Performance Monitoring in Advanced Computer Architecture

Richard J. ENBODY[1], Kelley PELLINI[1], and William MOORE[2]

[1]Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824-1226
enbody@cse.msu.edu

[2]Sun Microsystems
Menlo Park, CA 94025
billm@eng.sun.com

Abstract
Imagine that you could directly monitor the performance of a processor while it is running. How would that affect a computer architecture course?

The current generation of microprocessors has performance monitoring registers on chip which can be read by users. The result is real-time monitoring of processor performance, and a new opportunity for computer architecture education. In particular, it allows experimentation which addresses the question "Should Computer Scientists Experiment More?" from the cover of the May 1998 IEEE Computer magazine.

This paper describes the incorporation of performance monitoring into a graduate-level, advanced computer architecture course based on Hennessy & Patterson's *Computer Architecture A Quantitative Approach*.

Introduction
Imagine that you could gather information from a running CPU with minimal overhead. What would this reveal about the inner workings of a CPU? On-chip performance monitoring has recently become available, and it presents interesting educational opportunities. To appreciate some of the potential of such a tool consider Figure 1 which shows CPU behavior data taken from UltraSPARC I performance registers while 129.compress from the SPEC95 benchmark suite was running. The data were read from the performance registers using PERFMON, a tool for CPU performance monitoring which can read the UltraSPARC performance registers.

Hennessy & Patterson's *Computer Architecture A Quantitative Approach* is a popular textbook which combines performance equations with simulation results to quantitatively consider architectural issues. Students can explore the equation-based analysis with pencil and paper. Simulators based on the generic DLX architecture have been developed to allow students to observe and explore pipeline execution. The missing leg is exploration of actual chip performance. On-chip performance registers provide student with that exploratory opportunity.

The approach used at Michigan State University is to assign some homework based on exercises from Hennessy & Patterson's text to exercise the analytical equations and have other assignments which explore the DLX architecture using available simulators. Those assignments are weekly and are kept relatively short. Processor performance is then explored in a semester-long project using PERFMON.

The next section will describe performance monitors in more detail followed by a section describing a set of student projects to illustrate the variety that is possible.

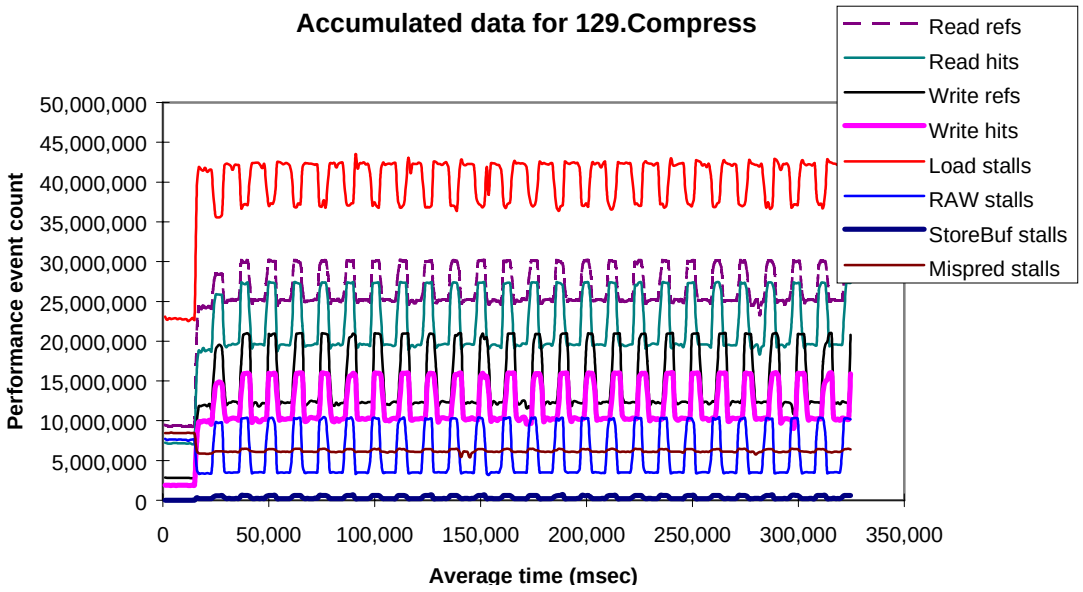

Figure 1 UltraSPARC Performance Data from 129.compress (SPEC95)

