A Low-cost JAVA-based Parallel Computing Platform for Integrated Measurement and Data Processing

Chun Che Fung, Carl Engel and Halit Eren
School of Electrical and Computer Engineering
Curtin University of Technology
GPO Box U1987, Perth 6845
Western Australia, AUSTRALIA
Phone: +61 8 9266-2575, Fax: +61 8 9266-2584
Email: TFUNGCC@cc.curtin.edu.au

Abstract

This paper reports an investigation into the feasibility of utilizing a low-cost JAVA-based parallel computing platform for measurement and data processing purposes. The proliferation of high speed, low latency networking has provided a useful and inexpensive alternative to supercomputers and other specialized data processing hardware. The proposal is to use a number of networked PC's for the execution of distributed tasks. In the past, these functions were mainly executed exclusively on high performance computers. Several arbitrary tasks are presented that are executed in parallel on a number of desktop PC's, each of which are networked by a standard IP stack. The programming environment utilized is Java 2.0, from Sun Microsystems. The Java programming language has definite advantages such as independence from platforms and suitability for network executions. The proposed approach has good potential for distributed measurement and data processing applications.

1. Introduction

With the advancements in computer and network engineering, majority of computers are now interconnected in the business, industrial and academic environments. In the business world, networking facilitates communication and sharing of information. The latest development in Electronic Commerce is expected to bring upon new trends in business operations and consumer behaviour. In industrial applications, networking enables remote measurements, distributed control, data acquisitions and processing. Generally, computer networks improve productivity and quality [1].

While the computing power of CPU's has increased greatly over the past years, there are computationally intensive applications that demand high performance computing platforms such as mainframes or supercomputers. Examples of such applications are large volume data acquisition and processing, Computer-aided-designs, high-resolution graphics and animations, simulations and modelling of complex systems.

On the other hand, many computers in the networked configurations are idling or under-utilised. For example, most business computers will be turned off or unattended during after office hours. Even during normal working hours, the chances of a desktop computer being fully utilised all the time are extremely rare. Hence, if the collective computing power in a networked configuration can be harvested to solve problems that demand high amount of computing time, the result will be a marked improvement in productivity without additional investments in specialised hardware or equipment.

This study investigates the development of a low-cost JAVA-based parallel computing platform for integrated measurement and data processing. The applications of multiple networked standard PCs to typical tasks associated with parallel computation are studied. The proposed system requires no hardware modifications, and only depends on a typical office or industrial set-up. Application of the proposed system to the Neural Network (NN) and Genetic Algorithms (GA) are also described in this paper.

2. Parallel Programming Paradigms

While much of the literature on parallel programming have discussed the various benefits and disadvantages of the variety of inter-connection networks [2] and [3], most of these cases and analyses are highly theoretical and bear little or no relevance in the real world. This is due to the unavailability of the system or the impracticality of their applications to real life problems. In addition, practitioners acknowledge that different networks have their unique benefits and disadvantages [4].

In this study, instead of specialized network configurations such as linear arrays, cubic or hyper cubic

0-7803-5890-2/00/$10.00 © 2000 IEEE

connections, inter-connected systems based on a TCP/IP socket connection are used. During the experimental studies, physical structure of the network was based on Ethernet. Messages are directed to nodes on the bus by specifying IP addresses in each packet. All nodes are connected to all others, but from the point of view of the Java virtual machine, the number of interconnections is fully dependent on the TCP/IP stack. In order to realize a parallel structure in the specified environment, a set of library routines have been developed that allows the linking of multiple computers with each other and for task synchronization. This program developed was named the *Parallel Worker*, or Para Worker. It is described in the following sections.

3. Para Worker

The Para Worker proposed in this paper is a command line driven program designed to facilitate the easy implementation of parallel algorithms over the defined structure. It can be operated both as a slave node and a master node with no reduction in functionality. Commands are issued to the program via the standard input and output streams. In the event that a node becomes slave to another, the standard input and output streams are redirected to the node that works as the slave, via the master-slave socket connection [5], [6], [7] and [8].

There are four distinct mechanisms that facilitate the implementation of parallel algorithms. These are the Master Mechanism, Peer Mechanism, Class Transfer Mechanism and the Message Passing Mechanism. Each mechanism performs a fundamental function that eases development of parallel and distributed algorithms.

3.1 Master Mechanism

The Master Mechanism is used by a node that is acting as a master node to a number of slave nodes. There are a variety of functions available via this class that ease the process of administrating and controlling the slaves.

