Manual CPU Throttling

Nick Saric
Computer Science Department
San Jose State University
San Jose, CA 95192
650-291-3717
nicksaric@gmail.com

ABSTRACT

In the Computer Science industry, having more processing power required us to change our computer systems to using multiple CPUs. This allowed computer programmers significantly more power and opened a wide variety of new features and options, but it also presented new problems that haven't been an issue before. This paper will go over some ways to help avoid certain problems related to parallel processing through the use of throttling the CPUs themselves. Through the use of power management and/or manipulating the scheduler, there were some benefits to using the two methods, but there were still some problems that posed an issue for Parallel Processing. In the end, the user should be careful when using either of these two techniques in order to ensure they receive the maximum benefit while minimizing the detriments they pose.

1. INTRODUCTION

In order to understand the methods discussed in the paper, there is some information that should be brought up beforehand in order to clear up any confusion. A process is usually referred to as an abstraction of a running program that is a core concept in operating systems [7]. Processes are made up of threads which can be single-threaded or multithreaded.

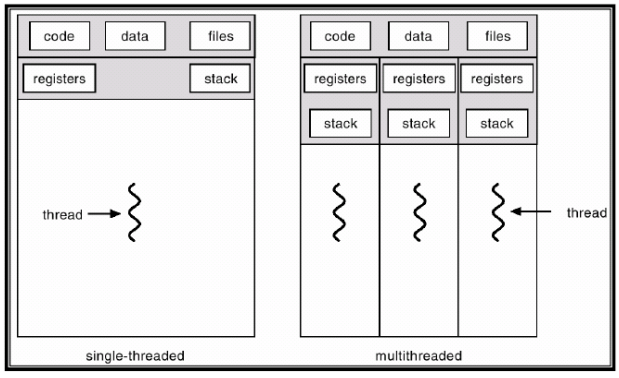

Figure 1. Note, each thread gets its own registers and stack.

Because of this setup it should be noted that processes themselves use significantly more resources than threads. With process switching, interaction with the operating system is required, whereas with threads, this isn't necessarily the case. In multiprocess environments, each process may execute the same code, however, each has separate memory and file resources. With threads, this is not the case, as each thread in the same task can share the same set of open files/child processes. It should also be noted that once a process is blocked then no other process can execute until the previous process is unblocked or releases control of the CPU but with threads, one thread can be blocked but a second within the same task can still proceed forward. Multiple processes using only one thread each is much more resource efficient than using a smaller amount of processes with more threads. Each process operates independently of each other whereas threads can read, write, and change another thread's data. Threads also help to minimize context switching time, help provide concurrency within a process, provide efficient communication, is more economical, and their utilization of multiprocessor architectures are to a greater scale and efficiency than with processes [10].

2. ALTERING THE VOLTAGE/FREQUENCY

One of the two ways we can Throttle the CPU's speed is through the use of power management. With this method the CPU's supply voltage and clock frequency would change.

2.1 HOW IT CAN BE DONE

This technique is already in use today but it requires that you use specific software or have on-chip integration to use it. The on-chip integration is more prevalent than the software control. Some examples of software and hardware uses of this program include the Dynamic Power Management Project, Intel's Cool'n'Quiet and SpeedStep series, LatticeSemiconductor's ispClocK5600 family, as well as the use of Power Gating in some of the newer Intel Core Processors and AMD's CoolCore.

The power management signals are

Speed_Sel_0, Speed_Sel_1 – These are the command signals and are decoded as follows:

00 - Full Power, CPU Voltage = 1.5V, Operating Frequency = 300 MHz

01 - Medium Power, CPU Voltage = 1.0V, Operating Frequency = 100 MHz

10 - Low Power, CPU Voltage = 1.0V, Operating Frequency = 33 MHz

11 - Controls the second power plane using the signal

Secondary_Plane_Control -

0 - Turn secondary plane off and 1 = secondary plane on

Speed_Sel_Strobe - latches the command from the power manager controller port

Figure 2. The different voltages and frequencies.

It should be noted that when using software, the extent of

the voltage or frequency change is determined as the program is running. For example, with LatticeSemiconductor's ispClocK5600 family, the on-chip clock generators can manually change the clock frequencies to 5 different speeds, ranging from 10MHz to 320MHz by using high-performance Phase-locked loops and clock multiple and divide facilities [11]. Figure 2 shows the settings for the 5 power management settings.

