An Enhanced Parallel & Distributed Implementation of the Harmony Search Based Supervised Training of Artificial Neural Networks

Conference Paper · August 2011

DOI: 10.1109/CICSyN.2011.65 · Source: IEEE Xplore

2 authors:

Ali Kattan
Noble Institute, Erbil, Iraq
33 PUBLICATIONS **149** CITATIONS
SEE PROFILE

Rosni Abdullah
Universiti Sains Malaysia
109 PUBLICATIONS **615** CITATIONS
SEE PROFILE

An Enhanced Parallel & Distributed Implementation of the Harmony Search Based Supervised Training of Artificial Neural Networks

Ali Kattan[1], Rosni Abdullah[2]
School of Computer Sciences
Universiti Sains Malaysia
11800 USM, Penang, Malaysia
[1]kattan@ieee.org, [2]rosni@cs.usm.my

***Abstract*—The authors have published earlier a parallel & distributed implementation method for the supervised training of feed-forward artificial neural networks using the Harmony Search algorithm. Such implementation was intended to address the training of larger pattern-classification problem. The implementation platforms included both a homogeneous and a heterogeneous system of Master-Slave processing nodes. The latter heterogeneous implementation utilized a node benchmarking score obtained via independent software in order to determine the load balancing ratios for the different processing nodes. In this paper an enhanced alternative benchmarking technique is proposed that is based on the actual workload execution times for each heterogeneous processing node. Using the same pattern-classification problem on the same heterogeneous platform setup used in the previous technique, results show that the proposed technique has attained higher speedup in comparison with the former.**

Keywords: neural network; harmony search; parallel & distributed processing; pattern-classification

I. Introduction

A relatively young stochastic global optimization method is the Harmony Search (HS) algorithm. This method draws its inspiration not from biological or physical processes but from the improvisation process of musicians [1]. This is illustrated in Fig. 1 where each musician would generate a note that is analogous to a variable in the overall solution vector. The quality of such vector is measured using a fitness function and the overall optimization process seeks to find the best solution vector having the best fitness value. The application of HS algorithm can be found in various engineering and industrial applications [2, 3].

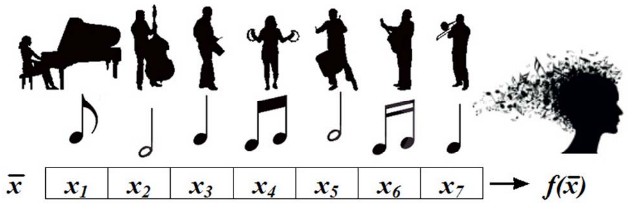


Figure 1. The HS algorithm concept

The authors have published earlier a technique for adapting the HS algorithm for the training of feed-forward artificial neural networks (FFANN) targeting pattern-classification applications [4] as well as a technique using a parallel & distributed implementation to address the training of larger problems [5].

This paper proposes an enhanced parallel & distributed implementation for the Master-Slave heterogeneous platform. The proposed implementation uses a new method for heterogeneous node benchmarking to determine the load balancing ratios. Actual workload execution time benchmarking is used instead of the independent software benchmarking in the former technique [5]. For the same pattern-classification problem on the same platform, results show a higher speedup in comparison with the results obtained earlier.

The rest of this paper is organized as follows: Section II gives the background and related works, section III introduces the proposed method, section IV is the methodology and implementation, section V contains the experimental results, comparisons and discussion, and finally, the conclusion is given in section VI.

II. Background & Related Work

The previously published FFANN training algorithm in [4] is known as Harmony Search using Best-to-Worst (HS-BtW). The algorithm is illustrated in the flowchart of Fig. 2 whereby each harmony vector in the harmony memory (HM) represents a complete set of FFANN weights. The sum of squared errors (SSE) is used as the main fitness function. This represents the error between FFANN output compared to target patterns of the original training set for the pattern-classification problem considered. The SSE computations are needed in every iteration and the analysis has showed that this is where most of the execution time is spent. This is more obvious in case of using a larger network and a larger training data set.

In order to address the training of larger pattern-classification problems, the sequential algorithm in [4] is inadequate due to the elongated convergence time needed for such problems. The iteration execution time is proportional to the number of weight traversal times for which forward-pass computations take place. In every iteration, the total number of weight traversals is equal to the product of the harmony vector size used, representing the total number of network's weights, by the total number of patterns in the training file. Such number is referred to as the forward-pass computation index (FPCI). The formerly published work in [5] considered a problem with an FPCI of 2,177,280 compared to an FPCI of 54,782 used in the sequential implementation in [4].

