Using Hardware Event Counters for Continuous, Online System Optimization: Lessons and Challenges

Christos D. Antonopoulos and Dimitrios S. Nikolopoulos
Department of Computer Science
The College of William and Mary
McGlothlin–Street Hall
Williamsburg, VA 23187–8795

Most modern processors offer hardware support for monitoring performance events related to the interaction of applications with specific subunits of the processor [4, 7, 8, 9, 10]. The insight attained from performance monitoring counters is useful for both application programmers and processor manufacturers. Programmers typically employ them as a powerful tool for post-mortem analysis, identification and resolution of performance bottlenecks in their applications. Processor manufacturers, on the other hand, can collect valuable information on the performance of their products while the latter are used in production environments. This knowledge is then exploited during the design phase of future products.

Our project, MOHCA (*MOnitoring of Hardware for Continuous Adaptation*) exploits performance monitoring counters in a different way. The counters are used for online monitoring of hardware events. The information collected is fed back to OS scheduling policies, providing them with awareness of the dynamically changing characteristics of the execution environment and allowing them to continuously adapt to these characteristics and reach more educated scheduling decisions. The scheduling policies have been implemented in the context of a processor manager, i.e. a server process which applies kernel-level scheduling decisions from user-level. Although this approach introduces practically negligible overhead, since counters do not need to be sampled frequently, it has been totaly neglected by current OS schedulers. We have already implemented a successful prototype in Linux, and used it to efficiently schedule workloads on SMPs consisting of multiple Intel HyperThreaded (HT) processors. The same prototype can be used on generic multi-SMT or future multi-CMP systems. The performance gains due to the use of feedback-driven scheduling policies are significant, even in current architectures. Moreover, the performance impact is projected to be huge in future multicore architectures, where a full-blown multiprogrammed/multiprocessor operating system with distributed functionality will be contained in a single chip. More information and results from this project are presented in recent publications [1, 2, 6].

The current implementation of HyperThreaded (HT) processors from Intel shares performance monitoring hardware among both execution contexts available on the same physical processor. As a result, conflicts may arise if both threads executing on the same processor attempt to use performance monitoring. The strategy typically adopted by system software to deal with the problem is to disallow the execution of two threads using performance counters on the same physical processor. This restriction has adverse effects on the usability of performance monitoring facilities on HT processors. In particular, disabling threads makes online performance monitoring for continuous adaptation infeasible. We have managed to overcome this limitation by using 2 sets of performance monitoring registers for each event

to be measured. Both sets are activated at the thread which is executed on the first virtual processor of each physical processor. At the same time, we take advantage of a bitmask in the performance monitoring configuration registers which allows the association of each event with the virtual processor that triggered it. Given that the processor manager has full control on the execution of threads on virtual processors, events can be correctly attributed to specific threads at the end of each scheduling quantum.

Our thesis is that continuous, online system optimization using feedback from hardware counters is not only a useful but also a necessary measure to optimize performance and productivity while taming the complexity of high-end systems. It is therefore important to consider both hardware and system software support for transparent, accurate and portable online monitoring of hardware events. The current hardware and software support for this task is immature and although some workarounds can be applied from user-level with minimal, or no kernel-level modifications, as HT, SMT and multicore processors become gradually more widespread, a systematic solution, either at the hardware- or system software-level is required. In the rest of this position paper we outline some hardware and software features that would facilitate the implementation of a viable online performance monitoring infrastructure for continuous adaptation.

Cost-Effective, Thread-Local Event Monitoring: Ideally, performance monitoring hardware should be replicated in future processors, once for each context executing concurrently on the same physical package. If this is not possible, due to either cost or technical restrictions, the problem should be dealt with by the system software, transparently to the applications programmer. Some processors - Intel HT and IBM Power series processors are typical examples [4, 10] - introduce complex dependencies between the events to be measured, the event counters and the configuration registers to be used. However, even for these processors, it is possible to partition performance monitoring hardware in sets, so that each set is capable of measuring almost any event. In other words, it is generally possible for system software to provide the virtual notion of per thread performance monitoring hardware infrastructure, at the expense of limitations in the number of concurrently measurable events.

