PARALLEL PROGRAMMING WITH DISTRIBUTED SHARED MEMORY

By

Lee Pau Hua

Dissertation submitted to
UNIVERSITI SAINS MALAYSIA

As a fullfillment of a part of the requirement for
a degree with honours

BACHELOR OF ENGINEERING (ELECTRONIC)

Pusat Pengajian Kejuruteraan
Elektrik dan Elektronik
Universiti Sains Malaysia

May 2006

ABSTRACT

There is a continual demand for greater computational speed from a computer system than is currently possible because it seems that there will always be applications that require still more computation power. One way of increasing the computation speed is by using multiple computers that linked through network operating together on a single problem that split into parts, and each performed by a separate computer in parallel. Such application can gain advantageous on performance, scalability, and even price. In this reseach, a parallel matrix multiplicaton system will be designed using Parallel Programming with Distributed Shared Memory. With the message-passing method, this system will perform the matrix multiplication computation in parallel by using multiple computers that linked through network. The computation time for the parallel matrix multiplication will be compared with the computation time for serial matrix multiplication and will expecting to have a faster computation speed from the parallel comutation method on the larger matrix scale. In overall, the computation cabability of the conventional computer can be improved by the application of parallel computer system.

ABSTRAK

Dalam dunia komputer, terdapatnya suatu permintaan yang berterusan untuk mengejar kelajuan pengiraan yang lebih pantas daripada sistem komputer masa kini disebabkan selalunya akan terwujud aplikasi yang masih memerlukan lebih kuasa pengiraan. Salah satu cara untuk meningkatkan kelajuan pengiraan ialah dengan menggunakan komputer-komputer yang disambungkan melalui rangkaian untuk beroperasi bersama dalam menyelesaikan satu masalah tunggal yang dipecahkan kepada beberapa bahagian, dibahagikan kepada setiap komputer untuk membuat pengiraan secara selari. Aplikasi seperti ini mempunyai kelebihan dalam keberkesanan, kestabilan dan juga harga. Dalam penyelidikan ini, satu sistem pendaraban matriks selari akan direkabentuk menggunakan pengaturcara selari dengan penyimpanan berkongsi secara terbahagi (distributed shared memory). Dengan menggunakan kaedah "message-passing", sistem ini akan mebuat pengiraan pendaraban matriks secara selari menggunakan komputer-komputer yang disambung melalui rangkaian. Masa pengiraan untuk pendaraban matriks selari akan dibandingkan dengan masa pengiraan pendaraban matriks selari dan dijangkakan masa pengiraan pendaraban matriks selari adalah lebih laju untuk skala matriks yang lebih besar. Secara keseluruhannya, kebolehan pengiraan komputer biasa akan dapat dipertingkatkan lagi dengan aplikasi sistem komputer selari.

CONTENTS

FIGURE LIST

ACKNOWLEDGEMENT

I would like to take this opportunity to thank the many people that have helped me tremendously.

First and foremost to Dr. Kamal Zuhairi b. Zamli, the person who guided me all the way. As my supervisor he had patiently taught and advised me and put me in the right path of enlightenment. Without him, this project would be a failure.

Secondly, my gratitude to my friends and coursemates; Yoke Leen, Li Feong, Han Choong, Gan and Kim Aun that had given me ideas and helped me to complete this project, thanks to you all.

Lastly, to my beloved parent and my sisters that always giving me the support. God bless you forever.

Lee Pau Hua

Chapter 1

INTRODUCTION

1.1. Introduction

Conventional computer aren't fast enough to do the work we need nowaday. Even supercomputers may not even be fast enough, and the price puts them out of range in any case. That's why parallel computer is important for solving computationally intensive problems. Parallel computer are likely to be around as applications for us become more widespread. It gives us the ability to ask questions and examine answers more quickly, in greater detail, and more cost effectively.

For multi-year research problems involving extremely large calculations, parallel supercomputers win on performance, scalability, and even price. Nevertheless, for a small scale research with limited budget, multiple conventional computers can be linked through network to form a system likely the supercomputer in order to solve the computation problem in the lab.

There will be a more detail explanation about the use of parallel computation application in Chapter 2, also the explanation about the topic of my final year project which is Parallel Programming with Distributed Shared Memory.

1.2. Purpose of this Project

The main purpose of this project is to design a system that uses the concepts of parallel computation to compute the matrix multiplication problem which needs complex iteration computation.

The design will be concentrate on the difference of computation speed between the parallel computation and serial computation techniques, that is to get a faster computation speed by using parallel computation techniques.

1.3. Aim of This Project

- The aim of this project is to study the concept and usage of parallel programming and distributed shared memory.

1.4. Objectives of This Project

The objectives of this project is as below:

- To learn the core issues of parallel computation application.
- To learn the use of distributed shared memory in parallel programming.
- To learn the developing of object oriented software.
- To learn the process of designing an parallel programming system.
- To learn Java technology and Java programming language.

1.5. Tools for The Design of Parallel Programming

The tools that being use are:,

- Java 2 Standard Editon (J2SE1.5.0_04)
- Bluej v2.05 (object oriented Java Editor tools)
- Borland C++ Builder 6

1.6. Steps for Preparing and Finishing This Project

Before designing and implementing the parallel programming, studies and understanding of the concepts and requirements are necessary for developing the software system. There are a lot of information needs to be studies for getting start of this final year project. Later on, the object analysis and design of the software system is started. The design, implementation, testing and integration of unit system will be done after that. And finally, implements all the unit systems and doing an overall system testing and experimentations.

Meanwhile can start to prepare and do the documentation (thesis). The thesis can be completed after all the designs, implementation, solutions and results are collected.

The methods of finishing this final year project and thesis will have further discuss in chapter 3.

1.7. Structure of Thesis

Chapter 1 of this thesis discusses about the short explanation, purpose, aim and objectives of final year project topic.