Performance Monitors

A Generic Performance Monitor

Performance registers are used by performance monitors to count a set or group of events occurring in the CPU or memory hierarchy. Usually there are two performance registers: the Performance Control Register (PCR), and the Performance Instrumentation Counter (PIC), as they are referred to in the UltraSPARC I architecture. The PIC is equally divided into two 32-bit counters: PIC0 and PIC1. When gathering performance data, the first step is to determine two events of interest. An event is a pre-defined list of actions that occur in the CPU: for example, the number of read references to the level one cache or the number of read hits to the level one cache. Then the user must convey the chosen events to the PCR (step A, Figure 2). Then the PIC register is cleared to prevent previous events from being included in the current run. In order to have events to count, the CPU must be doing work, and a program is run to "exercise" it. As an event occurs during a run, a signal is sent into one of a pair of multiplexers (mux) (step B, Figure 2). The PCR acts as the control signal for the mux (step 3, Figure 2) and only allows the event signals defined in the PCR to pass into PIC0 or PIC1. PIC0 or PIC1 is incremented when it receives a signal (step D, figure 2).

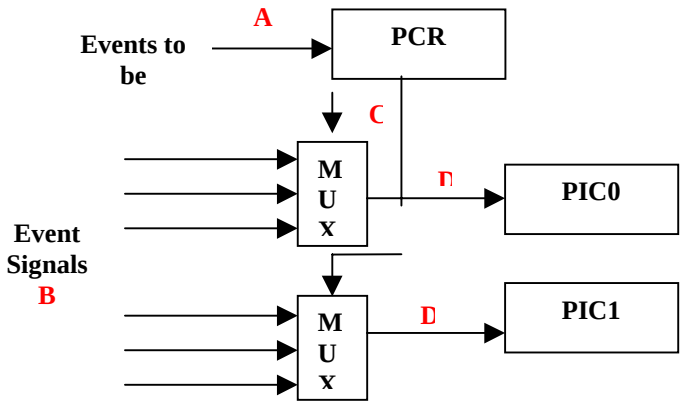

Figure 2 The PCR control signal determines which event will increment the PIC

Since current processors with performance monitoring registers are pipelined, low-level mechanisms are needed to provide barriers before and after monitored code. In addition, routines are needed to flush various levels of the memory hierarchy, especially caches.

PERFMON

PERFMON was developed at Michigan State University. It runs under Solaris and can monitor the performance registers of the UltraSPARC I and II, the Pentium Pro, and the Pentium II. It uses a loadable device driver that allows user-level code to access the performance registers. Its advantages include low overhead, an auto_install feature, and user accessibility. The design of PERFMON is interesting, but is beyond the scope of this abstract.

Other Performance Monitors

Digital offers their Digital Continuous Profiling Infrastructure (DCPI) which collects data based on samplings from the performance counter of the Alpha platform. SGI has Perfex which is designed to non-intrusively monitor the MIPS R10000 chip. Linux tools have been developed to access Intel's Pentium and successors. Also, tools exist which run on NT which can access Pentium performance registers.

Student Examples

One of the greatest advantages of real measurements is that the experimental platform is quite noisy. One characteristic of the UltraSPARC performance counters is that they have no concept of process. That is, the counters count everything on the chip whether it is in your process or another process. Interrupts and other operating system effects add a lot of noise to data. We make use of Solaris' real-time class (using `priocntl -c RT`) to keep other processes at bay, but much noise remains. Making sense of the noise and eliminating as much noise as possible is a wonderful exercise both in understanding how much interference occurs as well as an exercise in experiment design. Furthermore, insufficient information exists about various features to cleanly eliminate them. Our department will have an “isolation lab” available for students the next time the course is run where students will be able to run experiments detached from a network and where students will have the ability to modify the operating system to eliminate noise.

A sampling of student projects is provided—more will be provided in the full paper.

