International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

A Survey on Dynamic Scheduling Method for Heterogeneous, Multi-Core Processors

Dr. S. Sai Satyanarayana Reddy[1], Debashree Rupalin[2], K. Sravanthi[3]

[1]Professor, Vardhaman College Of Engineering, Department of Computer Science and Engineering, Shamshabad, Hyderabad, Telangana, India -501218

[2]Assistant Professor, Vardhaman College Of Engineering, Department of Electronics and Communication Engineering, Shamshabad, Hyderabad Telangana, India -501218

[3]Assistant Professor, Vardhaman College Of Engineering, Department of Electronics and Communication Engineering, Shamshabad, Hyderabad, Telangana, India -501218

Abstract: ***Heterogeneous system create unlimited opportunities and challenges in the field of parallel processing or parallel computing for design of algorithms, partitioning and mapping of parallel tasks where scheduling plays an important role. In the other hand multi core processors /multi core technologies is offering a great potentials for computational power for scientific and industrial applications, however another challenge for software development. In this work, we provide a survey on current multi core technology and threaded building blocks. The goal of scheduling is to utilize all the processors with minimum execution time by proper allocation of tasks to the processors. Scheduling of task gives the high performance in heterogeneous system. We motivate the necessity of dynamic scheduling and summarize the challenges arising from high performance heterogeneous computing. Because of two main factors such as performance and power consumption, Heterogeneous multi core processors (HMP) are better to schedule the job as compare to homogeneous multi core processors.***

Keywords: Multi core processors; Heterogeneous systems; Task Scheduling and Load balancing; Threaded building blocks

1. Introduction

Heterogeneous multi core processor which may expose a common ISA (Instruction set architecture) but different in size, feature, performance and energy consumption. It has potential to reduce the power consumption and improve the performance. In the architecture individual cores have different computational capabilities though architecture consists of a combination of small and big cores. Another challenge that it opens up the possibilities for thread scheduling , load balancing and energy management.

The HMP decreases the frequency of the processors which reduce the temperature of the system. In these processors the amount of parallelism increased because there is a simultaneous execution of instructions on individual cores. The scheduler can dynamically select the relevant core to fulfill power and performance requirements. In general small cores provide good performance for compute intensive workloads.

2. Multi Core Processor

A multi core processor is one which combines two or more independent processors into a single package often a single integrated circuit.
An independent processor = a core
So, More cores = better performance
Multiple cores are made to work in parallel to achieve better performance.

It has a lot of advantages such as,

a) Performance upgrade, it means if each core is working at say 2.5 GHz, then a dual core processor can work at speeds from 4-4.5GHz,which is never achievable using a single core processor.
b) Low power usage and overheating issues.
c) Multi core processor gave rise to multi core programming which is said to be an important leap in software development.

2.1 Structure of a Multi-Core Processor

A simple multi-core consists of 2 independent working processors. Each thread is assigned to each and individual respective cores. All the cores use a common shared L2 Cache which is connected to main memory (RAM).

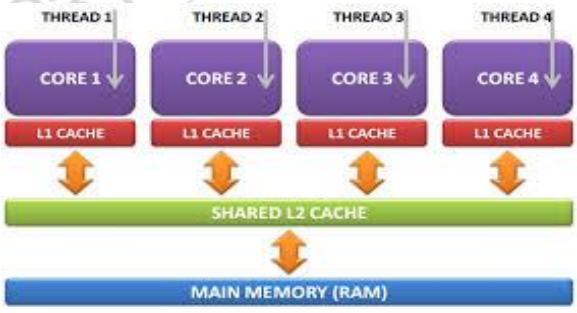

Figure 1: Multi core processor

3. Heterogeneous System

A Heterogeneous computing system consists of a number of autonomous and independently scheduled heterogeneous computers. A primary objective in many research projects dealing with heterogeneous computing is the minimization of the job completion time. It consists of applications running on a platform that has more than one computational unit with different architectures, such as a multi-core CPU and a many

core GPU. A Heterogeneous system involves multiple heterogeneous modules that interact with one another to solve a problem. In a heterogeneous system applications have sub tasks that have diverse execution requirements. The subtasks must be assigned to machines (processors) and ordered for execution such that overall application execution time is minimized.

