International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Selective Scheduling Based on Number of Processor Cores for Parallel Processing

Ravinder Jeet[1], Upasna Garg[2]

[1]Guru Kashi University, Punjab 151302, India

[2]Assistant Professor), Guru Kashi University, Punjab 151302, India

Abstract: ***Scheduling is essential for the proper functioning of multi-core processors for parallel processing. Scheduling of tasks onto multi-core processors is an interesting problem that is well defined and documented in the literature. Multi-core processor systems are increasingly commonplace, and have found their way into desktop machines, laptops, and even mobile devices. The rise of the multi-core processor, in which multiple CPU cores are packed onto a single chip, is the source of this proliferation. These chips have become popular as computer architects have had a difficult time making a single CPU much faster without using too much power.With the advent of multi core processors, there is a need to schedule multiple tasks on multiple cores. In this paper we are proposing a new scheduler algorithm which schedules the multiple tasks on multiple cores of a single chip processor. One task is assign to a single core and the one is on another core processor within a single chip processor for parallel processing. The main goal of this work is to allow jobs to scale well with the number of cores. The proposed algorithm will allow each processor core to execute at least one task on single instance of time. This means if the number of processor cores is four then four tasks will be execute simultaneously on all four cores.***

Keywords: Selective Scheduling, Multi-Core System, Parallel Processing, Number Of Processors

1. Introduction

Scheduling of tasks is essential for every system. Scheduler is one of the most important parts of an Operating system. Without scheduler, tasks may not execute in that order which a user or operating system itself want. Scheduling of tasks on single core processor is much easy by choosing existing scheduler programs like Round-Robin, Priority based, First Come First Serve bases, Shortest job First etc. Multi-core processors have two or more processing elements or cores on a single chip. These cores could be of similar architecture (Synchronous Multi-core Processors, SMPs) or of different architecture (Asynchronous Multi-core Processors, AMPs). All the cores necessarily use shared memory architecture. Multi-core processors have existed previously in the form of MPSoC (Multi-Processor System on Chip) but they were limited to a segment of applications such as networking [13]. The easy availability of multi-core has forced software programmers to change the way they think and write their applications [13]. Unfortunately, the applications written so far are sequential in nature. Multi-core processors do not automatically provide performance improvements to applications the way faster processors did. Instead applications must be redesigned to increase their parallelism. Similarly, CPU schedulers must be redesigned to maximize the performance of this new application parallelism. CPU scheduling policy (and in a large part mechanism) is unimportant to a serial application running on its own machine [2]. An important part of parallel processing in multi-core reconfigurable systems is to allocate tasks to processors to achieve the best performance. The objectives of task scheduling algorithms are to maximize system throughput by assigning a task to a proper processor, maximize resource utilization, and minimize execution time.

In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multi-tasking is to have some process running at all times, to maximize CPU utilization. Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design. CPU scheduling determines which processes run when there are multiple run-able processes

2. Scheduling Parameters

- **CPU Utilization**: It is the average fraction of time, during which the processor is busy.
- **Throughput**: It refers to the amount of work completed in a unit of time. The number of processes the system can execute in a period of time. The higher the number, the more work is done by the system.
- **Waiting Time:** The average period of time a process spends waiting. Waiting time may be expressed as turnaround time less the actual execution time.
- **Turnaround time**: The interval from the time of submission of a process to the time of completion is the turnaround time.
- **Response time:** Response time is the time from submission of a request until the first response is produced.
- **Priority:** give preferential treatment to processes with higher priorities.
- **Fairness:** Avoid the process from starvation. All the processes must be given equal opportunity to execute.

3. Multi-core Processor Architecture

In Multi-core processor architecture two or more chips are embedded in to single processor chip that are called cores. A single core is a single complete microprocessor in itself which can perform all operations as a full microprocessor. Memory is shared by all available cores in single chip. Most of Multi-core processor have single shared cache storage which leads to mutual accesses in processing.

