# Rapid Prototyping in Architecture Research using Hardware Hooks in COTS Systems

Smruti R. Sarangi, Brian Greskamp and Josep Torrellas
Department of Computer Science, University of Illinois
http://iacoma.cs.uiuc.edu

## Abstract

This paper makes the case for using little-known hardware hooks in COTS (Commodity-Off-The-Shelf) computer systems to prototype architectural research ideas. We survey a set of features in Intel processors and chipsets that can help us estimate the overheads and benefits of proposed architectural enhancements. We show that these features have diverse uses in performance evaluation, power management, reliability, and programming language support.

## 1 Introduction

It is well-known that architectural simulators are slow and/or not very accurate. Nevertheless, they are an indispensable tool in architectural evaluations because they are very flexible and results are repeatable. While FPGAs and full custom designs are more accurate, they are significantly more difficult to design and verify. Furthermore, the results may not be repeatable.

In this paper, we argue that there exists an intermediate approach that, for a set of scenarios, can be a better tradeoff than either simulation or hardware prototyping. This approach involves using little-known hardware hooks in existing COTS processors and chipsets that can be used to emulate certain architectural enhancements. These hooks allow us to create simple and efficient prototypes in a short time period. Moreover, evaluations are significantly faster and more realistic than with simulations, benchmarks are run to completion, and operating system and network interactions are accounted for.

In the following, we first list the hardware hooks in a typical COTS processor (Section 2), describe a set of architectural ideas that can be prototyped using these hooks (Section 3), and then conclude.

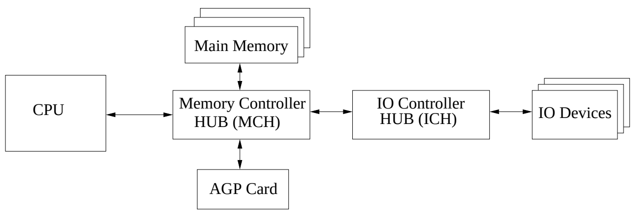

Figure 1: Intel chipset architecture.

## 2 Hardware Hooks

Figure 1 shows the architecture of a typical Intel chipset. The main components in this system are: processor(s), Memory Controller Hub (MCH), and IO Controller Hub (ICH).

**Processor** : We can modify the processor for architectural prototyping by modulating the duty cycle of its clock, applying dynamic voltage-frequency scaling (DVFS), turning hyper-threading on and off, and writing back or invalidating caches or TLBs. Many of these facilities are not available to the normal user. They are enabled by privileged-mode instructions that can be issued by a program running with supervisor permissions. One way of using such instructions is to embed them in a kernel module in Linux.

**MCH** : The MCH controls the main memory, graphics card and the bus. We can typically change the properties of main memory (e.g., refresh rate and line scrubbing policy), main memory access scheduling policies, bus traffic, ECC support, and memory power management policies by reconfiguring the MCH. The MCH is visible to system software as a set of PCIX registers. A subset of these registers were formerly PCI registers. The current PCIX register set subsumes the PCI registers. All the PCIX registers are memory mapped and are accessible by any user with superuser permissions. In Linux, the superuser can access any memory location, including memory-mapped PCIX registers. By setting the values of these locations, it is possible to control different attributes of the MCH. Further details are in [2].

**ICH** : The ICH controls the attributes of the I/O controller. Its interface is similar to the MCH. Both of them use PCIX registers for communication. We can configure the ICH to change DMA policies and to change the attributes of the controllers for the USB, disk, LAN, and other devices.

## 3 Examples of Prototypes

In this section we enumerate a few of the ways in which these hooks have been (or can be) used to prototype ideas.

### 3.1 Performance Evaluation

COTS systems can be shaped into different machine configurations for performance evaluation. For example, systems using the Intel SpeedStep technology can dynamically change their frequency and the duty cycle of their clock. Moreover, it is possible to change the latency to main memory and its bandwidth, disable/enable memory banks, and change the timing of messages to memory. These hooks allow the study of the effects of memory latency and bandwidth, memory size, and processor frequency on system performance. Current processors also have an extensive set of hardware performance counters [1] that can provide further insights.

We can also change multiprocessor-related characteristics. For example, we can dynamically change the number of processors or the number of hardware contexts. We can also change the memory ordering, e.g., from TSO to PSO in SPARC processors.