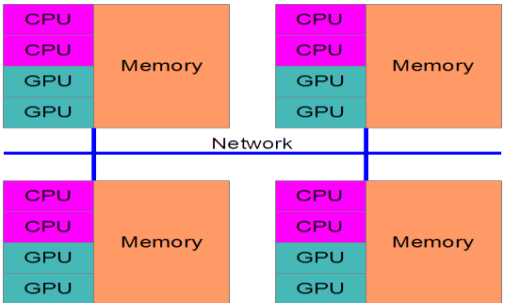

Figure 2: Heterogeneous system architecture

4. Task Scheduling and Load Balancing

4.1 Task Scheduling

When a large multi user cluster needs to access very large amount of data, task scheduling becomes a challenge. In a heterogeneous CPU-GPU cluster with a complex application environment, the performance of each job depends on the characteristics of the underlying cluster. Therefore mapping tasks onto CPU cores and GPU devices provides significant challenges. This is an area of ongoing research.

The two key concepts involves in scheduling a task are triggers and actions. A trigger causes a task to run and an action is the work that is performed when the task is run.

In dynamic scheduling, tasks are allocated to processors upon their arrival and scheduling decisions must be made at runtime. Scheduling decisions are based on dynamic parameters that may change during run-time. In dynamic scheduling tasks can be reallocated to other processors during the run time. The problem of scheduling is a weighted directed acyclic graph (DAG) also called a task graph or macro dataflow graph. The objective of this study include proposing a set of benchmarks and using them to evaluate the performance of a set of DAG scheduling algorithms with various parameters and performance measures.

- **Performance measurement :** The performance of a DSA(DAG scheduling algorithm) is usually measured in terms of the quality of the schedule(total duration of the schedule) and the running time of scheduling algorithm. Sometimes the number of target processors allocated is also taken as a performance parameter.
- **Use of benchmarks:** There does not exist any set of benchmarks that can be considered as a standard to evaluate and compare various DSAs on a unified basis. The most common practice is to use random graphs. The use of task graphs derived from various parallel application. However, in both cases there is again no standard that can provide a robust set of test cases. Therefore, there is a need for a set of benchmarks that are representative of various types of synthetic and real test cases.

4.1.1 DAG Model

An example of Directed Acyclic Graph G = (V, E) where V is a set of v nodes or vertices and E is a set of directed edges. Source node of an edge is called parent node and sink node is called child node

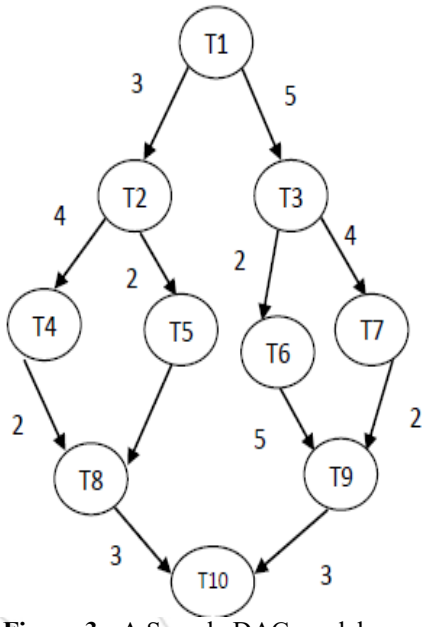

Figure 3 : A Sample DAG model

A node with no parent is called entry node. A node with no child is called exit node. A sample DAG model is shown in Figure 4 which contains T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9 and T_{10} tasks. T_1 is an entry node which has no parent node and T_9 is an exit node , with no child node. T_2 is a parent node of T_4 and T_5.

4.2 Load Balancing

Our research in load balancing focuses on two primary areas : object migration and seed balancing.

Object migration

- Periodic load balancing for bipartite object networks.
- Adaptive use of workstations clusters.
- Optimal object migration to handle background load variation.

A major reason showing the deployment of parallel programs is that efficient parallel programs are difficult to write. A vast number of parallelizable applications do not have a regular structure for efficient parallelization. Such application require load balancing to perform efficiently in parallel. The load in these applications may change over time, requiring rebalancing.

Seed Load Balancing

Seed load balancing involves the movement of object creation messages or seeds to create a balance of work across a set of processors. Several variation of strategies are being analyzed. Some strategies use averaging of loads to determine how seeds should be distributed, while others use receiver initiated strategies, where a processor requests work from elsewhere when it is about to go idle. A strategy that places seeds randomly when they are created and does no

movement of seeds thereafter is used as a baseline for comparison on numerous benchmarks.

