Shadow Register File Architecture: A Mechanism to Reduce Context Switch Latency

Jagan Jayaraj, Pravin Lawrence Rajendran and Thiruvel Thirumoolam
{jagan, pravinrajendran,t_thiruvel}@cs.annauniv.edu
School of Computer Science and Engineering
College of Engineering Guindy
Anna University, Chennai, India, 600025.

Abstract

In a multiprocessing environment context switch plays an important role in the overall performance of the system. The context switch latency affects the execution time of a process. In this paper we propose certain architectural modifications to reduce context switch latency. A dedicated hardware unit performs context switching operations with the help of Shadow Register File, simultaneously during the normal execution of the processor. The working of the architecture along with state transition diagram is explained in the paper.

1 Introduction

In a multiprocessing environment, the operating system shares the processor among the various processes. This is implemented by switching between the processes after a fixed time quantum. This is known as context switching. Context switch plays an important role in the overall performance of the system. The context switch latency affects the execution time of a process.

Context switch is implemented by swapping out the register values of the current process and swapping in the new register values. Moreover, the scheduler and the interrupt service routine (ISR) have to be executed. During this process, the processor remains idle. Hence a lot of research has gone into minimizing the effect of context switch. Context switches create hidden problems such as decreasing cache performance [4]. Hence the operating system should keep in mind the effects of context switching.

Generally effective context switching mechanisms have been developed for multithreaded processors [1], [2] and [3]. The threads are constructed by the operating system and many techniques like dribbling registers [2] and register window models are used, as in [3], [5]. In these architectures, when a thread stalls due to a long latency operation (like load/store), context switch takes place between threads. Register window models are effective in case of recursive function calls but they require a large register file.

In this paper we propose architectural modifications that minimize context switching latency. This design uses very minimal register files, only 2, and simple hardware to swap in and swap out register values during a context switch.

The paper is organised as follows. Section 2 describes the concept of the proposed Shadow Register File Architecture. Section 3 deals with the implementation issues. Section 4 gives concluding remarks.

2. Shadow Register File Architecture

The Shadow Register File Architecture consists of the two register files RF1, RF2 and the Shadow Update Unit (SUU). The SUU is a dedicated hardware unit to update the shadow register file when another process is being executed. The support required from the operating system is that the scheduler should be capable of selecting a group of ready processes for successive execution.

The Shadow Register File Architecture has the following characteristics:

- The context switch takes place by switching between the two register files.
- The Shadow Update Unit (SUU) makes ready the register file for the next process to be executed, simultaneously when the current process is being executed.

2.1 Concept of Shadow Register File Architecture

In our architecture we avoid processing the interrupt which is raised when a time quantum expires. Also the scheduler need not be run at each context switch. It is enough if it is run after a few context switches. The effect of context switches is thus minimized.

2.1.1 Constraints involved in context switching

1. The next process to be scheduled is determined by the scheduler. Hence the scheduler should be executed for every context switch.
2. Once the allocated time quantum has expired, the scheduler has to be woken up. This is done by an interrupt service routine. So the ISR has to be executed.
3. The register values of all the processors are to be loaded explicitly by the processor. This wastes processor time.

Therefore for one context switch to take place between processes three context switches are involved: one for ISR, second for scheduler and third for the actual process to be executed. We shall see how to tackle each of the constraints.

2.2 Shadow Update Unit (SUU)

The Shadow Update Unit (SUU) is a load/store unit with a dedicated cache or without a dedicated cache. The SUU saves the register contents of the previous process to a structure in cache and loads the register contents of the next process from the cache. This relieves the processor of explicitly loading the register values (constraint 3). Thus processor time is saved.

2.3 Working of Shadow Register File Architecture

1. The scheduler gives a list of processes (scheduled list SL). This scheduled list of processes is to be executed after which the scheduler is invoked again.
2. The number of processes to be executed is taken as the initial value for a down counter (DC).

3. Each time the interrupt (Itqe – Interrupt due to Time Quantum Expiry) is raised as a result of the expiry of the time quantum, the counter DC is decremented.
4. The Itqe interrupt is not serviced by an ISR. Instead Itqe is used to signal the shift between the register files thereby bringing about context switching. The current file and the shadow file are swapped.
5. The Itqe is also used by the SUU to save the previous current register file and load the register structure of the next process (from the SL) to the shadow file. We save only the modified register values and not the entire register structure.
6. Once the DC's value becomes zero in the shadow file, there are no more scheduled processes. The scheduler becomes the next process to be executed. Hence, the SUU starts loading the scheduler's register structure when the value in DC becomes one. Hence when DC becomes zero, the scheduler would be immediately ready for execution.