The formula below shows the correlation to energy saved and the voltage and clock frequency.

$$P = C * V^2 * F$$

where P is power consumption, C is total capacitance of all the circuits needing to be charged, V is the supply voltage to all devices, and F is the frequency of signal toggle [11]. This formula was provided courtesy of LatticeSemiconductor. This is the underlying principle of Dynamic Power Management. "Dynamic Power Management identifies low processing requirement periods and reduces operating voltage (voltage scaling) and frequency (frequency scaling), resulting in reduced average operating power consumption" [11].

2.2 METHOD SPECIFIC BENEFITS

There are some benefits that are specific to using this method. One benefit is that the lowering of voltage or frequency will reduce the energy consumed by the CPU. With reduced energy consumed, that also means less energy costs for the user. Over a longer period of time, I can eventually add up to a sizeable amount.

Another benefit is prolonged battery life with mobile devices. Through less power usage, the battery can still power the mobile device for longer periods. One example includes the HTC-One's Power Saver mode where it manually lowers the phone's CPU's clocking speed. This follows directly alongside the formula above and can prove useful when heavy-duty usage is not needed.

A third method specific benefit from this is where with less power usage comes with reduced heat buildup. This benefit will not help for larger computers such as desktops but can prove useful for mobile devices where physical contact with the device can be an issue if the machine runs too hot.

Lastly, while not as beneficial as the previous 3 benefits, this method will reduce the noise generated by the CPU itself.

2.3 METHOD SPECIFIC DETRIMENTS

When using this method, there are some potentially serious detriments that can affect your system. "Many CPUs may not operate reliably when their power supply voltage or input clock frequency is changing" according to Lattice Semiconductor. LatticeSemiconductor has also stated "In such cases, it is advisable to halt the CPU during the voltage and frequency transition" [11]. Different external hardware may be required in order to ensure that the CPU does not execute during these transitional periods. Complete halting of the CPU is a problem specific to using this method (not seen when altering the scheduler's algorithms for benefits).

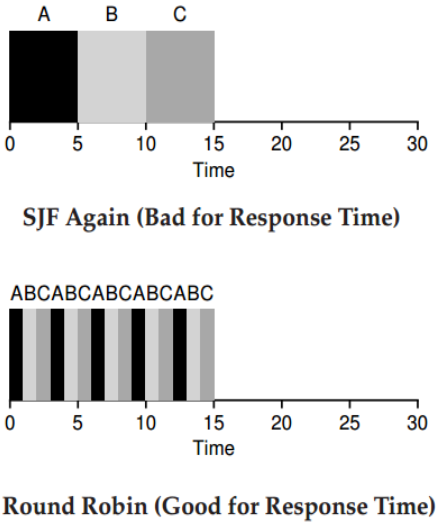

Figure 3. Time quantums reduced from 5 to 1.

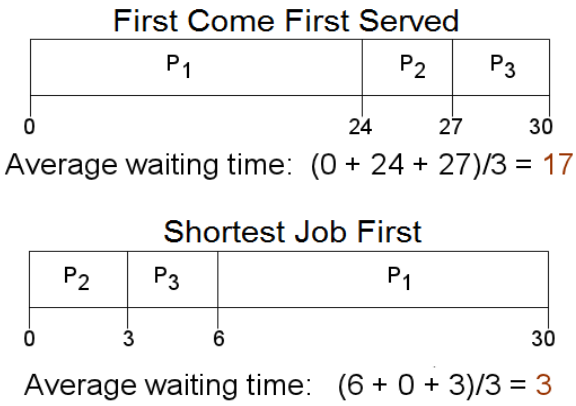

Figure 4. Long waiting time of job 1 is negated.

Another issue specifically related to this method is a latency when speeding up from the slower processing speed back to standard processing speeds. This poses the added problem that when using this method for code synchronization purposes, multiple CPUs will need to be slowed down in order to have maximum synchronization while they speed up. It is advised that extra precautions be made when using this method.

3. USING THE SCHEDULER

Altering the voltage or frequency of the CPU isn't the only way to throttle the CPU's speeds. Another method involved manipulating the operating system's scheduler in order to better suit the user's needs. Before deciding to manipulate the scheduler, the user should be informed of all of the

available assets for deciding on how to manipulate the scheduler.

3.1 TYPES OF SCHEDULING ALGORITHMS

There are many different algorithms for scheduling. Examples include First Come First Served, Shortest Job First, Round Robin, and more [3]. There are different characteristics when using different algorithms. One example would be how how Round Robin scheduling would yield better response time between requests compared to Shortest Job First, despite the two having the same potential runtime. Figure 3 demonstrates why this is the case. This is a result of the two formulas using two different time slices per job, also known as a scheduling quantum. Another example, the runtime difference between First Come First Serve and Shortest Job First are shown in Figure 4. This is because of the large waiting time imposed by job 1 which will affect the start times of jobs 2 and 3 as well. Pictures are courtesy of [4] and [7] respectively.

