Load Balancing in the Macro Pipeline Multiprocessor System using Processing Elements Stealing Technique

O. O Olakanmi¹ and O.A Fakolujo (Ph.D)²
Department of Electrical & Electronic Engineering
University of Ibadan
Nigeria
olakanmi.oladayo@mail.ui.edu.ng¹, daofakolujo@mail.ui.edu.ng²

ABSTRACT
The exponential increase in amount of processing power required by applications has been observed to be disproportionately leading the available processing power. Recent researches have shown that the key to balance this disproportionate is to shift the processor architecture towards multiprocessor or multicore architecture.

In spite of the high performance of multiprocessor architecture, load imbalance is one of the key factors which mar its optimal performance. This paper focuses on how load balancing can be enforced on multiprocessor architecture in order to execute problems that are susceptible to domain decomposition.

A technique for enforcing load balancing, called processing-elements-stealing, is proposed and a heuristic to implement this technique in multiprocessor architectures is developed. The objective of this paper is to show how processing element stealing technique can be used to prevent load imbalance in microprocessors architecture.

Keywords: Heuristic, Load balancing, Multiprocessor architecture, Processing-elements-stealing, Work stealing.

1. INTRODUCTION
Load balancing is one of the key techniques exploited to improve the performance of multiprocessor architecture. In this paper, processing-elements-stealing (PEs-S) technique is used to enforce load balancing in multiprocessor architecture that executes computer problems which are susceptible to domain decomposition. This domain of problems is typically approached using static partitioning within some grid computation [1][7]. Most times these problems are typically scientific or engineering applications where parallelism is used to exploit the local-based nature of physical problems, for example, encoding of images, data analysis, sorting and searching database.

The main goal of the parallel programmer is to balance the load of these computations across the processors in multiprocessor architecture. However, the programmer typically chooses a static partition of physical space onto the multiprocessor system. Thus, any load balancing performed by the programmer must be performed at a fairly coarse level - specifically, the programmer can only hope to balance the average loads of each processor [7]. Optimal performance however can only come by balancing the instantaneous load at the processor. Instantaneous load balancing can either be achieved in multiprocessor architecture by taking excess load from over loaded parallel unit(s) to under loaded unit(s) (referred to as work stealing) or by temporarily taking appropriate number of processing elements from idle parallel unit to overloaded parallel unit (processing element stealing). Either of these techniques increases the parallel unit(s) processing power thereby meet up with the required processing power of the assigned workload. However, work stealing introduces some overheads and jeopardizes scheduling queue affinity.

Instance of load imbalance can be imagined in multiprocessor architecture when different problems with domain decomposition are parallel assigned to different cluster in the architecture. Most time the input problems vary in terms of their required processing power, especially in a situation where job scheduling in the architecture is dynamic and each cluster at default has equal number of processing elements. There will be recurring instances of imbalance as the multiprocessor architecture is loaded with variant
workloads intermittently. The occurrences of load imbalance bring diminishing return to the performance of any multiprocessor architecture. Apart from the fact that programmers always find it difficult to create balance at the parallel programming level, many factors may introduce imbalance, for example, size of the data and number of iterations in processes may imbalance physically space balanced sub problems or problems. Therefore, a viable technique is needed to reduce load imbalance in multiprocessor architecture so as to enjoy its optimal performance.

In this paper, we address the problem of load imbalance in multiprocessor architecture. To solve this problem a load balancing technique is proposed and a heuristic which implement this technique is developed. The rest of the paper is organized as follows. Section 2 presents the related research works on load balancing in multiprocessor architecture. In Section 3, background of processing elements stealing technique is described. The heuristic is described in Section 4 and the test and conclusions are presented in Section 5.

2. RELATED WORKS

Work stealing and work sharing are the common techniques used to prevent load imbalance in multiprocessor architectures. In work sharing, whenever any processor attempts to generates new threads, such processor or scheduler will attempt to move some threads to idle processors. However, in work stealing idle processors attempts to take some threads from probably busy processors. Work sharing takes place before execution whereas work stealing is done during task execution. This makes work sharing less susceptible to queue or processor affinity than work stealing. Recent research works on load balancing in multiprocessor architecture are directed towards work stealing due to its dynamism; that is, work stealing prevent load imbalance during task execution.

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 affinity for such processor and during stealing priority is given to queues which have affinity for such processor.

In [3] an algorithm was proposed which implements work stealing technique to prevent load imbalance in 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 result shows that work stealing has lower communication cost than work sharing. Also, in [6] a differential equation was used to model work stealing technique. In [7] work stealing architectural technique is employed to prevent load imbalance in homogenous shared-memory multiprocessor architecture.

