A Performance Comparison of DSM, PVM, and MPI

Paul Werstein
University of Otago
Dunedin, New Zealand
Email:werstein@cs.otago.ac.nz

Mark Pethick
University of Otago
Dunedin, New Zealand
Email:mpethick@cs.otago.ac.nz

Zhiyi Huang
University of Otago
Dunedin, New Zealand
Email:hzy@cs.otago.ac.nz

***Abstract*— We compare the performance of the Treadmarks DSM system with two popular message passing systems (PVM and MPI). The comparison is done on 1, 2, 4, 8, 16, 24, and 32 nodes. Applications are chosen to represent three classes of problems: loosely synchronous, embarrassingly parallel, and synchronous. The results show DSM has similar performance to message passing for the embarrassingly parallel class. However the performance of DSM is lower than PVM and MPI for the synchronous and loosely synchronous classes of problems. An analysis of the reasons is presented.**

I. Introduction

A number of computational problems cannot be solved in a reasonable time on currently available computers. Parallel computing provides a method to increase the performance of computationally intensive programs by distributing their execution across multiple processors. Traditionally this parallel processing was done on expensive purpose-built multiprocessor systems. An alternative which is becoming more popular is the use of multicomputer or cluster computer systems.

A cluster computer system consists of a collection of computers connected by a network. Typically the computers are standard off-the-shelf personal computers. An example of such a system is a Beowulf cluster [1].

Each node in the cluster has its own memory, accessible only to the processes running on that node. Programs running in parallel on different nodes require some way to communicate. The most common solution is the use of libraries which provide primitives to enable the parallel processes to communicate. Many of these libraries run in user space so no alteration of the operating system is required.

To date, most of the libraries are based on the message passing paradigm. This requires a programmer to specify explicitly the data to be passed between processes and the time when the communications should take place. An alternative approach provides the abstraction of a single memory address space available to all processes running in the cluster. This approach is known as distributed shared memory (DSM). It frees a programmer from having to deal explicitly with data exchange, and therefore, provides an easier method of programming. The DSM library is responsible for memory consistency maintenance and ensures each process has access to the most recent version of data.

Since cluster computers are fairly new, relatively little is known about the performance of DSM compared to message passing systems on them. This report details the results of a series of tests carried out to compare the performance of a DSM system with two popular message passing systems on cluster computers.

In [2], the performance of DSM is compared with that of PVM (Parallel Virtual Machine) [3] on an 8-node cluster. Our tests show the scalability of DSM, PVM and MPI (Message Passing Interface) [4] on 1, 2, 4, 8, 16, 24, and 32 nodes. Our tests show the number of nodes becomes significant when more than 8 nodes are present.

The DSM system used is Treadmarks (version 1.0.3.3) developed at Rice University [5]. The message passing systems are PVM (version pvm3.4.3) [3] and the MPICH implementation [4] of the MPI standard. We used MPICH, version 1.2.4, developed at the University of Chicago and Mississippi State University [4]. Tests were carried out on a cluster of 32 PC's running Linux.

Three programs were used representing a range of applications. One program is mergesort which is a loosely synchronous problem. Another program is a Mandelbrot set generator which is an embarrassingly parallel application and gives an indication of performance under almost ideal conditions. The last program is a back propagation neural network which is a synchronous application, giving an indication of the performance of the systems in the general case. The terms, loosely synchronous, embarrassingly parallel, and synchronous, are from [6].

The next section outlines the parallel programming environments and the libraries used in these tests. Section III describes three applications used in the tests. Section IV describes the methodology used. The results are presented and discussed in Section V. Conclusions are in Section VI.

II. Parallel Program Environments

There are two kinds of programming environments for cluster computers: message passing and distributed shared memory. These environments are based on libraries running in user space. Thus they require no alterations to the underlying operating system. Each library provides a set of functions or primitives, allowing a programmer to create distributed programs.

When a program is parallelised and run on multiple processors, it is necessary to share some part of the program memory space. Examples of this shared data include shared variables and arrays. This leads to the concept of distributed memory. The most common approach for distributed memory is message passing. The supporting libraries provide message passing primitives which use a network to pass data between nodes. Since the processes do not share memory, they do not need memory sharing primitives such as locks or semaphores. However the processes still require some way to synchronise themselves. This can be done with process synchronisation primitives such as barriers. Unfortunately, ensuring that all message passing is correct places an extra burden on programmers and increases the chance for errors.

An alternative to the message passing paradigm is distributed shared memory (DSM). In the DSM paradigm, a portion of each processors' memory appears as shared memory which is available to all processes in a parallel program. This reduces the burden on programmers. DSM provides the illusion of shared memory [7]. Each processor has its own memory in which the DSM system stores a copy of the shared memory. The primary role of the DSM system is to maintain the consistency of each copy of the shared memory as well as provide a means to ensure the correct execution of the program. A DSM library typically provides a full range of synchronisation primitives, including locks, semaphores, and barriers.

The two message passing systems used in our tests are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). The distributed shared memory system used is Treadmarks. These systems are detailed in the next subsections.

