Load Balancing Through Task Shifting and Task Splitting Strategies in Multi-core Environment

Hameed Hussain1, Sadiq Shah2, Muhammad Shoaib3, Muhammad Bilal Qureshi4
1COMSATS Institute of Information Technology, Islamabad
2School of Electrical Engineering and Computer Sciences (SEECS) National University of Sciences and Technology, Islamabad
3University of Engineering and Technology, Peshawar
4COMSATS Institute of Information Technology (CIIT) Islamabad, Pakistan
ham_hamdard@yahoo.com, 08msitsshah@seecs.nust.edu.pk, eshoaibkhan@gmail.com

ABSTRACT: Load balancing among cores in High Performance Computing (HPC) is the demand of the day. Efficient results may not be obtained unless a specific load is properly balanced among systems or cores in HPC. This work is focusing on task migration or task shifting and task splitting strategies for load balancing among cores in multi-core environment. It concludes from the given results that task splitting mechanism fully balance a specific load among cores but it is more time consuming as compared to task shifting strategy.

Keywords: HPC, Multi-core, Load balancing, Task migration, Task splitting

Received: 2 March 2014, Revised 8 April 2014, Accepted 13 April 2014

© 2014 DLINE. All Rights Reserved

1. Introduction

Efficient resource management and load balancing among systems is a key and fundamental requirement to the success of any HPC environment like cluster, grid cloud and multi-core systems [1]. The main aim of Cluster Computing system is to design an efficient computing platform that uses a group of computer resources integrated through hardware, networks, and software to improve the performance and availability of a single computer resource. The concept of Grid Computing system is based on using the Internet as a medium for the wide spread availability of powerful computing resources [2]. Computational grid can be thought of as a distributed system of logically coupled local clusters with workloads that involve a large number of files [2]. The concept of Cloud Computing system describes a model for services based on the Internet, and it typically involves over-the-Internet provision of dynamically scalable and often virtualized resources [3, 4]. It is the ease-of-access to remote computing sites provided by the Internet [5]. This frequently takes the form of web-based tools or applications that users can access and use through a web browser [6].

Another trend in HPC is increasing cores to the same chip which is known as multi-core systems. There are dedicated machines and software for resource allocation in cluster, grid and cloud systems. In multi-core systems operating system is
responsible for resource allocation. Normally cores runs symmetrically [7] but it is not necessary that cores runs symmetrically [8, 9]. When cores run asymmetrically there is an issue of balancing load and performance. There are two solutions of asymmetric cores running. The first one is; add dynamic voltage circuitry which is a hardware solution. The second choice is schedule tasks among cores to operate on same clock frequency (speed) which is a software solution. The main disadvantage of the hardware solution is the power leakage at higher frequency [8] so the software solution (task scheduling) is a promising alternative for maintaining cores speed and balancing performance. Being promising alternative, the software solution is relatively unexplored from the point of view of scheduling. Considering this gap, partition of a given workload among all the cores is done for load balancing with the intention that all the cores operate on the same clock frequency for maximum energy savings.

The paper only focuses on one dimension of HPC systems which is multi-core and applies the load balancing strategies such as task migration and task splitting for load balancing. The ideas of the aforementioned strategies can easily be extended to distributed HPC systems (Cluster, Grid and Cloud).

The rest of the paper is organised as follows: section II gives a quick overview of existing load balancing techniques. Section III describes the task migration and task splitting strategies used for load balancing. Section IV is devoted to results and discussions. Section V concludes the work and the last section gives some future directions of the work.

2. Literature Review

There is huge amount of literature on load balancing. All the load balancing strategies can be broadly categorised into two types: one is called static [10] load balancing and the second one is called dynamic [11, 12] load balancing. Static load balancing has statistical information of application and uses it for load balancing. Dynamic load balancing mechanisms only uses the current state of the system for load balancing. There are three ways for the solutions of static load balancing problem. The three ways are global, cooperative and non-cooperative. Our approach is somehow hybrid in between the static and dynamic approach and for load balancing solution, we are using global approach i.e. we are interested in the overall load balancing of a system.

Kameda et al. [13] developed some algorithms for load balancing in non-cooperative games. For a single class the static load balancing problem is formulated by D. Grosu et al. [14]. Andrey G et al. [15] shows a gradient decent algorithm for load balancing. Some of the variations of the gradient decent algorithm are found in [16, 17, 18]. Our strategy of lightest task migration is somehow resembles to the gradient decent algorithm.

3. Load Balancing Mechanisms

In this section two mechanisms are discussed for load balancing. The first one is task migration or task shifting and the second one is task splitting strategy. Before applying these load balancing strategies; first tasks from task set are assigned to different cores such that the tasks are feasible on that cores. The task set \( \Gamma = \{\tau_1, \tau_2, ..., \tau_n\} \) consists of \( n \) tasks and can be divided into subsets such that \( \Omega = \{\Omega_1, \Omega_2, ..., \Omega_n\} \), where \( \Omega_1 = \{\tau_1, \tau_2, ..., \tau_k\} \) and \( \Omega_2 = \{\tau_{k+1}, \tau_{k+2}, ..., \tau_l\} \) and so on.

