Bandwidth Measurement using Performance Counters for Predictable Multicore Software*

Rafia Inam[†], Mikael Sjödin[†], Marcus Jägemar[#†]
[†]Mälardalen Real-Time Research Centre, Mälardalen University, Sweden
[#]Ericsson AB, Kista, Sweden
Email: {rafia.inam, mikael.sjodin, marcus.jagemar}@mdh.se

***Abstract*—Memory contention is one of the largest sources of inter-core interference in statically partitioned multicore systems, and the contention reduces the overall performance of applications and causes unpredictable execution-times. A first step in achieving predictable execution is to accurately measure the amount of consumed memory bandwidth for each application. Such measurements can be used to track down bottlenecks, provide better partitioning among cores, and ultimately be used to arbitrate and police access to the memory bus.**

We propose to use hardware performance counters to continuously track the memory-bandwidth consumed by different applications executing in parallel. In this paper we describe ongoing efforts exploring suitable performance counters on core-level and on system-on-chip level for the 8-core Freescale P4080 processor. The aim is to accurately and efficiently track consumed memory bandwidth per application; with the final goal to use these measurements to improve predictability of multicore real-time software.

***Keywords*-performance counters, performance monitoring, memory bandwidth, OSE.**

I. Introduction

Most modern processors host a range of hardware performance counters which can be used to infer the amount of consumed computing-resources. Different processor architectures provide different sets of performance counters which make the determining consumption of various resources more or less easy and accurate. In this paper we discuss on-going work on using the hardware performance counters to measure consumed memory-bandwidth on multicore architectures. The ultimate goal of our work is to use these measurements to achieve predictable execution of real-time software on multicore architectures.

Tracking consumed memory-bandwidth in multicores has several potential uses. Memory contention is one of the largest sources of inter-core interference in statically partitioned multicore systems. Thus, evaluating the consumed bandwidth by each core can be a key to understand and resolve performance bottlenecks in multicore applications and can reduce the sources of indeterminism in the execution-times. In future, we envision that the tracking of the consumed memory-bandwidth could also be used to make intelligent scheduling decisions to prevent contention and to spread memory accesses over time to even out the load on the memory bus. Hence, we believe that an efficient and accurate tracking of the consumed memory-bandwidth will be a key technology to realize predictable real-time systems on multicores.

Contemporary scheduling of real-time tasks on multicore architectures is inherently unpredictable, and activities in one core can have negative impact on performance in unrelated parts of the system (i.e. on other cores). A major source of such unpredictable negative impact is the contention for shared physical memory. In commercially existing hardware, there are currently no mechanisms that allow a core to protect itself from negative impact if another core starts stealing its memory bandwidth. Hence, we need to develop software technologies to track, and eventually police, the consumed memory bandwith in order to achieve predictable multicore software.

Our work target performance critical real-time systems, which is a class of hard and soft real-time systems where it is imperative to achieve both high performance and predictable throughput. For hard real-time systems, this situation typically occurs when the designer wants to maximize the utilization of hardware resource (e.g., using the cheapest possible hardware, or fitting the maximum number of functions into an already existing hardware). For soft real-time systems, the situation occurs when the system's value is determined both by its delivered quality of service and its capacity to handle large volumes of computations.

The hardware performance counters are registers frequently used for off-line and on-line performance analysis without slowing down the system. Since they provide non-intrusive performance monitoring, they are good candidates to be used for monitoring in performance critical systems. The counters are used to track certain low-level operations or events within the processor accurately and with minimal overhead. The performance counters can be used when evaluating performance of a computer system. For example, Eranian describes the use of performance counters to measure four interesting memory-related metrics: measuring cache misses, measuring memory bandwidth, measuring latency and access locality for the x86 platform [1].

We propose to use performance counters to measure memory bandwidth for our target hardware platform the Freescale P4080 processor [2] which hosts eight e500mc cores. Ongoing work is to implement the performance counters for an industrial real-time multicore operating-system kernel, OSE

* This work is supported by the Knowledge Foundation (KK-Stiftelsen), via the research programme PPMSched.

[3], developed by ENEA. The characteristics monitor, denoted *charmon*, uses the performance monitor facility located inside the P4080 processor. We plan to implement the performance counters into the charmon, for online monitoring of the consumed memory bandwidth.

Paper Outline: Section II presents the background on performance monitors and memory bandwidth measurement. Hardware and software technologies used in our work are presented in section III. Section IV describes on-going work on memory bandwidth measurement approach. Finally, section V concludes the paper with a description of future work.

II. Background

In this section we describe performance monitoring software and other available tools to measure application cache and memory bandwidth usage.