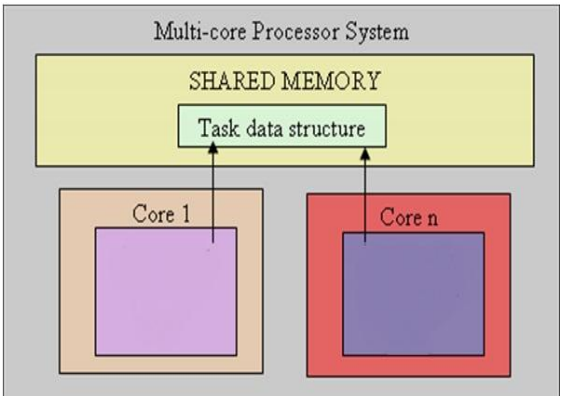

Figure 1: Block diagram of Multi-core Processor

The efficiency of a parallel computing system is commonly measured by completion time, speedup, or throughput, which in turn reflect the quality of the scheduler. Many algorithms have already been developed which provide effective solutions. Most of these methods, however, can solve only limited classes of the scheduling problem [1]. The scheduling problem is known to be NP-complete for the general case and even for many restricted cases [15].

4. Related Work

Many researchers have defined various scheduling algorithms for multi-core processor systems. Vinay G. Vaidya's "*Dynamic Scheduler for Multi-core Systems"* scheduler program runs on each processor core to schedule tasks on each core individually. But the limitation is that a main program is needed to control all cores [13]. Some other multi-core schedulers are uses one scheduling algorithm like "*SUBCONTRARY MEAN DYNAMIC ROUND ROBIN (SMDRR) SCHEDULING ALGORITHM"* on each core [21]. SMDRR scheduler runs on each core but there is no main scheduler to control all tasks which decides which tasks execute on which core.

5. Literature Review

1) **Joseph t. Meehean(2011)** - et al - made two advances in the area of applying scientific analysis to CPU schedulers. The first, CPU futures, is a combination of predictive scheduling models embedded into the CPU scheduler and user-space controller that steers applications using feedback from these models. Developed these predictive models for two different linux schedulers (cfs and o(1)), based on two different scheduling paradigms (timesharing and proportional-share). Using three different case studies, has demonstrate that applications can use our predictive models to reduce interference from low-importance applications by over 70%, reduce web server starvation by an order of magnitude, and enforce scheduling policies that contradict the CPU scheduler's. The second contribution is a framework and set of experiments for extracting multiprocessor scheduling policy from commodity operating systems. And used this tool to extract and analyze the policies of three linux schedulers: o(1), cfs, and bfs. These schedulers often implement strikingly different policies.[2]
2) **Andrew j. Page (2010)** - et al - present a multi-heuristic evolutionary task allocation algorithm to dynamically map tasks to processors in a heterogeneous distributed system. It utilizes a genetic algorithm, combined with eight common heuristics, in an effort to minimize the total execution time. It operates on batches of unmapped tasks and can preemptively remap tasks to processors. The algorithm has been implemented on a java distributed system and evaluated with a set of six problems from the areas of bioinformatics, biomedical engineering, computer science and cryptography. Experiments using up to 150 heterogeneous processors show that the algorithm achieves better efficiency than other state-of-the-art heuristic algorithms.[3]
3) **Raj Mohan Singh (2013)** - et al - discuss some basic job scheduling strategies and also propose a new scheduling strategy which is based on the criticality i.e. How much important the job is for the user and priority of jobs with an effort towards improving the response time of the jobs. The idea is to motivate the users to submit more jobs and to minimize the chances of the users leaving the session. Interactive jobs usually require much less resources and are much more critical to the users than the batch jobs that execute over nights and weekends. The jobs are executed by first applying criticality to round robin scheduling and then applying priority to round robin scheduling. These scheduling strategies are then compared and their performance is evaluated on the basis of the three parameters viz. Average waiting time, average turnaround time, and average response time. It is found that by applying criticality and priority on round robin scheduling there is significant improvement in the values of the three parameters especially the response time.[4]
4) **Jonathan weinberg (2007)** - et al - primarily work about with job scheduling, a discipline whose purpose is to decide when and where each job should be executed from the perspective of the system. This is related to, but distinct from, the study of application/task scheduling where the question is how a single parallel application should schedule each of its threads. Each job is characterized along two dimensions: its length as measured by execution time and its width or size as measured by the number of threads; assuming each of a job's threads executes on a separate processor. For well over a decade, the field of job scheduling has been the subject of great scrutiny, producing a sizeable body of work and increasing returns on hpc investments by millions of dollars. Despite this progress, one may argue that the problem of scheduling on parallel systems may not be closer to being solved today than it was a decade ago. Scheduling is an inherently reactive discipline, mirroring trends in hpc architectures, parallel programming language models, user demographics, and administrator priorities. No scheduling strategy is optimal for all of today's scenarios.[6]
5) **Albert y. Zomaya (1999)** - et al - a genetic algorithm (ga) is a search algorithm which is based on the principles of evolution and natural genetics. It combines the exploitation of past results with the exploration of new areas of the search space. By using survival of the fittest techniques combined with a structured yet randomized information exchange, a Ga can mimic some

