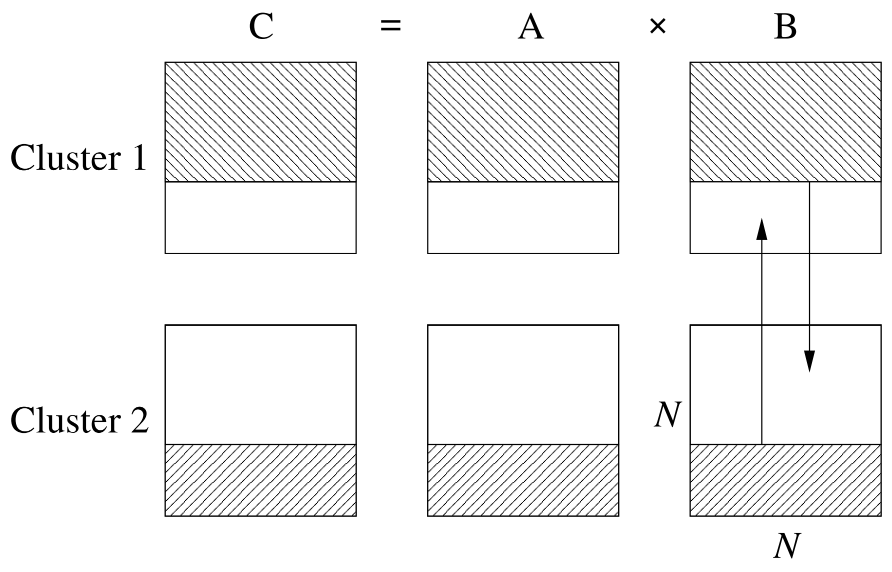


Figure 5.1: The Straight-Line Partitioning and communication steps required to carry out $C = A \times B$.

occurs only when $\alpha = \beta$, and therefore when the partition of area s_2 is a square. We then see that the second derivative of C is always positive and therefore any other partition will result in an increase in C.

$$\begin{aligned}
C &= \alpha \times N + \beta \times N \\
C &= \alpha \times N + \frac{s_2}{\alpha} \times N \\
\frac{dC}{d\alpha} &= N - \frac{s_2 \times N}{\alpha^2} \\
N - \frac{s_2 \times N}{\alpha^2} &= 0 \\
s_2 &= \alpha^2, \quad \therefore \quad \alpha = \beta \\
\frac{d^2C}{d\alpha^2} &= N + 2 \times \frac{s_2 \times N}{\alpha^3} > 0
\end{aligned}$$

Q.E.D

Figure 5.2 shows the Square-Corner Partitioning and the necessary data movements required to calculate a matrix product $C = A \times B$. Clearly the TVC is dependent on the size of the square partition and therefore the ratio ρ between the two partitions. The communication steps follow:

1. Cluster 1 needs to receive the entire square partition of matrix A from Cluster 2.

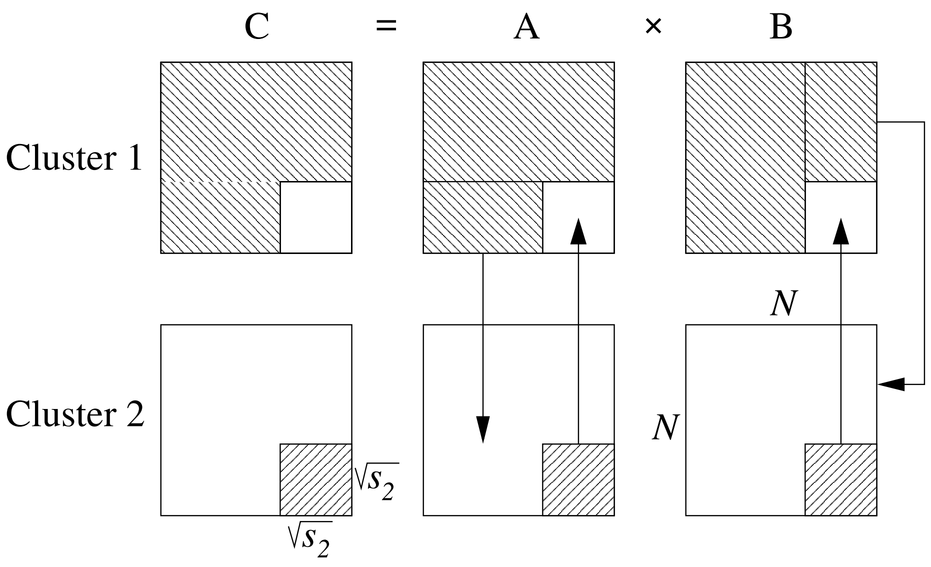


Figure 5.2: The Square-Corner Partitioning and communication steps required to carry out $C = A \times B$. The square partition is located in a corner of the matrix.

2. Cluster 1 needs to receive the entire square partition of matrix B from Cluster 2.
3. Cluster 2 needs to receive a set of partial rows of matrix A from Cluster 1.
4. Cluster 2 needs to receive a set of partial columns of matrix B from Cluster 1.

It will be shown in Section 5.5.1, based on Figures 5.1 and 5.2, that the Square-Corner Partitioning is not a special case of any Straight-Line Partitioning or vice-versa.

5.3 Serial Communications

First we will investigate the case where communication between nodes or clusters is serial—no parallel communication is allowed to occur. In fact it is this scenario that we examined in Chapter 4 and so far in this chapter (because it is in this case that the communication time is proportional to the TVC). This will not be the case if parallel communications are allowed (Section 5.4). This serial communication model is viable and realistic, particularly for grids and other geographically distributed architectures, as physical distance, other traffic and cost may force a serial connection between entities. Indeed for massively distributed projects, such as SETI@home, the number of links that connect computing entities to servers is so great that one of them has a good chance of being serial, and becoming a bottleneck that will limit the whole communication channel to the performance limitations of that bottleneck.

5.3.1 Straight-Line Partitioning

For the Straight-Line Partitioning, the TVC is always N^2, regardless of power ratio, as shown in Figure 5.1. To be consistent with three more cases that will be examined in this section and Section 5.4 we will state this as a limit:

$$\begin{aligned} TVC_{SLP} &= N^2 \\ \lim_{s_2 \to 0} TVC_{SLP} &= N^2 \end{aligned}$$

This can be expressed in terms of ρ as

$$\lim_{\rho \to \infty} TVC_{SLP} = N^2 \tag{5.4}$$

5.3.2 Square-Corner Partitioning

The TVC for the Square-Corner Partitioning is given by Equation 5.1. Additionally, we can see that the Square-Corner Partitioning is optimal if we look at the limit of Equation 5.1:

$$\begin{aligned} TVC_{SCP} &= 2 \times N \times \sqrt{s_2} \\ \lim_{s_2 \to 0} TVC_{SCP} &= 0 \end{aligned}$$

This can be expressed in terms of ρ as

$$\lim_{\rho \to \infty} TVC_{SCP} = 0 \tag{5.5}$$

Figure 5.3 Shows the Square-Corner Partitioning TVC compared to that of the Straight-Line Partitioning with serial communications for power (partition) ratios $\rho = 1:1 \to 25:1$. At a ratio of $3:1$ the partitionings are equivalent in TVC, and for $\rho = 15:1$ the Square-Corner Partitioning's TVC is one-half that of the Straight-Line Partitioning.

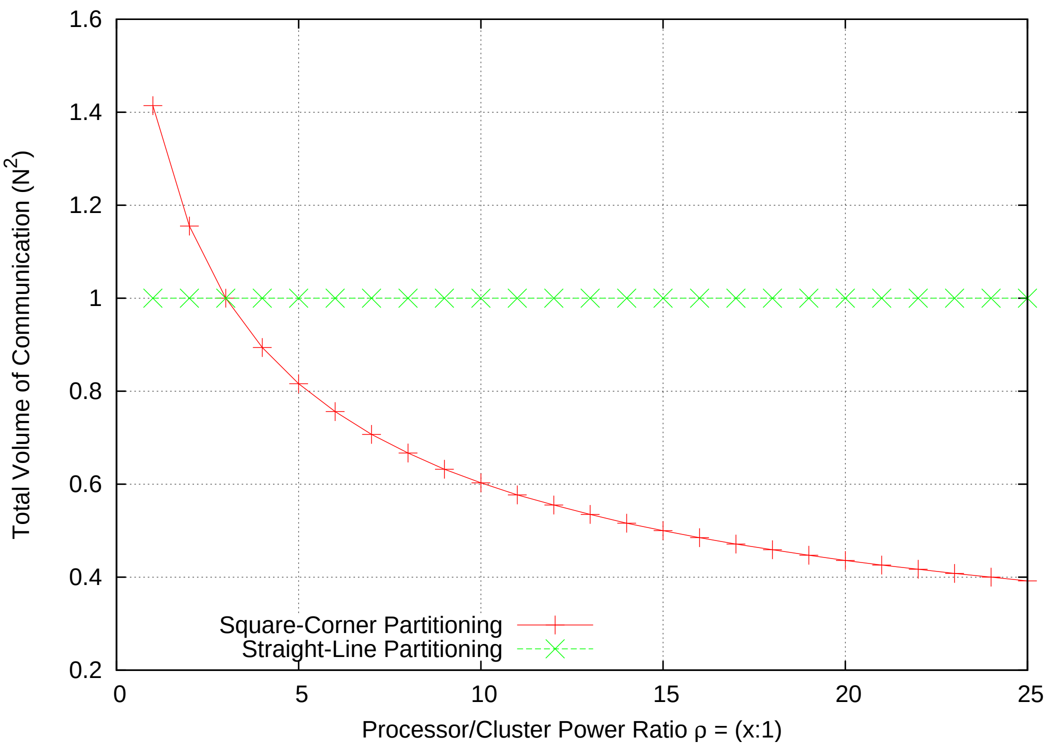


Figure 5.3: The TVC for the Square-Corner and Straight-Line Partitionings with serial communication, in terms of cluster power (partition area) ratio.

5.3.3 Hybrid Square-Corner Partitioning for Serial Communications

Since for ratios $\rho < 3 : 1$, the Square-Corner Partitioning has a greater TVC than that of the Straight-Line Partitioning, the two can be combined to create a hybrid algorithm. This Hybrid Square-Corner Partitioning for Serial Communications (HSCP-SC) is equivalent to the Square-Corner Partitioning for $\rho \geq 3$, and equivalent to the Straight-Line Partitioning for $\rho < 3$.

Figure 5.4 shows the TVC of the HSCP-SC compared to that of the Straight-Line Partitioning.

Since we know that for $\rho < 3$, the HSCP-SC and SLP are equivalent by definition, we will not compare the HSCP-SC and the SLP experimentally, but will compare the SCP to the SLP experimentally as we have been doing theoretically.

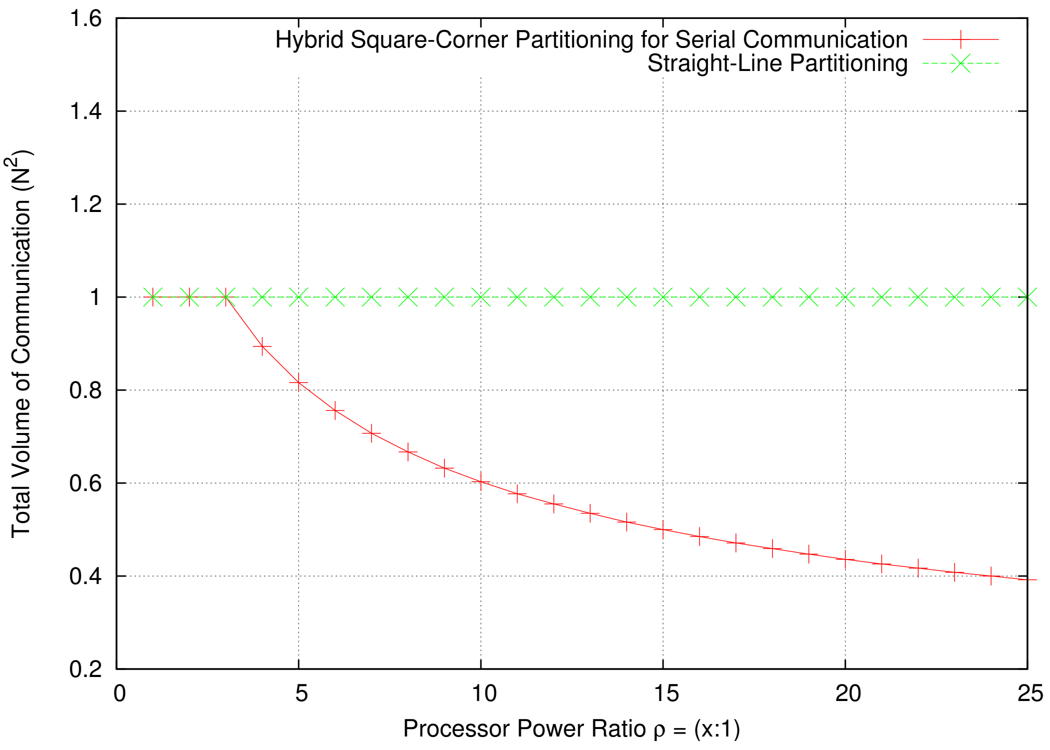


Figure 5.4: The TVC for the Hybrid Square-Corner for Serial Communications and Straight-Line Partitionings with serial communication, in terms of cluster power (partition area) ratio.

5.4 Parallel Communications

Next we will investigate the case where communication between nodes or clusters is parallel. In this case, there are two communications happening at the same time: Cluster 1 is transmitting to Cluster 2, and Cluster 2 is transmitting to Cluster 1 as per Figures 5.1 and 5.2. This is also a reasonable model, as many network interconnects do allow parallel communications, even over large geographic distance, but at a greater cost than that of serial communication links.

The equations for the total volumes of communication so far have been expressed in terms of the area of the smaller, square partition s_2. It will be advantageous to express these in terms of the ratio $\rho = \frac{s_1}{s_2} = s_1 : s_2$:

$$\begin{aligned}
\rho &= \frac{s_1}{s_2} \\
s_2 &= \frac{s_1}{\rho} \\
s_2 &= \frac{1-s_2}{\rho} \qquad \text{(see Figure 5.2)} \\
s_2 &= \frac{1}{\rho} - \frac{s_2}{\rho} \\
s_2 + \frac{s_2}{\rho} &= \frac{1}{\rho} \\
\rho \times s_2 + s_2 &= 1 \\
s_2 \times (\rho + 1) &= 1 \\
s_2 &= \frac{1}{1+\rho} \qquad (5.6)
\end{aligned}$$

5.4.1 Straight-Line Partitioning

For the Straight-Line Partitioning, the TVC from Cluster 1 to Cluster 2 ($TVC_{SLP\ \ 1\to 2}$) is always greater than that from Cluster 2 to Cluster 1 and therefore dominant (except at a 1:1 ratio where $TVC_{SLP\ \ 1\to 2} = TVC_{SLP\ \ 2\to 1} = \frac{N^2}{2}$). As per Figure 5.1:

$$\begin{aligned} TVC_{SLP\ 1\rightarrow 2} &= (1-s_2)\times N^2 &&(5.7)\\ TVC_{SLP\ 1\rightarrow 2} &\propto \left(1-\frac{1}{1+\rho}\right) \\ TVC_{SLP\ 1\rightarrow 2} &\propto \left(\frac{\rho}{1+\rho}\right) &&(5.8)\end{aligned}$$

As with the serial communication case, it can be shown that the Straight-Line Partitioning is not optimal by examining the limit of Equation 5.7:

$$\begin{aligned} TVC_{SLP} &= (1-s_2)\times N^2 \\ \lim_{s_2\rightarrow 0} TVC_{SLP} &= N^2 \end{aligned}$$

This can be expressed in terms of ρ as

$$\lim_{\rho\rightarrow\infty} TVC_{SLP} = N^2 \qquad (5.9)$$

Note that Equation 5.9 is equal to Equation 5.4, thus the Straight-Line Partitioning performs the same with serial and parallel communications.

5.4.2 Square-Corner Partitioning

For the Square-Corner Partitioning, again the volume of communication from Cluster 1 to Cluster 2 ($TVC_{SCP\ 1\rightarrow 2}$) is greater and therefore dominant for $\rho > 3:1$. For $\rho < 3:1$, $TVC_{SCP\ 2\rightarrow 1}$ is dominant. At $\rho = 3:1$ both TVC values are equivalent. As per Figure 5.2, we can determine:

$$\begin{aligned} TVC_{SCP\ 1\rightarrow 2} &= 2\times\sqrt{s_2}\times(1-\sqrt{s_2})\times N^2 &&(5.10)\\ TVC_{SCP\ 1\rightarrow 2} &\propto 2\times(\sqrt{s_2}-s_2) \\ TVC_{SCP\ 1\rightarrow 2} &\propto 2\times\left(\sqrt{\frac{1}{1+\rho}}-\frac{1}{1+\rho}\right) &&(5.11)\end{aligned}$$

$$\begin{aligned} TVC_{SCP\ 2\rightarrow 1} &= 2 \times s_2 \times N^2 \\ TVC_{SCP\ 2\rightarrow 1} &\propto 2 \times s_2 \\ TVC_{SCP\ 2\rightarrow 1} &\propto \left(\frac{2}{1+\rho}\right) \end{aligned} \tag{5.12}$$

Where $TVC_{SCP\ 1\rightarrow 2}$ is the total volume of communication moving from Cluster 1 to Cluster 2 and $TVC_{SCP\ 2\rightarrow 1}$ is the total volume of communication moving from Cluster 2 to Cluster 1.

Therefore the dominant communication for a given ratio ρ for the Square-Corner Partitioning using parallel communications is:

$$max(TVC_{SCP\ 1\rightarrow 2},\ TVC_{SCP\ 2\rightarrow 1}) =$$

$$max\left[2 \times \left(\sqrt{\frac{1}{1+\rho}} - \frac{1}{1+\rho}\right), \left(\frac{2}{1+\rho}\right)\right] \tag{5.13}$$

Figure 5.5 shows a plot of Equations 5.8 and 5.13. This illustrates the Straight-Line and Square-Corner Partitionings with parallel communications for power (partition) ratios $\rho = 1 : 1 \rightarrow 25 : 1$. Since communications are parallel, only the dominant communication is taken into account. For the SCP this is achieved with the *max* function in Equation 5.13.

The discontinuity in the TVC of the Square-Corner Partitioning at $\rho = 3$ is due to the transition from $TVC_{SCP\ 2\rightarrow 1}$ being dominant for $\rho < 3$ to $TVC_{SCP\ 1\rightarrow 2}$ being dominant for $\rho > 3$. ($TVC_{SCP\ 1\rightarrow 2} = TVC_{SCP\ 2\rightarrow 1}$ when $\rho = 3$). If we included the non-dominant terms, the curves for both $TVC_{SCP\ 2\rightarrow 1}$ and $TVC_{SCP\ 1\rightarrow 2}$ are continuous (See Figure 5.7) .

We can also show that for parallel communications the Square-Corner Partitioning is optimal, as is the case for serial communications by examining the limit of Equation 5.10:

$$\begin{aligned} TVC_{SCP} &= 2 \times \sqrt{s_2} \times (1 - \sqrt{s_2}) \times N^2 \\ \lim_{s_2 \rightarrow 0} TVC_{SCP} &= 0 \end{aligned}$$

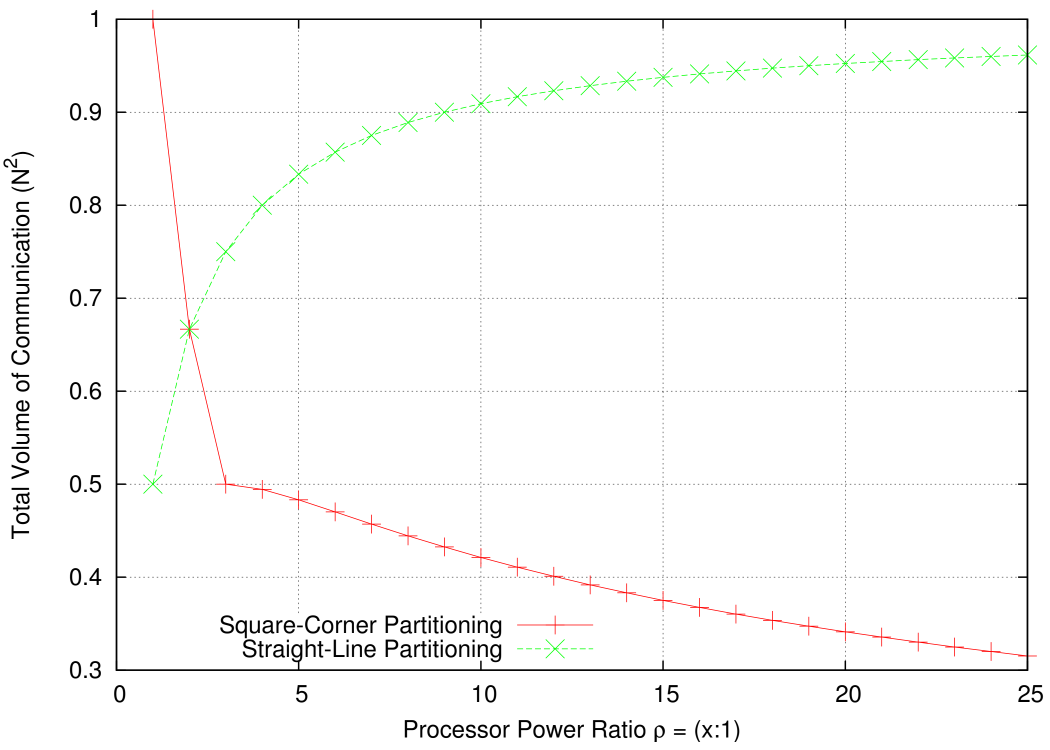


Figure 5.5: The TVC for the dominant communication of the Straight-Line and Square-Corner Partitionings utilizing parallel communications.

This can be expressed in terms of ρ as

$$\lim_{\rho \to \infty} TVC_{SCP} = 0 \tag{5.14}$$

5.4.3 Hybrid Square-Corner Partitioning for Parallel Communications

As Figure 5.5 shows, for ratios less than $\rho = 2 : 1$, the Square-Corner Partitioning has a greater TVC than that of the Straight-Line Partitioning when parallel communications are utilized. Thus the two can be combined to create a hybrid algorithm. This Hybrid Square-Corner Partitioning for Parallel Communications (HSCP-PC) is equivalent to the Square-Corner Partitioning for $\rho \geq 2$, and equivalent to the Straight-Line Partitioning for $\rho < 2$. The HSCP-PC will give the best performance of all known algorithms regardless of ρ. Figure 5.6 shows the HSCP's TVC compared to that of the Straight-Line Partitioning.

The TVC of the HSCP-PC is given by:

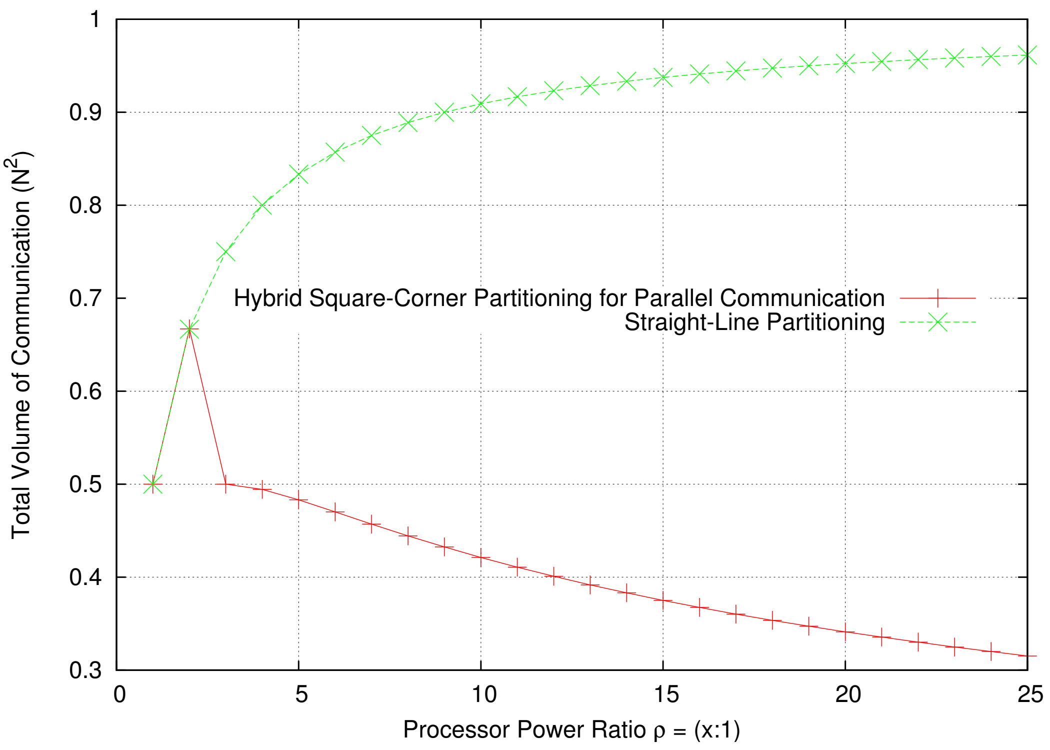


Figure 5.6: The TVC for the Hybrid Square-Corner for Parallel Communications and Straight-Line Partitionings with parallel communication, in terms of cluster power (partition area) ratio.

$$min\left[\frac{\rho}{1+\rho}, max\left[2\times\left(\sqrt{\frac{1}{1+\rho}}-\frac{1}{1+\rho}\right),\left(\frac{2}{1+\rho}\right)\right]\right] \tag{5.15}$$

As Equation 5.15 and Figure 5.7 show, the HSCP-PC is a combination of three functions, as summarized in Table 5.1.

ρ	Algorithm	Function
$1 \leq \rho < 2$	SLP	$\left(\frac{\rho}{1+\rho}\right)$
$2 \leq \rho < 3$	$\text{SCP}_{s_2 \to s_1}$	$\left(\frac{2}{1+\rho}\right)$
$3 \leq \rho$	$\text{SCP}_{s_1 \to s_2}$	$2\times\left(\sqrt{\frac{1}{1+\rho}}-\frac{1}{1+\rho}\right)$

Table 5.1: The HSCP-PC algorithm in terms of ρ. The algorithm is composed of the SLP and different components of the SCP depending on the value of ρ. $\text{SCP}_{x \to y}$ is the TVC of the SCP from partition x to partition y.

Figure 5.7 shows the HSCP-PC's TVC compared to that of the Straight-Line

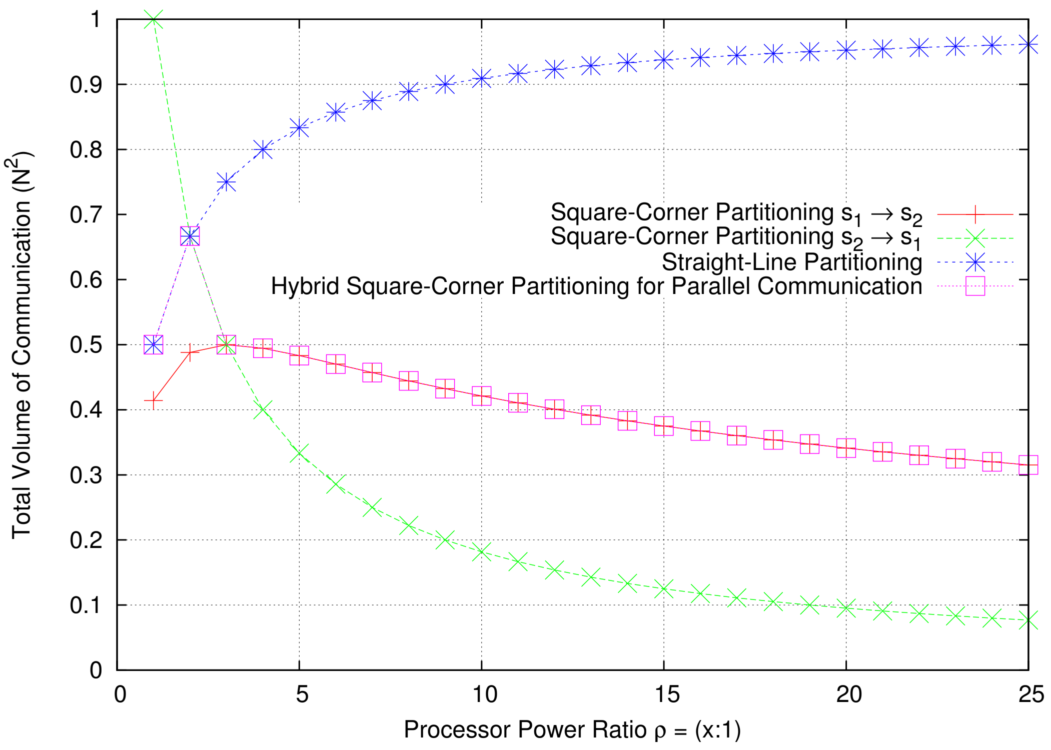


Figure 5.7: Parallel Communications: The TVC for $\text{SCP}_{s_1 \rightarrow s_2}$, $\text{SCP}_{s_2 \rightarrow s_1}$, SLP and HSCP-PC, where $\text{SCP}_{x \rightarrow y}$ is the TVC of the SCP from partition x to partition y.

Partitioning and the two components of the Square-Corner Partitioning.

Since we know that for $\rho < 2$, the HSCP-PC and SLP are equivalent by definition, we will not compare the HSCP-SC and the SLP experimentally, but will compare the SCP to the SLP experimentally as we have been doing theoretically.

5.5 Choosing a Corner For the Square Partition

We choose a corner for the position of the square partition for several reasons of convenience as follows. The location does not affect the TVC. Placing the square partition in the corner position of the matrix:

1. Minimizes the number of communication steps necessary, and reduces the complexity of the communication schedule discussed in Section 5.5.1.
2. Allows the use of the SHP metric as proportional to the TVC as discussed in Section 5.5.2.
   - In showing that the SHP is proportional to the TVC for the Square-Corner Partitioning (as it is for the rectangular, or Straight-Line Partitioning) we come up with a new metric, the total number of row and column interrupts, I. This is explored in section 5.5.3.
3. Maximizes an area of the matrix which can allow for the overlapping of communication and computation which will be discussed in Section 5.5.4.
4. Maximizes the potential size of multiple partitions as will be discussed in Chapters 6 and 8.

5.5.1 Minimizing the Number of Communication Steps

Placing the square partition in the corner of the matrix minimizes the number of communication steps necessary to calculate $C = A \times B$. Figure 5.2 shows that the Square-Corner Partitioning requires a total of four communication steps. Figure 5.8 shows a partitioning similar to the Square-Corner Partitioning and the necessary data movements required to calculate a matrix product $C = A \times B$ with the square partition adjacent to one of the sides of the matrix.

As Figure 5.8 shows, the number of communication steps in this case is five. Just as in the case where the square partition is located in one of the corners of the matrix, both square partitions and a number of partial rows and partial columns need to be communicated.

Figure 5.9 shows a partitioning similar to the Square-Corner Partitioning and the necessary data movements required to calculate a matrix product $C = A \times$

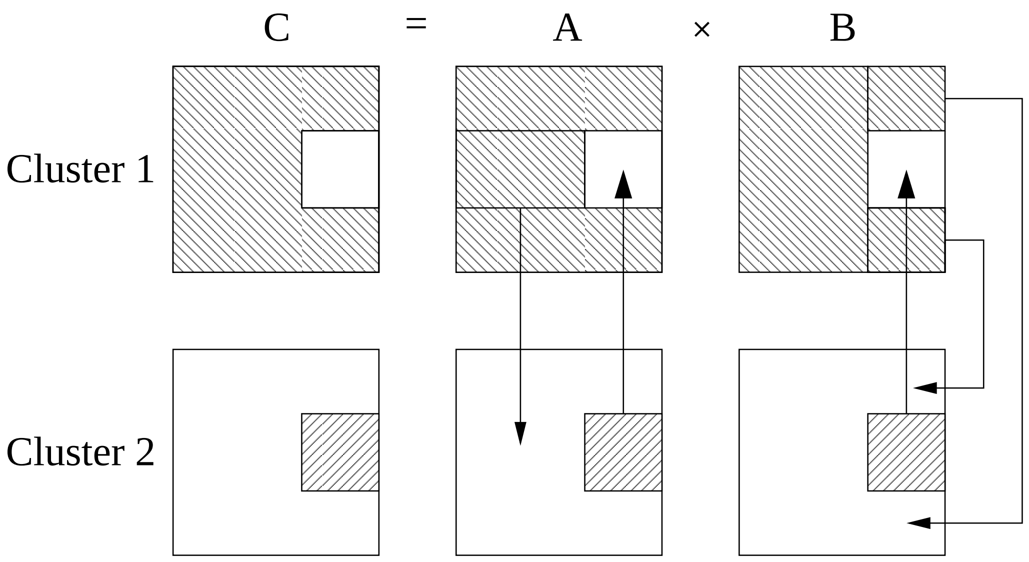


Figure 5.8: A partitioning similar to the Square-Corner Partitioning and the communication steps required to carry out $C = A \times B$. The square partition is located adjacent to one of the sides of the matrix.

B with the square partition in the center of the matrix, not adjacent to any sides. As Figure 5.9 shows, the number of communication steps is six. Just as in the case where the square partition is located in one of the corners of the matrix, both square partitions and a number of partial rows and partial columns need to be communicated.

A very important point to note is that the Square-Corner Partitioning requires at least four communication steps to compute $C = A \times B$. All Straight-Line Partitionings require only two. *Thus the algorithms are distinct and one is not a special case of the other*.

5.5.2 Sum of Half Perimeters - A Metric

In Chapter 4 we used the sum of half perimeters as a metric which is proportional to the total volume of communication. This is common practice for rectangular partitionings but demands closer inspection for the non-rectangular case.

First let us use two methods to determine the sum of half perimeters for the non-rectangular (Square-Corner) solution on the unit-square.

- Add the entire perimeter of the partitions s_1 and s_2 and divide the sum by two. Equation 4.2 gives us $\hat{C} = 2 + 2 \times \sqrt{s_2}$.
- Beaumont *et al.* (2002b) provides the second way: It is the length of the

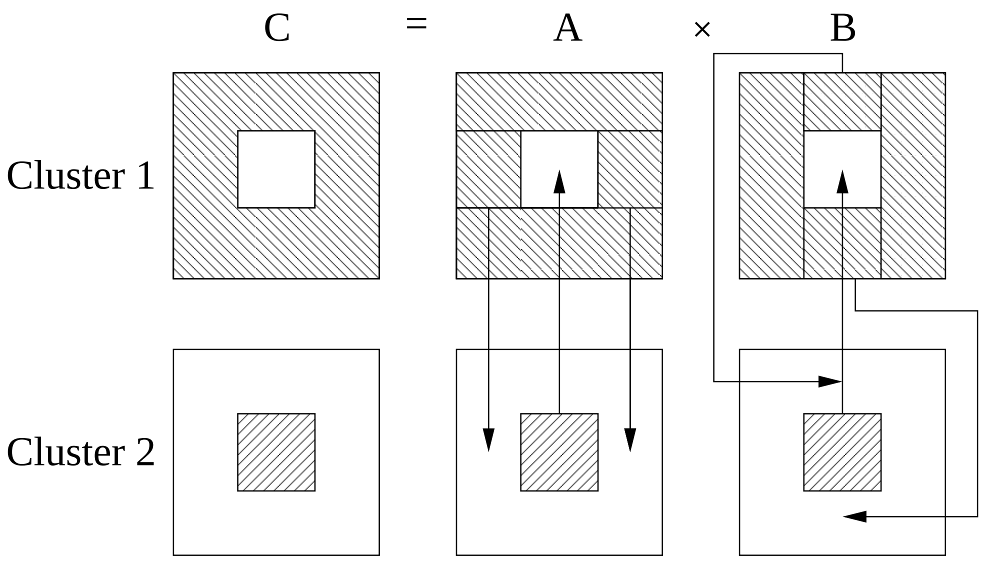


Figure 5.9: A partitioning similar to the Square-Corner Partitioning and the communication steps required to carry out $C = A \times B$. The square partition is not adjacent to any sides of the matrix.

lines drawn to make the partition(s) plus 2. This also results in $\hat{C} = 2 + 2 \times \sqrt{2}$.