First, the Master Mechanism provides a storage location for information about each of the slave nodes. It also decides whether or not other potential slaves are permitted to be connected to this master node. In addition, it also allows the slaves to be disconnected at will. Facilities are included that allow the master to issue instructions to one or all of the nodes.

The Master Mechanism is named as such because of the definite master-slave relationship between the nodes.

3.2 Peer Mechanism

The Peer Mechanism is a logical opposition to the Master Mechanism. Instead of defining relationships between nodes as master and slave, the nodes operating such Peer Mechanism are of equal importance and priority.

Similar to the Master Mechanism, the Peer Mechanism allows establishment or elimination of peer interconnection networks between the nodes. It also facilitates simple message passing between them.

3.3 Class Transfer Mechanism

The Class Transfer Mechanism provides facilities to transfer executable classes between nodes. The premise behind this mechanism is that a slave node does not need to possess a class object that a master node intends it to execute. If this event occurs, the master node can simply transfers the appropriate class to the slave, and then order the slave to execute the class.

The Class Transfer Mechanism provides a storage location for all downloaded classes. It also provides appropriate functionality for the transfer of such classes between the nodes.

3.4 Message Passing Mechanism

The Message Passing Mechanism facilitates the exchange of messages between nodes. It uses the User Datagram Protocol (UDP) as the communications protocol. The messages that are transmitted are not guaranteed to arrive at the destination, nor are they guaranteed to be correct. The advantage of this mechanism is that the message transfer is extremely fast and efficient. It also operates as a connection-less service. There is no need to establish a tangible link between nodes before transmission commences, and messages can be immediately sent.

The Message Passing Mechanism operates on a peer-to-peer basis and there is no differentiation between the importance of two particular nodes. This protocol has been adopted because it is assumed that the network configuration is in close proximity and minimal interference will be experienced.

4. Experimental Results

In order to ascertain the performance potential of Java as a distributed computing environment, several experiments have been developed and performed. Four experiments are outlined in this paper, each of which tests a different aspect of what is possible with the Para Worker. The Prime Number Generator is the simplest of all, utilizing only a master-slave relationship. The Parallel Sorting Network incorporates a complex peer interconnection network, and the Mandelbrot Image Generator divides the task into discrete work packages to be executed by idle nodes. The final test uses the cluster's

disk storage rather than its computational abilities to create a distributed file system.

In most cases, the experimental platform is based on a local network used in a laboratory environment. The network consists of fifteen standard PC's using Windows 95 Operating System. The computers were networked together using a 10Base2-Ethernet network. All of the machines had 16Mb of memory and the CPU clock speeds ranged between 100MHz and 300MHz. The application of such parallel network for measurement and data processing purposes will be discussed in subsequent sections.

4.1 Prime Number Generation

This test utilizes a master-slave interconnection network to generate prime numbers. Each slave is assigned with a particular range of integers to check for the existence of prime numbers. If a slave finds a prime number, then it returns the number to the master node for listing.

This algorithm works for an arbitrary number of slave nodes, and is therefore easily able to highlight the expected reduction in execution time due to increasing numbers of slaves. This is illustrated below in Figure 1.

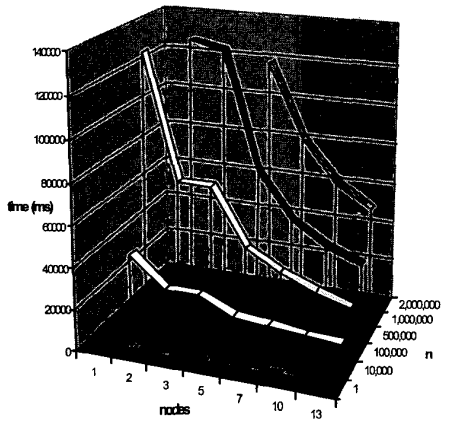

Figure 1: Prime Number Generation

This figure shows the amount of time taken to test a sequence of 5,000 integers, starting at 'n', for primeness. As illustrated, increasing the number of slaves reduces the execution time. Theoretically, doubling the number of slave nodes should half the total execution time. In this situation, the overhead processing time associated with the master node is minimal, resulting in ideal preferred results.

Figure 1 also shows the performance variation as a result of changes in the ratio between computation and communication requirements. For low values of 'n', the slave nodes have a relatively light workload, and the master node is saturated as it receives results, so the performance is similar no matter how many slaves are there. As the relative computational workload increases, communications overhead becomes less of an issue, and the expected hyperbolic relationship between execution time and number of nodes becomes apparent.

