This is a summary of the original paper, entitled "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors" which appears in MICRO 2007.

Stall-Time Fair Memory Access Scheduling (STFM):
Enabling Fair and High-Throughput Sharing of Chip Multiprocessor DRAM Systems*

Onur Mutlu Thomas Moscibroda
Microsoft Research
{onur,moscitho}@microsoft.com

ABSTRACT

In a chip multiprocessor (CMP) system, where multiple on-chip cores share a common memory interface, simultaneous memory requests from different threads can interfere with each other. Unfortunately, conventional memory scheduling techniques only try to optimize for overall data throughput and do not account for this inter-thread interference. Therefore, different threads running concurrently on the same chip can experience extremely different memory system performance: one thread can experience a severe slowdown or starvation while another is unfairly prioritized by the memory scheduler.

Our MICRO-40 paper proposes a new memory access scheduler, called the *Stall-Time Fair Memory scheduler (STFM)*, that provides *performance fairness* to threads sharing the DRAM memory system. The key idea of the proposed scheduler is to *equalize the DRAM-related slowdown experienced by each thread due to interference from other threads*, without hurting overall system performance. To do so, STFM estimates at run-time each thread's slowdown due to sharing the DRAM system and prioritizes memory requests of threads that are slowed down the most. Unlike previous approaches to DRAM scheduling, STFM comprehensively takes into account inherent memory characteristics of each thread and therefore does not unfairly penalize threads that use the DRAM system without interfering with others. We show how STFM can be configured by the system software to control unfairness and to enforce thread priorities.

Our results show that STFM significantly reduces the unfairness in the DRAM system while also improving system throughput on a wide variety of workloads and CMP systems. For example, averaged over 32 different workloads running on an 8-core CMP, the ratio between the highest DRAM-related slowdown and the lowest DRAM-related slowdown reduces from 5.26X to 1.4X, while system throughput improves by 7.6%. We qualitatively and quantitatively compare STFM to one new and three previously-proposed memory access scheduling algorithms, including Network Fair Queueing. Our results show that STFM provides the best fairness, system throughput, and scalability.

1 Summary

1.1 The Problem

Today's chip multiprocessors share the DRAM memory system among different cores. Threads executing on different cores can interfere with each other while accessing the shared DRAM system. If inter-thread interference is not controlled, some threads could be unfairly prioritized over others while other, perhaps higher priority, threads could be starved for long time periods waiting to access the DRAM. Unfortunately, conventional high-performance DRAM memory controller designs do not take into account interference between different threads when making scheduling decisions. Instead, they try to maximize the data throughput obtained from the DRAM, usually via the state-of-the-art first-ready first-come-first-serve (FR-FCFS) policy [7, 6]. FR-FCFS prioritizes memory requests that hit in the row-buffers of DRAM banks over other requests, including older ones. If no request is a row-buffer hit, then FR-FCFS prioritizes older requests over younger ones. This scheduling algorithm is thread-unaware. Therefore, different threads running together on the same chip can experience extremely different memory system performance: one thread (e.g. one with a very low row-buffer hit rate) can experience a severe slowdown or starvation while another (e.g. one with a very high row-buffer hit rate) is unfairly prioritized by the memory scheduler.

In our MICRO-40 paper, we first quantitatively demonstrate the problem of unfair DRAM scheduling. To do so, we define a thread's *memory-related slowdown* on a CMP as the memory stall time (i.e. number of cycles in which a thread cannot commit instructions due to a DRAM access) the thread experiences when running simultaneously with other threads, divided by the memory stall time it experiences when running alone on the same CMP. We show that, in a representative desktop workload (among 256 evaluated workloads) run on a 4-core CMP system, one thread (*omnetpp*) experiences a slowdown of 7.74X whereas another (*libquantum*) experiences almost no slowdown at all (1.04X). The problem becomes even worse in 8-core and 16-core systems where the ratio between the maximum slowdown and the minimum slowdown significantly increases. In our previous work, published at USENIX Security 2007 [3], we demonstrated on a *real* dual-core CMP that the unfairness in the DRAM scheduler can be exploited by malicious *memory performance hog* programs to perform denial of service against other programs sharing the DRAM system: on an existing dual-core system a malicious *memory performance hog* causes another program to slow down by 2.9X whereas the malicious program itself slows down by only 18%.

