Experimental Evaluation of the Performance of Processing Stealing Technique: A Scalable Load Balancing Technique for a Dynamic Multiprocessor System

O. O Olakanmi
Electrical and Electronic Engineering
University of Ibadan, Nigeria

O.A Fakolujo (Ph.D)
Electrical and Electronic Engineering University of Ibadan, Nigeria

ABSTRACT

This paper reports preliminary experimental evaluation of a Processing Elements Stealing (PE-S) technique which was targeted as efficient and scalable load balancing technique for dynamically structured multiprocessor systems. The multiprocessor system is imagined as a dynamic cluster based multiprocessor. Each cluster of the multiprocessor system is a node in symmetric multiprocessor architecture and the number of Processing Element (PE) in each cluster is dynamically determined at runtime. The PE-S technique dynamically computes the configuration ratio using the number of threads in the dynamically assigned tasks to generate the new number of PE for each cluster. This new configuration ratio is thereafter used to balance the additional computational work generated by runtime instantiation of current workloads for each cluster.

In this work, the efficiency of the PE-S was evaluated using memory traces of some tightly parallel applications where the amount of parallelism is parameterized. These traces were used as workloads on two different simulation setups; the first is a dynamic multiprocessor with PE-S while the other was also a dynamic multiprocessor but without PE-S. This is to evaluate the performance of the PE-S load balancing technique on the targeted multiprocessor. Also the efficiency of PE-S reconfigurations was compared with other possible reconfiguration ratios. The experimental results showed that the load balancing algorithm is efficient and scalable for balancing at least 100,000 instructions tasks and PE-S generated ratios are averagely better than any other reconfiguration ratios.

General Terms

Parallel computing, Load balancing, multiprocessor

Keywords

Load balancing, multiprocessor, parallel application, work stealing and sharing, processing element stealing

1. INTRODUCTION

The rapid developmental trends in hardware and software technologies have led to increased interest in the use of multiprocessor systems for online database, real-time, defence strategy systems, and power intensive commercial applications. One of the major problems of multiprocessor systems is how to evenly distribute (or schedule) the processes among processing elements to achieve some performance goal(s), such as minimizing execution time, minimizing communication delays, and/or maximizing resource utilization. Therefore, load balance has become integral factor in maximising the speed up of parallel and distributed environments. In recent time, multiprocessor systems have been a subject of interest. Present researches had shown that uniprocessor technology can hardly be subjected to reasonable improvement thereby could no longer meet up with processing power requirement of the current applications. This is due to insatiable demand for computing power by users which is generated from development of powerful applications in order to meet up with the users' demand. Parallel and Distributed processing has proffered solution to this by combining many processing elements together to behave as a single processor.

Research works are still on-going on how to perfect some of the performance bottlenecks in multiprocessor systems through the adoption of some of the computer network speedup metrics to multiprocessor architecture. For example, different network topologies had been modelled, evaluated and implemented in multiprocessor systems which have brought variant multiprocessor architectures. Apart from this, concept of memory hierarchy and optimum scheduling techniques had been introduced just to achieve efficient multiprocessor systems. In spite of all these improvement metrics, multiprocessor systems performance is still marred with inefficiency in job distribution during execution which affects overall throughput of the systems. A few researches had been done, and many are still on-going on how to get a perfect load balancing technique; however, a perfect technique has become elusive. One of the biggest performance issues in the current load balance techniques is that they are system specific and some of the loads have more affinity for certain processing elements than the others. This mars the performance gain of most of the available load balance techniques. Many techniques for load balancing in multiprocessor systems had been proposed. The prominent among them are work stealing and work sharing. Recently, another technique was proposed in [15], called Processing Elements or worker stealing technique. This paper performed experimental evaluation of this technique and evaluates its performance in terms of its performance influences on the speed-up, when the technique is implemented on a dynamic multiprocessor.

The rest of the paper is organized as follows; Section 2 presents the reviews on the related research works on load balancing in multiprocessor architecture. In Section 3, background work on PE-S technique is described. The experimental evaluation of PE-S technique is described in Section 4, and the discussion as related to the obtained simulation results is presented in Section 5.

