Parallelization of Noise Reduction Algorithm for Seismic Data on a Beowulf Cluster

Izzatdin Aziz , Thayalan Sandran, Nazleeni Haron, Mohd Hilmi Hasan and Mazlina Mehat,

Universiti Teknologi PETRONAS, Tronoh, Perak, MALAYSIA

Summary
This paper presents the parallelization of a sequential noise reduction algorithm for seismic data processing into a parallel algorithm. The parallel algorithm was developed using C language with the utilization of the Message Passing Interface (MPI) library. The proposed algorithm has been implemented on an experimental Beowulf cluster which consists of 12 nodes operating on Linux Ubuntu platform. The system was tested with various test scenarios to gauge its performance. Based on the results obtained, it can be concluded that parallel implementation of the noise reduction algorithm has significantly reduced the processing time.

Key words:
Beowulf cluster, Fast Fourier Transform, F-k Filter, MPI, Parallel Programming

1. Introduction

The existence of noise or unwanted signal in seismic data has always posed a major challenge in the seismic data processing field. There are two stages to reduce noise in the seismic data which is during the seismic acquisition itself and the other is the post-acquisition. Eliminating noise after seismic acquisition often requires enormous computing power which translates to long processing duration with the conventional sequential algorithms. For the oil and gas industry, long duration means higher cost since it will lead to development delays. At the same time, improper suppression of noise would contribute towards misinterpretation of the data thus causing economic disaster to the industry concerned.

Even though the data processing field has discovered and continuously experimenting new noise filters to be used in the seismic processing, there seem to be one common problem which is the processing duration. The most common remedy for such problem is the usage of high performance mainframe or supercomputers. Acquiring such systems do not however justify the cost involved. The problem now is to efficiency suppress noise in seismic at shorter time without high financial investment.

According to [1], we must make sense of the recorded seismic 'squiggles' to produce the truest possible image of the Earth's sub-surface geologic structure. Reflected seismic response is a mixture of our output pulse, the effect of the Earth upon that pulse, and background noise, all convolved together. We must remove the output pulse and the noise to leave just the 'Earth model'. This is the role of seismic data processing, which requires accuracy, reliability, speed and substantial computing power. The advanced mathematical algorithms and complex geophysical processes applied to 3D seismic data require enormous computing resources. Not to mention the massive volumes of data involved. For example, the amount of seismic data recorded by CGGVeritas during just ONE medium-sized marine 3D survey would fill more than 20,000 compact disks, forming a stack over 650 feet high.

It has become very difficult for a processing facility to build around a serial architecture machine to cope up. Because serial computers have their physical limitations and they cannot go beyond certain speed, all over the world it has been physically realized that parallel processing is the only answer to this challenging application [2]. Serial computers present severe limitations in certain applications involving large data volumes. It simply takes too much time to cycle the data through a single central processor. If the application requires both a large data volume and a large amount of computation, serial computer limitations are especially acute [3].

The implementation of the converted sequential algorithm into parallel processing environment has foreseen to accelerate up the process mainly because of divide and conquer concept in parallel processing. This concept enables a large task to be further breakdown into smaller pieces which are going to be processed concurrently by multiple processors or nodes. There are a few types of parallel computers that can be choose from to perform the parallelization namely grids, clusters or massively parallel processors (MPPs). In this research, we have opted for Beowulf cluster which consists of commercial off-the-shelf computers that are linked by

Manuscript received January 5, 2010
Manuscript revised January 20, 2010

TCP/IP Ethernet local area networks under a single administration domain. The chosen parallel architecture offers a cost-effective approach due to xxxxxx

The aim of this work is to parallelize noise reduction algorithm and to present the implementation of the parallel algorithm using master/slave architecture and MPI on a Beowulf cluster.

2. Background of Study

2.1 Seismic Acquisition

Seismic acquisition process is basically consists of 2 crucial components which are namely the seismic source and the receivers known as geophones on land and hydrophones in marine. Figure 1 illustrates a marine seismic acquisition activity with a seismic ship propagating sound waves to the seabed while the streamer consisting of hydrophones records the reflected and refracted signals. In a typical acquisition activity as many as 6 sets of streamers could be attached to the vessel with each as long as 6 meters. The vessel contains 2 ultrasound generators which alternatively transmits sound waves at every designated position.

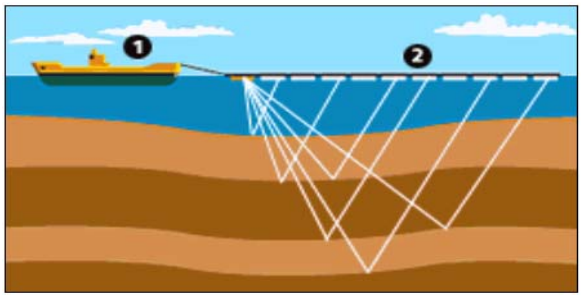

Fig. 1 Marine seismic acquisition [1]

During each of the sound wave transmission, the signal travels horizontally and also downwards. The vertical signals are reflected and refracted according to the various densities of the earth strata as illustrated in Figure 2.

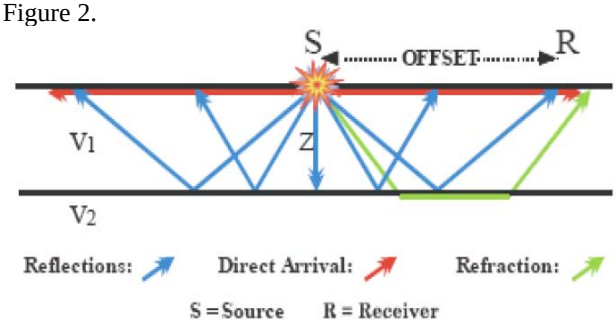

Fig. 2 Subsurface seismic characteristics [4]

The recorded signals are collected in the form of Shot Records as shown in Figure 3. It illustrates clearly that in the Time-Offset Domain the noise is overlapping the required signal thus, making noise separation more difficult.