Unfair DRAM scheduling is a significant impediment to building viable and scalable CMP systems because it can result in 1) very unpredictable program performance, which makes performance analysis and optimization extremely difficult, 2) significant discomfort to the end user who naturally expects threads with higher (equal) priorities to get higher (equal) shares of the computing system performance, 3) significant productivity and performance loss due to new denial of service vulnerabilities [3]. Therefore, to enable viable, scalable, and predictable CMP systems, DRAM scheduling must be made fair.

1.2 The Solution

Our MICRO-40 paper proposes a novel, low-cost, and high performance DRAM scheduler that provides a configurable fairness substrate for shared DRAM memory. Our scheduler operates based on a fundamentally new definition of DRAM fairness that takes into account inherent memory performance of each thread sharing the DRAM system.

DRAM Fairness: Defining fairness in DRAM systems is non-trivial. Simply dividing the DRAM bandwidth evenly across all threads is insufficient because this penalizes threads with high row-buffer locality, high memory parallelism, or in general, threads that by virtue of their memory access behavior are able to achieve better DRAM performance than others (as we show in Sections 4 and 7 of our paper). Our fundamental observation is that sustained DRAM bandwidth does not correspond to observed DRAM performance when threads interfere.

The key discovery that enables our solution is that a thread's performance degradation due to DRAM interference is primarily characterized by its *extra memory-related stall-time* caused due to contention with requests from other threads. This extra stall-time takes into account *all* factors that affect the thread's performance, including changes to bandwidth, locality, and parallelism due to inter-thread interference. Based on this intuition, *we define a DRAM scheduler to be fair if it schedules requests in such a way that the extra memory-related slowdown (due to interference caused by other threads) is equalized across all threads.*

*The full paper is: "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," O. Mutlu and T. Moscibroda, *In the 40th International Symposium on Microarchitecture (MICRO-40)*, December 2007. The full paper can be downloaded from http://research.microsoft.com/~onur/pub/mutlu_micro07.pdf

Basic idea: To achieve this fairness goal, our stall-time fair memory scheduler (STFM) estimates two values for each thread: 1) T_{shared}: Memory stall time the thread experiences in the shared DRAM system when running alongside other threads, 2) T_{alone}: Memory stall time the thread would have experienced had it been running alone (without any contending threads on other cores). Based on these estimates, the scheduler computes the *memory-slowdown* S of each thread where $S = T_{shared}/T_{alone}$. If *unfairness*, i.e. the ratio between the maximum slowdown value and the minimum slowdown value in the system, exceeds a threshold α (the threshold of maximum tolerable unfairness), the scheduler prioritizes memory requests from threads that are slowed down the most, thereby equalizing the experienced memory slowdown across all threads. Otherwise, the scheduler uses the baseline throughput-oriented FR-FCFS scheduling policy to maximize DRAM data throughput.

1.2.1 How STFM Works

The key design issue in STFM is how to estimate T_{shared} and T_{alone}. Accurately estimating T_{shared} is simple. The processor increases a counter when it cannot commit instructions due to an L2-cache miss. This counter is communicated to the memory scheduler. Accurately estimating T_{alone} is more difficult because STFM needs to determine how much memory stall-time a thread would have accrued if it had executed by itself on the CMP system (even though the thread is *not* running alone). Our paper describes in detail how to accurately estimate T_{alone}. Here, we briefly sketch our estimation technique.

Estimating T_{alone}: Since directly determining T_{alone} is difficult, we express T_{alone} as $T_{alone} = T_{shared} - T_{Interference}$ and estimate $T_{Interference}$ instead.[1] Initially, each thread's $T_{Interference}$ value is zero. *$T_{Interference}$ of each thread is updated whenever the STFM scheduler schedules a request.* When a request R from thread C is scheduled, STFM updates $T_{Interference}$ values of all threads: STFM updates $T_{Interference}$ differently for the request's own thread C versus for other threads.