Thus the unit square SHP for the Square-Corner Partitioning is $(2 + 2 \times \sqrt{s_2})$. For a real matrix of size N, the SHP is given by Equation 5.2.

$$\hat{C} = 2 \times N + 2 \times \sqrt{s_2} \tag{5.16}$$

As we have seen in Equations 5.1 and 5.16, the TVC and SHP for the Square-Corner Partitioning are expressed in terms of $\sqrt{s_2}$. We can in turn define $\sqrt{s_2}$ in terms of the cluster power ratio, ρ in Equation 5.17. As always we normalize this ratio so that the power of the slower cluster is equal to 1, so a ratio of ρ is understood to be a ratio of $\rho : 1$.

$$\sqrt{s_2} = \frac{N}{\sqrt{\rho + 1}} \tag{5.17}$$

To study the proportionality of the SHP and TVC relationship, Table 5.1 lists some actual SHP/TVC ratios for the Square-Corner Partitioning, along with the partitionings in Figures 5.8 (square partition adjacent to one side) and 5.9 (square adjacent to no sides). The value of $\sqrt{s_2}$, and therefore the ratio between the two partitions, is varied. Note that we are working on the unit square.

It is desirable to make a meaningful comparison of the proportionality of

$\sqrt{s_2}$	SHP/TVC Square-Corner	Figure 5.8	Figure 5.9
0.1	11.0	11.5	12.0
0.25	5.0	5.5	6.0
0.5	3.0	3.5	4.0
0.75	$2.\overline{66}$	$2.8\overline{33}$	$3.\overline{33}$
0.9	$2.\overline{11}$	$2.6\overline{11}$	$3.\overline{11}$

Table 5.2: SHP/TVC values for the Square-Corner Partitioning and two Square-Corner-Like partitionings (Figures 5.8 and 5.9), $\sqrt{s_2} \in (0.1, 0.25, 0.5, 0.75, 0.9)$, on the unit square.

the SHP/TVC ratio for the Square-Corner and "Square-Corner-Like" Partitionings and the SHP/TVC ratio for the Straight-Line Partitioning. Initially this proves troublesome, as varying the ratio between only two partitionings for the Straight-Line Partitioning will yield the same ratio regardless of SHP and TVC values, as both are constant. The SHP is always 3, and the TVC is always 1 on the unit square. Table 5.3 lists values for the Straight-Line Partitioning SHP/TVC ratios for a varying number of *partitions*. To keep calculations simple, each partition is given an equal area, and the partition numbers are kept to perfect squares.

Number of Partitions	SHP/TVC
2	3
4	2
9	3/2
16	4/3
25	5/4
36	6/5

Table 5.3: SHP/TVC values the Straight-Line Partitioning for six different numbers of partitions, $p \in (2, 4, 9, 16, 25, 36)$, on the unit square.

Interestingly, plotting the SHP and TVC while varying $\sqrt{s_2}$ for the Square-Corner and Square-Corner-Like Partitionings (keeping p constant), but varying the number of partitions p for the Straight-Line Partitioning, we can see their relative relationships. This is shown in Figure 5.10.

Figure 5.10 shows that the Square-Corner Partitioning and Straight-Line Partitionings have *equivalent* SHP/TVC ratios. Those for the other two Square-Corner-Like Partitionings are also linear, and have the same intercepts as the other lines, only their slope varies.

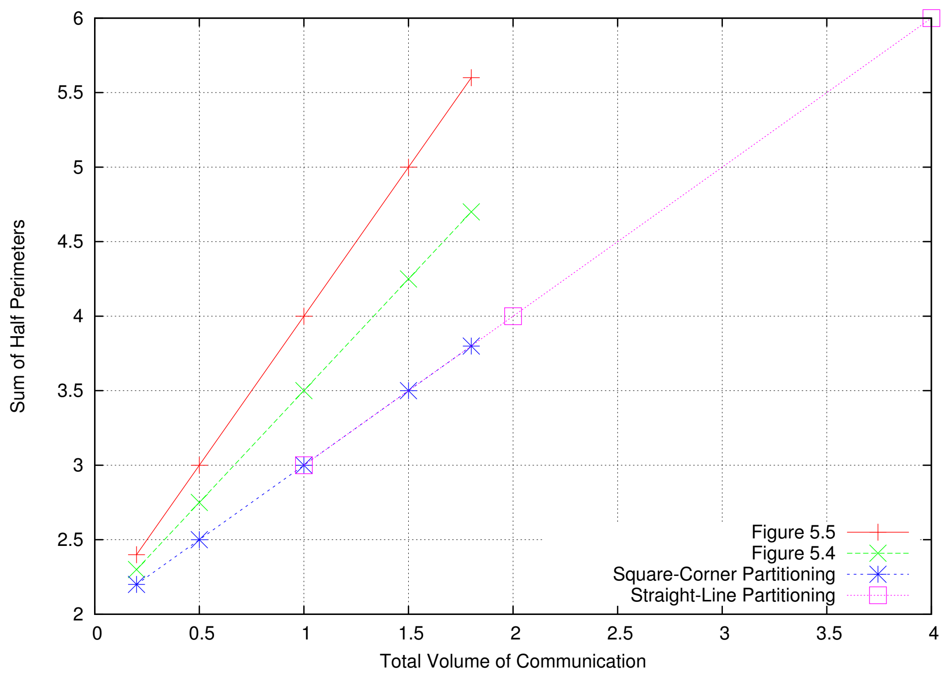


Figure 5.10: A plot of the SHP vs. TVC values for Square-Corner and Square-Corner-Like Partitionings (Figures 5.8 and 5.9), and the Straight-Line Partitioning on the unit square. For the Square-Corner and Square-Corner-Like Partitionings the value of $\sqrt{s_2} \in (0.1, 0.25, 0.5, 0.75, 0.9)$ is varied. For the Straight-Line Partitioning the number of partitions $p \in (2, 4, 9)$ is varied.

The intercept for all lines appears to be $(0, 2)$. This corresponds to a TVC of 0 (obviously optimal), and a SHP of 2, also optimal (The SHP of the unit-square itself is 2). This however is not the case if we inspect the region below SHP $= 3$ with greater detail. It is seen that once the SHP reaches 3, the Straight-Line Partitioning cannot proceed towards the optimal $(0, 2)$. This is because the SHP cannot be less than 3, and therefore the TVC can never be below 1. However the Square-Corner Partitioning carries no such restriction and does approach, in the limit SHP $\to 0$, the optimal value of TVC $= 0$. Interestingly, the other two Square-Like Partitionings also approach the optimal value of TVC $= 0$.

5.5.3 A New TVC Metric - Row and Column Interrupts

In section 5.5.1 we saw that for each communication step a number of partial rows and columns had to be communicated. In fact, all communications make up partial rows and columns. Although once all communications have completed it is entire rows and columns that have been communicated in aggregate. This holds for the Straight-Line Partitioning as well. This leads us to conclude that a more general metric, proportional to the TVC, is the number of rows and columns interrupted by partitions boundaries.

For the Square-Corner and Square-Corner-Like Partitionings, the number of rows and columns interrupted is $\sqrt{s_2}$ rows and $\sqrt{s_2}$ columns for the square partition, and the same for the polygonal partition for a total of $4 \times \sqrt{s_2}$. The TVC is $2 \times N \times \sqrt{s_2}$. For the Straight-Line Partitioning, the number of rows interrupted is N for the first partition and N for the second for a total of $2 \times N$. The TVC is N^2. From this the following observations can be made:

- For the Square-Corner and Square-Corner-Like Partitionings,

$$\frac{\text{TVC}}{I} = \frac{2 \times N \times \sqrt{s_2}}{4 \times \sqrt{s_2}} = \frac{N}{2}$$

- For the Straight-Line Partitioning,

$$\frac{\text{TVC}}{I} = \frac{N^2}{2 \times N} = \frac{N}{2}$$

where I is the total number of rows and columns interrupted by each partition in each partitioning, given by Equation 5.18.

$$I = \sum_{i=1}^{p} (r_i + c_i) \tag{5.18}$$

where p is the number of partitions, r is the number of rows interrupted by partition i, and c is the number of columns interrupted by partition i.

We conclude, but do not prove, that Equation 5.18 is a more general metric than the SHP for determining the relative TVC of matrix partitionings.

This is due to the fact that the SHP fails to be proportional to the TVC for polygons whose perimeters are not equal to that of a bounding rectangle. In Figure 5.8 the partition owned by Cluster 1 is a polygon which falls into this category. In Figure 5.9 the partition owned by Cluster 1 is not even polygonal, but a polygon with a “hole“ cut in it. Nonetheless if one considers the perimeter of the hole to contribute to the SHP as we did in section 5.5.2, the SHP does behave linearly and converge on the optimal TVC. Thus the SHP does seem appropriate as a metric, however the sum of row and column interrupts, I, is again more appropriate, as it matches exactly the I value of the other Square-Corner and Square-Corner-Like Partitionings, when their TVC values are equivalent.

The non-rectangular partition in the Square-Corner Partitioning works perfectly with both the SHP and I metrics, because the SHP is equal to the SHP

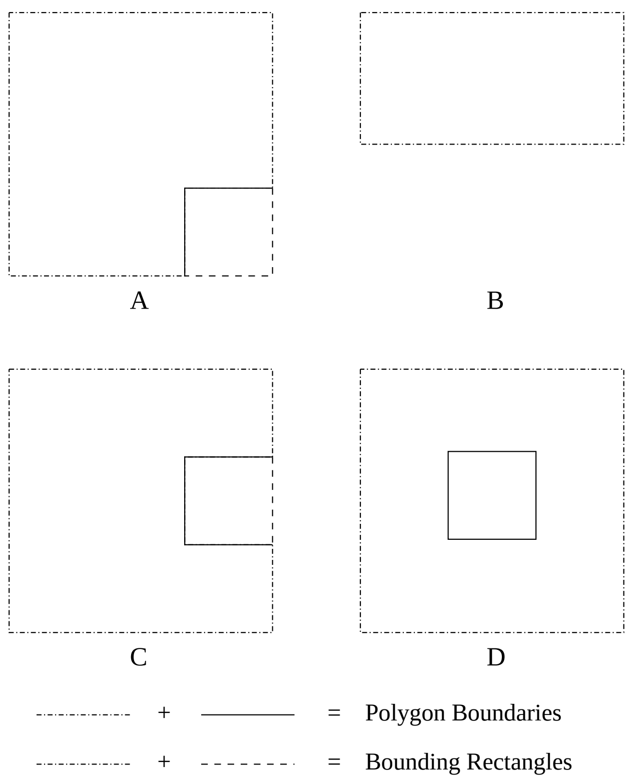


Figure 5.11: Four polygonal partitionings seen so far, along with their bounding rectangles. Partitions A and B have a TVC proportional to the SHP. C and D do not. All partitions have a TVC proportional to their *I* value, the total number of row and column interrupts.

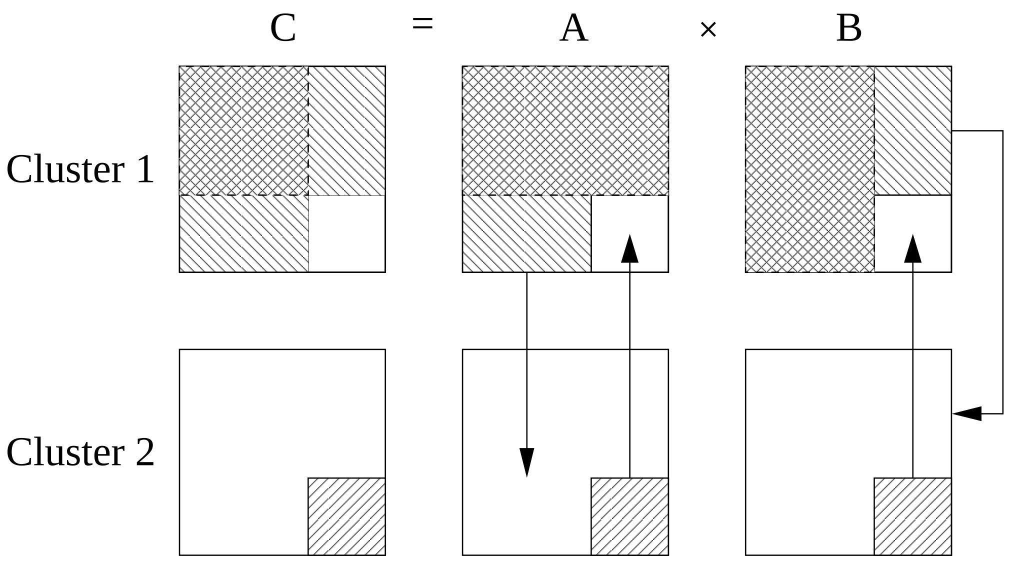


Figure 5.12: The Square-Corner Partitioning showing the area (hatched) of Cluster 1's *C* matrix which is immediately calculable. No communication is necessary to calculate this area. This is possible because Cluster 1 already owns the areas of *A* and *B* necessary to calculate *C* (also hatched).

of the partition's bounding rectangle. (Also note that the same is true for all rectangular partitions.) Figure 5.11 shows the polygons so far investigated. A is the non-rectangular partition from the Square-Corner Partitioning, B is a rectangular partitioning from the Straight-Line Partitioning. C and D are from the Square-Corner-Like Partitionings. The TVC of A and B is proportional to the SHP while that of C and D are not. At the same time all partitionings A,B,C and D have an *I* value proportional to their TVC.

5.5.4 Overlapping Communication and Computation

The Square-Corner Partitioning has another advantage over the Straight-Line Partitioning. There is always a part of the product matrix C which can be immediately calculated. This is shown in Figure 5.12 as the hatched area. The hatched areas of matrices *A* and *B* are those required to calculate area *C* with no communication. Cluster 1 owns these areas from the outset. In Chapter 6 this will be shown to greatly reduce execution times. Placing the square partition in one corner of the matrix maximizes the area of this immediately calculable sub-partition, in the corner opposite. We explore this further in Sections 6.1.4 and 6.2.

5.6 HCL Cluster Simulations

To experimentally verify algorithm 5.1, the Square-Corner Partitioning, we implemented it and the Straight-Line Partitioning in Open-MPI (Gabriel *et al.*, 2004). The experiments were carried out on two identical machines on the HCL Cluster (hcl03 and hcl04), (See Chapter 3). This was done so we could focus solely on the partitioning method without worrying about any contributions made by architectural differences. Local matrix computations utilize ATLAS (Whaley and Dongarra, 1999). The machines were connected with a switch allowing the bandwidth between the nodes to be specified, up to 1Gb/s.

5.6.1 Serial Communications

Since the HCL switches allow parallel communications, we simulated a serial communication link in code. We do this by forcing all communications from Processor 1 to Processor 2 to complete before the communications going from Processor 2 to Processor 1 commence. This is done using `MPI_Barrier()` calls. Both partitionings carry out all communications first, then all computations. Thus preliminarily there is no communication/computation overlap. All times are averaged over five runs.

The ratio of speeds between the two nodes was varied by slowing down the CPU of one node relative to the other using a CPU limiting program as proposed in (Canon and Jeannot, 2006). This program supervises a specified processes and using the `/proc` pseudo-filesystem, forces the process to sleep when it has used more than a specified fraction of CPU time. The process is then woken when enough idle CPU time has elapsed for the process to resume. Sampled frequently enough, this can provide a fine level of control over the fraction of CPU time used by the process. Comparison of the run-times of each node confirmed that this method results in the desired ratios well within 2%.

5.6.2 Comparison of Communication Times

We ran matrix matrix multiplications using the Square-Corner Partitioning and the Straight-Line Partitioning for power ratios ranging from 1 : 1 to 1 : 25 and for bandwidth values ranging from 50Mb/s to 1Gb/s. In all cases other

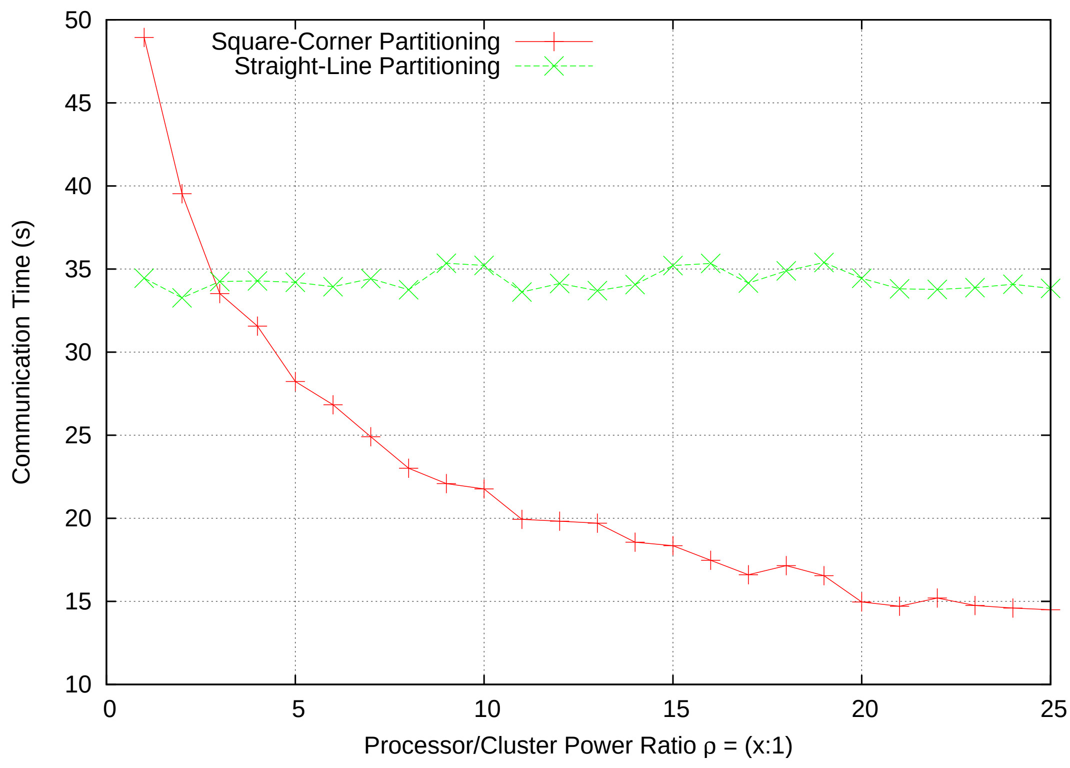


Figure 5.13: Average communication times for the Square-Corner and Straight-Line Partitionings using serial communications. Network bandwidth is 100Mb/s, $N = 4,500$.

than ratios of $1:1, 1:2$, and $1:3$, the total communication time for the Square-Corner Partitioning was less than that of the Straight-Line Partitioning.

Figure 5.13 shows the communication times for the Square-Corner and Straight-Line Partitionings for ratios $1:1-1:25$ and a network bandwidth of 100Mb/s. The shape of the curves in this figure reflect the TVC relationship between the Square-Corner and Straight-Line Partitionings as theoretically shown in Figure 5.3.

Figure 5.14 shows the communication times for the Square-Corner and Straight-Line Partitionings for ratios $1:1-1:25$ and a network bandwidth of 500Mb/s. Again the experimental results match the theoretical predictions.

The most important feature of all communication plots is that there is the predicted "crossover" between the Square-Corner Partitioning and Straight-Line Partitioning at a ratio of $\rho = 3:1$, and for all greater ratios the Square-Corner Partitioning has a lower TVC and therefore communication time. In fact the gap between the two partitionings continues to widen with increasing ratio, because $\lim_{\rho\to\infty} TVC_{SLP} = N^2$ (Equation 5.4), and $\lim_{\rho\to\infty} TVC_{SCP} = 0$ (Equation 5.5).

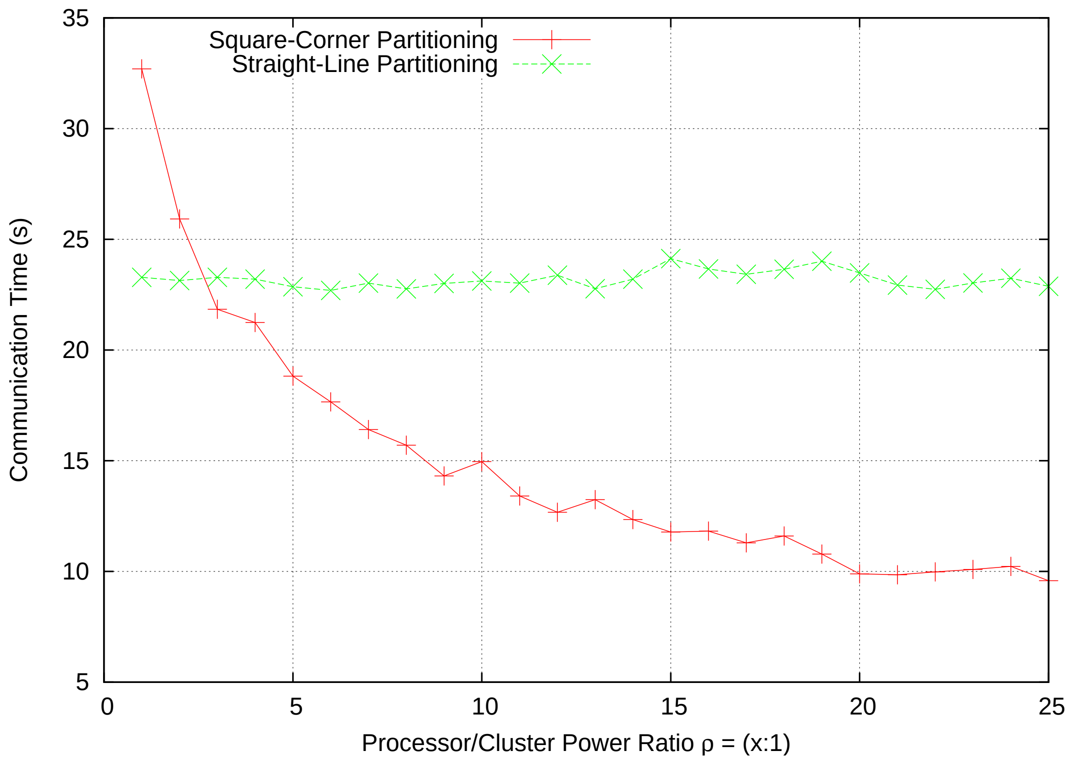


Figure 5.14: Average communication times for the Square-Corner and Straight-Line Partitionings using serial communications. Network bandwidth is 500Mb/s, $N = 4,500$.

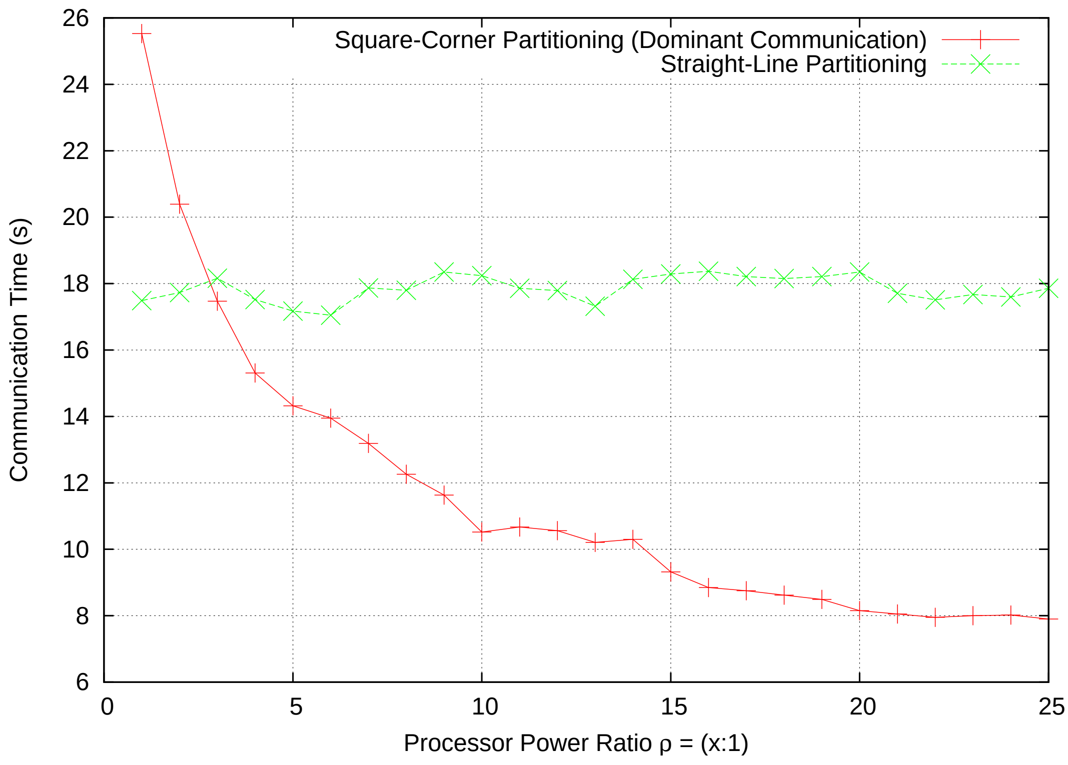


Figure 5.15: Average communication times for the Square-Corner and Straight-Line Partitionings using serial communications. Network bandwidth is 1Gb/s, $N = 4,500$.

5.6.3 Comparison of Total Execution Times

The objective of the Square-Corner Partitioning is to reduce the inter-cluster communication time, resulting in a lower total execution time compared to all other partitionings when the number of partitions, $p = 2$. Since the total execution time is dependent on communication and computation time, any savings in total execution time will be dependent on how dominant communication time is in the overall execution time. It can be seen comparing the communication and execution times that the reduction in communication times directly impacts the execution times.

Figures 5.16, 5.17 and 5.18 show the total execution times for the Square-Corner and Straight-Line Partitionings for ratios $1 : 1 - 1 : 25$ and network bandwidths of 100Mb/s, 500Mb/s and 1Gb/s respectively, using serial communications. The crossover at $\rho = 3 : 1$ which was predicted theoretically and observed experimentally in the communication times is also present in the execution times. For all ratios $\rho > 3 : 1$, the Square-Corner Partitioning out performs the Straight-Line Partitioning.

As bandwidth increases, the ratios where both the Square-Corner and Straight-Line Partitionings are faster than the sequential time (performing the multiplication on the fastest processor only) increase. For the Square-Corner Partitioning, this ratio increases from approximately 17 : 1 to 23 : 1 as the bandwidth increases from 100Mb/s to 1Gb/s.

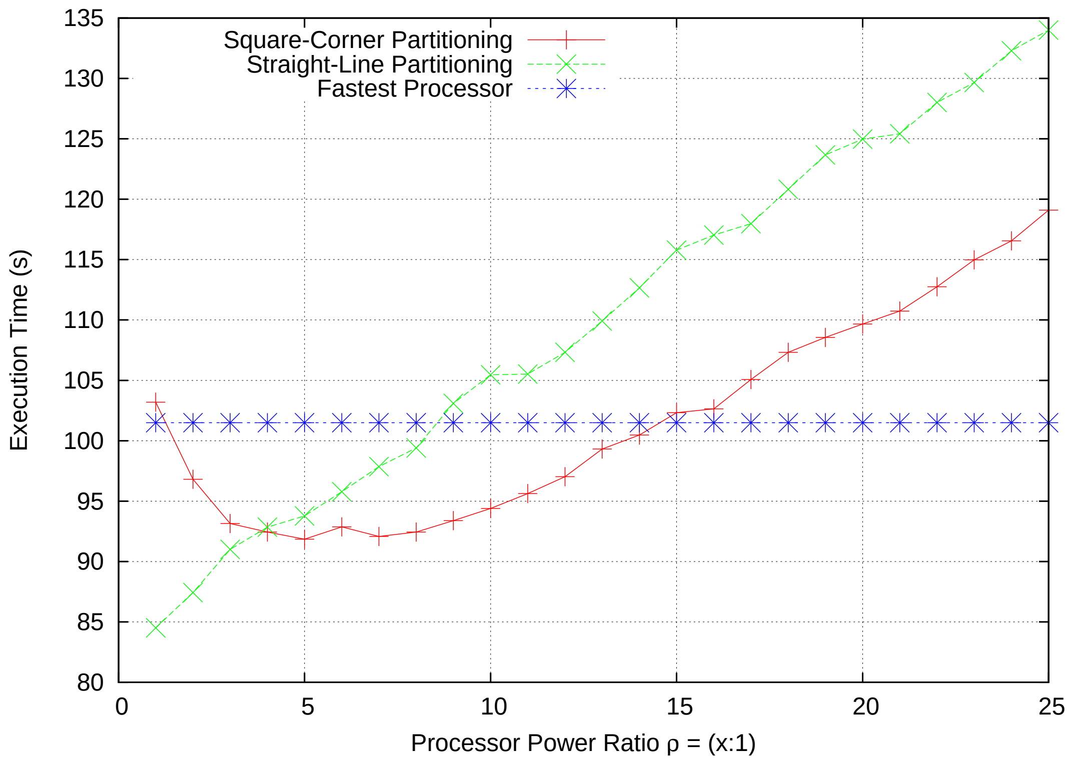


Figure 5.16: Average execution times for the Square-Corner and Straight-Line Partitionings using serial communications. Network bandwidth is 100Mb/s, $N = 4,500$.

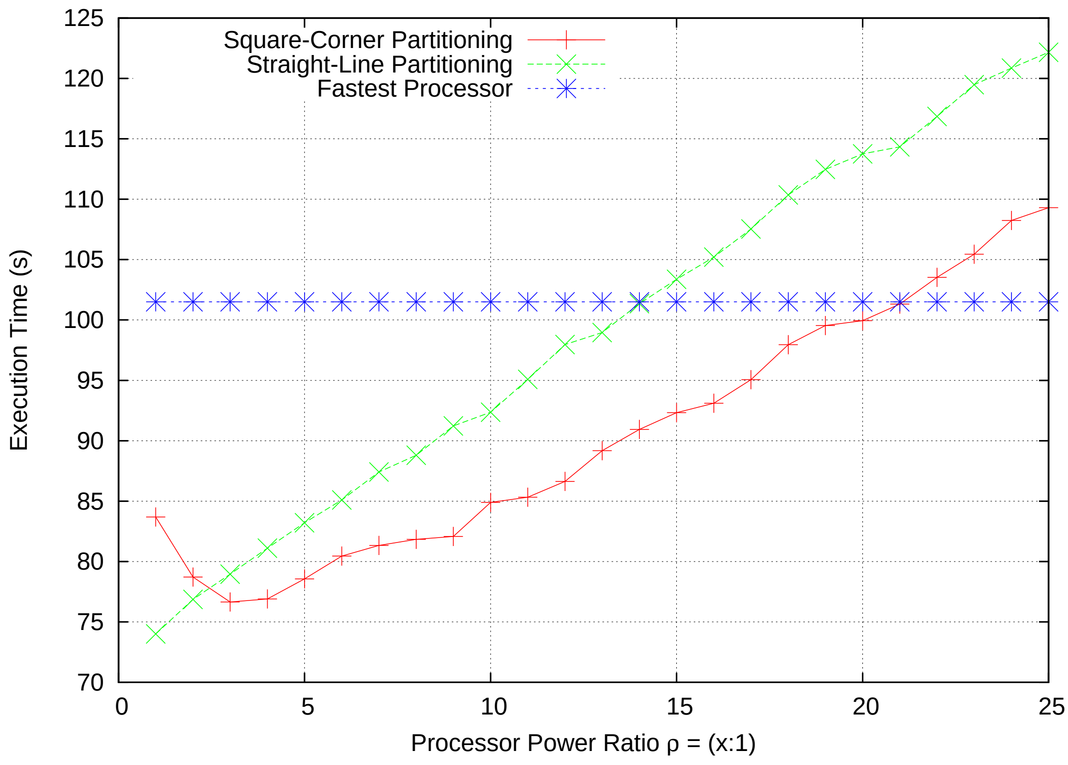


Figure 5.17: Average execution times for the Square-Corner and Straight-Line Partitionings using serial communications. Network bandwidth is 500Mb/s, $N = 4,500$.

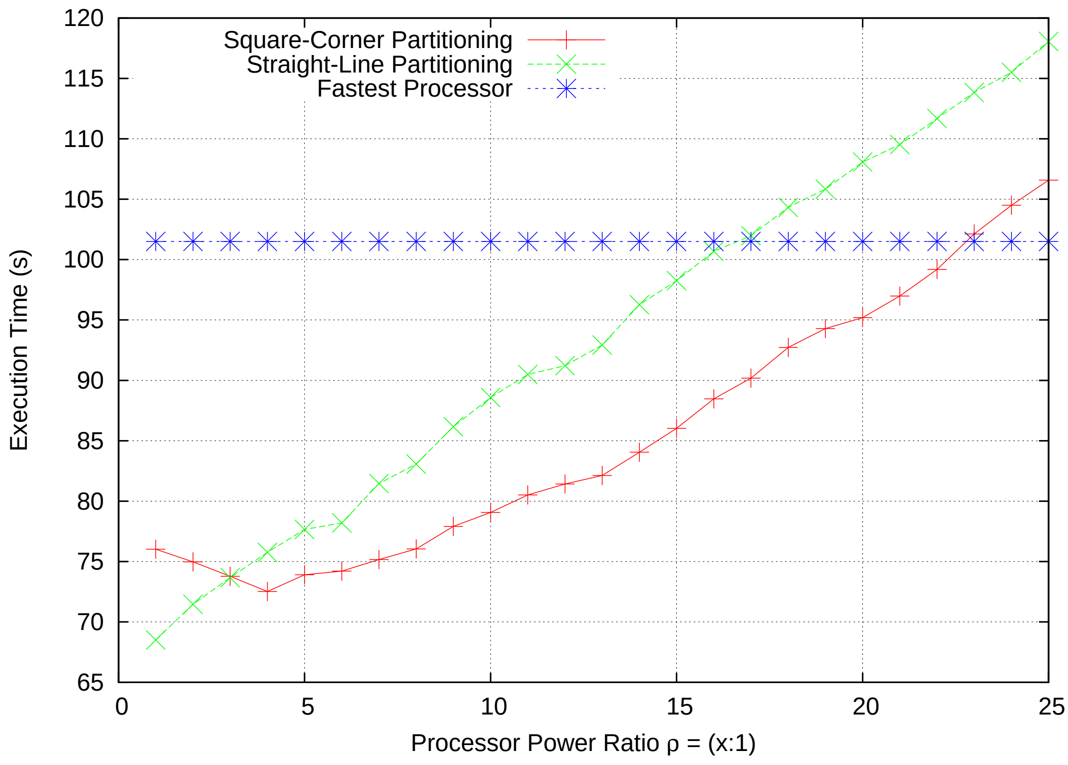


Figure 5.18: Average execution times for the Square-Corner and Straight-Line Partitionings using serial communications. Network bandwidth is 1Gb/s, $N = 4,500$.

5.6.4 Parallel Communications

In this section we explore the performance of the Square-Corner Partitioning when parallel communications are utilized. All experimental parameters and techniques are the same as Section 5.6.1. The only difference is a modified communication schedule, which is described in the following section.

5.6.5 Comparison of Communication Times

We ran matrix matrix multiplications using the Square-Corner Partitioning and the Straight-Line Partitioning for power ratios ranging from 1 : 1 to 1 : 25 and for bandwidth values ranging from 50Mb/s to 1Gb/s. In all cases where $\rho > 2 : 1$ The total communication time for the Square-Corner Partitioning was less than that of the Straight-Line Partitioning, as predicted by theory.