5. Threaded Building Blocks

Threading building blocks offers a rich and complete approach to expressing parallelism in a C++ programs. It is a library that helps the multi core processors for performance and scalability without having to be a threading expert. TBB implements work stealing to balance a parallel work load available processing cores in order to increase core utilization and therefore scaling. Initially, the workload is evenly divided among the available processor cores. If one core completes its work while other cores still have a significant amount of work in their queue. TBB reassigns some of the work from one of the busy cores to the idly core. This dynamic capability decouples the programmer from the machine, allowing application written using the library to scale to utilize the available processing cores with no changes to the source code or the executable file.
TBB is a C++ library developed by Intel for making use of multi core processors.

It has some key features as follows,

- It relies on generic programming to deliver high performance parallel algorithms with broad applicability.
- It provides a high level abstraction for parallelism.
- It facilitates scalable performance which strives for efficient use of cache and balances load.
- It can be used in concert with other packages such as native threads and OpenMp.

TBB parallel algorithms map tasks onto threads automatically. Task scheduler manages the thread pool. It is unfair to favor tasks that have been most recent in the cache. Oversubscription and under subscription of core recourses is prevented by task-stealing technique TBB scheduler.

6. Generic parallel Algorithms

1. Loop parallelization: -
Parallel_ for and **parallel_ reduce** are used as a load balanced parallel execution of fixed number of independent loop iterations.
Parallel_ for partitions original range into sub ranges and deals out sub ranges to worker threads in a way that which balances the load, scales and uses cache efficiently.
Parallel_ Scan is a template function that computes parallel prefix (y[i] = y[i-1]).

A processor is a container for threads, as far as the OS scheduler is concerned. Each process has at least one thread. When there are multiple threads in a single process, there may be extra rules on which threads get scheduled. Other than that running multiple processes and running multiple threads are mostly the same thing: after each time slice, the OS scheduler determines the next thread to run for each CPU core and switches the context to the thread.

7. Conclusion and Future Scope

- Tasks on different micro-architectures will be assigned.
- Traces can be generated by considering different micro-architectures, profiling and performance counters.
- Scheduling algorithms will be designed and implemented.

References

[1] V. Jimi´enez, L. Vilanova, I. Gelado, M. Gil, G. Fursin, and N. Navarro. Predictive runtime code scheduling for heterogeneous architectures. In Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers, pages 19–33. Springer, 2008.
[2] F. A. Bower, et al. "The Impact of Dynamically Heterogeneous Multicore Processors on Thread Scheduling", IEEE Micro, pp 17-25, May 2008.
[3] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. In Proceedings of the 32nd Annual International Symposium on Computer Architecture (Madison, Wisconsin USA, June 04–08, 2005). ISCA ‘05. IEEE Computer Society, Washington, DC, USA, 506–517.
[4] M. Becchi and P. Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In Proceedings of the 3rd Conference on Computing Frontiers (Ischia, Italy, May 02–05, 2006). Computing Frontiers ‘06. ACM, New York, NY, USA, 29– 40.
[5] K. Hoste and L. Eeckhout. Microarchitecture-Independent Workload Characterization. IEEE Micro, 27(3), 2007. IEEE Computer Society Press, Los Alamitos, CA, USA, 63–72.
[6] R. Kumar et al. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In Proceedings of the 31st Annual International Symposium on Computer Architecture (München, Germany, June 19–23, 2004). ISCA ‘04. IEEE Computer Society, Washington, DC, USA, 64.
[7] D. Shelepov and A. Fedorova. Scheduling on Heterogeneous Multicore Processors Using Architectural Signatures. In Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with the 35th International Symposium on Computer Architecture (Beijing, China, June 21–25, 2008). WIOSCA ‘08.
[8] K. Asanovic et al. The Landscape of Parallel Computing Research: A View from Berkeley. UC Berkeley Technical Report UCB/EECS- 2006-183, 2006.
[9] M.Maheswaran and H.J.Siegel, “A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems”, Proc. Heterogeneous Computing Workshop, pp. 57-69, 1998.

Author Profile

Dr. S. Sai Satyanarayana Reddy, Principal and Professor in the Department of Computer Science and Engineering, B.E in Civil Engineering, M.E in Systems and Information from BITS, Pilani and PhD in Data Warehousing area, Having 22 years of teaching experience. He was Principal Investigator of 3 projects of AICTE, DST and published 20 papers in international journals, attended 10 international conferences and recipient of Vidya Rathna Award.