Overhead and Intrusiveness: An important issue, beyond guaranteeing the unhindered ability to use performance monitoring facilities in all modern architectures, is the minimization of the overhead for configuring the counters and collecting their values. Although our kernel schedulers have not been hampered by performance monitoring overhead due to their coarse time quanta, low overhead is necessary if counter values are to be sampled at high rates, in order to achieve a more accurate correlation of code segments or program phases with the monitored performance metrics. In particular, low-overhead monitoring is paramount if continuous optimization is to be applied to individual programs. The resolution of inter-thread conflicts for performance monitoring units at the hardware level would permit the use of unprotected, user-level instructions for counter configuration and sampling, thus significantly lowering the corresponding overheads. An alternative and more aggressive strategy with the same goal is to offer - once again at the hardware level - functionality for non-intrusive hardware monitoring. An example of such functionality is given by reconfigurable Liquid architectures [5]. Liquid architectures have specialized hardware for continuous streaming of `<PC,event>` pairs on an instruction by instruction basis, as instructions are executed by the processor. The event streams are fed to compilers and used to customize reconfigurable hardware modules, such as caches and prefetchers, to the characteristics of the stream. By using dedicated hardware for online monitoring, the architecture offloads most of the monitoring overhead from the processor and is capable of

extremely fine-grain and efficient adaptation to observed hardware metrics.

Standardization: Another fundamental impediment for the widespread exploitation of performance monitoring hardware is the lack of standarization, in terms of both the set of measurable events on each processor and the API exported to the programmer. As a result, code designed to use performance monitoring hardware is often not portable across different generations, or even different models of the same processor. Moreover, some processors do not support monitoring of basic performance metrics, such as read / write miss rates at all levels of the cache or, even worse, introduce inaccuracies of up to two orders of magnitude. Representative examples, on Intel P4 processors, are the events `IOQ_allocation` which is configured differently on different P4 models and `BSQ_cache_reference` which may, under certain circumstances, over- or under-count by a factor of two [4]. In addition to the noteworthy effort of standardizing the interfaces to the hardware counter configurations of different architectures, which is exemplified by PAPI [3], it is important to define formally a standard set of events that should be supported reliably by all architectures. According to our experience, it is the lack of reliable hardware support and functionality that impedes a continuous hardware monitoring infrastructure more often than the lack of a standardized interface.

Conveying Information on Contention: An additional obstacle in the use of hardware counters for system optimization is the absence of information about the contention between threads for shared resources. Queues, either virtual or physically implemented, are used as buffers at the front-end of shared resources, at all levels of modern system architectures. The average length of these queues is a valuable metric for the estimation of the expected latency to the shared resource. Intel P4 processors, for example, allow such an estimation for resources like the cache or the front-side bus, as the ratio of the cumulative number of outstanding accesses to a resource divided by the number of distinct accesses to the same resource. Similar data would be very useful for other internal queues, such as the microops queues between the instruction issue unit and the execution units. The estimation of latencies for access to shared resources is of critical importance, especially for HT, SMT or multicore designs which inherently increase the degree of sharing inside or at the border of the processor package.

In HT and SMT architectures it would be possible to accurately characterize contention for shared resources by threads executing on the same processor, at the cost of some additional tagging of microops or cache lines. For example, tagging microops according to the execution context that issued them would allow the detection of stalls due to interference with other threads on execution units, cache ports or memory read / write buffers. Tagging cache lines, on the other hand, according to the threads that have touched them, facilitates the estimation of the cache footprint of each thread - a valuable hint for scheduling policies targeting cache affinity - or the detection of cross-thread eviction in the form of conflict misses. In general, an accurate characterization of stalls or conflicts would help performance-driven scheduling policies to enforce optimal thread pairings, resulting to significant performance gains.

Characterizing Misses: The characterization of cache misses is another feature which is notably absent from current hardware performance monitoring infrastructures. To some extent this is reasonable as the algorithms for classifying cache misses on uniprocessors and multiprocessors can be quite complex. However, some information besides raw numbers can be enlightening for online optimization. For example, cross-thread eviction of cache lines can characterize the contention between threads in the cache and help the system identify easily cache thrashing. Regional counters of the per-thread population of cache lines in each cache

bank, or each page-size region in the cache, can be used to illustrate the affinity of threads for specific ranges in the cache. Such information can enable sophisticated online memory optimizations, most notably, dynamic page coloring and page remapping from the operating system. We foresee that online optimization of memory allocation and management will be equally critical to the optimization of scheduling decisions in operating systems designed for emerging multithreaded and multicore microprocessors.