Chapter 2 of this thesis provides all the theories referred to implement the parallel programming system with distributed shared memory. There is also an introduction and explanation of the Java tools (J2SE) which be used.

Chapter 3 of this thesis discuss about the methods to be used and also the flow of finishing this final year project and thesis.

Chapter 4 of this thesis discusses about the implementation of the parallel programming system with distributed shared memory. The concept, software design architecture, implementation, results and discussion are all included in this chapter.

Chapter 5 of this thesis is the conclusion of this final year project topic and the thesis.

Chapter 2

THEORY

2.1 Introduction

This project is about the design of Parallel Programming with Distributed Shared Memory system. It implements the parallel computaion techniques with the use of distributed shared memory.

In this chapter, we will discuss about all the theories referred to implement the parallel programming system with distributed shared memory system, including the explanation of the parallel programming, distributed shared memory, and the tools that used to implement this system.

2.2 Parallel Programming

Parallel programming uses multiple computers, or computers with multiple internl processor, to solve a problem at a greater computational speed than using a single computer. It also offers the opportunity to tackle larger problem that with more computational steps or more memory requirements, the latter because multiple computer and multiprocessor systems oftn have more total memory than a single computer.

One of the parallel computing technique is by using multiple computing process operating together on a single problem. The overall problem is split into parts, each of which is performed by a separate process in parallel. Writing programs for this form of computation is known as parallel programming.

The computing platform, a parallel computer, could be a specially designed computer system containing multiple processors or several independent computers interconnected in some way. The idea is that *n* computer could provide up to *n* times the computational speed of a single computer, with the expectation that the problem would be completed in 1/*n*th of the time.

The use of multiple computer/processor often allows a larger or a more precise solution of a problem to be solved in a reasonable amount of time. A related factor is that multiple computers often have more total main memory than

a single computer, enabling problems that require larger amount of main memory to be tackled.

Here we concentrate upon the use of multiple computers that communicate between themselves by sending messages; hence the term message-passing parallel programming. The computer we use can be different platforms (PC, UNIX, SUN, etc.) but must be interconnected by a network, and a software environment must be present for intercomputer message passing. Suitable networked computers are very widely available as the basic computing platform for students so that acquisition of specially designed multiprocessor systems can usually be avoided [1].

2.3 Explanation of "Distributed Shared Memory"

2.3.1 Distributed Shared Memory

Parallel programming requires a suitable computing platform, which have described as either a single computer with multiple internal processors or multiple interconnected computers. Examples, shared memory multiprocessor system, message-passing multicomputer system, distributed shared memory (DSM) system and multiple instruction stream-multiple data stream-multiple data stream (MIMD) computer system.

In a Distributed Shared Memory system the memory is physically distributed with each processor, but each processor has access to the whole memory using single memory address space. For a processor to access a location not in its local memory, message passing must occur to pass data from the processor to the location or from the location to the processor, in some automated way that hides the fact that the memory is distributed. Of course, access to remote locations will incur a greater delay, and usually a significantly greater delay, than for local accesses.

Multiprocessor system can be designed in which the memory is physically distributed but operates as shared memory and appears from the programmer's perspective as shared memory. Perhaps the most appealing approach is to use networked computers. Distributed Shared Memory systems can be implementing

the shared-memory abstraction on multi-computer architectures, combining the scalability of network based architectures with the convenience of shared-memory programming. One way to achieve distributed shared memory on a group of networked computers is to use the existing virtual memory management system of the individual computers which is already provided on almost all systems to manage its local memory hieracchy. The virtual memory management system can be extended to gives the illusion of global shared memory even when it is distributed in different computers [2].

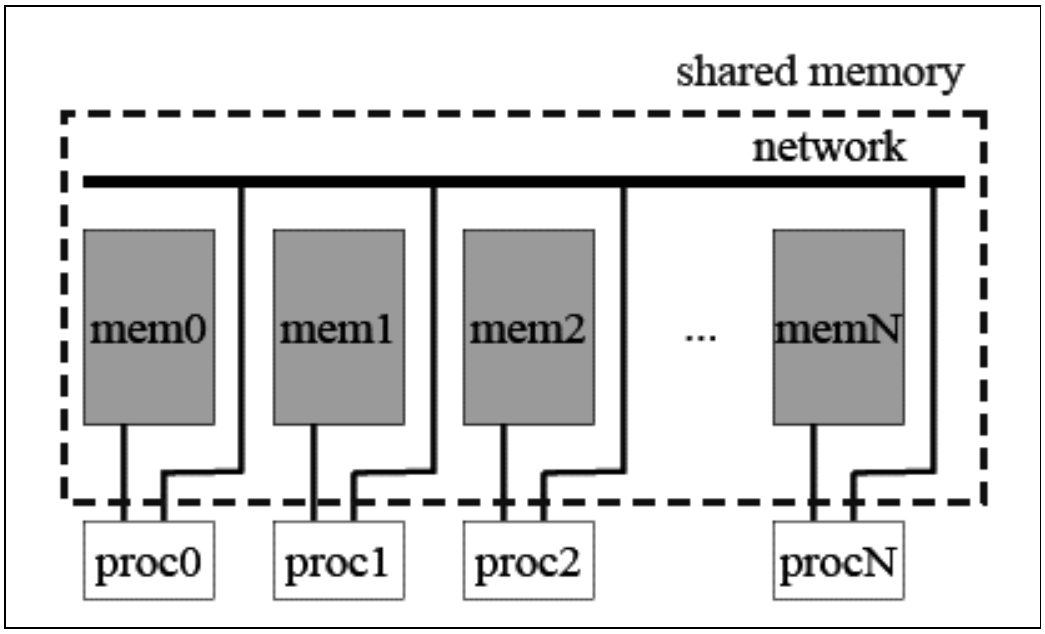

Figure 2.1: Distributed Memory Platform

Distributed Shared Memory enables programs to access data in traditional virtual memory. It is primarily a tool for parallel application or a group of applications in which individual shared data items can be accessed directly. A cluster is a form of multi-computer system which uses a collection of independent computers that are connected by a high-speed interconnection network. Since all communication between concurrently executing processes must be performed over the network, in such a system, until recently the programming model was limited to a message passing paradigm. However, recent systems have implemented the shared-memory abstraction on top of message-passing distributed-memory systems. The shared memory abstraction gives these

systems the illusion of a physically shared memory and allows programmers to use the shared-memory paradigm.