A. Treadmarks Distributed Shared Memory

Treadmarks was developed at Rice University for research into DSM systems. It provides locks and barriers as primitives. Other synchronisation primitives such as semaphores can be simulated using locks, barriers, and shared variables. Treadmarks uses lazy release consistency which is a weak consistency model. It takes advantage of the fact that in a data race-free program the synchronisation primitives divide the program into non-overlapping blocks. If a shared variable is accessed by process *A* in a given block, it is guaranteed that no other process will access that variable until process *A* has left the block that protects the shared variable [8].

The granularity of shared data in Treadmarks is a page. Treadmarks uses the virtual memory system of the host machine to invalidate modified copies of shared pages. A page fault on a shared page causes an invalid page fault signal (SIGSEGV) to be raised, which is caught by the library. The lazy release consistency protocol uses invalidation to achieve time and processor selection. When a process acquires a lock or exits a barrier, it learns of any changes to shared memory from the process which last released the lock, and invalidates modified pages. When a process first accesses an invalidated page, a page fault is generated. This scheme reduces the amount of data being transferred by only sending pages that are accessed.

B. Parallel Virtual Machine

The PVM library is developed around the concept of a virtual machine that provides a single logical machine across the distributed memory cluster. From a programmer's perspective, all processes appear to be on a single machine. A virtual machine is associated with the user that starts it. The virtual machine exists as a daemon on each of the nodes in the cluster. Processes communicate via the daemon, which is responsible for handling communication between nodes [3].

Within PVM, resource management is dynamic. PVM includes a console program to manage the virtual machine. The console allows nodes to be added to or deleted from the virtual machine and provides process management utilities.

C. Message Passing Interface

The second message passing system is an implementation of the Message Passing Interface (MPI) standard. The MPI specification was defined by the MPI Forum in the mid 1990s to provide a standard message passing interface for system vendors [9].

The specification is primarily concerned with providing communication primitives. It defines a large set of functions for point to point and group communications.

III. Applications

Three programs were used in the performance tests: mergesort, Mandelbrot set generator, and a back propagation neural network. Each is described below.

A. Mergesort

Mergesort sorts data items by recursively dividing the items to be sorted into two groups, sorting each group, and merging them into a final sorted sequence. Given the recursive nature of this algorithm, the parallelisation strategy used is divide and conquer. Each node in the cluster sorts a subsection of the list. Subsequently, for each pair of nodes, one node merges the sorted subsections for the two nodes. This process continues until one node has the fully sorted list. Synchronisation only occurs between pairs of nodes. Thus mergesort belongs to the loosely synchronous class of problems.

B. Mandelbrot Set

A Mandelbrot set contains all complex numbers which do not run to infinity when iterated through some function [10]. The most commonly used function, and the one used in this article, is $Z_{k+1} = Z_k^2 + C$ where C is a point in the complex plane and $Z_0 = 0 + i0$. C is presumed to not run to infinity if its size is less than two after X iterations of the function. The value used for X is these tests is 1000. Two is used as the limit for C since it is known that no points with C greater than two belong in a Mandelbrot set.

The Mandelbrot program not only determines which points are in the set, but also how quickly points not in the set run to infinity. This determines the colouring of pictures in a Mandelbrot set. Thus the result of calculating a Mandelbrot set is the colour of each pixel representing points in the visible

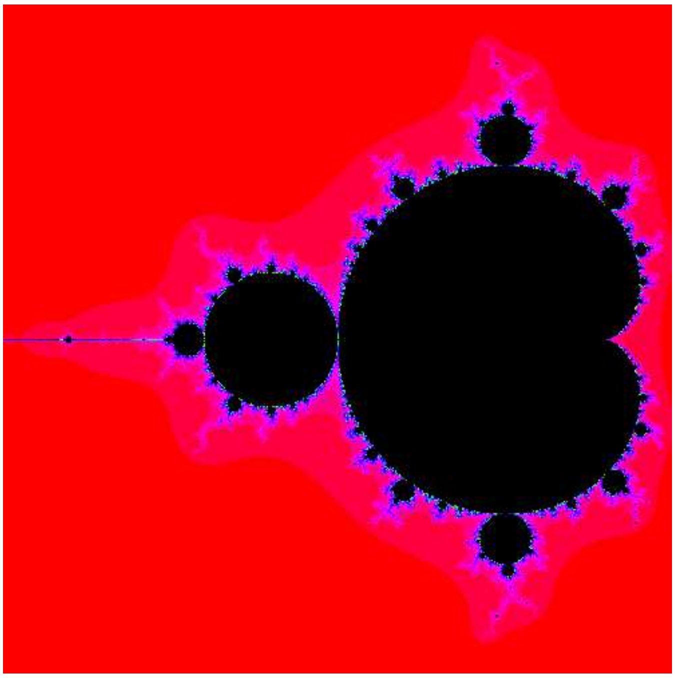

Fig. 1. A Mandelbrot Set

part of the complex plane. Figure 1 gives an example of a Mandelbrot set.