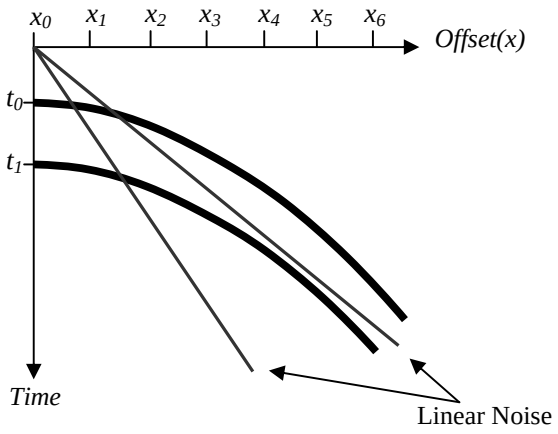

Fig. 3 Shot record [5]

2.2 Noise Reduction Algorithm

The noise reduction algorithm can be summarized in three steps as shown in Figure 4. The following sub-sections discuss the steps involved in the chosen algorithm.

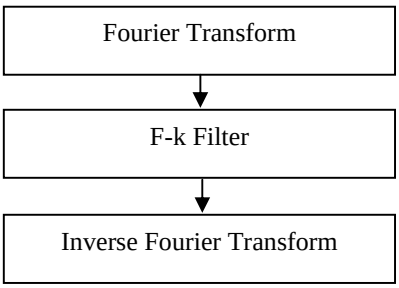

Fig. 4 Process flow of noise filter [5]

2.2.1 Fourier Transform

According to [6], the process of obtaining the spectrum of frequencies H(f) comprising a time-dependent signal h(t) is called Fourier Analysis and it is realized by the so-called Fourier Transform (FT). In most cases, when a signal is recorded it is displayed in the form of *x*-*y* axis whereby *x* represents the time while *y* the amplitude of the signal. This form of signal representation is known as the time domain spectrum. However, in order to filter the signal, it has to be converted into the frequency domain.

In the case of this research however, according to [5], a 2-Dimensional Fourier Transformation has to be performed. This means that the *time-space* graph needs to be converted to that of *frequency-wavenumber*. The above concept can be further analyzed with the Seismic Shot Record as shown in Figure 3. The offset(x) indicates the distance between the source and the receivers (hydrophones/geophones) with x_0 being the location of the source. The *y-axis* shows the time taken for the waves to travel from the source to the receivers.
Figure 3 can be represented by the following mathematical function:

$$S(t,x) \quad (1)$$

By applying Fourier Transform, Figure 3 would be converted into the following Figure 5:

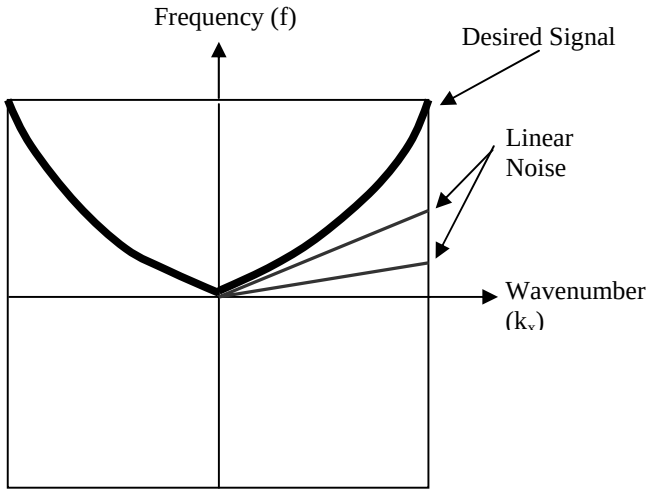

Fig. 5 Fourier transform of Seismic Shot Record [5]

Figure 5 can be represented by the following mathematical function:

$$S(f,k_x) \quad (2)$$

Therefore the conversion process can be mapped in the following mathematical model:

$$\mathbf{S(t,x)} \xrightarrow{FT} \mathbf{S(f,k_x)} \quad (3)$$

2.2.2 F-K Filter

With the Seismic Shot Record in the form of Fourier Transform or in the frequency domain, the noise can be easily identified and removed with the F-k filter.

2.2.3 Inverse Fourier Transform

Upon removal of noise, the data would be Inverse Fourier Transformed and the result would be as illustrated in Figure 6:

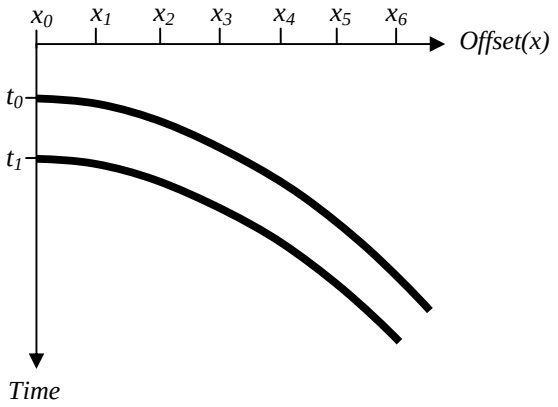

Fig. 6 Filtered inverse Fourier transform result [5]

2.3 Parallel Processing

Traditionally, parallel programs are designed using low-level message passing libraries, such as PVM or MPI. Message Passing (MP) provides the two key aspects of parallel programming: (1) synchronization of processes and (2) read/write access for each processor to the memory of all other processors [7]. The earliest (MP) library created for parallel processing is PVM or Parallel Virtual Machine. The main feature of this library is that it allowed portability. HPC from different vendors were able to be put to work together with the compatibility issues resolved. Later the MPI library was created. This API was developed for improved performance and was packed with richer communication functions. The primary goal of both MPs has always been to facilitate communication in the form of IO among the nodes in the cluster.