2. RELATED WORKS

Many load balancing algorithms have been proposed for parallel and cloud computing to prevent load imbalance [3][5][8][12][15][18]. Each of these algorithms uses different techniques to achieve load balancing among the processing elements. Work stealing and work sharing technique gained tremendous popularity due not only to their provable efficiency but their practicability which could be easily explained within the circle of multiprocessor designers. However, it has been shown and proved from different experimental results performed that work stealing or sharing techniques are susceptible to so much overheads incurred from some computational tasks having affinity for certain set of processor and some other computational resources[15]. For example, there are some communication costs involved in transferring the job which come from, both the extra contention for the system bus and the latency of the transfer. Besides, some jobs have some affinity for the queue where they are assigned. This affinity emanates from the fact that the applications we are considering have a spatial breakdown of the computations. It follows that a large amount of the data needed by computations will be cached. However, when processor steals or shares a task from another processor it can no longer take advantage of cached computation. These inherently show that although work stealing may be popular and common however it is not a perfect technique for load balancing in parallel and distributed system [1][3][4].

In work sharing busy processors voluntarily redistribute their excess workload in their respective queue amongst the less busy or idle processors. Most times, a dedicated processor does the monitoring of the queues of all the processors in the system, detects the busy or idle processors and performs the distribution amongst them. This not only incurs communication cost but also eliminates the dedicated processor from computational work. More so, the issue of job affinity or processor affinity still explicitly affects the system. It is likely that in the presence of extreme communication costs or very strong affinity of jobs for the processors they are assigned to, work sharing will mar the optimal performance as a result of the fact that shared works will be transported from its initial processing element to the new processing element. In that case, any attempt to share or steal works will disrupt the scheduling affinity. In [6] work stealing was proposed as better alternative to work sharing if this holds, it suggests a very interesting question that can multiprocessor architecture ease the burden on the parallel programmer by allowing work stealing? In other words, is work stealing a substitute for affinity scheduling? Specifically, if the architecture performs work stealing and the programmer does a reasonable, but not optimal job of balancing the load, how will the system performance compare to the performance of a perfectly balanced system? The answer to these is that the work stealing technique cannot proffer perfect solution to load imbalance in most multiprocessor architecture especially architectures that exhibit scheduling affinity. Also, as mentioned earlier, work stealing introduces communication overhead which reduces the performance efficiency of the architecture. This was substantiated with the analog given [15] that; “work stealing and sharing can be analogized to getting job that one’s lacks the tools or acquiring problem without having all the require problem-solving-tools”. This is in reality amounts to waste of time and scarce resources.

A work stealing algorithm which uses locality information was used in [1]; this outperforms the standard work stealing algorithm benchmarks. In this algorithm, each processor maintains a queue of pointer to threads that have an affinity for such processor and during stealing priority is given to queues which have an affinity for such processor. In [6], an algorithm was proposed which implements work stealing to prevent load imbalance in a multiprocessor system. The algorithm has one dequeue processor and the algorithm assumes that processors on the architecture can work independently but can still steal from any of the processors which have empty dequeue. The results show that work stealing has lower communication cost than work sharing. Also in [11], differential equation was used to model work stealing technique. In [12], work stealing architectural technique was proposed to prevent load imbalance in homogenous shared-memory multiprocessor architecture.

The evaluation results in [19] corroborate some of the aforementioned demerits of work stealing. In their work, the limitation of work stealing scheduler was explored and evaluated with another load balancing technique which is based on graph partitioning. The experimental results obtained on a multi-core workstation machine showed that the main cause of performance degradation of work stealing is when works of very little processing time are involved. Meanwhile this is the type of workload in which graph partitioning approach has the potential to achieve better performance than work-stealing. This was further strengthened in [5] where design and preliminary evaluation of an integrated load distribution-load balancing algorithm which was targeted to be both efficient and scalable for dynamically structured computations was reported. In their work computation was represented as a dynamic hierarchical dependence graph. Each node of the graph might be a sub graph or a computation and the number of instances of each node is dynamically determined at runtime. The algorithm combines an initial partitioning of the graph with application of randomized work stealing on the basis of sub graphs to refine imbalances in the initial partitioning and balance the additional computational work generated by runtime instantiation of sub graphs and nodes. Dynamic computations are modeled by an artificial program (k-nary) where the amount of parallelism was parameterized. The experimental results carried out on IBM SP2s suggested that the load balancing algorithm is efficient and scalable for parallelism up to 10,000 parallel threads for closely coupled distributed memory architectures.

