AGENT BASED TWO BUFFER HIERARCHICAL SCHEDULING ALGORITHM FOR MULTICORE ARCHITECTURE

G.Muneeswari[1] and E.M.Malathy[2]

[1]Associate Professor, SSN College of Engineering
muneeswarig@ssn.edu.in

[2]Assistant Professor, SSN College of Engineering
malathyem@ssn.edu.in

ABSTRACT

In the current era, we have moved from multiprocessor system to multicore system. The main travel towards multicore system is the tremendous performance achievement over processor execution. Although there are many hardware challenges imposed on this architecture, the software design also brought into the attention of designing efficient operating system, building intelligent compiler etc., Though many processor scheduling algorithms are developed, keeping all the cores in the active state is a major challenge which conventional algorithms do not implement. In this paper we propose a new agent based two buffer hierarchical scheduling algorithm that enhances the performance of the processor by 20% compared with the traditional algorithms. There are two levels in the overall design wherein in the first level all the tasks are assigned with equal priority and a buffering method is implemented. Whereas in the second level, we consider the real time task and affinity based scheduling is incorporated. For the evaluation results modified linux 2.6.21 kernel along with the FLAME tool is used. Ultimately, the overall results proves that this agent based algorithm outperforms in cpu performance and reduces average waiting time of the process by 4.5% compared with the conventional scheduling algorithms.

Keywords: *agent, buffer, hierarchical scheduling, multicore, affinity*

1. Introduction

Multicore architectures, consists of several processors on a single chip [12], are being widely used as a solution to sequential execution problems which is a major disadvantage of a single-core designs. In every multicore platforms, different cores share the common centralized memory. Conventional schedulers mainly concentrate on placing jobs on cpu units by assigning each core a job / process to run. Schedulers must be designed in such a way that it ensures high utilization of the processor and reduces the idleness of processors. In homogeneous multicore systems shown in figure.1, every core has the similar computing power based on the clock speed. The major challenge and real difficulty of such system lies in the efficient scheduling since the cores performance related to the computing power must be utilized. Schedulers in today's operating systems have the primary goal of keeping all cores busy by executing some process from the memory.

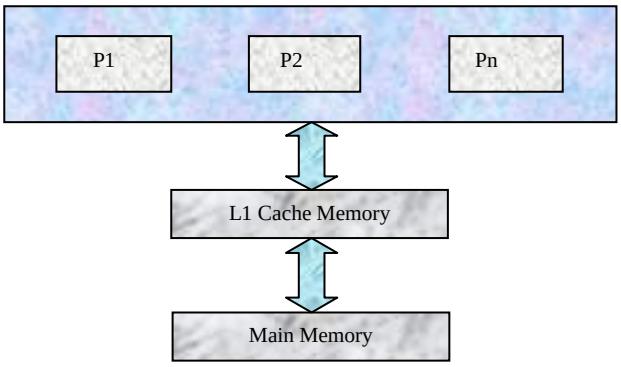


Figure 1: Multicore System Architecture

The paper [10] discusses several different mechanisms for generating self-organization in multi-agent systems [11]. A theory has been proposed (called AMAS for Adaptive Multi-Agent Systems) in which cooperation is the engine thanks to which the system self-organizes for adapting to changes coming from its environment. Cooperation in this context is defined by three meta-rules: (1) perceived signals are understood without ambiguity, (2) received information is useful for the agent's reasoning, and (3) reasoning leads to useful actions toward other agents. Interactions between agents of the system depend only on the local view they have and their ability to cooperate with each other.

The paper is organized as follows. Section 2 reviews related work on scheduling. In Section 3 we introduce the buffering method with equal priority for multicore architecture. In section 4 we describe the affinity scheduling for real time tasks which consists only the hard affinity scheduling. In section 5 we discuss the evaluation and results and finally, section 6 concludes the paper.