According to [8], massively parallel processing (MPP) vendors need to be able to deliver high performance which thus became a focus in the design of the MPI API. Given this design focus, MPI is expected to be always faster than Parallel Virtual Machine (PVM) on any parallel hosts. Therefore, MPI is used as the programming language in this research to create the parallel algorithm. Brief description of MPI and PVM libraries is given in Table 1 [9].

Table 1 : Brief description of MPI and PVM [9].

Libraries	Description	Remarks
MPI	A message passing interface, generally language independent even though binding are included in the standard for C and FORTRAN	One of the most popular libraries for cluster computing
PVM	Permits a heterogeneous collection of machines to be joined together to produce Parallel Virtual Machine	Trade some speed for the virtual machine ideal

According to [8], the main issue pertaining to parallelization is how efficient it is compared to the sequential system. If the overhead of parallel processing is larger than the processing time, then it is a clear indication that a parallel process is either not needed in that particular situation or it is not optimized to harness the full capability of the parallel architecture. This particular point places is vital to the load balancing aspect of the program. If the slave nodes are able to complete the process before the data is completely loaded into the neighboring slave nodes then the usage of parallel system in such scenario will not justify the performance.

3. Motivation

This section presents the scenario and the analysis that we have conducted in form of arithmetic cost that motivates our work to parallelize the noise reduction algorithm using cluster computing.

3.1 Arithmetic Cost for Transformation

In the effort to compute the arithmetic cost for transforming the input data into a via Fourier Transform, a total of 1000 Shot Record data with 6 sec long recording, 2 millisecond sampling rate, 480 traces per shot, 12.5 m trace spacing were envisioned. This step is crucial to determine the computation requirement for a particular scenario. It is important to note though that the actual scenario may vary since it is dependent on the nature of seismic acquisition. The seismic data would be F-K filtered on a 10 node High Performance Computer cluster. Each of the shot-records would be filtered on a single compute node. The following illustrates the computation requirements for the project case which is similarly represented in diagram form as in Figure 7.

1 shot record = 480 traces (receivers)
1 recording = 6 seconds long
Sampling rate = 500Hz (Every 2 milliseconds)
Trace spacing = 12.5 meters

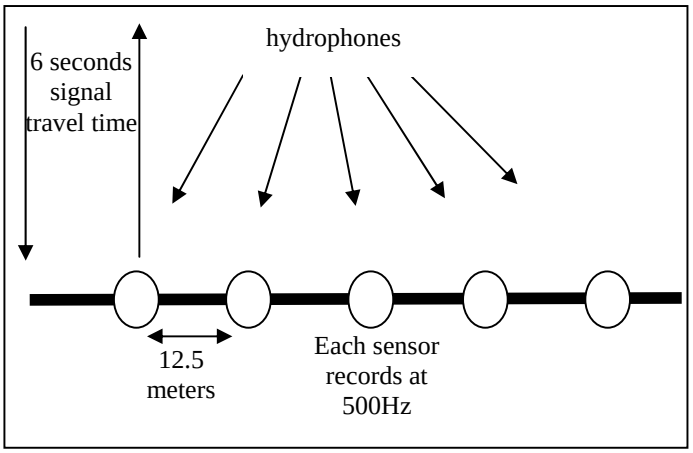

Fig. 7. Seismic Acquisition Streamers

Total samples at each hydrophone per shot record
6sec / (2 x 10^{-3}) = 3000 samples

Total samples from all hydrophones every 2millisec
480 samples

Total samples from all hydrophones per second
480 traces x 500Hz = 240 000samples

Total samples from all hydrophones per shot record
3000 samples x 480 traces = 1.44M samples

Based on the above parameters, each f the shot record can be presented by the matrix shown in Figure 8. The value ***i*** shows the number of channels or receivers used in the acquisition while the value ***j*** is the number of samples collected in each of the shot records.

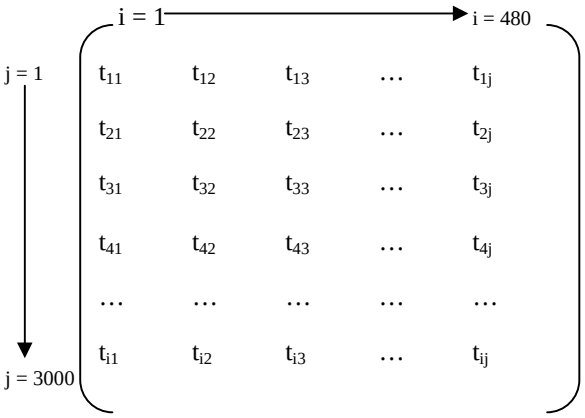

Fig. 8 Shot record in the form of matrix

1-D FFT 1-D FFT
D(i,j) => $D_1(i,\omega)$ => $D_2(k_x,\omega)$

Number of FLOPS to compute 1-D FFT (**D_1**) for a shot record:
$5Nlog_2N$ x 3000 samples = $5(480)\ log_2(480)$ x 3000
= 64.13 M Flops

The above calculation determines the computation requirement to do a 1-D FFT horizontally.