A simple algorithm to distribute loads evenly on multiprocessor computers with hypercube interconnection networks was proposed in [10]. This algorithm was developed based upon the well-known dimension exchange method. However, the error accumulation suffered by other algorithms based on the dimension exchange method is avoided by exploiting the notion of regular distributions, which are commonly deployed for data distributions in parallel programming. This algorithm achieves a perfect load balance over P processors with an error of 1 and the worst-case time complexity of O(M log2 P), where M is the maximum number of tasks initially assigned to each processor [10]. Furthermore, perfect load balance is achieved over subcubes as well—once a hypercube is balanced, if the cube is decomposed into two subcubes by the lowest bit of node addresses, then the difference between the numbers of the total tasks of these subcubes is at most 1. However, this algorithm was tailored towards a particular multiprocessor system architectural network.

3. PROCESSING ELEMENT STEALING (PE-S) TECHNIQUE

In view of the importance of load balancing to performance of multiprocessor architecture and the deficiencies of the existing load balancing techniques, an alternative technique was proposed in [15] which exploits the possibility of stealing workers instead of stealing work. Worker stealing involves stealing the processing elements with all the resources of such elements. This apparently removes affinity for either job or processor. PEs-stealing technique not only solves the problem of load imbalance but does not affect queue affinity schedule and with little or no communication cost. PE-S provides some benefits as a result of the fact that it balances the load on a more instantaneous level than work stealing especially for interdependent sub problems which initially brought about affinity schedule. Whenever a cluster is over loaded, the PE-stealing heuristic senses this and reconfigures the architecture by reassigning more processing element(s) to that cluster. The technique concurrently monitors over-loading and under loading in each cluster by releasing some of idle clusters' processing elements to its neighboring clusters that are overloaded using the reconfiguration ratio. The practicability of this technique was done in [15] by implementing the technique in a heuristic called PE-S heuristic which uses the number of threads in the assign computational tasks to determine the current reconfiguration ratio of the multiprocessor. However, the efficiency of the heuristic with provable laboratory or simulated experimental results was not done.

The heuristic as shown in fig.1 calculates the total number of threads in each workload assigned to the modularized unit of the multiprocessor system. This is used to dynamically get the current reconfiguration ratio $n\_PE'_i$ of the architecture as shown in equation 1.The $n\_PE'_i$ is compared with previous configuration ratio n_PE_i and the processing elements are joggled (stolen or released) until the $n\_PE'_i = n\_PE'_i$ The heuristic is highlighted below:

```
1.  Reset the multiprocessor to default.
2.  Accept the parallel tasks (t1 ….tn) where n <= 4
3.  Span through the tasks (t1…tn) determine number of
    processes in each parallel task
4.  Initialize n to 1
5.  Start from task n and cluster n
6.  For task n and cluster n calculated current configuration
    ratio
        while (current_configuration_ratio >
            previous_configuration_ratio)
                1. Remove one node from next cluster
                   and change the status of the node's.
                2. index = ( index of last node in the
                   cluster + 1)
                3. Increment
                   Pprevious_configuration_ratio by 1
7   Increment n by 1 and Go to 5
8.  Store the status of all the reconfigured nodes
9.  Assign all the input parallel tasks to the clusters in the
    HMPM
10. Stop
```

This can be mathematically represented as follows:

Assuming

$$n\_PE_i = Total\ number\ of\ processing\ elements\ \ in\ the\ cluster_i\ \ before\ \ reconfiguration$$

$$n\_PE'_i = Total\ number\ of\ \ processing\ elements\ \ in\ cluster_i\ \ after\ \ reconfiguration$$

Then, the PE-S says;

$$n'_{PE_i} = \frac{(number\ of\ thread\ in\ task\ assigned\ to\ cluster_i * 16)}{Total\ number\ of\ thread\ assigned\ to\ the\ multiprocessor} \ldots\ldots.\ldots (1)$$

For all the clusters i (where i<=4) the architecture performs these:

```
while (n_PE'_i ≠ n_PE_i)
{
 if n_PE'_i > n_PE_i then
 n_PE_i = n_PE_i + 1   i.e. borrow processing element from the next cluster
 C_iE_{n+1} = C_{i+1}E_1   i.e change the status of the borrowed processing element
 n = n+1
 else
 n_PE_i = n_PE_i − 1   i.e released processing element from the next cluster
 C_{i+1}E_1 = C_iE_n   i.e change the status of the released processing element
 n = n-1
}
```

4. EXPERIMENTAL EVALUATION OF PE-S LOAD BALANCING TECHNIQUE

All the experimental workloads used during simulation are the traces of a wide variety of real application programs. These traces represent real parallel applications as shown in table 1a-b. Twenty three experimental simulations were carried out, using combination of four parallel applications traces with different number of parallelism, to evaluate the performance of PE-S. Nineteen experimental workloads shown in table 1a were used to evaluate the performance of PE-S technique in a dynamic multiprocessor mode, and four workloads, as shown in table 1b, were used to compare PE-S and manually generated configuration ratios. The multiprocessor model used is a Hybridised Macro Pipeline Multiprocessor (HMPM) with 64KB main memory. The replacement policy used for the two level 4KB caches was Least Frequently Used (LFU) with fully associative mapping. .