Prime number generation and analysis is frequently used in the context of data encryption and data security techniques. In order to decode an encrypted file, a public key is required along with a private key. Both of these keys are very large prime numbers, and their product has only these keys as factors. Due to the fact that security levels are increasing phenomenally and more users are interested in the technology, increasingly larger prime number pairs must be found.

4.2 Parallel Sorting Network

Sorting, being one of the most popular classes of algorithm studied, certainly could benefit from implementation on a parallel structure. There are many permutations by which a network could be configured to facilitate efficient sorting, all of which depend on the amount and type of data being sorted.

The parallel sorting network used in this project is essentially a two-stage structure that uses slave interconnections. The first stage uses four nodes that perform a sequential sort on a data set. The second stage consists of a five-element merging network [4] that eventually feeds a stream of sorted data back to the master node.

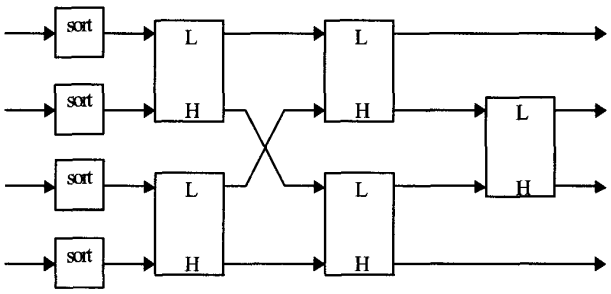

Figure 2: Parallel Sorting Network

The data set is initially generated by the master node, divided into four sections, and each section transmitted to the nodes in the first stage. When sorting is completed, the four ordered sets are fed into the merging network.

Each merging node takes two inputs; the greater one of the two inputs goes into a specific output, then the lesser goes into the other. Because this network is intended for many pieces of discrete data, no data will be sent through the second output until a predefined number of items are sent through the first.

Eventually a sorted data stream returns to the master node after a short period of time. The data arrives from separate streams that must be read from in sequential order to ensure data integrity. This particular application has relatively high communication requirements compared to its relative low computational requirements. This is especially apparent in the merging network stage. In order to achieve optimum performance, the communications network must be as fast as possible.

High-speed sorting mechanisms are especially useful in the realm of database management systems. Complex queries are often made to the system that requires significant work on the part of the server. In many cases, data must be scanned, searched, sorted and formatted many times before it becomes presentable to the user. The implementation of structures such as that described above would certainly benefit future practitioners of information systems and data mining, as complex database analysis could be completed in a short period of time.

4.3 Mandelbrot Image Calculation

The Mandelbrot Set is the oldest and most popular fractal image. The image is calculated by determining the number of recursive iterations of a simple complex number equation. Image calculation traditionally takes a long time to perform and is inherently parallel. Like the Prime Number Generation Algorithm outlined above, there is no need for the slave to slave communications, as there are no data inter-dependencies.

In this project, an algorithm was developed that calculates a specific area of the Mandelbrot Set by distributing the load amongst an arbitrary number of slave machines. The master node divides the work up into sections called "Work Packages", and sends consecutive packages to any idle nodes. The test is deemed complete when the master node notes that all work packages have been completed.

In the event that a slave node is found idle. It is issued a command to download and execute a class from the master. This class contains the code and parameters required to complete the work package. In this case, a subsection of the Mandelbrot Image is generated, and sent back to the master node.

The master node will then receive this data, mark the relevant node as idle, and then draw the appropriate data to the screen.

Figure 3 shows some preliminary results for the Parallel Mandelbrot Image generator. The chart shows the fastest execution time in seconds of the algorithm for between one and six slave nodes. As expected, the execution time decreases as the number of slave nodes increase, though not as much as with in the case of the previous tests.

With this particular test, load balancing becomes a fairly important issue. It is possible to alter the relative size of the work packages allocated to each node. If the work package size is too small, then most of the execution time will be spent during master-slave communication. A slave may spend only a fraction of its time computing the work package, and spend the majority transmitting, receiving and waiting for work packages. If the work packages are too large, then the master slave becomes inundated with large segments of data, and reduces performance as a result. Obtaining optimum performance with test requires a careful balance between work package size and the number of slave nodes.