Number of FLOPS to compute the second phase of the 1-D FFT (**D_2**) for a shot record:
$5Nlog_2N$ x 480channels =$5(3000)\ log_2(3000)$ x 480
= 83.17 Mega Flops

The above calculation determines the computation requirement to do a 1-D FFT vertically. The combination of 1-D FFT horizontally and 1-D FFT vertically results in 2-D FFT as required in the algorithm.

Total flops to compute FFT for all samples in a shot record:
64.13 Mega Flops + 83.17 Mega Flops =147.3 Mega Flops

Upon completion of the filtering process, the shot records have to be reverted to their original dimension which is the time-space domain. This process is known as the inverse Fast Fourier Transform or abbreviated as inverse FFT. The number of floating point operation for inverse FFT is exactly the same as the FFT.

Total flops to compute the FFT and Inverse FFT for all samples in a shot record:
147.262 Mega Flops x 2 = 294.6 Mega Flops.

In the defined scenario, a total of 1000 shot records are provided as input data. If they are equally distributed to all the nodes (assuming all nodes have similar processing capacity), then each of the slave node would receive precisely 100 shot records. The total number of calculations that each node would have performed after filtering all the 100 shot records is as follows:
294.6 Mega Flops x 1000 shot records = 294.6 Giga Flops.

Good to note that in a typical seismic acquisition the total number of shot records collected is in the range of 200,000 to 500,000. Assuming close to half a million shot records are to be filtered the amount of computation required is as follows:

294.6 Mega Flops x 500,000 shot records = 147.3 Tera Flops.

The above computation strongly justifies the need of parallel processing in the quest to filter seismic data.

4. Proposed Approach

4.1 Program Work Flow

4.1.1 Existing Sequential Algorithm

The following pseudo code illustrates the conventional sequential noise filtering algorithm:

Start
Node stores multiple seismic shot records in the secondary memory.
Node loads a shot record into the primary memory.
Node processes the shot record
Converts shot record converted from time domain to frequency domain via Fast Fourier Transform
Filters the noise
Applies Inverse Fast Fourier Transform to the filtered data
Store result in the secondary memory
Node repeats the above process to fetch the following shot record.
Node then combines all the filtered outputs and produces the final results.
End

4.1.2 Proposed Parallel Algorithm

The following pseudocode illustrates the proposed parallel noise filtering algorithm:
Start
Master fetches of shot record input from the memory.
Master distributes the shot record one by one to each of the nodes.
Each slave will receive the shot record and begin processing it.
Each slave will convert the shot record from time domain to frequency domain via Fast Fourier Transform.
Each slave will then apply filter to truncate the noise form the shot record.
Each slave will perform inverse Fast Fourier Transform to revert the filtered data back to time domain.
Each slave will return the output to the Master.

Master receives the individual processed data and stores them.
Master fetched the next data in the memory to be distributed to the completed slave node.
Master repeats the above process till there is no more data in the memory.
Master then signals the slave of the completion and terminates the process.
Master displays the summary of the processes.

End

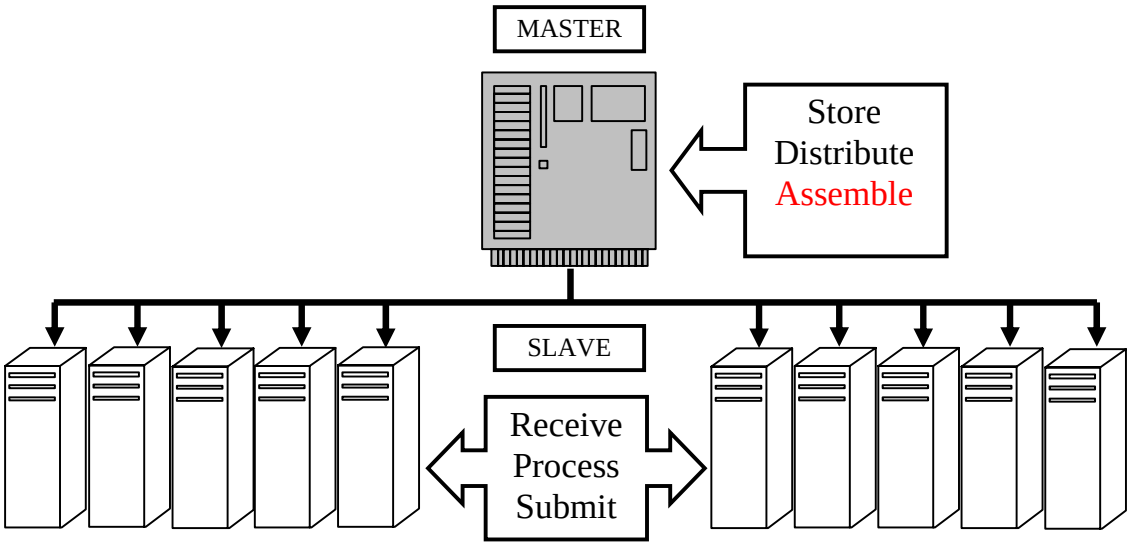

Fig. 9 Master-Slave Architecture

4.2 Master-Slave Architecture

The pseudocode mentioned in 4.1.2 was implemented on the Master-Slave architecture. In this setup, the master node acts as the coordinator in terms of load distribution to the other nodes and eventually gathers and stores all the processed data. The slave nodes primary task is to receive the input from the master node and execute the codes destined for the slave nodes. The illustration of the architecture is as shown in Figure 9.