The Mandelbrot set problem belongs to the embarrassingly parallel class of problems [11] and gives an indication of the performance of the libraries under near ideal conditions. The problem is parallelised using data partitioning. The image is divided into sections, and a node calculates a value for the colour of each pixel in its section. The time taken to calculate each section differs. Points lying inside the set require iterating the function one thousand times, while points outside the set may require as little as one iteration. Therefore a work pool is used. The visible part of the complex plane is divided into at least twice as many sections as there are processors. Each processor is allocated an initial section. As each process completes, its section is returned to the primary process, and another section is assigned.

C. Neural Network

A neural network is an attempt to replicate the way a biological brain works. Because it simulates the brain, it is able to solve some problems which humans do well, but computers perform poorly. Examples include pattern recognition and motor control [12]. A neural network is represented as a weighted, directed graph. Weights associated with the edges of the graph represent the synaptic weights of the brain and are responsible for storing knowledge. Vertexes in the graph represent the neurons of the brain and are processing elements in the network. The input to a neuron is the sum of the inputs of the edges leading to the neuron. The input is added to a bias for the neuron and passed through an activation function to determine the output of the neuron.

One class of neural network is the multilayer perceptron or back propagation network. The neurons are organised into at least three layers such that connections only exist between adjacent layers. The first layer represents inputs to the network. The last layer represents the output. Intermediate layers are known as hidden layers. Input is presented to the input layer, passes through the intermediate layers, and emerges at the output layer.

The network is trained by a two pass algorithm. In the first pass, an input pattern is propagated forward through the network using a sigmoidal activation function. In the second pass, weights associated with the neurons are updated by calculating an error between the actual output and the expected output. The error is then propagated backward through the network to modify the synaptic weights in the hidden layers. This is repeated for each pattern in the training set.

The presentation of the complete set of training patterns is known as an epoch. The neural network is trained for a number of epochs until the error measure, the sum of the squares of the difference between the expected and actual outputs, is lower than a specified value. When batch training the network instead of applying the update directly to the network, we sum the changes in a weight change matrix and apply the summed changes at the end of the epoch.

The most efficient way to parallelise a neural network is to use data partitioning [13]. Each processor trains the network on a part of the training set, the changes are combined, and applied to the network at the end of each training epoch. A neural network belongs to the synchronous class of problems. The same operation is carried out for each pattern in the training set. Synchronisation occurs at the end of each training epoch. Rogers and Skillicorn in [13] note that data partitioning provides the best results with large data sets.

D. Summary

The applications created represent the three primary problem classes described by [6]. They represent the majority of problems that are typically capable of being parallelised.

Mergesort is a loosely synchronous problem. Loosely synchronous problems are similar to synchronous problems but with coarser grained synchronisation. The algorithms used may be temporally different leading to irregular synchronisation points. Loosely synchronous applications are characterised by alternating phases of computation and global communication.

The Mandelbrot Set generator represents the embarrassingly parallel class of applications. Embarrassingly parallel problems are the ideal sort of problem to parallelise. While they may involve complex computation they can be partitioned with very little or no communication between sections.

The neural network program represents synchronous class of applications. Synchronous applications are problems characterised by an algorithm that carries out the same operation on all points in the data set. Processes in a synchronous application are synchronised at regular points. Synchronous computation often applies to problems that are partitioned using data partitioning. [6] found that 70 percent of the first set of applications they studied belonged to the synchronous class of applications.

IV. Performance Tests

The performance tests are made on a cluster of 32 Intel Pentium III (Coppermine) machines with a clock speed of 800 MHz and 128MB or 256MB of memory. Results are determined for 1, 2, 4, 8, 16, 24, and 32 machines in the cluster to demonstrate the scalability of the programs. The operating system is Red Hat Linux 7.2. The machines are interconnected with standard 100 Mbps Ethernet. All applications are written in C and compiled with gcc-2.96 using optimisation flag -O2.

All experiments are carried out while the cluster and network are not being used for other purposes. The run-times of the applications are determined by the gettimeofday() function. Total run-time is the difference between the start and finish times of the applications.

The results reported are the time taken to complete the core algorithm only. The time required for initialisation of processes is not considered. This is done for two reasons. First, the method of initialisation is different for each library so the time cannot be calculated for all of them. Second, the applications being tested have relatively short run-times compared to more complex applications which may run for days or weeks. In the case of long running applications, initialisation time becomes negligible. Including initialisation time may lead to a distortion of results for a library which has slower initialisation.

V. Results

This section compares the performance of each of the libraries for the test applications and discusses the results.

A. MergeSort

The result of parallelising mergesort was poor for the three libraries. The complexity of this problem is $O(n \log n)$ which is similar to the complexity of communications. This causes communications between nodes to override the increased performance of adding additional nodes.

Figure 2 shows the speedup achieved when sorting five million integers, and Figure 3 shows the speedup achieved for ten million integers. We see a small speedup for all three libraries with two nodes when sorting five million integers, however this quickly falls off with the performance for DSM falling more rapidly. When sorting ten million integers only MPI shows any improvement as nodes are added. While PVM maintains a constant performance, the performance of the DSM program degrades rapidly. This is due primarily to virtual memory paging.