Work stealing technique involves monitoring the size of processors’ queues and joggle excessive workload among idle processors or lightly loaded processors. Computation work stealing allows the idle processor to take the excess job loads from overloaded processors’ queues. A common phenomenon can be seen when JPEG encoding application is pipeline at data level, there will be some instances when although equal numbers of images are assigned but some of the images in some clusters queue are large in terms of their sizes compare to those assigned to other clusters. Those clusters will be forced to perform more computation while other are left with very little computation. This load imbalance could not be prevented by programmer or architectural job scheduler. One important architectural technique commonly employed to accomplish this instantaneous load balancing is work stealing. However, as stated in [7] that some real world constraint complicates the issue. In particular, there is some cost associated with stealing a job from another processor's queue. First, there are the communication costs involved in transferring the job - these costs come from both the extra contention for the system bus and the latency of the transfer. Second is the fact that jobs have some affinity for the queue where they are assigned. This affinity results 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 we steal a task from another processor we can no longer take advantage of cached computation.

Considering all the overheads stated in [3], [6], and [7] one can easily conclude that both work stealing and work sharing have little or restricted usage domain. Although they added features that detect queue affinity, which not only introduce overhead but also not solve the problem. This brings the need for alternative technique of preventing load imbalance in multiprocessor architecture which is called processing-elements-stealing technique. This technique steals workers instead of assigned workload as is the case in work stealing technique. Meanwhile in order to understand the concept of processing-element-stealing-technique a heuristic employing this technique is developed and modeled in one of the multiprocessor
architectures called Hybridize Macro Pipeline Multiprocessor (HMPM) architecture. This multiprocessor architecture was proposed in [5] as shown in figure 1. HMPM proposes the methodology that combines both the macro pipeline and parallelism execution techniques in multiprocessor system. The HMPM architecture contains six clusters, which are abstracted as execution nodes for all the parallel tasks. In each cluster are four pipelined processing elements (PEs) to execute the assigned task in sequential fashion. The architecture is design to provide the novel solutions to many of on chip architectures mentioned in the [4], [8], [9], [10] and [11]. Some of the bottlenecks common in multiprocessor architecture are solved in the HMPM architectural design, for examples, methodology for coexisting the two multiprocessing techniques, reduction of memory access time, reduction of transmission overhead in the multiprocessor.

The HMPM consists of two operational layers; the first level of interconnection contains six clusters each representing parallel unit in massive parallelism multiprocessor architecture. The second level of interconnection cluster contains number of processing elements PEs, which uses macro pipeline to execute subtask. The intercluster and intercluster communication are performed through the on-chip inter and intra cluster buses. The six clusters in HMPM can be assigned six different tasks load in parallel. For example, two different images to be recognized using image recognition application with three processes viz; normalization, features extraction and recognition, and four images to be encoded using JPEG encoder application with six sequential process (RGB to YCbCr conversion, Level shift, Vertical DCT, horizontal DCT, Quantization, Huffman Encoding and Writing out JPEG file). The six loads are assigned in parallel to all the clusters in the architecture. In each cluster, all the processes in the task load are executed in pipeline.

3. PROCESSING ELEMENTS-STEALING TECHNIQUE (PES-S) IN MULTIPROCESSOR

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 stealing will mar the optimal performance as a result of the fact that stolen works will be transported from its initial processing element to the new processing element. Also some multiprocessing works have affinity to their assigned processor or cluster. In that case, any attempt to steal such works will disrupt the scheduling affinity. If this holds, it suggests a very interesting question raised in [4]: can a 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, work stealing introduces communication overhead which reduces the performance efficiency of the architecture.

In view of importance of load balancing to performance of multiprocessor architecture, another technique, called processing-element-stealing, is proposed to enforce load balancing. PEs-stealing technique will not only solve the problem of load imbalance but also will not affect queue affinity schedule and with little or no communication cost. Processing-elements-stealing will provide 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. In order to model PEs-stealing technique in HMPM, the architecture is split into modularized units called cluster. In each cluster, there is a specific number of processing elements arranged in pipeline pattern. The number of the processing elements for each cluster is not static; it depends on the workload assigned to the cluster. Whenever a cluster is over loaded, the PES-stealing heuristic senses this and reconfigures the architecture by reassigning more processing element(s) to that cluster. The technique 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.