2. Background and Related Work

The research on contention for shared resources [1] significantly impedes the efficient operation of multicore systems has provided new methods for mitigating contention via scheduling algorithms. Addressing shared resource contention in multicore processors via scheduling [2] investigate how and to what extent contention for shared resource can be mitigated via thread scheduling. The research on the design and implementation of a cache-aware multicore real-time scheduler [3] discusses the memory limitations for real time systems. The paper on AMPS [4] presents, an operating system scheduler that efficiently supports both SMP-and NUMA-style performance-asymmetric architectures. AMPS contains three components: asymmetry-aware load balancing, faster-core-first scheduling, and NUMA-aware migration. In Partitioned Fixed-Priority Preemptive Scheduling [5], the problem of scheduling periodic real-time tasks on multicore processors is considered. Specifically, they focus on the partitioned (static binding) approach, which statically allocates each task to one processing core. In the traditional multi processor system, the critical load balancing task is performed through hardware.

In [6] The cooperative load balancing in distributed systems is achieved through processor interaction. dynamic load balancing algorithm [7] deals with many important issues: load estimation, load levels comparison, performance indices, system stability, amount of information exchanged among nodes, job resource requirements estimation, job's selection for transfer, remote nodes selection. In ACO algorithm [8] for load balancing in distributed systems will be presented. This

algorithm is fully distributed in which information is dynamically updated at each ant movement. The real-time scheduling on multicore platforms [9] is a well-studied problem in the literature. The scheduling algorithms developed for these problems are classified as partitioned (static binding) and global (dynamic binding) approaches, with each category having its own merits and de-merits. So far we have analyzed some of the multicore scheduling approaches. Now we briefly describe the self-organization of multiagents, which plays a vital role in our multicore scheduling algorithm.

The Cache-Fair Thread Scheduling [14] algorithm reduces the effects of unequal cpu cache sharing that occur on the many core processors and cause unfair cpu sharing, priority inversion, and inadequate cpu accounting. The multiprocessor scheduling to minimize flow time with resource augmentation algorithm [15] just allocates each incoming job to a random machine algorithm which is constant competitive for minimizing flow time with arbitrarily small resource augmentation. In parallel task scheduling [16] mechanism, it was addressed that the opposite issue of whether tasks can be encouraged to be co-scheduled. For example, they tried to co-schedule a set of tasks that share a common working were each 1/2 and perfect parallelism ensured. The effectiveness of multicore scheduling [17] is analyzed using performance counters and they proved the impact of scheduling decisions on dynamic task performance. Performance behavior is analyzed utilizing support workloads from SPECWeb 2005 on a multicore hardware platform with an Apache web server. The real-time scheduling on multicore platforms [18] is a well-studied problem in the literature. The scheduling algorithms developed for these problems are classified as partitioned (static binding) and global (dynamic binding) approaches, with each category having its own merits and de-merits. A new approach for multiagent based scheduling [12] for multicore architecture and load balancing using agent based scheduling [13] have improved cpu utilization and reduces average waiting time of the processes.

3. Buffering Method with Equal Priority

Before starting the process execution, the operating system scheduler selects the processes from the ready queue based on the first come first served order. Each process in the centralized queue has an associated process id, burst time, arrival time etc., Here in this first level, we do not discriminate the process as critical tasks or non critical tasks instead all the process will be assigned with equal priority. The main idea of reducing the average waiting time of the process in the ready queue is to use an extra buffer for placing the ready process. We are actually reducing the time of memory access. By this method whichever process that comes first to the ready queue will be given to the buffer and depends on the size of the buffer say for example 'n', the number of tasks are placed. Scheduler selects the process from the buffer and assigns to the middle agent (shown in figure2). Now it is the responsibility of the middle agent to allocate the process to the respective processor.