Treadmarks generates a copy of modified pages to allow it to create a page differential. The differential is then run-length encoded to compress the data before transmission over the network. Thus Treadmarks maintains two copies of modified memory on a given node. Since a sorting algorithm tends to modify much of the data allocated to a node, a large number of page differentials are created. Such action creates a large memory requirement causing Treadmarks to use all the available memory faster than the message passing libraries.

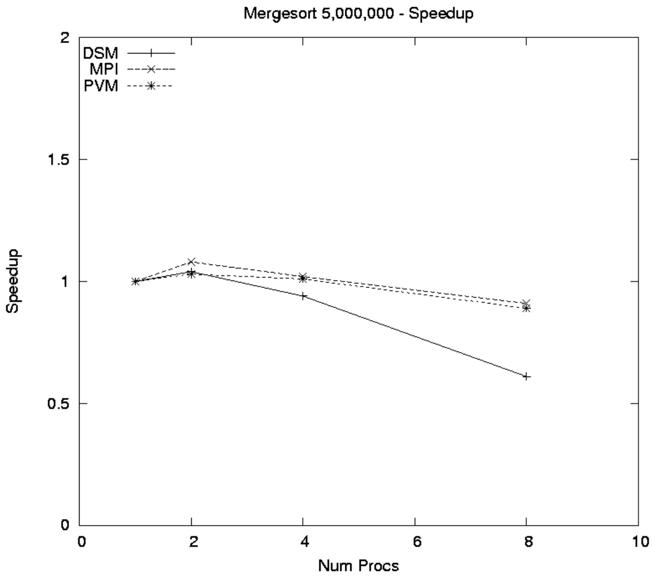

Fig. 2. Speedup for mergesort for five million integers

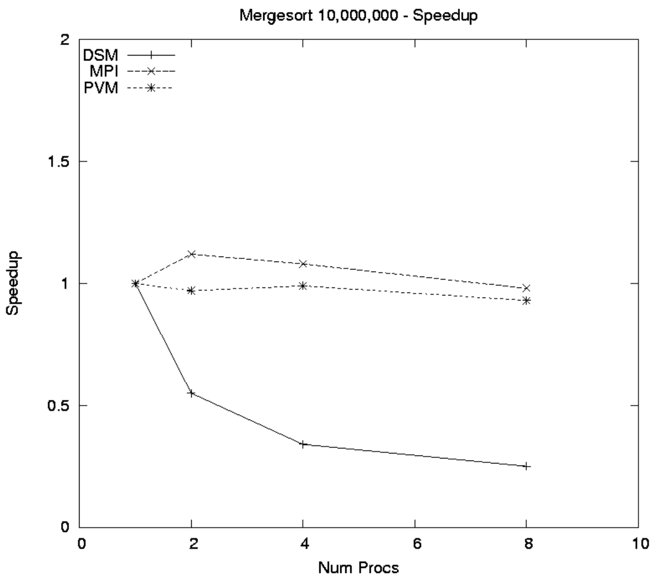

Fig. 3. Speedup for mergesort for ten million integers

Furthermore a DSM program will allocate the total amount of memory required on every node, even if it only accesses a subset of that memory. In comparison, the message passing libraries allow us to manage more carefully memory allocation. On each node, we only need to allocate enough memory for the largest subset of the dataset that the node will be required to handle.

Increasing the amount of available memory improves the performance of the Treadmarks mergesort program. Figure 4 shows the times for the DSM and MPI programs running on