An experimental setup which represents a multiprocessor model with PE-S and another multiprocessor model without PE-S was used as controls in order to perform the evaluation of PE-S technique. The first experimental setup represents the reconfigurable multiprocessor setup while the latter stands for non reconfigurable multiprocessor setup. The performance was measured in terms of the execution time (in second) of each cluster when parallel workloads were mapped into the clusters.

The simulation results obtained from the two experimental setups are analysed and discussed in the next section.

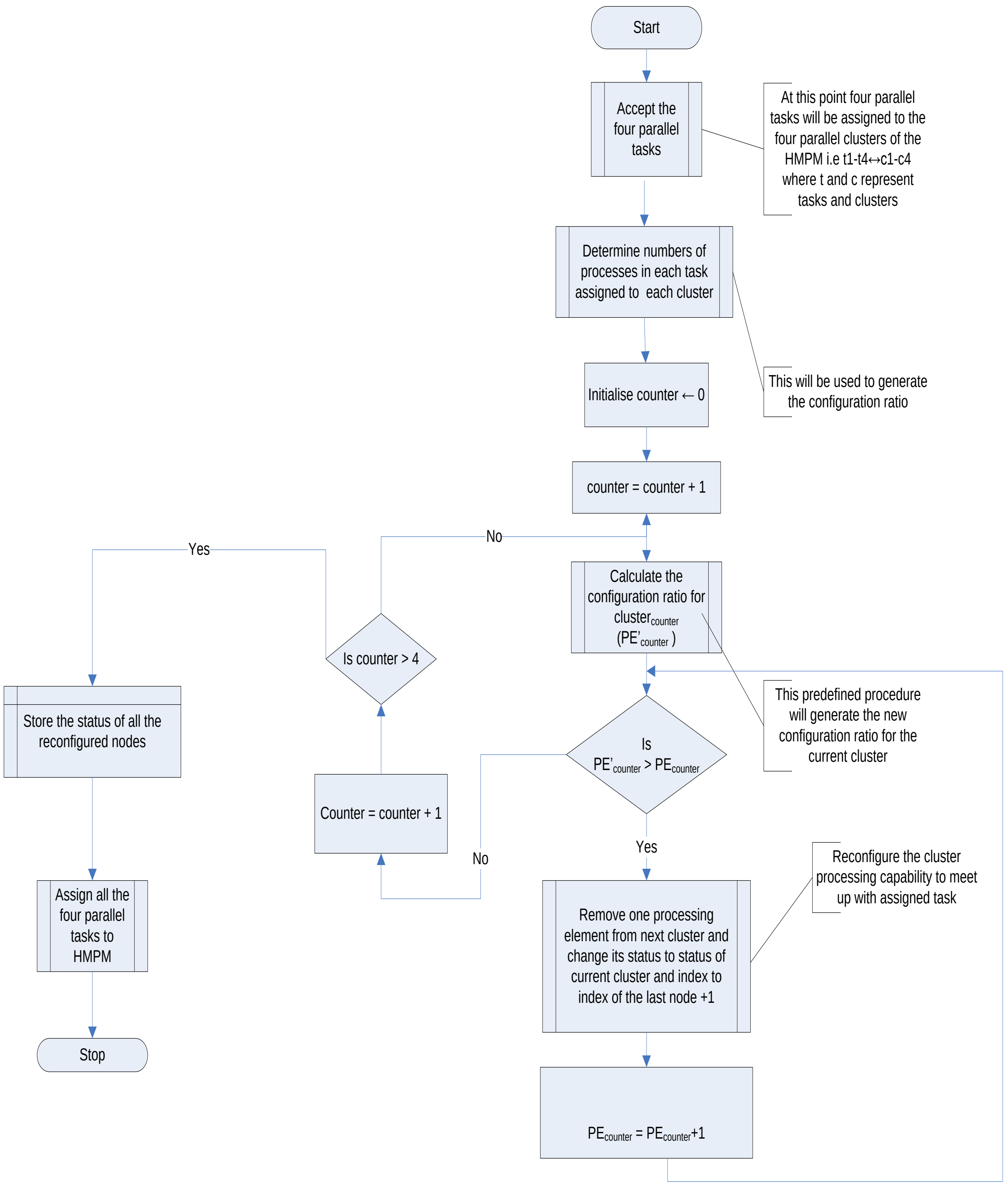
Start
Accept the four parallel tasks
At this point four parallel tasks will be assigned to the four parallel clusters of the HMPM i.e t1-t4↔c1-c4 where t and c represent tasks and clusters
Determine numbers of processes in each task assigned to each cluster
This will be used to generate the configuration ratio
Initialise counter ← 0
counter = counter + 1
No
Yes
Calculate the configuration ratio for cluster$_{counter}$ (PE'$_{counter}$)
Is counter > 4
Store the status of all the reconfigured nodes
This predefined procedure will generate the new configuration ratio for the current cluster
Is PE'$_{counter}$ > PE$_{counter}$
Counter = counter + 1
No
Yes
Reconfigure the cluster processing capability to meet up with assigned task
Assign all the four parallel tasks to HMPM
Remove one processing element from next cluster and change its status to status of current cluster and index to index of the last node +1
Stop
PE$_{counter}$ = PE$_{counter}$+1