In systems that support Distributed Shared Memory, data moves between secondary memory and main memory as well as between main memories of different nodes. Each node can own data stored in the shared address space, and the ownership can change when data moves from one node to another. When a process accesses data in the shared address space, a mapping manager maps the shared memory address to the physical memory. The Distributed Shared Memory spares the programmer the concerns of message passing when writing applications that might otherwise have to use it. Shared memory provides the fastest possible communication, hence the greatest opportunity for concurrent execution.

Although DSM gives users an impression that all processors are sharing a unique piece of memory, in reality each processor can only access the memory it owns. Therefore the DSM must be able to bring in the contents of the memory from other processors when required. This gives rise to multiple copies of the same shared memory in different physical memories. The DSM has to maintain the consistency of these different copies, so that the any processor accessing the shared memory should return the correct result. A memory consistency model is responsible for the job. “Instantaneous” consistency is the one which written values are immediately available to others. The atomic consistency is just interesting theoretically for comparison. It ensures all procs sees all reads and writes in same order, where the order is corresponds to real-time order. The third one is sequential Consistency which all processors observe the same order. It must correspond to someserial order and only ordering constraint that reads or writes appear in the same order, but no restrictions on relative ordering between processors.

A few differences between Symmetric Multiprocessor (SMP) (Figure 1.2) and Software DSM are the delay tradeoffs, such as block size, cost of read/write misses and connections, where is using bus and long networks that depends on serialization and broadcast. The consequent is differences in protocols and applications with the bigger block size that cost amortization, higher hit ratio for larger blocks and reduced overhead. Therefore, migration and replication concept

is uses and also the false sharing increases. DSM protocol is more complex, it must handle lost, corrupted, and out-of-order packets.

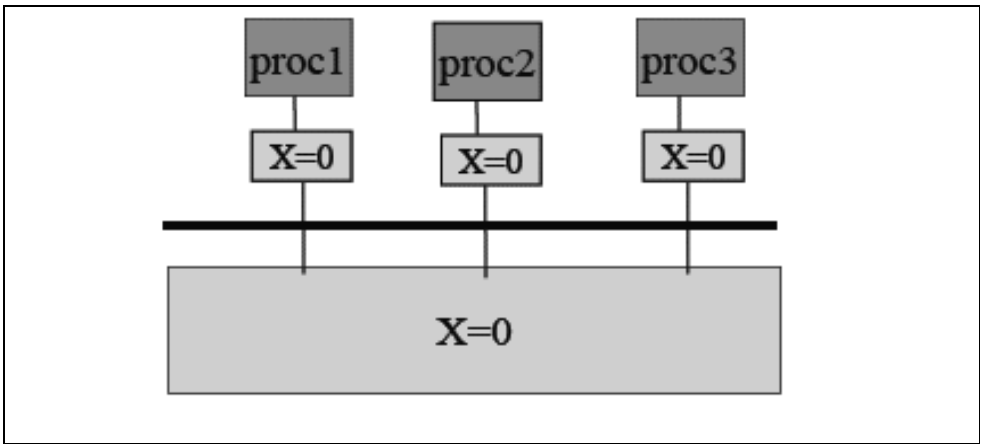

Figure 2.2: Multiprocessor (SMP)

2.3.2 Synchronization

Since the DSM processes may share memory, sharable objects must be created and associated with parallel processes. **Synchronization** constructs must be initialized in order to access sharable objects correctly.

Barriers, which are used as the coordination mechanism for parallel processes, must also be initialized. Distributed shared memory system uses messages passing to implement synchronizaitons. The synchronization of DSM processes sharing memory takes the form of semaphore-type synchronization for mutual exclusion and barriers are used to coordinate executing processes.

2.3.3 Barrier

The **barrier** that being used to coordinate executing processes is a method of synchronization that designates one processor as barrier manager. It is a mechanism that prevents any process from continuing past a specified point until all the processes are ready. A barrier is inserted at the point in each process where it must wait. All processes can continue from this point when all the process have reached it. When a process waits at a barrier, it sends an arrival message to the barrier manager and waits. When barrier manager has received all messages, it sends a departure message to all processes.

2.3.4 Semaphore

The **semaphore** is a protected variable (or abstract data type) and constitutes the classic method for restricting access to shared memory in a multiprogramming environment. Semaphores remain in common use in programming languages that do not intrinsically support other forms of synchronization. The trend in programming language development, though, is towards more structured forms of synchronization like monitors and channels. In addition to their inadequacies in dealing with deadlocks, semaphores do not protect the programmer from the easy mistakes of taking a semaphore that is already held by the same process, and forgetting to release a semaphore that has been taken.

2.3.5 Lock

Lock is another method for synchronization that designates one process as the lock manager for a particular lock. When a process acquires a lock, it sends an acquire message to the manager and waits. Manager forwards message to last acquirer. If lock is free, it send lock grant message. If lock is held, it holds on to request until free, and then send lock grant message.

2.3.6 Deadlock

An important consideration is being aable to avoid deadlock, which prevents prcesses from ever proceeding. Deadlock will occur for a specific condition when two processes are each waiting for the other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlocks are a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a lock. They are particularly troubling because there is no general solution to avoiding deadlocks.

2.3.7 False Sharing

One of the problem that might be facing is false sharing where the concurrent access to different data within the same consistency unit. Two flavors for false sharing occur is read-write false sharing in Figure 1.3 and write-write false sharing in Figure 1.4.

