Volume 3, Issue 4, April 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Junk Computing: Performance Evaluation and Comparison of an MPI based Heterogeneous Cluster

Aman Madaan
UG Research scholar
CSE, Bharati Vidyapeeth's College of Engineering
New Delhi, India.

Dr. Sunil K. Singh
Associate Professor
CSE, Bharati Vidyapeeth's College of Engineering
New Delhi, India.

Ankur Aggarwal
UG Research scholar
CSE, Bharati Vidyapeeth's College of Engineering
New Delhi, India.

Abstract— This paper elucidates the implementation and performance evaluation of an "off the shelf" cluster. A distributed memory cluster was designed using "dumped" heterogeneous machines and the gain in performance was measured for various configurations. The idea has been coined as junk computing. Creating a faster computer utilizing existing computers is an idea that has been long studied and widely researched. We havei attempted to add a new dimension to this study by running several programs on our cluster made up of 3 ancient Pentiums and a young i3 processor, and comparing the performance gain achieved under various configurations. Of special interest is the case where the performance of the three-node cluster of Pentium 4 is compared with the individual performance of an i3-330M processor. The gain achieved for linear problem is more in the cluster made of two P4 processors than the gain achieved in a single i3 processor. It is also evident from the research results that two Pentium 4 processors can compete with younger generation processor like Intel i3-330M. The study also confirms the intuitive and known notions of the parallel computing, so we would also bring about some implications of the obvious. For example, it was noticed that distributing the computations over the cluster yields gain only after a certain threshold that is after the point when the communication costs become negligible as compared to computation cost. It was also noticed that we might not gain much by adding a relatively slower node to the cluster. Before concluding the paper, we examine a shortcoming of the cluster; the case of distributed memories whereby a performance hit of the factor of up to 20 was observed.

Keywords— Cluster, Intel(R) Pentium(R), E waste management, Amdahl's Law, mpih2.

I. INTRODUCTION

The mankind is hushing down one of the biggest problems of the 21st century, the electronic waste. Millions of computers are manufactured each day only to get outdated and abandoned ultimately. The motivation of this research came from a myriad of computers that were lying idle in the labs of our college. We were curious to know whether these machines are as dumb as they look, or do they have the flare left. Surely, we cannot expect any one of these Pentiums to fight the mighty Intel I-series generation processors, but we get interesting results when we combined computational power of old systems. In this research, we compared the performance of some of these abandoned machines and the results were surprising. More and more organizations nowadays prefer implementing their own cluster for several reasons. The intriguing question is weather to invest in new hardware or will the existing resources suffice. The results from this paper aim to answer some of these answers. Parallel computing is a form of computation in which a problem is divided into smaller ones, which are solved simultaneously. Broadly, there are 2 categories of parallelism: shared memory and distributed memory parallelism. *Shared memory parallelism* is exploited using the concept of threads. It is the ability for all processors to access all memory as global address space. Shared memory parallelism is generally used to solve sub tasks with the same task. *Distributed Memory Parallelism* is implemented by distributing the computations across different nodes (computers). The computations are distributed by means of a message-passing library. The implementation used in this paper is called *mpich2*; it is a widely portable and free implementation of the MPI standard. MPI provides portability and performance for complex applications on a variety of architectures. The approach is very similar to the divide and conquers algorithms. A large problem is broken down into pieces; each node is given a chunk of the piece to solve. The final solution is obtained by combining the individual results. A very common example is performing matrix operations quickly by distributing the parts of the matrix to the nodes to work on.

Difference between Reducing the complexity and Reducing the Running time

It is worth noticing that the performance gain is obtained because we execute more than one instruction in a single time unit. An algorithm that is O (n^2) will still run 10^4 slower if the input size if increased by a factor of 100. However, intuitive notions direct us to believe that an algorithm with a running time of T (n), when executed by P nodes working together, can be expected to finish in T (n)/P time under ideal conditions. Amdahl's law (Discussed below) provides a more formal treatment of this intuitive notion. As expected, the factors that prevent us from getting the ideal speed up are

1. Communication Delays.
2. There may be some portions of the algorithm that are inherently parallel, so there may be a fraction of code that has to be executed serially, thus keeping us apart from the ideal speedup.