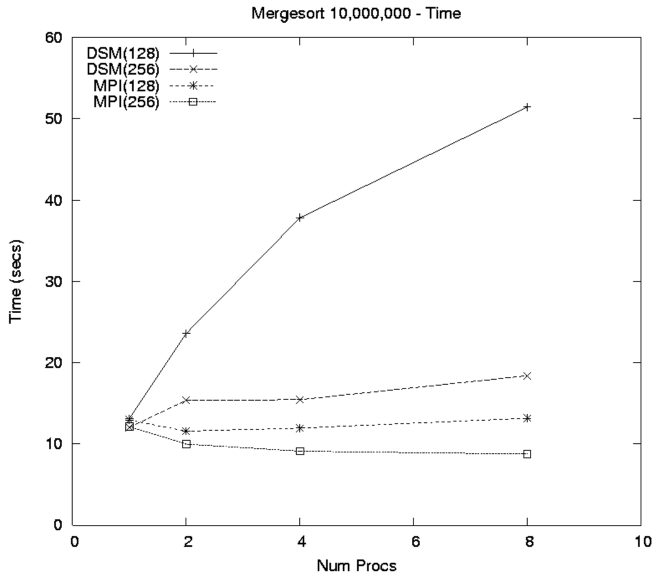

Fig. 4. Comparison of performance of DSM with 128MB and 256MB of memory

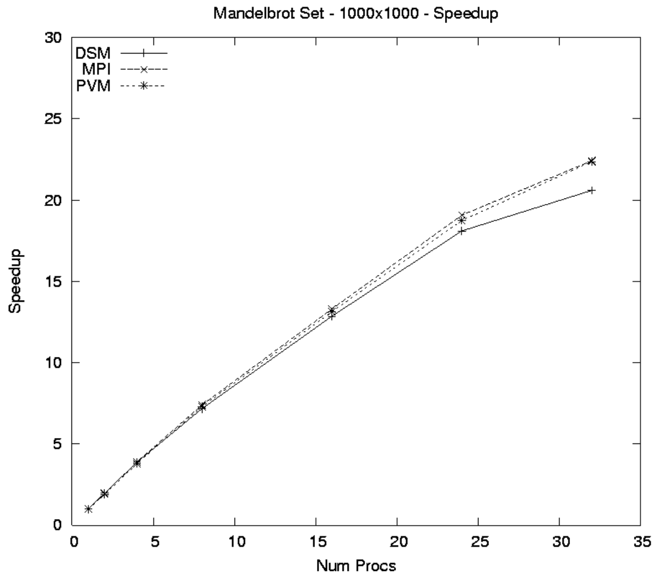

Fig. 6. Speedup for Mandelbrot set for 1000x1000 image

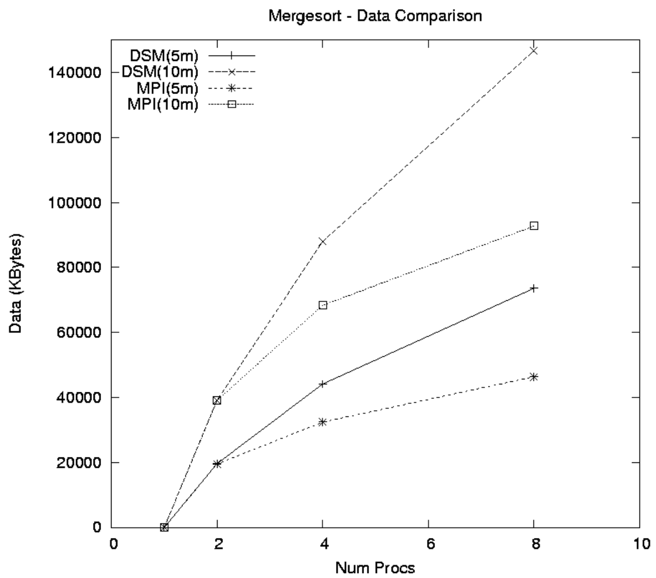

Fig. 5. Data sent by mergesort with five/ten million integers

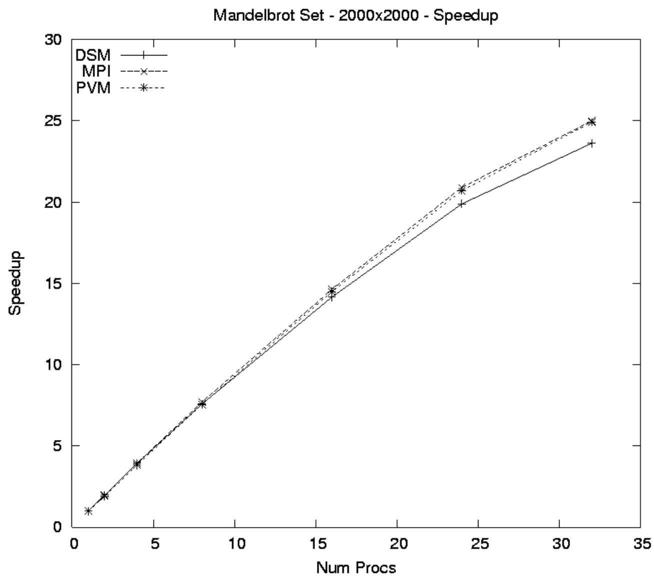

Fig. 7. Speedup for Mandelbrot set for 2000x2000 image

the cluster with 128MB and 256MB of memory. While the MPI program only shows a small increase in performance, the DSM program shows a major improvement.

However as the number of nodes increases, the DSM program still performs poorly compared to the message passing programs. Profiling of the libraries shows that the DSM program generates significantly more network traffic than the message passing libraries. Figure 5 shows the amount of traffic generated by all nodes involved in the computation for the DSM and MPI programs. While both programs send a similar amount of data when run on two nodes, the DSM program sends approximately sixty percent more data when run on eight nodes. The cause of the extra data is the false sharing effect whereby logically different data items are shared because they reside on the same physical memory page.

B. Mandelbrot Set

Figures 6 and 7 show the speedup of the Mandelbrot set for a 1000x1000 pixel and a 2000x2000 pixel image, respectively.

All three libraries performed similarly, with a greater speedup with the 2000x2000 pixel images. This is expected as parallel programs generally perform better on larger data sets. The speedup scales almost linearly up to 24 processors.

After that point, adding additional machines has a decreasing performance gain. The near linear speedup reflects the embarrassingly parallel nature of this problem.