The previously proposed parallel & distributed implementation in [5] utilized the fact that there are two main repetitive operations in the training process: loading the pattern data from the training file and performing the feed-forward computations to find SSE and as indicated by the shaded components of the flowchart of Fig. 2

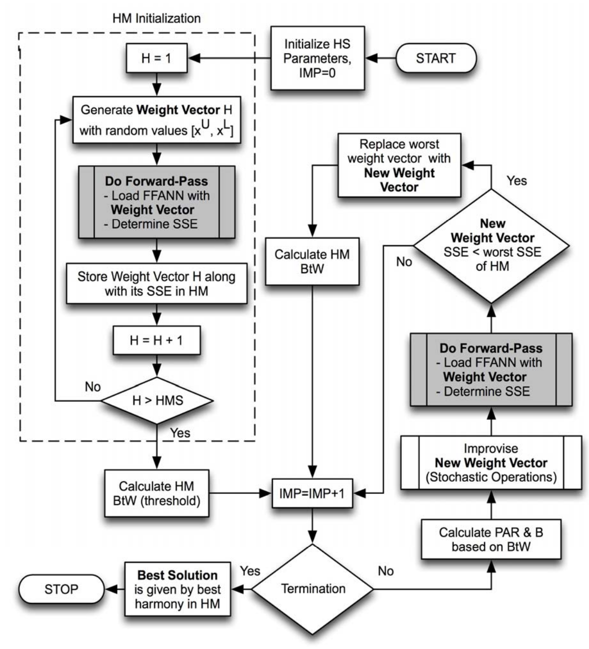
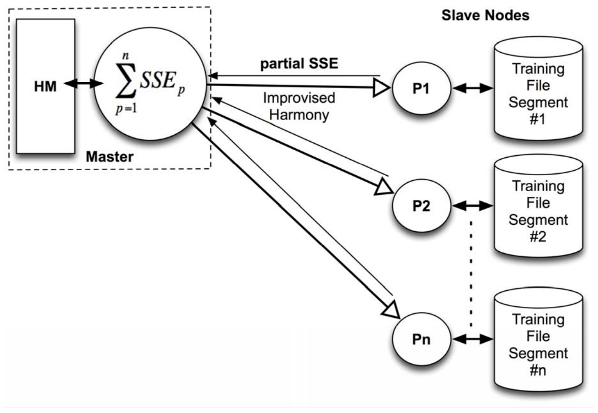

HM Initialization
H = 1
Initialize HS Parameters, IMP=0
START
Generate Weight Vector H with random values [x^U, x^L]
Do Forward-Pass - Load FFANN with Weight Vector - Determine SSE
Store Weight Vector H along with its SSE in HM
H = H + 1
No
H > HMS
Yes
Replace worst weight vector with New Weight Vector
Yes
New Weight Vector SSE < worst SSE of HM
No
Calculate HM BtW
Do Forward-Pass - Load FFANN with Weight Vector - Determine SSE
Improvise New Weight Vector (Stochastic Operations)
Calculate HM BtW (threshold)
IMP=IMP+1
Calculate PAR & B based on BtW
STOP
Best Solution is given by best harmony in HM
Yes
Termination
No

Figure 2. FFANN training using the HS-BtW algorithm [4]

Using the Master-Slave heterogeneous platform, the problem's training data set is partitioned into local segment files assigned for each processing node. Such segmentation would assign a number of training patterns for each processing nodes based on its computing power. This process would take place once only and prior to starting training where each node would have access to a local training file representing a segment of the original training file. All HS related operations are still carried out on the master node having only one copy of HM that is maintained by the master. The master would improvise new harmonies (weight vectors) and send to each slave node for partial fitness function computation and as shown in Fig. 3 [5].

Load balancing plays a crucial part in assigning suitable workloads for each heterogeneous node. With the overall workload initially at the master node, the master, and according to some policy, would divide this work and distribute to slave nodes and recollect results to compute the final solution. Such technique is common for such platforms [6].