5. Results and Discussion

5.1 Experimental Setup

The experiment was conducted on a Beowulf cluster that is consists of 12 SGI computers powered by the Intel i386 based dual processor Pentium 3 – 733MHz processors with 512MB memory. They are inter-connected via a Fast Ethernet 100Mbps switch. The cluster is operated on Linux Ubuntu 5.10 operating system, MPICH-1.2.7p1, parallel High Performance Linpack (HPL) version 1.0a and Flops.c version 2.0 both for parallel benchmark and individual node flops benchmark, GCC-3.3.6 with Basic Linear Algorithm Subroutine (BLAS) version 3.0 as the program compiler and its supporting math library [10].

As for the test data, since the seismic data exists in the SEG-Y format, pseudo data was generated. However, it is important to note that the seismic data no matter in what format it is captured can be basically disintegrated into a series of complex and imaginary mathematical numbers. It is on this basis that the pseudo test data resembles the actual data [5]. The variables in the testing process can be basically divided into 3 categories namely, the number of nodes used for the computation, the amount of data set, and lastly the size of each of the data set since the actual size of the seismic data can vary according to requirements. As for the nodes the test were conducted in groups of even quantity and the maximum number of slave nodes tested were 12 nodes due to the limitation of the system architecture. The data size is basically in the multiples of 10. The data generated were in the form of 2 dimensional arrays with the number of rows being the variable. The number of rows was varied from 2^{13} till 2^{18} which is also the threshold that the clusters could handle. The number of rows is also in the base 2 denomination to emulate the actual data. The test scenarios for each of the test cases are stated for each of the tests in the next section.

The tests were devised to illustrate the performance status and also the efficiency and speedup aspects in comparison of sequential process versus parallel execution.

5.2 Performance Test

5.2.1 Consistency Test

The consistency test was conducted to determine the consistency of the processing time for a repeated test. The following is the test case and the test result in the form of chart highlighted in Figure 10.

TEST CASE		
Data	:	1000 sets of data
Array Dimension :		[131072][2]
Number of Nodes	:	1 Master + 12

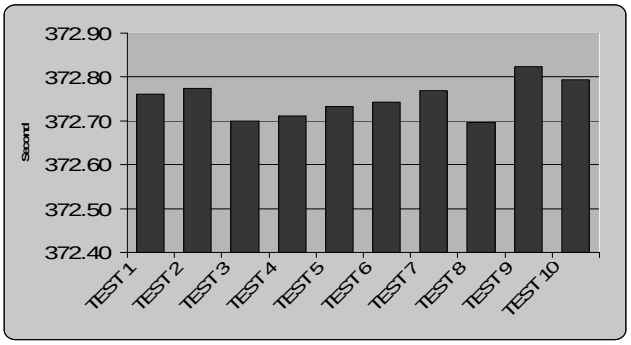

Fig. 10 Processing time versus repeated tests.

The consistency test result was expected to show a consistent reading given a same process repeated several times. From the test result illustrated in Figure 10, the computed standard deviation among all the 10 tests is 0.041876 which suggests that the average deviations among the tests are very minimal. This suggested that the test results are very consistent and therefore the result from this program is reliable.

5.2.2 Variable Data Amount

The performance test was conducted with 2 sets of data. The main objective of this test is to observe the pattern of performance gain by processing a total of 100 data in the first test and 1000 data in the second test while at the same time varying the number of nodes used. Two data sets were used as to check the consistency of the result and at the same time consolidate the conclusion from the test results.

TEST CASE		
Data	:	100 set, 1000 set
Array Dimension :		[131072][2]
Number of Nodes	:	1 Master + 12

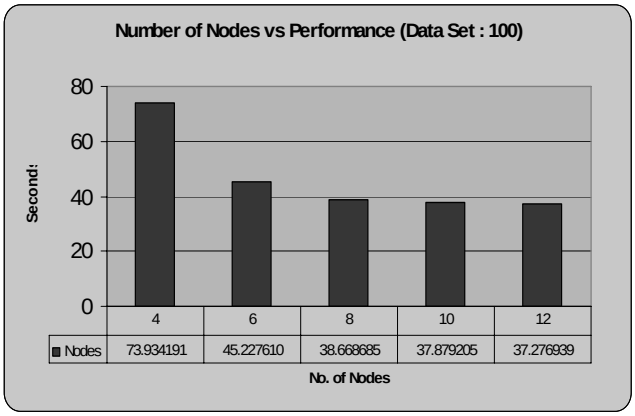

Fig. 11 Processing time versus number of processors for 100 input data

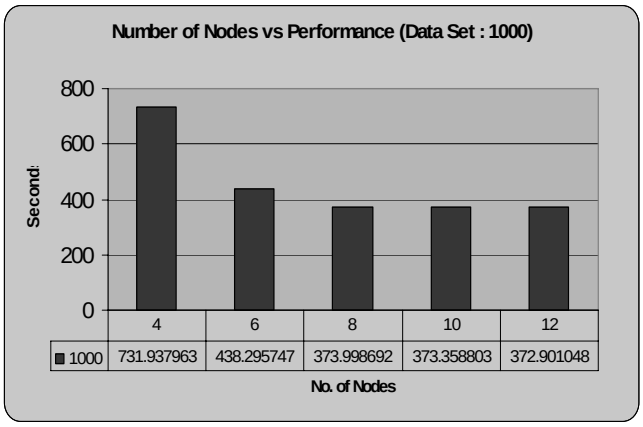

Fig. 12 Processing time versus number of processors for 1000 input data

Based on the result in Figure 11 and Figure 12, it can be concluded that by increasing the number of processors, the total processing time is reduced significantly. It is also important to note from both Figure 11 and Figure 12 that the performance improvement reaches a saturation point as the number of nodes is increases especially between 8 nodes and 12 nodes test. The underlying reason for such phenomena is basically due to the high communication overhead between master node and slave nodes when the number of nodes is increased and also the dynamic load balancing feature incorporated in the system. This is because when more nodes are involved, by the time the master fetches input data and communicates it to the next computer in the queue, the other nodes would have completed the processing task and waits for the master for the next input.