All communications are carried out before computations begin. Non-Blocking `MPI_ISend()` and `MPI_IRecv()` communications are used, followed by `MPI_Wait()` statements. This is done to give control of the communication scheduling to the network layer allowing Ethernet parallelism. The only `MPI_Barrier` is between the communication and computation code segments.

Figures 5.19, 5.20, and 5.21 show the communication times for the Square-Corner and Straight-Line Partitionings for ratios $1 : 1 - 1 : 25$ and network bandwidths of 100Mb/s, 500Mb/s, and 1Gb/s respectively. The shape of the curves in these figures reflect the TVC relationship between the Square-Corner and Straight-Line Partitionings as theoretically shown in Figure 5.3. Note that with parallel communications this results in a lower communication time when $\rho > 2 : 1$.

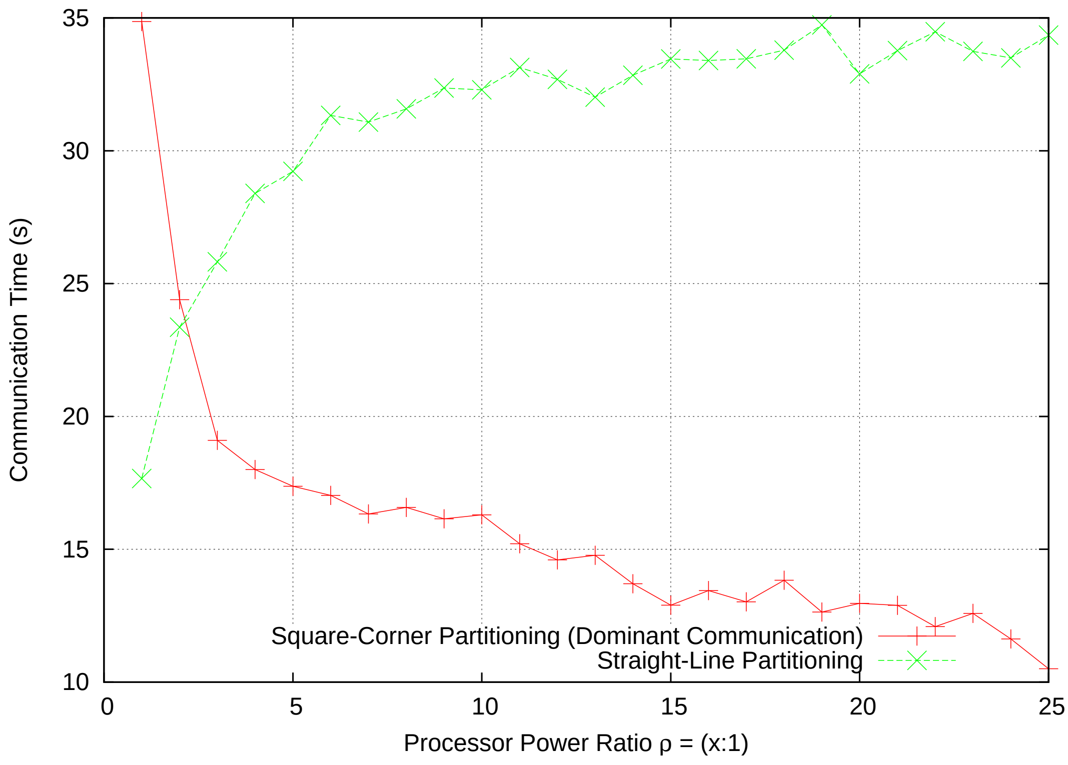


Figure 5.19: Average communication times for the Square-Corner and Straight-Line Partitionings using parallel communications. Network bandwidth is 100Mb/s, $N = 4,500$.

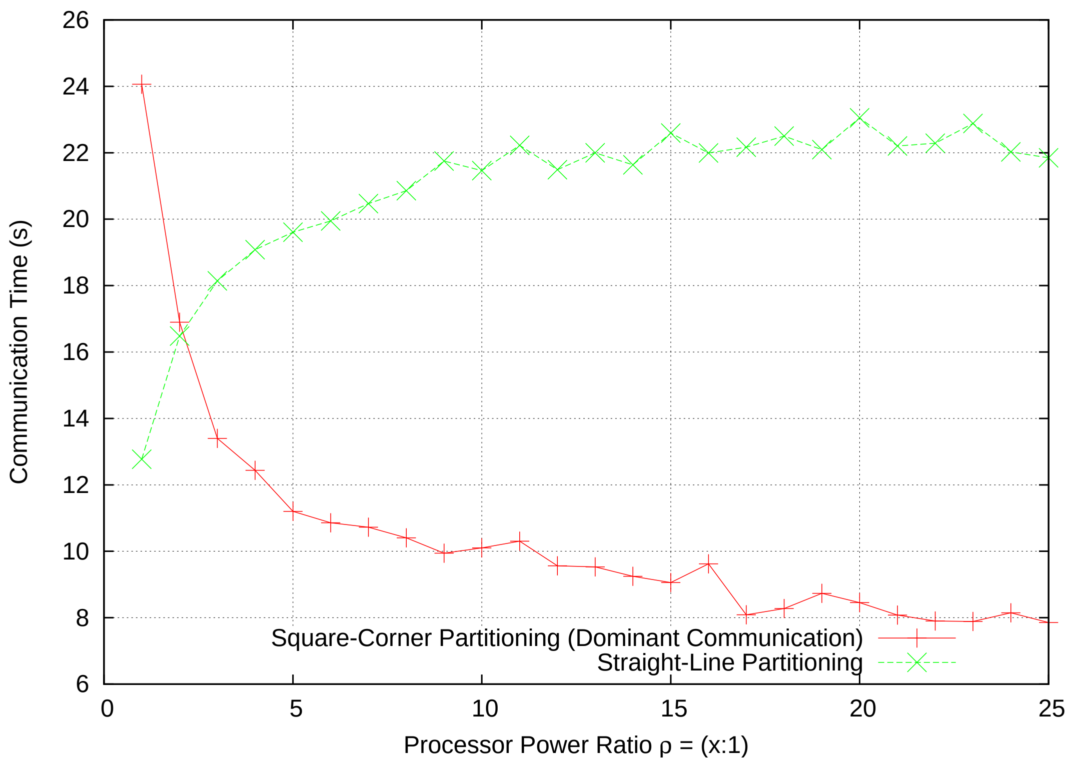


Figure 5.20: Average communication times for the Square-Corner and Straight-Line Partitionings using parallel communications. Network bandwidth is 500Mb/s, $N = 4,500$.

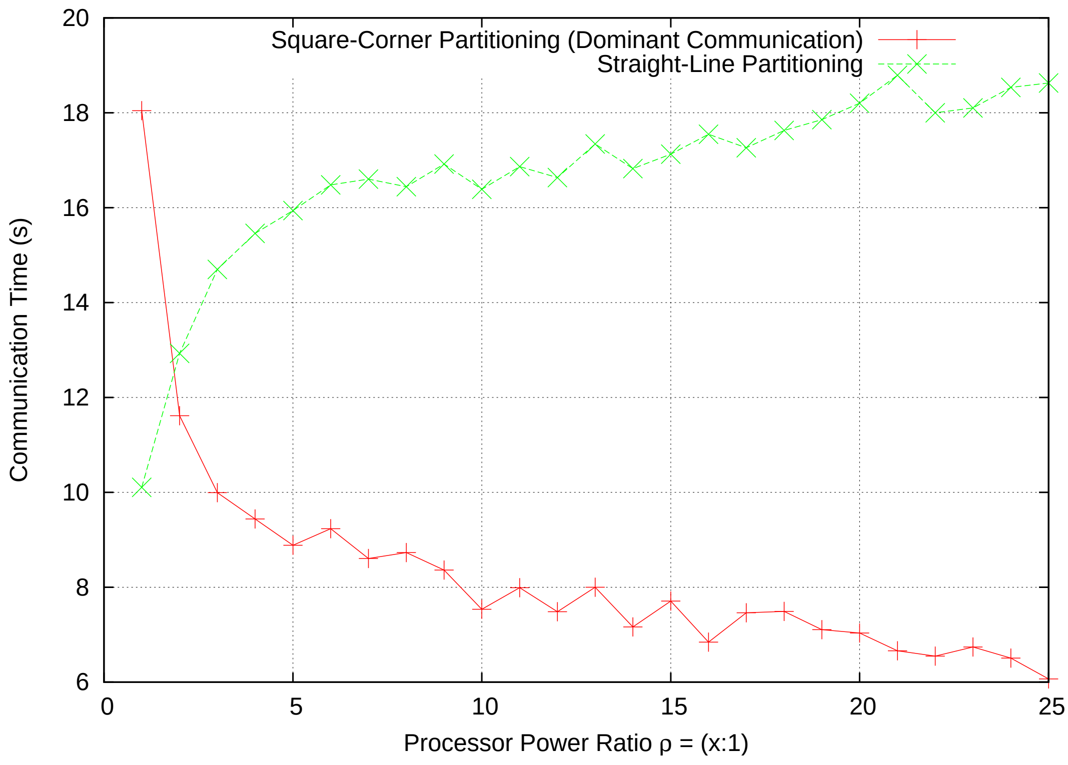


Figure 5.21: Average communication times for the Square-Corner and Straight-Line Partitionings using parallel communications. Network bandwidth is 1Gb/s, $N = 4,500$.

5.6.6 Comparison of Total Execution Times

The objective of the Square-Corner Partitioning is to reduce the inter-cluster communication time, resulting in a lower total execution time compared to all other partitionings when the number of partitions, $p = 2$. Since the total execution time is dependent on communication and computation time, any savings in total execution time will be dependent on how dominant communication time is in the overall execution time. It can be seen comparing the communication and execution times that the reduction in communication times directly impacts the execution times.

Figures 5.22, 5.23 and 5.24 show the total execution times for the Square-Corner and Straight-Line Partitionings for ratios $1 : 1 - 1 : 25$ and network bandwidths of 100Mb/s, 500Mb/s and 1Gb/s respectively. These figures also show the execution time for the fastest processor executing the multiplication time sequentially on the fastest processor.

As bandwidth increases, the ratio where both the Square-Corner and Straight-Line Partitionings are faster than the sequential time (performing the multiplication on the fastest processor only) increases. For the Square-Corner Partitioning, this ratio increases from approximately $19 : 1$ to $22 : 1$ as the bandwidth increases from 100Mb/s to 1Gb/s.

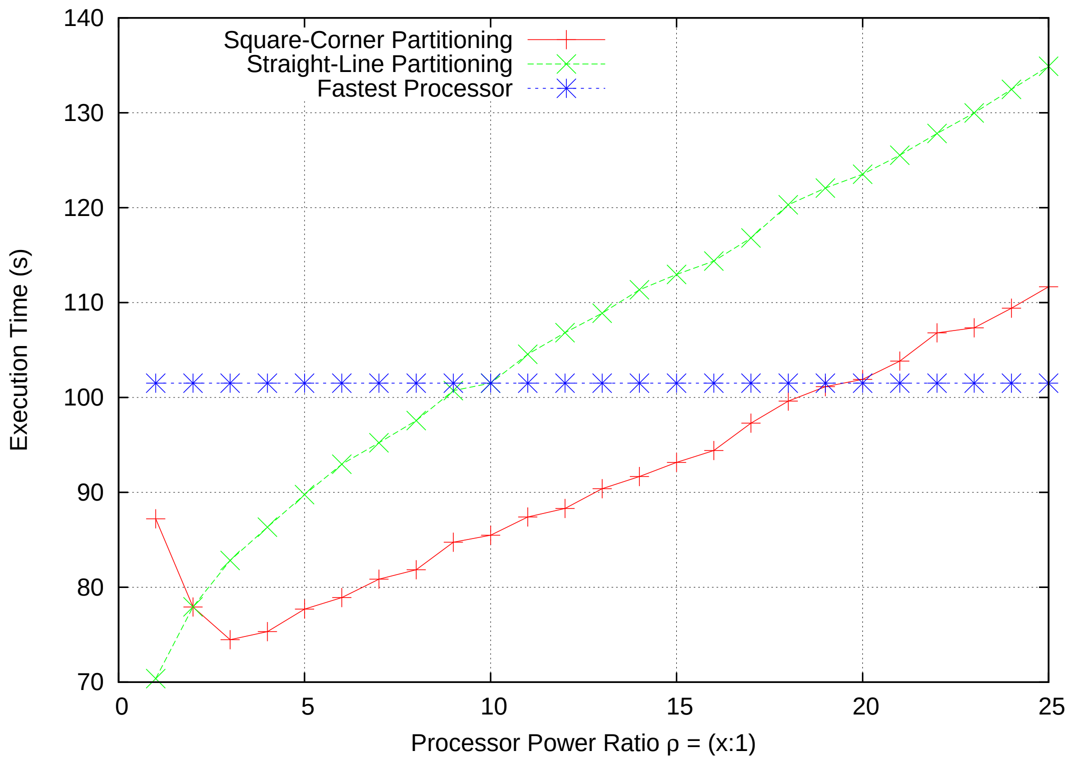


Figure 5.22: Average execution times for the Square-Corner and Straight-Line Partitionings using parallel communications. Network bandwidth is 100Mb/s, $N = 4,500$.

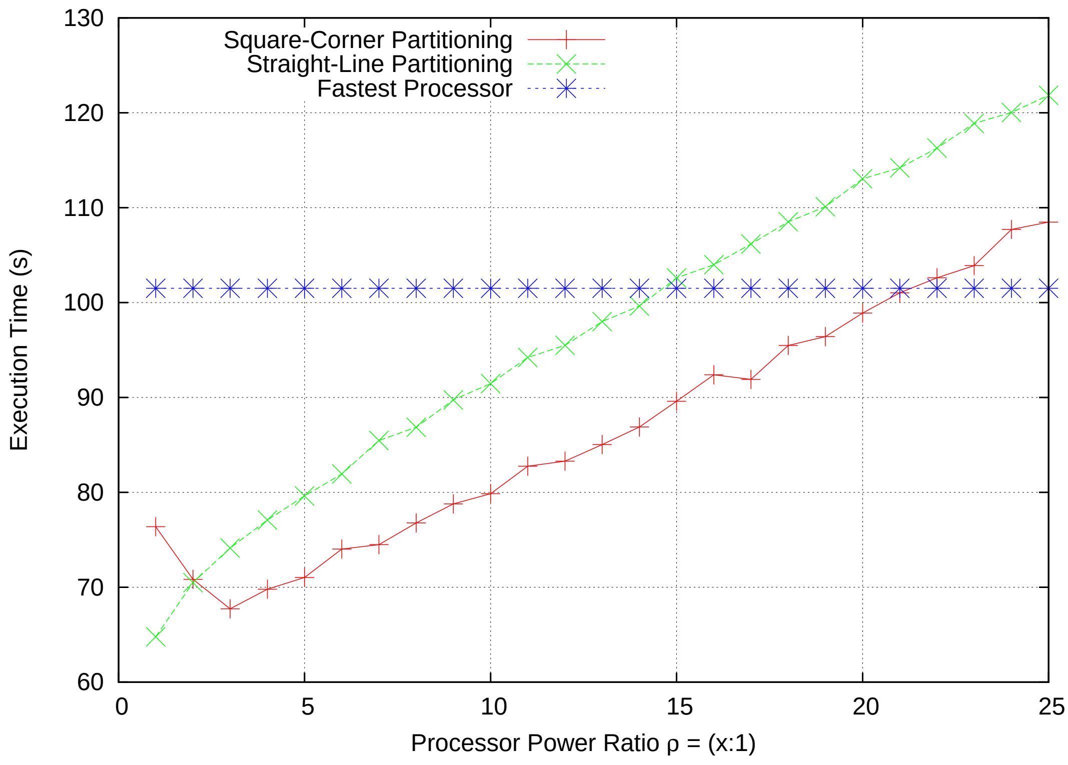


Figure 5.23: Average execution times for the Square-Corner and Straight-Line Partitionings using parallel communications. Network bandwidth is 500Mb/s, $N = 4,500$.

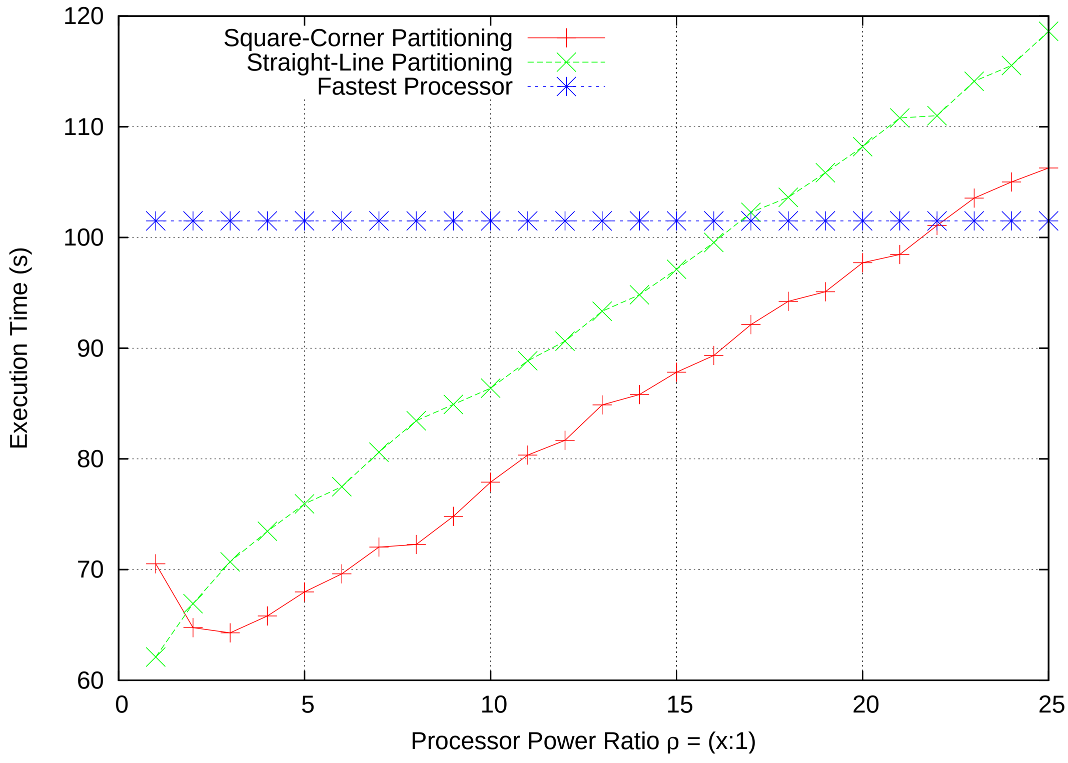


Figure 5.24: Average execution times for the Square-Corner and Straight-Line Partitionings using parallel communications. Network bandwidth is 1Gb/s, $N = 4,500$.

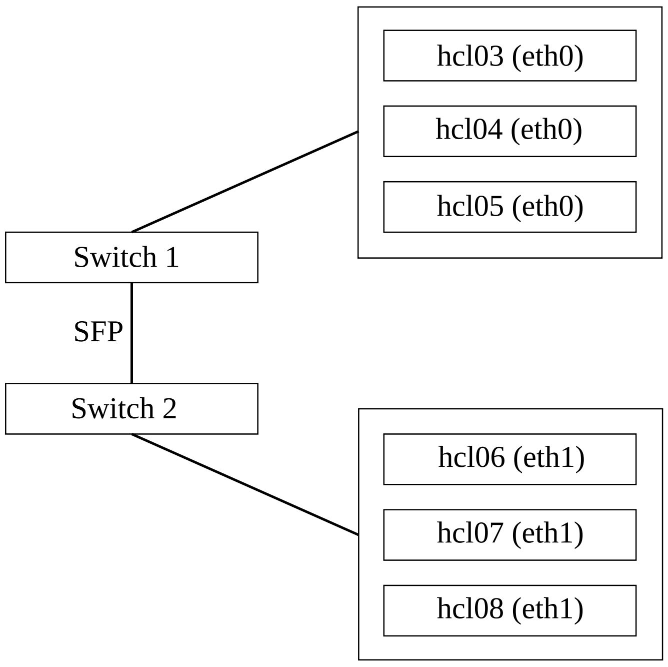


Figure 5.25: A two cluster configuration of the HCL Cluster. By bringing up NIC1 and bringing down NIC2 on hcl03 - hcl05, and the opposite for hcl06 - hcl08, and enabling the SFP connection between the switches, two homogeneous connected clusters of three nodes each are formed.

5.7 HCL Cluster Experiments

This section presents results of MPI matrix matrix multiplication experiments on a small two cluster configuration of the HCL Cluster (See Chapter 3). Six machines (hcl03 - hcl08) were used, connected through two switches utilizing the SFP connection between them as shown in Figure 5.25. Cluster 1 (hcl03, hcl04, hcl05) all have their CPU speeds restricted as in Section 5.6. Thus we have a two cluster architecture which should be well modelled by the simulations in Section 5.6.

5.7.1 Comparison of Communication Times

The communication times for this architecture were very close to the simulation using two machines in Section 5.6. Overall the communication times were slightly higher, most likely due to the increased number of machines communicating. Only the bandwidth of the SFP connection was varied as this is the only link between the two clusters.

Figures 5.26, 5.27, and 5.28 show the communication times for the Square-Corner and Straight-Line Partitionings for ratios $1 : 1$ to $1 : 15$ and a network

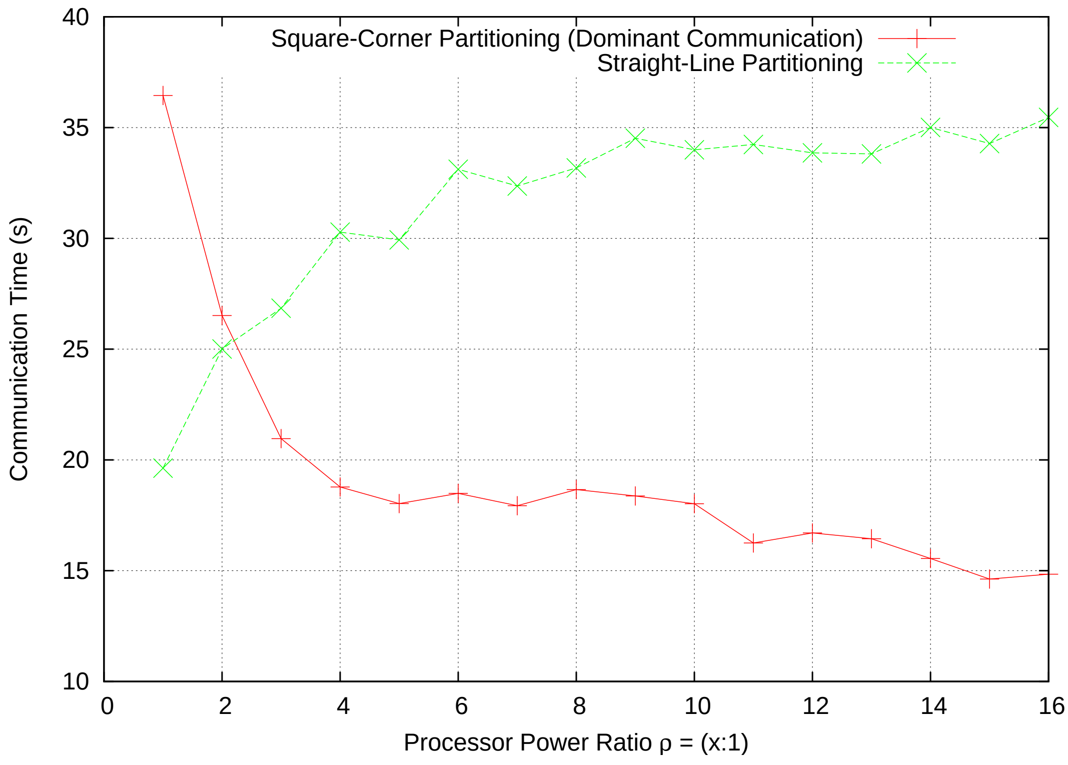


Figure 5.26: Average communication times for the Square-Corner and Straight-Line Partitionings utilizing parallel communication. Network bandwidth is 100Mb/s, $N = 4,500$.

bandwidths of 100Mb/s, 500Mb/s and 1Gb/s respectively. Parallel communications are utilized in all experiments. As with two processors and parallel communications, the Square-Corner Partitioning has a lower communication time for all $\rho > 2 : 1$. N = 4,500 for all experiments.

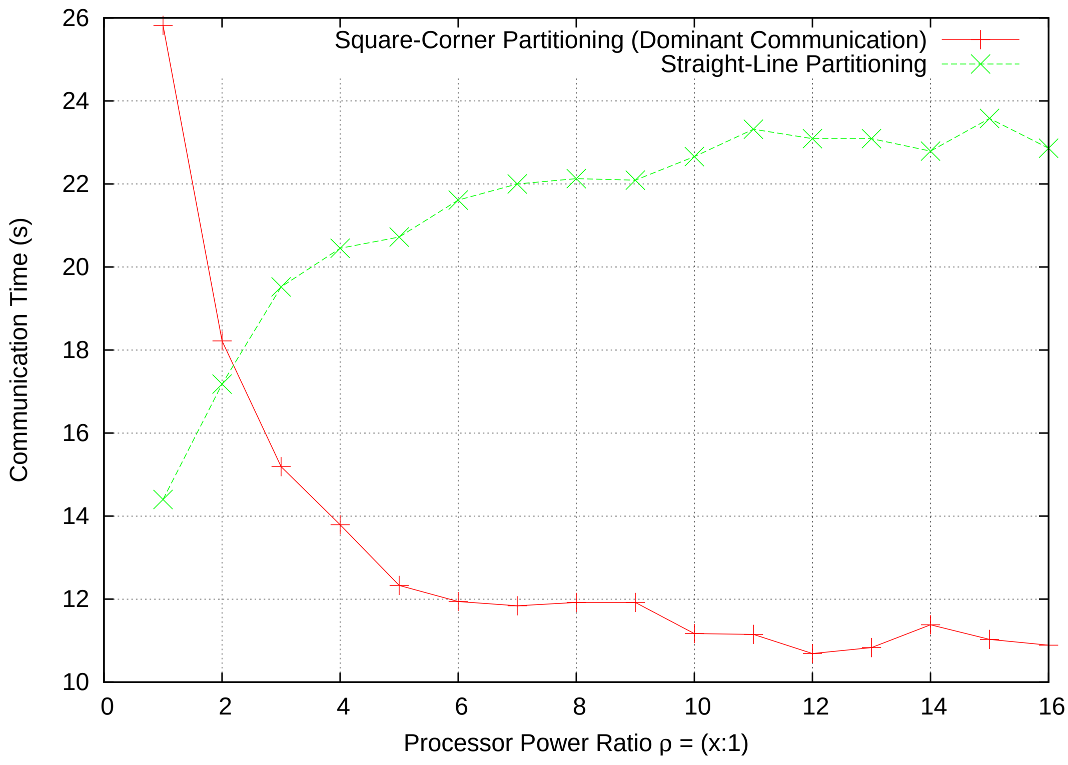


Figure 5.27: Average communication times for the Square-Corner and Straight-Line Partitionings utilizing parallel communication. Network bandwidth is 500Mb/s, $N = 4,500$.

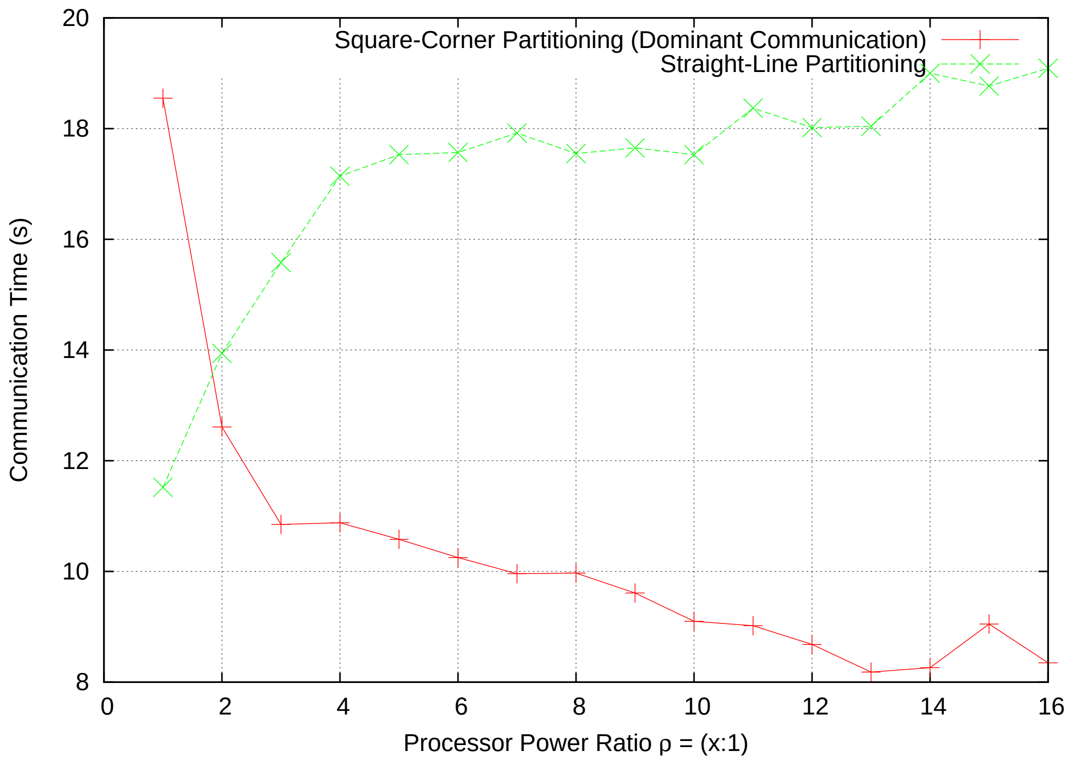


Figure 5.28: Average communication times for the Square-Corner and Straight-Line Partitionings utilizing parallel communication. Network bandwidth is 1Gb/s, $N = 4,500$.

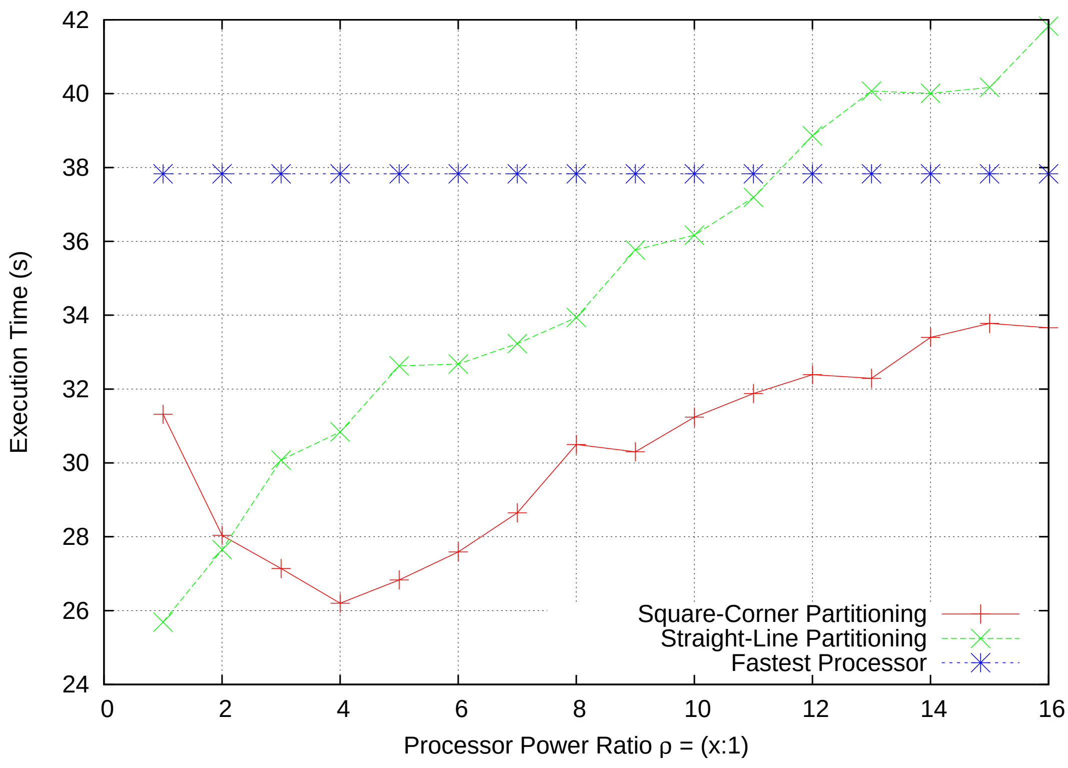


Figure 5.29: Average execution times for the Square-Corner and Straight-Line Partitionings utilizing parallel communication. Network bandwidth is 100Mb/s, $N = 4,500$.

5.7.2 Comparison of Total Execution Times

The overall execution times were again directly influenced by the communication times. Execution times were lower than simulations as expected, due to a three-fold increase in computational power through increased parallelism. It is seen that the increase in bandwidth and corresponding reduction in communication times is the largest factor in reducing the execution times.

Figures 5.29, 5.30, and 5.31 show the total execution times for the Square-Corner and Straight-Line Partitionings for ratios $1 : 1$ to $1 : 15$ and network bandwidths of 100Mb/s, 500Mb/s and 1Gb/s respectively. As with two processors and parallel communications, the Square-Corner Partitioning has a lower execution time for all $\rho > 2 : 1$.

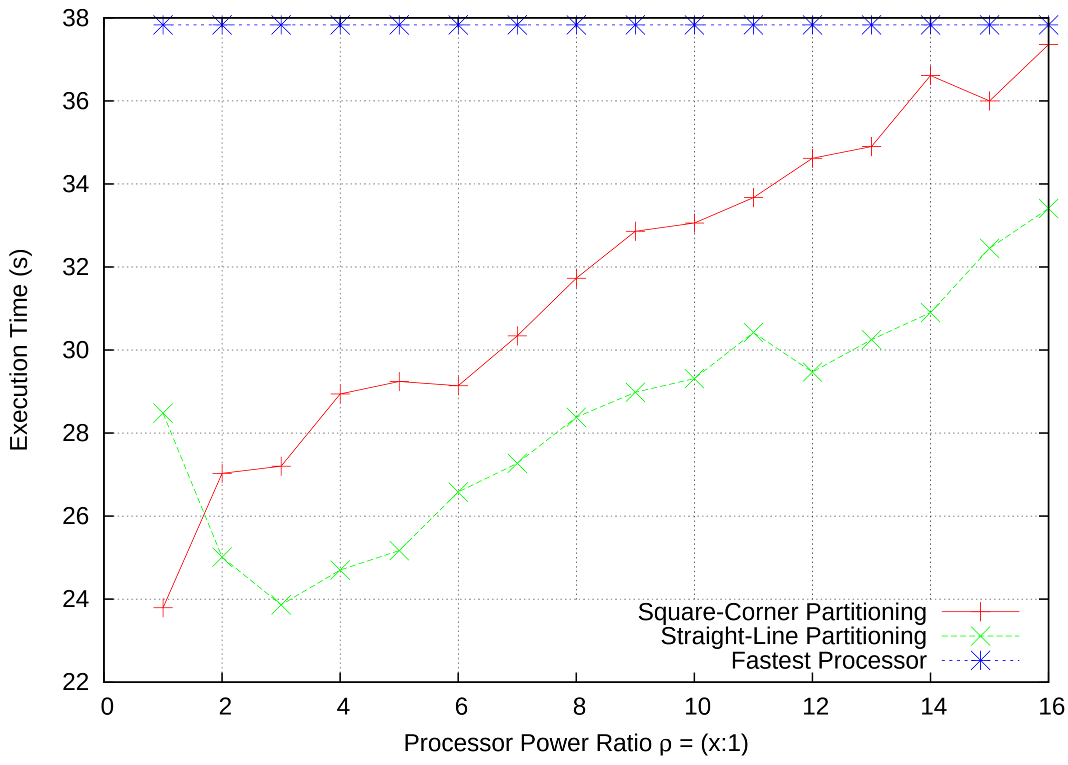


Figure 5.30: Average execution times for the Square-Corner and Straight-Line Partitionings utilizing parallel communication. Network bandwidth is 500Mb/s, $N = 4,500$.