of the innovative flair of human search. A generation is a collection of artificial creatures (strings). In every new generation a set of strings is created using information from the previous ones. Occasionally a new part is tried for good measure. Ga's are randomized, but they are not simple random walks. They efficiently exploit historical information to speculate on new search points with expected improvement. The basic type of gas, known as the simple Ga (SGA), uses a population of binary strings, single point crossover, and proportional selection. Many other modifications to the sga have been proposed, some of these are adopted in this work.[1]

6) **Sourav Kumar Bhoi (2014) –** et al - Round Robin (RR) Algorithm is considered as optimal in time shared environment because the static time is equally shared among the processes. If the time quantum taken is static then it undergoes degradation of the CPU performance and leads to so many context switches. In this paper, we have proposed a new effective dynamic RR algorithm SMDRR (Sub contrary Mean Dynamic Round Robin) based on dynamic time quantum where we use the sub contrary mean or harmonic mean to find the time quantum. The idea of this approach is to make the time quantum repeatedly adjusted according to the burst time of the currently running processes. Our experimental analysis shows that SMDRR performs better than RR algorithm in terms of reducing the number of context switches, average turnaround time and average waiting time.

6. Problem Formulation

Various schedulers are available for scheduling jobs on multi-core processors. Many scheduling algorithms have been proposed in the past. With the advent of multi core processors, there is a need to schedule multiple tasks on multiple cores. The scheduling algorithm needs to utilize all the available cores efficiently. The multi-core processors may be SMPs (Symmetric multi-processor) or AMPs (Asymmetric multiple-processors) with shared memory architecture [13]. This work propose a dynamic scheduling algorithm called 'Selective Scheduling' in which the scheduler resides on all cores of a multi-core processor and accesses a shared Task Data Structure (TDS) to pick up ready-to-execute tasks. This method is unique in the sense that the processor has the capability of picking up tasks whenever it is idle. In MS windows by default only one processor core is active if the hardware has multiple cores in its architecture.