Since the data is dynamically distributed on a first come first serve basis and coupled by the communication overhead, some nodes will be processing less load then the others and significant performance improvement cannot be observed. However, for the same experimental setup as above, if the size of the data which is the array length were to be increased from 2^{17} to 2^{19}, we can expect a significant boost of performance by between 8 nodes and 12 nodes since now the communication cost is still higher but the complexity of each data has been also increased proportionally. The test was conducted for array sized at 2^{19} however due to system limitations, the cluster terminated and crashed.

5.2.3 Variable Length Array

The following test was conducted by varying the size of the input data which is in the form of two dimensional arrays. The objective of this test is to prove that the increment in the size of the array is directly proportional to the increase in the processing duration. To reinforce the point, the test was done on a set of 12 slave nodes and 6 slave nodes to illustrate the difference in duration when the number of or nodes varied. The test case and the test result are depicted in Figure 13.

TEST CASE

Data	:	100 set.
Array Dimension	:	[131072][2]
[65536][2]		
[32768][2]		
[16384][2]		
[8192][2]		
Number of Nodes	:	1 Master + 12 Slaves,
1 Master + 6 Slaves		

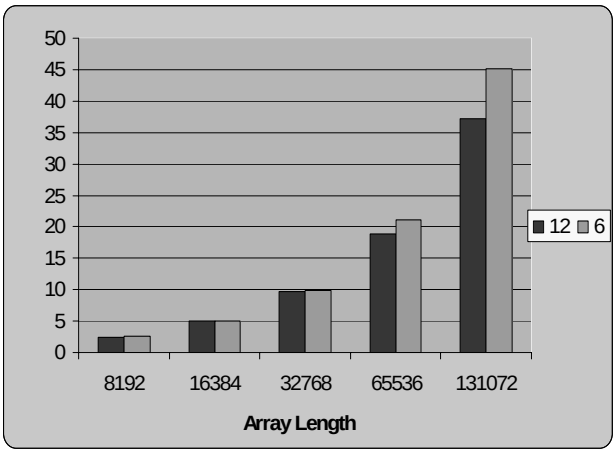

Fig. 13 Processing time versus variable array length

Based on Figure 13, the duration of the processing increases as the size of the arrays are increased. However, the pattern of duration increment is exponential. The underlying cause for such symptom is because when the size of the array is increased, the overhead of the processing increases and as such the duration will follow an exponential pattern of increment.

5.2.4 Speedup Test

This part of the test benchmarks the speedup value of parallel executing in multi-nodes against sequential executing on a single node. The test case is as follows:

TEST CASE

Data	:	100 set.
Array Dimension	:	[131072][2]
Number of Nodes	:	1 Single Node
		1 Master + 4 Slaves,
		1 Master + 6 Slaves
		1 Master + 8 Slaves
		1 Master + 10 Slaves
		1 Master + 12 Slaves

Table 3 Speedup results

Number of Nodes	Execution Time (Second)	Speedup
1	220.013503	NA
4	73.934191	2.97580
6	45.227610	4.86458
8	38.668685	5.68971
10	37.879205	5.80829
12	37.276939	5.90213

Based on the data collected Table 3, it is apparent that the speedup factor increases as the number of nodes involved in the computation is increased. This result basically assists in achieving the objective of the project which suggests that the seismic processing time can be reduced if done in a parallel processing environment. To further illustrate the pattern of the speedup, the graph in Figure 14 was plotted.

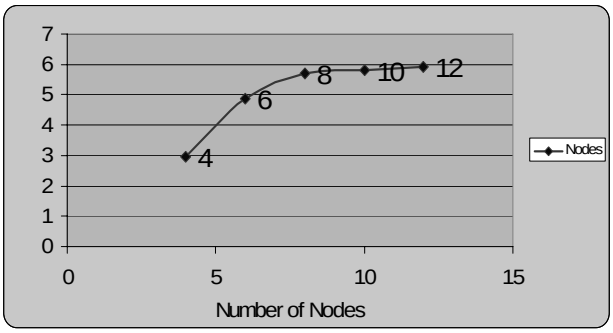

Fig. 14 Speedup pattern in multi processor environment

It is apparent that there is saturation in performance at node 8 till 12 as the speedup is less significant. This is mainly due to the higher communication cost and can be eliminated by increasing the size of the array.