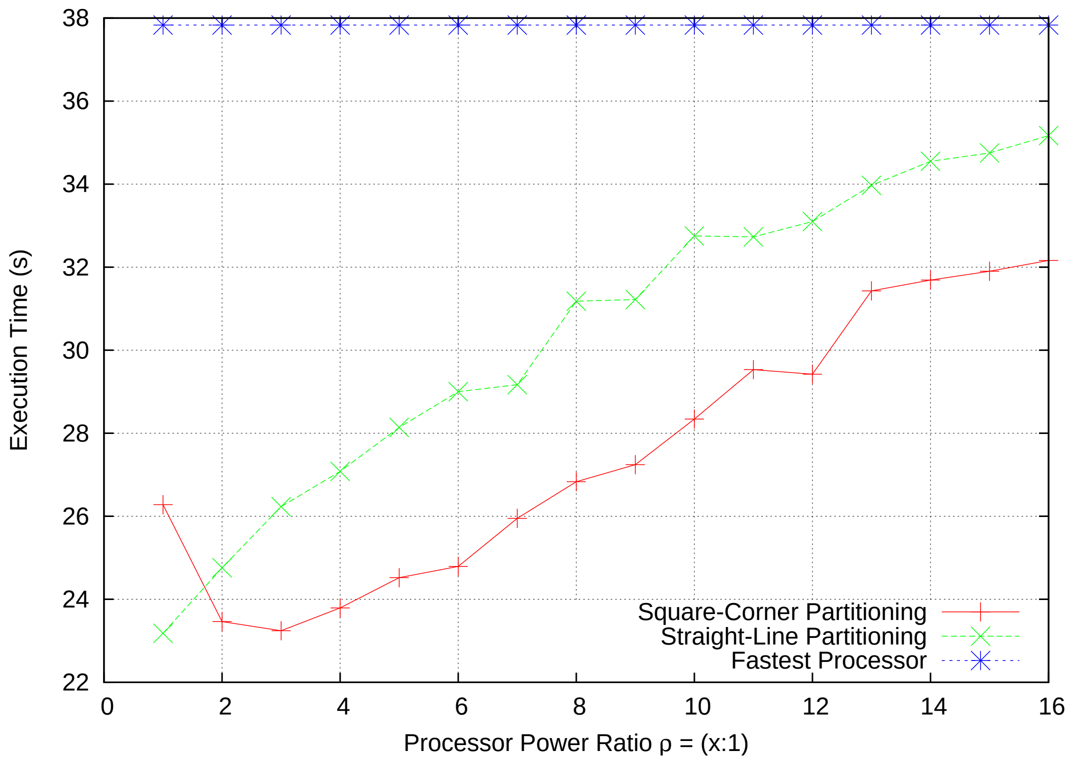


Figure 5.31: Average execution times for the Square-Corner and Straight-Line Partitionings utilizing parallel communication. Network bandwidth is 1Gb/s, $N = 4,500$.

5.8 Grid'5000 Experiments

To investigate the scalability of the Square-Corner Partitioning we utilized Grid'5000 (See Section 1.1.2.2). Grid'5000 is located across nine sites in France and has 1,529 nodes from Altix, Bull, Carri, Dell, HP, IBM and SUN. There is a total of 2,890 processors with a total of 5,946 cores from both AMD and Intel.

In this section we used two clusters of machines at the Bordeaux site. The first are IBM x4355 dual-processor nodes with AMD Opteron 2218 2.6GHz Dual Core Processors with 2MB L2 Cache and 800MHz Front Side Bus. Each node has 4GB of RAM (2GB per processor, 1GB per core). The second are IBM eServer 325 dual-processor nodes with AMD Opteron 248 2.2GHz single core processors with 1MB L2 Cache. Each node has 2GB of Ram (1GB per processor).

The problem size is $N = 15,000$.

5.8.1 Comparison of Communication Times

All communications were 1GB/s Ethernet. Parallel communications are utilized. It can be seen that an effect of keeping the overall computational power constant while increasing the relative power ratio served to flatten out some of the communication curves, particularly those for the Straight-Line Partitioning, where there is a constant amount of communication regardless of the power ratio.

Figure 5.32 shows the communication times for ratios $1:1$ to $1:8$.

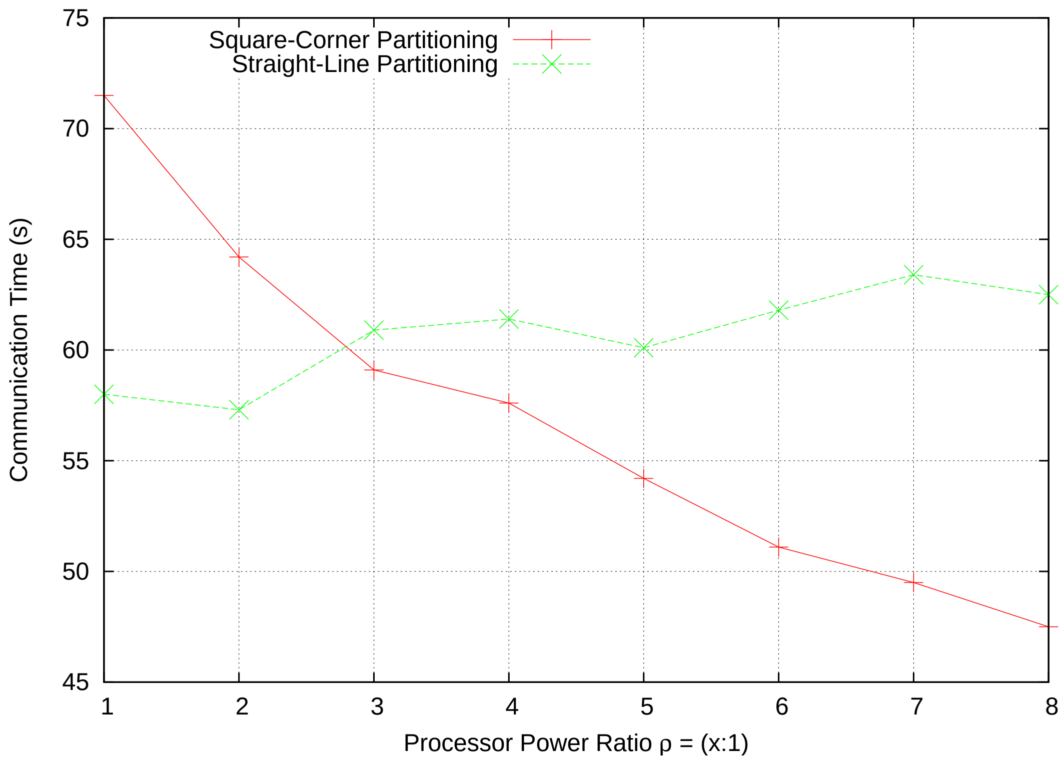


Figure 5.32: Average communication times for the Square-Corner and Straight-Line Partitionings utilizing parallel communications. Network bandwidth is 1Gb/s, $N = 15,000$.

5.8.2 Comparison of Total Execution Times

Power ratios were varied from 1 : 1 to 1 : 8 by varying the number of CPUs in each cluster while trying to maintain a relatively constant total power. This represents a departure from our simulation and experiment results in Sections 5.6 and 5.7, where the overall computational power was not kept constant. All local computations utilized ATLAS.

It can be seen that the reduction in communication time directly impacts the execution time. The expected crossover at power ratio 2:1 is present and for greater power ratios, the Square-Corner Partitioning has a lower execution time.

Both the Square-Corner and Straight-Line Partitionings resulted in lower execution times than the problem being solved on just the fastest cluster, which ran in 198s.

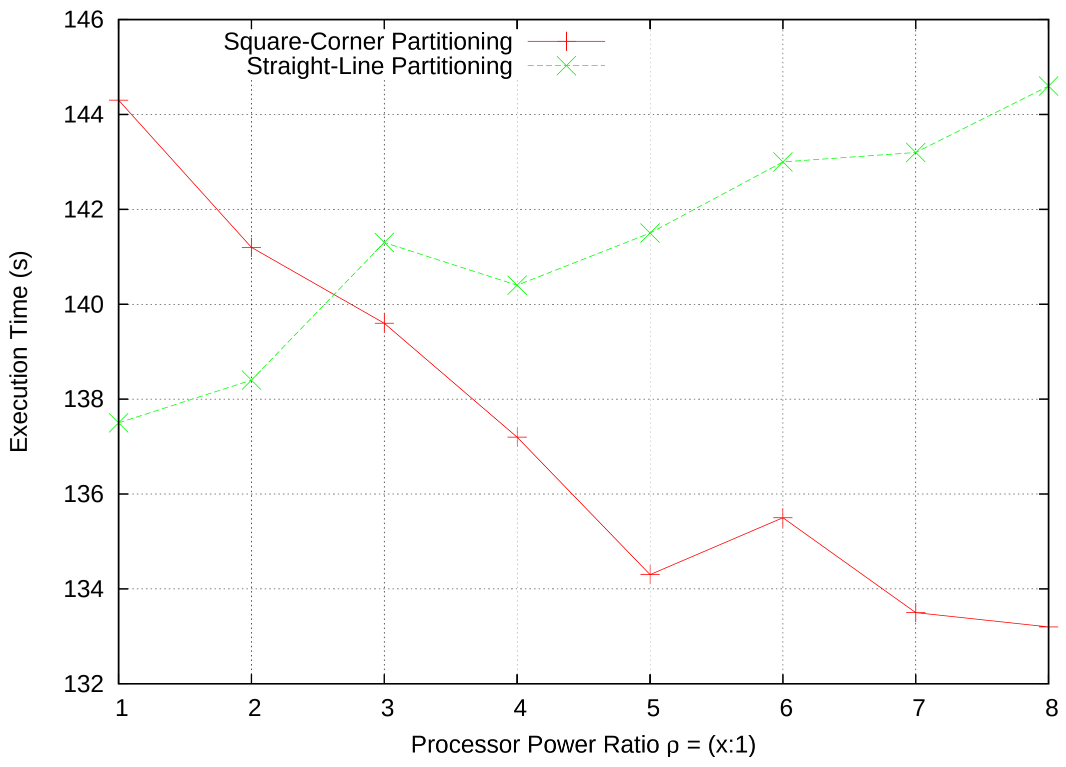


Figure 5.33: Average execution times for the Square-Corner and Straight-Line Partitionings utilizing parallel communications. Network bandwidth is 1Gb/s, $N = 15,000$.

5.9 Conclusion

This chapter presented a new data partitioning algorithm, the Square-Corner Partitioning, for matrix matrix multiplication on two heterogeneous interconnected clusters. Compared to more general partitioning algorithms which result in simple “Straight-Line” rectangular partitions on a two-cluster architecture, this new partitioning is proven to reduce the total volume of inter-cluster communication when the power ratio of the two clusters is greater than 3 : 1 when serial communications are utilized, and greater than 2 : 1 when parallel communications are utilized. This results in a lower execution time for architectures with these ratios.

This partitioning algorithm can be utilized as the top-level partitioning of a hierarchal algorithm that is to multiply matrices across more than two connected clusters. A tree-like network could deploy this partitioning at each level of the network, allowing individual clusters to handle their computations in any manner locally. When serial communications are utilised, a hybrid algorithm utilizing this new Square-Corner Partitioning for power ratios equal to or greater than 3 : 1, and the existing Straight-Line Partitioning for ratios of less than 3 : 1 guarantees that the total volume of communication will be equal to or less than previously existing algorithms for all ratios. When parallel communications are utilised, a hybrid algorithm utilizing this new Square-Corner Partitioning for power ratios equal to or greater than 2 : 1, and the existing Straight-Line Partitioning for ratios of less than 2 : 1 guarantees that the total volume of communication will be equal to or less than previously existing algorithms for all ratios.

The Square-Corner Partitioning has several advantages over other Straight-Line Partitionings including the possibility of overlapping communication and computation. This is theoretically and experimentally explored in Chapter 6.

CHAPTER

SIX

THE SQUARE-CORNER PARTITIONING ON THREE CLUSTERS

6.1 Introduction

In this chapter the Square-Corner Partitioning is extended from an architecture of two clusters to three. We do this because the move from two to three clusters represents a theoretical hurdle, and the three cluster topology is much closer to a n cluster topology than two. A large part of this is due to the fact that a three cluster system introduces a new degree of freedom, in the availability of more than one interconnection topology. Thus we are sacrificing theoretical generality to explore deeper with experimental analysis. We will see in Chapter 8 that generalizing to four or more clusters is a qualitative extension of the three cluster case.

The cases where the Square-Corner Partitioning has a lower total volume of communication than the Straight-Line Partitioning are explored theoretically then experimentally. Topological limitations are also discussed. As in the two cluster case we will start our experiments with three processor simulations before presenting experimental results on Grid'5000. Experimental results correlate well with theory and simulation.

In this research we model three clusters using three processors because a three processor network provides a controllable and tunable environment whose structure is similar to three clusters. In the case of connected clusters, local communications are often an order of magnitude faster than the interconnecting link. Due to physical distance this link is often serialized or of some limited parallelism but depending on architecture parallel links are not ruled out nor considered exotic. Similarly, with three processors local communications

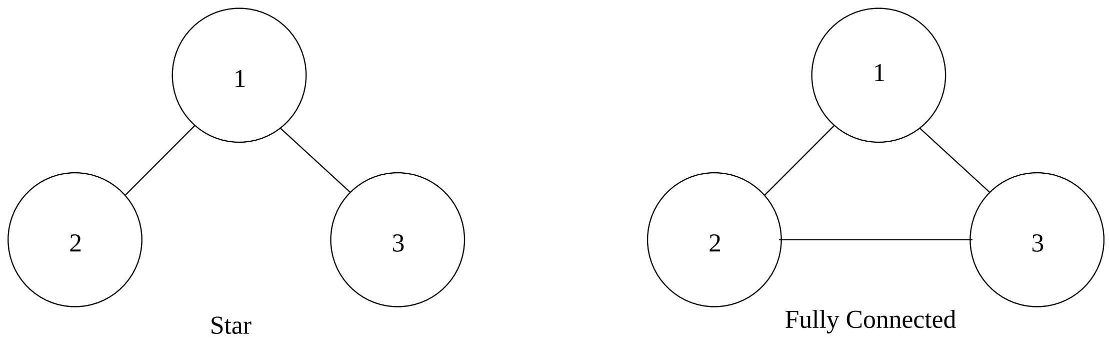


Figure 6.1: Star and Fully Connected topologies for three clusters.

(within processor-local registers and memory) are typically fast compared to the inter-processor connection speed. Using an Ethernet switch with a configurable bandwidth (discussed in Chapter 3) allows us to model many different scenarios.

As the case of three clusters brings up a new degree of freedom that was not found in the two cluster case, namely network topology, we explore the impact of this on the partitioning's effectiveness. In the case of three clusters there are "Star" and "Fully Connected" topologies now available (see Figure 6.1). We will also explore the case where communications and computations can be overlapped resulting in lower overall execution times.

To our knowledge no research has been conducted to optimize data partitioning techniques for the specific architecture of three connected nodes. The most related work is Beaumont *et al.* (2001b), which introduced a partitioning for matrix matrix multiplication designed for any number of nodes including three. This partitioning exclusively utilizes rectangular partitions, organized in columns, with each rectangle being proportional in area to the speed of the node which is to calculate that partition. This Straight-Line Partitioning in the case of three clusters results in a partitioning as shown in Figure 6.2.

The Square-Corner Partitioning differs in that the matrix is not partitioned into rectangles. We create three partitions, the first being a square located in one corner of the matrix, the second being a square in the diagonally opposite corner, and the third is polygonal, comprised of the balance of the matrix, as seen in Figure 6.3. On a star topology where the fastest node is the middle node, this partitioning always results in a lower total volume of communication (see Section 6.1.1). The benefit of a more efficient communication schedule further reduces communication time, which in turn drives down total execution time. On a fully connected topology this minimizes the total volume of communication between clusters for a defined range of power ratios (see Section 6.1.2).

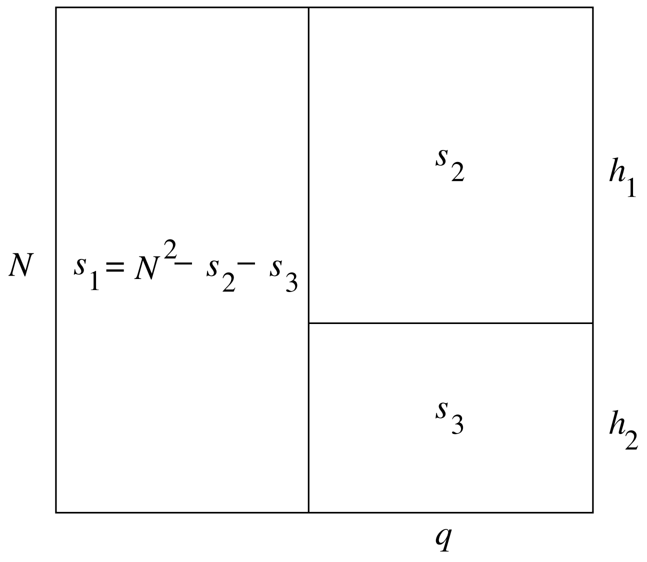


Figure 6.2: A Straight-Line Partitioning for three clusters.

This partitioning also allows for a sub-partition of the matrix product to be calculated without any communications needed. When dealing with hardware that has a dedicated communication sub-system, this can further reduce execution time.

As with the two cluster case in Chapter 5 the total volume of communication of the Square-Corner Partitioning approaches the theoretical lower bound as the power ratio between the nodes grows, unlike existing partitionings which have a TVC bounded by a constant or a function that does not approach the theoretical lower bound.

As discussed in Chapter 4 the TVC for a rectangular partitioning is proportional to the sum of the half-perimeters $\hat{C}$ of all partitions, given by Equation 6.1 (for a unit square)

$$\hat{C} = \sum_{i=1}^{p}(h_i + w_i) \tag{6.1}$$

where p is the number of nodes, and h_i and w_i are the height and width of the rectangle assigned to node i, respectively.

Since the perimeter of any rectangle enclosing a given area is minimized when that rectangle is a square, there is a natural lower bound LB for $\hat{C}$ given by Equation 6.2, where a_i is the area of the partition belonging to node i.

$$LB = 2 \times \sum_{i=1}^{p} \sqrt{a_i} \tag{6.2}$$

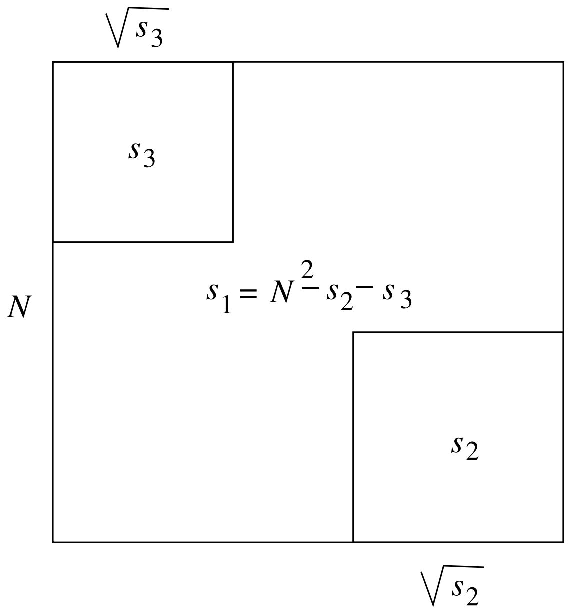


Figure 6.3: The Square-Corner Partitioning for three clusters.

As in the case for two clusters, the lower bound cannot always be met by the Straight-Line Partitioning. However as Figure 6.3 shows, as s_2 and s_3 approach 0, the SHP of the Square-Corner Partitioning approaches 2, which is the perimeter of the unit square itself, and therefore optimal. Figure 6.4 shows the Straight-Line Partitioning and necessary data movements to carry out the matrix matrix multiplication $C = A \times B$.

Although the general rectangular partitioning problem is NP-complete it is easy to show that for the simple case of three partitions the best possible rectangular partitioning is of the form shown in Figure 6.2 (where s_2 and s_3 are the smaller partitions), as this arrangement minimizes q in Equation 6.3, the only variable quantity in the TVC of the Straight-Line Partitioning for a matrix size N.

$$N^2 + N \times q \tag{6.3}$$

In order to calculate its partition of C, Cluster 1 needs to receive the respective partitions of A from Clusters 2 and 3, Cluster 2 needs to receive Cluster 3's partition of B, and part of Cluster 1's partition of A, and Cluster 3 needs to receive Cluster 2's partition of B and the remaining part of Cluster 1's partition of A, as shown in figure 6.4.

If we define the area of Cluster 2's partition to be s_2 and Cluster 3's partition to be s_3, Equation 6.3 can be expressed as Equation 6.4.

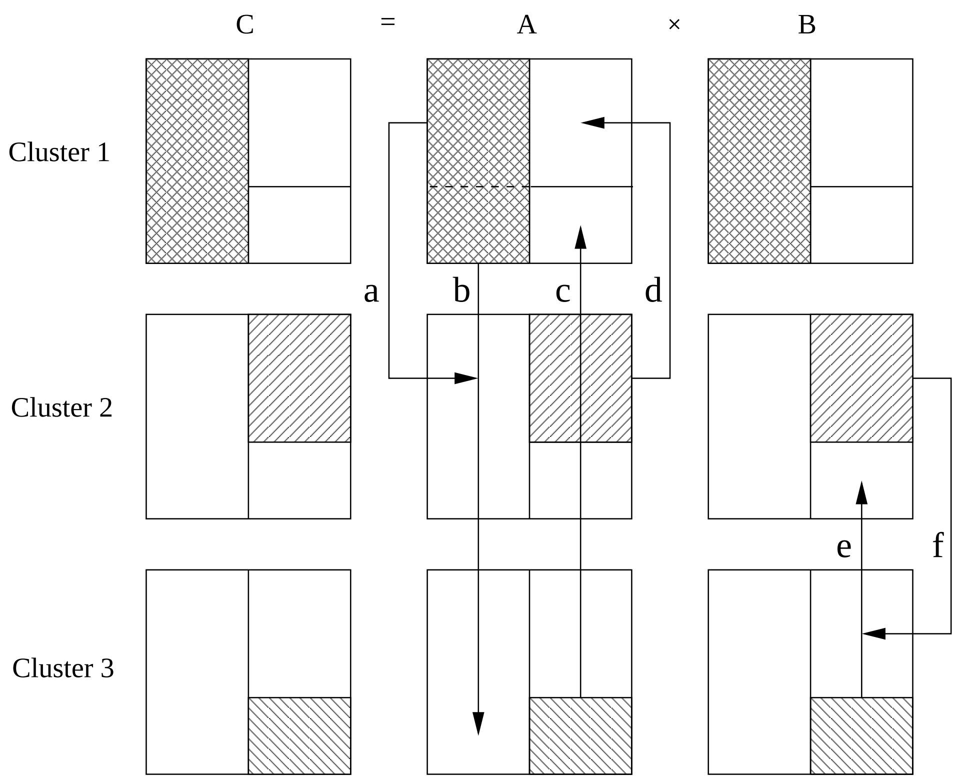


Figure 6.4: A two-dimensional Straight-Line Partitioning and data movements required to carry out $C = A \times B$ on three heterogeneous nodes or clusters.

$$N^2 + s_2 + s_3 \tag{6.4}$$

When dealing with a star topology where Cluster 1 (the fastest cluster) is the middle topologically, the communications between Clusters 2 and 3 must go through Cluster 1. This has the effect of doubling the total volume of communication between Clusters 2 and 3, as all communications between this pair must first be sent to and received by Cluster 1 before the data can be sent on to the recipient node. This raises the TVC to Equation 6.5.

$$N^2 + 2 \times (s_2 + s_3) \tag{6.5}$$

The Square-Corner Partitioning differs from the Straight-Line (rectangular) partitioning described above by relaxing the restriction that all partitions must be rectangular. We extend the two node partitioning presented in Chapter 5 by creating two square partitions in diagonally opposite corners of the matrix. Since the total volume of communication is proportional to the sum of half perimeters of the partitions, it is easy to show that the sum of half perimeters is at a minimum when the two slower nodes are assigned square partitions. There-

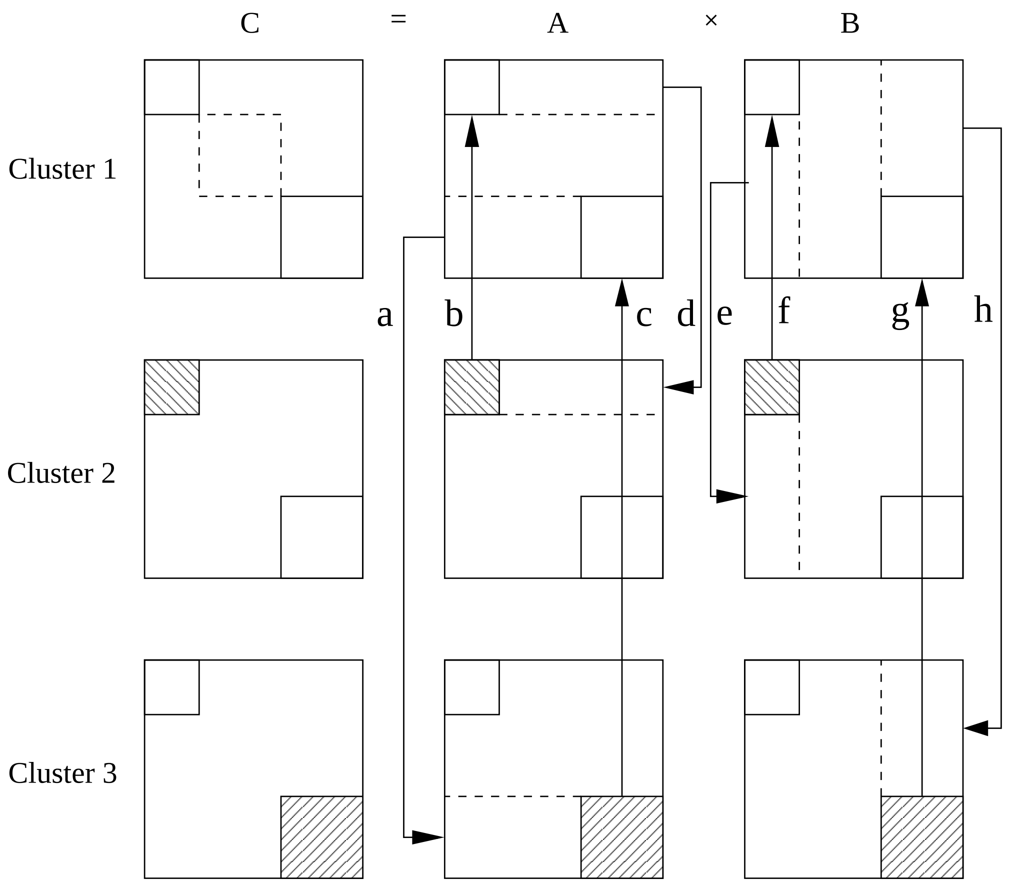


Figure 6.5: The Square-Corner Partitioning and data movements necessary to calculate $C = A \times B$.

fore, the optimal Square-Corner Partitioning assigns the balance of the matrix to the fastest node. Since a square has the smallest perimeter of any rectangle of a given area we do not consider non-square rectangular corner partitions. Figure 6.5 shows the partitioning scheme used by the Square-Corner Partitioning and the necessary data movements to calculate $C = A \times B$.

The total volume of communication of the Square-Corner Partitioning is given by Equation 6.6, where s_2 and s_3 are the areas assigned to Clusters 2 and 3 (the two slower clusters).

$$2 \times N \times (\sqrt{s_2} + \sqrt{s_3}) \tag{6.6}$$

As Figure 6.5 shows, Clusters 2 and 3 do not communicate at all, thus the TVC is equal to Equation 6.6 for both the fully connected and star topology where Cluster 1 is the middle Cluster topologically.

Other similar (but non-Square-Corner) partitioning methods were also investigated. Examples of two partitionings explored are shown in Figure 6.6. In the

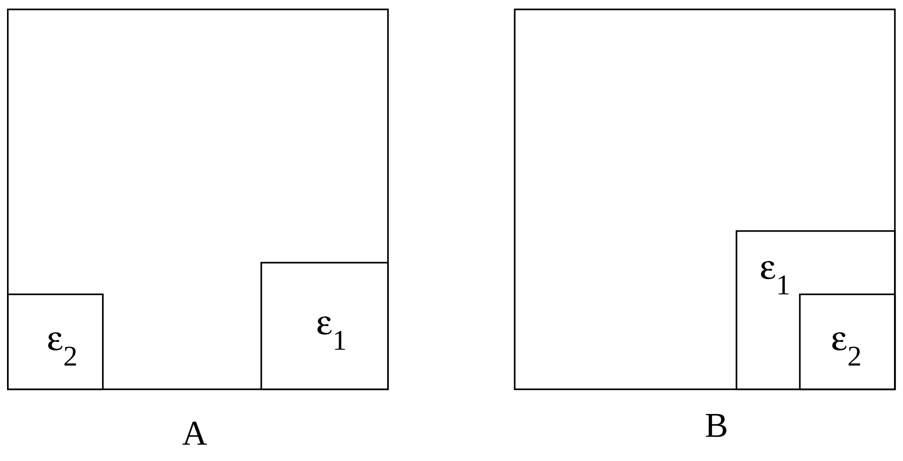


Figure 6.6: Examples of non-Square-Corner Partitionings investigated.

Square-Corner Partitioning, diagonally opposite corners are chosen to minimize the number of communication steps necessary as well as for the reasons discussed in Section 5.2. As shown in Figure 6.5 the number of communication steps is eight. Placing the squares in corners that are not diagonally opposite as in Figure 6.6.A requires ten communication steps provided $\epsilon_1 \neq \epsilon_2$. If $\epsilon_1 = \epsilon_2$ the number of steps remains at eight. The total volume of communication is still equal to Equation 6.6 regardless. If the partitions are nested as in Figure 6.6.B, the number of communication steps is 12 and a higher TVC results. All other (more exotic) partitioning methods investigated resulted in an increased TVC also.

In the Square-Corner Partitioning, the square partitions cannot overlap. This imposes the following restriction on the relative speeds of the clusters:

$$\frac{s_2}{s_1} \times \frac{s_3}{s_1} \leq \frac{1}{4} \tag{6.7}$$

where $s_1 + s_2 + s_3 = 1$ and s_1 is the relative speed of the cluster owning the balance of the matrix (the fastest cluster). A consequence of this is that the possible cluster ratios are somewhat limited. Another way of visualizing this is by noting that the areas of s_2 and s_3 cannot overlap. This eliminates certain ratios such as 1:1:1.

6.1.1 Comparison of Communications on a Star Topology

Since in the Square-Corner Partitioning Clusters 2 and 3 do not have to communicate at all, the total volume of communication on a star where Cluster 1 is the middle cluster remains equal to Equation 6.6. The Straight-Line Partitioning has a TVC equal to Equation 6.5. In terms of cluster speeds, the Square-Corner Partitioning has a lower TVC when Inequality 6.8 is satisfied.

$$(\sqrt{s_2} + \sqrt{s_3}) < 1.5 - s_1 \tag{6.8}$$

where $s_1 : s_2 : s_3$ is the ratio representing the node speeds, normalized so that $s_1 + s_2 + s_3 = 1$, and subject to the restriction of Inequality 6.7.

In order to see when the Square-Corner Partitioning has a lower total volume of communication than the Straight-Line partitioning, we examined the surface

$$z = (\sqrt{s_3} + \sqrt{s_2}) - 1.5 + s_1 \tag{6.9}$$

which represents the Straight-Line Partitioning's total volume of communication subtracted from that of the Square-Corner Partitioning. Since $z < 0$ for all positive values of s_1, s_2, and s_3, the Square-Corner Partitioning always results in a lower total volume of communication.

Additionally, the fact that Cluster 1 must now relay data from Cluster 2 to Cluster 3 and vice-versa introduces a section of the communication schedule that is necessarily serialized in the Straight-Line Partitioning. The Square-Corner Partitioning has no such section and can therefore exploit in full any existing network parallelism.

For the Straight-Line Partitioning, the total volume of communication becomes dependent on which cluster is the middle node topologically, and in turn on the values of s_2 and s_3. These three topologies and their required communications are shown in Figure 6.7. In the figure, the following are the volumes of each communication:

$$\begin{gathered}\mathrm{a} + \mathrm{b} = N^2 - s_2 - s_3 \\ \mathrm{c} = \mathrm{e} = s_2 \\ \mathrm{d} = \mathrm{f} = s_3\end{gathered}$$

The resultant TVCs are shown in Table 6.1.

In Table 6.1 the TVC for cases where Clusters 2 and 3 are the center nodes topologically are presented as inequalities because it is impossible to separate a and b in a general manner. Only their sum (a + b = $N^2 - s_2 - s_3$) can be quantified (see Figure 6.4). Therefore where (a + 2 $\times$ b) and (2 $\times$ a + b) appear,

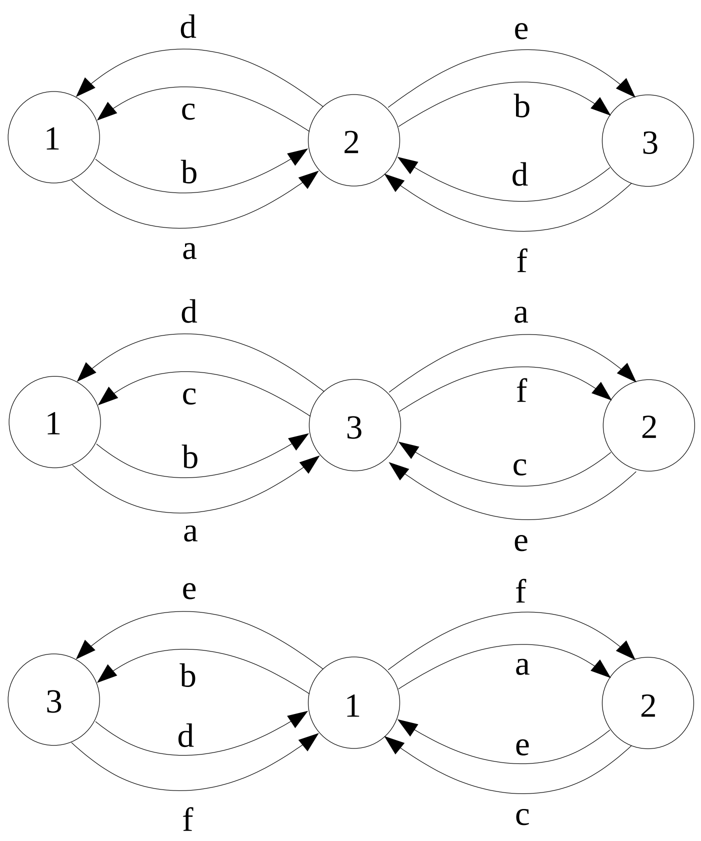


Figure 6.7: The three possible star topologies for three clusters using the Straight-Line Partitioning and associated data movements. Labels refer to data movements in Figure 6.4. Volumes of communication are shown in Table 6.1.

only (a + b) is used, thus giving a conservative value for the TVC in these cases.

Note that in Table 6.1, the TVC calculated for the case where Cluster 1 is the center cluster is equivalent to Equation 6.5, the TVC calculated in Section 6.1.