Fig. 1: Flowchart of Processing Element Stealing technique

Table 1a: Randomly Generated workloads used in the evaluation of PE-S Technique

Load No.	Cluster 1	Cluster 2	Cluster 3	Cluster 4
1	HY5	SIMPLE4	WEATHER 4	SPEECH3
2	FFT4	SIMPLE3	SIMPLE3	SPEECH6
3	SPEECH3	FFT5	SPEECH4	WEATHER 4
4	SIMPLE5	FFT6	-	SPEECH5
5	HY2	WAVE4	SPEECH4	FFT6
6	WAVE4	SPEECH3	FFT5	HY4
7	SPEECH3	FFT4	HY5	WAVE4
8	FFT3	HY4	SPEECH3	WAVE6
9	FFT4	SIMPLE3	WEATHER5	SPEECH4
10	FFT4	SIMPLE6	WEATHER3	SPEECH3
11	FFT6	SIMPLE4	WEATHER4	SPEECH2
12	FFT4	SIMPLE3	WEATHER4	SPEECH5
13	FFT3	SIMPLE5	SIMPLE5	HY4
14	FFT4	SIMPLE5	SIMPLE4	SPEECH3
15	SPEECH3	FFT6	SPEECH4	WEATHER3
16	SIMPLE4	FFT6	-	SPEECH6
17	SIMPLE5	FFT5	-	FFT6
18	SIMPLE6	FFT5	-	SPEECH5
19	SIMPLE3	FFT4	HY6	SPEECH3

Table 1b: Randomly generated workloads used in performance evaluation of PE-S and manually generated configuration ratios

Load No.	Cluster 1	Cluster 2	Cluster 3	Cluster 4
20	FFT6	SIMPLE4	WEATHER3	SPEECH2
21	FFT4	SIMPLE3	SIMPLE3	SPEECH6
22	SPEECH2	FFT5	SPEECH4	WEATHER4
23	SIMPLE5	FFT6	-	SPEECH5