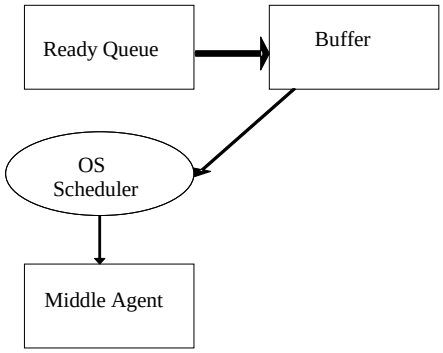
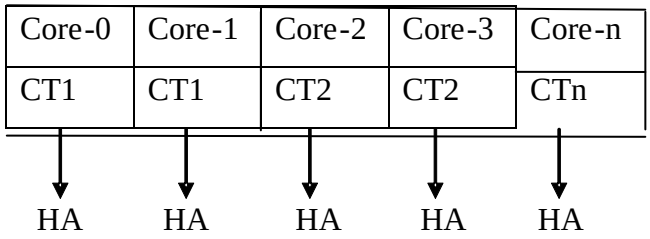


Figure 2 : Buffering method for Process Allocation

Actually the use of buffer is an additional cost for the overall design but the tremendous improvement over the performance in the processor execution and the reduction in the average waiting time compromises the hardware design and the cost. In the second level of scheduling we consider the real time tasks by allocating the process in the affinity based scheduling.

4. Affinity based Scheduling for Real Time Tasks

The scheduler [19], after selecting certain processes from the ready queue, places in the middle agent. The middle agent is implemented as a queue data structure. Middle agent holds M tasks which is greater than N (Number of processor cores). Precisely, the middle agent is acting as a storage space for faster scheduling. In Figure 3, all the critical tasks are assigned with hard affinity. It is very important that the critical tasks are not preempted after the time quantum expires. Most of the critical tasks are real time tasks, and it is not desirable to context switch after the system call or I/O operation.



HA – Hard Affinity CT-Critical Task(Real Time Task)

Figure 3: Middle Agent Queue Implementation

Processor affinity is maintained only for critical tasks. The conventional round robin scheduling algorithm is employed. During the context switching time, if it is not a critical task, then it can be allocated to the idle processor to improve the overall efficiency (no resource contention). But, if it is a critical task, it should not be context switched and it has to be executed for its full burst time. If it is a critical task, then agent will assign the process to the same processor.

Otherwise, it will assign the process to the idle processor. Initially all the jobs from the ready queue are selected by the OS scheduler on FCFS basis and then all the selected processes are placed in the middle agent. The individual agent of every processor selects the job from the middle agent queue and assign it to the processor. This agent actually eliminates the job of the dispatcher. Threads restricted by a hard affinity mask will not run on processors that are not included in the affinity mask. Hard affinity used with scheduling can improve performance of a multicore processor system substantially. However, hard affinity[19] might cause the processors to have uneven loads. If processes that have had their affinity set to a specific processor are causing high CPU utilization on that processor while other processors on the system have excess processing capacity, the processes for which a hard affinity has been set might run slower because they cannot use the other processors.

In the overall design, the programmer can prescribe their own affinities and that will be termed to be hard affinities. The tag field of the critical tasks consists of high priority and affinity to the processor. Every processor has a dedicated processor agent and that is responsible for maintaining agent control block (ACB). This agent control block will be useful to identify the free idle processor for next scheduling. In the case of critical tasks, since it is not preempted, it is not mandatory to establish an agent control block.

5. Evaluation Results

The execution time for various scheduling algorithms are shown in Figure 4. we are keeping the number of cores=100 as an assumption. From the simulation study, it is found out that E5 is the agent based scheduling algorithm and its total execution time almost 50% equal or similar to other scheduling algorithms.