For the Square-Corner Partitioning, the total volume of communication is always less when the most powerful cluster (Cluster 1) is the center node. This is shown in Figure 6.8. Additionally, when Cluster 1 is not the center node, the communication schedule gets more complicated and one side of the network becomes busier than the other. In the figure, the following are the volumes of each communication:

$$\begin{gathered}
\text{a} = \text{h} = \sqrt{s_2} \times (N - \sqrt{s_2}) \\
\text{e,} = \text{d} = \sqrt{s_3} \times (N - \sqrt{s_3}) \\
\text{c} = \text{g} = s_2 \\
\text{b} = \text{f} = s_3,
\end{gathered}$$

The resultant TVCs are shown in Table 6.2.

Center Cluster	TVC
1	a + b + c + d + 2 × e + 2 × f $= N^2 + 2 \times s_2 + 2 \times s_3$
2	a + 2 × b + c + 2 × d + e + f $> N^2 + s_2 + 2 \times s_3$
3	2 × a + b + 2 × c + d + e + f $> N^2 + 2 \times s_2 + s_3$

Table 6.1: TVC values for the Straight-Line Partitioning on the three possible star topologies shown in Figure 6.7. Letters a - f refer to labels in Figure 6.7.

Note that in Table 6.2, the TVC calculated for the case where Cluster 1 is the center cluster is equivalent to Equation 6.6, the TVC calculated in Section 6.1.

Center Cluster	TVC
1	a + b + c + d + e + f + g + h $= 2 \times N \times (\sqrt{s_2} + \sqrt{s_3})$
2	2 × a + b + 2 × c + d + e + f + 2 × g + 2 × h $= 2 \times N \times (2 \times \sqrt{s_2} + \sqrt{s_3})$
3	a + 2 × b + c + 2 × d + 2 × e + 2 × f + g + h $= 2 \times N \times (\sqrt{s_2} + 2 \times \sqrt{s_3})$

Table 6.2: TVC values for the Square-Corner Partitioning on the three possible star topologies shown in Figure 6.8. Letters a - h refer to labels in Figure 6.8.

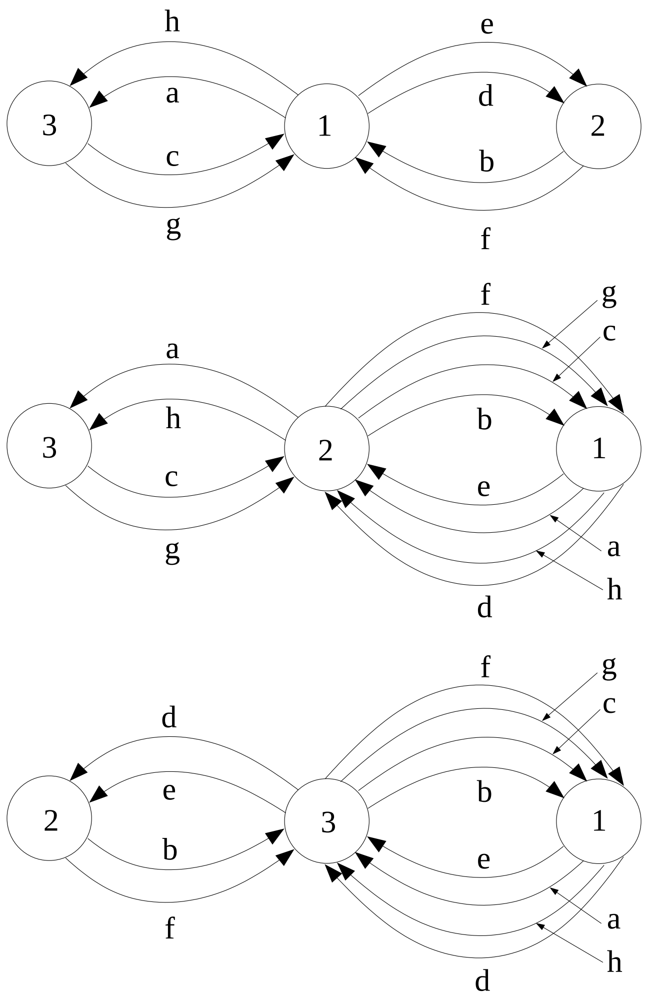


Figure 6.8: The three possible star topologies for three clusters using the Square-Corner Partitioning and associated data movements. Labels refer to data movements in Figure 6.5. Volumes of communication are shown in Table 6.2.

6.1.2 Comparison of Communications of a Fully Connected Topology

On a fully connected topology the Square-Corner Partitioning has a total volume of communication equal to Equation 6.6 and the Straight-Line partitioning has a total volume of communication equal to Equation 6.4. In terms of cluster speeds the Square-Corner Partitioning results in a lower total volume of communication when

$$(\sqrt{s_2} + \sqrt{s_3}) < 1 - \frac{s_1}{2} \tag{6.10}$$

is satisfied, where again $s_1 : s_2 : s_3$ is the ratio representing the cluster speeds, normalized so that $s_1 + s_2 + s_3 = 1$, and subject to the restriction of Inequality 6.7.

This inequality shows that the total volume of communication is dependent on the values of s_2 and s_3 (s_2 can be expressed as $1 - s_2 - s_3$). To investigate what values of s_2 and s_3 result in a lower total volume of communication compared to the rectangular partitioning, we plotted the surface

$$z = (\sqrt{s_2} + \sqrt{s_3}) - 1 + \frac{s_1}{2} \tag{6.11}$$

which is negative when the Square-Corner Partitioning's total volume of communication is less than that of the Straight-Line Partitioning. Figure 6.9 shows this surface, and Figure 6.10 shows a contour plot of this surface at $z = 0$. In Figure 6.10 only the values of $z < 0$ are shown. It is for these values that the TVC of the Square-Corner Partitioning is less than that of the Straight-Line Partitioning.

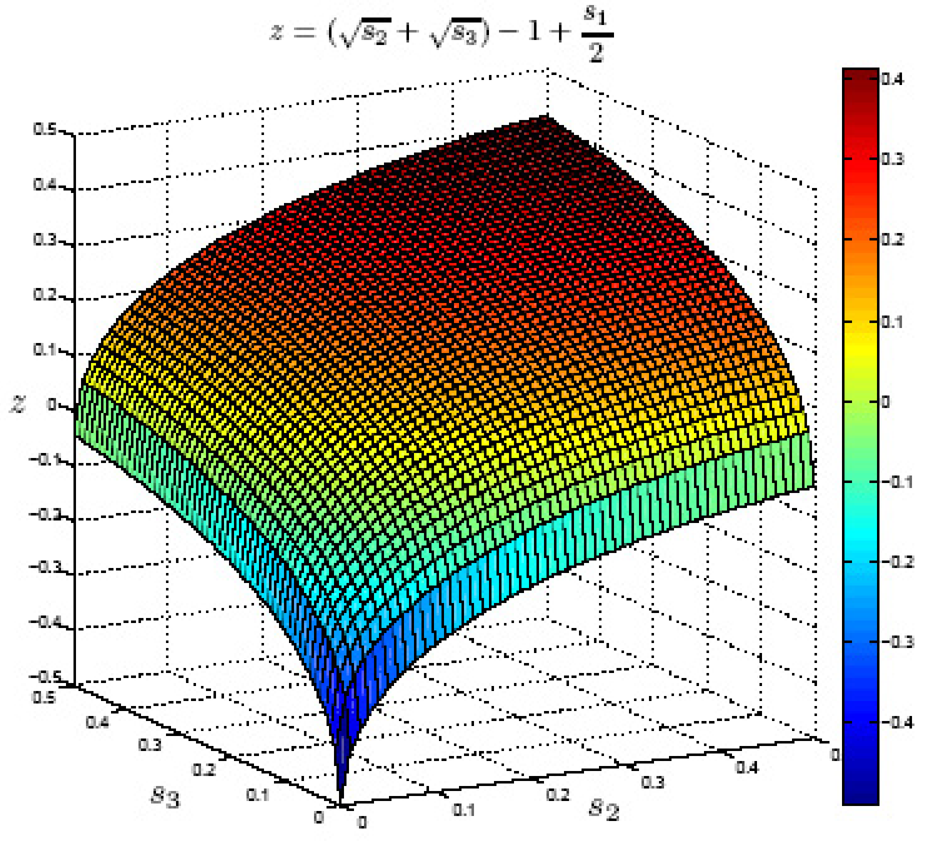


Figure 6.9: The surface defined by Equation 6.11. The Square-Corner Partitioning has a lower TVC for $z < 0$.

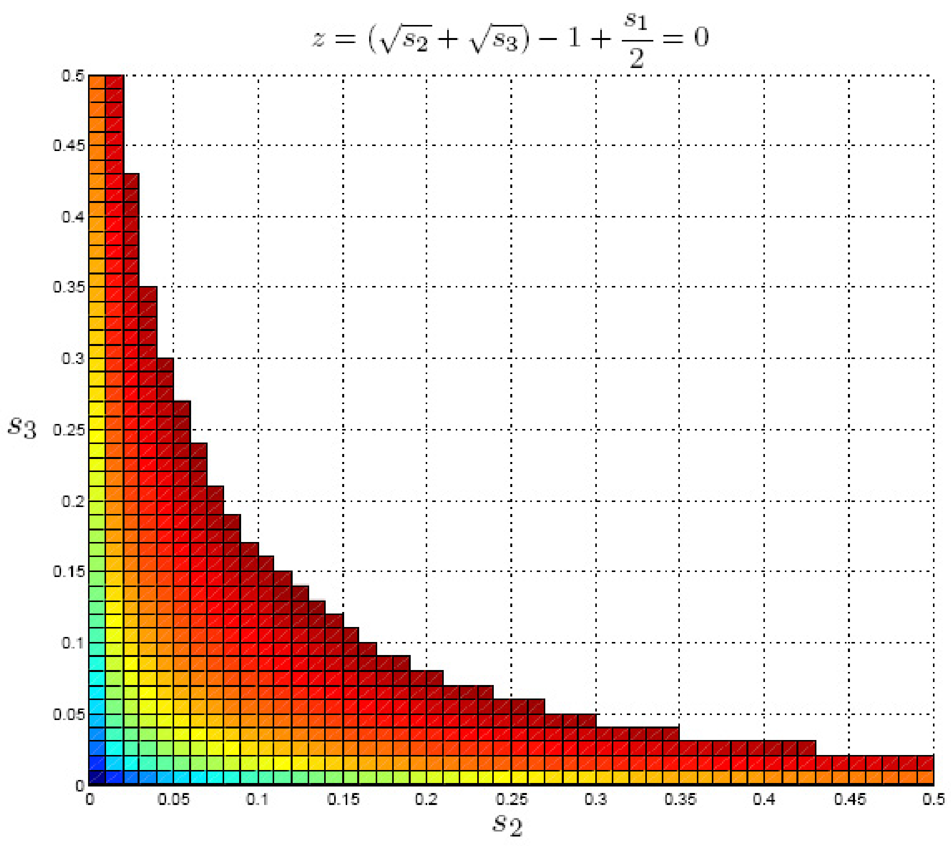


Figure 6.10: A contour plot of the surface defined by Equation 6.11 at $z = 0$. For simplicity, $s_2 = s_3$, but this is not a restriction of the Square-Corner Partitioning in general.

6.1.3 Comparison with the State-Of-The-Art and the Lower Bound

In Section 5.2 we summarized the work of Beaumont *et al.* (2001b) for the specific case of two nodes or clusters. The authors present an algorithm to find an optimal rectangular partitioning with the restriction that the rectangles are arranged in columns. For three nodes, this algorithm results in a partitioning similar to Figure 6.2, with three rectangles proportional in area to the relative powers of the nodes which own them. For the unit square, the sum of half-perimeters $\hat{C}$ which is proportional to the total volume of communication was given by Equation 6.1, and in the case of three nodes is:

$$\hat{C} = \sum_{i=1}^{p}(h_i + w_i) = 3 + q \tag{6.12}$$

where $0 < q < 1$.

The lower bound of the sum of half perimeters LB is given by Equation 6.2, and for the case of three nodes is:

$$LB = 2 \times \sum_{i=1}^{p} \sqrt{s_i} = 2 \times (\sqrt{s_1} + \sqrt{s_2} + \sqrt{s_3}) \tag{6.13}$$

where s_i is the area of the partition belonging to node i.

In the case of three nodes, the Square-Corner Partitioning has a sum of half perimeters equal to Equation 6.14:

$$\hat{C} = 2 + \sqrt{s_2} + \sqrt{s_3}. \tag{6.14}$$

and therefore

$$\lim_{s_2+s_3 \to 0} \hat{C} = 2 \tag{6.15}$$

which is equal to the lower bound that cannot be met by the Straight-Line Partitioning.

To compare the Square-Corner sum of half-perimeters with that of the Straight-Line Partitioning and the lower bound, we adopted the same approach as in Beaumont *et al.* (2001b). We generated 2,000,000 random values for the partition areas s_2, s_3 and $s_1 = 1 - s_2 - s_3$, and calculated values for the sum of half-perimeters $\hat{C}$ and the lower bound LB. Since we already know that the total volume of communication for the Square-Corner Partitioning (on a fully connected network) is greater for the cases where Equation 6.11 is positive, we restrict the random areas s_1, s_2, and s_3 accordingly.

The average sum of half-perimeter to lower bound ratio for the rectangular partitioning is 1.128, while that of the Square-Corner Partitioning is 1.079. Considering that 1.0 is the optimum value, this is an improvement of 38%. The minimum value for the sum of half-perimeter to lower bound ratio for the rectangular partitioning is 1.0595, while that of the Square-Corner Partitioning is 1.0001, an improvement of well over 99%. This also shows that the Square-Corner Partitioning does approach the lower bound which cannot be met by the rectangular partitioning.

In generating 2,000,000 random areas, there are bound to be many that have very large ratios, making them computationally unrealistic. Surely nobody would use two entities in parallel if one of them is slower than the other by an order of hundreds or thousands or greater. We therefore imposed the tighter but more realistic restriction of $s_{max}/s_{min} \leq 100$. Even with these much tighter restrictions, the average sum of half-perimeter to lower bound ratio for the rectangular partitioning is 1.104 while that of Square-Corner Partitioning is 1.062, an improvement of 40%. The minimum is improved from 1.059 to 1.008, an improvement of 86%.

6.1.4 Overlapping Communications and Computations

The other primary benefit of the Square-Corner Partitioning is overlapping communications and computations. As seen in Figure 6.5, and in more detail Figure 6.11, there is a sub-partition C_1 of Cluster 1's C partition which is immediately calculable—no communications are necessary to compute the product of this sub-partition. On an architecture which has a dedicated communications sub-system this quality can be exploited to overlap some communications and computations.

Figure 6.12 shows a schematic of the overlapping of communication and computation from an execution time point of view. As the areas C_2, X, and Y (in Figure 6.11) are calculated to be proportional to the speed of the nodes owning them, it is expected that steps III, IV, and V will finish their computations at the same time. The same is not true for steps I and II, as they represent unrelated tasks.

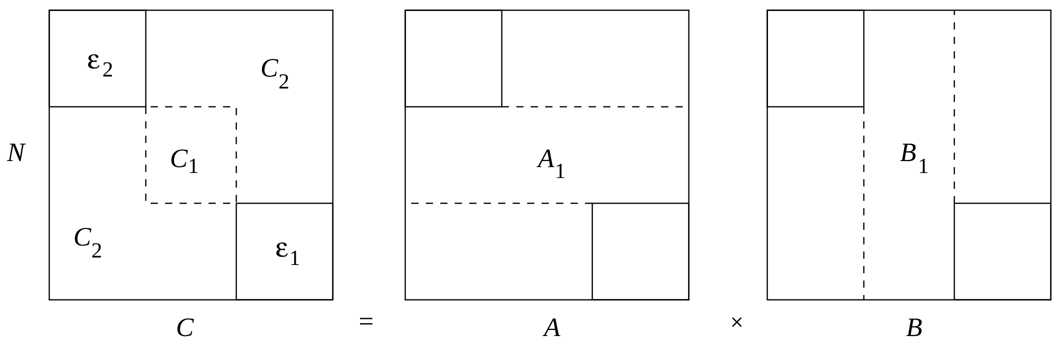
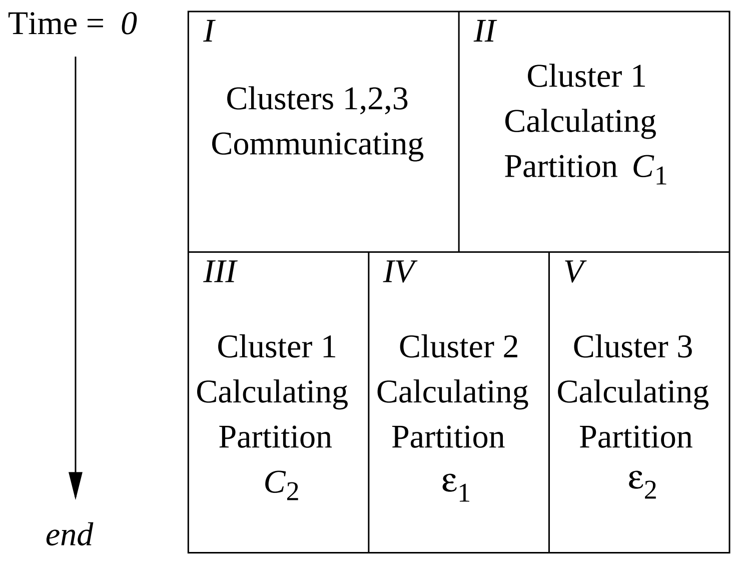


Figure 6.11: Cluster 1′s partition is $(C_1 \cup C_2 \cup C_3)$. The sub-partition $C1 = A1 \times B1$ is immediately calculable—no communications are necessary to compute its product.



Time = 0
end
I
Clusters 1,2,3 Communicating
II
Cluster 1 Calculating Partition C_1
III
Cluster 1 Calculating Partition C_2
IV
Cluster 2 Calculating Partition ε_1
V
Cluster 3 Calculating Partition ε_2

Figure 6.12: Overlapping Communication and Computation from an execution-time point of view.

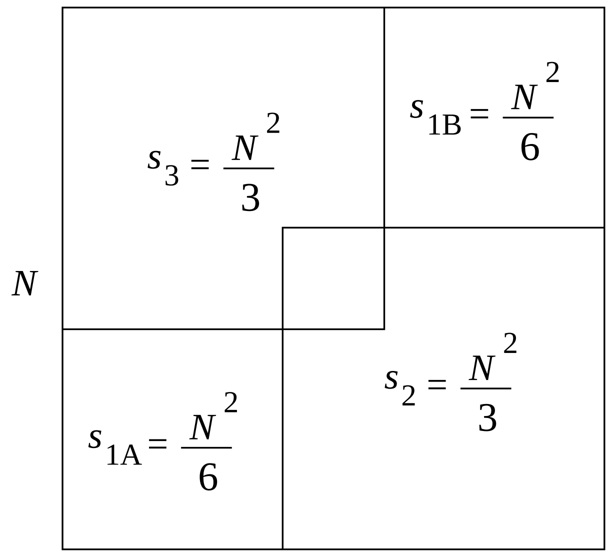


Figure 6.13: Demonstration that the Square-Corner Partitioning for three clusters with a 1:1:1 ratio is not possible, as this ratio forces s_2 and s_3 to overlap, which is not allowed by definition. $s_1 = s_{1A} + s_{1B}$.

A solution exists which would minimize the overall execution time but this would alter the approach of the Square-Corner Partitioning. Thus we formulate the total execution time as $t_{exe} = max(I, II) + max(III, IV, V)$

6.1.5 Topological Limitations

It would be incomplete not to note that on three cluster topologies the Square-Corner Partitioning has some limitations.

1. On a star topology, the areas of s_2 and s_3 cannot overlap. This restricts the possible power ratios eliminating some ratios such as 1:1:1 (see Figure 6.13). The closest to $1 : 1 : 1$ that the Square-Corner Partitioning can get is $2 : 1 : 1$, ($s_2 = s_3 = \frac{s_1}{2}$). This occurs when the corners of s_2 and s_3 "meet" (but do not overlap) in the middle of the matrix.

2. On a star topology, the center cluster or node must be the fastest (s_1). The reasons for this are discussed in Section 6.1.1.

3. On a fully connected topology the ratios where the Square-Corner Partitioning out performs the Straight-Line Partitioning are limited (see Figure 6.10). When $s_2 = s_3$ these ratios account for about 20% of possible ratios. This figure would change when $s_2 \neq s_3$, but it gives a good indication of this limitation.

4. Optimizing the overlapping of communication and computation could prove difficult and would be platform dependent, unlike other aspects of the Square-Corner Partitioning.

6.2 HCL Cluster Simulations

To experimentally verify this new partitioning, we implemented matrix multiplications utilizing the optimal Square-Corner Partitioning and the Straight-Line (rectangular) Partitioning in Open-MPI (Gabriel *et al.*, 2004). Local matrix multiplications utilize ATLAS (Whaley and Dongarra, 1999). Experiments were carried out on three identical machines to eliminate contributions of architectural differences. The machines were connected with a full duplex Ethernet switch that allows the bandwidth between the nodes to be finely controlled.

The ratio of speeds between the three nodes were varied by slowing down CPUs when required using a CPU limiting program as proposed in (Canon and Jeannot, 2006). This program supervises a specified processes and using the `/proc` pseudo-filesystem, forces the process to sleep when it has used more than a specified fraction of CPU time. The process is then woken when enough idle CPU time has elapsed for the process to resume. Sampled frequently enough, this provides a fine level of control over the CPU speed. Comparison of the run-times of each node confirmed that this method results in the desired ratios to within 2%.

For simplicity we present results where the speeds of the slower nodes (s_2 and s_3) are equal. We varied this relative value from 5 to 25, where $s_1 = 100 - s_2 - s_3$. Network bandwidth is 100Mb/s, and $N = 5,000$.

Figure 6.14 shows the communication times for the Square-Corner and Straight-Line Partitionings on a star topology. The Square-Corner Partitioning has a lower communication time than the Straight-Line Partitioning in all cases. On average the Square-Corner Partitioning results in a reduction in communication time of approximately 40%.

A plot of the communication volumes agrees well with Figure 6.14 with one exception. The Square-Corner and Straight-Line communication volumes converge as s_2 and $s_3 \to 25$. The reason that the communication times do not converge is due to the necessarily sequential component of the Straight-Line Partitioning's communication schedule. This component can not make

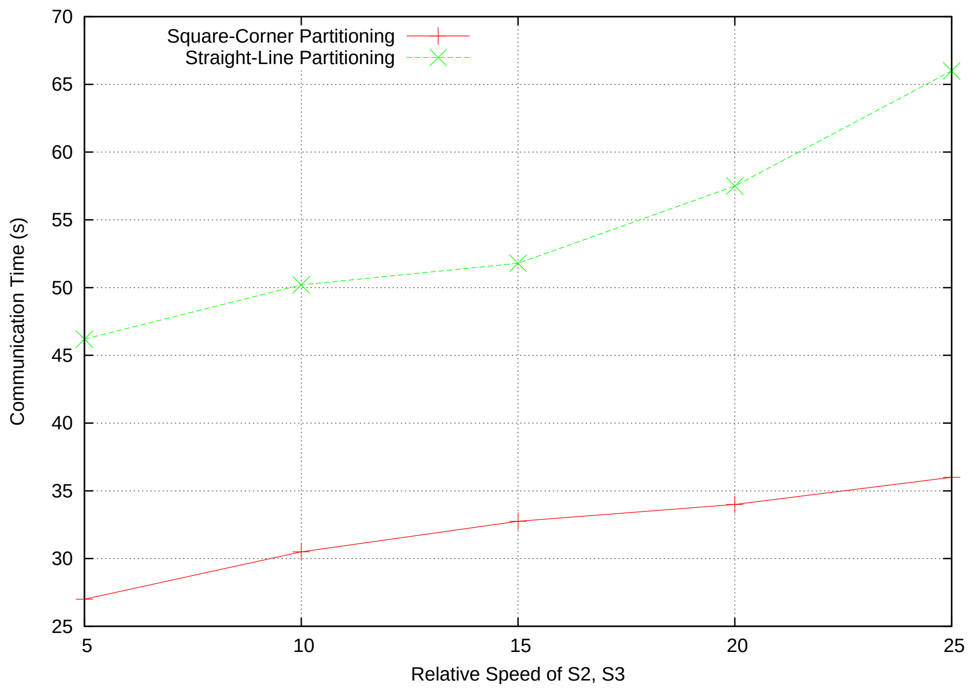


Figure 6.14: Communication times for the Square-Corner and Straight-Line Partitionings on a star topology. Relative speeds $s_1 + s_2 + s_3 = 100$.

use of network advantages such as Ethernet's full duplex. Experiments "illegally" altering the rectangular partitioning's communication schedule (by de-serializing necessarily serial communications) confirm this.

Figure 6.15 shows a plot of the execution times for the Square-Corner and Straight-Line Partitionings on the star topology. For the Square-Corner Partitioning two values are plotted, the execution time obtained with no overlapping of communication and computation, and the values obtained with overlapping. It is seen that with no overlapping (only taking into account the communication differences) the execution time for the Square-Corner Partitioning is on average 14% less than that of the Straight-Line, and that the reduction in communication times seen in Figure 6.14 directly influence the execution times.

The introduction of overlapping communication and communications significantly influences the performance of the Square-Corner Partitioning. For a ratio of $90 : 5 : 5$ it is 38% faster than the rectangular partitioning. As the ratio approaches $50 : 25 : 25$, the amount of overlap possible approaches zero, and the execution times of the Square-Corner Partitioning with and without overlap converge.

Figure 6.16 shows the communication times for the fully connected topology.

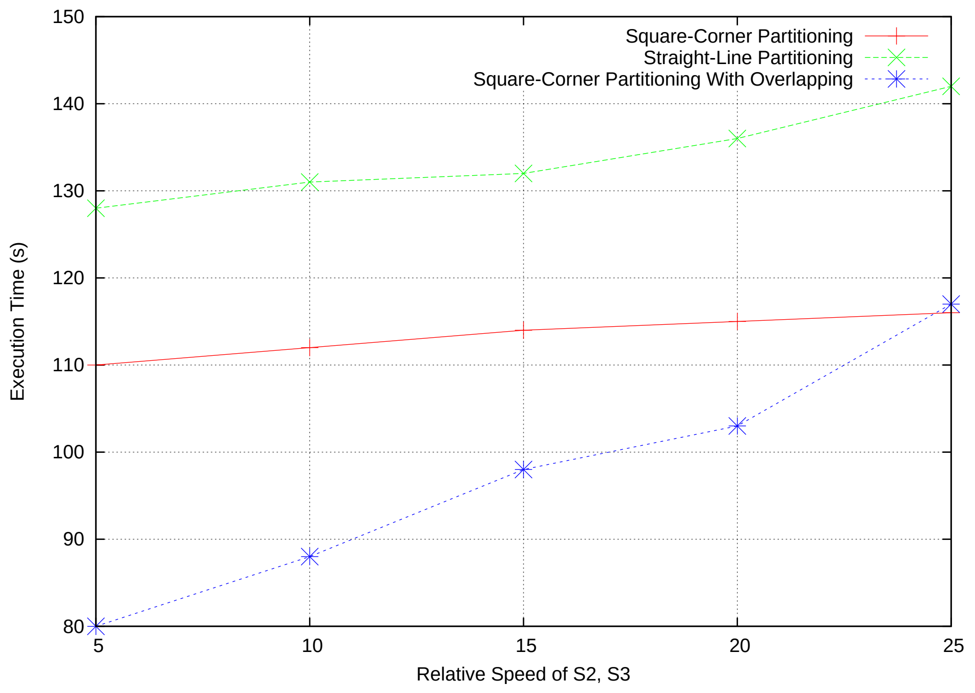


Figure 6.15: Execution times for the Square-Corner, Square-Corner with Overlapping and Straight-Line Partitionings on the star topology. Relative speeds $s_1 + s_2 + s_3 = 100$.

A notable aspect is that the Straight-Line Partitioning's communication times decrease despite the fact that it is dealing with increasingly higher communication volumes. The reason for this is that the total volume of communication for this partitioning increases much slower than that of both the Straight-Line Partitioning on the star and the Square-Corner Partitioning. It increases so much slower that the increased benefit of more computational parallelism (as s_2 and s_3 get closer to s_1) outweighs the higher communication burden. Still, for ratios more heterogeneous than about 80 : 10 : 10, the Square-Corner Partitioning has a lower total volume of communication, and therefore lower communication times.

Figure 6.17 shows the overall execution times for the Square-Corner and Straight-Line Partitionings on a fully connected topology. Again the Square-Corner partitioning is shown with and without overlapping. Again, without overlapping the execution times are directly affected by the communication times. For ratios more heterogeneous than about 80 : 10 : 10, the Square-Corner Partitioning out performs the Straight-Line. The introduction of overlapping again significantly influences the performance of the Square-Corner Partitioning. For a ratio of 90 : 5 : 5 the Square-Corner Partitioning is 30% faster that the rectangular. Additionally, the range of ratios where the Square-

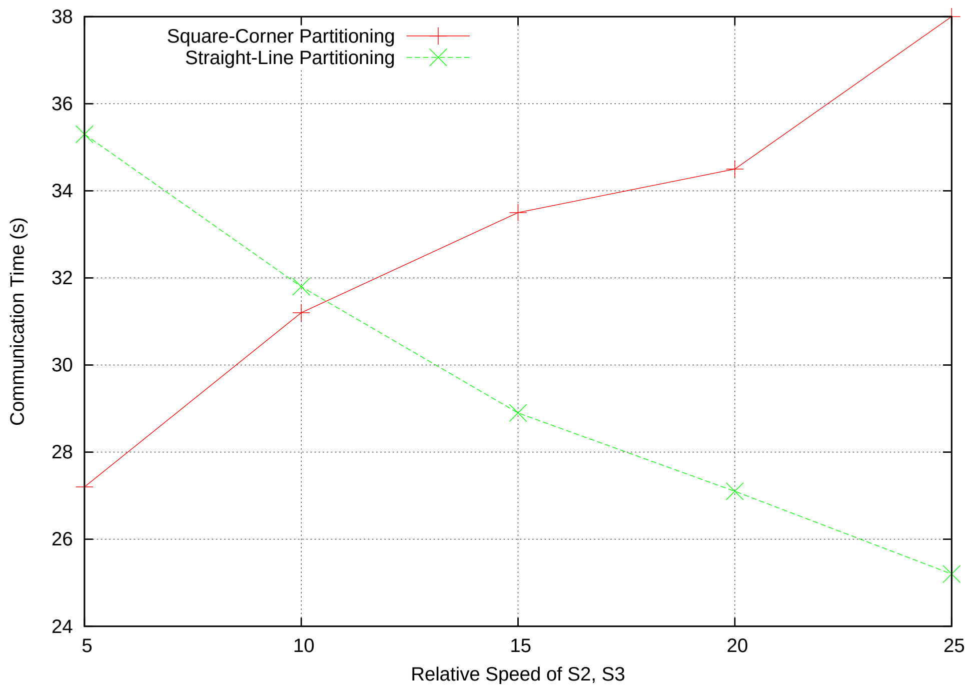


Figure 6.16: Communication times for the Square-Corner and Straight-Line Partitionings on a fully connected topology. Relative speeds $s_1 + s_2 + s_3 = 100$.

Corner Partitioning is faster than the rectangular is broadened from about $80 : 10 : 10$ to about $60 : 20 : 20$. Similar results were seen at other bandwidths, values of N, and power ratios, including where $s_2 \neq s_3$.

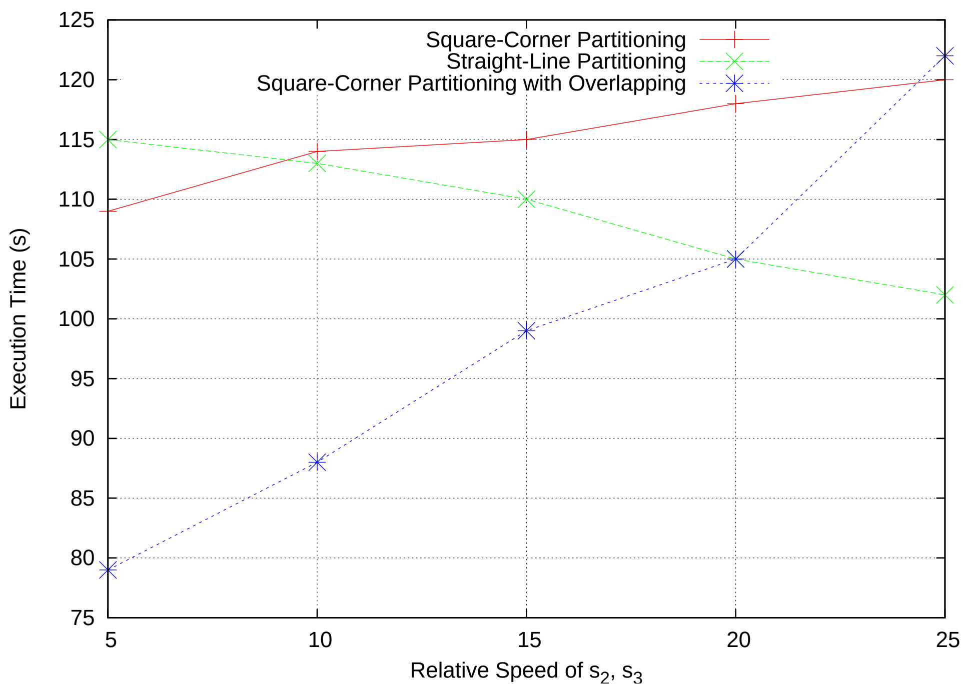


Figure 6.17: Execution times for the Square-Corner, Square-Corner with Overlapping and Straight-Line Partitionings on a fully connected topology. Relative speeds $s_1 + s_2 + s_3 = 100$.

6.3 Grid'5000 Experiments

To experiment with the Square-Corner Partitioning on a large scale, geographically distributed scientific computing platform we chose three sites on Grid'5000, arranged in a star. Actually it is not the physical topology but the fact that the Square-Corner Partitioning forces Clusters 2 and 3 to communicate through Cluster 1 that "creates" the star topology.

We chose the sites Orsay, Rennes, and Sophia. Orsay was chosen as the center cluster (Cluster 1) as it has the greatest overall power. The desired power ratios were achieved by varying the number of CPUs and cores used at each site. It was not always possible to achieve perfect ratios but they were achieved within a few percent. To determine relative power ratios, test code consisting of serial matrix matrix multiplications were carried out on each type of machine.

Figure 6.18 shows the communication times for the Square-Corner and Straight-Line Partitionings. Both curves tend towards higher communication times with higher ratios (and therefore increased parallelism). This is attributed to the increase in the TVC with increased ratios. Instability in the results could be due to shared communication links between sites. In addition, we noticed that in a large multi-user environment, network traffic serves to affect the communication of each partitioning without discrimination, unlike in simulations where only one user is allowed to fully exploit or fully saturate one communication link. In the latter scenario it is clear that the partitioning with a higher TVC will saturate a link before one with a lower TVC.

Figure 6.19 shows the corresponding execution times. As the power ratio between clusters increases, so does parallelism. It seems that the links between sites are fast enough for the increased parallelism to decrease overall execution time despite a higher TVC. The inter-site bandwidth is 10Gb/s, however this does not take into account links internal to each site running at different speeds which do represent a possible bottleneck.

In all cases the Square-Corner Partitioning had a lower TVC, lower communication time, and lower execution time than the Straight-Line Partitioning.