Amdahl's Law

The famous Amdahl's law models the exact speedup for the problem in hand because of the fact that it estimates the expected speedup with fixed load. In other words, it helps in calculating the speed up we get by increasing the number of parallel units, keeping the problem size fixed. The law is concerned with the speedup achievable from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S. Amdahl's law states that the overall speedup of applying the improvement will be:

The speedup expression, $\mathbf{S_n = 1/ ((1-P)+P/S) \quad – (1)}$

Embarrassingly parallel problems

A special class of problems having value of P close to 1 is termed as embarrassingly parallel. Intuitively, it means that *whole* of the problem has been parallelized i.e. the problem can be divided into chunks that can be executed independent of each other. Moreover, solution of such chunks is sufficient for solving the problem in its entirety. There is no fraction of the code that needs to be executed in a serial manner. Usually the problems that can be divided into partitions of disjoint sub problems fall under this category. Clusters such as one implemented by us are good candidates for solving such problems, as the speedup observed can be tremendous.

From (1), it is clear that the maximum speedup is achieved when P=1. In such a case the speedup becomes equal to the number of parallel units i.e. S. It should be noted that such a gain is only theoretical and is not observed because of communication delays. However, one can expect values very close to S for such class of problems.

Junk Computing

Junk Computing is the practice of utilizing the abandoned hardware, which is outdated, in the construction of a newly made system. The performance gained by using the cluster of existing Pentium 4 processors has been evaluated against the new generation processors like Intel's i3 processor for various algorithms. The performance of the cluster is better as compared to the Intel's i3 processor for various configurations of clusters when the number of iterations in the algorithm is sufficiently high to overcome the communication delay.

II. TESTING ENVIRONMENT

Node configuration

There are 3 similar nodes

Nodes (Node 1, Node 2, Node3)

Number of nodes	3
Processor	Intel® Pentium® 4
Speed	2.60GHz
RAM	512MB
Cache	512KB

Fox Node:

Number of nodes	1
Processor	Intel Core i3-330M
Speed	2.13GHz
RAM	3GB
Cache	3 Level (64,256,302)

*All the nodes runs Ubuntu 12.04 Long term support version.

Network setup:

Nodes are interconnected via an unmanaged fast Ethernet switch (10 MB/s). The network configuration thus is essentially meshed. Some other details of the switch are as follows:

Protocols Supported	IEEE 802.3, IEEE 802.3u, ANSI/IEEE 802.3, IEEE 802.3x
Forward rate	10 Mbps port: 14,880 packets/sec, 100 Mbps Port: 148,800 packets/sec
MAC Address database	2000
Management Protocol	CSMA/CD

III. MESSAGE PASSING INTERFACE

Once we have a network setup and machines are able to ping each other, the next logical step before we can move on to solving problems using out cluster is to incorporate a mechanism that shall facilitate communication among the nodes. For this purpose, we use the popular mpich2 library, a free implementation of the Message Passing Interface Standard. The latest implementation (MPICH2) implements the MPI-2.2 standard.

Message passing interface requires us to write only a single program, which contains all the information that describes what all the processes that are cooperating towards a single goal have to do. An Mpi program has the following basic structure.

```
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv) {
  int myRank;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myRank);

  printf("Hello World from Node %d\n",myRank);

  MPI_Finalize();
}
```

It is worth noting that the same code is executed on every machine that is a part of the cluster. The programmer divides task among the processes by their rank.

For example,

```
if(myRank == 2)  {
//work to be done by machine ranked 2 to goes here
}
```

A communicator defines the set of processes that are working together for solving the problem. Each process is uniquely identified by a rank in such a Communicator. MPI_COMM_WORLD is the default communicator.