Finally, another interesting feature is the ability to dynamically remap DRAM addresses. This is useful, for example, to optimize accesses that follow scatter-gather patterns, through physical memory remapping. Such support could emulate the schemes proposed in the Impulse project [9].

### 3.2 Power and Temperature Management

The typical means of system reconfiguration for power and temperature management is some form of dynamic voltage and/or frequency scaling. We can do it by slowly ramping up the voltage and waiting for the PLL to relock, or by modulating the duty cycle of the clock as in Pentium-M [4]. We can also estimate power consumption, temperature, and their phases using hardware performance counters [5, 6].

In an Intel system, it is also possible to regulate the power and temperature of the memory. Each DIMM has a counter that counts the activity in a certain window of time. It then compares this activity to a threshold to identify temperature or power violations. If the activity exceeds the threshold, then traffic to that DIMM is regulated. It is possible to modify the thresholds and thus control memory temperature and power.

### 3.3 OS and Compiler

There are multiple ways in which COTS systems can be made to emulate special support for OS and compiler features. For example, it is possible to configure modern Intel systems in virtual machine mode. In this case, the system separates part of the memory space. This memory space can be used as a scratch pad, can implement an in-memory buffer for storing encrypted data, or can be used to run a separate OS. We can then have specific policies for I/O and interrupt handling.

Another example has to do with support for run-time type checking and security. The Pentium memory space has several segments [3]. Each segment descriptor has a 4-bit type field that can be set by the user. As a result, the user can set a segment to contain only objects of type $X$. If any task that is not supposed to touch an object of type $X$ accesses the segment, there is an exception. This feature can be used to support compiler optimizations. It can also be used for security research. For example, let us assume that objects of type $X$ are classified. With this feature, we can ensure that accesses to these objects are password-protected.

### 3.4 I/O

The attributes of I/O devices can be changed by reconfiguring the ICH. For example, it is possible to change the temperature and power thresholds of the disk. This is useful to researchers who are studying performance-power tradeoffs for disks. We can also change the DMA policy from burst mode to cycle-stealing mode. This can be done dynamically in response to changes in application characteristics.

### 3.5 Reliability

There are several examples of using the proposed techniques as tools for reliability research. For example, [7] focuses on detecting memory leaks in a real system. The authors mark a set of suspected main memory addresses by corrupting their ECC. If these addresses are later accessed, there is an ECC failure. At that point, the debugging software strikes the addresses out of the list of addresses suspected of being involved in a memory leak. In this work, the authors not only reconfigure the MCH to allow software to corrupt the ECC; they also turn the scrubbing hardware off, so that the incorrect ECCs are not corrected.

As another example, [8] looks at the problem of cycle-accurate deterministic replay for hardware debugging. The authors perform multiple changes to emulate a fully-deterministic computer system. They include: increasing the bus latencies to deterministically set the message transmission delays to worst case, change the refresh and scrubbing policies to make them deterministic, and write back and flush caches and TLBs to perform checkpoints. The first two changes are achieved by setting the appropriate bits in the MCH. Writing back and flushing caches and TLBs is accomplished by using the WBINVD instruction.

## 4 Conclusion

We observe that there are a multitude of hardware hooks in COTS systems that can be used to emulate hardware features for architecture research. We encourage the research community to take a close look at them.

## References

[1] Intel VTune performance analyzer. http://www.intel.com/vtune.

[2] Intel E7525 Memory Controller Hub (MCH) Datasheet, 2003.

[3] Intel Corporation. Pentium processor system programming manual.

[4] S. Gochman et. al. The Intel Pentium M Processor: Microarchitecture and Performance. *Intel Technology Journal*, 7(2), May 2003.

[5] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA*, 2006.

[6] K. Lee and K. Skadron. Using performance counters for runtime temperature sensing in high-performance processors. In *IPDPS*, 2005.

[7] F. Qin, S. Lu, and Y. Zhou. Safemem: Exploiting ECC-memory for detecting memory leaks and memory corruption during production runs. In *HPCA*, 2005.

[8] S. R. Sarangi, B. Greskamp, and J. Torrellas. CADRE: cycle-accurate deterministic replay for hardware debugging. In *DSN*, 2006.

[9] L. Zhang et al. The impulse memory controller. *IEEE Trans. Computers*, 50(11):1117–1132, 2001.