6.4 Conclusion

We extended the Square-Corner Partitioning from the two cluster scenario in Chapter 5 to three interconnected clusters. This partitioning has two advan-

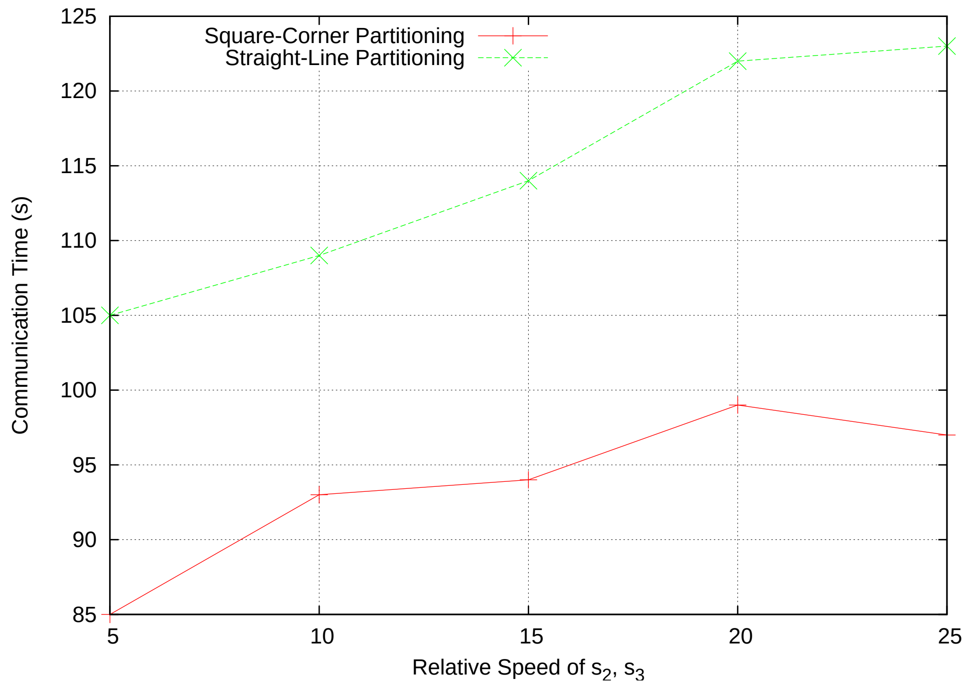


Figure 6.18: Communication times for the Square-Corner and Rectangular Partitionings on a star topology. Relative speeds $s_1 + s_2 + s_3 = 100$.

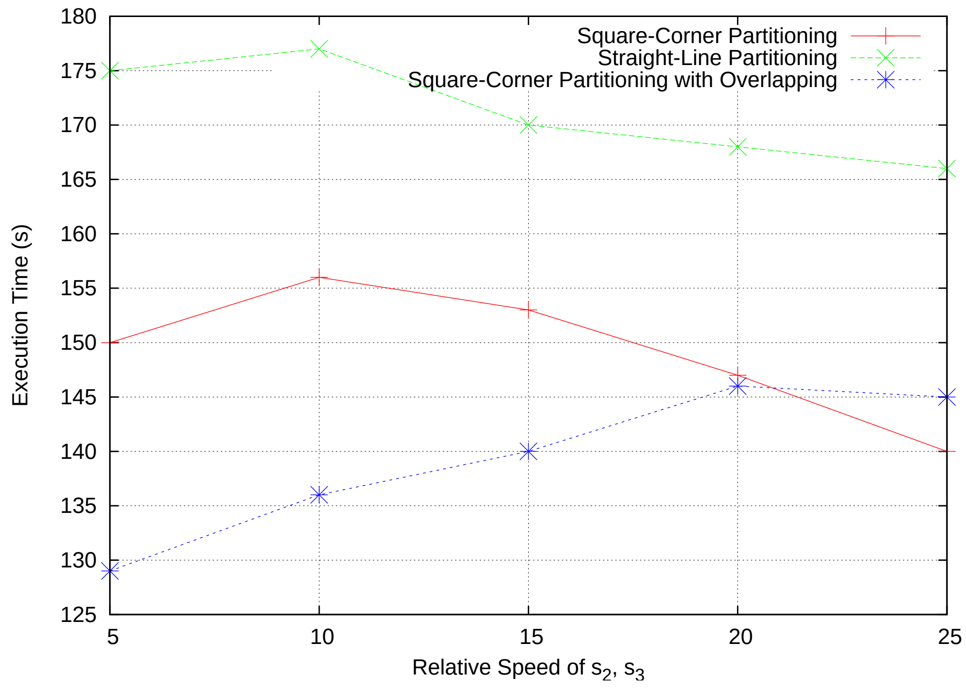


Figure 6.19: Execution times for the Square-Corner, Square-Corner with Overlapping and Straight-Line Partitionings on a star topology. Relative speeds $s_1 + s_2 + s_3 = 100$.

tages over existing partitionings. First it reduces communication time due to a lower total volume of communication and a more efficient communication schedule. The total volume of communication is shown to approach the known lower bound unlike existing partitionings. Second it allows for the overlapping of communication and computation.

To determine the viability of this partitioning we modeled the three cluster topology with three processors performing matrix matrix multiplications. Compared to more general partitionings which result in simple Straight-Line Partitionings, the Square-Corner Partitioning is shown to reduce the total volume of communication in all cases for the star topology and in most cases for a fully connected topology. We experimentally show that this directly translates to lower communication times. In the case of the star topology, we show average reductions in communication time of about 40%.

Further experimentation shows that this reduction in communication time directly translates to a reduction in the overall execution time, aided by a more efficient communication schedule. Overlapping communication and computation brings further benefit, in both reducing the execution times significantly, and broadening the ratio range where the Square-Corner Partitioning out performs the Straight-Line Partitioning on a fully connected topology. MPI experiments demonstrate reductions in execution times of up to 38%.

CHAPTER

SEVEN

MAX-PLUS ALGEBRA AND DISCRETE EVENT SIMULATION ON PARALLEL HIERARCHAL HETEROGENEOUS PLATFORMS

7.1 Summary

In this chapter we demonstrate possible areas of application for the Square-Corner Partitioning and the theoretical results of this thesis. We do this by exploring computing max-plus algebra operations and discrete event simulations on parallel hierarchal heterogeneous platforms. When performing such tasks on heterogeneous platforms parameters such as the total volume of communication and the top-level data partitioning strategy must be carefully taken into account. Choice of the partitioning strategy is shown to greatly affect the overall performance of these applications due to different volumes of inter-partition communication that various strategies impart on these operations. Max-plus algebra is regularly used in discrete event simulations and many other important computational applications thus warranting the exploration of and improvement upon the running times of basic max-plus operations on parallel platforms which are inherently hierarchal and heterogeneous in nature. The main goal of this chapter is to present benefits waiting to be exploited by the use of max-plus algebra operations on these platforms and thus speeding up more complex and quite common computational topic areas such as discrete event simulation.

7.2 Introduction

This chapter presents results of running fundamental max-plus algebra (MPA) operations and discrete event simulations (DES) on parallel hierarchal heterogeneous platforms. The top-level data partitioning strategy is shown to greatly affect the overall performance of these applications due to different volumes of inter-partition communication that various strategies impart on these operations. The Square-Corner Partitioning in particular is shown to reduce the execution times of these operations more than other, more traditional strategies.

Max-plus algebra is a relatively new field of mathematics which grew from the advent of tropical geometry in the early 1980s and has since been shown to have many diverse application areas. MPA is (along with min-plus algebra) a sub-category of tropical algebra. MPA obeys most laws of basic algebra with the operations of addition ($a + b$) and multiplication ($c \times d$) replaced by the operations $max(a, b)$ and addition ($c + d$) respectively. Min-plus algebra is similar, but with the maximum operation replaced with a minimum operation.

Discrete event simulation is an extremely expansive area of continuing and intense research which may broadly be characterised as a collection of techniques and methods which when applied to the study of discrete-event dynamical systems generate sequences which characterize system behavior. This includes modeling concepts for abstracting essential features of a system into a set of precedence and mathematical relationships, which can be used to describe the system and more importantly for system design, to predict behavior, performance, and drawbacks/bottlenecks. DES is used to design and model a great number of systems including travel timetables, operating systems, communication networks, autonomous guided vehicles, CPUs and other complex systems. There are many approaches to designing DES including Petri nets, alphabet based approaches, perturbation methods, control theoretic techniques and expert systems design. Recently MPA and other techniques involving both logical and algebraic components have shown to be capable of simplifying simulations while maintaining the desired outputs (Kirov, 2009). One such method is explored later in this chapter.

The Square-Corner Partitioning (Chapter 5) is a top-level partitioning method for parallel hierarchal heterogeneous computing which when applied to problems such as matrix matrix multiplication (MMM) and all linear algebra kernels reducible to MMM, optimally reduces the total volume of communication (TVC) between computing entities (processors, clusters, etc.) when the power ratios between entities meet certain, yet numerous and very common ratios. This partitioning also has other benefits including simpler communication schedules and the possibility of overlapping communication and computation (Becker and Lastovetsky, 2006, 2007). As this chapter demonstrates the SCP can extend these benefits to many application areas.

7.3 Max-Plus Algebra

Max-plus algebra is a relatively new field in mathematics, dating back approximately 30 years. It has since been shown to have several application areas such as discrete event simulation, dynamic programming, finite dimensional linear algebra, modeling communication networks, operating systems, combinatorial optimization, solving systems of linear equations, biological sequence comparisons and even problems such as crop rotation (Comet, 2003; Gaubert and Plus, 1997; Heidergott *et al.*, 2006; Kirov, 2009; Tacconi and Lewis, 1997). In many scientific and computational applications the structure of MPA matrix multiplication is an important aspect (Johnson and Kambites, 2009). Additionally, higher powers of MPA matrices are of significant interest and necessary in many application areas (De Schutter and De Moor, 1999; Kirov, 2009).

MPA is based on replacing the "normal" algebraic addition operation with a binary *max* function, and the "normal" multiplication operation with addition. Formally, if we define $\epsilon = -\infty$, $e = 0$, and denote $\mathbb{R}_{max}$ to be the set $\mathbb{R} \cup \{\epsilon\}$, then for elements $a, b \in \mathbb{R}_{max}$, the operations $\oplus$ and $\otimes$ are defined respectively by the following:

$$a \oplus b \overset{def}{=} max(a,b) \text{ and } a \otimes b \overset{def}{=} a + b \tag{7.1}$$

Therefore, $a \oplus \epsilon = max(\epsilon, a) = a$ and $a \otimes \epsilon = \epsilon + a = \epsilon$. We can now formally define max-plus algebra as

$$\mathcal{R}_{max} = (\mathbb{R}_{max}, \oplus, \otimes, \epsilon, e) \tag{7.2}$$

The basic algebraic rules of max-plus algebra are:

- Associativity

  $(A \oplus B) \oplus C = A \oplus (B \oplus C)$

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

- Commutativity

$$A \oplus B = B \oplus A$$

- Distributivity

$$(A \oplus B) \otimes C = A \otimes C \oplus B \otimes C$$

In general the $\otimes$ operation has precedence over the $\oplus$ operation.

MPA matrices are denoted $\mathbb{R}_{max}^{m \times n}$, where m and n are the matrix dimensions. For the MPA matrices $A \in \mathbb{R}_{max}^{m \times n}$ and, $B \in \mathbb{R}_{max}^{n \times p}$ the matrix product $A \otimes B$ is the same as in normal linear algebra, but following the operation substitutions in Definitions 7.1. That is every "$+$" operation is replaced with a $\oplus$ operation, and every "$\times$" (or "$\cdot$") operation is replaced with a $\otimes$ operation. From this, matrix powers are straight-forward, and represented $A^{\otimes k}$ for the k^{th} power of A. As max-plus matrix matrix multiplication and max-plus matrix powers are integral parts of many applications of MPA we further discuss this in Section 7.5.1.

7.4 Discrete Event Simulation

Discrete event simulation is a very broad and well-studied field and therefore the purpose of this section is to acquaint the reader with the specific technique utilized in this chapter. Briefly, DES is a collection of techniques and methods which when applied to the study of a discrete-event dynamical system generates sequences which characterize the system behavior. This includes modeling concepts for abstracting essential features of the system into a set of precedence and mathematical relationships, which can be used to describe the system and more importantly for design, and to predict its behavior, performance, and drawbacks/bottlenecks. For more see any good DES text such as Fishman (2001).

As most DES algorithms are computationally intensive, efforts to parallelize them are numerous. The complexity of most practical DES algorithms however poses numerous obstacles in effective and efficient parallelization. Amongst these are synchronization and timing inconsistencies, synchronous vs. asynchronous simulation, deadlock avoidance and detection, conservative vs. optimistic simulation, recovery strategies, and memory management to name a few (Ferscha and Tripathi, 1996).

In Section 7.5.2 we present results of the parallelization of a DES modeling technique which although as presented in Tacconi and Lewis (1997) is sequential, lends itself to parallelization due to a computationally intensive algorithmic core which can be efficiently ported to hierarchal heterogeneous parallel platforms. This core is very similar to a max-plus matrix matrix multiplication, but using logical 'and' and 'or' operations instead of max-plus operations. We employ this technique—called the Matrix Discrete Event Model (MDEM)—using MPI and utilizing the SCP (Becker and Lastovetsky, 2006, 2007), for the core routine.

7.4.1 The Matrix Discrete Event Model

The design, simulation, and analysis of large-scale, complex systems using existing DES techniques such as Petri nets, alphabet-based approaches, perturbation methods, control theoretic techniques, and expert systems design are often difficult to implement and are very labor and time intensive. The MDEM is a hybrid system with logical and algebraic components that seeks to make these processes more efficient. Although the examples in Tacconi and Lewis (1997) focus on manufacturing systems (see Figure 7.1), the formulation is also applicable to many DES situations such as travel timetables, operating systems, communication networks, autonomous guided vehicles, operating systems, and many others. Clearly the number of degrees of freedom, state possibilities, and general complexity of such systems often result in simulations with several thousands (or more) event components.

The MDEM approach is a rule-based model described by four equations: the Model State Equation, Start Equation, Resource Release Equation, and the Product Output Equation:

Matrix Discrete Event Model State Equation:

$$\overline{x} = F_v \times \overline{v}_c + F_r \times \overline{r}_c + F_u \times \overline{u} + F_D \times \overline{v}_D \tag{7.3}$$

Start Equation:

$$v_s = S_v \times x \tag{7.4}$$

Resource Release Equation:

$$r_s = S_r \times x \tag{7.5}$$

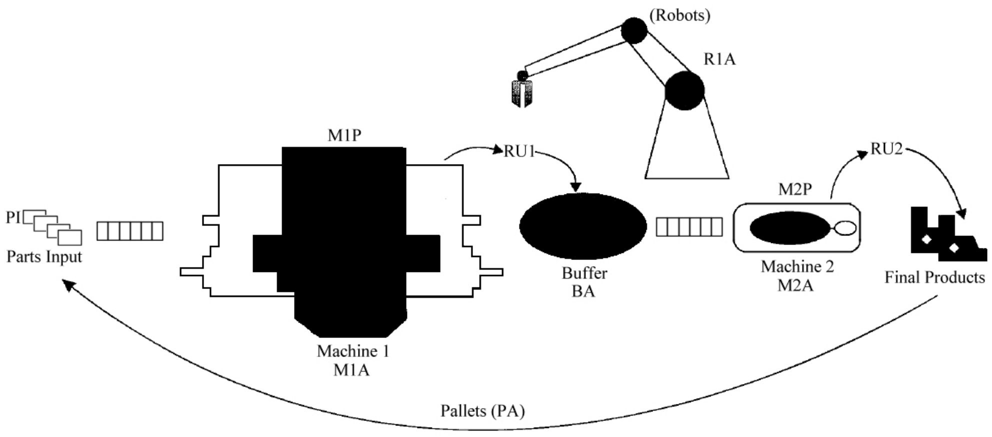


Figure 7.1: An MDEM workcell. From Tacconi and Lewis (1997).

Product Output Equation:

$$y = S_y \times x \tag{7.6}$$

Each of these equations are *logical*, only using *or*, *and*, and *negation* operations. Additionally, all vectors and matrices in these equations are binary. For instance, the vector which is the output of the start equation contains a '1' for each job which is to be started at the given state of the simulation, and a '0' otherwise.

The simulation itself is carried out by first calculating initial conditions from the description of the system. The core of the simulation is carried out by the successive calculation of 'firing vectors' which carry the simulation to the next state. This amounts to the repeated calculation of an equation which has the form of a matrix matrix multiplication except that since the approach of the MDEM technique is hybrid—having both algebraic and logical components—the algebraic multiplication and addition operations are replaced with logical 'or' and 'and' operations respectively. It is this step that constitutes the bulk of the calculation time for the MDEM technique as all other calculations only need to be carried out once.

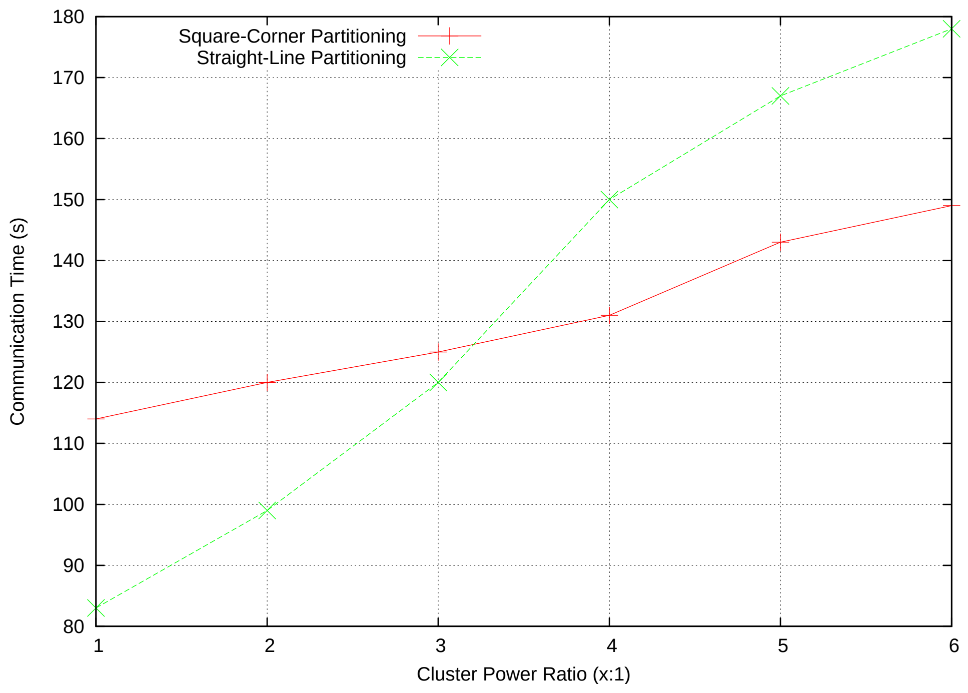


Figure 7.2: Comparison of the total communication times for the square-corner and straight-line partitionings for power ratios ranging from $1:1-6:1$. Max-Plus MMM, $N=7,000$.

7.5 MPI Experiments

In this section we present results of performing MPA matrix matrix multiplications and an MDEM discrete event simulation utilizing both the Square-Corner Partitioning and the Straight-Line Partitioning. Hardware setup is similar to that in Section 5.8, only differing in power ratios.

7.5.1 Max-Plus MMM Using the Square-Corner Partitioning

As outlined in Section 7.3 we experimented with performing a MPA MMM using *C* and MPI. We used a two cluster heterogeneous platform with power ratios between clusters ranging from $1:1$ to $6:1$. For all experiments we use double precision and $N=7,000$. The local interconnect was 2Gb/s Infiniband and the inter-cluster interconnect was 1Gb/s Ethernet. Figure 7.2 shows the communication times for both the Square-Corner and Straight-Line Partitionings.

Communications are serialized in code. As expected the SCP does not show improvement in communication time until the power ratio is $3:1$, as this is when the SCP results in a lower TVC. For ratios greater than this (as the system

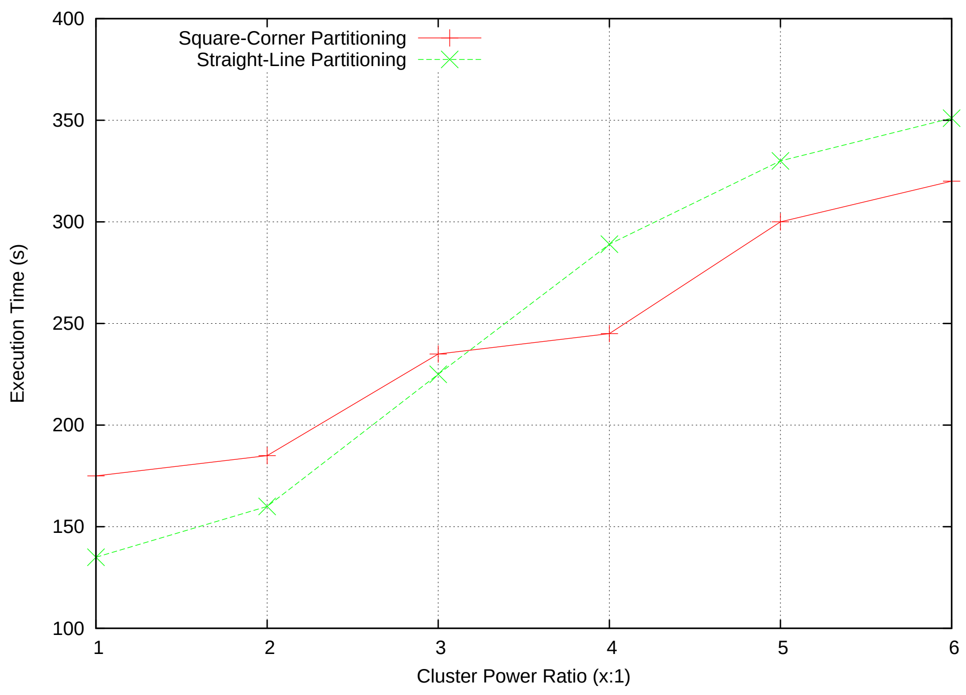


Figure 7.3: Comparison of the total execution times for the square-corner and straight-line partitionings for power ratios ranging from $1:1-6:1$. Max-Plus MMM, $N = 7,000$.

becomes more heterogeneous), the gap between the two communication times widens, and would be expected to widen. For a detailed analysis see Becker and Lastovetsky (2006).

Figure 7.3 shows the resulting difference in execution times between the SCP and SLP. As expected we also see the crossover around ratio $3:1$, and note that the lower TVC that the SCP brings also results in lower execution times for ratios above $3:1$. Again this gap would be expected to widen. It is worth noting that the execution time is higher than was initially expected but this is due to the lack of an optimizing library for MPA MMM, unlike normal MMM which can benefit from the `dgemm` routine in the BLAS (Blackford *et al.*, 2002), and ATLAS (Whaley and Dongarra, 1999).

It is worth noting that since carrying out a matrix power operation A^n amounts to nothing more than n repeated matrix multiplications, carrying out matrix power operations would also benefit from the above.

7.5.2 The Square-Corner Partitioning for Discrete Event Simulation

In Section 7.4 we outlined the MDEM model for discrete event simulations. We use the same experimental platform as in Section 7.5.1 to demonstrate results on a parallel, heterogeneous platform of the MDEM model. We utilize both the SLP and the SCP for the core routine which is a matrix "and/or" multiplication. We generate the initial conditions so that the core routine involves a large system ($N = 7,000$). All initial calculations and cleanup are carried out on a single processor as these calculations are carried out only once and make up a small percentage of the overall execution time and are not parallelizable.

Figure 7.4 shows the communication times which are relatively low due to the use of `char` as the data type (all data is binary). Due to the nature of the MDEM all communications are serialized. The results overall agree with theory, with the $3:1$ ratio crossover occurring. Figure 7.5 shows the execution times for carrying out the described DES using both partitioning techniques. It can be seen that the use of the SCP for the core kernel of the MDEM DES algorithm significantly reduces the execution time for ratios above $3:1$, but less so due to the lower communication time compared to the MPA MMM. Again the expected crossover occurs near the ratio of $3:1$. The overall shape of the curves are similar to those of Section 7.5.1 as the "and/or" MMM in the MDEM involves a similar computational cost as the max-plus MMM.

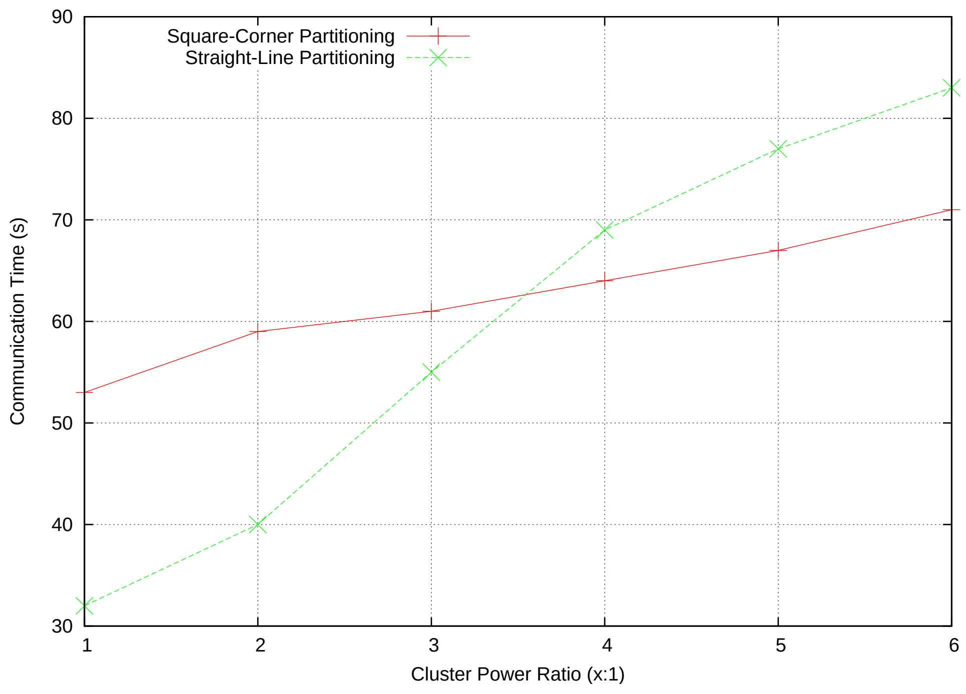


Figure 7.4: Comparison of the communication times for the MDEM DES model for both the Square-Corner and Straight-line Partitionings, $N = 7,000$.

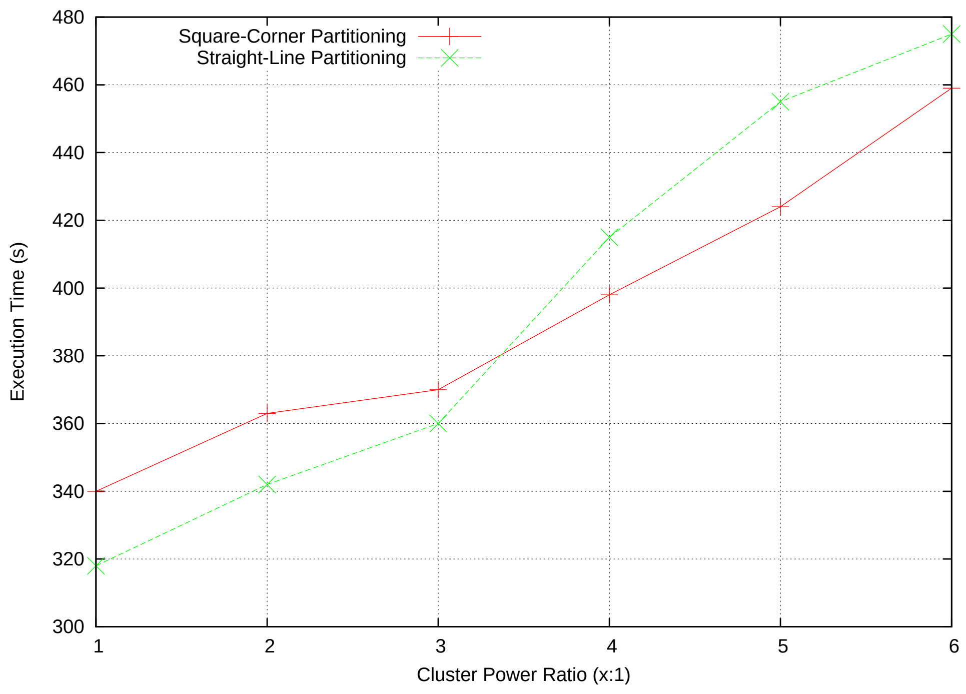


Figure 7.5: Comparison of the total execution times for the MDEM DES model for both the square-corner and straight-line partitionings, $N = 7,000$.

7.6 Conclusion

In this chapter we explored computing max-plus algebra matrix operations and a MDEM discrete event simulation on parallel hierarchal heterogeneous platforms. We found that the initial top-level data partitioning—particularly the use of the square-corner partitioning—significantly affects overall execution time due to the total volume of inter-cluster communication involved. Notably the square-corner partitioning out performed the straight-line partitioning in all cases where the power ratio between clusters was $\geq 3 : 1$. Future work involves applying similar strategies to speed up more complex routines, perhaps with more complicated and heavyweight communication loads, on parallel hierarchal heterogeneous platforms and experimenting on other parallel hierarchal heterogeneous networks.

CHAPTER

EIGHT

MOVING AHEAD – MULTIPLE PARTITIONS AND RECTANGULAR MATRICES

8.1 Multiple Partitions

We have seen in Chapters 4 - 7 that the Square-Corner Partitioning proves to have a lower total volume of communication and a lower execution time than the Straight-Line Partitioning in many cases for two and three partitions. The question is, can the Square-Corner Partitioning be extended to more than three partitions? We have gained some insight into this question based on the investigation into partition configurations such as those in Figure 8.1.

Configurations such as those in Figure 8.1 are not possible extensions of the Square-Corner Partitioning for three partitions. This is due to an increase in

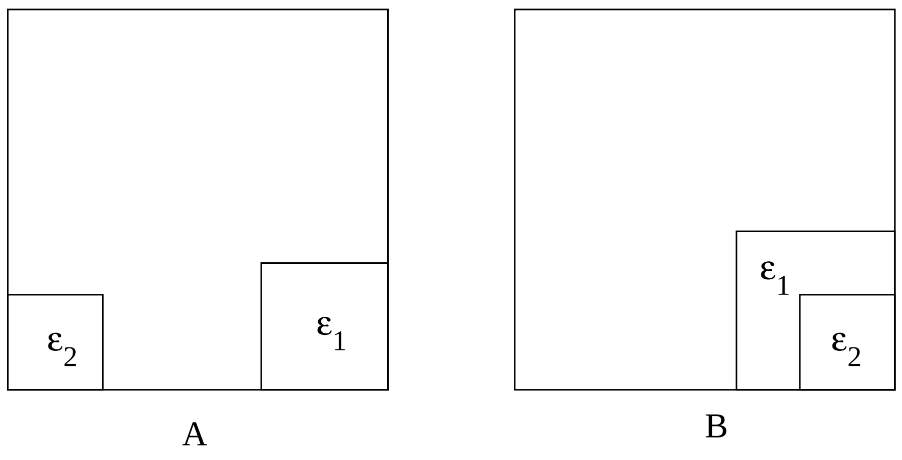


Figure 8.1: Examples of non-Square-Corner Partitionings investigated.

the number of communication steps and/or the TVC that violate the definition of the Square-Corner Partitioning. We conclude therefore that similar configurations would not be suitable for a number of partitions greater than three as well. The only configuration we see viable for an extension to four partitions is one such as that in Figure 8.2.

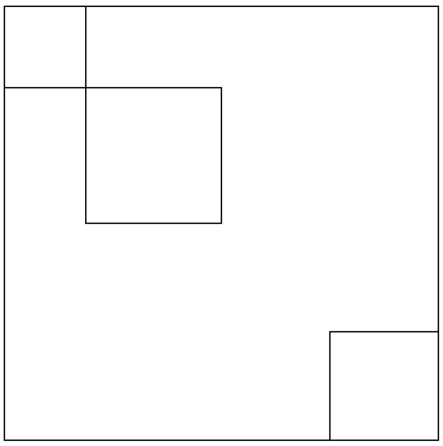


Figure 8.2: An example of a four partition Square-Corner Partitioning

Moving beyond four partitions, the Square-Corner Partitioning takes on a "diagonal" form such as that in Figure 8.3. This is because the Square-Corner Partitioning was designed to have as simple a structure as possible, and any other configuration has the possibility of violating the definition of a Square-Corner Partitioning. The name Square-Corner Partitioning is still appropriate, as the partitions "work" their ways from corner to corner.

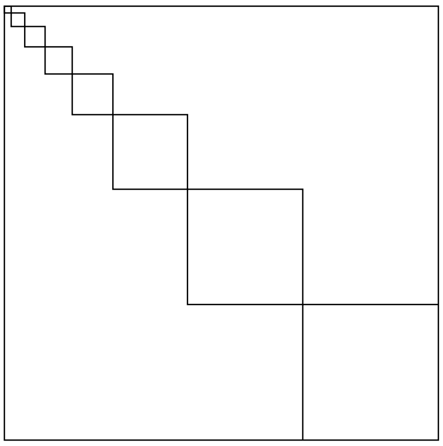


Figure 8.3: An example of a multiple partition Square-Corner Partitioning

In order to keep within our definition of the Square-Corner Partitioning, any extension to more than three partitions must:

1. Contain all square partitions except one, possibly two in the following

case:

- Two non-square partitions may be created only in the cases such as that depicted in Figure 8.3, where square partitions completely bisect the matrix being partitioned, in which case the two pieces formed (lower-left and upper-right in Figure 8.3) will represent one logical partition, as they are owned by the same cluster.

2. No square partitions may communicate with each other. In other words no two square partitions can interrupt the same row or column.
3. Maintain a one-to-one mapping between clusters and (logical) partitions
4. Map (logical) partitions to clusters so that the area of partition s_i is proportional to the speed of cluster c_i, where $s_i \in (s_1, s_2, \ldots, s_p)$ where p is the number of partitions and s_i are the areas of the partitions in non-decreasing order, and $c_i \in (c_1, c_2, \ldots, c_p)$ where c_i are the clusters being mapped in non-decreasing order of relative speed
5. Partition all matrices A, B, C in the same way

Clearly as the number of clusters (and therefore partitions) increases the number of possible network topologies will increase as well. It is believed that restricting the number and types of topologies will increase the performance of the Square-Corner Partitioning relative to Straight-Line (rectangular) partitionings in many cases.

8.2 The Square-Corner Partitioning for Partitioning Rectangular Matrices

In order to be as general as possible we are interested in the applicability of the Square-Corner Partitioning to rectangular matrices, not just the restricted case of square matrices. It is believed that the popular test case of the unit square is fitting for rectangular matrices because any partitioning that is optimal on the unit square can be scaled to a rectangle and remain optimal (Lastovetsky and Dongarra, 2009).

We will begin in the most general case. We assume only two clusters, each owning one partition. As always each partition has an area proportional to the speed of the cluster which owns it. We then create two partitions, one partition is a rectangle located in any corner of the matrix, and the other partition is polygonal—the rest of the matrix with the area of the first partition removed from it.

To make the case as general as possible we start with a real-valued rectangular area and partition it into two by creating a real-valued rectangular partition within the area. We place three restrictions on this system.

- There are two partitions to be created. This necessitates defining only one interior partition, as the other partition will be the area which lies outside the partition defined but inside the rectangular area being partitioned.
- The area to be partitioned and the interior partition are rectangular.
- The area of the two partitions are proportional to the speeds of the clusters owning them.

This gives us a rectangular area of dimension $x \times y$ and an interior rectangular partition $\alpha \times \beta$ as in Figure 8.4.

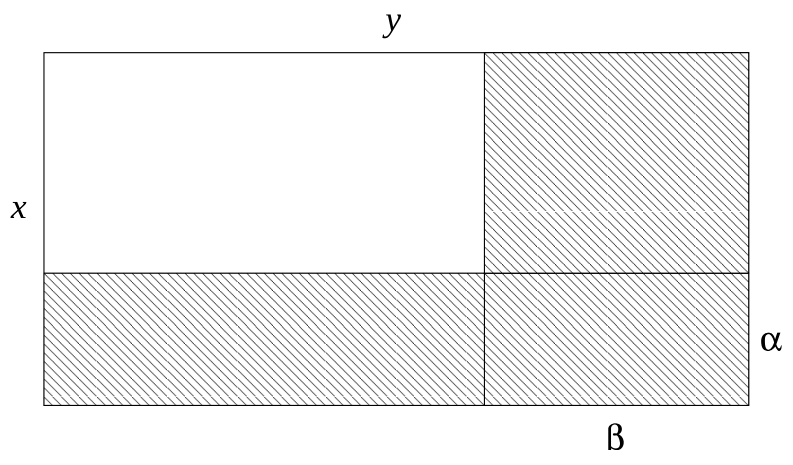


Figure 8.4: An example of a rectangular matrix with one rectangular ($\alpha \times \beta$) partition and one polygonal ($x \times y - \alpha \times \beta$) partition.