MPI has the best overall performance, with the fastest times, and the best speedup. While PVM has a better speedup than DSM, and similar to MPI, for 24 and 32 nodes both DSM and PVM run in a similar time. The increase in speedup for PVM is due to a slower time for a small number of nodes. This appears to be due to the overhead in the use of the "virtual machine" which is most noticeable with a small number of nodes.

The Treadmarks DSM program consistently generates approximately twice as much network traffic as the message passing programs. Thus it avoids the accelerating increase in data sent that is the main cause of the poor performance, with a high number of nodes, of the mergesort and neural network programs.

C. Neural Network

Testing of the neural network application was done using two data sets obtained from the UCI (University of California, Irvine) machine learning repository [14]. The first is the shuttle data set drawn from sensor data recorded during NASA space shuttle missions. Each item contains nine numerical attributes, with 44,000 items in the set. The neural network is a three layer 9x40x1 network. The total number of epochs taken to train the networks for the tests is 235.

The second training set is the forest cover data set. This data set contains 54 environmental attributes. The neural network is required to determine the type of forest cover. The data set contains a total of 581,012 items from which a random subset of 72,000 items is used. The neural network is a three layer 54x160x1 network. It takes 3542 epochs to train this data set resulting in a run time of approximately 25 hours for one processor. To simplify our tests, the network was trained for only 50 epochs.

Figures 8 and 9 show the speedup of the neural network with the shuttle and forest data sets respectively. With the shuttle data set, both the message passing libraries perform well, with the speedup scaling almost linearly up to sixteen nodes before falling off to achieve a speedup of approximately 21 and 23 with 32 nodes for PVM and MPI respectively. DSM performs well up to eight nodes after which its speedup drops to less than one.

With the forest data set, MPI provides the best performance with a speedup of 21 on 32 nodes. PVM performs similarly to MPI up to 24 nodes before slowing down a little on 32 nodes. DSM performs comparatively better with the forest data set obtaining a reasonable speedup with up to 16 nodes then reducing to a speedup of three with 32 nodes.

The slowdown of the DSM program is primarily due to the increase in network traffic which saturates the network. As the number of nodes is increased the volume of network traffic required also increases. The total network traffic of the MPI and DSM neural networks is shown in Figures 10 and 11. In the message passing libraries, where we can explicitly control the amount of data sent, this increase is linear. In a DSM environment we do not have direct control of the data being sent. Since the system must determine which data to send, extra messages can be generated. This leads to an exponential increase in the amount of data being transmitted for the neural network application.

The major cause of the extra traffic is due to the poor performance of a gather style operation. The changes that need to be applied to the neural network calculated at each node are stored locally while processing the epoch, and added to a global weight change matrix at the end of the epoch. One process then applies the summed weight change to the weight

