Exploiting the Produce-Consume Relationship in DMA to Improve I/O Performance

Dan Tang[1,2], Yungang Bao[1], Yunji Chen[1], Weiwu Hu[1], Mingyu Chen[1]
[1] *Institute of Computing Technology, Chinese Academy of Sciences*
[2] *Graduate School of Chinese Academy of Sciences*
{tangdan, cyj, hww}@ict.ac.cn {baoyg, cmy}@ncic.ac.cn

Abstract

In this paper, we investigate the nature of DMA mechanism wherein there is an explicit product-consume relationship. Base on this observation we propose a DMA Cache technique to improve performance of DMA operations. To evaluate this technique, we adopt a hardware-based memory trace collection tool and an FPGA-based trace-drive emulation system. Experimental results show that DMA Cache can improve I/O performance significantly.

1. Introduction

I/O accesses are essential on modern computer systems, whenever we load binary files from disks to memory or download files from network. Moreover, many commercial workloads (e.g., database server, web server etc.) are I/O intensive. I/O buses and I/O devices technologies have been improved dramatically in the past decade. New I/O bus technologies, such as PCI-Express 2.0, AMD's HyperTransport 3.0 and Intel's CSI (QPI), can provide bandwidths of over 20GB/s which are very close to that of a DDR2/3 DRAM memory system. The I/O devices performance also increases significantly so that a RAID system is able to provide a bandwidth of 600MB/s easily and the 10Gb Ethernet can offer a bandwidth of 1.25GB/s.

DMA technique is used to release processor from I/O process, which provides special channels for CPU and I/O devices to exchange I/O data. However, as the throughput of the I/O devices grows rapidly, memory data moving operations have become critical for DMA scheme, which becomes a performance bottleneck for I/O operations. We investigate the nature of DMA and find that 1) there is an explicit producer-consumer relationship between CPU and DMA and 2) memory only play a role of transient place for I/O data. If we replace memory with a high speed buffer (e.g. a cache) as this transient place, the I/O performance may be improved significantly.

We propose a DMA cache technique which is adopting a dedicated cache (called DMA Cache) for buffering I/O data. This technique enables CPU and I/O device to exchange data between last level cache (LLC) and a DMA cache. The DMA cache technique can significantly reduce unnecessary memory data moving operations and largely improve I/O performance.

To evaluate the DMA Cache technique, we enhance a hardware-based memory trace collection tool, i.e. HMTT [1], to be able to distinguish CPU memory references and DMA memory references. With such an approach, we collect all CPU and DMA memory reference traces of various real applications (file copy, TPC-H and SPECweb2005) on a real machine. We adopt a RTL-based trace-driven emulation system mainly consisting of commercial components, e.g., L2 Cache, memory controller and DIMM model, to evaluate the proposal. We also use the FPGA-based RTL emulation accelerating system which is the Xtreme system [2] from Cadence to accelerate emulation.

Experimental results show that the DMA cache technique improves performance by an average of 11.8% for a 256KB DMA cache with sequential prefetching for all benchmarks (up to 20.7% for a file-copy application) when using snooping-cache as baseline, and it performs best among all schemes we have evaluated (e.g., snooping-cache scheme, shared-cache scheme).

The rest of the paper is organized as follows. In Section 2, we introduce the DMA mechanism and reconsider the role of memory in DMA process. Section 3 describes our efforts in promoting research methodology. Section 4 introduces our recent research results that we propose and evaluate a DMA Cache scheme for improving I/O performance. In Section 5, we discuss related work. We present future work in Section 6.

2. The Nature of DMA Mechanism

Figure 1 depicts the interaction of processor, memory and I/O device in DMA receiving mechanism. As shown in the figure, the interaction requires three data structures, i.e. DMA buffer, descriptor and APP buffer[1]. Usually DMA buffers' start address can vary in physical address space and their size also varies. DMA descriptors are used to manage these discrete DMA buffers. Each descriptor includes a pointer of DMA buffer's start address and a variable of DMA buffer's size. Each descriptor also includes some variables of status information such as DMA buffer's owner (could be processor or DMA engine).

According to the DMA receiving mechanisms, we can find that the essence of DMA scheme is that processor and I/O device have an explicit producer-consumer relationship. For instance, DMA engine is producer and processor is consumer in DMA receiving mechanism. Moreover, usually once data is produced, it will be used once and only once in most cases. This essence also exists in the DMA transmitting mechanism.

