Thread Switching Policies in a Multi-threaded Processor

Multi-threaded processors can run multiple programs at the same time. With this advanced capability comes the question, “How should processing bandwidth be divided amongst the various programs?” The algorithm used for this is known as the thread scheduling policy. This paper descibes the thread scheduling policies and capabilties of the MIPS32® 34K® processor core family.

Document Number: MD00010
Revision 01.00
May 25, 2006

1 Introduction

The subject of this paper is the algorithm for the thread scheduling policy. This algorithm controls how much execution bandwidth is allotted to each thread over time on the processor. In a single-threaded processor, this is not an issue, since there is no choice; the processor always runs the one thread available. However, in a multi-threaded processor, there are multiple threads to choose from. Which one should the processor choose? This question is at the heart of the thread scheduling policy. Here are the questions this paper answers:

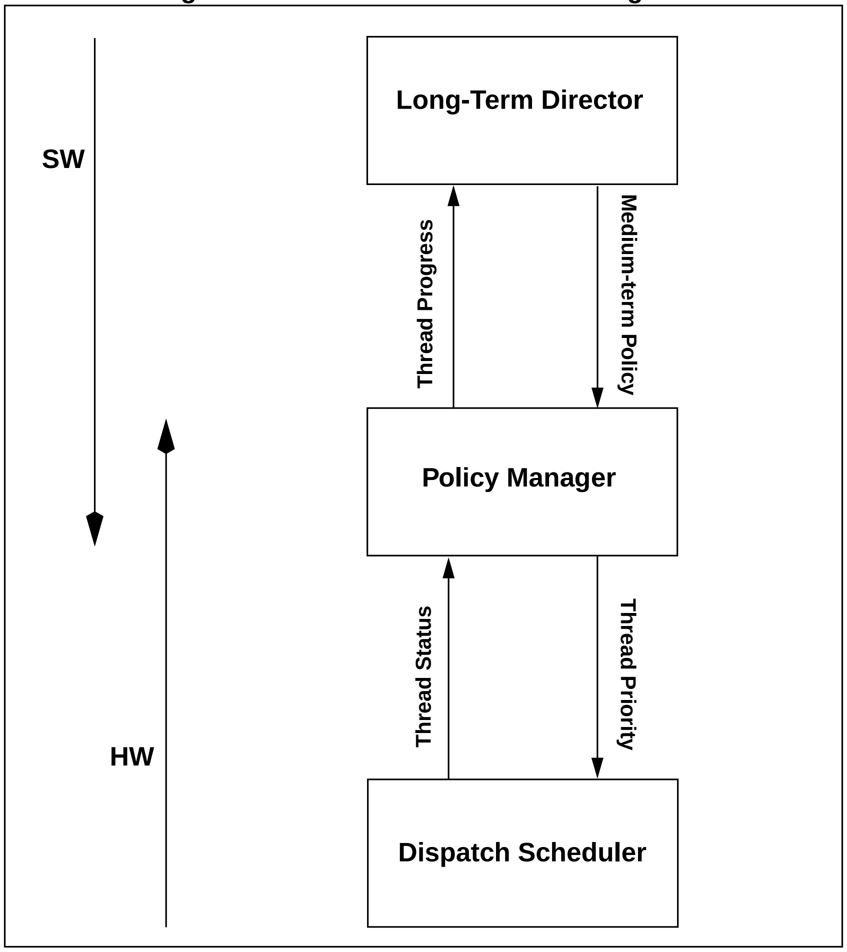
- What is a Policy Manager?
- What are the Policy Managers offered by MIPS?
- What more can be done with the Policy Manager?

This document is intended for hardware engineers, software developers, and system architects.

2 Policy Manager Defined

In the MIPS® 34K® processor core, thread switching is done at the dispatch point in the pipeline, not the fetch point. By doing it this way, the events which change thread status quickly affect the thread switch algorithm. The thread scheduling policy is implemented using a flexible combination of hardware and software. Figure 1 shows a block diagram of how these elements fit together. The following sections describe each element in detail.