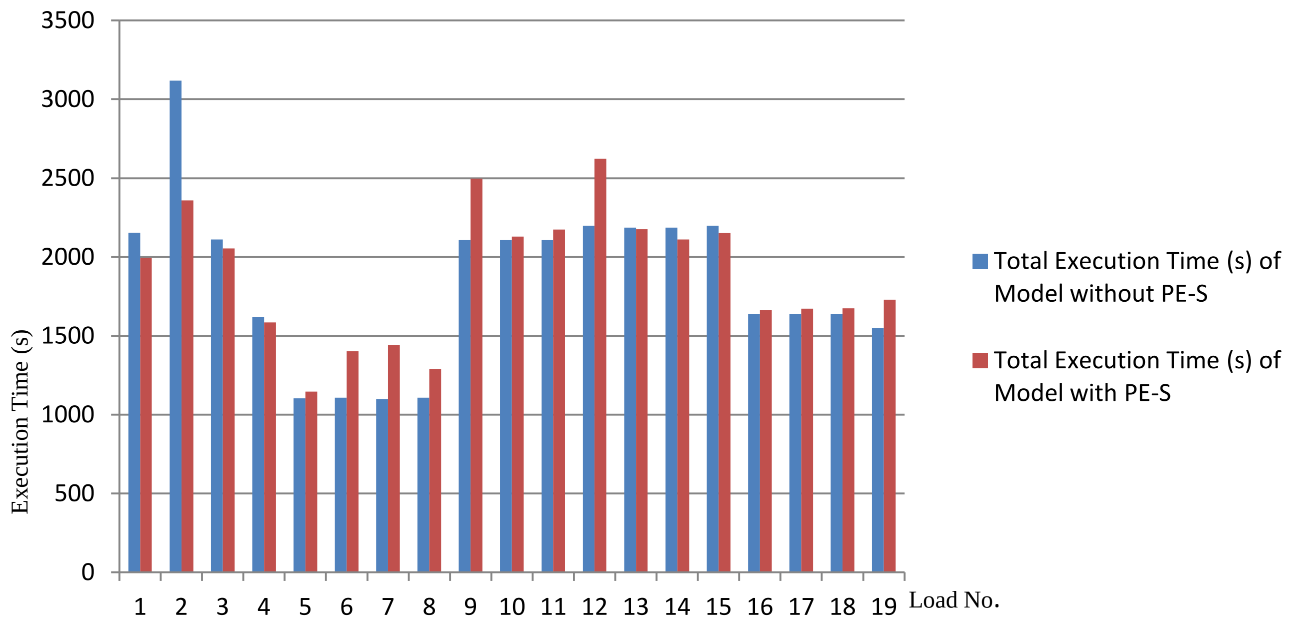

Fig. 2: Comparison of Execution times of Multiprocessor model with and without PE-S for nineteen different workloads