Compilers and Machine Performance

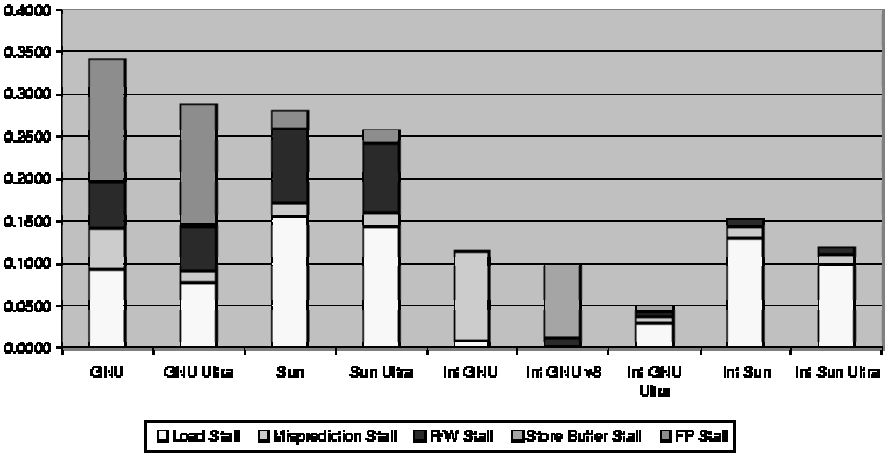

Figure 3 The various types of pipeline stalls that occur with different compilations of gsm.

Students studied how the compiler affects machine performance by comparing compilations of a program: one compiled specifically for the UltraSPARC, and another compiled for older models of the SPARC. Two different compilers were used (Sun & GNU) on two multimedia applications (audio & video) compiled for different architectures using integer or floating-point multiplication. The number of pipeline stalls and type of stall for each instruction were measured. The results showed that the type of stall affects performance and clock cycles per instruction (CPI). The types of stalls found for the audio program *gsm* are shown in Figure 3. The vertical axis is the percentage of CPI accounted for by the stall. Analysis cannot be presented here, but one can clearly see a wealth of data in this figure.

Incremental Sampling Method

One group wished to regularly sample a run to measure cache performance with respect to time. To accomplish that task the group developed Incremental Sampling Method (ISM), a procedure that can be used to periodically sample the registers. ISM works by creating a separate thread that sleeps using the UNIX sleep() call and then wakes up to read the PIC register. Figure 1 shown earlier presents data taken with PERFMON and ISM from runs of SPEC 95's 129.compress. It is easy to see that trends exist in the data.

The same group used PERFMON and ISM to illustrate the effect of context switches (Figure 4). A small program that uses a loop to access sequential elements of an array produced the readings in the left figure. When the UNIX facility *top* is run (*top* periodically gives a list of the top 18 processes) in the background and the looping program was run, the right figure was produced. The *top* facility updates the display of processes every 5 seconds, and the spikes occur approximately every 5,000 milliseconds (5 seconds). The large spikes in the right figure indicate where a context switch occurred as *top* took the CPU to list the processes.

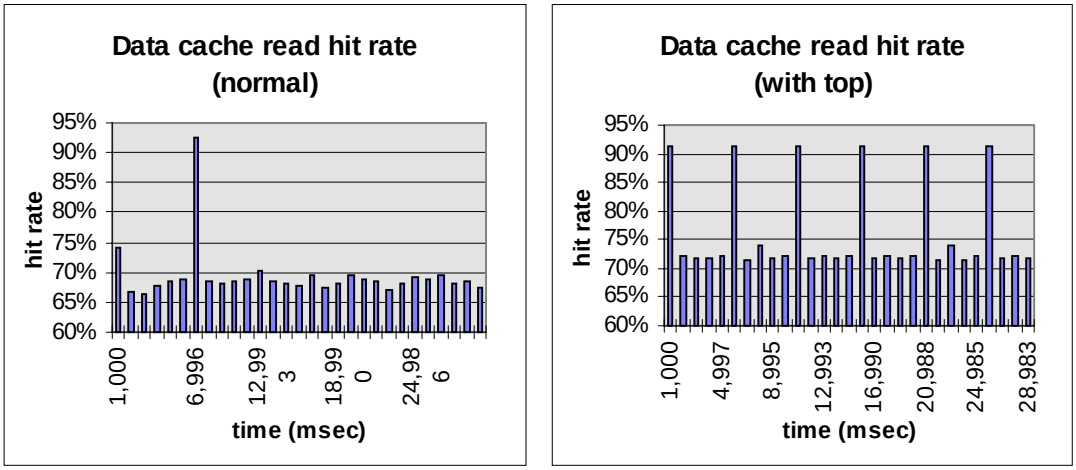

Figure 4: The Data cache read hits without top running and with top running

Conclusion

Performance monitors can be used in multiple ways; hardware design and analysis, software design and analysis, and as an educational tool. Students can gain valuable knowledge from monitoring the performance of a CPU. Even students that do not become chip designers will benefit because the performance of a chip is closely tied to other aspects of computer science. Also, it allows students to explore the design process—a process that translates to other areas such as software engineering.