Even though different algorithms will have different characteristics, each algorithm still tries to achieve a certain et of goals. These goals include: Fairness (give each process fair share of CPU), Policy enforcement (seeing that stated policy is carried out), Balance (Keep all parts of the system busy), Throughput (maximize number of jobs per hour), Turnaround time (minimize time between submission and termination), CPU utilization (keep CPU busy all the time), Response time (respond to requests as quickly as possible), Proportionality (meet user's expectations), Meeting deadlines (avoid losing data in a timely manner), Predictability (avoid quality degradation in multimedia systems).

While still maintaining these goals, the different algorithms also still allow for thread scheduling to occur. Not only is thread scheduling still enabled but each algorithm also still allows for interruption handling to occur. When an interruption occurs you still have to: Stack program counter, load new program counter form interrupt vector, save registers, set up new stack, C interrupt service runs (at this point, usually read and buffer the input), scheduler marks the waiting task as ready, scheduler decides which process to run next, C procedure returns to the assembly code, and then the assembly language procedure starts up the new current process [8]. It should be noted that each algorithm has their advantages and disadvantages.

3.2 ALGORITHMS USED TODAY

It is important to note that because each algorithm has their advantages and disadvantages, that there is no "best" algorithm, only different kinds. If we examine the scheduling algorithms used by some of the more prominent operating systems used today, we can use them as examples as to why the claim is true.

3.2.1 Windows Scheduling Algorithm

Since early Windows NT based operating systems, a multilevel feedback queue with a round robin algorithm was used (please see Figure 6). This combination had the intention to meet the following design requirements: Give preference to short jobs, give preference to I/O bound processes, and separate processes into categories based on their need for the processor [9]. Multiple First In First Out queues are used as well. Their operations are as follows: New process is positioned at the end of the top-level FIFO queue. Eventually that process reaches the head of the queue and assigned the CPU. If the process completes, it leaves the system. If the process voluntarily relinquishes control it leaves the queuing network and when the process becomes ready again, it enters the system on the same queue level. If the process uses up its quantum time slice, it is pre-empted and positioned at the end of the next lower level queue. This will continue until the process completes or it reaches the base level queue. When it reaches this level, it will circulate in a round robin fashion until completed. Please note, if a process blocks for I/O, it is moved up one level and placed at the end of the next higher queue. This allows for I/O bound processes to be favored by the scheduler and allows processes to 'escape' the base level queue [9] Below (Figure 5) is an example of Windows XP priorities for tasks.

Windows vista introduced a modification in the scheduler where it uses cycle counter registers instead of interval-timer interrupt routines. Vista also introduced a priority scheduler for the I/O queue so that defragmenters and other similar programs will not interfere with foreground operations [9].

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Figure 5.12 Windows XP priorities.

Figure 5.

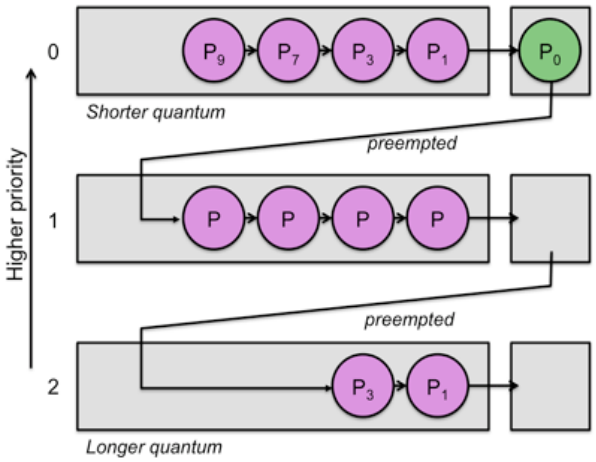

Figure 6. Each different operating system has different numbers of priority levels.

3.2.2 Apple Scheduling Algorithm

Mac OS 9 introduced cooperative scheduling, where one process controls multiple cooperative threads and also provides preemptive scheduling for multiprocess tasks. All Process Manager processes run within a special MP task called "blue task" and these processes are scheduled cooperatively using a round robin scheduling algorithm. Each process still has its own copy of the Thread manager and still cooperatively schedules a process's threads.