Single Program Multiple Data

One wonders where in the Flynn's classification this type of a setup lies. The MPI model, with same program running on all the machines, is said to run in Single Program, Multiple Data stream mode or SPMD [4]. SPMD usually refers to message passing programming on distributed memory computer architectures. A distributed memory computer consists of a collection of independent computers, called nodes. Each node starts its own program and communicates with other nodes by sending and receiving messages, calling send/receive routines for that purpose. [2] Covers MPI programming paradigm in sufficient details.

IV. TIME CAPTURE METHOD

MPI library calls were used to note the timings. The following template is used for the tests:

```
startwtime = MPI_Wtime();
Computations here
endwtime = MPI_Wtime();
wall clock time = endtime -starttime
```

It may be noted that this time is different from the CPU execution time, which is the time for which the CPU was dedicated to the process.

V. CREATING MPI ENVIRONMENT

The MPI cluster is based on the SPMD model. This requires that the same program be present on all the machines at the same time. We used a very simple way of achieving this, a network file system. A shared directory was created and it was mirrored through the nodes. The usernames were kept the same and the secure shell login was made password less to allow smooth execution of the Mpiexec command. The nodes to be involved in the computation were specified in a file called the machine file, which is a regular file containing the names of the hosts. The listing of hosts in this file affects the rank that is allotted to each node.

Observation And Analysis

1. Calculating the value of pi: Scatter And Reduce Model.

The program distributed the processing equally among all the participating nodes.
The outputs for various values of iterations are shown in figure 1 and figure 2.

Figure 1 shows the expected gain in performance with increasing number of nodes working together. The time taken with all nodes working together is least, as expected. The percentage gain is almost equal with every node added to the cluster. As the number of iterations is increased, the communication delay, a fixed constant, subdues and we see very sharp gains.

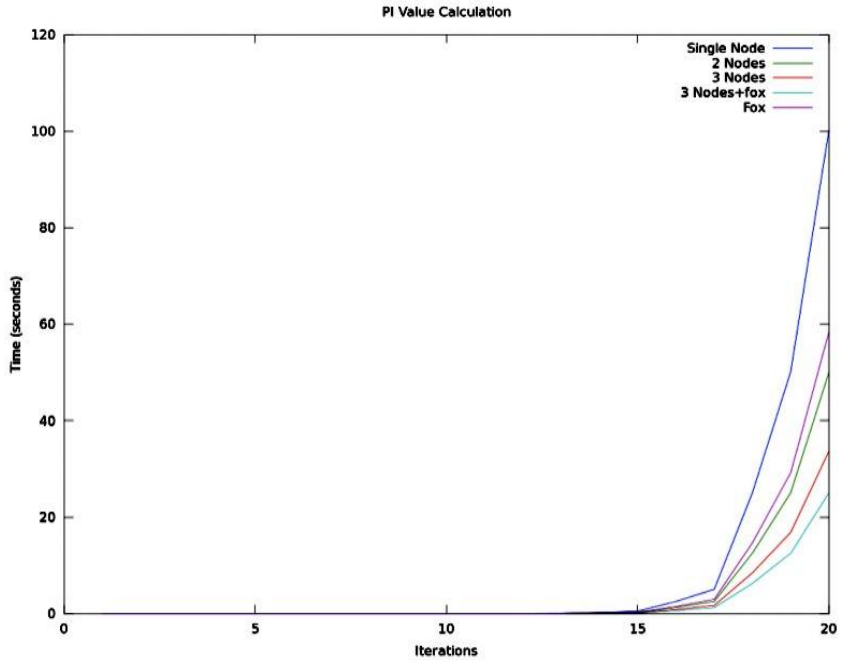

Figure 1:Time against number of iterations for different set of nodes combinations

TABLE I

Relative Speed up for pi value calculation.