A. Performance monitors

Usually a *performance monitor* is used to collect low-level information about events occurring in the processor during software execution (for example, the number of elapsed cycles, the number of cache misses, instructions executed, etc.). Performance monitor counters are currently implemented for many modern processor architectures such as x86, ARM and PowerPC. A brief information on different available performance monitor software is given below:

PerfCtr adds support to Linux kernel (2.6.x) for using hardware performance-monitoring counters on x86, x86-64, PowerPC, and certain ARM processors [4], [5]. Another light-weight performance monitor is perfctr-xen that provides access to hardware performance counters in virtualized environments using the Xen hypervisor [6].

PerfMon is developed by the HP Research [7] and is very similar to PerfCtr. Originally it was designed specifically for Itanium (IA-64) under Linux, and now it supports x86 and more architectures. Development of perfmon has been moved to an open community under the name of perfmon2. The perfmon2 is a generic kernel interface to access the performance counters on Linux with the help of a user level library [8] that offers kernel-level sampling buffers and event set multiplexing.

Performance Application Programming Interface (PAPI) is developed by the Innovative Computing Laboratory [9]. It provides a standard set of performance monitoring events and a standard API to access hardware counters in a portable way. It uses perfmon and perfctr drivers on Linux. Performance Counters for Linux (PCL) is another preferred performance monitoring framework for Linux that allows monitoring of events per task and per CPU counters [4]. Another tool when investigating performance counters is cachegrind/valgrind. In general terms the cachegrind Linux tool performs in a similar way as the charmon.

All the above mentioned tools operate on Linux. Since we are using a different operating system than Linux, normal Linux-based tools are not usable out-of-the-box and needs to be ported. Also some tools does not support the PowerPC platform we are using which narrows the list of available tool. We are using charmon because it is already implemented for the P4080 processor and it also provides a continuous system monitoring functionality and supports clients to be linked to it performing various tasks depending on system behavior.

One additional example of system monitoring is presented in [10]. In their approach they implement a low intrusive (1%-3%) sample based mechanism to gather system wide information. The sampling is implemented by means of periodically executing sampling interrupts generated by performance counters. In our work this is done by a periodically executing process gathering performance counters in a ring buffer.

B. Measuring application bandwidth usage

In cache pirating described in [11] and bandwidth bandit [12] stealing cache and memory bandwidth causes higher system load. They have implemented tools that predictably steal cache and memory bandwidth to predict application resource usage. We could use these techniques to individually measure the desired memory usage for a set of applications that should be mapped to different partitions of a system.

III. Hardware and software technologies

Here we give an overview of our target hardware and software platforms, and describe the performance monitor software we use to measure the memory bandwidth on these platforms.

A. Hardware used

We target the Freescale P4080 processor [2] hardware platform which hosts eight identical e500mc cores that have uniform access to the main memory. The processor has a set of execution-resources that are private to each core (e.g. CPU, L1 caches, and L2 caches) and a set of execution-resources that are shared among all cores (e.g. L3 cache, memory bus and main memory), as exemplified in Figure 1. In this initial paper we focus on the shared memory and assume that all accesses to the shared resources goes through the same bus, and that the bus serves one request at the time.[1]

B. OSE operating system

OSE is a deterministic, distributed real-time, fully preemptive OS, optimized to provide high performance with bounded response times and is used in many soft and hard real-time embedded systems (e.g. telecommunications, automobiles, and medical devices) [13]. The minimal standard configuration for OSE is 350 kbytes including the kernel services and core basic services layers (smaller configurations are possible by excluding portions of the core basic services layer).

In addition to basic services layer (that provides preemptive priority-based scheduling, asynchronous message passing for inter-process communication and synchronization, and error

[1]These assumptions are consistent with typical architectures used in industry today. However, architectures with e.g. non-uniform memory access and interleaved access to the shared memory-bus are predicted for coming generations of hardware.