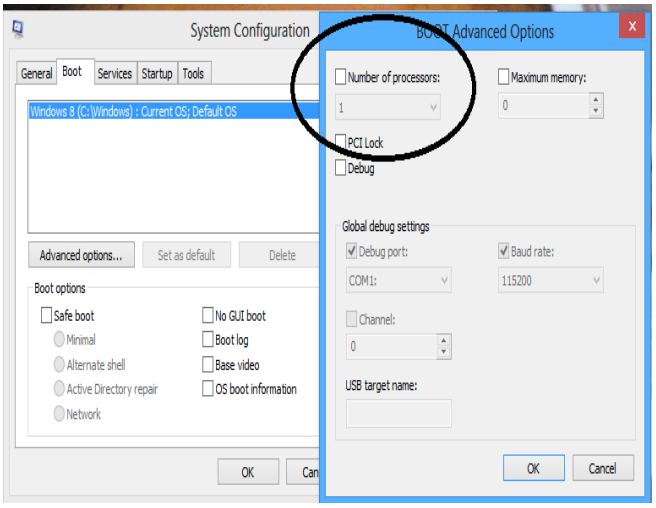

Figure 2: windows 8 default single core setting

When open this setting by default active number of processor core is one. This can be changed by checking the option of number of processor. Windows 8 uses Round-Robin scheduling [22], which is not enough for multi-core processor system architecture. In this work we proposes a new strategy which uses existing traditional algorithms like Priority based, First Come First Serve bases, Shortest job First etc.

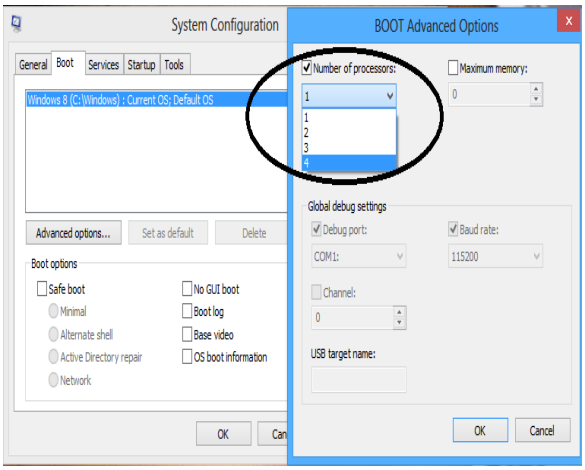

Figure 3: Total number of cores in processor

This setting can be changed by type 'msconfig' in run command window and setting can be seen to set 1 core by default. But the limitation is by activating all the cores there is no difference in the performance of the system, because the scheduling is Round-Robin in windows 8 [22]. Thus to solve this problem 'Selective Scheduling' is proposed in this work. This scheduling algorithm take care of all active cores available in the hardware of the system and assigns jobs to all cores for better performance.

Objectives

The objective of this work is to introduce a new approach algorithm for multi-core processors. The objective of scheduling is to maximize the utility of system. The objectives of this work are given below.

1. **Maximize Throughput-** overall throughput of system is enhanced than previous scheduling algorithms. At a single time 4 cores are active and all 4 cores executes jobs on the same time therefore throughput is maximized.

2. **Reduce Response Time**- response time reduced as much as possible. Every time a new is created there is always a processor core available with lowest priority job for first response to new job.
3. Job assigning function to processor cores.
4. Priority assigning function to jobs according to their nature.
5. Work stealing function.
6. **Cache Affinity**- this notion is simple; a process when run on a particular CPU builds up a fair bit of state in the caches of the CPU. The next time the process runs it is often advantageous to run it on the same CPU as it will run faster if some of its state is already present in the caches on that CPU. If instead one runs a process on a different CPU each time, the performance of the process will be worse, as it will have to reload the state each time it runs. This work prefers to keep a process on the same CPU if at all possible.

There are many other factors which are kept in mind while developing algorithm such as:-

1. **Performance** - this quality is often measured using variations on response time [6]. Response time also known as flow time or turnaround time is the amount of time elapsed between a job's submission and completion. The intuition behind this metric is that users are happier with speedier response times.
2. **Predictability** - predictability is the gap between a job's responses or flow time and the user's expectation as created through previous experience. Predictability can indirectly increase productivity by enabling users to anticipate job completion times and plan resource usage accordingly. Some have proposed that predictability, under other realistic assumptions, may be even more central to the user experience than performance.
3. **Fairness** - most scheduling algorithms make the minimum fairness guarantee that no job will be starved, that is, each job will eventually execute. Stronger fairness guarantees are necessary in the scheduling scheme.

7. Research Methodology

The scheduling technique explores in this work is based on number of cores and can be implemented in the operating system to schedule jobs.