The overall workload represents the single training file composed of the problem patterns. To achieve good load balancing, node benchmarking is required in order to compute the size of the training file segment that should be allocated to each node in proportion to its computing power and as given in equation (1) and (2). The single number benchmarking approach [7, 8] was used whereby each node benchmark is represented by a real valued number representing its computational power and referred to as the benchmark score. The former parallel and distributed technique used independent benchmarking software, namely SciMark2, in order to compute *bm* in equation (1) for each node. This software offers a single composite Java benchmark score measuring the performance of numerical codes occurring in scientific and engineering applications [9].



$$SSE = \overset{n}{\underset{p=1}{}} SSE_p = SSE_{p1} + SSE_{p2} + ... + SSE_{pn}$$

Figure 3. The parallel & distributed system architecture [5]

$$S_k = \text{round}(\frac{bm_k}{\overset{n}{\underset{i=1}{}} bm_i} \cdot TP) \qquad (1)$$

$$\overset{n}{\underset{k=1}{}} S_k = TP \qquad (2)$$

where

S_k	training file segment size in patterns for node k
bm	node benchmark score
n	total number of nodes
TP	total training patterns in the original training file

The workload for each node is homogeneous, i.e. various nodes are executing essentially the same task and differ only in their computing powers. Thus, the speedup analysis considered in [5] were obtained relative to a fixed reference machine not necessarily belonging to the processor set [7] and as given in equation (3). SciMark2 benchmarking did not account for the I/O performance for each of the processing nodes involved in the computation. The I/O times were simply neglected assuming that their effect is minimal for all nodes. The overall communication times were also neglected based on the fact that all nodes are using the same-speed dedicated connection network. In comparison with the homogeneous Master-Slave implementation of the previous work [5], the heterogeneous speedup results were considerably inferior (as illustrated by Fig. 4 given later).

$$S_N = \frac{T_{ref}}{T_N} \tag{3}$$

where

S_N	heterogeneous speedup ratio for *N* heterogeneous nodes
T_{ref}	elapsed time for running the sequential program version on a reference node
T_N	elapsed time for running the parallel program version on *N* heterogeneous nodes.

III. The Proposed Method

It is stated that in order to achieve the maximum performance on a heterogeneous set of processing nodes consisting of a variety of machines having different hardware and different processing speeds, it is very beneficial to exploit both parallelism and distributed processing with good load balancing [10]. The performance degradation caused by the heterogeneity of the processing nodes could be handled using good load balancing policy such that the processing time gap between the fastest and the slowest node becomes minimal. With Java being the implementation language for the previously proposed parallel & distributed implementation in [5], I/O time is considered as one of the most important problems for high performance computing in Java [11]. Due to the nature of the FFANN training application where frequent I/O access is needed to obtain training patterns data, the I/O time would be a factor to consider in the node benchmarking process.

In this paper, the benchmark score used in equation (1) is obtained differently. The one metric that almost all benchmarking approaches agree on is the overall elapsed time to execute a certain task [7, 12]. The enhanced technique would consider the average elapsed time taken by each node to complete the same size workload. The master would send such workload to the different heterogeneous slave nodes and record the elapsed time needed by each node to complete its work. Such benchmarking approach would not only incorporate the overall time needed for computations but also the overall I/O and communication time required by each node.
Based on such scenario, for a workload represented by application ***A***, the reciprocal of each node's elapsed execution time needed to complete ***A*** divided by the sum of all nodes elapsed times could be used as a benchmark. This is given in equation (4) as the elapsed time benchmark (*ETB*) whereby faster nodes would have a higher benchmark value since they have smaller execution times. The obtained benchmark scores are to be substituted in equation (1) to obtain the training file segment size for each node based on its *ETB* score.

$$ETB_k(A) = \left(\frac{t_k(A)}{\sum_{p=1}^{n} t_p(A)}\right)^{-1} \tag{4}$$

where

$ETB_k(A)$	elapsed time benchmark for node k running application A
$t_k(A)$	elapsed time needed by node k to complete A
n	total number of slave nodes

The workload considered for such benchmarking process is the elapsed time required by each node to perform a fixed number of actual training iterations using the original training file of the problem considered. The main idea is to allow enough time for each node's Java Virtual Machine to locally optimize its code until the execution time becomes stable [13-16]. Usually a few number of iterations (≤ 10) is sufficient to obtain such stable execution time.

