Performance monitors for multiprocessing systems

Alexey Borisenko

School of Information Technology
University of Ottawa

ELG7187, Fall 2010

Outline

1. Introduction
2. Monitors classification
   - Performance counters for autotuning and scheduling
   - OS connection
   - Trace implementation
3. Research directions
4. References

Performance monitors

Performance monitors collect run-time data, which can be used for:

- Analysis
- Verification
- Management (i.e. scheduling)

What needs to be monitored?

Software: Inefficient algorithms, synchronization, load imbalance.
Hardware: Instruction scheduling, memory stalls, interrupts, branch misprediction.

For multiprocessor systems, a list of 23 performance metrics was set out[Lemieux96]. These show all the possible sources of performance loss in multiprocessor systems.

Important metrics from the subset:

1. Dynamic instruction count
2. Cycles lost due to NOPs and pipeline slips
3. Cycles lost due to stalls, pipeline flushing and restarting
4. TLB faults
5. Branch prediction
6. Cache-profile watchpoints or thresholds

Types of monitors[Liu89, Jain91]

Software vs Hardware vs Hybrid

Depending upon the level at which a monitor is implemented. Most common classification.

Passive vs Active

Observes, collects, stores vs analyses the data and takes corrective action

Centralized vs Distributed

Taps into critical points vs associates multiple monitors with corresponding processors

Event-driven vs Sampled

Activated only by the occurrence of certain events vs activated at fixed time intervals.

Real-time vs Batch monitors

Display the system state either continuously vs only collect data, let other programs analyse.

Real-world software monitors

`gprof`[Graham82]

Records the number of calls to each function and the amount of time spent there, on a per function basis, not cycle accurate.

Quartz[Anderson90]

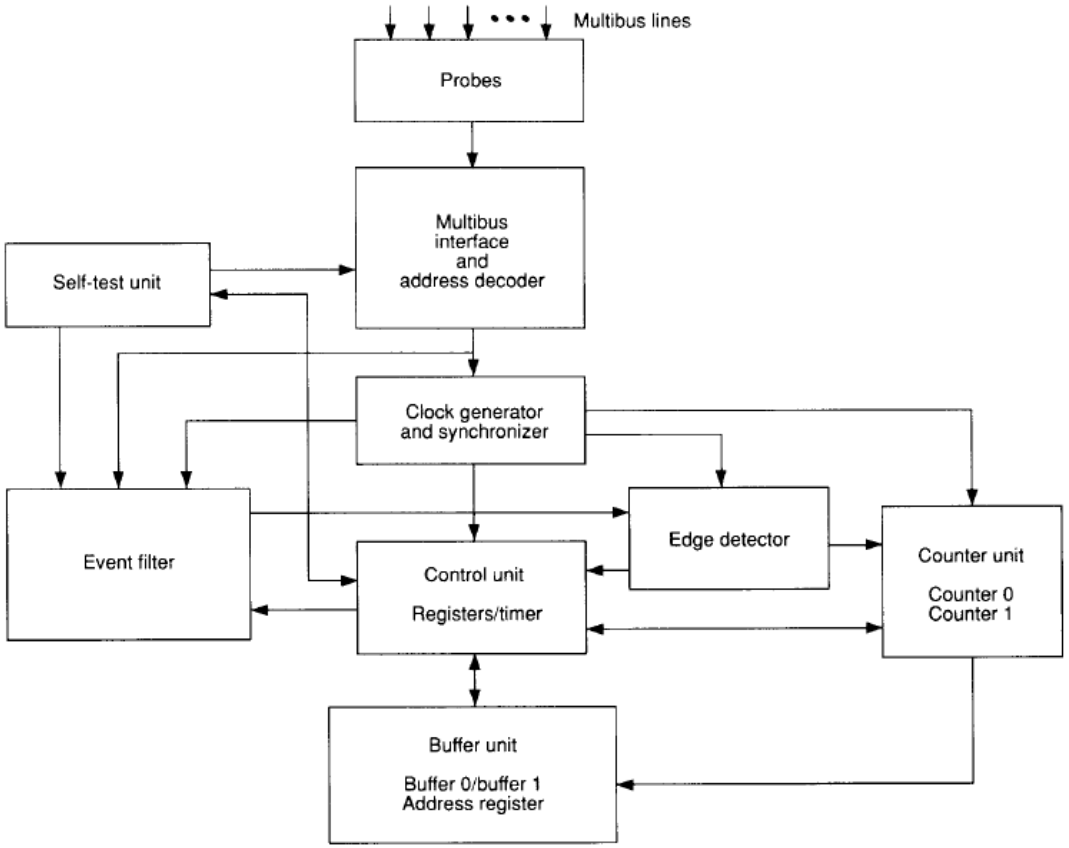
Principal metric of Quartz is normalized processor time: the total processor time spent in each section of code divided by the number of other processors that are concurrently busy when that section of code is being executed. A software sampling monitor.

