Memory Access
The Third Dimension of Scheduling

Frank Bellosa

January 1997 TR-I4-1-97

Technical Report

Institut für
Mathematische Maschinen
und Datenverarbeitung
der
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Lehrstuhl für Informatik IV
(Betriebssysteme)

Memory Access
The Third Dimension of Scheduling

Frank Bellosa

bellosa@informatik.uni-erlangen.de

IMMD IV - Department of Computer Science
University of Erlangen-Nürnberg

Abstract

Up to now, two internal events influence scheduling decisions of contemporary operating systems: timer events and I/O-related interrupts. Timing information supports preemption and priority adjustment. Knowledge about issued or completed I/O operations helps to wake-up sleeping processes and to boost their priority. Preferring deblocked processes at the end of I/O operations improves the interactive performance and saves buffer space because available data is consumed short after its availability.

With the upcoming of processors clocked with hundreds of megahertz, the processor speed exceeds the speed of affordable memory by factors. If it is accepted to influence execution priorities by slow I/O events, why should scheduling neglect events related to other slow devices like main memory and memory data paths?

Our novel approach to scheduling is based on knowledge derived from counters in the memory subsystem. We demonstrate that the usage of information related to cache- and main-memory access opens new dimensions in real-time and time-slice scheduling of shared-memory architectures. Advanced cache-affinity scheduling, memory-bandwidth guarantees for real-time jobs, and new processor assignment strategies are three basic concepts of a new scheduling philosophy focusing more on memory access than on CPU cycles as the performance determinant factor.

1 Introduction

Operating systems need information to control operation. One of the major control operations is dynamic scheduling, the task to find a suboptimal schedule for runnable jobs. Scheduling decisions are based on internal runtime information and data coming from process-control calls. The more exhaustive the information is, the better the schedule can be computed.

Two types of events build the internal runtime information for scheduling in contemporary kernels: timing information in the form of timer interrupts or clock counters and I/O related interrupts. Timer interrupts make time-sharing scheduling possible whereas clock information is the basis of processor affinity-scheduling and real-time frame- or deadline-scheduling [SGI 96]. Interaction with slow I/O devices leads to blocking of processes in the kernel. Deblocked processes due to finished I/O operations normally are preferred executed to improve the interactive performance and to save buffer resources. Furthermore, the importance of specific types of I/O operations influences the priority boost for deblocked processes. Interaction with slow devices is therefore an indication for priority adjustment managed by the scheduler.

The advances in memory technology concerning performance could not keep pace with those in processor technology. Processors clocked with hundreds of megahertz exceed the speed of affordable memory by factors. Consequently, memory access operations, the reading and writing of data to main memory, has to be considered analogous to I/O operations. High speed processor caches are the pendant to I/O buffers. The main memory access of high priority processes via interconnection networks is equivalent to high priority I/O operations. If it is accepted to influence execution priorities by slow I/O events, why should scheduling neglect events related to other slow devices like main memory and memory data paths?
We introduce a third dimension of scheduling parameters: memory access patterns. In the following, we describe ways to acquire memory-access information and how we can use this information to improve scheduling.

2 Where to Find Memory Access Information

Memory access information is the fullness of countable events related to memory operations. Examples are load/store-operations, cache misses, cache invalidations, issued main memory requests or processor stall cycles due to memory requests. A countable event is only useful, if it can be assigned to the object, responsible for the event. In the next subsections, we show the locations in shared-memory architectures, where memory access information can be gathered by event counters, such that this information is usable in memory-conscious scheduling.