Furthermore, we can find that memory only plays a role of transient place for storing I/O data transferred between processor and I/O device. In DMA receiving processes, the data source is DMA engine (or I/O device) which writes source data (I/O data) to DMA buffer

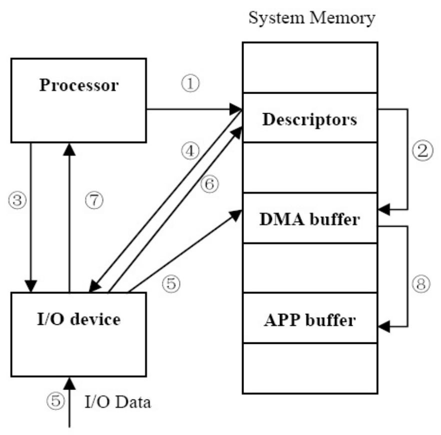

Figure 1. DMA Receiving Mechanism

[1] It should be noted that the "APP buffer" could be a buffer in kernel space or user space.

(see Step 5 in Figure 1). To provide I/O data for user applications, device driver performs an additional memory copy operation which copies I/O data from DMA buffer to the eventual destination – APP buffer[2] (see Step 8 in Figure 1). Once I/O data is copied to the eventual destination (APP buffer), DMA buffer is freed. Device driver will allocate a new DMA buffer for a new DMA receiving operation. However, the start address of the new DMA buffer is usually different from the previous freed one.

Based on the findings, we can conclude that the I/O performance should be improved significantly if the overhead of these additional memory-copy operations can be reduced. However, only a few people (e.g., DCA proposed by Huggahalli, Iyer and Tetrick [3]) realize the nature of DMA mechanism and, there are a large space for leveraging the nature of DMA mechanism for improving I/O performance while design computer architectures.

Next, we will introduce our research approaches in section 3 and some particular results on I/O subsystem section 4.

3. Research Methodology

Usually we use trace-driven simulation (or emulation) method for memory system and I/O system research. However, there are two substantial obstacles of effective simulation and emulation, i.e., 1) it is hard to collect an ideal memory trace which should be complete, detailed, undistorted and so forth [4]; 2) software simulation usually has speed and scalability limitations.

To overcome the two obstacles, we adopt a hardware snooping tool to collect all memory reference traces and an FPGA-based trace-driven emulation system to evaluate our proposals.

3.1 HMTT: A Hyper Memory Trace Tool

HMTT [1] is a platform independent full system memory trace monitoring system. The system adopts a DIMM-snooping mechanism, which uses hardware boards plugged in DIMM slots to track virtual memory

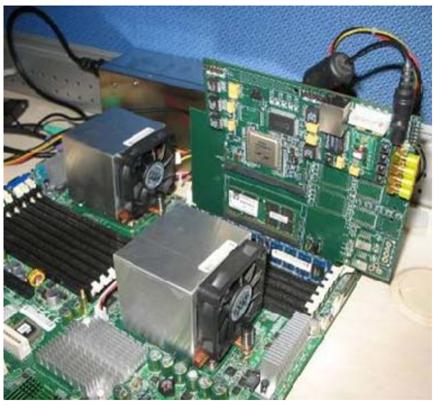

Figure 2. The HMTT Tool

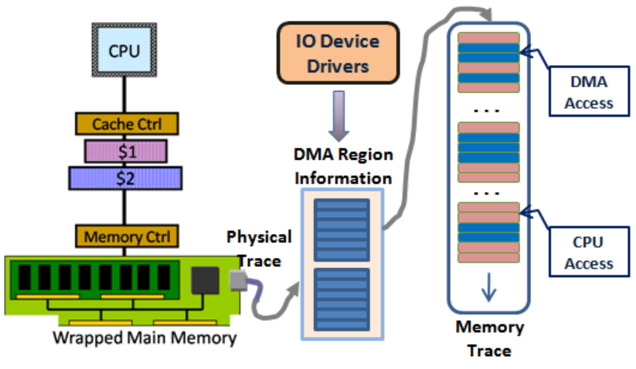

Figure 3. The Memory Trace Collection Framework

[2] I/O data may not be copied from DMA buffer to APP buffer. For example, the copy operations are not performed upon invalid or duplicate network packets.

reference trace of full systems (including OS, VMMs, libraries, and applications). Figure 2 shows a photo of HMTT in working state. Furthermore, HMTT provides APIs for user to inject user-defined tags into memory trace.

