PACMAN: A PerformAnce Counters MANager for Intel Hyperthreaded Processors

Matthew Curtis-Maury, Dimitrios S. Nikolopoulos
Department of Computer Science
Virginia Tech
{mfcurt, dsn}@cs.vt.edu

Christos D. Antonopoulos
Department of Computer Science
College of William and Mary
cda@cs.wm.edu

1 Introduction

Performance monitoring counters (PMCs) are registers within a processor which can be programmed to count the occurrences of particular processor events, such as L2 cache misses, stall cycles, etc. Due to the insight that they provide into the execution of an application on a given architecture, hardware performance counters are seeing increasing popularity in both the research [1] and industrial communities [3].

Despite the pervasiveness of Intel Hyperthreaded processors [5], the support for collection of hardware performance counters on this architecture is limited. Tools designed to work on single-threaded processors fail to provide sufficient functionality when ported over. The difficulty stems from the sharing of the performance monitoring unit (PMU) between the two execution contexts on Hyperthreaded Pentium 4 processors. Perfctr [6], the standard interface to Pentium 4 performance counters for Linux, overcomes this problem by disallowing the use of the second execution context on each processor when collecting events in *per-thread* mode [1]. PAPI [1], being built on top of unaltered Perfctr, suffers from the same problems. Intel's VTune Performance Analyzer [4] provides thread-local event counter statistics offline, however it does not provide functionality for online and accurate event counter collection. It is important that applications be able to use all available contexts while still exploiting the full set of hardware event counting features at runtime.

In order to use performance counters for online adaptation of applications, it is necessary to be able to retrieve per-thread counter values from within the target application during its execution. Beyond this, it is also necessary to have fine-grain access to counter values to support monitoring of short-lived regions of code. In the following section, we discuss how we enabled such functionality in PACMAN (available at http://people.cs.vt.edu/~mfcurt).

[1]In *per-thread* mode event counters are recorded separately for each thread, whereas in *global* mode counters are recorded for all processors in the system with all executing threads included.

2 PACMAN Implementation

An Intel Hyperthreaded processor has 18 PMC registers that are shared between the two co-executing threads. Each register can record a single event. Should two threads on the same processor attempt to use the same register, only one thread's configuration would ultimately be used. PACMAN prevents overlapping of PMC register usage between co-executing threads by introducing a logical partitioning of the registers when events are recorded in per-thread collection mode. Within each processor, half of the PMCs are provisioned to each execution context. Intel has divided the registers into four sets and any given event can only be recorded within its designated set. We create our partition such that each set of registers is divided evenly between the two threads. There are also configuration registers associated with the PMCs which we have partitioned similarly. During performance counter initialization each thread is configured to use the partition for the execution context on which it is currently executing. To prevent a thread from migrating away from the execution context for which its PMCs were set up, PACMAN appropriately binds threads to execution contexts.

PACMAN uses the interface provided by Perfctr for low-level access to the performance monitoring counters, after removal of the internal checks from Perfctr that enforce usage and monitoring of only the first execution context on each processor. In addition to the extended support for Hyperthreaded processors, PACMAN retains the full functionality already present in

Perfctr. This includes the use of performance counters with only a single thread per processor using all 18 available PMCs, as well as global mode collection.

Although global collection does allow events to be collected on both execution contexts, even in Perfctr, there are two shortcomings of this approach for online use. First, event counts summarize collective performance of all executing applications, not just the one to be monitored. Second, the granularity of monitored regions must be very coarse (on the order of hundreds of milliseconds at least) since results are stored in the operating system and are only periodically updated between consecutive time quanta. These were motivating factors in the development of PACMAN.

The process of initializing Pentium 4 performance counters can be cumbersome. One hurdle to using performance monitoring counters is the creation of the bitmasks written to specific registers as part of the configuration process. Although Perfctr does abstract away the manual process of loading the registers, it must be given the exact contents for each desired register. These values specify what events to record, but they also provide additional constraints on recording, and creating the desired bitmasks can be a very complicated process. Further, for any event to be recorded, a configuration register must be specified and correctly initialized, however, only certain configuration registers are legal for a given event. PACMAN simplifies the initialization process by having predefined values for any desired event which allow the user to specify an event to be recorded by an intuitive name, such as STALL_CYCLES, and handles the low-level details internally. Another difficulty is that, due to sharing of the PMU, a given PMC register cannot be allocated to both co-executing threads. However, since registers are allocated in PACMAN according to the partitioning scheme described above, this problem is overcome as well. In these ways, PACMAN greatly reduces the difficulty of configuring PMC registers.

3 Example Use of PACMAN

In [2], we describe an online technique to use performance counters collected at runtime to predict the performance of recurring execution phases of parallel applications, were they to be run on a different number of processors and/or threads on an SMT-based SMP. Runtime prediction is essential for fast performance and power adaptation of multithreaded codes, since it overcomes the runtime overhead of direct search approaches.

In our scheme, we collect sets of counters using PACMAN during the first two executions of each phase of a parallel program with threads executing on all execution contexts of all processors. We calculate coefficients offline using multiple linear regression with input from a training set and then apply these to each event count to predict the IPC when the same phase is executed with different configurations of processors and threads per processor. Our results show that the average IPC prediction accuracy is almost 90%.

Once IPC predictions for different execution scenarios have been made, it is possible to adapt the number of threads and processors used in future executions of each phase according to some optimization metric, such as execution time, energy, or a combination of the two. This runtime adaptation scheme results in large gains in both execution time and energy consumption over a range of parallel benchmarks on multi-Hyperthreaded SMPs. Such adaptation would not be feasible without PACMAN, which provides runtime access to performance counters at a fine granularity on a thread by thread basis and the exclusion of events incurred by external noise from other applications or system activity.

Acknowledgments

This research is supported by the National Science Foundation (Grants CCF-0346867 and ACI-0312980), the U.S. Department of Energy (Grant DE-FG02-05ER2568) and an equipment grant from the College of William and Mary.

References

[1] 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 (SC'00)*, Dallas, TX, November 2000.

[2] M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos. Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction. In *Proc. of the 20th International Conference on Supercomputing*, Queensland, Australia, June 2006.

[3] S. Eranian. The Perfmon2 Interface Specification. Technical Report HPL-2004-200R1, HP Labs, February 2005.

[4] Intel Inc. Intel VTune Performance Analyser. http://www.intel.com/software/products/vtune, 2003.

[5] D. Koufaty and D. Marr. Hyperthreading Technology in the Netburst Microarchitecture. *IEEE Micro*, 23(2):56–65, March 2003.

[6] M. Pettersson. A Linux/x86 Performance Counters Driver. http://user.it.uu.se/~mikpe/linux/perfctr/.