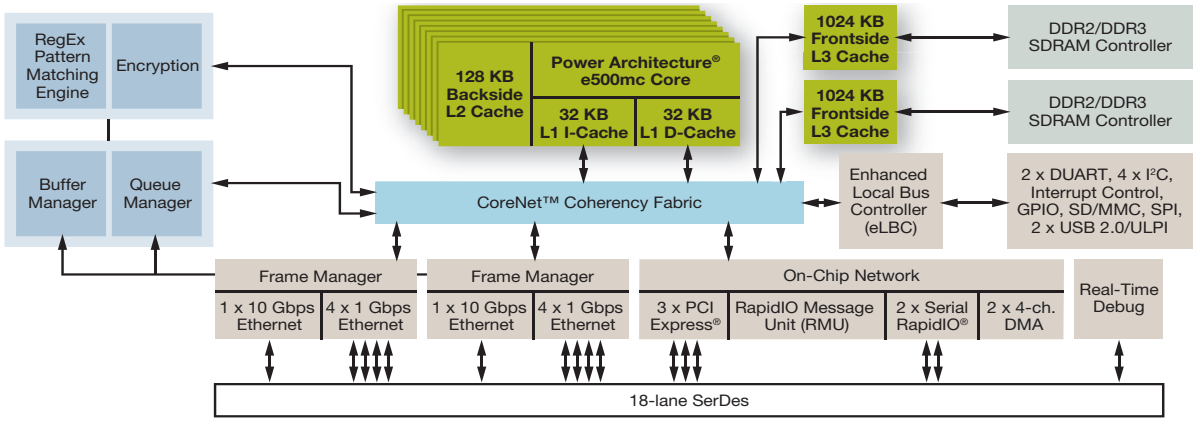

Fig. 1. P4080 multicore processor architecture model [2]

handling), the OSE microkernel also provides memory management facilities that enables the operating system to take advantage of MMU hardware for memory protection [13]. On-top of the microkernel, are core basic and core extension layers. The core basic layer provides file-system functionality, networking services, device driver management, debug, and C/C++ runtime support while the core extension offers other optional services.

C. The characteristics monitor

The charmon uses the performance monitoring facility located inside the P4080 processor. It allows a client application to monitor events on CPU level regarding low-level functionality such as cache hits/misses, branch statistics, TLB hits/misses and other hardware related events that can be used to calculate key performance indexes *KPIs*, used for performance and/or behavior evaluation. A KPI can be a function of a number of events such as L1 I-cache, L1 D-cache, L2 I-cache, L2 D-cache, etc.

Implemented within the platform, the charmon is a continuously running performance monitoring tool. It gathers information about HW-usage for the complete system by periodically sampling performance monitor counters (PMCs) and storing the results in a local database. PMCs are described in general in [2] and with more details in Table 9-47 in e500mc Core Reference Manual [14] . The charmon can simultaneously measure PMC events for all cores and group them on a per core basis for viewing together with calculated KPIs. The current implementation of the charmon supports coarse-grain sampling on a per-core basis. This means that current metrics can not be determined per executing process, but on a system level. One way to remedy this is to implement counter storage when processes are being swapped in and out of the ready queue, but with an increased probe effect.

Selecting a proper sample period for each counter set is difficult and too short period increases the probe effect and too long gives course grain samples. Currently we have selected 1 second as a tradeoff. In our platform there are 13 counter sets which gives a sample periodicity of 13 seconds.

The probe effect has been neglected in our investigation since the PMCs are located inside the CPU with low or no cost and the DB-storage and PMC reprogramming occurs infrequently as described earlier. The characteristics monitor is easily extendable with additional counter sets depending on what to monitor. Current sets include Cycles per Instruction (CPI), branch statistics, TLB information, L1-Cache, L2-Cache and L3 cache etc. Furthermore, using PMC gives the the opportunity to measure non-instrumented code further reducing the intrusiveness of the monitor.

IV. Memory Bandwidth Measurement

It is our intention to study the P4080 architecture and the associated memory-controller on our target system to determine which events are needed to be counted in order to accurately determine the amount of consumed bus-bandwidth.

A. Determining the consumed memory bandwidth

In many cases, a continuous determination and tracking of the consumed memory bandwidth is very difficult without using a dedicated external hardware that monitors the memory-bus. Since we target the use of standard hardware, we describe in this section how we can estimate the bandwidth using the performance counters.

For any given CPU architecture, the accuracy in such a prediction will depend on issues like available events to count, number of available counters (often a small set of counters are available to be programmed to count various events), and the characteristics of the memory-bus. For soft real-time systems, it may be enough to use rough estimates of correlation between some counted events and actual bandwidth usage to get an acceptable estimate of the consumed bandwidth. However, for hard real-time systems, the estimate needs to be safe (i.e. we are not allowed to underestimate the consumed bandwidth).

In our current project we target the Freescale P4080 processor which hosts eight e500mc cores [14]. The e500mc has 128 countable events which we can use to infer how much memory-bandwidth has been consumed. Unfortunately there is no event which directly tells us how many memory-bus

cycles that have been used. Instead a set of events, like "Write-through stores translated", "Data L1 cache reloads", and "L2 linefill buffer", can be monitored to try to determine the number, and size, of memory accesses. Furthermore, the P4080 has a system-on-chip (SoC) unit which hosts a set of additional performance counters. Here we find,e.g., counters related to L3-cache. Unfortunately (for our purpose) there does not exist a single bus access event like the BUS_TRANS_BURST in Inte Core 2 Duo architectures [1]. Instead the P4080 SoC has a more sophisticated interface that allow larger degree of configurability, and thereby a larger set of events to count.

B. Online monitoring

To provide continuous online monitoring of the consumed bandwidth, we cannot resort to simply resetting counters and observing them at the end of a test run (as is done in many performance measuring tools). Instead we need to continuously monitor counters and store their values.