To distinguish a memory reference issued by DMA engine or processor, we have inserted HMTT's APIs into the device drivers of hard disk controller and network interface card (NIC) on Linux platform. Figure 3 illustrates the memory trace collection framework. When the modified drivers allocate and release DMA buffers (see details in section 2), they record start address, size and owner information of a DMA buffer. Meanwhile, they send synchronization tags to the HMTT system. When the HMTT system receives synchronization tags, it injects tags (DMA_BEGIN_TAG or DMA_END_TAG) into physical memory trace to indicate that those memory references between the two tags and within the DMA buffer's address region are DMA memory references initiated by DMA engine. The status information of DMA requests, such as start address, size and owner, is stored in a reserved physical memory and is dumped into a file after memory trace collection is completed. Thus, there is no interference of additional I/O access. We can differentiate DMA memory reference from processor memory reference by merging physical memory trace and status information of DMA requests.

We run all the benchmarks on a server machine. The server is an AMD Opteron 2.2GHz processor and 2GB dual-channel DDR memory machine. We use HMTT to collect memory reference traces of three real applications (file-copy, TPC-H, and SPECweb2005).

3.2 FPGA-based Trace-driven Emulation System

We adopt an FPGA platform to emulate cache system. Our cycle-accurate experimental system consists of a last level cache and a DDR2 controller from the Godson-2F processor, a DDR2 DIMM model from Micron Technology [5]. The whole cache emulation system is implemented in synthesizable RTL code. We use an FPGA based RTL emulation accelerator, the Xtreme system [2] from Cadence, to accelerate this system. A new design can be embedded into the cache emulation system for evaluation.

A combination of the HMTT memory trace collection tool and the trace-driven FPGA-based cache emulation system has significantly reduced our research periods. For example, we can use HMTT to collect memory traces with user-aware tags from various real machines without any slowdowns, analyze memory traces to gain some observations and insights and evaluate new ideas in the FPGA-based emulation accelerate system. In such a method, the research period can be reduced to only a few hours from trace collection, off-line analysis to emulation and result data collection.

4. The DMA Cache for Improving I/O Performance

In this section, we demonstrate our recent research results of a DMA Cache technique for improving I/O performance.

4.1 DMA Cache Scheme

The DMA cache is a dedicated cache for buffering I/O data. Like shared-cache scheme (i.e., DCA [3]), it is advantageous in reducing

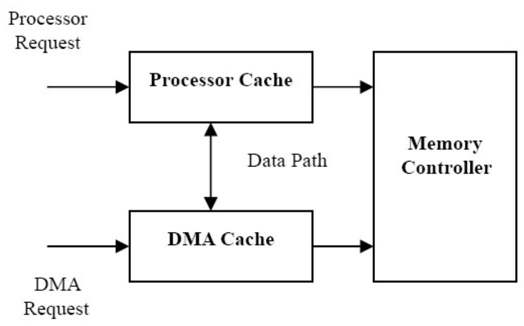

Figure 4. The Organization of the DMA Cache Scheme

the overhead of unnecessary memory copy operations. Unlike shared-cache scheme, the DMA cache technique is able to avoid interferences between processor and DMA engine. Furthermore, the DMA cache is beneficial from adopting straightforward prefetching technique because it only serves I/O data which has very regular reference pattern. Thus, CPU and I/O device can exchange data between the DMA cache and last level cache (LLC) to improve I/O performance.

Figure 4 illustrates the organization of the DMA cache scheme. In this scheme, DMA cache is a dedicated cache placed at the same level of the processor's last level cache (LLC) in memory hierarchy. A data path exists between the processor cache and the DMA cache for data exchange and coherence maintaining. In this paper, we mainly investigate the DMA cache system in a uniprocessor platform.

4.2 Design Issues

There are several important design issues.

Data Coherency: Usually, adding a new cache would raise data coherency problem. In DMA cache scheme, we solve this problem by the following method. In DMA cache scheme, both processor cache and DMA cache will be accessed upon each memory reference (could be issued by CPU or DMA engine). In the DMA Cache scheme, the state diagram of processor cache is nearly same as that of a typical uniprocessor which can handle I/O data coherency. The state diagram for DMA cache is similar to that of processor cache except driving sources. In such way, the DMA scheme is able to ensure that data would never exist in both caches at any time.