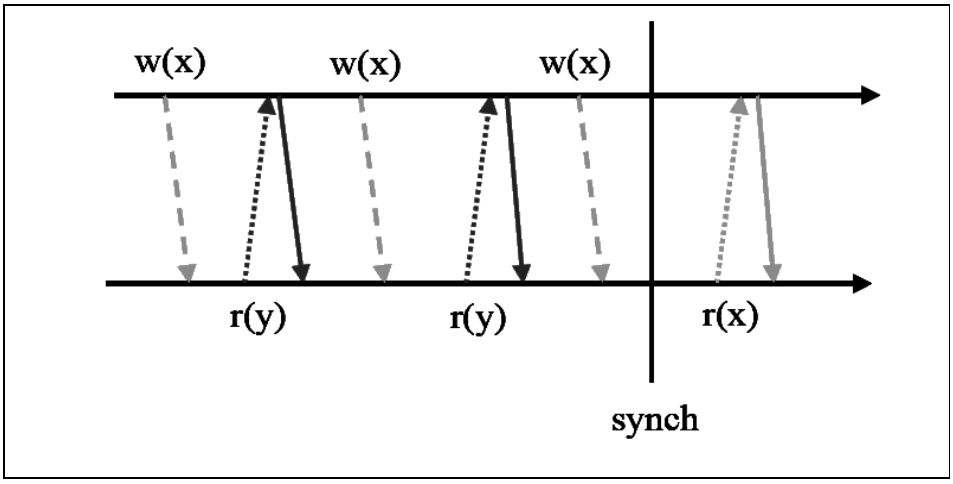

Figure 2.3: Read-Write False Sharing

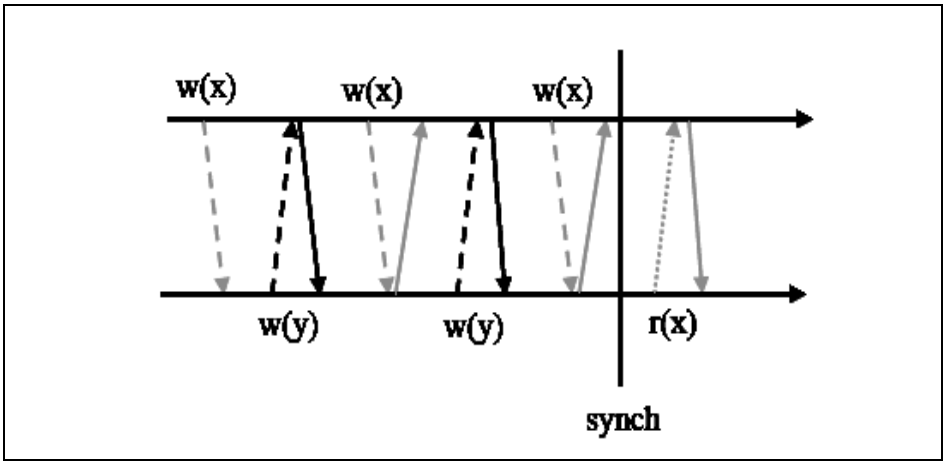

Figure 2.4: Write-Write False Sharing

As summary, the software implementation of DSM requires adding a software layer between the operating system and the application. One method is to structure the shared memory as objects in the distributed object-oriented systems on distributed memory hardware using virtual memory. The home migration to improve locality is important because of high latencies.

2.4 Java technology

Java technology is both a programming language and a platform.

2.4.1 Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the Java compiler (javac). A .class file does not contain code that is native to your processor; it instead contains bytecodes-- the machine language of the Java Virtual Machine. The Java launcher tool (java) then runs your application with an instance of the Java Virtual Machine.

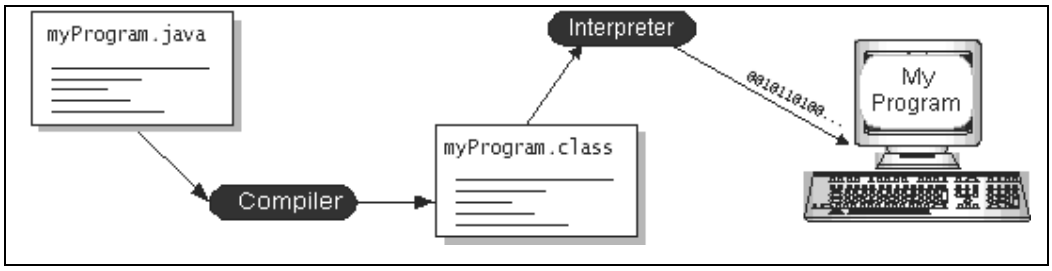

Figure 2.5: Java programming language

Because the Java Virtual Machine is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris TM Operating System (Solaris OS), Linux, or MacOS. Some virtual

machines, such as the Java HotSpot Virtual Machine , perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently-used sections of your code [8].

2.4.2 The Java Platform

A platform is the hardware or software environment in which a program runs. We have already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and MacOS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The Java Virtual Machine
- The Java Application Programming Interface (API)

The Java Virtual Machine is the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. It is grouped into libraries of related classes and interfaces; these libraries are known as packages.

The following figure depicts how the API and the Java Virtual Machine insulate the program from the hardware.

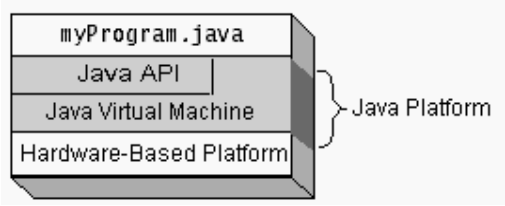

Figure 2.6: API & JVM

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability [8].

2.5 BlueJ

BlueJ is an integrated Java environment specifically designed for introductory teaching that developed at a University specifically for the purpose of teaching object orientation with Java to beginners.