Monitoring Memory Pressure on Specific Address Ranges: Monitoring per thread accesses to virtual address ranges is useful for cache indexing, since it permits the identification of 'hot' and 'cold' areas in the virtual address space. Moreover, it may be useful for associating objects with addresses at run-time and providing feedback to run-time optimizers or profile guided compilation. Monitoring of accesses to physical address ranges, on the other hand, can be exploited, in NUMA systems, to optimize data placement in memory for either locality or energy control. As NUMA architectures become more popular and NUMA systems appear even at low-end, low-cost configurations, such as AMD Opteron-based multiprocessors, the ability to monitor thread accesses to memory address ranges gains importance and such a provision is necessary in future processors. At first, it may seem inevitable to combine information for events both internal and external to the processor, however the required support can actually be consolidated in the processor. The processor has all the necessary information on both virtual memory accesses and their association with physical addresses.

We have successfully employed hardware counters to implement continuous optimization infrastructures for kernel schedulers. Our experience suggests that a number of drawbacks in the currently available infrastructures preclude the characterization and summarization of hardware events, as required by an online optimization system. In our view, a coordinated effort by both hardware designers and system software developers is needed to address these issues. Hardware designers should focus on the characterization of hardware events that are direct indicators of performance losses and their sources, such as inter-thread contention, as well as on reliability and portability issues. System software designers should keep working on integrating low-overhead and non-intrusive performance monitoring into transparent system modules that will eventually enable runtime monitoring and adaptation. We believe that current operating systems can be extended with a reasonable effort to employ continuous optimization in their fundamental resource management modules, including the CPU scheduler, memory management and networking modules. More efficient hardware support for non-intrusive hardware monitoring is needed to employ continuous adaptation in individual applications. We see both system-level and application-level adaptation via hardware monitoring as a promising path towards leveraging the computational power of future high-end systems.

Acknowledgements

This work is supported by an NSF ITR grant (ACI-0312980), an NSF CAREER award (CCF-0346867), an IST grant (No. 2001-33071), and the College of William and Mary. We would like to thank Rob McGregor and Theodore Papatheodorou for earlier contributions to this project.

References

[1] C. Antonopoulos, D. Nikolopoulos, and T. Papatheodorou. Scheduling Algorithms with Bus Bandwidth Considerations for SMPs. In *Proc. of the 33rd International Conference on Parallel Processling (ICPP'2003)*, pages 547–554, Kaohsiung, Taiwan, October 2003.

[2] C. Antonopoulos, D. Nikolopoulos, and T. Papatheodorou. Realistic Workload Scheduling Policies for Taming the Memory Bandwidth Bottleneck of SMPs. In *Proc. of the 2004 IEEE/ACM International Conference on High Performance Computing (HiPC'2004)*, pages 286–296, Bangalore, India, December 2004. LNCS Vol. 3296.

[3] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In *Proc. of Supercomputing'2000: High Performance Networking and Computing Conference (SC'2000)*, Dallas TX, November 2000.

[4] Intel Corporation. *Intel Architecture Software Developer's Manual. Volume 3: System Programming Guide*, 2004. http://developer.intel.com.

[5] P. Jones, S. Padmanabhan, D. Rymarz, J. Maschmeyer, D. Schuehler, J. Lockwood, and R. Cytron. Liquid Architecture. In *Proc. of the 2004 NSF NGS Workshop (held in conjunction with IPDPS'2004)*, Santa Fe, NM, April 2004.

[6] R. McGregor, C. Antonopoulos, and D. Nikolopoulos. Scheduling Algorithms for Effective Thread Pairing on Hybrid Multiprocessors. In *Proc. of the 19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS'2005)*, Denver, CO, April 2005.

[7] SGI. *Topics In Irix Programming, Chapter 4*. http://techpubs.sgi.com.

[8] A. Singhal and A. J. Goldberg. Architectural Support for Performance Tuning: A Case Study on the SPARCcenter 2000. *ACM SIGARCH Computer Architecture News, Proc. of the 21st Annual International Symposium on Computer Architecture (ISCA 94)*, 22(2), April 1994.

[9] L. Smolders. PowerPC Hardware Performance Monitoring. Technical report, AIX Performance, IBM Server Group, November 2001.

[10] L. Smolders. *System and Kernel Thread Performance Monitor API Reference Guide*. IBM, RS/6000 Division, 2001.