Data Migration between Processor Cache and DMA Cache: While a DMA memory read reference misses in DMA cache and hits in processor cache, the hit block will be migrated from processor cache to DMA cache. For a CPU memory read reference, the data migration direction is changed.

Replace Policy and Write Policy: We use an LRU-like replacement policy in which the DMA cache blocks are selected for replacement in the following priority: 1) First, the invalidated blocks will be selected. 2) Second, the least used clean block will be selected. 3) Finally, the least used dirty block will be selected. In addition, the DMA cache adopts a write back and write allocate policy.

Prefetching: In the direction of the "CPU write and DMA read" produce-consume relationship, where CPU produces I/O data into processor cache, the DMA Cache scheme cannot gain more benefits because the behavior of processor cache is unpredictable. In fact, DMA read has very regular access pattern, being linear within a DMA buffer. We find that even though the DMA cache adopts a sequential prefetching scheme [6] with a prefetching degree [6] of four cache blocks for DMA cache, it is able to significantly improve performance.

4.3 Evaluations

We have evaluated four schemes: (1) snooping-cache, (2) shared-cache, (3) DMA cache without prefetching and (4) DMA cache with prefetching. We have also evaluated various DMA caches with capacities of 256KB, 128KB, 64KB and 32KB. Figure 5 illustrates the normalized speedup of various schemes (the baseline is total cycles of snoop cache scheme). We can find that the DMA cache scheme with the size of 256KB and prefetching outperforms all other schemes. The performance of file-copy benchmark is improved by 20.7% due to its large portion of DMA memory reference (31%). In case of TPC-H benchmark which has 20% DMA memory reference, its speedup is about 8.0%. Although SPECWeb2005 only