Figure 1 Thread Scheduler Block Diagram



2.1 Dispatch Scheduler

At the lowest level is the Dispatch Scheduler (DS). This is a purely hardware module; it has no direct software programmability at all. It makes cycle by cycle decisions about which threads to run in the pipeline. The algorithm used in the DS attempts to maximize pipeline efficiency - it wants to keep the pipeline as full as possible. It follows these rules:

- **Only dispatch threads which are runnable.** Threads are not runnable when they are blocked due to a long-term event like a cache miss dependency, or a divide dependency.
- **If more than one thread is runnable in any given cycle, choose the highest priority thread as specified by the Policy Manager.** The Policy Manager can set each thread to one of 5 priorities (described below).
- **If there are multiple threads runnable and at the same highest priority, then rotate dispatch between them using a round-robin algorithm.** This enables a fair sharing of processor resources when the Policy Manager does not have a preference.

In the 34K processor core, the DS is an integral part of the processor pipeline, and has been carefully tuned for timing. In order to insulate customers from timing problems, it cannot be customized.

2.2 Policy Manager

The Policy Manager (PM) is what sets thread priority for the DS. It is a hardware module which has software control and programmability. The PM is designed to control the medium-term thread scheduling policy. It gets the following information from the DS about the status of threads:

- When a thread issues or completes an instruction
- When a thread is blocked, and why it is blocked
- When the processor takes an exception

Using the above information, the PM makes adjustments to the immediate thread priority so that, over time, threads get the correct allotment of processor bandwidth.

In the 34K processor core, the interface between the PM and the DS was designed to be non-timing-critical. Because of this, the PM can be customized to fit a specific application or system. The capabilities of the PM are explained in more detail later in this paper.

2.3 Long-Term Director

In many systems, some portion of the thread scheduling policy will need to be managed by software. The Long-Term Director (LTD) is the software module which implements this function: the LTD controls the PM.

In operation, the LTD periodically analyzes thread progress and massages the PM parameters. There is a trade-off between software and hardware as implemented in the LTD and PM, respectively. With a more sophisticated hardware PM, the LTD software can be simpler and be run less often, for less software overhead. Alternatively, the hardware PM may be relatively simple, but require that the LTD be smarter and/or run more often.

The interface between the LTD and the PM is implemented through privileged registers and is totally implementation defined. In other words, depending on your system, the PM can provide more or less information to the LTD and the LTD can have more or less control over the PM. The LTD may need a variety of system status and progress indicators. At a minimum, this includes thread execution progress. Depending on the system, the LTD could gather higher level task information from the OS and applications as well.

2.4 Quality of Service

Quality of Service (QoS) is a generic term which is used here to mean that different threads can get differing amounts of processor bandwidth. For instance, if you have a DVR system with a video decoding function running on one thread, and a menu drawing function running on another, then the decoding program is likely to be more important, since it needs to keep up with the video stream. The menu drawing program can be lower priority, since a human is waiting, and a few more milliseconds will not be noticed. So, in this example, better QoS is needed for the video decoding thread.

The thread scheduling policy as implemented on the 34K processor core allows intricate control over QoS. As stated earlier, the PM sets thread priority for the DS. If necessary, the PM can slow down or even stop threads completely in order to give more processing power to high-QoS threads.

Key to this capability is the knowledge of the progress that each thread is making in its particular task. At the hardware level, this can be tracked by the PM by counting instructions executed. As stated above, this could also be measured using higher-level software indicators transmitted to the PM via the LTD.

MIPS MT enables a direct upper-level communication from an application to the PM through the YIELD instruction as well. One operational mode of the YIELD instruction is used as a PM hint, but does not deactivate the thread. For instance, a video-decoder may know that it has a certain number of pixels to process to complete one frame. It can execute a YIELD instruction for every N pixels completed. The PM can see when a YIELD is executed, thus making it a progress indicator to the PM.

The normal YIELD function can be used to help QoS as well. In this mode of operation, YIELD deactivates a thread, making it non-runnable. If an application YIELDs its threads when they are complete, then this leaves the processor with more bandwidth to complete the remaining threads.

3 MIPS-Designed Policy Manager Options

With a basic understanding of the thread scheduling policy, we can now dive into the details of several policy manager modules offered by MIPS and included with the 34K processor core. These PM modules are fully verified and can be used as-is in customer systems. They are generic modules that are useful to a wide-range of applications and systems.

3.1 Simple Round-Robin

Round-robin is an algorithm which simply chooses all threads for dispatch one after another, switching threads every-cycle. In a 3 TC system, this would look like: TC1, TC2, TC3, TC1, TC2, TC3, etc. Note that the 34K processor core will not dispatch a thread that is not runnable, so if TC2 stalls for some reason, then the sequence might become: TC1, TC2, TC3, (TC2 blocks), TC1, TC3, TC1, TC3, etc.