Configuration	**Average Speed up factor**
Single Node	1
Two Nodes	2.0011
Three Nodes	2.9749
Fox	1.73143
Three Nodes + Fox	4.0017

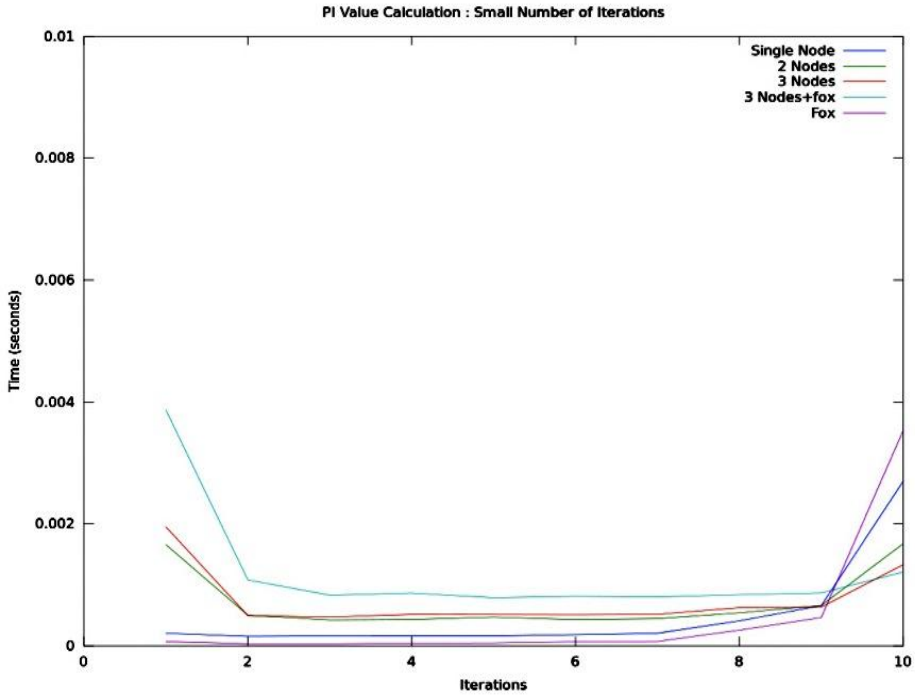

Figure 2:Time against comparatively less number of iterations for different set of nodes combinations

Figure 2 shows that there is ***negative gain*** when the numbers of iterations are small. The plausible reason for this is the communication overhead, a constant, which comes into play for dividing the problem. The single node gives the best result for solving small problems. Even if the other node (Fox) has more processing speed than the node working alone, there is no performance gain. Hence for small number of iterations the single Pentium 4 processor gives best results working alone. We have claimed that the communication delay is **constant** for this problem. This is due to the fact that the only the number of iterations is broad casted to the cluster and partial results from all the nodes are then collected. Thus, for a given number of nodes, the communication cost is independent of the number of iterations. It may also be noted that the gains achieved indicate that the problem is embarrassingly parallel.