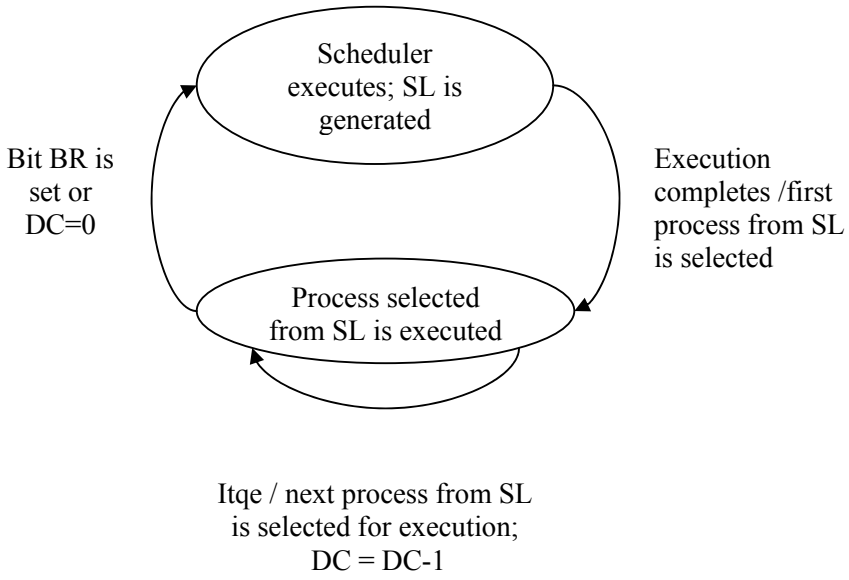

Fig 1: Transition Diagram

3. Implementation

3.1 SUU with a Dedicated Cache

A small dedicated cache (C1) is used along with the SUU to store only the register structures of the processes. The advantage of having this dedicated cache is that this cache can be accessed without interfering with the memory accesses of current process. So extra read and write ports are not required for the main cache. The cache replacement (if any) will be closely related to scheduling and hence the scheduler will do the cache replacement. The structure (or structures) to be replaced is decided by the scheduler and the instructions are issued to the SUU. The disadvantage of this approach is the extra cache hardware.

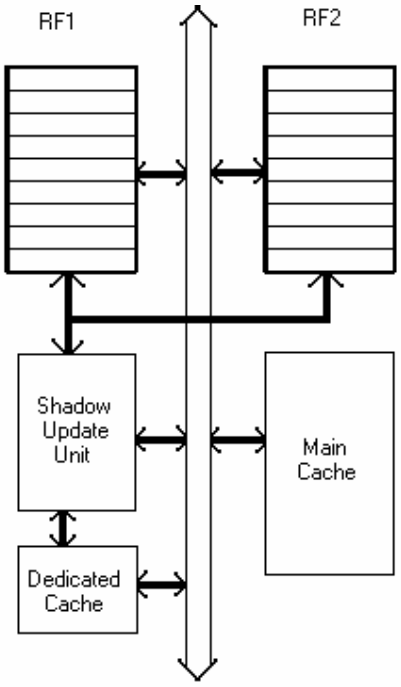

Fig 2 : Shadow Register File Architecture with Dedicated Cache and Shadow Update Unit

3.2 SUU without a Dedicated Cache

A portion of the L1 cache can be used to contain the register structures. In such a case both the main processor and the SUU will compete for the read / write ports of the cache. We have two options: increase the number of read / write ports, which will increase the cost of hardware, or do the operations of SUU when the cache is not being accessed by the processor. Since the number of values read / written by the SUU is comparatively less, we can follow the priority scheme and the SUU can use the cache only when it is not used by the main processor. The disadvantage of this scheme is the time delay to access the cache.

3.3 Interrupts and New Processes

1. Whenever an interrupt other than Itqe occurs, ISR can store the values of current register file to the cache directly, instead of storing the values to the Process Control Block (PCB) of the process.
2. Whenever a new process is spawned, a bit BR is set to indicate that new scheduling should be done.
3. When interrupt Itqe is raised and bit BR is set, the scheduler is loaded and the scheduler prepares a new list based on the priority of the new process and the scheduling algorithm used.

3.4 Improvements

Instead of invoking the scheduler every time a new process is spawned, we can call the scheduler only if needed. We have a separate register containing the minimum priority value (MPV) of all the processes in the list SL. If a new process is spawned with a priority value less than MPV then there is no need to invoke the scheduler when Itqe is raised and the BR bit need not be set. If the new process has priority value greater than MPV then BR is set. Thus the scheduler is invoked only if a higher priority process enters the system. The disadvantage with this method is that hardware complexity increases.

4. Conclusion

This paper proposes architectural modifications to reduce context switching latency. The scheme works by preparing the register file of the new process in the shadow of the current program's execution. The scheduling algorithm should be designed such that it is efficient for scheduling the processes in groups.

References

[1] Anant Agarwal, Beng-Hong Lim, David Kranz, and John Kubiatowicz, "APRIL: A Processor Architecture for Multiprocessing", Proceedings of 17th Annual International Symposium on Computer Architecture, pg104 -114, 1990.

[2] Vijayaraghavan Soundarajan and Anant Agarwal, "Dribbling registers: A mechanism for reducing context switch latency in large scale microprocessors", 1992.

[3] Yasuo Hidaka, Hanpei Koike, Hidehiko Tanaka, "Multiple Threads in Cyclic Register Windows", Proceedings of 20th annual international symposium on Computer architecture, San Diego, California, United States, 1993.

[4] Jeffrey C. Mogul and Anita Borg, "The Effect of Context Switches On Cache Performance", In Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 75--84, Santa Clara, CA, USA, April 1991.

[5] The SPARC Architecture Manual: Version 8.

[6] Andrew S. Tanenbaum and Albert S. Woodhull, "Operating Systems: Design and Implementation", 2nd Edition,2001.

[7] James L. Turley, "Advanced 80386 Programming ".