IV. Methodology & Implementation

With the exception of the load-balancing technique used in the authors' previous work in [5], the same system, pattern-classification problem and original Java code were used for the current implementation to establish a valid comparison. The Thyroid problem specifications are given in Table I and the heterogeneous Master-Slave system, shown earlier in Fig. 3, is represented by the PCs given in Table II. The Master-Slave heterogeneous platform represents a heterogeneous cluster composed of a set of six commodity PCs connected via a 100Mbps dedicated Ethernet switch. PC1 represents the master node as well as the reference node used for the sequential implementation. The Java code used in the previous implementation in [5] was modified to account for the newly proposed load balancing technique, namely the *ETB* method.

In order to determine the benchmarking score for each participating heterogeneous node, the master would initially send the complete data set file to each node. The nodes would independently load this file and actually perform ten training sessions using the given FFANN structure. This would represent an equal workload for each node. The overall elapsed times needed to complete the ten sessions, inclusive of sending the data and receiving back the completion signal are recorded by the master. The master would then determine the ETB scores using equation (4). Once the benchmarking scores are determined, load balancing would take place by segmenting the data set among heterogeneous nodes using equation (1) and (2) whereby the actual parallel & distributed training session can commence.

V. Results & Discussions

The average iteration elapsed time was 10,520.51ms measured at the master and used as a reference for the speedup calculations using equation (3). Table III lists the obtained results using the *ETB* method along with those obtained previously using SciMark2 benchmarking method (indicated by the shaded table rows). As before, the standard deviation indicates the effectiveness of the static load balancing policy used where in an ideal situation this value should be zero. It is obvious from this table that the standard deviation values attained using the newly proposed *ETB* method are much lower than those obtained previously using SciMark2 benchmarking. Consequently, the speedup values for the newly proposed method are higher in comparison. The speedup graph given in Fig. 4 shows the previously obtained results in [5] along with those using the *ETB* method.

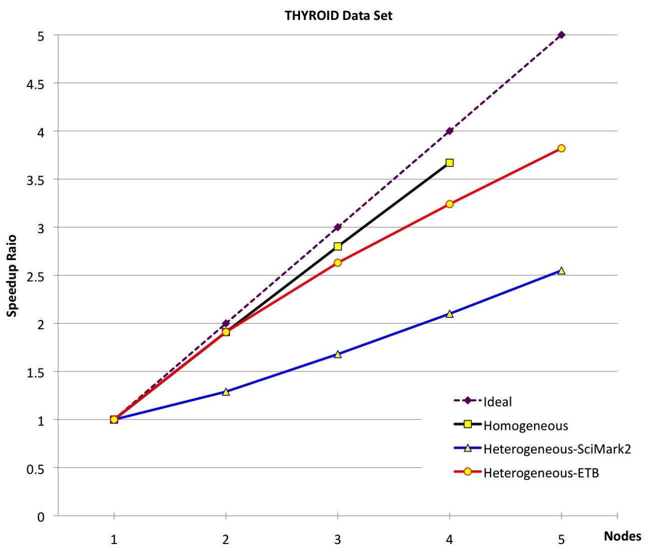


Figure 4. Iteration speedup graph for the Thyroid data set training

The improved speedup results can be justified with the help of Table IV listing the benchmark scores determined by both SciMark2 and *ETB* for each PC. These are illustrated in Fig. 5 as ranks where it can be clearly seen that the nodes were ranked very differently. SciMark2 scores are data independent and measure the performance of each node, represented by a PC, regardless of the workload carried out. *ETB* scores on the other hand measure the benchmark based on the workload used, represented by carrying 10 feed-forward training iterations using the Thyroid data set. The nodes' score ranks achieved by the SciMark2 benchmarking are different from those achieved by the workload *ETB*. For instance, SciMark2 has ranked PC5 as the fastest and most powerful node. Consequently, and based on equation (1), PC5 was assigned a larger training file segment with respect to the other nodes. In the workload *ETB* however, PC4 was ranked as the fastest and most powerful node. Scimark2 scores were solely based on CPU power not accounting for other factors such as I/O and communication times. Checking the hardware specifications of both PC5 and PC4 in Table II, it is clear that PC5 has a more powerful CPU than PC4 but the latter is equipped with much faster hard disk enabling faster I/O access to training data in comparison with PC5.