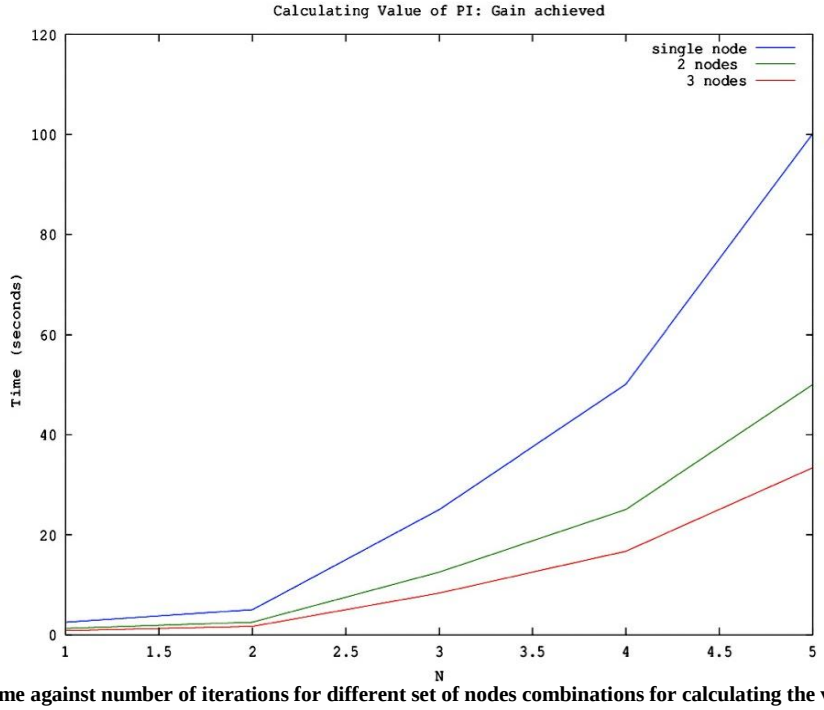

Figure 3:Time against number of iterations for different set of nodes combinations for calculating the value of pi

2.Calculating ∑N

Value calculation of sigma N is another problem that showed remarkable gains, the reason being minimal communication overhead and a very remarkable parallelism in the problem. N nodes are given a task of summing their share. Each node is assigned the task of summing an equivalence class of N mod k, where k is the number of nodes. Eg, for k =3, we have 3 equivalence classes, [0]={0,3,6...}, [1]={1,4,7...} and [2]={2,5,8...}.

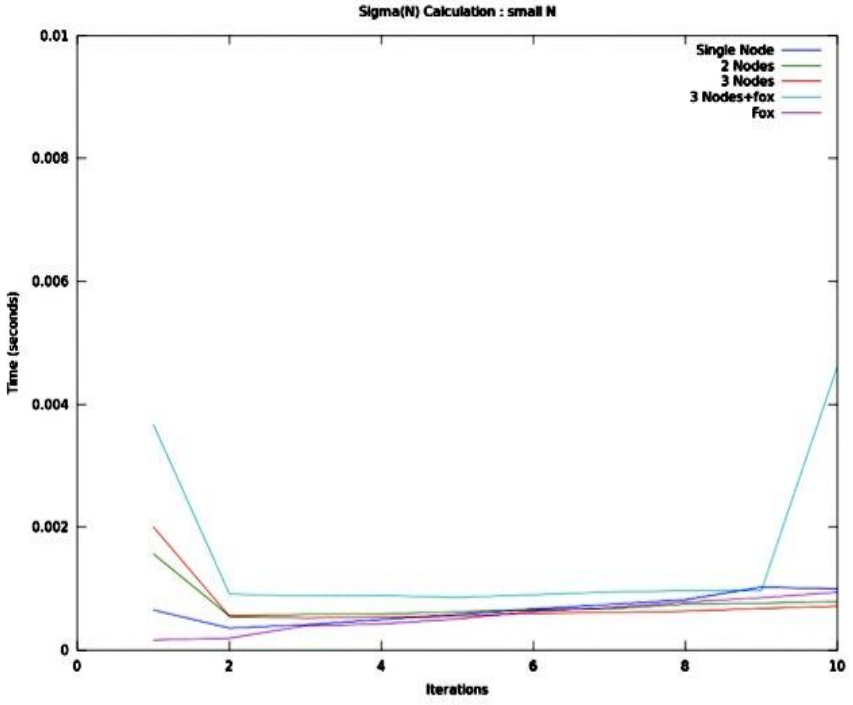

Figure 4:Time against number of iterations for different set of nodes combinations for calculating the ∑ N

The time taken for small number (. 3to4) of iterations to be solved for single Pentium 4 processor is less than the time with single i3 processor. It can be inferred from this search result that there is a possibility that a problem takes lesser amount of time to complete on older generation processor (Pentium 4) than the younger generation (i3). This is thus evident that the addition of new technologies in the i3 processor adds up the computation delay for certain size of problems.