Moreover, a set of cores \( \Delta = \{\Delta_1, \Delta_2, ..., \Delta_m\} \), \( m \leq n \) is available. The individual task \( \tau_j \) utilization at a specific core is given as

\[
U(\Delta) = \frac{C_i}{P_i}
\]

Where \( C_i \) is the execution time needed by a task and the \( P_i \) is the period of the task. The problem that we are addressing is to map \( \Omega \) over \( \Delta \) such that tasks are feasible on each core and then balance the load among all cores \( P_i \) through task shifting and task splitting strategies. For feasibility checking first, cumulative work load of task \( \tau_i \) is calculated through the following equation number (1) at any time instance \( t \). After calculating cumulative work load feasibility is checked through equation number (2).

\[
L_i(t) = C_i + \sum_{j=1}^{i-1} \frac{t}{P_j} (C_j) \quad (1)
\]

A task \( \tau_i \) is always feasible on a generic core \( \Delta_i \) at any instance of time \( t \) if and only if

\[
\min_{t: \tau_i \in \Delta_i} L_i(t) \leq t \quad (2)
\]
On each scheduling point cumulative work load is calculated and if the work load is feasible the task is assign to a core and if it is not feasible then the task is assigned to another core.

3.1 Task migration or task shifting
After assigning tasks to cores the next step is to balance the load among cores. Task migration or task shifting is one strategy used for load balancing. In this strategy a core having maximum utilization i.e. having maximum load is selected and on that core a task having minimum utilization is selected for shifting. Task utilization is calculated through:

\[ U_i(\tau_i) = \frac{C_i}{P_i} \]  

A task having low utilization is shifted from a highly utilized core to a low a utilized core and the process is repeated until utilization of all cores becomes approximately equal to the average utilization of all cores. Through this strategy the utilization of all cores did not becomes exactly equal because the \( C_i \) and \( P_i \) parameters are different for various tasks, therefore the utilization of the tasks are also different that leads to unequal cores utilization even after shifting lightest tasks. The lightest task is selected for shifting to balance the load among cores gradually. If a task having maximum utilization is selected for shifting then there may be greater fluctuation of balancing load among cores i.e. the load on cores will quickly increase and decrease from the average utilization.

3.2 Task splitting
This is another strategy used for load balancing among cores. Task shifting strategy did not guarantee equal load among all cores as compared to task splitting. In task splitting strategy \( C_i \) is the only parameter to play with, because the parameter \( P_i \) is constant and we can not change it. Task splitting strategy balance load among all cores by splitting the \( C_i \) parameter into two parts in such a way that cores utilization becomes equal after assigning one part to one core and the second part to another core. Task splitting strategy is more time consuming as compared to task shifting policy because in task splitting extra time is required to split a task into two parts and then transfer a part of the split task to another core for balancing load.

Task splitting strategy can be implemented in two ways:

i) Assign tasks to cores and apply task splitting strategy directly.

![Figure 1. Number of tasks on cores before load balancing](image)
ii) First apply the task shifting strategy and then apply task splitting. The second way of implementation is less time consuming as compared to the first because after applying task shifting strategy the cores are approximately balance as compared to the first choice. We apply the second way of implementation for task shifting. The results of task shifting and task splitting are given under the results and discussions heading.

4. Results and Discussions

This section shows the results of the strategies discussed in previous section. Matlab is used for simulation of the above discussed strategies. A task set of 60 tasks is generated, mapping it to four cores and then the discussed strategies are applied. Figure 1 show tasks assigned to cores, before applying any of the load balancing strategy. It shows that before load balancing core 3 has maximum tasks which are 19 and core 2 and core 4 has 13 numbers of tasks and remaining tasks are assigned to core 1 as depicted in Figure 1.
Figure 2 depicts utilization of cores corresponding to the tasks assigned in Figure 1. It is clear from Figure 2 that core 3 has maximum utilization which is 0.8149 and core 4 has minimum utilization which is 0.6704 before load balancing.

Figure 3 shows the number of tasks on the four cores after applying task migration or task shifting strategy. Figure 3 depicts that 7 tasks are shifted to core 4 from core 1 and core 3 so now core 4 has maximum tasks which is 20. It should be clear that in task shifting the lightest task is shifted from a high utilized core to a low utilized core. Core 2 also gains 2 tasks from core 1 and core 3. After shifting tasks core 3 has minimum task numbers which is 12 as depicted in Figure 3.

Figure 4 depicts utilization of cores corresponding to the number of tasks in Figure 3. It is clear from the following Figure 4 that after applying task shifting strategy all the cores utilization is not fully equal but approximately equal.

In Figure 4 core 1 has minimum utilization as compared to other cores which is 0.7382 and core 2 has maximum utilization which is 0.7550.
In order to make the utilization of all cores equal, task splitting strategy is applied. The result of task splitting is depicted in Figure 5 and is clear from the figure that all cores have now equal utilization which is 0.7453.

Although task splitting leads to equal cores utilization but is more time consuming than task shifting strategy.

5. Conclusion

In this paper authors presented two strategies for load balancing among cores or systems in HPC environment. The first strategy for load balancing is task migration. In Task migration a lightest task is transferred from a highly utilized core to a low utilized core and the process is repeated unless the load among cores is approximately balanced. The other strategy is task splitting. In task splitting strategy cores are fully balanced by splitting a task i.e. the execution time of a single task is divided among high utilized core and a low utilized core in such a way that cores utilization becomes fully balanced. As compared to task migration, task splitting strategy fully balances a specific load among cores but it is more time consuming than task migration, because it takes extra time in splitting a task execution time in such a way to balance load among cores.

6. Future Work

In the future it is decided to extend these concepts to distributed HPC systems where a lot of new issues come like delay time in transferring a task and so on. More interesting results will be obtained by incorporating the delay time in these concepts in distributed HPC environment. A distributed HPC environment is more challenging than multi-core, while implementing these concepts in distributed HPC environment, may be some more interesting research topic and issue will appear.

References