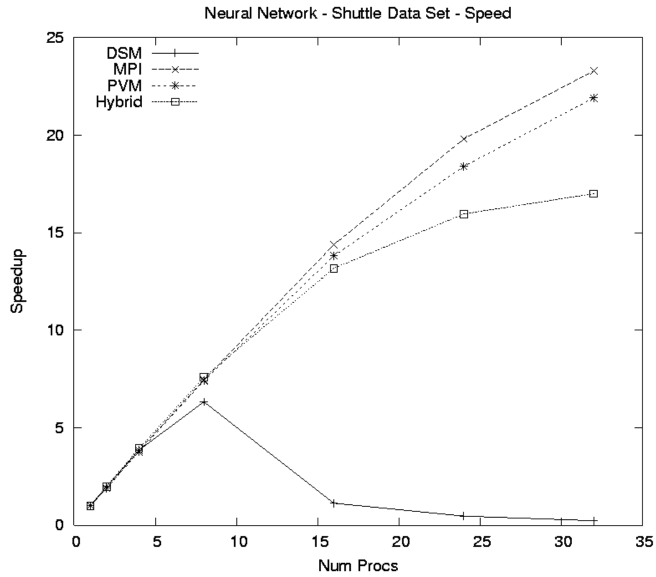

Fig. 8. Speedup for Shuttle Data Set

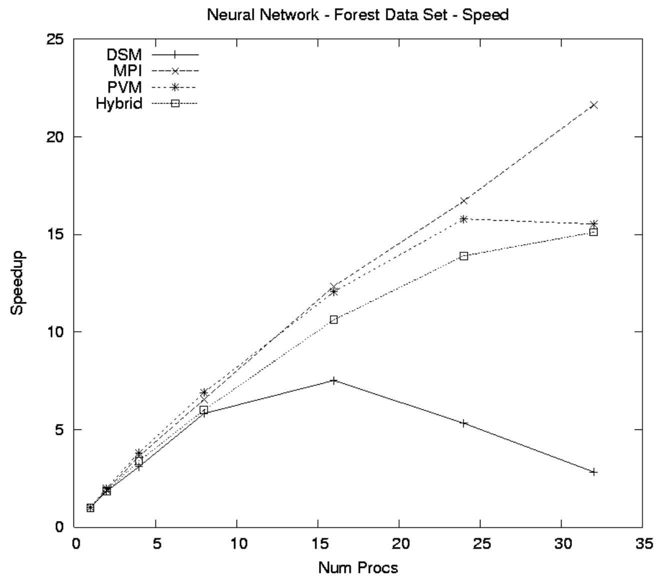

Fig. 9. Speedup for Forest Cover Data Set

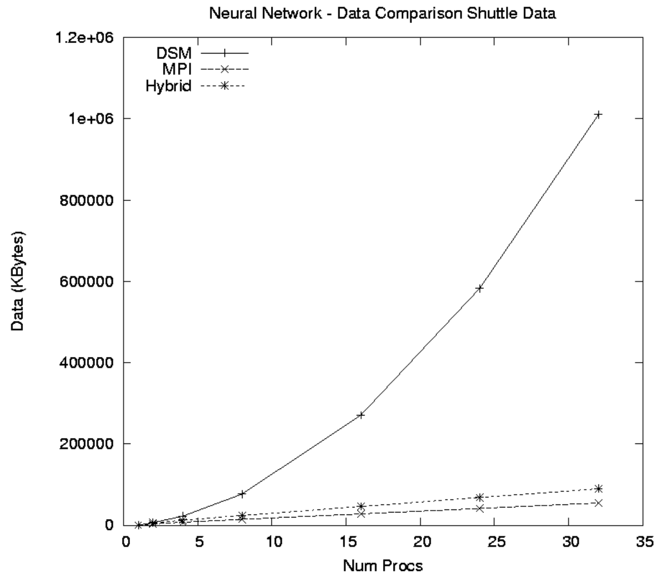

Fig. 10. Data sent by neural network with shuttle data set

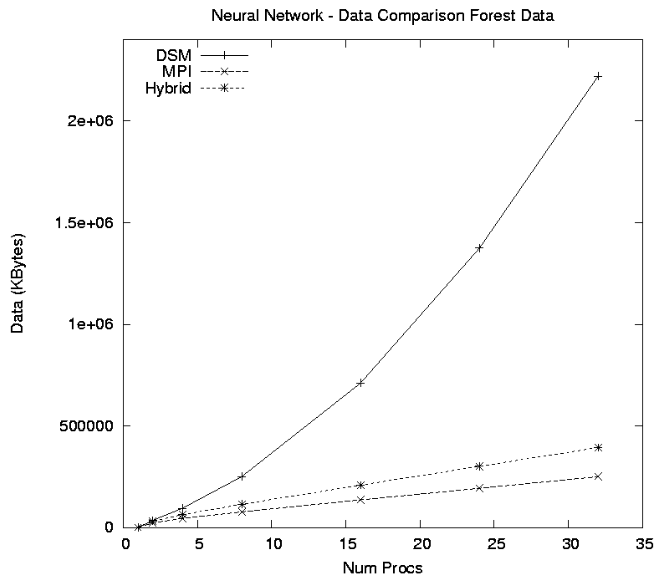

Fig. 11. Data sent by neural network with forest data set

matrix. This is done to avoid the high level of synchronisation required if each node updates the global weight change matrix for each input pattern.

This results in a gather operation in the message passing libraries where the process which applies the changes gathers the partial changes calculated at each node (processor). In the DSM program this is achieved by each node updating the global weight change matrix. For DSM, this causes the occurrence of the differential accumulation problem described by [2]. Each node typically modifies all the values in the weight change matrix which spans multiple memory pages. Thus the differential generated for each of those pages by each node has nearly the same size as that of the page, and the number of differentials generated for each page equals to the number of processors. Consequently, the data injected into the network traffic is N times as much as MPI at this stage, where N is the number of nodes (processors) involved. Therefore, the accumulation of these differentials generates a significant amount of extra traffic especially with a high number of nodes. This problem is exacerbated from 16 nodes as the size of the accumulated differentials exceeds the size of a UDP packet requiring the message to be split into multiple packets.

Treadmarks also generates a large number of messages compared to the message passing libraries. This impacts performance due to the latency associated with each message. The cause is that Treadmarks generates a new message for each page that needs to be sent. For the forest data set, the size of the weight change matrix is approximately 72 kilobytes, while a page size is 4096 bytes. Thus updating the weight change matrix requires approximately eighteen messages to achieve what the message passing libraries achieve in a single message.

This theory was tested by creating a hybrid version of the DSM neural network program which used PVM to send directly the partial weight change from each node to the process which applies the changes. The speedups for this program are shown in Figures 8 and 9. The relevant data is designated as Hybrid in those figures. The network traffic is shown in Figures 10 and 11.

The hybrid version obtains a speedup through to 32 nodes for both tests. However the performance beyond eight nodes for the shuttle data set, and sixteen nodes for the forest data set is less than the message passing libraries. With 32 nodes the performance is almost as good as PVM for the forest data set as PVM begins to slow down. The cause for the increased performance is the reduction in the amount of network traffic. Instead of the exponential increase seen for the DSM version, the hybrid version maintains a linear increase in the amount of traffic, at approximately 30-60 percent greater than the MPI version.