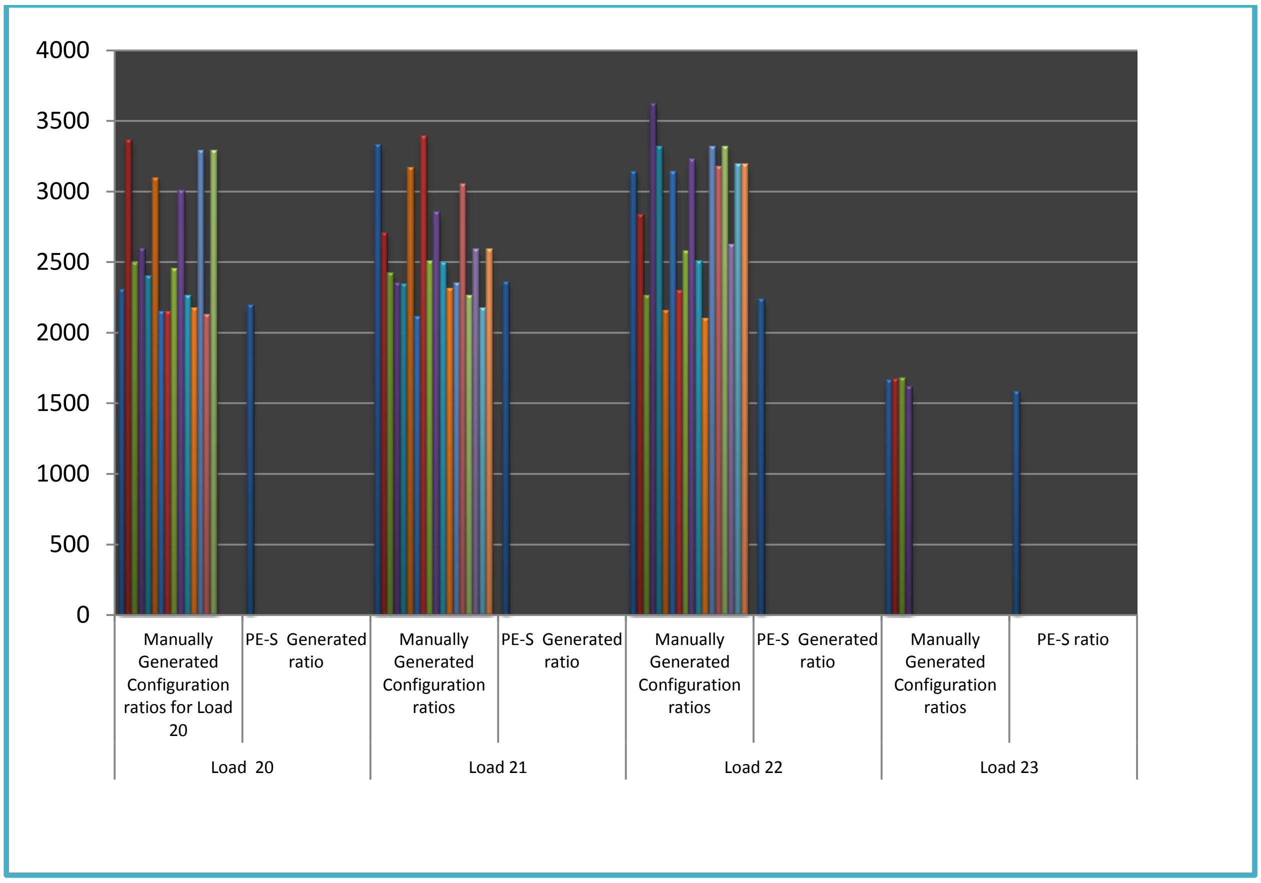

Fig. 3. : Comparison of PE-S generated reconfiguration ratio and possible manually generated ratios

5. RESULTS AND DISCUSSION

The results of the experimental simulation are described and analysed in this section. Multiprocessor model with and without PE-S simulation setups were used to evaluate the performance contribution of the PE-S load balancing technique using the nineteen randomly generated workloads shown in table 1a The results obtained were statistically analysed. The mean execution times obtained for model with and without PE-S were 1726.8 ± 421.5 seconds and 1742.0 ± 386.63 seconds respectively. This indicates that the use of PE-S technique improves the speed of the multiprocessor model with PE-S. This is further illustrated with fig. 2, where the execution times obtained for the model with and without PE-S for nineteen different workloads are graphically represented.

Also the possibility that the PE-S generated configuration ratios might not be perfect was experimented and evaluated. That is, the possibility of some other possible reconfiguration ratios to be more efficient in terms of reduced execution time than the one generated by PE-S. Four different workload instances in table 1b were assigned to the simulated multiprocessor model with PE-S. The PE-S reacted to these workloads by generating configuration ratios 6:4:3:2, 4:3:3:6, 2:5:4:4, and 5:6:9:5 for the four workload instances respectively. Each was used by the PE-S to reconfigure the multiprocessor model which then executed the corresponding workload instance in table 1b. Also, as a control, a few possible configuration ratios were manually generated for each of the workload instances and were used to reconfigure the multiprocessor model. The results are shown in fig.3.

The results obtained for the load number 20 show that only 4 out of fifteen manually generated reconfiguration ratios were better than PE-S generated ratio in term of execution time. This implied that with workload number 20, PE-S generated reconfigure ratio is 75% closer to the best reconfiguration. For workload number 21, the execution times of the eighteen manually generated reconfiguration ratios and the execution time of PE-S generated reconfiguration ratio were compared. It was observed that only 6 out of the 18 possible reconfiguration ratios have execution time better that PE-S generated reconfiguration ratio’s execution time. This indicated that for the workload number 21, PE-S reconfiguration ratio is 77% closer to the best reconfiguration ratio. Based on workload number 22, another eighteen possible manually generated reconfiguration ratios were manually generated. The execution time of the PE-S generated reconfiguration ratio was compared with execution times of the eighteen manually generated reconfiguration ratios. It was observed that only 1 out of the 18 possible reconfiguration ratios is better than PE-S generated ratio. This shows that the PE-S reconfigured HMPM is almost the best reconfiguration ratio. The results of the manually generated reconfiguration ratios and PE-S generated ratio for workload number 23 (SIMPLE, FFT, -, SPEECH) were also evaluated. Only 5 reconfiguration ratios were possible for workload 23. It was noted that none of the total execution time of the 5 reconfiguration ratios is better than PE-S generated configuration ratio’s execution time. This implied that the PE-S reconfiguration ratio is the best possible reconfiguration ratio for this workload instance.

6. CONCLUSION & RECOMMENDATION

In this paper the performance evaluation of PE-S load balancing technique for dynamic multiprocessor was carried out. This was done not only to ascertain the performance contribution of PE-S but to show that PE-S generated reconfiguration ratios are more efficient than manually generated reconfiguration ratios. It was observed from the results of the simulation that the proposed PE-S technique obviously reduced the effect of load imbalance in the multiprocessor by increasing the performance of multiprocessor. For few cases of better manually generated ratios the overheads incur through the manual generation will not only nullify the gain but worsen the performance. The proposed load balancing algorithm could be adapted to some other functional unit based optimisation problems.