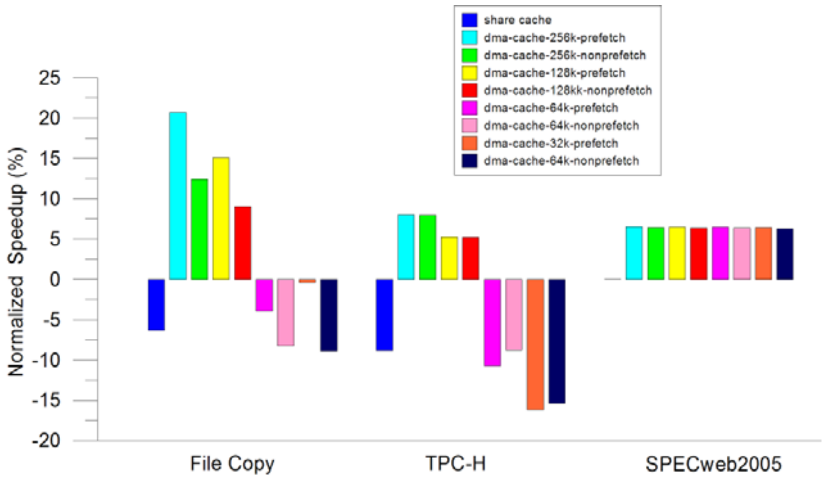

Figure 5. Normalized Speedup

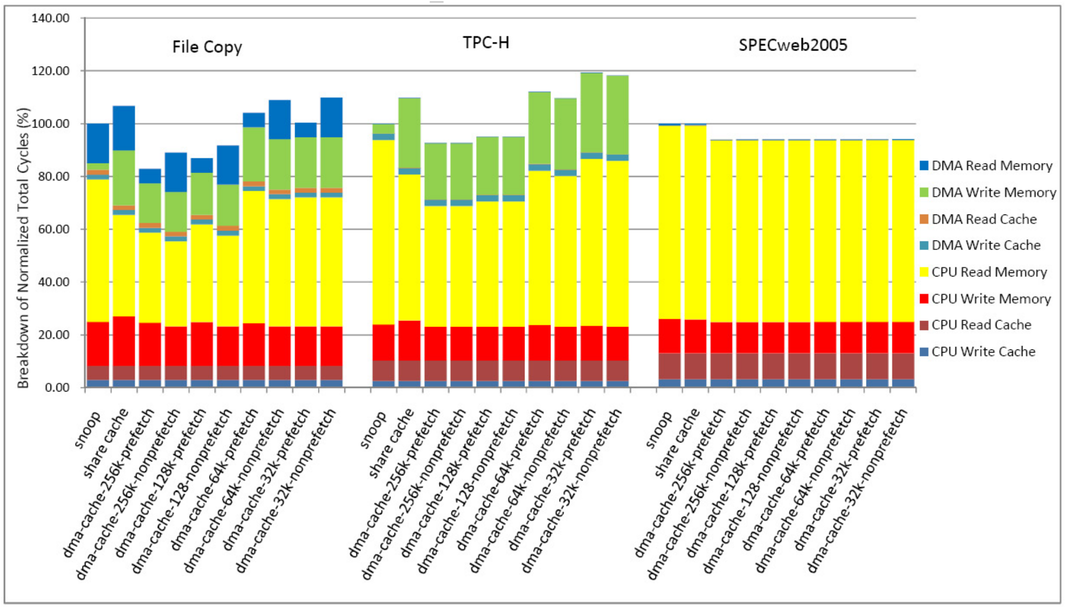

Figure 6. Breakdown of Normalized Total Cycles

has about 1% DMA memory reference, its performance is improved by 6.5%.

Figure 5 and Figure 6 illustrate the normalized speedup and the breakdown of normalized emulated cycles of shared cache and DMA caches with various configurations. From these two figures, we can see two interesting phenomena that: (1) the shared-cache scheme decreases performance for all benchmarks, especially for file-copy (-6.3%) and TPC-H (-8.8%). (2) DMA caches with the size of less than 128KB (i.e., 64KB and 32KB) exhibit even worse performance than the baseline (i.e., snooping-cache scheme) for file-copy and TPC-H benchmarks.

We investigate these two phenomena and find out the same reason that they both have poor DMA write memory performance (See the portion of DMA Write Memory of Figure 6). In snooping-cache scheme, all data is written into memory directly. In such situation, accessing DDR2 DRAM can achieve quite good performance due to few row buffer conflicts because the I/O data is written into a continuous address region. However, in shared-cache scheme, I/O data is injected into cache, incurring cache block replacement. Unfortunately, because the replaced dirty blocks are usually non-continuous, DDR2 DRAM exhibit poor performance due to lots of row buffer conflicts.

For those DMA caches with the size of less than 128KB (i.e., 64KB and 32KB), they also cause lots of row buffer conflicts in some benchmarks. Figure 7 depicts the cumulative distributions of the size of DMA requests for three benchmarks respectively. The average sizes of DMA write requests are about 110KB and 121KB for file-copy and TPC-H respectively. For SPECweb2005, the sizes of all DMA requests issued by NIC are smaller than 1.5KB because the maximum transmission unit (MTU) of Gigabit Ethernet frame is only 1518 bytes. The size of DMA requests issued by IDE controller for SPECweb2005 is also very small, an average of about 10KB. When DMA request size exceeds the size of the DMA cache, it will lead to frequent replacement for dirty blocks.

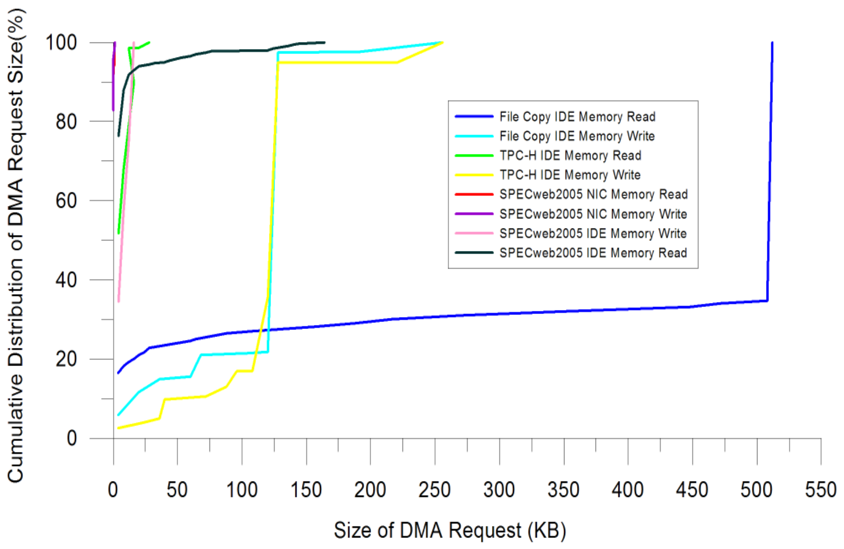

Figure 7. Cumulative Distribution of DMA Request Size

As mentioned before, those replaced dirty blocks are non-continuous, leading to poor DDR2 memory performance due to lots of row buffer conflicts.