BlueJ is implemented in Java, and regularly being tested on Solaris, Linux, Macintosh, and various Windows versions. It should run on all platforms supporting a recent Java virtual machine [7].

In overall, BlueJ offers:

- a project manager, a compiler, an editor, a debugger, a virtual machine
- full tool integration (compilation from within the editor, compiler error message display in the editor, setting breakpoints in the editor, etc.)
- abstraction from operating system
- class structure visualisation
- direct object interaction
- simplicity, easy-to-use interface

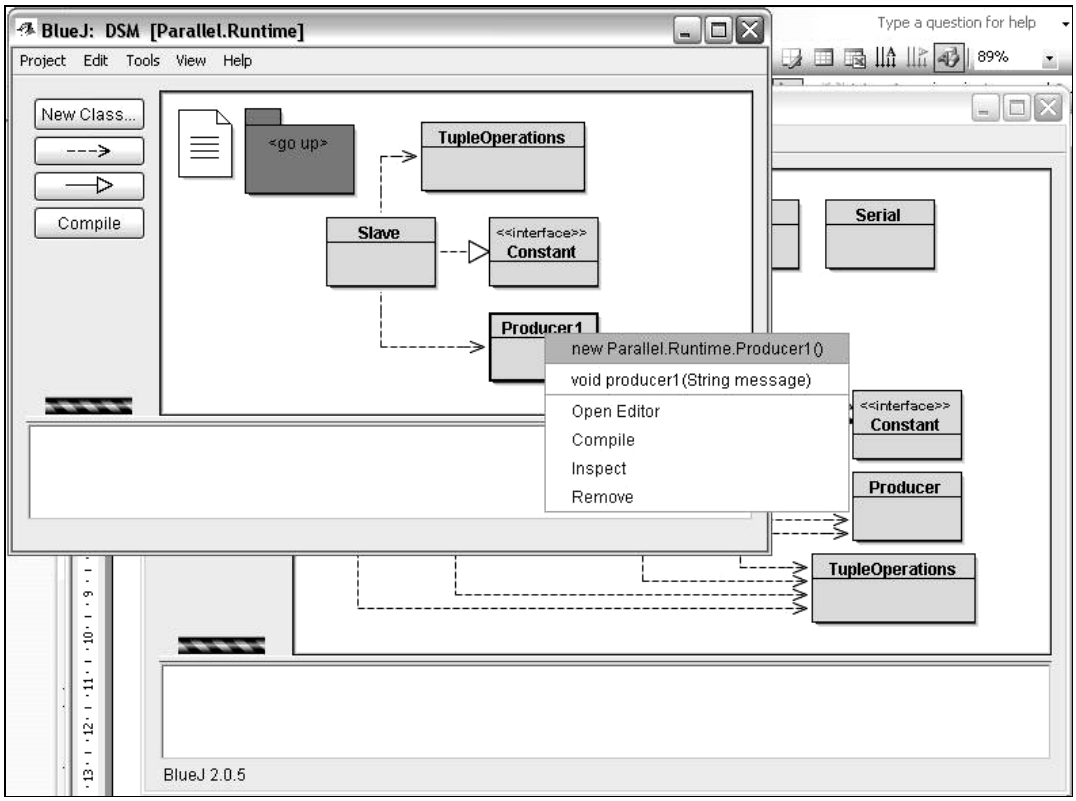

Figure 2.7: BlueJ interface

In this project, BlueJ is use as the main editing tools for design, compiling and implementing the program with Java programming language.

2.6 Borland C++ Builder 6

Borland C++ Builder 6 is used to implement the graphical user interface (GUI) for this project. In this parallel programming with distributed shared memory system, it acts as a shell to pass the parameter and trigger the running of the program (which is written in Java programming language) in command promt.

Chapter 3

METHODOLOGY

3.1 Introduction

In order to complete a piece of research, a well-founded methodology is required. In the context of the word reported in this thesis, the methodology is as usual have some phases which will be discuss from the chapter.