The cause of the reduced performance of the PVM version of the neural network compared to the MPI version is due to the poor performance of the broadcast operation with PVM. Figure 12 shows the broadcast performance of both libraries with message sizes of 16 kilobytes and 1024 kilobytes. These represent the size of a message containing the weight matrix, and one containing a section of the dataset respectively. In both cases the broadcast time for PVM is significantly longer than the time for MPI, with PVM taking 20 and 10 times longer to broadcast the small and large messages respectively, to 32 nodes. (Note: In Figure 12, MPI(16KB) is adjacent to the horizontal axis.)

VI. Conclusion

The performance of the Treadmarks DSM system was compared to the PVM and MPI message passing systems. Applications were chosen to represent three classes of problems

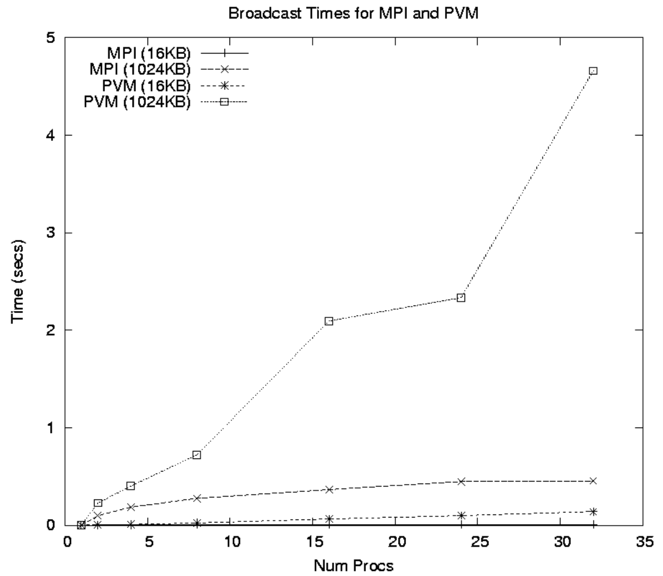

Fig. 12. Broadcast Performance of MPI and PVM (Note: MPI (16KB) is along the horizontal axis)

commonly run on distributed computing systems. Mergesort represented the loosely synchronous class. A Mandelbrot set represented the embarrassingly parallel class, while a neural network was used for the synchronous class.

The performance of DSM is poorer than PVM and MPI with mergesort especially as the number of nodes increases. This is due to virtual memory paging, and DSM performance is improved by increasing the amount of memory. With the Mandelbrot set, the performance of DSM is almost as good as PVM and MPI.

For the neural network, DSM shows an improvement as nodes are added. However after a point, adding nodes reduces performance. This is the result of a large increase in network traffic caused by two factors. First, there is false sharing since Treadmarks shares memory based on entire memory pages. The second reason is the gather operation involved in updating the weight matrix.

References

[1] R. Brown, "Engineering a Beowulf-style computer cluster," January 2002, http://www.phy.duke.edu/brahma/beowulf_online_book/beowulf_book.html.

[2] H. Lu, S. Dwarkadas, A. Cox, and W. Zwaenepoel, "Quantifying the performance differences between pvm and treadmarks," *Parallel and Distributed Computation*, vol. 43, no. 2, pp. 65–78, June 1997.

[3] W. Jiang, A. Beguelin, J. Dongarra, A. Geist, R. Mancheck, and V. Sunderam, *PVM 3 Users Guide and Reference Manual*, Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA, 1994.

[4] W. Gropp, E. Lusk, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, pp. 789–828, 1996.

[5] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "Treadmarks: Shared memory computing on networks of workstations," *IEEE Computer*, vol. 29, no. 2, pp. 18–28, 1996.

[6] G. Fox, R. Williams, and P. Messina, *Parallel Computing Works*. San Francisco: Morgan Kaufmann, 1994.

[7] *Concurrent Programming with Treadmarks*, ParallelTools L.L.C, 1996.

[8] Z. Huang, C. Sun, M. Purvis, and S. Cranefield, "A view-based consistency model based on transparent data selection in distributed shared memory," *Operating Systems Review*, vol. 35, no. 2, pp. 51–60, April 2001.

[9] G. Geist, J. Kohl, and P. Papadopoulos, "PVM and MPI: A comparison of features," US Department of Energy, Tech. Rep., 1996.

[10] A. Dewdney, "Computer recreations," *Scientific American*, pp. 16–24, 1985.

[11] B. Wilkinson and M. Allen, *Parallel Programming*, 1st ed. New Jersey: Prentice Hall, 1999.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New Jersey: Prentice Hall, 1994.

[13] R. Rogers and D. Skillicorn, "Strategies for parallelizing supervised and unsupervised learning in artificial neural networks using the BSP cost model," Queens University, Kingston, Ontario, Tech. Rep., 1997.

[14] C. Blake and C. Merz, "UCI repository of machine learning databases," 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.