4. HEURISTIC FOR IMPLEMENTING PES-STEALING TECHNIQUE IN HMPM ARCHITECTURE

To eschew load imbalance, using PEs-stealing technique, during reconfiguration of HMPM, the new configuration must satisfied both the new input workload and dependence constraints introduced. The
input dependence constraint would be taken care of by the partitioning controller because it is assumed that for HMPM critical path and dependence information would be stored with the application (during programming). However, no matter how perfect the scheduler or controller may be, changing the workload will fundamentally introduced imbalance in the multiprocessor system if reconfiguration of the architecture is not done. Possibilities of allocating big chunk of task to low capacity cluster may occur. The heuristic calculates the total number of processes in each workload assigned to the modularized unit of the multiprocessor system. This is used to get the current reconfiguration ratio $PE_i'$ of the architecture as shown in equation 3. The $PE_i'$ is compared with previous configuration ratio $PE_i$ and the processing elements are juggled (stolen) until the $PE_i = PE_i'$. The heuristic is highlighted below:

1. Reset the architecture to default.
2. Accept the parallel tasks (t1 ….tn) where n <= 6
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 PE’ while (Pcurrent_configuration_ratio > Pprevious_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

Given that,

- $Total\ number\ of\ processing\ elements\ in\ architecture\ before\ reconfiguration\ PE_i = \sum_{i=0}^{n} E_i$(1)
- $Total\ number\ of\ processing\ elements\ in\ architecture\ after\ reconfiguration\ PE_i' = \sum_{i=0}^{n} E_i'$(2)

Then,

$PE_i' = \frac{Total\ no\ of\ Process\ in\ task[i] \times (4\times\ No\ of\ Parallel\ task)}{\sum_{i=n}^{Total\ no\ of\ Process\ in\ task[i] + 4\times\ No\ of\ Parallel\ task)}$.........................................(3)

For all the clusters i (where i <= 6) the architecture performs these:

- if $PE_i' > PE_i$ then
  - $C_i E_{n+1} = C_{i+1} E_i$
  - i.e. borrow processing element from the next cluster and change its cluster status.
  - $PE_i' = PE_i - 1$
- else
  - $C_{i+1} E_{n+1} = C_{i+1} E_i$
  - i.e release excess processing element from the cluster to the next cluster and change its cluster status.
  - $PE_i' = PE_i + 1$

5. TEST AND CONCLUSION

The processing element stealing technique is modeled on HMPM architecture by subjecting the HMPM architecture to two different six parallel workloads in order to observe its implementation. The architecture in figure 1 is subjected to the first six-parallel workloads. The algorithm, using step 1 to 6, generates reconfiguration ratio $PE_i' = 5:3:2:6:5:3$, for each cluster respectively. The architecture detects load imbalance due to the variant number of processes in the assigned workloads. That is, cluster 2, cluster 3 and cluster 6 are lightly loaded and only need 3, 2, and 3 processing elements respectively as against the default number (4 PEs each). However, the cluster 1, 4 and 5 are overloaded with more tasks and needed more processing elements in order to upset load imbalance introduced by the variant workload.

The algorithm reacts to this reconfiguration ratio $PE_i'$ through step 6 to 9 by stealing their needed processing element(s) from the neighboring clusters until their $PE_i' = PE_i$ as shown in figure 2. The processing elements stealing technique prevents a situation whereby clusters 2, 3 and 6 would have been idle at the expense of other clusters thereby reducing the throughput of the architecture.

Also, the architecture was reset and subject to the second five-parallel workload. The algorithm through step 1 to 6 generates reconfiguration ratio $PE_i'$ =
This reconfiguration ratio shows that three clusters have more than require processing capacity for executing their assigned work at the expenses of the remaining clusters. That is, cluster 2, 3 need one more processing element each, while cluster 1 needs 2 more processing elements to avoid being overloaded. The reconfiguration ratio also shows that clusters 4 is not assigned workload, and cluster 5 and 6 workloads commensurate with the available processing elements. This state will create load imbalance in the architecture, Figure 3 is the new work load configuration which shows how processing elements stealing technique prevents load imbalance in order to maintain the architecture throughput.

In conclusion, it has been shown how processing elements stealing technique can be used to prevent load imbalance in multiprocessor architecture. A modeled heuristic which detects load imbalance and implements processing element stealing technique in HMPM has been used to evaluate the practicability of this technique. The graphical results in figure 1, 2 and 3 shows that the technique corrects load imbalance in the architecture.

ACKNOWLEDGEMENTS
This work is supported in part by Petroleum Technology Development Fund (PTDF) in Nigeria.

6. REFERENCES


Fig. 1: Default Architecture of Hybridized Macro Pipeline

Fig. 2: Architecture of HMPM showing implementation of Processing-Stealing Technique after subjecting to workloads of 5:6:3:5:2:3 reconfiguration ratios
Fig. 1: Architecture of HMPM showing implementation of Processing-Stealing Technique after subjecting to workloads of 6:5:5:0:4:4 reconfiguration ratios

Legends

- Cluster 1 Processing element
- Cluster 2 Processing element
- Cluster 3 Processing element
- Cluster 4 Processing element
- Cluster 5 Processing element
- Cluster 6 Processing element
- On Chip Inter Cluster Channel
- On Chip Intra Cluster Channel
- Level 1 Cache
- Level 2 Cache