Selective Scheduling

Selective scheduling is based number processor cores. Various scheduling algorithms are used in this work such as FCFS, SJF, Round-Robin, Priority and Pre-emptive scheduling. Priority is assigned to each job according to nature of job. In this work scheduling is implemented on four cores processor. In spite multiple core processors are available in modern systems [13]. But the limitation is that at a particular time period one job is executed at one core, at the same time all other cores are in idle state. Therefore CPU utilization is decreased continuously. So there is a solution of this problem in this work. Scheduling named selective scheduling based on number processor core is introduced in this work.

Assumptions are, at start point CPU is idle and performance is measured from the time when the jobs are available on the system for execution. Then priority is assigned to each job by the nature of the jobs like Run time, Burst time, job is either system or Application process. After assigning priorities each job is assigned to a particular processor core out of four processor core. There are four cores in processor therefore 4 jobs are executed at single piece of time. Each core has its cache memory. Main memory is shared by all four cores and memory is also divided in to four sections based on four cores so that memory access is made fast by each core and collision does not occur between all cores at the time of execution.

Response time is also reduced by this scheduling. When a new job is occurred it is assigned to one processor core for first response and running job on that core is primped at that time. Context switching is used in this condition. At first response priority of new job is compared with priority of existing job. If priority of new job is higher than existing job then new job is assigned to that core and existing job is put in waiting state.

Algorithm

Scheduler program named **selsch.**

1. System is idle at the start point.
2. Selsch searches for jobs in the memory.
3. If jobs are found
   {

Divide the jobs in categories according to
Length= measure as total run time.
Width= measured as size of job.
Burst time
Job is either **system** or **application**.

   }

4. **Priority Function**

Assigning priorities to jobs according to categories

{

Lower the size of job= higher priority
Lower the running time= higher priority
System jobs have higher priority than application job.

}

5. **Scheduling Function**

First processor core= system jobs
Remaining three processor cores= all application jobs by priority

6. **Response Function**

When new job created during execution

{

- Check for job is either **system** process or **application.**
- Core with lower priority job primps the current job.
- Context switching occurs to save state of current job.

Call priority function

- Compare priorities of new job and current job.
- Priority with higher value is assigned to processor core.

}

7. **call response function for every new created job**

8. long term function

- Scans for jobs which are in waiting state for 5 minutes long.
- Assign new priorities to those jobs.
- Re-schedule them by calling step 5.

9. Repeat steps 2 to 8 for continuing processing.

10. Exit.

8. Results

Results are shown in the table by comparing two other scheduling algorithms.

Parameters	Round Robin algorithm	SMDRR (sub-contrary mean dynamic round robin) algorithm	Priority Pre-emptive algorithm	Selective algorithm
Avg. Response time (in seconds)	0.43	0.31	0.37	0.23
Avg. Scheduling time (in seconds)	0.24	0.20	0.22	0.16
Communication b\w cores %	0.478	0.436	0.543	0.522
Efficiency %	89.3	91.2	90.3	93.1

Figure 4: Results are shown by different parameters Results shown in Figure-4 depict that response time, scheduling time and efficiency are higher than previous algorithms but communication between cores are less effective than other algorithms.

9. Conclusion

Results are well obtained as desired but sometimes a few mistakes were present in the work. Here selective algorithm is compared with two other algorithms. Four parameters are compared and three of them which are most important foe overall efficiency of the system.

But the limitation is that there is less communication between CPU cores which leads to a little bit efficiency is reduced. This work is built under the 'C++' programming language therefore this algorithm can be easily implemented in windows environment.

10. Future Work

For future work one can implement this technique in operating system for pc's, laptops and in operating system compatible smart phones because in every device which has an operating system consist of multi-core hardware processing chips.

References

[1] Albert Y. Zomaya, Chris Ward, and Ben Macey "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues", IEEE Transactions On Parallel And Distributed Systems, Vol. 10, No. 8, August 1999.

[2] Joseph T.Meehean Doctoral Dissertation at University of Wisconsin–Madison, "Towards Transparent CPU Scheduling" 2011.