In Chapter 5 we showed that the TVC was minimized when the area equal to the following was minimized:

height of small partition × width of matrix + width of small partition × height of matrix

In Figure 8.4, this is represented by the shaded area, which equates to Γ below:

$$\Gamma = \alpha \times y + \beta \times x \tag{8.1}$$

Proposition 8.1: $\Gamma = \alpha \times y + \beta \times x$ is minimized when the dimensions of the rectangular partition $\beta \times \alpha$ is scaled to the matrix $x \times y$ such that $\frac{\alpha}{\beta} = \frac{x}{y}$.

Proof: We will prove Proposition 8.1 by showing that whenever the small partition is not scaled as stated, Γ will always be larger.

Equation 8.2 describes the state when the rectangular partition is not scaled, i.e. when its height α is decreased by some quantity $c < \alpha$, and its width β increased by some other quantity $c' < \beta$, while maintaining the same area $\alpha \times \beta$.

$$\alpha \times \beta = (\alpha - c)\left(\beta + c'\right) \tag{8.2}$$

We determine c' (for simplification later), then we start with the equation for Γ, but substitute in the values on the RHS of Equation 8.2. We then have a function $\mathbb{S}(c)$ which represents the new system, including the non-scaled partition, where c is the amount that the previously scaled partition has been changed by. Note that a function in terms of only c is satisfactory as c is proportional to c'. We show that when $c = 0$, $\frac{d\mathbb{S}}{dc} = 0$, and that at this point $\frac{d^2\mathbb{S}}{dc^2}$ is positive. It is then shown that any change in c, either positive or negative, will increase Γ (within the definition of the problem), thus concluding the proof.

$$\begin{aligned}
\alpha \times \beta &= \alpha \times \beta + \alpha \times c' - c \times \beta - c \times c' \\
c \times \beta &= \alpha \times c' - c \times c' \\
c \times \beta &= c'(\alpha - c) \\
c' &= \frac{c \times \beta}{\alpha - c}
\end{aligned}$$

$$
\begin{aligned}
\Gamma &= x \times \beta + y \times \alpha \\
\mathbb{S}(c) &= x(\beta + c') + y(\alpha - c) \\
&= x \times \beta + x \times c' + y \times \alpha - y \times c \\
&= x \times \beta + \frac{x \times c \times \beta}{\alpha - c} + y \times \alpha - y \times c \\
\frac{d\mathbb{S}}{dc} &= \frac{x \times \alpha \times \beta}{(\alpha - c)^2} - y \\
&= \frac{x \times \alpha \times \beta}{\alpha^s - 2 \times \alpha \times c + c^2} - y \\
&= x \times \alpha \times \beta - y \times \alpha^2 + 2 \times \times \alpha \times c \times y - y \times c^2 \\
&= \frac{x \times \alpha^2 \times y}{x} - y \times \alpha^2 + 2 \times \alpha \times c \times y - y \times c^2 \\
&= 2 \times \alpha \times c \times y - y \times c^2 \therefore \text{ when } c = 0, \frac{d\mathbb{S}}{dc} = 0 \\
\frac{d^2\mathbb{S}}{dc^2} &= 2 \times \alpha \times y - 2 \times c \times y \\
&= 2 \times y(\alpha - c) \\
c &\overset{def}{<} \alpha \therefore \frac{d^2\mathbb{S}}{dc^2} \text{ is positive}
\end{aligned}
$$

It must now be shown that for any $c > 0, \frac{d^2\mathbb{S}}{dc^2}$ is positive, and that for any $c < 0, \frac{d^2\mathbb{S}}{dc^2}$ is negative, to show that the minimum in $\frac{d^2\mathbb{S}}{dc^2}$ when $c = 0$ is not a local, but a global minimum.

$$
\begin{aligned}
\frac{d^2\mathbb{S}}{dc^2} &= 2 \times \alpha \times c \times y - y \times c^2 \\
&= c \times y \times (2 \times \alpha - c) \\
c &\overset{def}{<} \alpha \\
&\therefore \text{ when } c \text{ is positive, } \frac{d^2\mathbb{S}}{dc^2} \text{ is positive} \\
\text{and } & \text{ when } c \text{ is negative, } \frac{d^2\mathbb{S}}{dc^2} \text{ is negative}
\end{aligned}
$$

Q.E.D.

Thus we have proven that the area $\Gamma = \alpha \times y + \beta \times x$ is a minimum when the rectangular partitioning $\alpha \times \beta$ is scaled so that $\frac{\alpha}{\beta} = \frac{x}{y}$, but does minimizing Γ minimize the total volume of communication when we are *multiplying* non-square matrices? The answer is no, as we will see in the next section.

8.3 The Square-Corner Partitioning for MMM on Rectangular Matrices

Figure 8.5 shows that the TVC is at a minimum when we are dealing with a *rectangular matrix matrix multiplication* as opposed to *partitioning a rectangular matrix* is

$$\alpha \times n + \beta \times n = n \times (\alpha + \beta) \tag{8.3}$$

where n is one of the three given matrix dimensions, where a $m \times n$ matrix multiplied by a $n \times p$ matrix results in a $m \times p$ product, and $\alpha \times \beta$ is the area of the partition placed in the corner of the rectangular matrices.

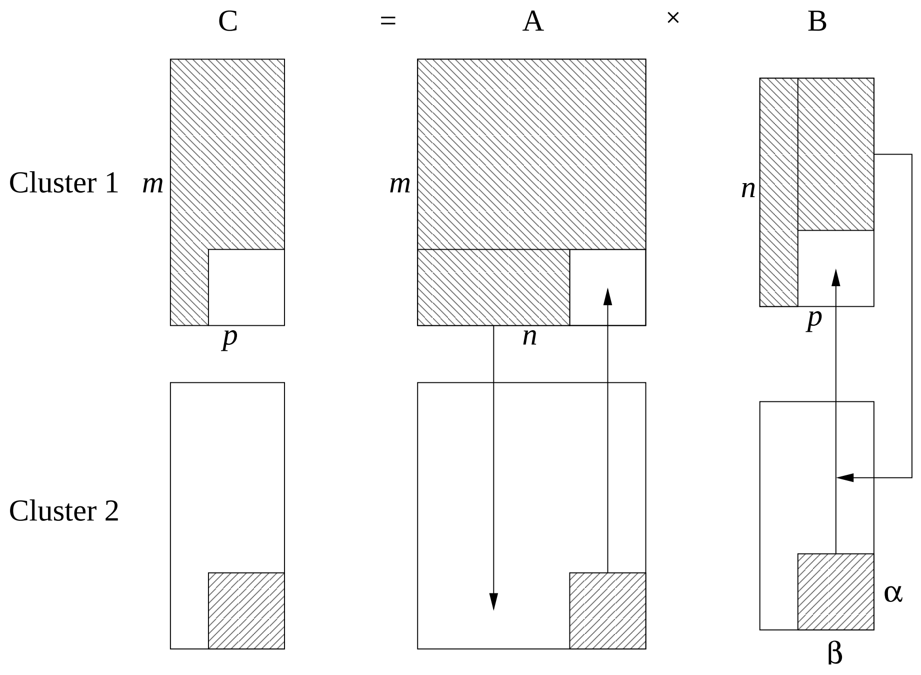


Figure 8.5: The necessary data movements to calculate the rectangular matrix product $C = A \times B$ on two heterogeneous clusters, with one square and one polygonal partition per matrix.

Proposition 8.2: The TVC, $n \times (\alpha + \beta)$, for a multiplication of three rectangular matrices $C = A \times B$ is minimized when $\alpha = \beta$, that is when the partition of area $\alpha \times \beta$ is a square, provided $\alpha = \beta < m, n, p$ where A has dimensions $m \times n$, B has dimensions $n \times p$ and C has dimensions $m \times p$.

Proof: n is a given constant, therefore the goal is to minimize $\alpha + \beta$, where $\alpha \times \beta$ is a constant. Since $\alpha + \beta$ is proportional to the perimeter of the rectangle with area $\alpha \times \beta$, and the perimeter of any rectangle of a constant given area is minimized when that rectangle is a square, we conclude that $\alpha = \beta$.

Q.E.D.

It is interesting to note that the TVC only depends on one matrix dimension, n, which is the only matrix dimension that does not feature in the product matrix C. (n only possibly affects the *values* of the elements of C.) We can conclude that minimizing the SHP on a unit square, then scaling that square to a rectangle does not necessarily minimize the TVC of a matrix matrix multiplication involving that rectangle. This topic area demands further investigation particularly what to do in the case where $\alpha = \beta$ are $>$ m, n, and/or p.

CHAPTER

NINE

CONCLUSIONS AND FUTURE WORK

"They've got a name for the winners in the world, I want a name when I lose"
– Walter Becker and Donald Fagen (1976)[1]

9.1 Conclusions

The current state and foreseeable future of high performance scientific computing can be described in three words: heterogeneous, parallel and distributed. These three simple words have a great impact on the architecture and design of HPC platforms and the creation and execution of algorithms and programs designed to run on them. We have seen that heterogeneity and hierarchy have infiltrated every aspect of computing from supercomputers, GPUs and cloud computing down to individual processors and cores. We have also seen that in many, many ways all of these technologies are interwoven and joined to form hybrid entities themselves. As a result of the inherent heterogeneity, parallelism and distribution which promises to continue to pervade scientific computing in the coming years the issue of data distribution is unavoidable. This, combined with a lack of research into the area of parallel computing on small numbers of (possibly powerful) heterogeneous computing entities provided us with our motivation.

This thesis presented a new top-level data partitioning algorithm, the Square-Corner Partitioning, for matrix and linear algebra operations. This partitioning was designed from the outset to be parallel and heterogeneous, not relying on homogeneous counterparts as a starting point. In practice this partitioning distributes data between a small number of clusters (each of which can have great computational power in themselves) in a hierarchal manner, which allows it

[1]Dan, Steely. (1977). *Deacon Blues*. LP: Aja. MCA Records.

the flexibility to be employed in a great range of problem domains and computational platforms. This partitioning minimizes the total volume of communication between clusters in a manner proven to be optimal for a great range of cluster power ratios, thus minimizing overall execution time. In a hybrid form, working with existing partitionings, the total volume of communication can be minimized regardless of the power ratio that exists between the clusters. It also places no restriction on the algorithms or methods employed on the clusters themselves locally, thus maximizing flexibility.

The Square-Corner Partitioning is shown to have several advantages over more traditional Straight-Line (rectangular) partitionings, not only reducing the total volume of communication in an optimal manner, but allowing the overlapping of communication and computation, and necessitating fewer communication steps.

This partitioning was compared to the state-of-the-art theoretically and experimentally. Both its benefits and deficits were discussed and demonstrated. The Square-Corner Partitioning showed to be beneficial in performing matrix matrix multiplications in several scenarios:

- Two processor MMM
- Small two cluster MMM
- Large two cluster MMM
- Three processor MMM
- Large three cluster MMM

The Square-Corner Partitioning was experimentally shown to be applicable to max-plus algebra operations as well as discrete event simulations.

The Square-Corner Partitioning was also theoretically shown to be applicable to non-square matrix matrix multiplications and minimizing the total volume of communication in this most general realm of matrix computation. Additionally it is shown to be extendable to more than three clusters.

Most heterogeneous algorithms and partitionings are designed by modifying existing homogeneous ones. With this in mind the last goal of this thesis was to demonstrate that non-traditional and perhaps unintuitive algorithms and partitionings *designed with heterogeneity in mind from the start* can result in better, and in cases optimal, algorithms and partitionings for heterogeneous platforms. The importance of this given the current outlook for, and trends in, the future of high performance scientific computing is obvious.

9.2 Future Work

Future work precipitating from the work in this thesis include the following:

- Experiment with the Square-Corner Partitioning on more platforms and architectures, possibly including Grid Ireland.
- Optimize the overlapping of communication and computations while remaining within the bounds of the Square-Corner definition.
- Experiment with the possible benefits of the Square-Corner Partitioning on low-level architectures such as multi-core processors.
- Apply the Square-Corner Partitioning to more complex linear algebra routines and applications with more complex and heavy weight communication schedules and volumes.
- Experiment with the Square-Corner Partitioning on non-square matrix systems and operations.
- Experiment with more than three partitions to determine what configurations can reduce the TVC and under what conditions this occurs.
- Develop fully the Hybrid Square-Corner Partitioning concept.
- Apply the Square-Corner Partitioning to sparse matrix systems.
- Investigate partitionings for matrix matrix multiplication on non-square matrices as discussed in Section 8.3.
- Explore the role of the Square-Corner Partitioning as well as other novel partitioning algorithms in the domain of cloud computing.

APPENDIX

A

HCL CLUSTER PUBLICATIONS

Theses and papers published using the Heterogeneous Computing Laboratory Cluster through April, 2011

Alonso, P., Reddy, R., and Lastovetsky, A. (2010). Experimental study of six different implementations of parallel matrix multiplication on heterogeneous computational clusters of multicore processors. *In 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010)*, pages 263 –270, Pisa, Italy.

Becker, B. and Lastovetsky, A. (2006). Matrix multiplication on two interconnected processors. *In Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain. IEEE Computer Society, IEEE Computer Society.* CD-ROM/Abstracts Proceedings.

Becker, B. and Lastovetsky, A. (2007). Towards data partitioning for parallel computing on three interconnected clusters. *In Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007),* pages 285–292, Hagenberg, Austria. IEEE Computer Society, IEEE Computer Society.

Becker, B. and Lastovetsky, A. (2010). Max-Plus algebra and discrete event simulation on parallel hierarchical heterogeneous platforms. *Euro-Par 2010 / HeteroPar' 2010, Ischia-Naples, Italy. Lecture Notes in Computer Science Proceedings of HeteroPar' 2010*, Springer.

Becker, B. (2011). High-level data partitioning for parallel computing on heterogeneous hierarchical computational platforms. Ph.D. thesis, School of

Computer Science and Informatics, University College Dublin, Dublin.

Brady, T., Guidolin, M., and Lastovetsky, A. (2008). Experiments with SmartGridSolve: achieving higher performance by improving the gridrpc model. *In The 9th IEEE/ACM International Conference on Grid Computing,* Tsukuba, Japan.

Brady, T. (2009). SmartGridRPC: A new RPC model for high performance grid computing and its implementation in SmartGridSolve. Ph.D. thesis, School of Computer Science and Informatics, University College Dublin, Dublin.

Brady, T., Dongarra, J., Guidolin, M., Lastovetsky, A., and Seymour, K. (2010). SmartGridRPC: The new RPC model for high performance grid computing. Concurrency and Computation: Practice and Experience. Wiley.

Clarke, D., Rychkov, V., and Lastovetsky, A. (2010). Dynamic load balancing of parallel computational iterative routines on platforms with memory heterogeneity. *Euro-Par 2010 / HeteroPar' 2010, Ischia-Naples, Italy. Lecture Notes in Computer Science Proceedings of HeteroPar' 2010,* Springer.

Dichev, K., Rychkov, V., and Lastovetsky, A. (2010). Two algorithms of irregular scatter/gather operations for heterogeneous platforms. *In Proceedings of EuroMPI 2010, Stuttgart, Germany* Lecture Notes In Computer Science, V. 6305, pp 289-293, Springer.

Guidolin, M. and Lastovetsky, A. (2009). Grid-enabled hydropad: a scientific application for benchmarking gridrpc-based programming systems. *In The 23rd IEEE International Parallel and Distributed Processing Symposium,* Rome, Italy.

Guidolin, M. (2010). Performance of GridRPC-based programming systems for distributed scientific computing: issues and solutions. Ph.D. thesis, School of Computer Science and Informatics, University College Dublin, Dublin.

Guidolin, M., Brady, T., and Lastovetsky, A. (2010). How algorithm definition language (ADL) improves the performance of smartgridsolve applications. *In The 7th High-Performance Grid Computing Workshop,* Atlanta, USA.

Higgins, R. and Lastovetsky, A. (2009). Managing the construction and use of functional performance models in a grid environment. *In The 23rd IEEE International Parallel and Distributed Processing Symposium,* Rome, Italy.

Higgins, R. (2011). Modelling the performance of processors in heterogeneous computing environments. Ph.D. thesis, School of Computer Science and Informatics, University College Dublin, Dublin Ireland.

Lastovetsky, A., Mkwawa, I., and O'Flynn, M. (2006a). An accurate communication model of a heterogeneous cluster based on a switch-enabled ethernet network. *In Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS 2006),* volume 2, pages 15–20, Minneapolis, Minnesota, USA. IEEE Computer Society Press, IEEE Computer Society Press.

Lastovetsky, A., Mkwawa, I., and O'Flynn, M. (2006b). Modeling performance of many-to-one collective communication operations in heterogeneous clusters. Technical Report UCD-CSI-2006-4

Lastovetsky, A., Reddy, R., and Higgins, R. (2006). Building the functional performance model of a processor. *In Proceedings of the 21st Annual ACM Symposium on Applied Computing (SAC 2006),* Dijon, France. ACM.

Lastovetsky, A. (2007). On grid-based matrix partitioning for heterogeneous processors. *In Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007),* pages 383–390, Hagenberg, Austria. IEEE Computer Society, IEEE Computer Society.

Lastovetsky, A. and O'Flynn, M. (2007). A performance model of many-to-one collective communications for parallel computing. *In Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, California, USA.* IEEE Computer Society, IEEE Computer Society. CD-ROM/Abstracts Proceedings.

Lastovetsky, A. and Rychkov, V. (2007). Building the communication performance model of heterogeneous clusters based on a switched network. *In Proceedings of the 2007 IEEE International Conference on Cluster Computing (Cluster 2007),* pages 568–575, Austin, Texas, USA. IEEE Computer Society, IEEE Computer Society. CD-ROM/Abstracts Proceedings.

Lastovetsky, A., O'Flynn, M., and Rychkov, V. (2007). Optimization of collective communications in HeteroMPI. In F. Capello, T. Herault, and J. Dongarra, editors, *14th European PVM/MPI User's Group Meeting, volume 4757 of Lecture Notes in Computer Science. Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 135–143, Paris, France. Springer-Verlag Berlin Heidelberg, Springer-Verlag Berlin Heidelberg.

Lastovetsky, A. and Reddy, R. (2007a). Data partitioning with a functional performance model of heterogeneous processors. *International Journal of High Performance Computing Applications*, 21, 76–90.

Lastovetsky, A. and Reddy, R. (2007b). A novel algorithm of optimal matrix partitioning for parallel dense factorization on heterogeneous processors. *In Proceedings of the 9th International Conference on Parallel Computing Technologies (PaCT 2007)*, volume 4671 of Lecture Notes in Computer Science, pages 261–275, Pereslavl-Zalessky, Russia. Springer, Springer.

Lastovetsky, A. and Reddy, R. (2007c). Data distribution for dense factorization on computers with memory heterogeneity. *Parallel Computing*, 33, 757–779.

Lastovetsky, A., Rychkov, V., and O'Flynn, M. (2008a). MPIBlib: Benchmarking MPI communications for parallel computing on homogeneous and heterogeneous clusters. *In A. Lastovetsky, T. Kechadi, and J. Dongarra, editors, 15th European PVM/MPI User's Group Meeting, volume 5205 of Lecture Notes in Computer Science. Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 227–238, Dublin. Springer-Verlag Berlin Heidelberg, Springer-Verlag Berlin Heidelberg.

Lastovetsky, A., Rychkov, V., and O'Flynn, M. (2008b). A software tool for accurate estimation of parameters of heterogeneous communication models. *In A. Lastovetsky, T. Kechadi, and J. Dongarra, editors, 15th European PVM/MPI User's Group Meeting, volume 5205 of Lecture Notes in Computer Science. Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 43–54, Dublin. Springer-Verlag Berlin Heidelberg, Springer-Verlag Berlin Heidelberg.

Lastovetsky, A. and Rychkov, V. (2009). Accurate and efficient estimation of parameters of heterogeneous communication performance models. *International Journal of High Performance Computing Applications,* 23, 123–139.

Lastovetsky, A., Rychkov, V., and O'Flynn, M. (2009). Revisiting communication performance models for computational clusters. *In IPDPS 2009,* Rome, Italy.IEEE.

Lastovetsky, A. and Reddy, R. (2010a). Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models. *In 7th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar 2009),* pages 91–101, Delft, Netherlands. Lecture Notes in Computer Science, vol. 6043, Springer, Lecture Notes in Computer Science, vol. 6043, Springer.

Lastovetsky, A. and Reddy, R. (2010b). Two-Dimensional matrix partitioning for parallel computing on heterogeneous processors based on their functional performance models. *In 7th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar 2009),* pages 112–121, Delft, Netherlands. Lecture Notes in Computer Science, vol. 6043, Springer, Lecture Notes in Computer Science, vol. 6043, Springer.

Lastovetsky, A., Rychkov, V., and O'Flynn, M. (2010). Accurate heterogeneous communication models and a software tool for their efficient estimation. *International Journal of High Performance Computing Applications,* 24, 34–48.

O'Flynn, M. (2009). Communication performance models for heterogeneous computational clusters. Ph.D. thesis, School of Computer Science and Informatics, University College Dublin, Dublin.

Reddy, R. and Lastovetsky, A. (2006). HeteroMPI + ScaLAPACK: Towards a ScaLAPACK (dense linear solvers) on heterogeneous networks of computers. *In Proceedings of the 13th IEEE International Conference on High Performance Computing (HiPC 2006), volume 4297 of Lecture Notes in Computer Science,* pages 242–253, Bangalore, India. Springer, Springer.

Reddy, R., Lastovetsky, A., and Alonso, P. (2008a). Heterogeneous PBLAS: a set of parallel basic linear algebra subprograms for heterogeneous computational clusters. Technical Report UCD-CSI-2008-2.

Reddy, R., Lastovetsky, A., and Alonso, P. (2008b). Heterogeneous PBLAS: optimization of PBLAS for heterogeneous computational clusters. *In 7th International Symposium on Parallel and Distributed Computing,* pages 73–80, Krakow, Poland.

Reddy, R., Lastovetsky, A., and Alonso, P. (2008c). Scalable dense factorizations for heterogeneous computational clusters. *In 7th International Symposium on Parallel and Distributed Computing,* pages 49–56, Krakow, Poland.

Reddy, R., Lastovetsky, A., and Alonso, P. (2009a). HeteroPBLAS: A set of parallel basic linear algebra subprograms optimized for heterogeneous computational clusters. Scalable Computing: Practice and Experience, 10, 201–216.

Reddy, R., Lastovetsky, A., and Alonso, P. (2009b). Parallel solvers for dense linear systems for heterogeneous computational clusters. *In The 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009),* Rome, Italy. IEEE Computer Society, IEEE Computer Society.

Valencia, D., Lastovetsky, A., O'Flynn, M., Plaza, A., and Plaza, J. (2008). Parallel processing of remotely sensed hyperspectral images on heterogeneous networks of workstations using HeteroMPI. *International Journal of High Performance Computing Applications,* 22, 386–407.

Zuo, X. and Lastovetsky, A. (2007). Experiments with a software component enabling NetSolve with direct communications in a non-intrusive and incremental way. *In Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007),* Long Beach, California, USA. IEEE Computer Society, IEEE Computer Society. CD-ROM/Abstracts Proceedings.

Zuo, X. (2011). Non-Intrusive and incremental evolution of grid programming systems. Ph.D. thesis, School of Computer Science and Informatics, University College Dublin, Dublin.

APPENDIX

B

HCL CLUSTER SOFTWARE PACKAGES

Software packages developed using the Heterogeneous Computing Laboratory Cluster. Details and latest releases are available at `http://hcl.ucd.ie/`

- ADL: Algorithm Definition Language, a new language and compiler that is designed to improve the performance of GridRPC/SmartGridRPC applications.
- CPM: Communication Performance Models of Heterogeneous Networks of Computers, a software tool that automates the estimation of the heterogeneous communication performance models of clusters based on a switched network.
- HeteroMPI: An extension of MPI for high performance heterogeneous computing.
- HeteroScaLAPACK: A linear algebra library for heterogeneous networks of computers.
- Hydropad: a grid enabled astrophysical application that simulates the evolution of clusters of galaxies in the universe
- MPIBlib: An MPI Benchmark library.
- NI-Connect: Non-intrusive and incremental evolution of grid programming systems.

- SmartGridSolve: High level programming system for high performance grid computing.

APPENDIX

C

HCL CLUSTER PERFORMANCE AND SPECIFICATIONS

Table C.1 shows the performance of each node of the HCL Cluster and the aggregate performance in MFlops. All performance values were experimentally determined. Table C.2 shows full HCL Cluster Specifications as of May 2010.

Node	**Absolute Speed (MFlops)**
hcl01	2171
hcl02	2099
hcl03	1761
hcl04	1787
hcl05	175
hcl06	1653
hcl07	1879
hcl08	1635
hcl09	3004
hcl10	2194
hcl11	4580
hcl12	1762
hcl13	4934
hcl14	4096
hcl15	2697
hcl16	4840
Total Cluster	**42,827**

Table C.1: Performance of each node of the HCL Cluster and the aggregate performance in MFlops. All performance values were experimentally determined.

Name	Make/Model	O/S	IP	Processor	Front Side Bus	L2 Cache	RAM	HDD 1	HDD 2	NIC
hclswitch1	Cisco Catalyst 3560G	12.2(25)SEB2	192.168.21.252	N/A	N/A	N/A	N/A	N/A	N/A	24 x Gigabit
hclswitch2	Cisco Catalyst 3560G	12.2(25)SEB2	192.168.21.253	N/A	N/A	N/A	N/A	N/A	N/A	24 x Gigabit
N/A	APC Smart UPS 1500	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Hcl01 (NIC1)	Dell Poweredge SC1425	FC4	192.168.21.3	3.6 Xeon	800MHz	2MB	256MB	240GB SCSI	80GB SCSI	2 x Gigabit
Hcl01 (NIC2)			192.168.21.103							
Hcl02 (NIC1)	Dell Poweredge SC1425	FC4	192.168.21.4	3.6 Xeon	800MHz	2MB	256MB	240GB SCSI	80GB SCSI	2 x Gigabit
Hcl02 (NIC2)			192.168.21.104							
Hcl03 (NIC1)	Dell Poweredge 750	FC4	192.168.21.5	3.4 Xeon	800MHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl03 (NIC2)			192.168.21.105							
Hcl04 (NIC1)	Dell Poweredge 750	FC4	192.168.21.6	3.4 Xeon	800MHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl04 (NIC2)			192.168.21.106							
Hcl05 (NIC1)	Dell Poweredge 750	FC4	192.168.21.7	3.4 Xeon	800MHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl05 (NIC2)			192.168.21.107							
Hcl06 (NIC1)	Dell Poweredge 750	FC4	192.168.21.8	3.4 Xeon	800MHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl06 (NIC2)			192.168.21.108							
Hcl07 (NIC1)	Dell Poweredge 750	FC4	192.168.21.9	3.4 Xeon	800MHz	1MB	256MB	80GB SATA	N/A	2 x Gigabit
Hcl07 (NIC2)			192.168.21.109							
Hcl08 (NIC1)	Dell Poweredge 750	FC4	192.168.21.10	3.4 Xeon	800MHz	1MB	256MB	80GB SATA	N/A	2 x Gigabit
Hcl08 (NIC2)			192.168.21.110							
Hcl09 (NIC1)	IBM E-server 326	Debian	192.168.21.11	1.8 AMD Opteron	1GHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl09 (NIC2)			192.168.21.111							
Hcl10 (NIC1)	IBM E-server 326	FC4	192.168.21.12	1.8 AMD Opteron	1GHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl10 (NIC2)			192.168.21.112							
Hcl11 (NIC1)	IBM X-Series 306	Debian	192.168.21.13	3.2 P4	800MHz	1MB	512MB	80GB SATA	N/A	2 x Gigabit
Hcl11 (NIC2)			192.168.21.113							
Hcl12 (NIC1)	HP Proliant DL 320 G3	FC4	192.168.21.14	3.4 P4	800MHz	1MB	512MB	80GB SATA	N/A	2 x Gigabit
Hcl12 (NIC2)			192.168.21.114							
Hcl13 (NIC1)	HP Proliant DL 320 G3	FC4	192.168.21.15	2.9 Celeron	533MHz	256KB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl13 (NIC2)			192.168.21.115							
Hcl14 (NIC1)	HP Proliant DL 140 G2	Debian	192.168.21.16	3.4 Xeon	800MHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl14 (NIC2)			192.168.21.116							
Hcl15 (NIC1)	HP Proliant DL 140 G2	Debian	192.168.21.17	2.8 Xeon	800MHz	1MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl15 (NIC2)			192.168.21.117							
Hcl16 (NIC1)	HP Proliant DL 140 G2	Debian	192.168.21.18	3.6 Xeon	800MHz	2MB	1GB	80GB SATA	N/A	2 x Gigabit
Hcl16 (NIC2)			192.168.21.118							

Table C.2: Hardware and Operating System specifications for the HCL Cluster. As of May 2010 and the upgrade to HCL Cluster 2.0, all nodes are operating Debian "squeeze" kernel 2.6.32.

APPENDIX

D

HCL CLUSTER STATISTICS APRIL 2010 - MARCH 2011

For the period April 2010 - March 2011, Figure D.1 Shows an overall cluster profile, D.2 shows a node by node load report, D.3 shows a node by node CPU report, D.4 shows a node by node memory report, and D.5 shows a node by node network report.

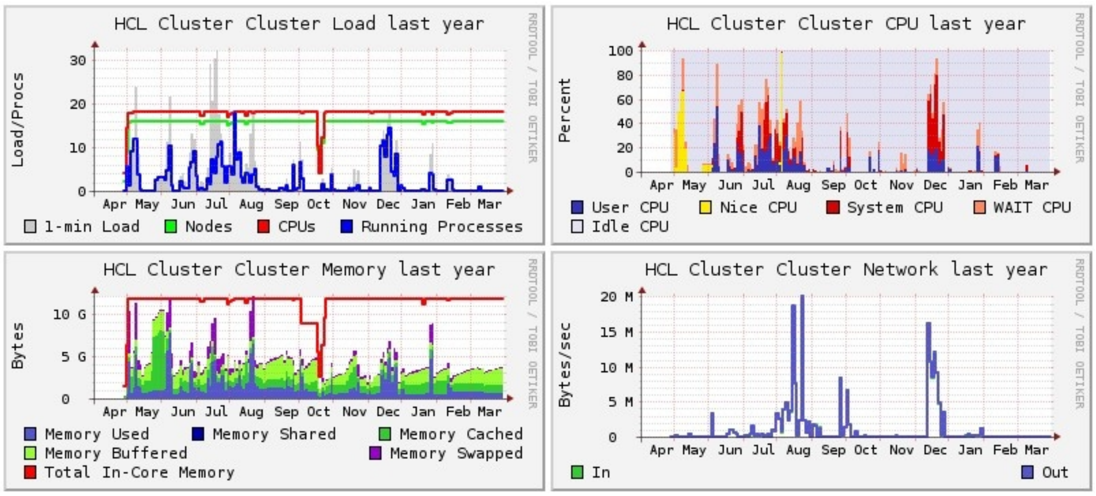


Figure D.1: HCL Cluster load, CPU, memory and network profiles for the year April 2010 - March 2011. Provided by Ganglia.

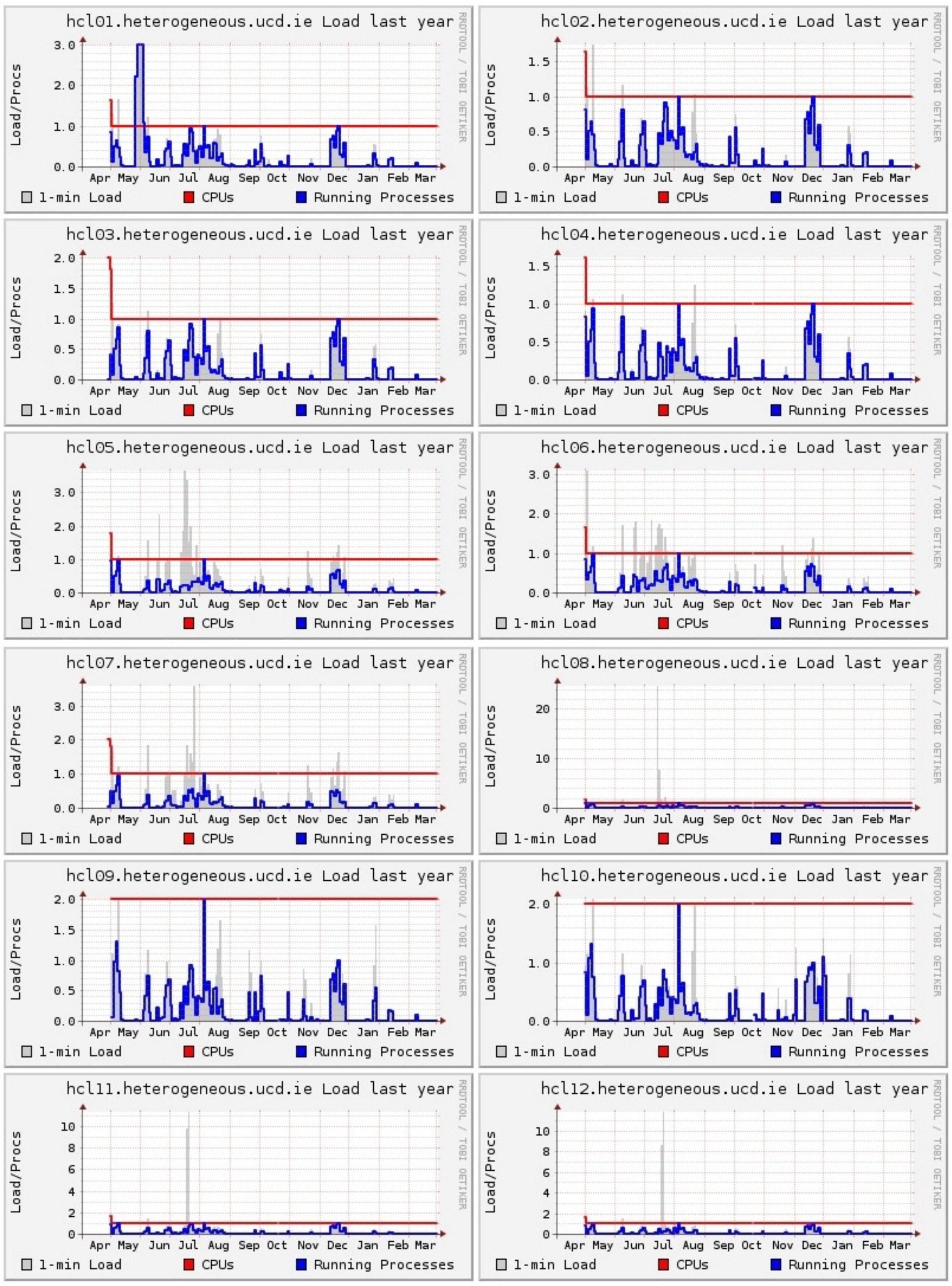


Figure continued on next page.

