Indian Journal of Science and Technology, Vol 8(22), DOI: 10.17485/ijst/2015/v8i22/79107, September 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Performance Analysis of OS Scheduling for a Reconfigurable Computing Environment

B. Abirami[1*], K. Sridhar[2] and V. Vaidhyanathan[3]

[1]Department of CSE, M.A.M. School of Engineering, Trichy - 621105, Tamil Nadu, India; nbs.abirami@gmail.com
[2]Department of ECE, M.A.M.C.E.T., Trichy - 621 105, Tamil Nadu, India; mkm.sridhar@gmail.com
[3]School of Computing, SASTRA, Tanjore – 613 401, Tamil Nadu, India; vvn@it.sastra.edu

Abstract

The primary objective of this proposal is to minimize the battery drain and to maximize the compatibility of the device so that the complexity of the device is much more reduced to enhance the battery period considerably. In this paper we adopted a method for increasing the response time using a priority based algorithm along with Field Programmable Gate Array (FPGA) in Reconfigurable Computing System (RCS) to handle complex applications and to increase the efficiency and throughput of the system. A primary comparison was made with all existing algorithms namely First Come First Serve (FCFS), Round Robin, Shortest Job First (SJF) for a given set of processes by taking process delay as a core parameter. The result obtained from this analysis gave a clear picture that the objective can be achieved by decreasing the waiting time and by increasing the throughput. By achieving this task, multitasking between the applications or processes can be done frequently. By adopting these kind of scheduling algorithms with programmable gate array, there is a considerable improvement and performance of the device. Also, the life span of the device is increased which is considered as most important criteria for any users in computing environment.

Keywords: Operating System, Priority, Response Time, Scheduling, Throughput

1. Introduction

The combined flexibility of hardware and software is the key characteristic of a Reconfigurable Computing System.

Reconfigurable devices are expected to be used in the future embedded systems because of their flexibility and increased performance. Field Programmable Gate Array (FPGA) architectures have become the unavoidable part of the newly developed digital systems. This scenario proposes new challenges and optimization requirement to construct a stable system with better performance. Management of the resources and mapping the users for improved performance is also a point of concern. A survey of the reconfigurable computing and the systems and software related to their development is given in[1] and different FPGA architectures are given in[2-5].

As shown in Figure 1, the reconfigurable hardware will act along with the Central Processing Unit (CPU) and it will be an accelerator to speed up operations which normally take more time when executed by a CPU. In this paper, we discuss the possible challenges faced by the Operating System in optimally utilizing the resources for a reconfigurable computing system. Among the many issues, we throw our focus on the scheduling strategies which is usually a major bottleneck for an operating system. We also propose a feasible solution for an optimized scheduling strategy.

2. Operating System Challenges

The role of the conventional operating systems is to support applications during run time. The existing systems developed for reconfigurable architectures have not

*Author for correspondence

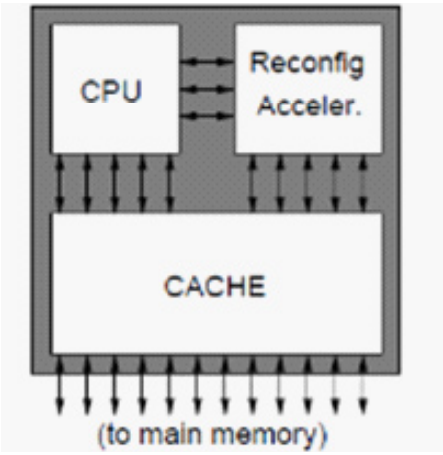

Figure 1. Reconfigurable accelerator as a co-processor.

yet concerned about the runtime support. This is mainly because, all these platforms have been developed based on the concept of single user load. Hence, in this paper, we attempt to propose a system which will recommend the scheduling strategies optimized to provide run time support for operations to be executed on the reconfigurable hardware.

To have a better insight in this arena, the scheduling policies and the challenges faced by the operating system are to be explored. The Operating System is basically responsible for loading and starting the execution of programs. In a system with reconfigurable architecture, this task is little more complex because a reconfigurable computing platform has been designed to go along with the conventional CPU. During runtime, the control is to be switched back and forth between the CPU dependant operations and FPGA dependant operations. This has been considered as a major bottleneck for the operating system to use the existing scheduling policies and load programs for either of these categories for execution. In the context of OS for reconfigurable architecture, time available for routing the tasks is the major constraint using the existing algorithms and also the optimization levels usually targeted and achieved. Yet another point to be noted is that whenever a program is loaded in the FPGA, the execution starts immediately whereas it is not so when a program is loaded in the RAM [1].