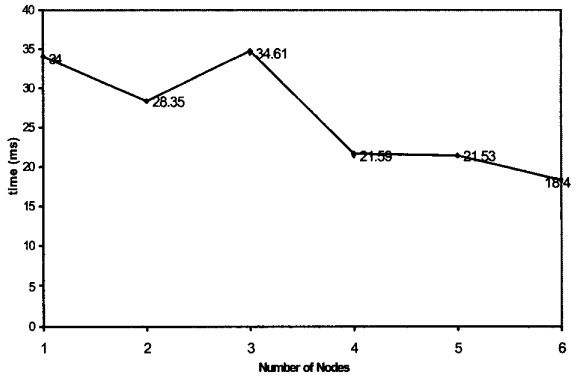

Figure 3: Mandelbrot Image Generation

This technique is used in a variety of image processing applications. While measurement based on images are gaining more common, the most notable example is the case of animated film industry. Recent computer generated films have often taken weeks to generate on state of the art machines. A typical two-hour film consists of up to 200,000 high quality images that take several minutes to generate. In most cases, scores of high-powered machines have the task subdivided between them to reduce the time it takes to create the films. The techniques used in this example are very similar to those used in the movie industries.

Another application of the image processing is in the field of remote sensing and satellite imagery. Everyday, surveying satellites transmit hundreds and thousands of detailed images to their base stations. These images range from astronomical observations, to surveying and weather imagery. In most cases the images need to be analyzed and processed by computers, and implementing techniques demonstrated with this application can immediately speed up the process.

4.4 Distributed file services

As detailed in the above examples, a cluster of machines is providing processor time to perform particular tasks. Another service that can be offered by

the same cluster of machines is storage space on their local secondary storage devices. It is also feasible to share physical memory between these machines, but the latency of an interconnection network far exceeds that of virtual memory devices such as the hard disk.

A distributed file system has been developed that was modeled on the file transfer protocol (FTP). Machines in the cluster are running a file transfer daemon (DFTPDaemon). Client machines, running a file transfer client (DFTPClient) transmit and receive files by issuing commands to the servers on an arbitrary UDP/IP Multicast address. The client need only transmit one single command, which will be received by all servers.

Load balancing semantics are based on the principle of "first in, first served". The first server to respond to a client is the server that will service the client. Using this philosophy, it is possible to infer the following facts about the successful server:

It is probably closest (network wise) to the client.
It is probably faster than the other servers.
It is probably less loaded than the other servers.
Hence, it is the ideal choice to service the request.

In this configuration, the servers exist entirely as a stateless program. They are constantly fielding and acting on commands from anonymous clients. The servers maintain no connection with each other, nor with the clients, with the exception of brief, transient, data transfer operations. The advantage of this environment is that the servers are completely autonomous, if a specific server suffers a failure, the others will perform as per normal.

5. Applications

While the above tests were designed to evaluate the feasibility of a low-cost network for parallel computing, implementation for practical applications has to be considered. Two algorithms particularly suitable for parallel executions have been investigated. They are Neural Networks and Genetic Algorithms.

An eight node neural network was implemented using one master machine and three slave machines. In total, there were two input nodes and three output nodes, with three nodes in the hidden layer. The architecture of the example neural network is illustrated in Figure 4. Two slave machines modeling the behavior of three nodes are connected in each layer. The third slave node models the remaining two nodes. The two nodes in the input layer are arbitrarily name "A" and "B", and the outputs are to be trained to provide a function matching between the input and output data patterns

Due to the work load divisions, there is a fair bit of internodal communication. For every iteration of the neural network, every node must send two signals to every other node in the network. These signals consist of a single floating point data type, and they are transferred through an Object I/O Stream.

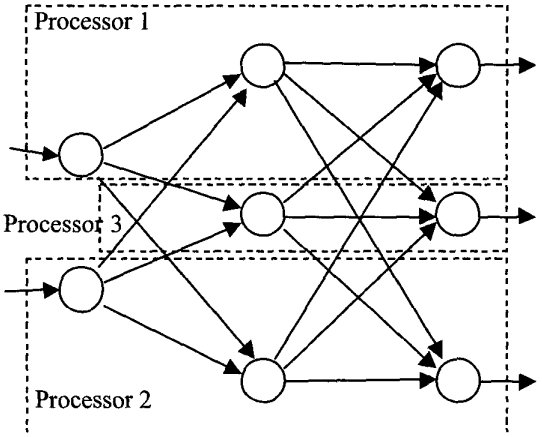

Figure 4: Neural Network Load Distribution

The end result of this configuration is that it worked, but slowly. Running the simulation on four computers achieved approximately one iteration per second, compared to hundreds for the single computer version. The major problem causing this is the internodal latency. The amount of time taken to transmit a relatively small piece of data is quite large. Considering the size of the data transmitted between nodes, the time taken to transmit the data is comparatively large.

Due to the fact that the neural networking test is inherently finely-grained, its performance has suffered significantly due to slow communications.