7. ACKNOWLEDGMENTS

This work was supported in part by a scholarship grant from Petroleum Development Trust Fund (PTDF) Nigeria.

REFERENCES

[1] Acar, U., Blelloch, G., & Blumofe, R. (2000). The Data Locality of Work stealing. *Proceedings 12th ACM Symposium on Parallel Algorithms and Architecture* (pp. Pp. 1-12). ACM.

[2] Alexandra, F., Margo, S., & Michael, S. (2007). Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler. *Proceeding of 16th of International Conference on Parallel Architecture and Compilation Technique (PACT 2007).* Brasov, Romania.

[3] Amir, M. R., & Mohammad, A. V. (2008). A novel task scheduling in Multiprocessor Systems with Genetic Algorithm by Using Elitism stepping method.

[4] Belady, L. (1966). A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal* , 78-101.

[5] Berger, J., & Browne, J. (1999). Scalable Load Distribution and Load Balancing for Dynamic Parallel Programs. *International Workshop on Cluster-Based Computing.*

[6] Blumofe, R., & Leiserson, C. (1994). Scheduling Multithreaded Computations by Work Stealing. *Proceedings 35th IEEE Conference on Foundations of Computer Science*, (pp. 356-368).

[7] Chou, T. C., & Abraham, A. J. (1983). Load redistribution under failure in distributed systems. *IEEE Trans. Computing , C-32* (9), 799-808.

[8] Chow, C., & Kohler, W. (1979). Models for dynamic load balancing in a heterogenous multiple processor system. *IEEE Trans. Computing , C-28* (5), 354-361.

[9] Gautam, G., & Soo-Young, L. (1998). Dynamic Reconfiguration of a PMMLA for High Throughput Applications. *Parallel and Distributed Processing Workshops Held in Conjunction with the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing, 10IPPS/SPDP'98* (pp. Pp 1-6). Florida: Springer.

[10] Gene, E. J., & Hwang, Y. S. (2003). An Efficient Algorithm for Perfect Load Balancing on Hypercube Multiprocessors. *The Journal of Supercomputing , 25* (1), 5-15.

[11] Mitzenmatcher, M. (1998). Analysis of Load Stealing Models Based on Differential Equations. *Proceedings of 10th ACM Symposium on Parallel Algorithms and Architectures* (pp. 212-221). ACM.

[12] Neil, D., & Wierman, A. (2011). *On the Benefits of Work Stealing in Shared Memory Multiprocessors.* Carnegie Mellon University.

[13] Nozar, T., Nader, B., Amir, H. K., & Haitao, D. (2004). MaRS: A Macro-pipelined Reconfigurable System. *ACM* .

[14] Olakanmi, O., & Fakolujo, O. (2012(a)). Design and Performance Analysis of Reconfigurable Hybridized Macro Pipeline Multiprocessor. *International Journal of Ubiquitous Computing and Communication , 7*, Pp. 17-24.

[15] Olakanmi, O., & Fakolujo, O. (2012(b)). Load Balancing in the Macro Pipeline Multiprocessor System using Processing Element Stealing Technique. *International Journal of Ubiquitous Computing and Communication , 7*, Pp. 25-31.

[16] R, D., & et, a. (2002). A Dynamically Reconfigurable Architecture Dealing with Future Mobile Telecommunications Constraints. *Proceeding of Parallel and Distributed Processing Symposium, IPDPS*, (pp. 156-163).

[17] Wall, D. W. (1993). *Limits of Instruction-Level Parallelism.* Digital Western Research Laboratory 93/6.

[18] Yi-Hsuan, L., & Cheng, C. (n.d.). A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems.

[19] Zeljko, V., Havard, E., Palm, H., & Carsten, G. (2009). Limits of Work-Stealing Scheduling. In E. Frachtenberg (Ed.), *14th International WorkshopJSSPP*, (pp. 280-300).

AUTHOR'S PROFILE

O.O Olakanmi received the B.Tech in Computer Engineering from Ladoke Akintola University of Technology, Ogbomosho 2000 and M.sc in Computer Science from University of Ibadan, Ibadan. He is a lecturer and PhD student in the Department of Electrical & Electronic Engineering, University of Ibadan and major in Parallel & Distributed Computing..

O.A Fakolujo received the B.Sc in Electronic and Electrical Engineering from University of Ife now Obafemi Awolowo University 1980. He received the PhD in Electrical Materials from University of London in 1988. He is currently a reader in the Department of Electrical & Electronic Engineering, University of Ibadan.