1. Other threads: When a command is issued to a DRAM bank, the extra stall-time $T_{Interference}$ of all other threads that have a *ready request* (i.e. a request that can be scheduled by the controller without violating timing constraints) waiting for the DRAM bus or the same bank increases. Our technique includes in its estimate of $T_{Interference}$ all of the following key characteristics that affect DRAM performance: 1) extra stall time due to interference in the DRAM bus, 2) extra stall time due to interference in the DRAM bank, and 3) the memory-level parallelism [2, 1] of the thread (since the extra stall time will be amortized across accesses that can proceed in parallel).

2. Own thread: The thread whose own request is being scheduled may also experience extra stall-time because other threads could change the state of the row-buffers. To account for this, STFM keeps track of what the row buffer state would have been had the thread run alone and estimates extra latency the scheduled request incurs. STFM uses this estimate and the memory-level parallelism of the thread to update $T_{Interference}$.

Fairness-Oriented Scheduling Policy: Every DRAM cycle, STFM determines the slowdown values of all threads and computes the unfairness in the system. If unfairness in the previous DRAM cycle is greater than α, rather than using FR-FCFS based prioritization, the controller prioritizes the ready requests in the following order: 1) requests from the thread with the highest slowdown value, 2) row-hit requests, 3) older requests over younger requests. This policy allows STFM to improve fairness while still considering DRAM data throughput.

1.2.2 STFM's Support for System Software

Any complete solution to DRAM fairness needs to provide support for system software (OS or VMM) since system software sometimes might not want 1) fairness to be enforced by the hardware at all, 2) every thread to be treated equally because some threads can be (and usually are) more/less important than others. STFM seamlessly incorporates support for both these requirements (configurable α and thread weights), as described in Section 3.3 and evaluated in Section 7.5 of our paper.

1.2.3 Comparisons with Other Paradigms

Our MICRO-40 paper comprehensively compares STFM with one new and three previously-proposed DRAM scheduling algorithms (baseline FR-FCFS [7], FCFS [7], FR-FCFS+Cap (Section 4), and Network Fair Queueing (NFQ) [5, 4]) both qualitatively (Section 4) and quantitatively in terms of fairness, system throughput, and configurability (Section 7). We provide an especially detailed comparison to NFQ, a technique suitably used in network systems to partition bandwidth among different flows. We show that NFQ-based fairness definitions are suited primarily for *stateless systems that do not have parallelism* (such as a network wire), where a previous scheduling decision does not affect the latency (or performance impact) of later decisions, as opposed to shared DRAM memory systems that have state (row buffers) and parallelism (multiple banks), which affect the future performance impact of scheduling decisions.

STFM achieves better fairness and system throughput than all other algorithms because other schemes are prone to unfairly prioritizing one thread over another as they do not sufficiently consider the inherent memory performance characteristics of each thread (as explained in detail in Section 4). For example, NFQ unfairly prioritizes threads with bursty access patterns and penalizes threads with poor DRAM bank access balance because it considers only DRAM bandwidth. In contrast, STFM's key idea is to consider the inherent memory performance, including not only *bandwidth* but also *latency (locality)* and *parallelism*, of each thread. STFM therefore does not unfairly penalize threads that use the DRAM system without interfering with others or have different memory access characteristics.

1.2.4 Implementation and Hardware Cost of STFM

Our paper describes a detailed implementation of STFM and shows that STFM's storage cost on an 8-core CMP is only 1808 bits (Section 5). To support system software, STFM requires minor changes to the instruction set architecture that allow the system software to convey α and thread weights to the hardware.

1.2.5 Experimental Evaluation

We evaluated STFM on a wide variety and large number of workloads consisting of SPEC CPU2006 benchmarks and Windows desktop applications run on 2-,4-,8-, and 16-core CMPs using an x86 CMP simulator.[2] All our evaluations compare STFM to four other memory scheduling paradigms on 256, 32, and 3 workloads for respectively the 4-,8-, and 16-core systems. We provide detailed case studies of different types of workloads to provide insights into the behavior of scheduling algorithms and results (Sec. 7.2), quantitatively evaluate STFM's support for system software and thread weights (Sec. 7.5), and analyze STFM's sensitivity to DRAM banks and row buffer size (Sec. 7.6).