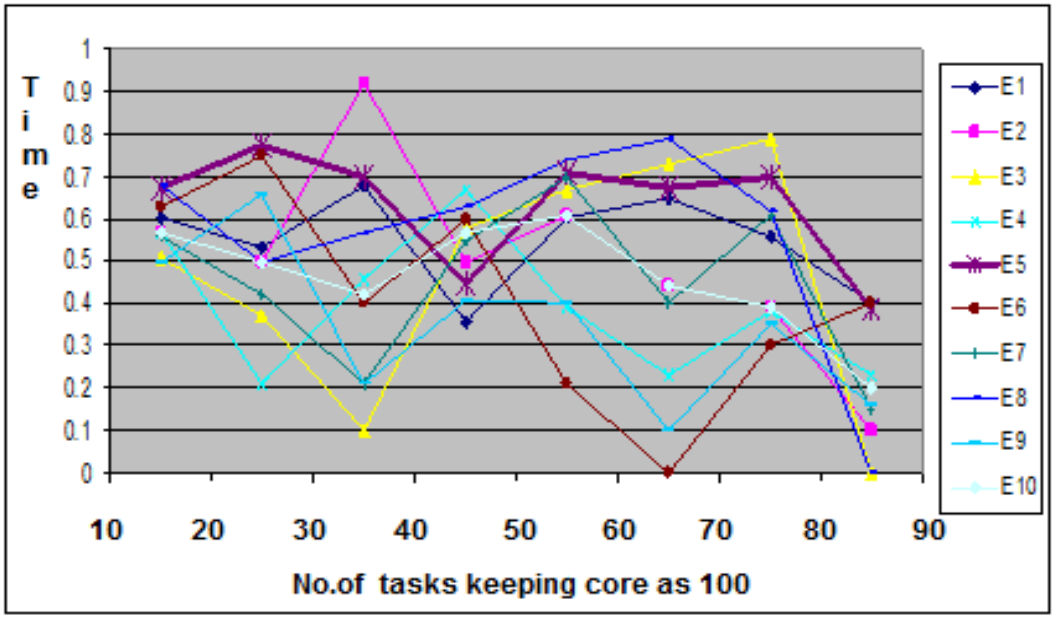


Figure 4: Execution time for various scheduling algorithms keeping the number of cores=100

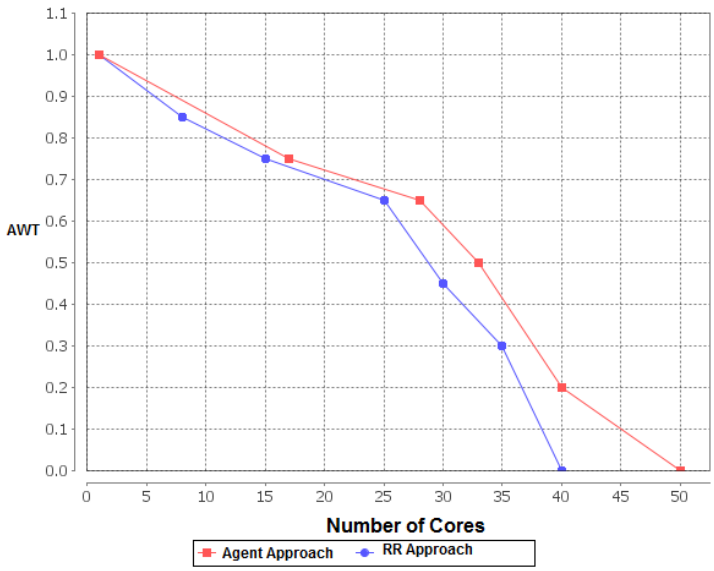


Figure 5 : AWT Vs. Number of cores for Agent and RR Approach

In Figure 5, the average waiting time of several processes is obtained for the selected number of cores. It is discovered that the average waiting time decreases slowly with the increase of the number of cores. The same numerical results had been explained in Table.1.

Table 1: Number of cores and average waiting time for Agent and Round Robin approach

Number of Cores	AWT (Agent)of 1000 processes(in ps)	AWT(RR) of 1000 processes(in ps)
1	1	1.2
10	0.7	0.8
20	0.6	0.7
30	0.4	0.5
40	0.2	0.3
50	0.1	0.2

The experimental results are shown for agent based approach and round robin approach. The average waiting time of the processes getting decreased and almost it reaches to zero for large number of cores in multicore architecture.

6. Conclusion

In this paper we propose a new agent based two buffer hierarchical scheduling algorithm that enhances the performance of the processor by 20% compared with the traditional algorithms. There are two levels in the overall design wherein in the first level all the tasks are assigned with equal priority and a buffering method is implemented. Whereas in the second level, we consider the real time task and affinity based scheduling is incorporated. For the evaluation results, we modified linux 2.6.21 kernel along with the FLAME tool. Ultimately, the overall results proves that this agent based algorithm outperforms in cpu performance and reduces average waiting time of the process by 4.5% compared with the conventional scheduling algorithms.