Yet another major constraint is the scheduling for a reconfigurable Operating System. Especially, while scheduling reconfigurable applications, it is very difficult to predict their completion time unless it is explicitly specified. Further, there are no fetch and decode cycles prior to execution, the scheduling strategy should suit to these applications. Generally, execution of FPGA applications is viewed as cooperative and there are some applications which do not prevent other applications to be loaded and executed but they delete them preemptively. Hence, in order to overcome these challenges, we propose a system in which the completion time of every FPGA application will be explicitly defined. So, the Scheduling can be done effectively to increase the performance of the system.

We propose a working model of a Reconfigurable Computing System which includes conventional CPU dependant applications as well as FPGA dependant applications. We are defining the completion time of the tasks executed by Reconfigurable hardware. For different types of applications, the various scheduling algorithms have been tested. The inference is that there is a significant increase in the speed of execution of the applications. On the other hand, the average waiting time of the jobs executed by the reconfigurable hardware is significantly reduced.

3. Scheduling Algorithms - An Insight

3.1 First Come First Served (FCFS)

A process that requests the CPU first will be allotted the CPU first. The newly incoming tasks will join at the tail of the queue. It is also called FIFO scheduling. They are normally used in Batch systems. With the given parameters of arrival time and burst time, the performance can be measured by the Average Waiting Time. Because of its disadvantages like, it is non-preemptive, the Average Waiting Time is not optimal and resources cannot be used in parallel, FCFS is considered to be an idealistic scheduling algorithm.

3.2 Shortest Job First (SJF)

Based on the burst time of the next job in queue, scheduling is done. Using the information on the burst time, the job with the shortest burst time will be chosen first to be executed. This scheduling method is suitable for both preemptive and non preemptive processes. This scheduling algorithm tends to give fair chance to all the jobs waiting in queue because of its advantage of achieving an optimal waiting time. It can prevent a shorter job starving for a longer time unnecessarily. But, practically, the burst times cannot be predicted and can be guessed. Estimating from the past behavior is common in computing systems based on the Principle of Locality that "if something happened recently, something similar is likely

to happen soon". Multiple desktop windows that are active at once are classical examples of SJF scheduling.

3.3 Round Robin Scheduling

Every job in the queue will be assigned a time slice for execution. The time interval will be uniform for all jobs. Irrespective of the completion of the jobs, they will exit the execution phase and join at the tail of the queue if they are incomplete. Thus, the Round Robin scheduling algorithm prevents the jobs waiting in the queue for indefinite amount of time. Obviously, this algorithm also favors shortest jobs to improve the throughput. But, as there are frequent context switches, the jobs are preempted on the expiry of their time slices, it is real technical overhead for the operating system.

3.4 Priority Scheduling

A number denoting the priority will be assigned for every process in the queue keeping in mind that smallest number will be given to the highest priority job. CPU will be allotted for the highest priority job and the mode can be either preemptive or non preemptive. But normally, the priority scheduling will be preemptive. Starvation is the major problem in this as lower priority jobs will have to wait infinitely. Starvation can be solved by increasing the priority based on the waiting time using the concept of aging.

3.5 Performance Analysis of Scheduling Algorithms

Thus, both the interactive scheduling and batch process scheduling algorithms try to resolve the conflicts of the executing processes. We assume a scenario where there is a blend of CPU dependant applications and FPGA dependant applications are there in a queue. We have implemented the different scheduling discussed so far on these two categories of jobs. From the results we have received, it can be inferred that if computationally intensive jobs or higher priority jobs are associated are FPGA hardware, the execution becomes faster. There is a considerable drop in the average waiting time of the processes invariably in all scheduling algorithms we have tried. The run time support given by the Operating System for these FPGA driven applications by passing the need of fetching and decoding cycles which collectively gains much amount of time, approximately ten times than the conventional CPU computations.

Hence, if this system is extended to larger real time applications, it will be a new paradigm with faster devices catering to solve very complex and computational applications. It is also a fact that there is going to be a great demand for runtime reconfigurable devices and applications developed using Verilog programming.

4. OS Process Scheduler Implementations

It is the thumb rule that while designing an operating system, the scheduling algorithm which performs the best is to be considered for the system. But it is an undeniable fact that there is no universal best scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms explained above.