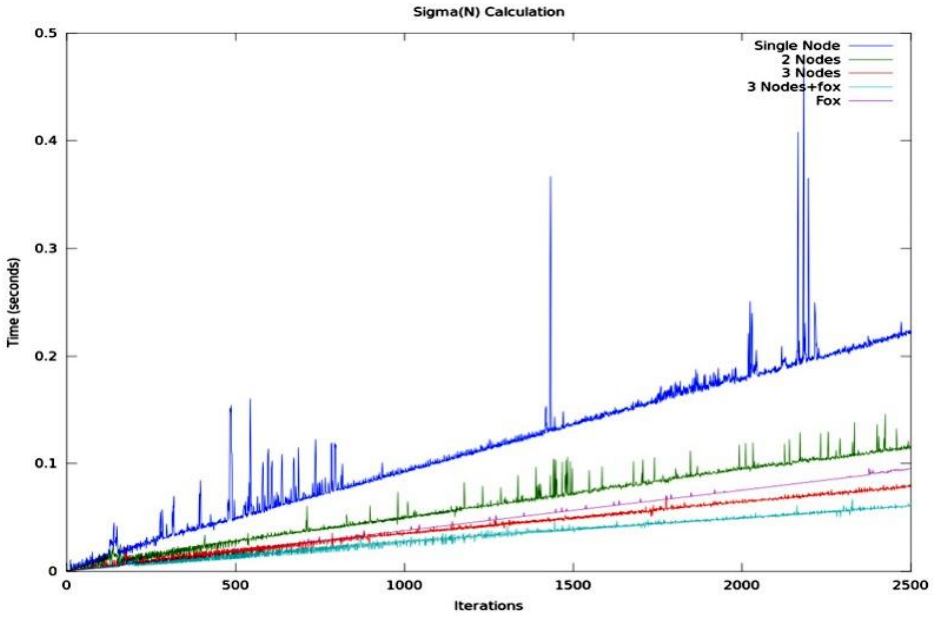

Figure 5:Time against moderate number of iterations for different set of nodes combinations for calculating the N

It can be inferred from the graph that for few hundred iterations to few thousands number of iterations, the graph shows the expected result. The execution time is decreasing as more nodes are added. The execution time decreases to almost its half with every node added. A closer look tells that we don't gain much by adding a third, slower node.

The occasional peaks in the graph demand some justification for some iterations because the Linux kernel employs a preemptive scheduler that takes decisions based on priorities. The user processes, such as ours is assigned a priority number from 100 to 139 and there might be cases when the process is the time taken may be exceptionally large due to the fact that the process might get preempted and before it gets the CPU again, another high priority process might sneak in the run queue, the data structure used by the kernel[11].