Figure 1 summarizes our key results, showing that STFM consistently provides the best fairness (i.e. lowest unfairness), the highest system throughput (in terms of weighted-speedup), and the best balance between throughput and fairness (in terms of hmean-speedup)[3] averaged over all the workloads run on 4-, 8-, and 16-core CMPs. The numbers above the STFM bars show STFM's improvement over the best previous DRAM scheduling technique for the given metric and system.[4] STFM improves unfairness by at least 1.27X, system throughput by at least 4.1%, and throughput-fairness balance by at least 9.5% compared to the best previous technique. STFM improves system throughput because it results in better system utilization (i.e. allows otherwise-starved cores to make progress and be utilized). We provide insights into *why* STFM performs better than other techniques via detailed case studies and analyses (Sec. 4 and 7).

[1] $T_{Interference}$ is a thread's extra memory stall time due to inter-thread interference. It can be positive if there is constructive interference due to data sharing. STFM seamlessly takes this into account.

[2] Our methodology and processor/DRAM parameters are described in detail in Section 6 of our paper.

[3] All metrics are defined and explained in Section 6.2 of our paper.

[4] No previous scheduling technique performs the best for all metrics and system configurations; e.g., for the 16-core system, FCFS provides the best fairness, but NFQ the best throughput. STFM consistently outperforms all other techniques for all metrics on all systems.

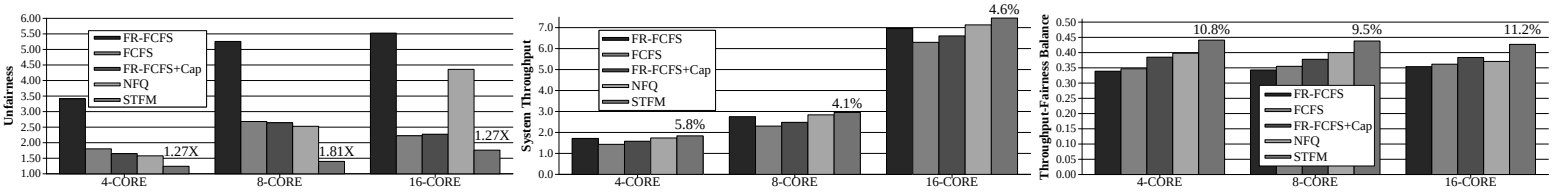

Figure 1: Comparison of STFM to other DRAM scheduling techniques in terms of (a) unfairness, (b) system throughput (weighted-speedup), and (c) throughput-fairness balance (hmean-speedup). The best achievable unfairness value is 1 in these workloads.

2 Significance

The DRAM system is a critical shared resource that determines the performance and scalability of CMP systems. Effective management (i.e. sharing) of the DRAM system performance among threads determines not only the quality of service (QoS) each thread running on a CMP experiences but also the overall performance of the system. As the number of cores/hardware-threads on a die increases, low-cost techniques that distribute the limited DRAM performance fairly across processing cores/hardware-threads becomes more desirable. Such techniques need to not only take into account all factors that affect DRAM system performance (bandwidth, locality, parallelism), but also be configurable enough to support the needs of the system software.

The primary contribution of our paper is a fundamentally new, comprehensive (including configuration support for system software), scalable, and low-cost technique for managing the shared DRAM resource such that the performance it provides is distributed fairly among different threads (in a way that also improves overall system performance).

2.1 Contributions of STFM

In our paper, we make the following major contributions:

1. *Our paper provides a fundamental and novel definition of DRAM fairness: stall-time fairness.* This definition takes into account all characteristics affecting DRAM performance (including not only DRAM bandwidth but also DRAM latency/locality and memory-level parallelism) of each thread executing on a CMP. We extensively compare the merits of stall-time fairness with previously proposed DRAM fairness definitions and provide insights into why stall-time fairness is a better-suited fairness definition for a shared DRAM system.
2. *STFM provides a simple fairness substrate that provides performance-based quality of service to threads sharing the DRAM system.* In previous research, it was not possible to fairly distribute *DRAM performance (i.e. slowdown)* among threads sharing the memory system. Our paper shows that fairly distributing *DRAM performance* provides better fairness and system throughput than fairly distributing DRAM bandwidth, which does not directly correspond to observed DRAM performance when threads interfere.
3. *STFM is fully configurable by system software and does not require information about applications for DRAM performance partitioning.* STFM seamlessly allows the system software to 1) choose the level of unfairness and 2) assign weights to threads. STFM does not require applications to specify their bandwidth requirements and can therefore provide weight-based performance partitioning to both existing and legacy applications and applications where such information might be input-dependent or might not be easily available. Thus, STFM offers a comprehensive solution to fair and configurable DRAM sharing.
4. *Our paper provides comprehensive performance, fairness, and scalability evaluations of different DRAM scheduling algorithms (FR-FCFS, FCFS, FR-FCFS+Cap, and NFQ) on a very large set of diverse workloads, including Windows desktop applications.* We find that STFM provides the best fairness and system throughput. STFM is also more scalable than other techniques in terms of both performance and fairness as the number of cores on a CMP increases.

2.2 Impact of STFM on Current/Future Processors

STFM, with its low cost and robustness to variance in workload behavior and system configuration, is immediately applicable to current-generation CMP systems to solve the problems caused by unfair DRAM sharing and provide protection against DRAM denial of service attacks. The importance of STFM will increase in future CMP systems due to two major reasons:

First, fairly distributing DRAM performance will become even more important in future CMP systems. The number of cores sharing the DRAM system will likely increase much faster than the performance (i.e. bandwidth, speed, latency, and parallelism) provided by the DRAM system. As more threads share the DRAM system, unfairness in the DRAM system will increase (and its consequences will be exacerbated [3]), as shown in our evaluations (Sec. 7). Hence, the DRAM system will be one of the primary limiters of the performance and QoS provided by future CMP systems. As our paper shows, STFM becomes more effective in terms of fairness and performance as the number of cores on a die increases (compared to other techniques).

Second, providing fairness and QoS to applications running on a CMP will become much more important because future CMP systems (handheld, desktop, and server) will likely host more diverse workloads that will have very different memory performance requirements and memory access characteristics. STFM provides an efficient and flexible way to supply significantly better QoS (and performance) to diverse sets of applications with varying memory characteristics than alternative DRAM scheduling techniques.

2.3 Scope of STFM and Stall-Time Fairness

STFM's scope is not limited to CMP systems. STFM is generally (and trivially) applicable to any shared DRAM system where different threads of execution can interfere, such as multi-threaded (e.g. SMT or GPU) systems and systems using helper threads.

Stall-time Fairness is a general fairness definition that we believe is applicable to shared resources beyond DRAM memory. Mechanisms can be developed to enforce this definition of fairness for shared processor, cache, or I/O resources.

We believe our method of estimating DRAM performance slowdown of a thread can be used for other purposes. For example, estimated slowdowns can be fed via performance counters to the operating system to adjust CPU scheduling and memory/cache-space allocation policies.

2.4 Impact on Future Research

DRAM controllers and QoS/fairness/resource-management issues are increasingly important research areas where not enough research has been done to build adequately fair, high-performance, and scalable CMP memory systems. Our paper provides a simple, effective, and comprehensive solution for supporting QoS in DRAM controllers. We believe and hope that the problems, the concepts of stall-time fairness and *performance partitioning*, and the solution described in our paper will attract the interest of researchers and lead to other novel solutions for effectively managing DRAM and other shared CMP resources.

REFERENCES

[1] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In *ISCA-31*, 2004.
[2] A. Glew. MLP yes! ILP no! In *ASPLOS Wild and Crazy Ideas*, 1998.
[3] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX Security*, 2007. `http://research.microsoft.com/~onur/pub/mph_usenix_security07.pdf`.
[4] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair queuing memory systems. In *MICRO-39*, 2006.
[5] A. K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Service Networks*. PhD thesis, MIT, 1992.
[6] S. Rixner. Memory controller optimizations for web servers. In *MICRO-37*, 2004.
[7] S. Rixner et al. Memory access scheduling. In *ISCA-27*, 2000.