For example, the operating system Windows NT/ XP/ Vista deploys the mechanism of a multilevel feedback queue, which is a combination of fixed priority preemptive scheduling, round-robin, and first in first out.

Mac OS involves a multilevel feedback queue which includes four priority bands for threads. The prioritization will be in the order of normal, system high priority, kernel mode only, and real-time. Here, the scheduling is done preemptively on the threads.

Linux Operating System makes use of a multilevel feedback queue. In this the priority levels range from 0 to 140. Out of this, the levels 0–99 are reserved for real-time tasks and 100–140 are considered nice task levels. It uses extensively the "Completely Fair Scheduling" algorithm designed based on the Rotating Staircase Deadline strategy. This is the first implementation of a fair queuing process scheduler which is greatly used in a general-purpose operating system.

Yet another Operating System Solaris solves the issues by using a multilevel feedback queue where the priorities range is between 0 and 169. In this, the priorities 0–59 are reserved for time-shared threads, 60–99 are used for system threads, 100–159 are dedicated for real-time threads, and 160–169 are used for low priority interrupts.

Based on the above details, it can very well be inferred that, there can be no universal scheduling mechanism for any OS. It depends on different parameters like criticality of the applications and performance issues of the Operating system. With reference to these data, our proposed system gives more weightage to the FPGA dependant applications which can handle computationally intensive complex applications at lesser time period.

5. Analysis of Algorithms

It is to be noted that the processes or tasks executed by a CPU is normally divided into four categories namely Operating System related processes, FPGA related processes, I/O related processes and applications related processes. Multiple threads use the CPU and the Reconfigurable hardware concurrently. Applications which are heterogeneous in nature are handled by the Operating System which is really a challenging task for the OS to prioritize them. The arrangement of the tasks of different types utilizing the CPU and the Reconfigurable Hardware (RH) will be as in the following Figure.

In general, Operating System processes relate to the memory management and resource sharing operations. Here with reference to the above discussions, the computations carried out by the Reconfigurable Hardware are called as the FPGA related processes. All the operations related to the Input and output devices connected to the system are termed as I/O related processes. Online gaming and other applications which are interactive in nature are called as applications related processes.

In order to find out a suitable algorithm for the RCS a sample data set was taken and all the above said four algorithms were tested. The data set consisted of 50 records. Each record had the arrival time, burst time and the priority of that process. Ten priorities were defined ranging from 0 to 9, 0 being the highest and 9 having the lowest preference. In each priority levels different number of processes was selected. The burst time was ranging from 3 to 99 units of time and the arrival time was also randomly selected. The priorities given to different processes are given in Table1.

For the sample data set considered, all the four algorithms were simulated using a Java program. The results were obtained in both non-preemptive and preemptive mode. The details of the total delay experienced by the processes in each category are given in Table 2.

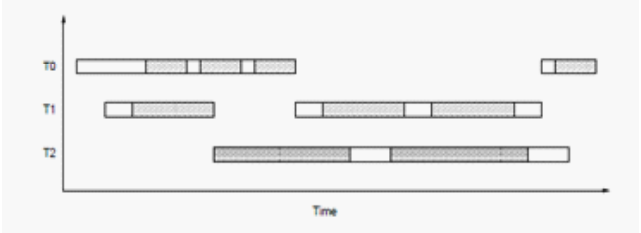

Figure 2. Allocation of jobs (threads) to the CPU and RH (Shaded for RH).

Table 1. Priorities given to processes

Process Type	Priority Level Assigned	No. of Processes in Each Level	Total Duration of Processes in Each Level
OS related processes	0,1	9, 7	436
FPGA related processes	2,3	8,4	251
I/O related processes	4,5	3,3	206
Applications	6,7,8,9	4,5,4,3	638

Table 2. Summary of the processes and their total delay time

Priority	No. of Processes	FCFS	Priority	Prio-Preem	RR	RR-Priority	SJF	SJF-PRE
0	9	90	62	21	178	82	105	89
1	7	177	68	36	120	169	80	39
2	8	465	192	71	321	592	193	73
3	4	211	89	11	113	210	91	32
4	3	34	48	44	276	316	108	163
5	3	1	1	0	1	1	1	1
6	4	241	139	83	226	180	137	107
7	5	306	285	310	344	298	210	206
8	4	163	495	962	345	185	198	191
9	3	47	47	179	74	47	47	11

It is to be noted that the arrival time of the processes were randomized to simulate the real time environment. Three processes with priority 5 arrive when the CPU is idle, serviced immediately and hence the delay is less.