2.1 The generalized case of a shared-memory multiprocessor

There are four potential regions in a shared memory architecture suitable for the placement of event counters:

- The processor:
  Counters inside the CPU can register issued load/store operations. If the first level caches reside inside the CPU, the effects of cache-access operations (e.g., data- and instruction-cache hits/misses, CPU stall cycles due to cache misses,...) can be registered inside the processor and assigned to the process currently running on this CPU.
- The cache controller for external caches:
  Cache misses, cache invalidations and stall cycles due to cache misses can be registered inside the cache controller. If the external cache is dedicated to a single CPU, the information can be assigned to the currently running process. Architectures with shared caches do not offer this feature.
- The interface between the processor-cache module and the interconnection network:
  Cache misses initiated by a CPU imply main memory access but not vice versa. Memory regions marked as uncachable (e.g., DMA buffers) become not encached. So, they cause no cache events when being accessed. Furthermore, some CPUs (e.g., Pentium MMX [Intel 96], UltraSPARC [SUN 95]) offer special load/store operations that bypass the cache to prevent cache corruption by operations working on data streams (e.g., MPEG operations). These load store operations initiate main memory operations without interfering the cache content.
  Counters inside the interconnect interface can register all main-memory related transfers. The information can be assigned to the currently running process only if the interconnect interface is not shared by multiple CPUs.
- The network interface of memory banks:
  Each memory bank can only service a limited number of requests in a certain time frame. If the number of serviced requests can be registered in counters, the load of the memory subsystem is known. Now the memory load can be influenced from the operating system by scheduling processes with a specific memory-access characteristic.

System designers have always observed the behavior of computer systems with hardware monitors to detect bottlenecks and to check the performance gain of architectural improvements. Because of increased clock speed reaching up to several hundred megahertz, external hardware, monitors became more and more expensive due to electrical reasons. Internal event monitors, which count events with full clock rate, are an affordable alternative to external monitoring hardware. Thus, monitoring hardware is embedded in advanced computer architectures to count events occurring inside the processor, the memory system or the I/O-subsystem.
If we look into todays multiprocessor systems we can see counters in some of the regions mentioned above. On the one hand, we can make the best of the available counters and use them in multiprocessor scheduling; on the other hand, we can influence system designers to enrich computer architectures with counters that are easy to access and fast to read.

2.2 SUN Enterprise X000 Server Architecture

The SUN Enterprise X000 Server Architecture [SUN 96] is the hardware platform for our prototype of a scheduler using memory access information. Each CPU includes two counters, which can register two events out of a set of more than 22 event-types. The MMU and the external cache controller is on-chip, so that external cache events like cache hits or cache references belong to the countable events as well as issued instructions and clock ticks (see figure 2.1). The bus-connector is implemented as a switch (**U**ltra **P**ort **A**rchitecture switch), connecting a pair of CPUs and two memory banks with the bus system (GigaPlane). Two configurable counters can register events like issued addresses, issued data packets or memory stall cycles of each memory bank. As the bus-connector is shared by two CPUs, the counter values can not be assigned to a specific CPU and therefore to a specific process. But the information about stall cycles gives valuable information about the current memory-load of the system.
Unfortunately, only two events can be registered on the CPU/cache- and the bus-connector-level although there are much more interesting events inside the hardware from the focus of scheduling. If memory access information is a proved mean to increase system performance and functionality, we strongly recommend to improve the number of counters.