References

1. Sergey Zhuravley, Blagoduroy, Alexandra Fedorova,2010. “Managing contention for shared resources on multicore processors”, Communications of the ACM Volume 53, Pages: 49-57 Issue 2 February.
2. Sergey Zhuravley, Blagoduroy, Alexandra Fedorova, 2010. “Addressing shared resource contention in multicore processors via scheduling”, Architectural support for Programming Languages and Operating Systems, Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems Pages: 129-142.
3. John M.Calandrino, James H. Anderson, 2009. “On the Design and Implementation of a Cache-Aware Multicore Real-Time Scheduler”, 21st Euromicro Conference on Real-Time Systems July 01-July 03.
4. Tong LiDan, BaumbergerDAvid A, KoufatyScott Hahn, 2007.“Efficient operating system scheduling for performance asymmetric multi-core architectures”, Conference on High Performance Networking and Computing Proceedings of the ACM/IEEE conference on Supercomputing.
5. Karthik Lakshmanan, Ragunathan (Raj) Rajkumar, and John P. Lehoczky,2009. “Partitioned Fixed-Priority Preemptive Scheduling for Multi-Core Processors”, Proceedings of the 21st Euromicro Conference on Real-Time Systems Pages: 239-248.
6. D. Grosu, A. T. Chronopoulos, M. Y. Leung, 2008. “Cooperative Load Balancing in Distributed Systems”, Concurrency and Computation,Practice and Experience. Vol. 20, No. 16, pp. 1953-1976, November
7. Ali M. Alakeel, 2010. “Load Balancing in Distributed Computer Systems”, International Journal of Computer Science and Information Security Vol. 8 No. 4 July.
8. Dahoud Ali, Mohamed A. Belal and Moh’d Belal Zoubi, 2010, “Load Balancing of Distributed Systems Based on Multiple Ant Colonies Optimization” , American Journal of Applied Sciences 7 (3): 433-438.
9. James H. Anderson, John M. Calandrino, and UmaMaheswari C. Devi, 2006. “Real-Time Scheduling on Multicore Platforms”, Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, Pages: 179 – 190.
10. Carole Bernon, 2006, “Applications of Self-Organising Multi-Agent Systems: An Initial Framework for Comparison”, IRIT, INRIA.
11. Di Marzo Serugendo G., Gleizes M-P. and Karageorgos, 2006. “Self-Organisation and Emergence in MAS: An Overview”, A INFORMATICA.
12. G.Muneeswari, A.Sobitha Ahila, Dr.K.L.Shunmuganathan, 2011. “A Novel Approach to Multiagent Based Scheduling for Multicore Architecture”, GSTF journal on computing, Singapore vol1.No.2.
13. G.Muneeswari, Dr.K.L.Shunmuganathan, 2011. “Improving CPU Performance and Equalizing Power Consumption for Multicore Processors in Agent Based Process Scheduling”, International conference on power electronics and instrumentation engineering, Springer-LNCS.
14. Alexandra Fedorova, Margo Seltzer and Michael D. Smith, 2006. “Cache-Fair Thread Scheduling for Multicore Processors”, TR-17-06.
15. Chandra Chekuri, 2004. “Multiprocessor Scheduling to Minimize Flow Time with Resource Augmentation”, STOC’04, June 13–15.
16. James H. Anderson and John M. Calandrino, 2006. “Parallel Task Scheduling on Multicore Platforms”, ACM SIGBED.
17. Stephen Ziemba, Gautam Upadhyaya, and Vijay S. Pai.,2004. “Analyzing the Effectiveness of Multicore Scheduling Using Performance Counters”.

18. James H. Anderson, John M. Calandrino, and UmaMaheswari C. Devi, 2006. “Real-Time Scheduling on Multicore Platforms”, Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, Pages: 179 – 190.
19. Muneeswari G, Shunmuganathan K. L. “A Novel Hard-Soft Processor Affinity Scheduling for Multicore Architecture using Multiagents” European Journal of Scientific Research, ISSN 1450-216X Vol.55 No.3 (2011), pp.419-429, European Journal of Scientific Research