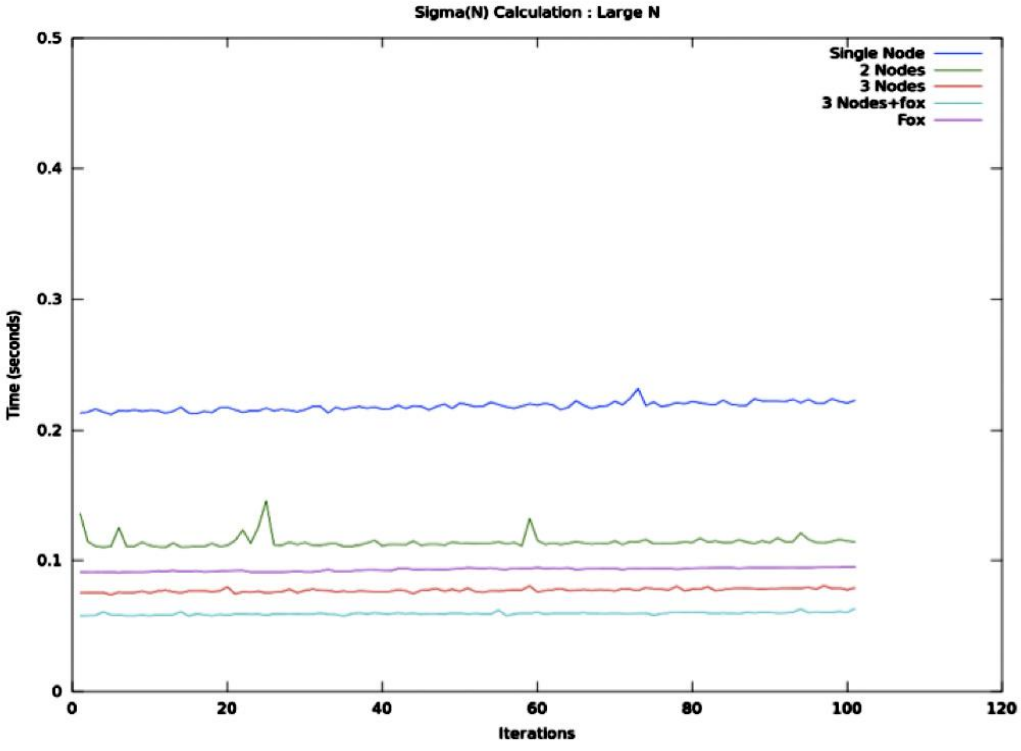

Figure 6: Time against large number of iterations for different set of node configurations for calculating N

The figure 6 shows the graph results for very large number of iterations where the lines seem to get parallel. The results at time number of iterations give more idea about the effect of increasing the number of nodes. The inference that can be drawn from this graph is that the gains start getting more explicit as the number of iterations is increased. We can clearly see that the time taken for 2 nodes working together is half of the time taken by a single node and so on.

TABLE II

Configuration	Average Speed Up Relative to Single Node
Single Node	1
Two Nodes	1.9028
Three nodes	2.7979
Fox	2.4731
Three Nodes + Fox	3.6212

3.Calculating the global maximum of an array: The curious case of distributed memories.

A program was made to calculate the maximum element in an array. Program divided the array into several nodes and calculating the local maximum. The results from each individual node were calculated and then the global maximum was calculated. Each node used its local memory to calculate the local maximum.

We used MPI_Scattter() to allocate to each of the nodes their share and then we used the MPI_Reduce() system call with the MPI_Max operation to get the global maximum. The entire code can be found at [12].

Distributed memories are usually very slow, with the order of performance hit coming out to be as much as 20 times. A similar kind of performance hit was observed with this program that involves a large amount of data interchange.