On the other hand, implementing a Genetic Algorithm (GA) in a parallel structure is beneficial because the algorithms are inherently well suited for parallel execution. GA's are typically applied to optimization problems. In the field of instrumentation and measurement, GA can form the backbone of an intelligent sensory system for the selection of the most appropriate parameters.

The process of finding a fitter chromosome is quite computationally intensive. Successive iterations must be performed on the chromosome and its fitness must be tested. Typically, a chromosome is created that contains several genes. GA operates on a population of "genes" which are essentially solutions in the problem space. The premise behind GA is the performance of certain operations on the chromosome will eventually lead to solutions such that it is deemed to be fitter than its ancestor. After several generations, the best, or near optimal solution (genes) will be found.

Two separate techniques used in this study. These were:

1. The Cluster Structure
2. The Distribute and Recombine Structure

The Cluster Structure created a parallel structure similar to that described in Figure 5. This implementation had the slave nodes transmitting their best chromosomes to adjacent slave nodes. The process would more or less work in a similar manner save that each slave node would not necessarily obtain the best chromosome from the entire group. This is not necessarily a problem due to the fact that the best chromosome may inevitably lead to stagnation in fitness for future generations. Local minimal points that prevent fitness from increasing beyond a certain value cause this anomaly.

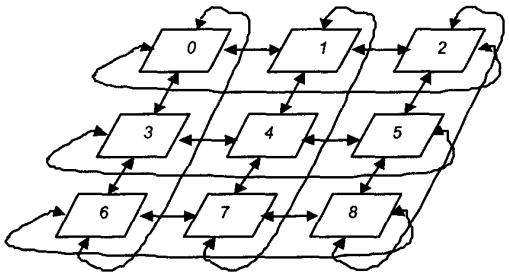

Figure 5: Genetic Algorithms Cluster Structure

The Distribute and Recombine Structure required no slave interconnections. There were ten slave nodes. Each slave node would iterate through one thousand successive generations, and return its results when complete. The master node would collect and collate these results. From the nine chromosomes received, the fittest chromosome would be found and redistributed to all of the slave nodes. Essentially the fittest chromosome is being retained whilst the others are discarded. This process would be repeated a number of generations so eventually a relatively fit chromosome would result.

The best fitness result was obtained using the cluster structure combined with a common list for the storage of the feasible solutions. Amongst all tests, this achieved the fifth fastest execution time. The test resulting in the fastest execution time was the same structure. This test also produced the worst fitness value.

6. Conclusion

Standard Java API provides a lot of the functionality required to implement parallel and distributed algorithms on multiple computers. The implementation of parallel and distributed algorithms has the potential to solve many problems and perform many tasks faster than a usual single processor machine. There is a great potential when a quantity of machines are combined together to help solve a problem. As networking proliferates around the world, the means by which these machines can work together is more easily realized. The Java programming language possesses a certain affinity with the Internet that gives it definite advantages when implementing parallel programs. Java classes can be executed on a variety of platforms without the need for porting and recompilation. Java classes can be transmitted over the network, executed on a remote machine, and have the results returned back to the instantiating machine. All of these functions can easily be performed without any physical modifications to common off-the-shelf components that are available in quantity and for low cost. Further works are carried out to evaluate the Para Worker's performance in a practical system for measurement and DP applications.

References

[1] Buyya R, 1999, "Cluster Computing: The Commodity Supercomputing", IEEE Computer Society Workshop, March 1999, Curtin University of Technology, Perth.

[2] Foster I & Kesselman C, 1999, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, Inc., San Francisco, USA.

[3] Hennessy JL & Patterson DA, 1998, "Computer Organisation and Design: The Hardware/Software Interface", Morgan Kaufmann Publishers, Inc., San Francisco, USA.

[4] Akl S G, 1989, "The Design and Analysis of Parallel Algorithms", Prentice-Hall International, New Jersey.

[5] Kistler T & Franz M, 1999, "A Tree-Based Alternative to Java Byte-Codes" in International Journal of Parallel Programming, Vol. 27, No. 1, Feb. 1999, pp21-33.

[6] Sun Microsystems Internet Web Site (1999) http://www.sun.com/

[7] Wolfe A, 1997, "Much-Needed Tune-Up Taking Shape For Java" at Internet Web Address: http://www.techweb.com/wire/news/1997/09/0916java.html

[8] Finkel D, Will CE, Brennan B & Brennan C, 1999, "Distriblets: Java-based distributed Computing on the Web" in Internet Research: Electronic Networking Applications and Policy, Vol. 9, No. 1, 1999, pp. 35-40.