Paradyn[Miller95]

Performance tool that can automatically search for performance problems in large-scale parallel programs.

Real-world hardware monitors

Ted[liu89]



Real-world hardware monitors

DASH perfromance-monitoring[Dash93]

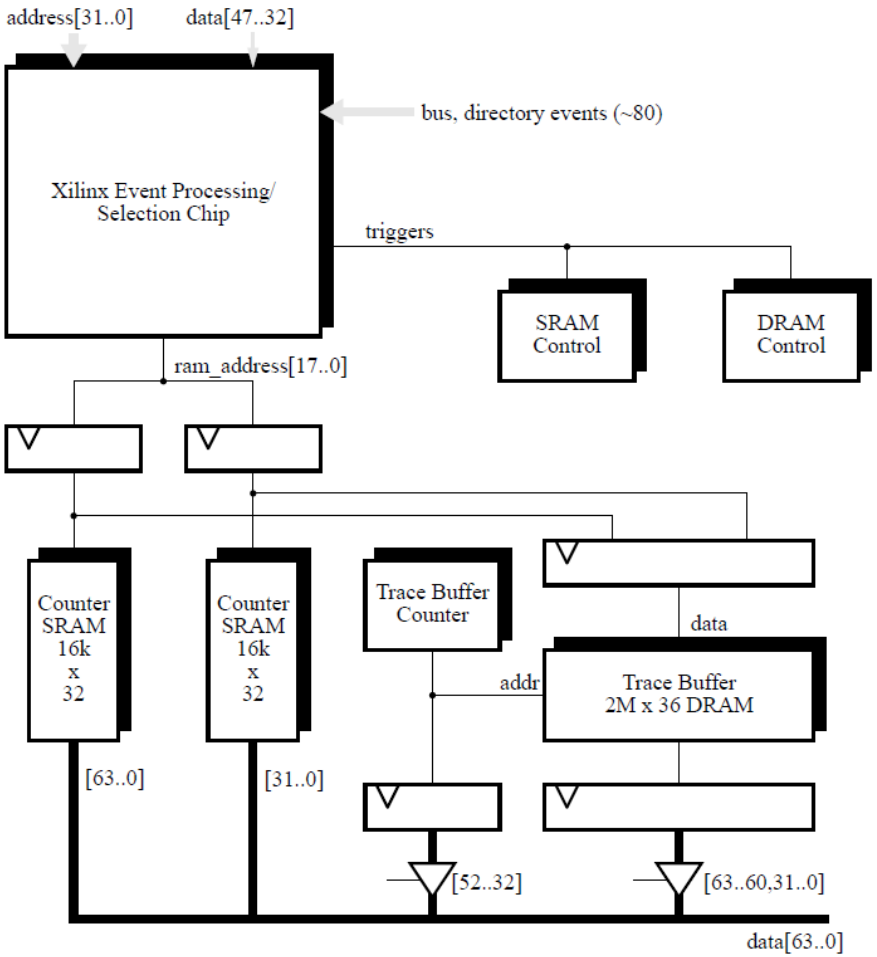


FIGURE 2.2. DASH performance-monitoring hardware.