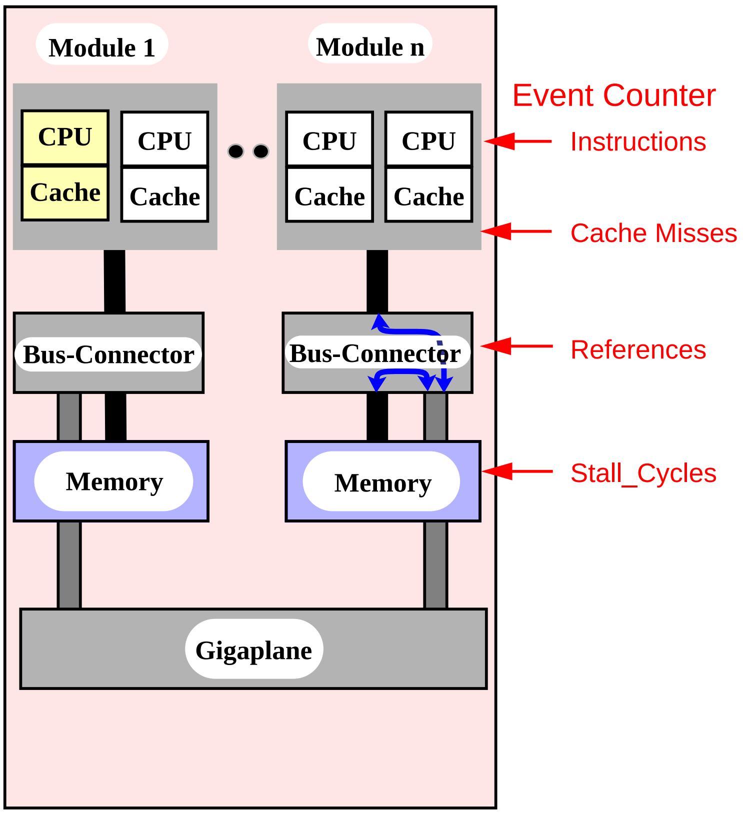

Abb. 2.1 Ultra Enterprise X000 Architecture

3 How to Use Memory Access Information?

3.1 Concept

There are some scheduling demands which can not be serviced by operating systems that neglect memory access information:

- Real-time scheduling classes claim to guarantee a determined CPU performance to fulfill a task with a guaranteed amount of locked (unswapable) memory. But, how can CPU performance be determined and guaranteed on a multiprocessor system, if processes running on other CPUs consume memory bandwidth and therefore slow down a processor performing a real-time task (e.g., real-time image video processing)?
- The affinity of a process to a processor can only be described insufficiently by timing information. Warm-Up and Cool-Down curves [BaBi 95] do not take into consideration the memory reference behavior of specific processes. Comparisons between a user-level scheduler, using timing information, and another, using additional information about cache misses, clearly demonstrated the superiority of the latter. Measurements on a Convex SPP1000/XA48 NUMA architecture have shown, that the performance of multithreaded adaptive applications in the field of scientific computing could be doubled by memory conscious schedulers using cache miss information [Bell 96]. This is an indication, that complex scheduling strategies inside the kernel could benefit form memory access information as well.

Scheduling Class	Application Example	Characterization	CPU Affinity	Time -Slice
Real-Time with Bandwidth Guarantee	Video Processing (ShowMe)	High Main Memory Demands Bandwidth Reservation	No	-----------
Interactive Scheduler	Shell (tcsh)	Frequently Blocking User Interaction via Terminal	No	Short
Affinity Scheduler	Image Transformation (xv)	Working Set fits into Cache Lots of Cache References Few Cache Misses	Yes	Long
Low Interference Scheduler	Ghostscript Interpreter (gs)	Minimal Working Set Few Cache Misses Few TLB Misses at Restart	No	Short
High Interference Scheduler	Adaptive Numerical Application (ug)	Large/No Working Set Lots of TLB Misses Numerous Memory Accesses	No	Long

Tab. 3.1 Scheduling Classes requiring memory access information

- Processes without reference-locality push reusable data out of the cache, even if this data belongs to a thread with high cache-affinity which is blocked just for a short time period. In this situation we have to find a trade-off between utilization and throughput. Sometimes it is better to keep the CPU idle for a short moment, than to corrupt the cache content that otherwise might be reused in the next future.

Our approach to memory-access-information usage are scheduling classes supporting strategies which are tailored to the demands of processes with a specific memory-access characteristic. The characterization of all processes has to run periodically (e.g., initiated once per second by a timer interrupt). Basis of the characterization are sampling data from memory access counters, assigned to threads and stored in thread control blocks. Once the process is assigned to a scheduling class, it will run under the class specific scheduler for the next time interval.
As a first approach, we propose a set of classes listed in table 3.1.

3.2 Implementation

We started implementing memory-conscious scheduling in December 1996 using the Solaris 2.5.1 source tree. The hardware platform is a SUN Enterprise 3000 server.