Also, courtesy of Apple themselves, they have put up a Kernel Programming Guide which includes the Mac scheduling algorithm used (please see Figure 6). the multilevel feedback queue is also used except unlike for the one for Windows, this one only has 4 separate bands for threads. All threads are scheduled preemptively and supports cooperatively scheduled threads in Carbon.

3.2.3 Linux Scheduling Algorithm

Linux 2.4 started out also using a multilevel feedback queue. However, unlike Apple and Microsoft, Linux increased the level count to include 0 through 140. The first 100 from 0 to 99 are reserved for real-time tasks while the rest are considered "nice" tasks levels. "Nice" task levels can have different priorities based off of their "niceness" with -20 having the highest priority and 19 or 20 being the lowest priority. Default niceness is usually set at 0. The interpretation of niceness is based off of how the scheduler is designed on that implementation of Unix. You can think of "nicer" processes allowing other processes to go first because they are so "nice". The time quantums for each level is 200ms and 10ms respectively. The scheduler goes through the run queue of all of the ready processes, again letting the highest priority processes go first and run through their quantums after which the process will be placed in an expired queue. Once the active queue is empty, the active and expired queues will switch back and forth until all processes complete [1].

Developed by Ingo Molnar and improved upon by Con Kolivas, the O(1) scheduler replaced the previously used O(n) and uses constant time scheduling services since Linux 2.6.0. This improvement helps minimize overhead and application programmers who use them will endure less of a performance impact [1].

Eventually in 2.6.23 Kolivas introduced the Completely Fair Scheduler as a replacement for the O(1) scheduler due to the fact that the Completely Fair Scheduler runs in O(log n) time. The Completely Fair Scheduler also uses a Rotating Staircase Deadline (example Figure 7) to ensure that the lowest priority tasks still received some CPU access time. The Scheduler also uses a red-black tree (example Figure 8) which holds the previous runtimes of tasks and from the tree, can then efficiently pick which used the least amount of time [1].

3.3 IDEAS ON USING THE SCHEDULER

IDE's in the future can help make programmers' lives easier. Some possible features can include "marking" lines of code to synchronize them. A scheduler can hold onto a specific line of code and pause all threads within a process and notifies other schedulers of its status. When another scheduler receives a

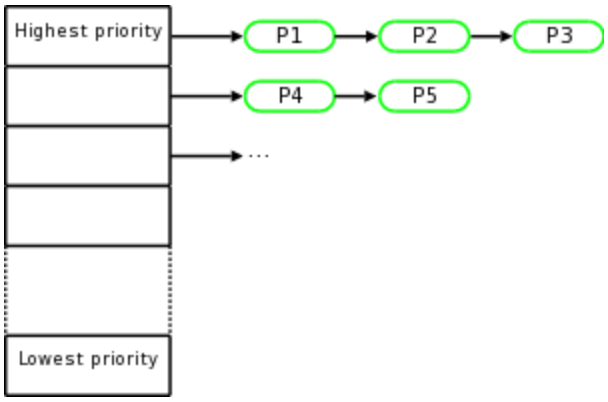

Figure 7.

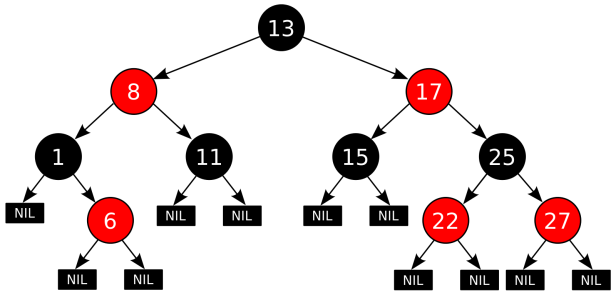

Figure 8. Note that all the NIL children are black.

mark, should the marks correspond to one another, then the two schedulers can run in synchronicity. The schedulers should be interruptible, though, in order to ensure no deadlocks or infinite waiting that can cause problems for computers even when the process ends.

It is also possible through the use of timed algorithm switching and quantum time slice alterations that the user can change the throughput of the scheduler in order to control the speeds. This benefit will be covered in the next section.

4. THE EFFECTS ON PARALLEL PROCESSING FROM THESE TWO METHODS

This section will pertain to explaining what potential effects these two methods will have when used within a Parallel Processing environment.