3.2 Design Flow Chart

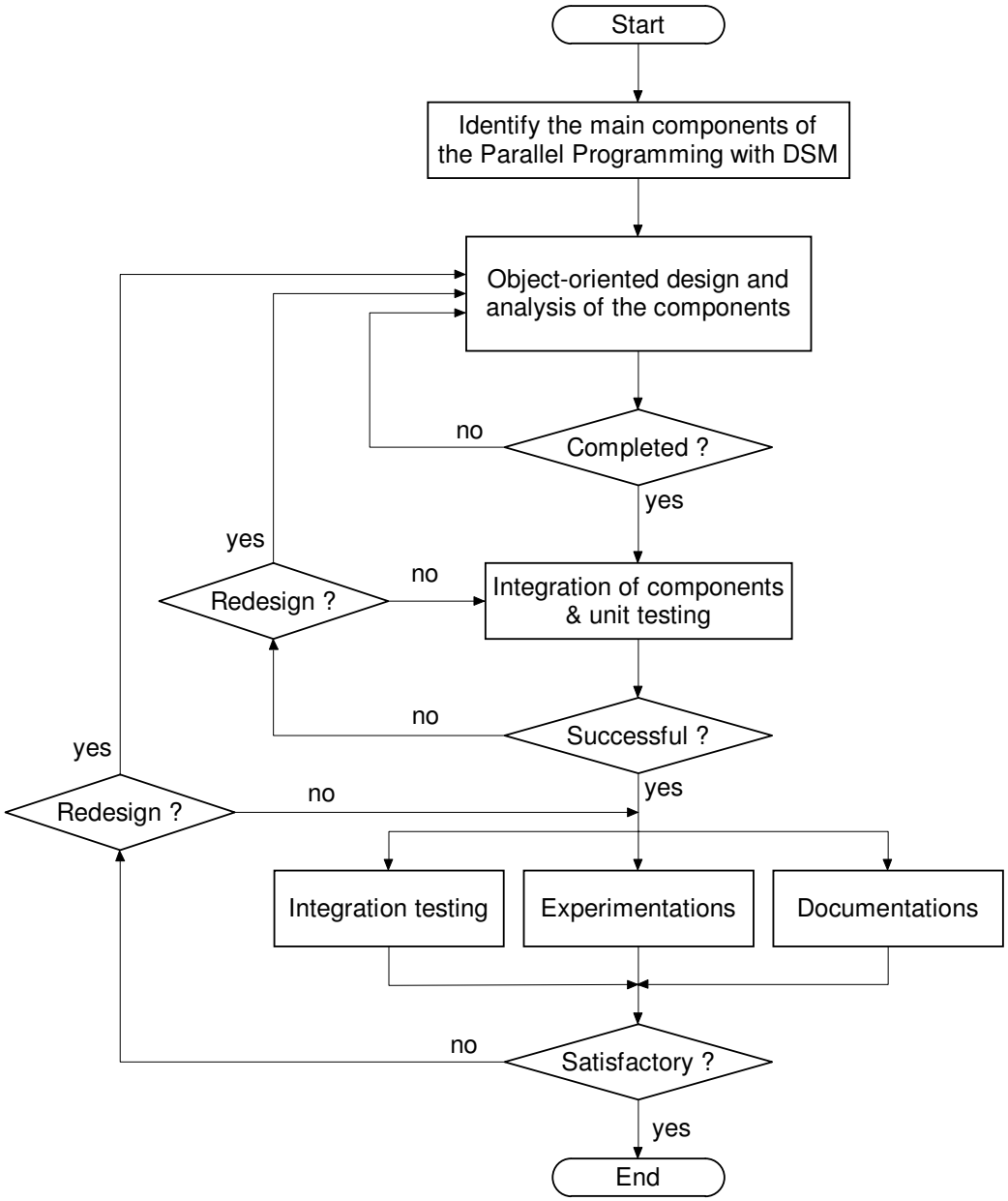

Figure 3.1: Design Flow Chart

3.3 Description of Phases

In overall, this project will be conducted into a number of phases:

Phase 1: Study & Learning

In this phase, the basic concept of the parallel programming with DSM is studied. The tools used to design the program are also being learned before start using it. All the information needed for completthis project will being search for reference.

Phase 2: Requirement elicitation

All the requirements for the parallel programming system with DSM will be identified. With these requirements, the main components of the system will bedecided.

Phase 3: Design and implementation of the program

Based on the decision in phase 2, the design and implementation of each of the components for the program will be considered utilising the object-oriented analysis and design tools. The overall design and implementation will undergo a number of iterations until all of the components are successfully completed.

Phase 4: Unit testing and integration

A number of test cases will be identified for each component. Unit testing will be performed to ensure that each component behaves properly as expected. The component that completed the testing will be integrated into the overall system.

Phase 5: Integration testing and experimentations

In this phase, the integration testing and evaluations will be performed. Finally, the overall system will be tested as to whether or not the program can be use in varies condition and system environment.

3.4 Gantt Chart

The following is the Gantt chart of the process to complete this project.

Table 3.1: Gantt Chart

<table>
<tr><th>Task / Month</th><th colspan="5">Week</th></tr>
<tr><th></th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>August</td><td colspan="4">Literature review</td><td>Write the proposal</td></tr>
<tr><td>September</td><td>Hand out proposal on 5th September</td><td colspan="4">Learning the concepts of parallel progrmming and Java Source code</td></tr>
<tr><td>October
November
December
January
February</td><td colspan="5">o Design the software system architecture
o Writing and compiling the program
o Improve the simulation results
o Finalize the software system
o Implement the user interface</td></tr>
<tr><td>March</td><td colspan="3">Write the FYP thesis</td><td>Hand out the FYP thesis draft on 27th March</td><td>-</td></tr>
<tr><td>April</td><td>-</td><td colspan="3">Viva</td><td>Finalize the final draft of FYP thesis</td></tr>
<tr><td>May</td><td>Finalize the final draft of FYP thesis</td><td>Hand out Final Draft of FYP thesis before 12th May</td><td colspan="3">-</td></tr>
</table>

Chapter 4

IMPLEMENTATION OF PARALLEL PROGRAMMING WITH DISTRIBUTED SHARED MEMORY

4.1. Introduction

This project is to design and implement a system that uses the concepts of parallel computation to compute the matrix multiplication problem which needs complex iteration computation. The design will be concerntrate on the comparison of the computation speed between the parallel computation and serial computation techniques. The goal is to get a faster computaion speed using the parallel computation techniques.

This chapter will outline all the implementations of parallel programming with distributed shared memory (DSM) include the overview of implementation of DSM, overview of the application of parallel programming with DSM in matrix multiplication, software design architecture, implementation of the matrix multiplication system and the results of the matrix multiplication.

4.2. Overview of implementation of Distributed Shared Memory

4.2.1. Distributed Shared Memory

From a programming viewpoint, ditributed shared memory (DSM) approach is where the memory is grouped together and sharable between the processor. In the software means approach, it can be used with ease on existing cluster a little or no cost except for the effort of installing the software, although the performance of software DSM will generally be inferior to using explicit message-passing on the same cluster.

Distributed shared memory is the designation for making a group of interconnected computers, each with its own memory, appear as though the physically distributed memory is a single memory with a single address space. Once distributed shared memory is achieved, any memory location can be accessed by any of the processor whether or not the memory resides locally, and normal shared memory memory programming techniques can be used.

In a DSM system implemented on a cluster, message are sent between computers to move data between them, but this message-passing is hidden from the user. The user does not have to specify the message explicitly in the program. Simply using the appropriate shared memory constructs or routines to access shared data will instigate the necessary message-passing. It will be up to the underlying DSM system to decide what messages to send and whether to replicate data or to actually move it from one computer to another [2].