5.3 Discussion

5.3.1 Dynamic Load Balancing

Throughout the test processes where multiple test cases were executed in the developed system, there were several observations that are worth noting. One such aspect is the elements of dynamic load balancing. It was observed through the cluster management system during runtime that the slave nodes were actively switching from high utilization to low and vice versa. This is a very good indication that the load distribution is done dynamically. This is because whenever a particular node is highly utilized, the under utilized nodes will be fed with input data more frequently thus after a period of time, the initially under utilized node increases in processing load while the highly utilized machines drops in load. This can also be observed based on the color codes of the nodes in the cluster management software known as Rocks Linux HPC Distribution. During runtime all the machines are almost equally loaded with input data which is illustrated by the uniform color which indicates the amount of load. Figure 15 below shows that all the 5 slave nodes involved in the processing are equally loaded as discussed earlier. The green shade on compute0-0 till compute 0-4 indicates equality in the load that is being processed. As the processing load increases the color shade would change to yellow and finally red indicating 100% utilization of the processor power of the nodes. The yellow shade in Figure 15 depicts that the master node is handling higher amount of load as compared to the slaves. This is because the master has to actively fetch, distribute and gather date from all the processing slave nodes.

To further consolidate the discussion above, the screen shot of 6 nodes processing summary is presented in Figure 16. In this screen shot the master node indicated as Node 0 has distributed 1000 input data. However, the slave nodes involved in the processing seems to have processed various amounts of data. Node 1 for instance processed 143 data while Node 3 only 72. This means that the dynamic balancing feature that was coded in the system is working effectively. This is because the system distributes more loads to the compute node that finished processing faster than the others. In the static balancing programming paradigm, all the nodes would receive equal amount of data and such is not the case in this system.

In addition to that, the real advantage of the dynamic balancing aspect can be harnessed optimally in a heterogeneous environment rather than the homogenous environment such as the UTPHPC cluster. This is because in the heterogeneous environment, the resources are of various capabilities and as such those with higher processing capacity which will obviously complete a task faster than its slower counterpart, will be relatively fed with more data.

5.3.2 Memory Swapping

A system bottleneck that was detected in the cluster is the insufficient memory on the whole. This was detected when an array of size 131072 with a total data of 1000 set being fed into the system. The entire operation was estimated to take approximately an hour but after 15 minutes, the total memory available in the system was fully utilized and the nodes started using the swap memory in their secondary memory. This is when the system crashed prematurely. The codes were then examined to find out any memory leaks. The examination reveals that the memory used were of dynamic and were de-allocated after usage. From there it was concluded that the operation requires high amount of memory especially when the size of each individual array is extremely large or exceeds 100000 in dimension to be specific. Figure 17 illustrates the acute usage of memory during execution of large data set.