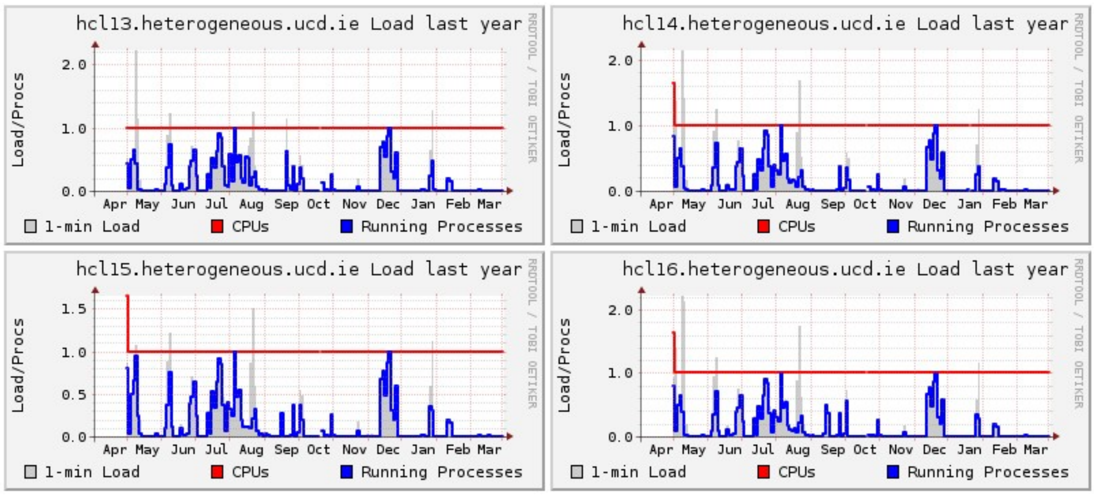


Figure D.2: HCL Cluster Load Report April 2010 - March 2011. Note that the load on hcl08, hcl11 and hcl12 is not lower than the other nodes. The scale of the y-axes of these graphs are affected by a very high 1 minute load in July. Provided by Ganglia.

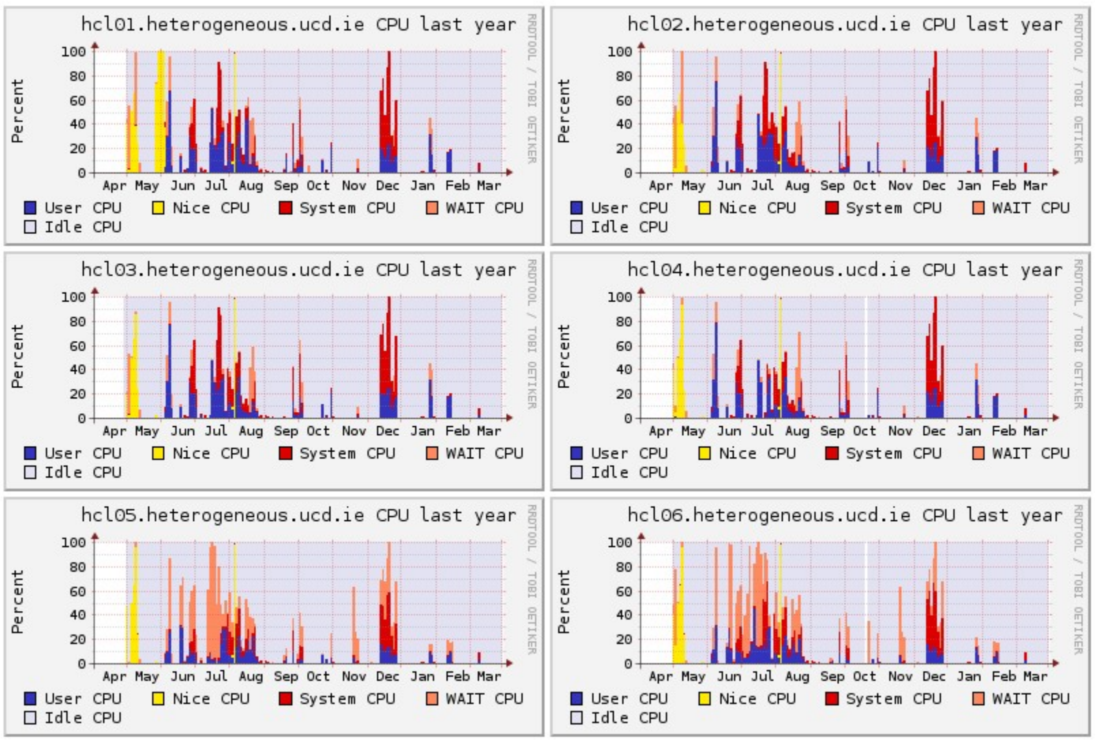


Figure continued on next page.

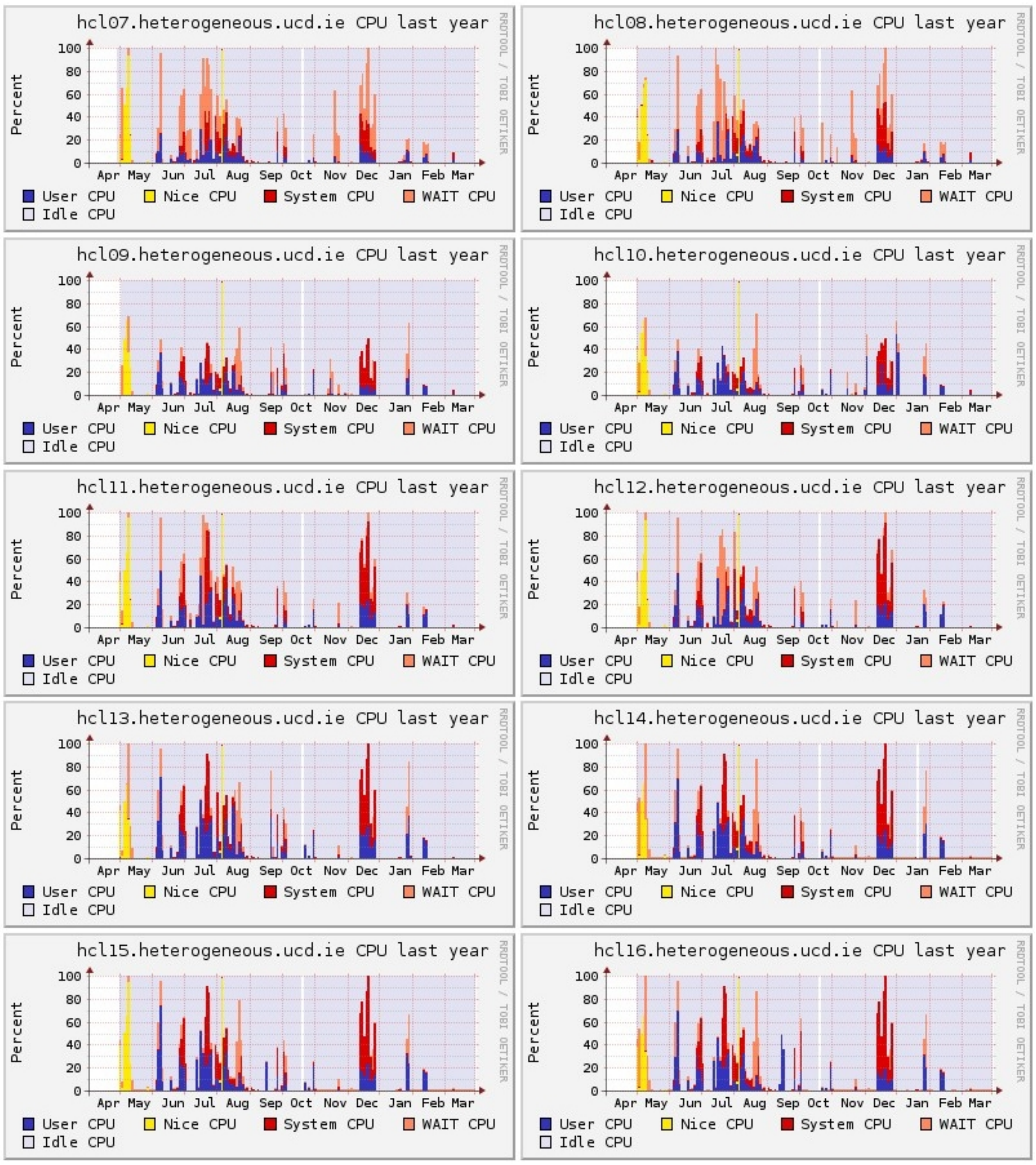


Figure D.3: HCL Cluster CPU Report April 2010 - March 2011. Provided by Ganglia.

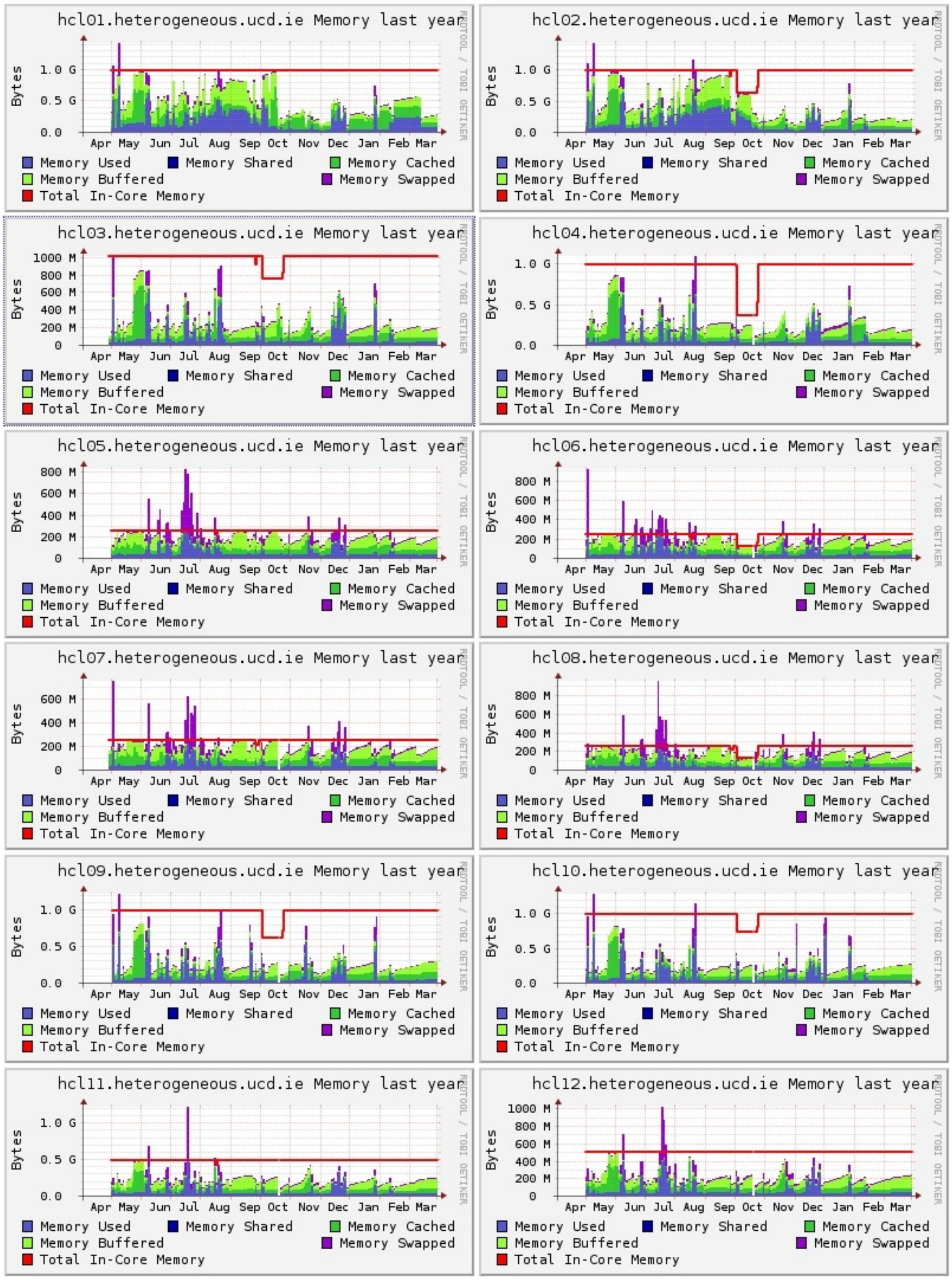


Figure continued on next page.

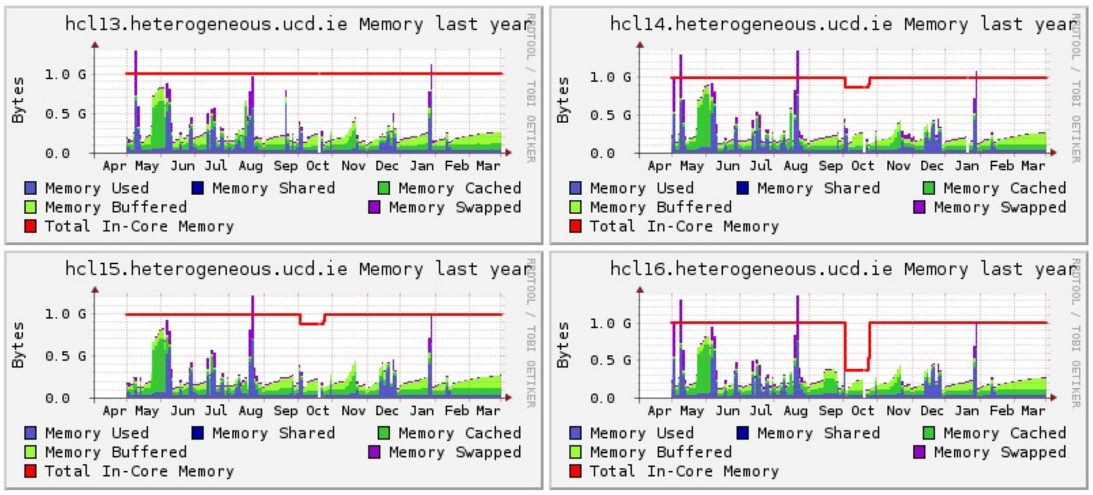


Figure D.4: HCL Cluster Memory Report April 2010 - March 2011. Provided by Ganglia.

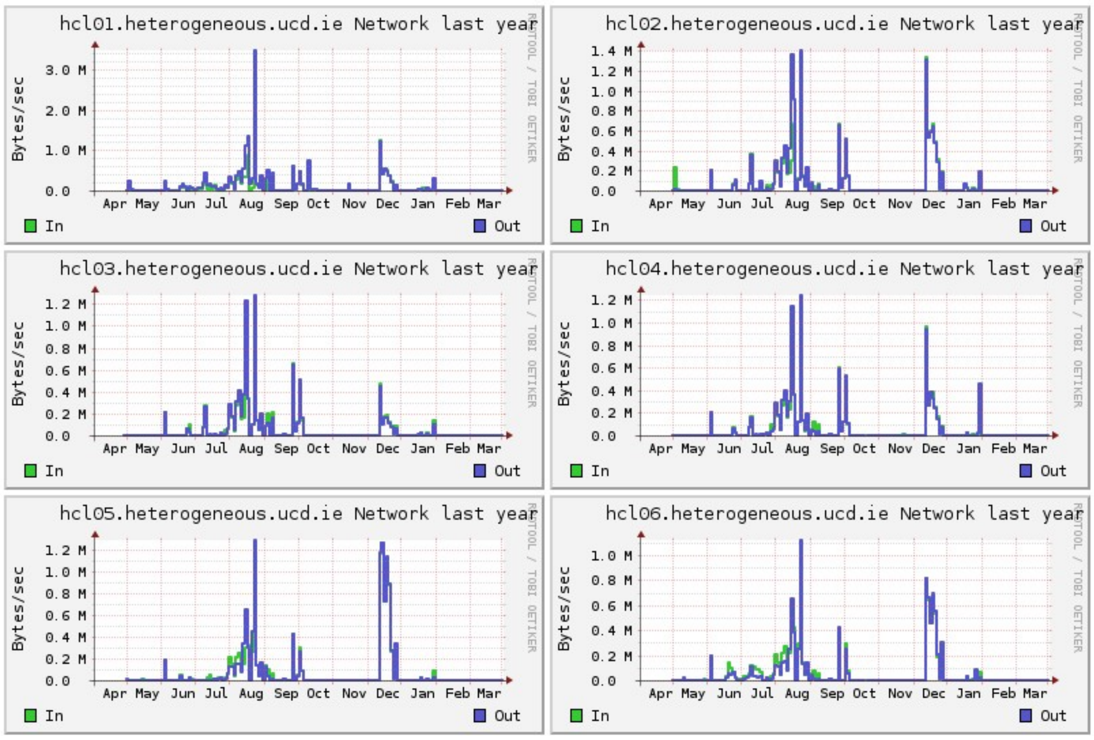


Figure continued on next page.

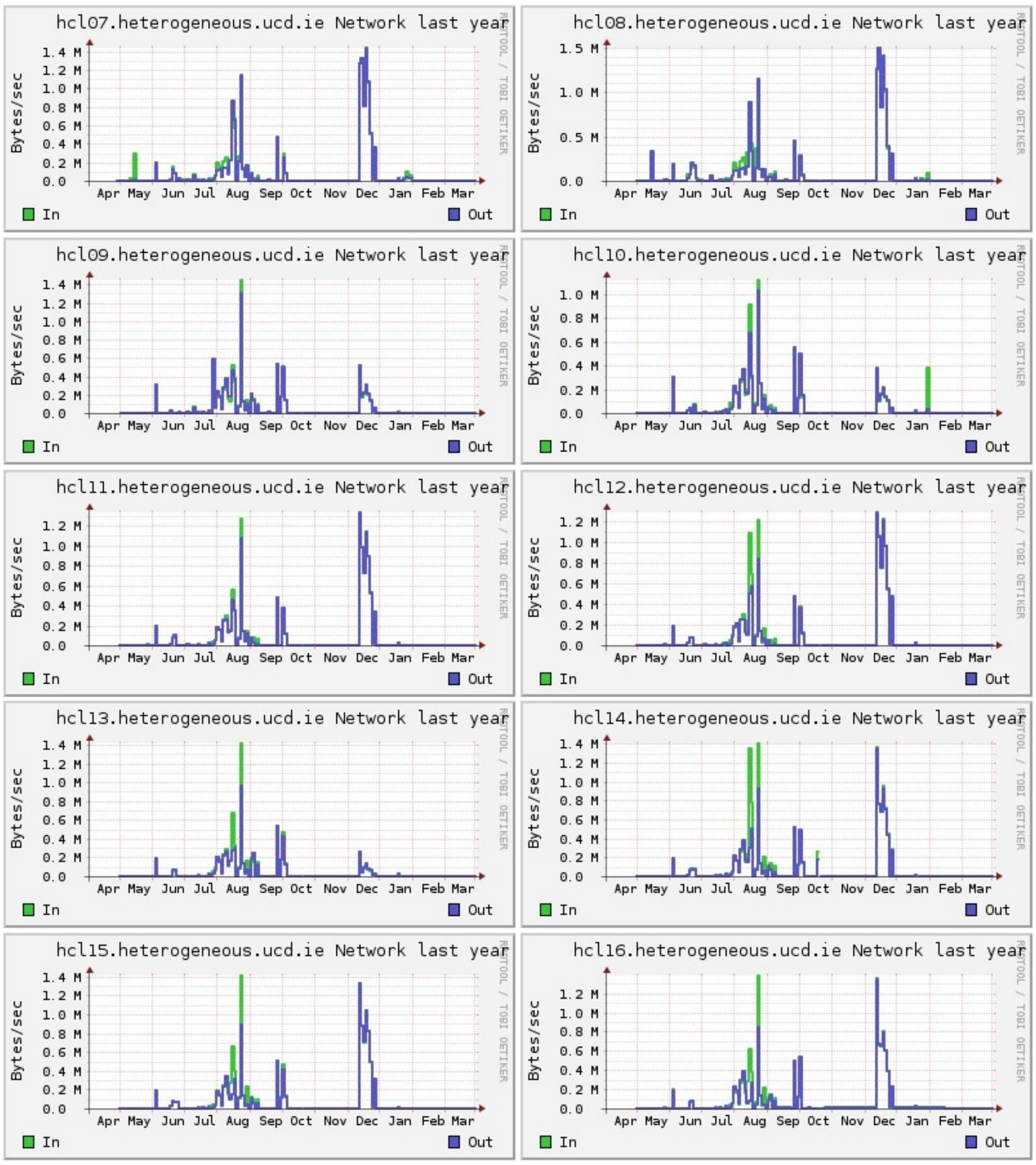


Figure D.5: HCL Cluster Network Report April 2010 - March 2011. Provided by Ganglia.

APPENDIX

E

HCL CLUSTER STREAM BENCHMARK

Results of a STREAM benchmark of the HCL Cluster August, 2010

```
Cluster STREAM Performance
——————————————————-
Double precision appears to have 16 digits of accuracy
Assuming 8 bytes per DOUBLE PRECISION word
——————————————————-
Number of processors = 18
Array size = 2000000
Offset = 0
The total memory requirement is 824.0 MB ( 45.8MB/task)
You are running each test 10 times
–
The *best* time for each test is used
*EXCLUDING* the first and last iterations
————————————————————–
Your clock granularity appears to be less than one microsecond
Your clock granularity/precision appears to be 1 microseconds
————————————————————–
```

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	24589.1750	0.0235	0.0234	0.0237
Scale:	24493.9786	0.0237	0.0235	0.0245
Add:	27594.1797	0.0314	0.0313	0.0315
Triad:	27695.7938	0.0313	0.0312	0.0315

Solution Validates!

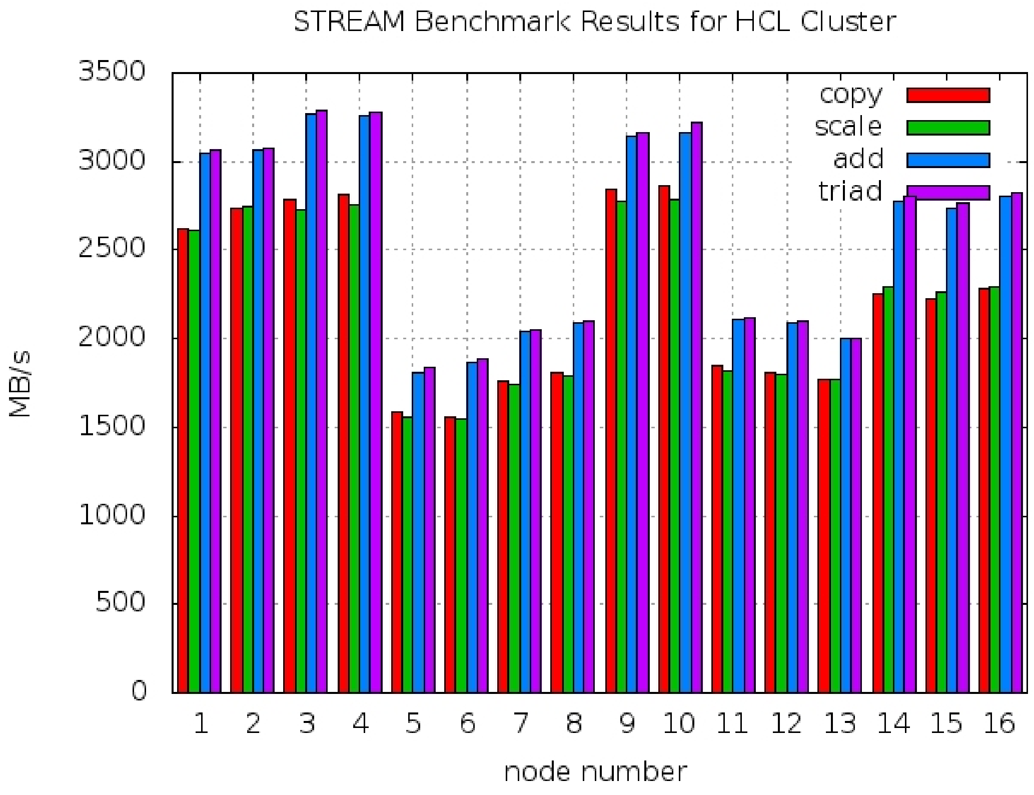


Figure E.1: STREAM Benchmark Results for HCL Cluster, August 2010.

APPENDIX

F

HCL CLUSTER HPL BENCHMARK

Results of a High Performance Linpack benchmark of the HCL Cluster August, 2010

```
$ mpirun -np 18 ./xhpl
================================================================
HPLinpack 2.0 – High-Performance Linpack benchmark – September 10, 2008
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory,
UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================
An explanation of the input/output parameters follows:
T/V : Wall time / encoded variant.
N : The order of the coefficient matrix A.
NB : The partitioning blocking factor.
P : The number of process rows.
Q : The number of process columns.
Time : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.
The following parameter values will be used:

N : 18432
NB : 64 96 128 144 160 172 192 224 256
PMAP : Row-major process mapping
P : 6
Q : 3
```

```
PFACT : Crout
NBMIN : 4
NDIV : 2
RFACT : Right
BCAST : 1ringM 2ringM
DEPTH : 0
SWAP : Mix (threshold = 64)
L1 : transposed form
U : transposed form
EQUIL : yes
ALIGN : 8 double precision words
--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
||Ax − b||_oo/(eps ∗ (||x||_oo ∗ ||A||_oo + ||b||_oo) ∗ N)
- The relative machine precision (eps) is taken to be 1.110223e-16
- Computational tests pass if scaled residuals are less than 16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2C4       18432    64     6     3             412.68              1.012e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0035149 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR03R2C4       18432    64     6     3             460.70              9.063e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0035149 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2C4       18432    64     6     3             243.45              1.715e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0039257 ...... PASSED
================================================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR03R2C4   18432    64   6   3  317.76       1.314e+01
------------------------------------------------------
```

$||Ax-b||_o o/(eps*(||A||_o o*||x||_o o+||b||_o o)*N) = 0.0039257$ ...... PASSED

```
============================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR01R2C4   18432    64   6   3  268.04       1.558e+01
------------------------------------------------------
```

$||Ax-b||_o o/(eps*(||A||_o o*||x||_o o+||b||_o o)*N) = 0.0039261$ ...... PASSED

```
============================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR03R2C4   18432    64   6   3  207.83       2.009e+01
------------------------------------------------------
```

$||Ax-b||_o o/(eps*(||A||_o o*||x||_o o+||b||_o o)*N) = 0.0039261$ ...... PASSED

```
============================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR01R2C4   18432    64   6   3  176.45       2.493e+01
------------------------------------------------------
```

$||Ax-b||_o o/(eps*(||A||_o o*||x||_o o+||b||_o o)*N) = 0.0040578$ ...... PASSED

```
============================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR03R2C4   18432    64   6   3  221.02       1.889e+01
------------------------------------------------------
```

$||Ax-b||_o o/(eps*(||A||_o o*||x||_o o+||b||_o o)*N) = 0.0040578$ ...... PASSED

```
============================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR01R2C4   18432    64   6   3  176.85       2.361e+01
------------------------------------------------------
```

$||Ax-b||_o o/(eps*(||A||_o o*||x||_o o+||b||_o o)*N) = 0.0039335$ ...... PASSED

```
============================================================
T/V            N    NB   P   Q    Time          Gflops
------------------------------------------------------
WR03R2C4   18432    64   6   3  322.95       1.293e+01
------------------------------------------------------
```

```
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0039335 ...... PASSED
================================================================================
          T/V          N    NB    P    Q        Time                 Gflops
--------------------------------------------------------------------------------
WR01R2C4    18432      64    6    3    339.39              1.230e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 4 0.0036399 ...... PASSED
================================================================================
          T/V          N    NB    P    Q        Time                 Gflops
--------------------------------------------------------------------------------
WR03R2C4    18432      64    6    3    254.28              1.642e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0036399 ...... PASSED
================================================================================
          T/V          N    NB    P    Q        Time                 Gflops
--------------------------------------------------------------------------------
WR01R2C4    18432      64    6    3    315.42              1.324e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0040343 ...... PASSED
================================================================================
          T/V          N    NB    P    Q        Time                 Gflops
--------------------------------------------------------------------------------
WR03R2C4    18432      64    6    3    220.11              2.066e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0040343 ...... PASSED
================================================================================
          T/V          N    NB    P    Q        Time                 Gflops
--------------------------------------------------------------------------------
WR01R2C4    18432      64    6    3    289.17              1.443e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0038685 ...... PASSED
================================================================================
          T/V          N    NB    P    Q        Time                 Gflops
--------------------------------------------------------------------------------
WR03R2C4    18432      64    6    3    177.57              2.351e+01
--------------------------------------------------------------------------------
||Ax − b||_oo/(eps ∗ (||A||_oo ∗ ||x||_oo + ||b||_oo) ∗ N) = 0.0038685 ...... PASSED
================================================================================
      T/V          N    NB   P   Q   Time              Gflops
--------------------------------------------------------------
WR01R2C4   18432    64   6    3   23.24             1.716e+01
--------------------------------------------------------------
```

$||Ax - b||_o o/(eps * (||A||_o o * ||x||_o o + ||b||_o o) * N) = 0.0035689$ ...... PASSED
==

```
      T/V          N    NB   P   Q    Time             Gflops
--------------------------------------------------------------
WR03R2C4   18432    64   6    3   188.46            2.215e+01
--------------------------------------------------------------
```

$||Ax - b||_o o/(eps * (||A||_o o * ||x||_o o + ||b||_o o) * N) = 0.0035689$ ...... PASSED
==

Finished 18 tests with the following results: 18 tests completed and passed residual checks, 0 tests completed and failed residual checks, 0 tests skipped because of illegal input values.

End of Tests.
==

BIBLIOGRAPHY

Beaumont, O., Boudet, V., Petitet, A., Rastello, F., and Robert, Y. (2001a). A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Transactions on Computers*, **50**(10), 1052–1070.

Beaumont, O., Boudet, V., Rastello, F., and Robert, Y. (2001b). Matrix-Matrix multiplication on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems*, **12**(10), 1033–1051.

Beaumont, O., Boudet, V., Legrand, A., Rastello, F., and Robert, Y. (2002a). Heterogeneous matrix-matrix multiplication or partitioning a square into rectangles: NP-Completeness and approximation algorithms. In *Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing, 2001*, pages 298–305. IEEE.

Beaumont, O., Boudet, V., Rastello, F., and Robert, Y. (2002b). Partitioning a square into rectangles: NP-Completeness and approximation algorithms. *Algorithmica*, **34**(3), 217–239.

Becker, B. and Lastovetsky, A. (2006). Matrix multiplication on two interconnected processors. In *Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006)*, Barcelona, Spain. IEEE Computer Society.

Becker, B. and Lastovetsky, A. (2007). Towards data partitioning for parallel computing on three interconnected clusters. In *Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007)*, Hagenberg, Austria. IEEE Computer Society.

Becker, B. and Lastovetsky, A. (2010). Max-Plus algebra and discrete event simulation on parallel hierarchical heterogeneous platforms. *Springer Lecture Notes in Computer Science, Euro-Par Workshops Proceedings 2010, as Part of Hetero-Par 2010*.

Blackford, L., Cleary, A., Choi, J., d'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., *et al.* (1997). *ScaLAPACK users' guide*. Society for Industrial Mathematics.

Blackford, L., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., *et al.* (2002). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software (TOMS)*, **28**(2), 135–151.

Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., *et al.* (2006). Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, **20**(4), 481.

Boulet, P., Dongarra, J., Rastello, F., Robert, Y., and Vivien, F. (1999). Algorithmic issues on heterogeneous computing platforms. *Parallel processing letters*, **9**(2), 197–213.

Brady, T., Dongarra, J., Guidolin, M., Lastovetsky, A., and Seymour, K. (2010). SmartGridRPC: The new RPC model for high performance grid computing. *Concurrency and Computation: Practice and Experience*.

Canon, L. and Jeannot, E. (2006). Wrekavoc: a tool for emulating heterogeneity. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 11. IEEE.

Choi, J., Dongarra, J., Ostrouchov, S., Petitet, A., Walker, D., and Whaley, R. (1996). A proposal for a set of parallel basic linear algebra subprograms. *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science*, pages 107–114.

Comet, J. (2003). Application of max-plus algebra to biological sequence comparisons. *Theoretical Computer Science*, **293**(1), 189–217.

Csikor, F., Fodor, Z., Hegedus, P., Katz, S., Piroth, A., and Horvath, V. (2000). The poor man's supercomputer. In *Distributed and parallel systems*, pages 151–154. Kluwer Academic Publishers.

De Schutter, B. and De Moor, B. (1999). On the sequence of consecutive powers of a matrix in a Boolean algebra. *SIAM Journal on Matrix Analysis and Applications*, **21**(1).

Dongarra, J. and Lastovetsky, A. (2006). An overview of heterogeneous high performance and grid computing. *Engineering the Grid: Status and Perspective*, pages 1–25.

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

Ferscha, A. and Tripathi, S. (1996). Parallel and distributed simulation of discrete event systems. *Parallel and Distributed Computing Handbook*, pages 1003–1041.

Fishman, G. (2001). *Discrete-event simulation: modeling, programming, and analysis*. Springer Verlag.

Foster, I. and Kesselman, C. (2004). *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann.

Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. (1988). *Solving problems on concurrent processors*. Prentice Hall Inc.

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation mpi implementation. *Proceedings of the 11th European PVM/MPI Users' Group Meeting*.

Gaubert, S. and Plus, M. (1997). Methods and applications of (max,+) linear algebra. In *STACS 97*, pages 261–282. Springer.

Golub, G. and Van Loan, C. (1996). *Matrix computations*. Johns Hopkins University Press.

Heidergott, B., Olsder, G., and van der Woude, J. (2006). *Max Plus at work: modeling and analysis of synchronized systems: a course on max-plus algebra and its applications*. Princeton University Press.

Johnson, M. and Kambites, M. (2009). Multiplicative structure of 2x2 tropical matrices. *Arxiv preprint arXiv:0907.0314*.

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

Kalinov, A. and Lastovetsky, A. (2001). Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. *Journal of Parallel and Distributed Computing*, **61**, 520–535.

Kirov, M. (2009). The transfer-matrix and max-plus algebra method for global combinatorial optimization: application to cyclic and polyhedral water clusters. *Physica A: Statistical Mechanics and its Applications*, **388**(8), 1431–1445.

Kruger and Westerman (2005). Linear algebra operators for GPU implementatin of numerical algorithms. *International Conference on Computer Graphics and Interactive Techniques*.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to parallel computing: design and analysis of algorithms*. The Benjamin/Cummings.

Lastovetsky, A. (2003). *Parallel computing on heterogeneous networks*. Wiley-IEEE.

Lastovetsky, A. (2007). On grid-based matrix partitioning for heterogeneous processors. In *Proceedings of the Sixth International Symposium on Parallel and Distributed Computing*, page 51. IEEE Computer Society.

Lastovetsky, A. and Dongarra, J. (2009). *High performance heterogeneous computing*. Wiley-Interscience.

Lastovetsky, A. and Reddy, R. (2004). On performance analysis of heterogeneous parallel algorithms. *Parallel Computing*, **30**, 1195–1216.

Lee, H.-J., Robertson, J. P., and Fortes, J. A. B. (1997). Generalized Cannon's algorithm for parallel matrix multiplication. In *ICS '97: Proceedings of the 11th international conference on Supercomputing*, pages 44–51, New York, NY, USA. ACM.

Lin, C. and Snyder, L. (1992). A matrix product algorithm and its comparative performance on hypercubes. In *Scalable High Performance Computing Conference, 1992. SHPCC-92. Proceedings.*, pages 190–194.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25.

Tacconi, D. and Lewis, F. (1997). A new matrix model for discrete event systems: application to simulation. *IEEE Control Systems Magazine*, **17**(5), 62–71.

van Berkel, C. (2009). Multi-core for mobile phones. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 1260–1265. IEEE.

van de Geijn, R. and Watts, J. (1997). SUMMA: scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, **9**(4), 255–274.

Whaley, R. C. and Dongarra, J. (1999). Automatically tuned linear algebra software. *Ninth SIAM Conference on Parallel Processing for Scientific Computing*.

finis...