4.2.2. Implementing Distributed Shared Memory

In the software approach, no hardware changes are made to the cluster and everything has to be done by software routines. Usually, a software layer is added between the operating system and the application. The kernel of the operating system may or may not be modified, depending upon the implementation. The software layer can be:

- Page based
- Shared variable based
- Object based

In the page-based approach, the system's existing virtual memory is used to instigate movement of data between computers, which occurs when the paged referenced does not reside locally. Major disadvantages of the page-based approach come from the fact that the unit of data being moved is a complete page. This lead to longer messages than are necessary. Also, false sharing effect appear at the page level and may be even more significant than at the cache level because of the large size of the page. Hence, page-based systems may not be very portable, sice thay are generally tied to particular virtual memory hardware and software.

In the shared-variable approach, only variables that are declared as shared are transferred, and this is done on demand. The paging mechanism is not to cause the transfer. Instead, software routines, called by the programmer directly or indirectly, perform the actions.

In the object-based approach, the shared data are embodied in object which include data items and the only procedures (methods) that may be used to access this data. In other aspects, it is the similar to the shared-variable approach and

can be regarded as an extention of this approach. It is relatively easy to implement using an object-based language such as C++ or Java and has the advantage over the shared-variable approach of providing an object-oriented discipline [2].

The software distributed shared memory system that being used in this research is implemented by using Java programming language with the object-based approach. The uses of this system will be discuss in another section in this chapter.

4.3. Overview of the Application of Parallel Programming with DSM in Matrix Multiplication

4.3.1. Review of Matrices

The underlying basis for many scientific problems is the matrix. A **matrix** is a two-dimensional array of numbers (or variables representing numbers). An $m \times n$ matrix **A** is shown in Figure 4.1. Thisstructure will be familiar from sequential programming as a two-dimensional array, and an array would typically be used to stored a matrix [1].

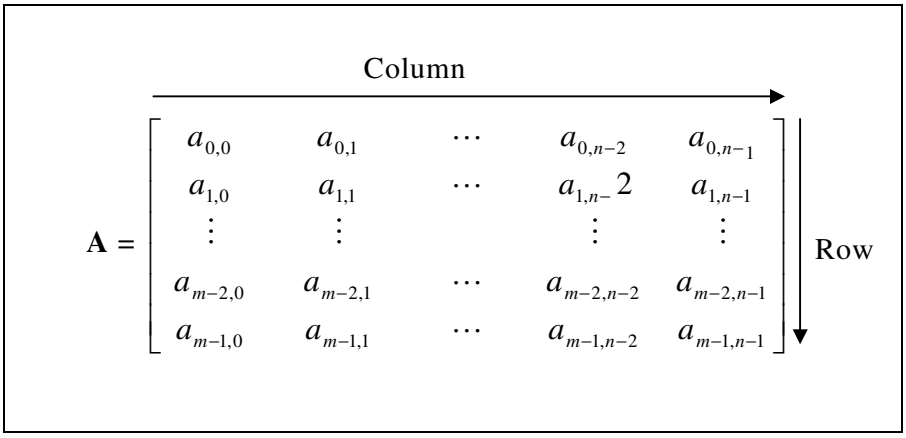

Figure 4.1: An $m \times n$ matrix

4.3.1.1. Matrix Addition

Matrix addition simply involves adding corresponding elements of each matrix to form the result matrix. Given the elements of A as $a_{i,j}$ and the elements of B as $b_{i,j}$, each element of C is computed as

$$c_{i,j} = a_{i,j} + b_{i,j} \quad (0 \le i < n, 0 \le j < m) \tag{4.1}$$

4.3.1.2. Matrix Multiplication

Multiplication of teo matrices, **A** and **B**, produces the matrix C whose elements, $c_{i,j}(0 \le i < n, 0 \le j < m)$, are computed as follows:

$$c_{i,j} = \sum_{k=0}^{l-1} a_{i,j}, b_{k,j} \tag{4.2}$$

Where **A** is an $n \times l$ matrix and **B** is an $l \times m$ matrix. Each element of the *i*th row of **A** is multiplied by an element of the *j*th column of **B** and the products summed together to obtain the value of the element in the *i*th row and *j*th column of **C**. The number of columns in **A** must be the same as the number of rows in **B**, but otherwise can be of different sizes. Matrices can also be multiplied by constants (all elements multiplied by the same constant) [1].

4.3.2. Implementing Matrix Multiplication

The implimentation of matrix multiplication can be done in two different way, which is in sequential and in parallel.

4.3.2.1. Sequential Matrix Multiplication

For convenience, let us assume throughout that the matrices are square ($n \times n$ Matrices). From the previous definition of matrix multiplication (4.2), the **sequential** way to compute $\mathbf{A} \times \mathbf{B}$ could be simply write as the code in Figure 4.2.

```
for ( i = 0; i < n; i++)
   for ( j = 0; j < n; j++) {
      c[i][j] = 0;
      for ( k = 0; k < n; k++)
         c[i][j] = c[i][j] + a [i][k] * b[k][j];
     }
```

Figure 4.2: Code of sequential matrix multiplication

This algorithm requires n^3 multiplications and n^3 additions, leading to a sequential time complexity of $O(n^3)$. For computational efficiency, a temporary variable, say `sum`, could be substituded for `c[i][j]` so that an address calculation is not specified within each iteration of the inner for loop [1].

4.3.2.2. Parallel Matrix Multiplication

Parallel matrix multiplication is usually based upon the direct sequential matrix multiplication algorithm. Even a superficial look at the sequential code reveals that the computation in each iteration of two outer loops is not dependent upon any other iteration and each instance of the inner loop could be executed in parallel. Hence, with n processors (and $n \times n$ matrices), a paralleltime complexity of $O(n^2)$ can be expected.