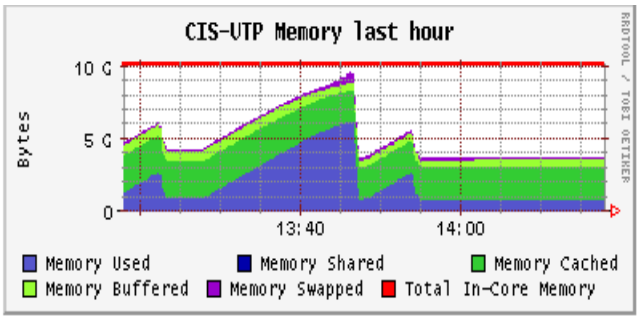

Fig. 17 High memory utilization

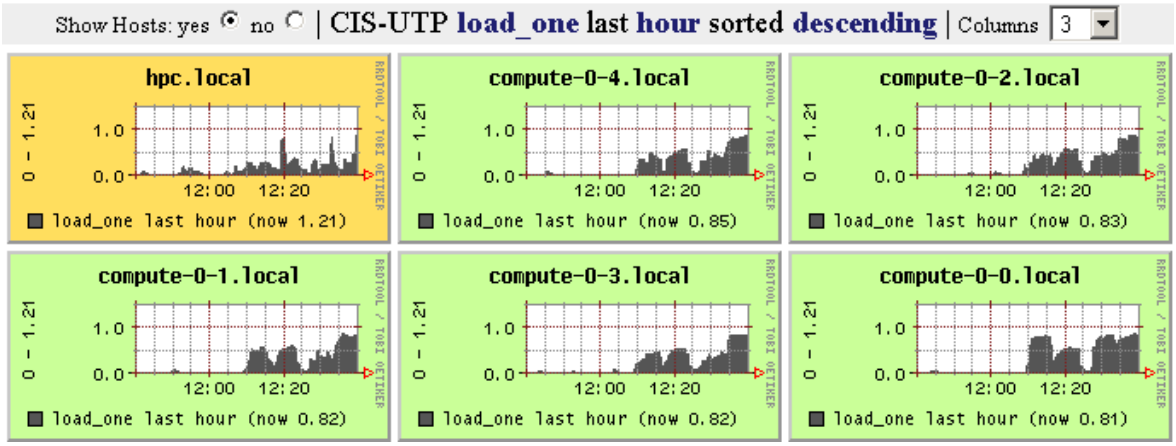

Fig. 15 Dynamic load balancing illustration

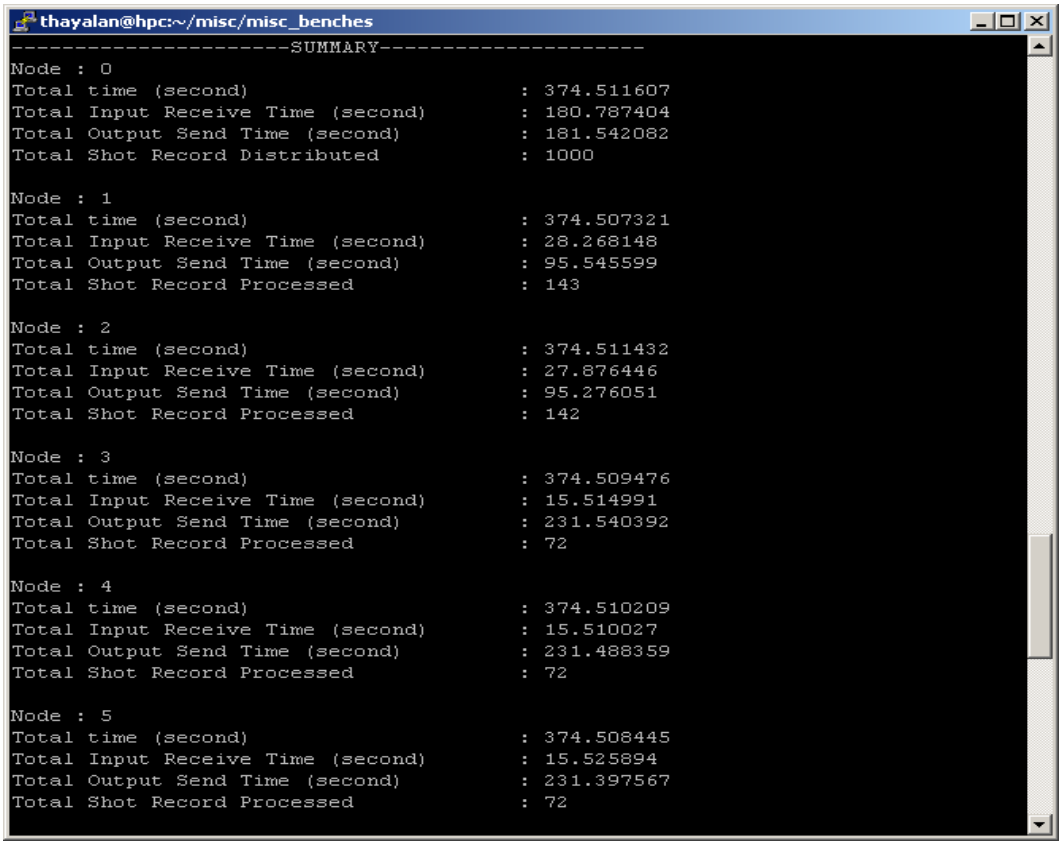

Fig. 16 Program execution summary

6. Conclusion and Future Work

The parallel implementation of the seismic noise reduction program underwent vigorous testing. Based on the results obtained, it can be concluded that there is a decrease in processing duration when the same noise filtering process is implemented in a parallel manner as opposed to sequential. In the effort to consolidate the results, the system was tested for consistency initially and followed by speedup tests in order to observe the degree of performance improvement.

In the future, we are going to design and investigate the performance of parallel implementation of noise removal algorithms using different filters such as band pass filter or Tau –p filter.

References

[1] Data Processing, http://www.cggveritas.com/popup_page.aspx?cid=1-24-163, accessed on 10 April 2008.

[2] Sudhakar Y., Chakraborty S., Bhandare S. and Rastogi R. (2002), Seismic Data Processing Initiatives in High Performance Computing, Tata McGraw Hill, pp. 164.

[3] Fricke J. R. (1988), Reverse-time migration in parallel : A tutorial, *Society of Exploration Geophysicists*, pp. 1143-1144.

[4] Seismic Data Processing, http://www.geo.arizona.edu/geophysics/rseis/resources/Lec-1-569-06.pdf , accessed on 28 February 2008.

[5] Dr Mehmut Ferruh Akalin, Staff Seismic Imaging & Processing, Exploration Geoscience Department, Exploration Division, Discussion session at PETRONAS CARIGALI, Level 15, Tower 2, PETRONAS Twin Towers, Kuala Lumpur City Center, 50088 Kuala Lumpur, Malaysia. <Meeting on 28 March 2008>

[6] Fourier Analysis and Signal Filtering by Constantinos E. Efstathiou, http://www.chem.uoa.gr/applets/AppletFourAnal/Appl_FourAnal2.html, accessed on 20 March 2008.

[7] Laurence T. Y. and Guo M. (2006), High Performance Computing Paradigm and Infrastructure, John Wiley & Sons Inc. Publication..

[8] Needham S. and Hansen T. (2002), Cluster Programming Environments.

[9] Mustafar A. N. , Aziz I. A. , Mehat M. , Haron N. S. , Jung L. T. (2008) , Solving Traveling Salesman Problem on High Performance Computing Using Message Passing Interface, CIMMACS'08, December 28-31, 2008, Cairo, Egypt.

[10] Adhipta D. , Aziz I. A. , Haron N. S. , Jung L. T. (2006), Performance Evaluation On Hybrid Cluster: The Integration Of Beowulf And Single System Image, ICTS'06, November 10, 2006, Surabaya, Indonesia.

Nazleeni Samiha Haron graduated from University College London, U.K. for her MSc Data Comm, Networks and Distributed Systems and Universiti Teknologi PETRONAS, Malaysia for her BTech (Hons.) in Information Technology. Currently, she is a lecturer at the Department of Computer and Information Sciences, Universiti Teknologi PETRONAS. Her research areas include distributed and grid computing.

Mohd Hilmi Hasan obtained his Master of Information Technology from Australian National University and BTech (Hons.) in Information Technology from Universiti Teknologi PETRONAS, Malaysia. Currently, he is a lecturer at the Department of Computer and Information Sciences, Universiti Teknologi PETRONAS. His research areas include mobile applications development and agent-based computing.

Anis Afzan Ab Rahman graduated with BTech (Hons.) in Business Information System from Universiti Teknologi PETRONAS. This paper is part of her final year project as requirement for her final year study.