[3] Andrew J. Page, Thomas M. Keane, Thomas J. Naughton, "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system" Journal of Parallel & Distributed Computing 70 (2010) 758_766, 2010.

[4] Raj Mohan Singh and Harsh K Verma," An Analysis of Scheduling Strategies Based on Criticality of Jobs" International Journal of Computing, Communications and Networking, ISSN 2319-2720, Volume 2, No.1, January – March 2013.

[5] Arpaci-dusseau," Multiprocessor Scheduling (Advanced)", 2014.

[6] Jonathan Weinberg," Job Scheduling on Parallel Systems", in Proc. International Conference on Job Scheduling Strategies for Parallel Processing *(IPPS)* CA 92093-0505, 2007.

[7] Y. Chow and W.H. Kohler,"Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System", IEEE Trans. Computers, vol. 28, pp. 354-361, 1979.

[8] Sangsuree Vasupongayya, Su-Hui Chiang "On Job Fairness in Non-Preemptive Job Scheduling", in Proc.International Conference on Parallel and Distributed Computing, Phoenix, AZ, USA, Nov 14-16, 2005.

[9] Jonathan Weinberg, "Job Scheduling on Parallel Systems**"**, in Proc. International Conference on Job Scheduling Strategies for Parallel Processing (IPPS), 2002.

[10] Edi Shmueli, Dror G. Feitelson. On "Simulation and Design of Parallel-Systems Schedulers: Are We Doing the Right Thing?" IEEE transaction on Parallel and Distributed System, Vol. 20, No. 7, pp 983-996, July 2009.

[11] Mahima Shrivastava, "Analysis and Review of CPU Scheduling Algorithms", International Journal of Scientific Engineering and Research (IJSER) ISSN (Online): 2347-3878 Volume 2 Issue 3, March 2014.

[12] Debashee Tarai, "Enhancing Cpu Performance Using Subcontrary Mean Dynamic Round Robin (Smdrr) Scheduling Algorithm", International Journal of Scientific Engineering and Research (IJSER) Volume 2 Issue 3, March 2011.

[13] Vinay G. Vaidya," Dynamic Scheduler for Multi-core Systems",2010 2nd International Conference on Software Technology and Engineering(ICSTE) 2010.

[14] Imran Qureshi, "CPU Scheduling Algorithms: A Survey", Int. J. Advanced Networking and Applications Volume: 05, Issue: 04, Pages: 1968-1973 (2014) ISSN : 0975-0290.

[15] Rakesh Mohanty, H. S. Behera, Khusbu Patwari, Monisha Dash, Design and Performance Evaluation of a New Proposed Shortest Remaining Burst Round Robin (SRBRR) Scheduling Algorithm, In Proceedings of International Symposium on Computer Engineering & Technology (ISCET), Vol 17, 2010.

[16] Yu-Kwong Kwok," Amalysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors", 1087-4089/96 $5.00 0 1996 IEEE.
[17] Thu D. Nguyen, "Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling", National Science Foundation (Grants CCR-9123308 and CCR-9200832) 1999.
[18] Thu D. Nguyen, "Parallel Application Characterization for Multiprocessor Scheduling Policy Design", National Science Foundation (Grants CCR-9123308 and CCR-9200832) 2001.
[19] Sujatha R. Upadhyaya." Parallel approaches to machine learning—A comprehensive survey", J. Parallel Distrib. Comput. 73 (2013) 284–292. 2012.
[20] Krste Asanovic. "A View of the Parallel Computing Landscape" communications of the acm vol. 52, no. 10, october 2009.
[21] Sourav Kumar Bhoi "Enhancing Cpu Performance Using Subcontrary Mean Dynamic Round Robin (Smdrr) Scheduling Algorithm" 2014.
[22] Kalpana Gupta "A Comparative Study Of Two Operating Systems:
[23] Windows 7 And Windows 8" *International Refereed Multidisciplinary Journal Of Contemporary Research* Eissn 2320-3145, Issn 2319-5789, 2013.