Usually, we want to use far fewer than n processor with $n \times n$ matrices because of the size of n. Then each processor operates upon a group of data points (data partitioning). **Partitioning** can be done very easily with matrix multiplication. Each matrix can be divided into blocks of elements called **submatrices**. These submatrices can be manipulated as if there were single matrix elements [1].

4.3.3. Recursive Implementation

The block matrix multiplication algorithm suggests a recursive divide-and-conquer solution. The method has significant potential for parallel implementations, especially shared memory implementations.

First, consider two $n \times n$ matrices, **A** and **B**, where n is a power of 2. Each matrix is divided to four square submatrices, as shown in Figure 4.3.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \times \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

(a) Matrices

$$\begin{matrix} & A_{0,0} & & B_{0,0} & & A_{0,1} & & B_{1,0} \end{matrix}$$
$$\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \times \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} + \begin{bmatrix} a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,3} \end{bmatrix} \times \begin{bmatrix} b_{2,0} & b_{2,1} \\ b_{3,0} & b_{3,1} \end{bmatrix}$$
$$= \begin{bmatrix} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} & a_{0,0}b_{0,1} + a_{0,1}b_{1,1} \\ a_{1,0}b_{0,0} + a_{1,1}b_{1,0} & a_{1,0}b_{0,1} + a_{1,1}b_{1,1} \end{bmatrix} + \begin{bmatrix} a_{0,2}b_{2,0} + a_{0,3}b_{3,0} & a_{0,2}b_{2,1} + a_{0,3}b_{3,1} \\ a_{1,2}b_{2,0} + a_{1,3}b_{3,0} & a_{1,2}b_{2,1} + a_{1,3}b_{3,1} \end{bmatrix}$$
$$= \begin{bmatrix} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} + a_{0,2}b_{2,0} + a_{0,3}b_{3,0} & a_{0,0}b_{0,1} + a_{0,1}b_{1,1} + a_{0,2}b_{2,1} + a_{0,3}b_{3,1} \\ a_{1,0}b_{0,0} + a_{1,1}b_{1,0} + a_{1,2}b_{2,0} + a_{1,3}b_{3,0} & a_{1,0}b_{0,1} + a_{1,1}b_{1,1} + a_{1,2}b_{2,1} + a_{1,3}b_{3,1} \end{bmatrix}$$
$$= \quad C_{0,0}$$

(b) Multiplying $A_{0,0} \times B_{0,0}$ to obtain $C_{0,0}$

Figure 4.3: Submatrix multiplication

Suppose the submatrices of **A** are labeled A_{pp}, A_{pq}, A_{qp} and A_{qq} and the submatrices of **B** are labeled B_{pp}, B_{pq}, B_{qp} and B_{qq} (p and q identifying the row and column positions). The final answer requires eight pairs of submatrices to be multiplied, $A_{pp} \times B_{pp}$, $A_{pq} \times B_{qp}$, $A_{pp} \times B_{pq}$, $A_{pq} \times B_{qq}$, $A_{qp} \times B_{pp}$, $A_{qq} \times B_{qp}$, $A_{qp} \times B_{pq}$ and $A_{qq} \times B_{qq}$, and pairs of results to be added, as shown in Figure 4.4. The same algorithm could do each submatrix multiplication, by decomposing each submatrix into four sub-submatrices, and so on.

Each of the eight recursive multiplication can be performed simultaneously by separate processors. More processors can be assigned after further recursive calls. Genarally, the number of processors needs to be a power of 4 if each processor is to be given one task of the tasks created by the recursive calls. The level of recursion can be limited by stopping not when the number of elements on each row and column of each submatrix is 1, but at some higher number dictated by the number of processor available. With four processors, it may be better to stop the recusion at the first level, because any further division of the problems still requires the tasks to be mapped onto these processors [1].

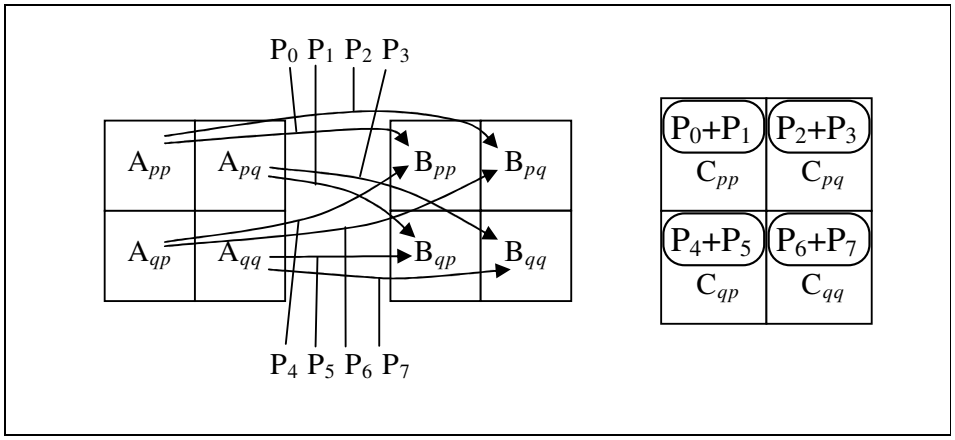

Figure 4.4: Submatrix multiplication and summation

A very advantageous aspect of the method is that at each recursion, the data being passed is edused and localized. This is ideal for the best performance of a multiprocessor system with cache memory. The method is especially suitable for shared memory systems.

4.4. Software Design Architecture

4.4.1. Introduction

The software design architecture of the Matrix Multiplication using Parallel Programming with DSM is by the uses of Java programming language with OOAD concepts and UML method.

4.4.2. Java and OOAD

Java is a high level language that supports OOAD (object oriented analysis and design) concept. By using OOAD concept, the system is viewed as a collection of interacting objects. Objects are instances of class and communicate by exchanging methods calls. OOAD concept is easily map to real world concepts (i.e. objects or classes). More over most new generations programming language are object oriented based. So it is based on this for the design in this project.