We have assumed that processes with priority levels 2 and 3 are related to FPGA related processes and we are interested to find out the algorithm which gives the minimum waiting time. It was found out that the Priority based preemptive scheduling gives the lowest delay time for the given data set. This can further be extended to Multilevel Queuing system also where the priorities differ dynamically.

6. Conclusion and Future Work

The results of the recent researches clearly reveal that the reconfigurable systems perform extremely when used in multimedia and communication systems[6]. Also, many articles illustrate a general approach to scale up a run-time environment system or an Operating System with

the capacity to manage hardware resources[7-15]. More specifically, the usage of Embedded Systems in hand held devices is increasing at a faster pace. For these types of devices, if a CPU with high end configuration is used, it will be underutilized. So, for the existing scenario where the different scheduling algorithms are used, if the Operating System assigns higher priority to the Embedded System related applications, lot of time can be saved thereby improving the performance of the system or device.

In most of the practical applications, scheduling strategies for reconfigurable hardware is still an interesting area of research. Hence, the proposed system we have suggested in this paper can be tested on real time applications like the hand held devices using embedded systems to execute complex tasks in lesser time span. Further, in multilevel queuing environment, if the FPGA dependant applications are assigned higher priorities dynamically, significant increase in performance can be achieved. The future tasks planned are to simulate environments where FPGA dependant applications exhibit higher throughput and better performance for computationally intensive complex applications.

7. References

1. Compton K, Hauck S. Reconfigurable computing: A survey of systems and software. ACM Computing Surveys. 2002; 34(2):171–210.
2. Trimberger S. A time multiplexed FPGA. Proceedings of 5th IEEE Symposium on FPGA-based for Custom Computing Machines, FCCM97. IEEE Computer Society; Nappa Valley, USA. 1997. p. 22–8.
3. Xilinx. XC6200 Field Programmable Gate arrays. Xilinx, Inc.; 1997 Apr.
4. Gunther B. SPACE 2 as a reconfigurable stream processor. Proceedings of PART'97, the 4th Australian Conference on Parallel and Real-time Systems; Singapore. 1997 Sep. p. 286–97.
5. Wigley G, Kearney D. The first real operating system for reconfigurable computers. 6th Australian Computing Science week (ACSAC'01); Gold Coast, Australia. 2001. p. 130–7.
6. Hartenstein RW, Kress R, Nageldinger U. An operating system for custom computing machines based on xputer paradigm. Proceeding of the 7th International Workshop on Field Programmable Logic. 1997 Sep. p. 304–13.
7. Vuletic M, Righetti L, Pozzi L, Ienne P. Operating system support for interface virtualization of reconfigurable coprocessors. Proceedings of the Design, Automation and Test in Europe Conference (DATE04); Paris, France. 2004 Feb. p. 748–749.
8. Sabeghi M, Bertels K. Toward a runtime system for reconfigurable computers: A virtualization approach. Proceedings of the Design, Automation and Test in Europe (DATE09) Conference; Nice, France. 2009. p. 1576–9.
9. Taher M, El-Ghazawi T. Virtual configuration management: a technique for partial runtime reconfiguration.The IEEE Transactions on Computers (TC). 2009 Jun; 58(10):1398–410.
10. Wigley GB, Kearney DA. Research issues in operating systems for reconfigurable computing.Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA02); Nevada, USA. 2002 Jun. p. 10–16.
11. Wigley G, Kearney D, Jasiunas M. ReConfigME: A detailed implementation of an operating system for reconfigurable computing. Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS); Rhodes Island, Greece. 2006 Apr.
12. Lubbers E, Platzner M. Recon OS: An operating system for dynamically reconfigurable hardware. Book Chapter in Dynamicaly Reconfigurable System. Springer; 2010 Feb. 269–90.
13. Lubbers E,Platzner M. Cooperative multithreading in dynamically reconfigurable systems.Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL09); Prague Czech Republic. 2009. p. 551–4.
14. Nollet V, Marescaux T, Verkest D, Mignolet JY, Vernalde S. Operating system controlled network on-chip. Proceedings of the 41st Design Automation Conference (DAC); San Diego, California, USA. 2004. 256–9.
15. So HK, Brodersen RW. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. ACM Transactions on Embedded Computing Systems (TECS). 2008 Feb; 7(2):290–302.