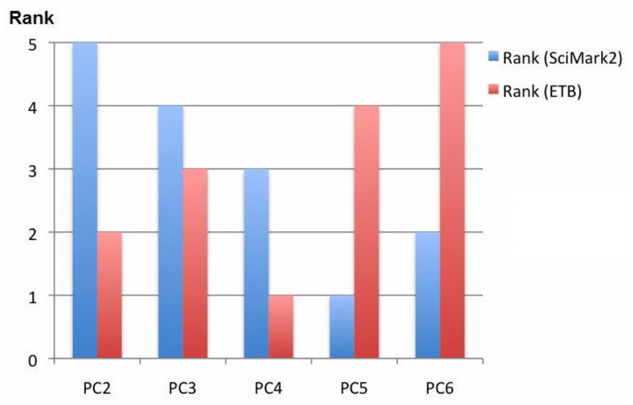


Figure 5. Benchmark as rank for each node as determined by the two load balancing methods (A lower rank indicates a faster machine)

It is also noted that the standard deviation value for the nodes elapsed times when using the SciMark2 benchmarking (Table III), decreases as the number of processing nodes increases. Increasing the number of nodes would decrease the local segment file size allocated for each node and hence the I/O access times are decreased due to the smaller file size. This would lessen the effect of I/O delays and give the CPU computing power a more dominant role. The opposite occurs in case of using the workload *ETB* where the standard deviation values would generally increase in proportion to increasing the number of processing nodes. The effect of I/O becomes less as the local segment file size becomes smaller. Nevertheless, the static load benchmarking using the workload *ETB* gave much better results than that of the SciMark2 benchmarking for the number of processing nodes considered for this work.

VI. Conclusion

The parallel and distributed implementation for the HS-BtW algorithm is required for addressing the FFANNs training of larger pattern-classification problems. Such implementation would reduce the overall convergence time required for such problems. Better speedup could be achieved by using a more efficient load balancing policy for the heterogeneous Master-Slave system. The elapsed time benchmarking (*ETB*) method proposed in this work gave much better results than those obtained using the SciMark2 software benchmarking. The newly proposed benchmarking method has accounted for both the overall I/O times as well as the communication times for the workload considered where these times were neglected by the former method. This has resulted in a more accurate load balancing in terms of assigning appropriate training file segment sizes to the heterogeneous nodes involved in the computation and thus attaining better speedup.

ACKNOWLEDGMENT

The first author would like to thank Universiti Sains Malaysia for accepting him as a postdoctoral fellow. This work is funded by the Fundamental Research Grant Scheme (Pindaan 2/2009) awarded to the 2nd author from “Jabatan Pengajian Tinggi Kementerian Pengajian Tinggi” (Project Account number 203/PKOMP/6711136).

REFERENCES

[1] K. S. Lee and Z. W. Geem, "A New Meta-heuristic Algorithm for Continuous Engineering Optimization: Harmony Search Theory and Practice," *Computer Methods in Applied Mechanics and Engineering,* vol. 194, pp. 3902-3933, 2005.

[2] Z. W. Geem, Music-Inspired Harmony Search Algorithm: Theory and Applications vol. 191: Springer, 2009.

[3] Z. W. Geem, "Harmony Search Applications in Industry," in Soft Computing Applications in Industry. vol. 226/2008: Springer Berlin / Heidelberg, 2008, pp. 117-134.

[4] A. Kattan, R. Abdullah, and R. A. Salam, "Harmony Search Based Supervised Training of Artificial Neural Networks," in International Conference on Intelligent Systems, Modeling and Simulation (ISMS2010), Liverpool, England, 2010, pp. 105-110.

[5] A. Kattan and R. Abdullah, "Parallel & Distributed Implementation of the Harmony Search Based Supervised Training of Artificial Neural Networks," in Intelligent Systems, Modelling and Simulation (ISMS2011), Kuala Lumpur & Phnom Penh, 2011, pp. 277-283.

[6] F. Almeida, D. Gonzalez, and L. M. Moreno, "The master-slave paradigm on heterogeneous systems: A dynamic programming approach for the optimal mapping," Journal of Systems Architecture, vol. 52, pp. 105-116, 2006.

[7] A. Clematis and A. Corana, "Modeling performance of heterogeneous parallel computing systems," Parallel Computing, vol. 25, pp. 1131-1145, 1999.