In the 34K processor core, the DS implements a round-robin algorithm when it has multiple runnable threads at the same highest thread priority. Because of this, implementation of this policy is a degenerate case: No PM or LTD is required, just hardwire all threads to have the same priority, and it defaults to round-robin.

Round-robin works well in systems which want high pipeline efficiency. Switching threads every cycle tends to minimize pipeline hazards and microstalls very well. This option also uses minimal hardware, so it is very area efficient.

Because round-robin is defined to be fair across all TC's, it gives the same Quality of Service to all threads. Individual threads cannot be preferred over others. Because of the inherent fairness of this algorithm, it is most suited to systems which have no bandwidth allocation requirements.

The total lack of software control is a significant limitation as well. Since the other PM options can be programmed to implement round-robin, they are more flexible.

3.2 Manual (Software-Controlled)

This PM module has no intelligence in hardware. Software simply sets the priority for each thread and the PM hardware forwards this unchanged to the DS. The obvious advantage to this module is flexibility. Virtually any thread scheduling policy can be implemented since it is done by the LTD. Furthermore, it can be changed at reset or even on the fly. The area required for this option is still modest, so not much penalty with this option.

The disadvantage to using this option is that it can require significant LTD overhead to implement the correct policy. This will usually come about because the LTD process will need to be called more often so as to react as the system changes optimally. In systems where the policy needs little adjustment, this is a viable option, however.

NOTE: By setting every thread to the same priority, software can implement a round-robin algorithm.

3.3 Weighted Round-Robin

This option is a compromise between hardware control and software flexibility. It enables software to set each thread into one of 4 priority “Groups”, with each one getting approximately double the processor bandwidth of the next lower priority Group. In other words, each successively higher Group gets about double the weighting of the next lower Group.

This module implements a rough weighting for threads. If the system requires more fine-grained control of weighting, this can be implemented in the LTD by periodically adjusting the weights of each thread. Off-loading this periodic control to the LTD keeps the hardware PM simple.

Since this option contains a software LTD module, it can be used to implement QoS, although that part of the algorithm is not handled in hardware.

4 More Advanced PM Capabilities

The MIPS-designed PM modules implement general algorithms which we believe apply to a wide variety of applications. However, specific applications may benefit from more customized algorithms. Each of the above RTL modules is available to a 34K processor core customer as RTL source and can be modified and used as a “Custom” PM module.

4.1 Intelligent Priority Changes in Hardware

One example of this would be in a system based on the Weighted Round-Robin PM described above. An extension could allow hardware to implement the priority changes over time instead of software.

In a networking application, the leaky-bucket algorithm might be one of interest. In this algorithm, the work to be done by each thread is represented by water in a leaky bucket. As the water level rises, the water leaks out faster due to increased water pressure. You could implement counters which keep track of the work for a thread to complete, and when the work goes beyond certain thresholds, the priority for that thread is increased automatically by the hardware PM.

By implementing priority changes in hardware, you simplify the LTD and dramatically lower the overhead required in software to implement the policy.

4.2 Hardware QoS

The above MIPS-supplied PM modules only control the dispatch of instructions. They do not react to the completion of instructions. Hardware QoS could be implemented by monitoring the number of completed instructions per thread. If one thread falls behind, then its priority could be increased until it catches back up. More sophisticated schemes could be built using various other thread progress indicators.

5 Summary

In multi-threaded systems, the thread switching policy can dramatically impact the delivered performance and efficiency of the processor. The 34K processor core has an extremely flexible design which enables hardware and software to share the burden of implementing the thread scheduling policy.

MIPS offers several hardware modules which implement general-purpose policies. However, customers can change these and customize their policy to the specific needs of their application, if necessary.

6 Document Revision History

MIPS documents include change bars (vertical bars in the page margin) that mark significant changes to the document since its last release. Change bars are removed for changes which are more than one revision old.

This document may refer to Architecture specifications (for example, instruction set descriptions and EJTAG register definitions), and change bars in these sections indicate changes since the previous version of the relevant Architecture document.

Date	Revision	Description
May 19, 2006	00.01	First draft
May 25, 2006	01.00	First release