- The first step was to enrich the kernel-thread context by counter information. We store the number of external cache references, the number of external cache hits and clock ticks for precise timing. This information has to be saved and restored during context switches.
  In the system boot procedure, we measure the relationship between the number of issued data packets on the system bus and the number of memory stall cycles. This relationship depends on the hardware configuration concerning processors and memory banks. Later, during normal operation, this information helps to detect shortfalls in memory bandwidth.
- In the next step, we build scheduling classes for different types of threads (see table 3.1):
  - Real-time Class with Bandwidth Guarantees:
    This scheduler class guarantees a memory bandwidth, that can be requested by I/O controls via the process file system. If the requested bandwidth is granted, the number of issued data packets will be compared to the guaranteed amount of data packets in a configurable time frame. Threads without guarantees and those exceeding their limit will be throttled by the system scheduler if a shortfall in memory bandwidth is detected. In the worst case, some CPUs stay idle, to save bandwidth. To avoid a starvation of system processes, the total amount of guaranteed bandwidth is smaller than the maximum memory-bandwidth of the hardware.
  - Interactive Scheduling Class:
    Interactive threads with a high number of blocking system calls are assigned to a conventional interactive scheduler.
  - Affinity Scheduling Class:
    Threads, showing high cache affinity by a high number of cache references and a low number of cache misses, receive a very long time-slice and are bound to a processor, so that their working set, that is stored in the cache, will be frequently reused.
  - Low Interference Scheduling Class:
    Processes with a minimal working set, performing low priority jobs, can be run on every CPU without pushing a lot of data out of the cache whenever threads with high cache-affinity block for a short time

  - High Interference Scheduling Class:
    Threads with a large working set, exceeding the cache size, and those, showing no measurable working set, will be scheduled rarely, but with a long time-slice. This helps to reduce the "cache-destructive" effects of this type of threads.

- An other project currently under development is a measurement interface to observe the positive effects of our memory conscious scheduler. Memory access information stored in thread specific memory regions can be mapped from a process doing monitoring tasks. The applications under investigation belong to the field of webservers (Apache), compilers (gcc), database systems (Ingres), interpreters (Ghostscript, Perl and Tcl/Tk), image processing (xv) and video conferencing tools (ShowMe).

We expect to receive measurements, which we can publish with good conscience, in April 1997.

4 Conclusion

Even in the earliest time-sharing and multiprocessing operating systems, timing information and I/O interrupt has been the basis of scheduling. A new generation of highly clocked processors demands new scheduling techniques taking the memory-access behavior of processes into account. We have motivated the use of memory-access information and demonstrated the possible locations in a shared-memory architecture, where this information can be gathered by event counters. The proposed approaches in scheduling mark only the beginning of a novel view in scheduling design. We expect that our advances will motivate further development, both in the architectural research but also in many other areas of software design where information about memory access may help to improve performance.

We have to keep in mind, that computation is more than modifying data. It comprises data modification as well as data movement. Consequently the memory system is a critical component of any high-performance computer system. Scheduling should now start to respect this development.

References

[BaBi 95] J. Barton and N.Bitar, "A scalable multi-discipline, multi-processor scheduler framework for IRIX", Proc. of IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing, April. 1995, LNCS Vol. 949, Springer, New York

[Bell 96] F. Bellosa, "The Performance Implications of Locality Information Usage in Shared-Memory Multiprocessors", Journal of Parallel and Distributed Computing, Special Issue on Multithreading for Multiprocessors, Vol. 37 No. 1, Aug. 96

[Intel 96] "MMX Technology Developer's Guide", Intel Inc. 1996

[SGI 96] "Cellular IRIX 6.4 Technical Report", White Paper, SiliconGraphics Inc., 1996

[SUN 95] "UltraSPARC Programmer Reference Manual, Revision 1.0", SUN Microsystems Inc., 1995

[SUN 96] "Ultra Enterprise X000 Server Family: Architecture and Implementation", Technical White Paper, Sun Microsystems Inc., 1995