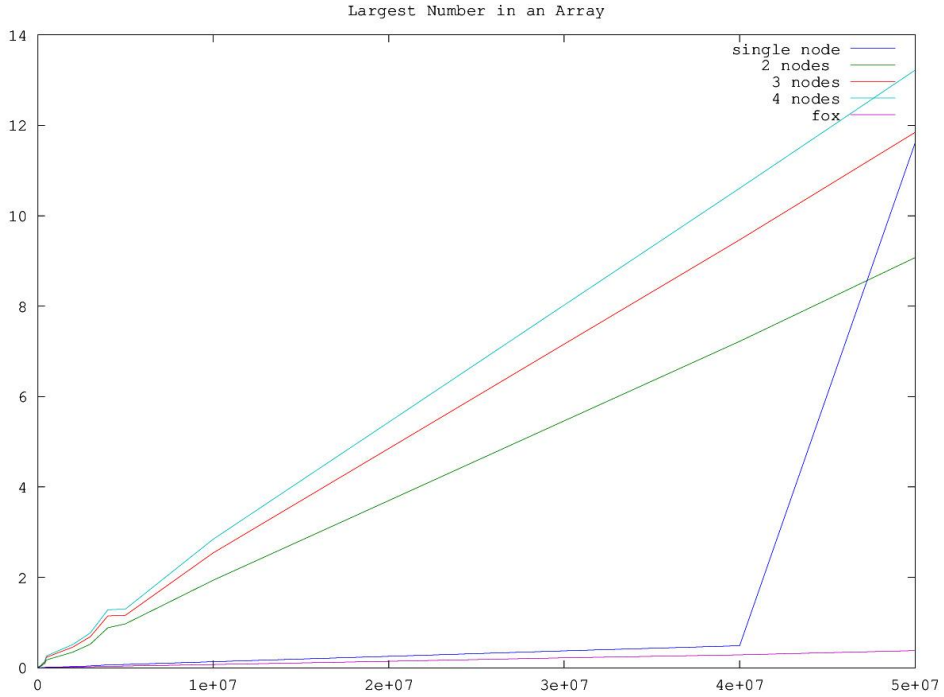

Figure 7: Programs involving heavy amount of data interchange suffer a performance hit: Distributed memories are slow.

Table III

Configuration	**Relative Speedup**
Single Node	1
Two Nodes	0.069 (1/16.66)
Three nodes	0.053 (1/ 18.868)
Fox	1.75
Three Nodes + Fox	.047 (1/21.2)

As can be seen from the table, with distributed memories, a hit of up to 21 times was experienced. Clearly, the implementation of such a system must move towards faster network devices for improving the speedup. We leave this as a future scope of research. The i3 processor again succeeded in reducing the running time to about a half.

VI. CONCLUSION

Parallel programming returns great results for embarrassingly parallel problems, however, as the amount of data to be exchanged increases, the communication latency eats up much of the gain. In some cases, the overhead may in fact be so large that we may suffer performance hit for small inputs. The dumped computers were really not worth dumping. When run together, they outperform modern machines by impressive factors. One of the issues that have to be considered is the power consumption. However, if that is not a problem, an organization should give a serious thought to utilizing existing machines along with fast networking software. Implementing a multicomputer and using it rather than a single machine will yield dazzling results for problems that are "embarrassingly parallel". Thus, for problems that have large parallelizable portions and minimum data exchange, a multicomputer system such as ours yield the best gains. The performance issues with distributed memories would be a bottleneck if the problem to be solved involves huge amounts of data flow among the nodes. The performance improvement that we will get for the distributed memories in our multicomputer (if anything at all) using faster network is a question still to be answered. We propose it as a scope for future research.

References

[1] Glenn R. Luecke, Marina Kraeva, Jing Yuan and Silvia Spanoyannis " Performance and scalability of MPI on PC clusters" 291 Durham Center, Iowa State University, Ames, IA 50011, U.S.A.

[2] Graham, R.L. Adv. Comput. Lab., Los Alamos Nat. Lab., NM Shipman, G.M. ; Barrett, B.W. ; Castain, R.H. ; Bosilca, G. ; Lumsdaine, A. "Open MPI: A High-Performance, Heterogeneous MPI", 25-28 Sept. 2006.

[3] http://www.mpi-forum.org/

[4]. Discussion on Parallel Computing By Ms Preeti At Indian Institute of Science, Bangalore on 12th July 2012.

[5]. The MPICH2 User's guide http://www.mcs.anl.gov/research/projects/mpich2/documentation/files/mpich2-1.4.1-userguide.pdf

[6]. The Wiki page on parallel computing http://en.wikipedia.org/wiki/Parallel_computing

[7]. Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, Katherine A. Yelick, titled "The Landscape of Parallel Computing Research: A View From Berkeley", December 18, 2006.

[8] Keqin Li, Yi Pan, Si-Qing Zheng, titled "Parallel Computing Using Optical Interconnections".

[9]David Patterson and a cast of thousands Pardee Professor of Computer Science, U.C. Berkeley Director, RAD Lab, U.C. Berkeley Past President, Association for Computing Machinery, titled "The Berkeley View: A New Framework & a New Platform for Parallel Research", November, 2006.

[10] Advanced Computer Architectures, Kai hwang, pg. 638-639.

[11] Modern Operating Systems, Andrew S tanenbaum, third reprint, ISBN-978-81-203-3904-0, page 752-755

[12] https://github.com/madaan/BVP-MPI-PROJECT