Some processors allow alarms or interrupts to be associated with the performance counters. If the bandwidth can be straightforwardly inferred from a single counter this could be used to avoid polling the counters and thus lowering the overhead of the monitoring. However, if the processor does not allow us to use such alarms or the consumed bandwidth needs to be estimated as a function of several performance counters then we need the periodic polling of the counters.

Furthermore, in the extension of this work we will implement schedulers that police the consumed bandwidth and thus the accurate and non-intrusive estimates of the bandwidth consumptions becomes even more important. For policing purposes using alarms could give the most accurate accounting of the consumed bandwidth. In the e500mc, we can set wrap-around alarms on counters. Thus, if we would like to allow an application to generate at most x events before being policed we could initialize the 64-bit event counter to $2^{64} - x$ before running the application. For L3-cache events and main-memory accesses where we use the SoC counters it is yet unclear to us whether we can set the alarms or not, and we are currently investigating this.

C. Relevant hardware resources

From the Figure 1, we identify that the key resource to monitor the consumed bandwidth is the CoreNet Coherency Fabric. Currently charmon is calculating different kinds of KPIs by measuring a number of events such as L1 I-cache, L1 D-cache, L2 I-cache, L2 D-cache, and L3 cache. To compute accurate memory bandwidth consumption, we require some more KPIs and need to measure more events for the L3 cache.

D. Extensions to the charmon

As outlined above, currently the charmon does not monitor all the events needed to accurately track L3-events and main-memory accesses. This will be extended in ongoing work. However, the current architecture of charmon is designed to performance coarse-grained monitoring on the system level. For our purposes, we need fine-grained monitoring on the application level.

For this reason, the counter-management (programming of which event to count, initialization of counter registers and reading of registers) needs to be managed in conjunction with the task-switches. In Enea OSE we can add a hook to the task-switch that allow us to do the proper counter-management at the application level.

V. Conclusions and Future Work

The measurement of the consumed memory bandwidth for all applications running in parallel on multicore platform could be used to not only reduce the performance bottlenecks but to also achieve predictable execution-times. The tracking of the consumed memory-bandwidth could also be used to make intelligent scheduling decisions to prevent contention and to spread memory accesses over time to even out the load on the memory bus. We have proposed to use the hardware performance counters at both core- and system-on-chip levels for the 8-core Freescale P4080 processor to monitor the consumed memory bandwidth by the running applications.

Currently, we are investigating different performance counters and all the related events needed to accurately track L3-cache events and main-memory accesses. The next step is to implement them into the characteristics monitor, charmon, for online monitoring. Once these implementation efforts are complete, we will have accurate online measurements of the consumed memory bandwidth for all applications. The next step will then be to implement schedulers that can be used to arbitrate and police the memory bus accesses by different applications.

References

[1] S. Eranian. What can performance counters do for memory subsystem analysis? In *ACM SIGPLAN Workshop on Memory Systems Performance and Correctness (MSPC'08)*, pages 26–30. ACM, 2008.

[2] P4 Series, P4080 multicore processor. cache.freescale.com/files/-netcomm/doc/fact_sheet/QorIQ_P4080.pdf.

[3] Enea AB, Sweden. Data Sheet ENEA OSE 5.5. http://www.enea.com/Documents/Resources/Datasheets/Enea%20OSE5%20021053.pdf.

[4] Thomas Gleixner. Performance counters for Linux. http://lwn.net/Articles/310176/.

[5] M. Pettersson. Perfctr library, 2011. http://user.it.uu.se/ mikpe/linux/perfctr/.

[6] Ruslan Nikolaev and Godmar Back. Perfctr-xen: a framework for performance counter virtualization. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 15–26, New York, NY, USA, 2011. ACM.

[7] S. Eranian. Perfmon2: A flexible performance monitoring interface for linux. In *Ottawa Linux Symposium*, pages 269–288, 2006.

[8] S. Eranian. The perfmon2 project. http://perfmon2.sourceforge.net.

[9] S. Browne, C. Deane, G. Ho, and P. Mucci. PAPI: A portable interface to hardware performance counters. In *Department of Defense HPCMP Users Group Conference*, 1999.

[10] J. Anderson, L. Berc, and J. Dean. Continuous profiling: where have all the cycles gone? In *ACM SIGOPS*, pages 357–390, Nov. 1997.

[11] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Cache pirating: Measuring the curse of the shared cache. In *Proc. of ICPP*, 2011.

[12] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Design and evaluation of the bandwidth bandit. In *IEEE*, 2012.

[13] Enea AB, Sweden. OSE Kernel User's Guide, 1998.

[14] e500mc Core Reference Manual, rev 1, 2012. cache.freescale.com/files/-32bit/doc/ref_manual/E500MCRM.pdf.