4.1 GENERAL THROTTLING BENEFITS

By slowing down the speed of CPUs the user can use this increase in control to allow for improved synchronicity within their code or other running tasks. Using these methods can help the user to avoid possible deadlocks, race conditions, and other errors that are visible with foresight and careful runthrough of the program or task. One example of a specific problem that can be avoided by using these methods could

be mutual exclusivity in programs where writing to the same piece of data can potentially cause a parallel processing hazard. By slowing down one of the accessors, the user is able to keep the CPUs going at a faster rate than if they were to pause one the CPUs during the accessing portion.

4.2 GENERAL THROTTLING DETRIMENTS

There may be benefits to using these methods but there are also some negatives with trying to throttle the CPU. In order to change the speed you'd need to know what you're doing. You would need to know when to change the voltage or frequency of the CPU or what scheduling algorithm to use to effectively use them. In the case of voltage change it would also be a potential problem to know exactly how long it would take for the CPU to return to normal processing speeds after the slowdown has stopped. Even with software, the user has to be able to program out solutions in order to avoid these problems. The common Parallel Processing bugs can still exist in the code but in other regions now. Always be mindful of possible lurking bugs. Because of the extra care needed to handle these problems, your program/process can slow down quite a bit even if you're careful. A fully functioning and complete Parallel Processing IDE is still in development so it is up to the user to be mindful of what is going on in their program.

5. FROM DIFFERING STANDPOINTS

Depending on what viewpoint the user has, their views on the potential net benefit from using these two methods changes. All of the benefits discussed in section 4.1 are considered unabated and still rely on the situation. However, from a business perspective looking at the usage of these methods, the user would be appalled. The reason being is because of the huge increase in costs. With creating a standard parallel processing program, the complexity of the problem should increase at a near linear rate, which would describe the relationship of having to accommodate more time, and by extension money, towards solving the extra amount of complexity. With using these methods, the complexity of the problem now becomes exponential and the net profits from the increase in control would not outweigh the new costs and increase in time, solving the new large curve of complexity and the increase in programmer skill required to be able to complete the task.

6. CONCLUSION

It is possible that by using these two methods, the user is able to achieve a net benefit with their system, however, in order to achieve this, great diligence and care must be taken in order to ensure the detriments are either avoided or negated as much as possible. The decision of whether or not to use these methods depends on the situation the user is in. In the end it would come down to what the user's viewpoint is, whether the user view it only from an engineering standpoint or take on a more business oriented view.

7. REFERENCES

[1] Aas, Josh. "Understanding the Linux 2.6.8.1 CPU Scheduler." Josh Aas. Silicon Graphics, Inc, 17 Feb. 2005. Web. 11 Apr. 2014. <http://joshaas.net/linux/linux_cpu_scheduler.pdf>

[2] Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. "The Abstraction: The Process." Operating Systems: Three Easy Pieces. 0.7 ed. Madison: Arpaci-Dusseau Books, Inc., 2014. 1-9. Print.

[3] Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. "Scheduling: Introduction." *Operating Systems: Three Easy Pieces*. 0.7 ed. Madison: Arpaci-Dusseau Books, Inc., 2014. 1-12. Print.

[4] Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. "Multiprocessor Scheduling (Advanced)." *Operating Systems: Three Easy Pieces*. 0.7 ed. Madison: Arpaci-Dusseau Books, Inc., 2014. 1-11. Print.

[5] Carter, Nicholas. *Schaum's Outline of Theory and Problems of Computer Architecture*. New York: McGraw-Hill Companies, 2002. Print.

[6] Newmarch, Jan. "Mutual Exclusion." *Techniques*. N.p., 16 Sept. 1995. Web. 9 Apr. 2014. <http://jan.newmarch.name/OS/l9_2.html>

[7] Mak, Ron. "CS149: Operating Systems", Operating Systems. San Jose State University. Department of Computer Science, San Jose. 5 Feb. 2014. Lecture.

[8] Mak, Ron. "CS149: Operating Systems", Operating Systems. San Jose State University. Department of Computer Science, San Jose. 10 Feb. 2014. Lecture.

[9] Russinovich, Mark E., and David A. Solomon. *Windows internals*. 6th ed. Redmond, Wash.: Microsoft Press, 2012. Print.

[10] "Tutorials Point - Simply Easy Learning."Operating System Multi-threading. tutorialspoint, n.d. Web. 9 Apr. 2014. <http://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

[11] "Dynamic Power Management In an Embedded System." *Lattice Semiconductor*. Lattice Semiconductor White Paper, n.d. Web. 10 Apr. 2014. <http://www.latticesemi.com/lit/docs/generalinfo/dpc_whitepaper.pdf>