Outline

1 Introduction

2 Monitors classification
- Performance counters for autotuning and scheduling
- OS connection
- Trace implementation

3 Research directions

4 References

Three scheduling policy aspects employed in a CMP platform with multiple shared caches:

1. Better to schedule applications that contend less with each other on the same cache
2. It is better to schedule applications or threads that share
3. It is better to affinitize applications to caches if their working set is still available on the cache from the previous scheduling interval data on the same cache

CacheScout[Zhao07]

A monitoring tool of shared caches in CMP platforms. A CacheScout enabled scheduler looks into detecting sharing between threads and employing this knowledge to co-schedule sharing threads on the same cache. Lower overhead than current methods.

Outline

1. Introduction
2. Monitors classification
   - Performance counters for autotuning and scheduling
   - OS connection
   - Trace implementation
3. Research directions
4. References

Connection between Software and performance counters

Performance Application Programming Interface (PAPI)

Common interface to performance-monitoring hardware for many different processors. Gives access to instruction counts, clock cycles, cache misses. Usage: DynaProf, SCALEA, TAU, SyPablo.

Intels VTune Performance Analyzer

Performance analysis tools such as call graph profiling, processor-specific tuning advice and cycle-level profiling.

The Rabbit Performance Counter Library [Heller]

Provides a high-level interface to Intel and AMD processors on Linux systems. Read and manipulate Intel or AMD processor hardware event counters in C under the Linux operating system

Outline

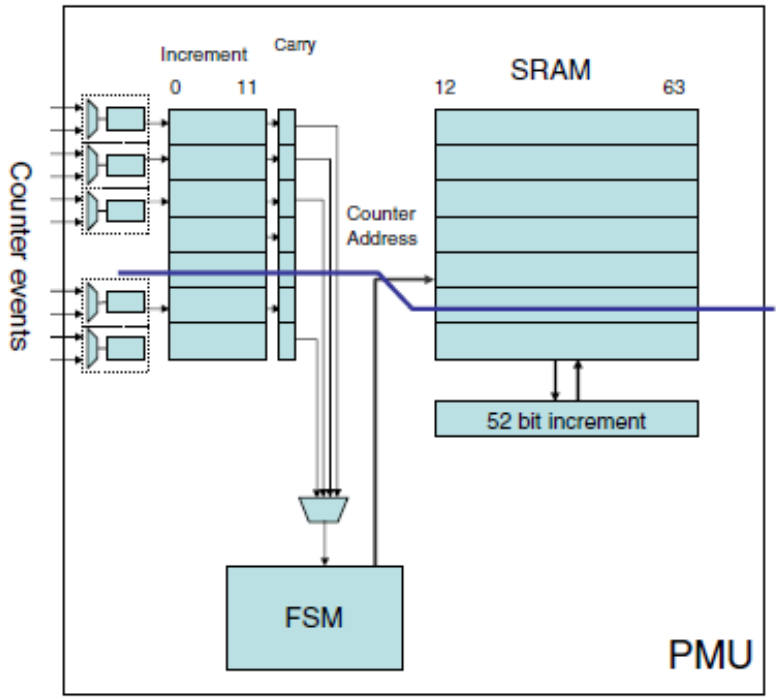
1. Introduction
2. Monitors classification
   - Performance counters for autotuning and scheduling
   - OS connection
   - Trace implementation
3. Research directions
4. References

`Tracing` - a method for debugging in computer programming.

Intel Trace Analyzer and Collector

Timeline view shows actual function calls and process interactions highlighting excessive delays or errors that stem from improper execution ordering

Tracking large numbers of concurrent events. Supporting more types of events: core events, memory hierarchy and coherence, I/O and network traffic. Implemented in BlueGene/P system



References [Lemieux96]Hardware Performance Monitoring in Multiprocessors
[Liu89]Hardware Monitoring of a Multiprocessor System
[Jain91]The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling
[Graham82]Gprof: A call graph execution profiler
[Anderson90]Quartz: A tool for tuning parallel program performance
[Miller95] The Paradyn parallel performance measurement tool
[Zhao07]CacheScouts: Fine-Grain Monitoring of Shared Caches in CMP Platforms
[Dash93]The DASH prototype: Logic overhead and performance
[Heller]Performance-Monitoring Counters Library, for Intel/AMD Processors and Linux