[8] R. M. Yoo, H.-H. S. Lee, H. Lee, and K. Chow, "Hierarchical Means: Single Number Benchmarking with Workload Cluster Analysis," in IEEE 10th International Symposium on Workload Characterization (IISWC 2007), Boston, MA, USA, 2007, pp. 204 - 213.

[9] H. Oi, "Local variable access behavior of hardware-translation based Java virtual machine," The Journal of Systems and Software, vol. 81, pp. 2059-2068, 2008.

[10] J. H. Park and B. A. Demirdag, "High Performance Pattern Matching with Dynamic Load Balancing on Heterogeneous Systems," in 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06): IEEE Computer Society, 2006, p. 6.

[11] J. M. Pérez, L. M. Sanchez, F. García, A. Calderón, and J. Carretero, "High performance Java input/output for heterogeneous distributed computing," in Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC 2005), Cartagena, Spain, 2005, pp. 969-974.

[12] J. Al-Jaroodi, N. Mohamed, H. Jiang, and D. Swanson, "Modeling parallel applications performance on heterogeneous systems," in Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS’03), Nice, France, 2003.

[13] J. Shirazi, Java Performance Tuning, 2nd ed.: O'Reilly, 2003.

[14] Sun Microsystems, Inc., "Java Tuning White Paper," Sun Microsystems, 2005.

[15] F. P. Miller and A. F. Vandome, Java Virtual Machine: Alpascript Publishing, 2009.

[16] B. Boyer, "Robust Java Benchmarking, Part 1: Issues: Understanding the pitfalls of benchmarking Java code," IBM developerWorks, 2008, pp. 1-20.

TABLE I. THE THYROID PATTERN-CLASSIFICATION PROBLEM DATA SET

Data Set	FFANN Architecture	Training Patterns	Total Weights	FPCI	Data Set Description
Thyroid	21-15-3	5,760	378	2,177,280	Thyroid Disease Data Set: Detect thyroid being (Normal, Hyper-function, Subnormal) based on certain biological features.

TABLE II. HETEROGENEOUS SYSTEM NODE SPECIFICATIONS

PC	Operating System	CPU Type & Speed	RAM	Hard Disk Type & Speed (R.P.M)
1	MS Windows XP (32-bit)	AMD Athlon 64x2 at 2.8GHz	2GB	PATA / 5,400
2	Linux Ubunut 8 (64-bit)	AMD Athlon 64 at 1.80GHz	2GB	PATA / 7,200
3	MS Windows XP (32-bit)	AMD Athlon 64 at 1.80GHz	512MB	PATA / 7,200
4	MS Windows XP (32-bit)	AMD Athlon 64 at 2.00GHz	1.5GB	SATA / 7,200
5	MS Windows XP (64-bit)	AMD Athlon 64x2 at 2.2GHz	2GB	PATA / 5,400
6	Linux Ubuntu 8 (32-bit)	Intel Core 2 at 1.86GHz	512MB	PATA / 5,400

TABLE III. THYROID DATA SET AVERAGE TRAINING ITERATION TIME AND SPEEDUP ON A HETEROGENEOUS CLUSTER

Training Iteration:	**2 Nodes**	**3 Nodes**	**4 Nodes**	**5 Nodes**
	PC: 4, 5	PC: 4, 5, 3	PC: 4, 5, 3, 2	PC: 4, 5, 3, 2, 6
(Using SciMark2) Avg. Elapsed Time (ms)	8125.91	6250.15	5000.92	4125.81
(Using ETB) Avg. Elapsed Time (ms)	5,499.59	3,999.5	3,250.94	2,750.88
(Using SciMark2) Avg. Elapsed Time Standard Deviation	2.381	1.6165	1.2011	0.8876
(Using ETB) Avg. Elapsed Time Standard Deviation	0.1624	0.2826	0.2507	0.2779
(Using SciMark2) Speedup Ratio	1.29	1.68	2.10	2.55
(Using ETB) Speedup Ratio	1.91	2.63	3.24	3.82

TABLE IV. HETEROGENEOUS NODE RANKING BASED ON BENCHMARK SCORES

PC	**SciMark2**		**ETB (Thyroid)**	
	Score	Node Rank	Score	Node Rank
2	301.7	5	5.12	2
3	308.9	4	5.10	3
4	340.6	3	6.51	1
5	596.3	1	4.56	4
6	363.8	2	4.25	5