The phenomena of shared-cache scheme are difficult to be removed because shared-cache scheme has to share all parts of processor's cache, including cache policies. However, the DMA cache scheme is able to avoid the phenomena by increasing the size of DMA cache or adopting a write-through policy, i.e., writing I/O data into memory directly and holding a clean-copy data in DMA cache simultaneously.

5. Related Work

Some studies have focused on reducing the overhead of additional memory copy operations for I/O data transfer. The key idea of these proposals is adopting shared-cache for DMA engine and processor to exchange I/O data. Iyer [7] proposed a chipset cache scheme to improve performance of Web Servers. The chipset cache handles the entire processor's last level cache misses as well as the DMA memory references. In fact, the chipset cache plays a role of last level cache shared by processor and DMA engine. Huggahalli et al [3] proposed a direct cache access (DCA) scheme that enables DMA engine and processor to exchange data within processor's cache. Their study focuses on network traffics and shows that the DCA scheme has poor performance for applications that have intensive disk I/O traffics (e.g. TPC-C). Some studies proposed the cache injection techniques [8,9,10,11] which is using some new instructions and some modifications of processor cache for I/O devices to inject I/O data into processor cache directly. In fact, the cache injection techniques can also be classified into the shared-cache scheme.

Although shared-cache scheme may be beneficial in reducing the overhead of additional memory copy operations, it can cause cache interferences between processor and DMA engine. Edgar et al [12] show that blind cache injection can be harmful to application performance. Our evaluations have shown that the interference can degrade I/O performance significantly when the size of I/O data increases from 1KB to over 100KB. Like shared-cache scheme, the DMA cache is advantageous in reducing the overhead of unnecessary memory copy operations. Unlike shared-cache scheme, the DMA cache technique is able to avoid interferences between processor and DMA engine. Furthermore, the DMA cache only serves I/O data which has very regular reference pattern. Thus, more special optimization techniques can be exploited for the DMA cache.

6. Future Work

Since the essence of DMA mechanism has been revealed, there would several other alternatives to improve I/O performance, such as improving shared-cache scheme with the same effect as the DMA cache scheme. We have been working on the following things:

- Exploring the optimization space of DMA cache scheme to achieve more improvements, such as adopting write-through policy.
- Evaluating the DMA Cache scheme under more complicated environments, such as multi-DMA channels and multicore platform.

- Designing and implementing a hardware/software cooperative approach to improve I/O performance by sharing last level cache.

References

[1] Y. Bao, M. Chen, Y. Ruan, L. Liu, J. Fan, Q. Yuan, B. Song, and J. Xu. HMTT: a platform independent full-system memory trace monitoring system. *In SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 229–240, New York, NY, USA, 2008.

[2] Xtreme system. http://www.cadence.com/products/fv/xtreme series/Pages/default.aspx.

[3] R. Huggahalli, R. Iyer, and S. Tetrick. Direct cache access for high bandwidth network I/O. *In ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 50–59, Washington, DC, USA, 2005.

[4] R. A.Uhilg, T. N. Mudge. Trace-driven memory simulation: A survey. *ACM Computing Surveys*, Jun., 1997.

[5] Micron technology inc. http://www.micron.com/

[6] S. P. Vanderwiel and D. J. Lilja. Data prefetch mechanisms. *ACM Comput. Surv.*, 32(2):174–199, 2000.

[7] R. Iyer. Performance implications of chipset caches in web servers. *In ISPASS '03: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 176–185, Washington, DC, USA, 2003.

[8] A. Milenkovic and V. Milutinovic. Cache injection on bus based multiprocessors. *In SRDS '98: Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, page 341, Washington, DC, USA, 1998. IEEE Computer Society.

[9] A. Milenkovic and V. M. Milutinovic. Cache injection: A novel technique for tolerating memory latency in bus-based smps. *In Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 558–566, London, UK, 2000. Springer-Verlag.

[10] V. M. Milutinovic and G. A. Sheaffer. The cache injection/cofetch architecture: initial performance evaluation. *In MASCOTS '97: Proceedings of the Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 63–64, Haifa, 1997. IEEE Computer Society.

[11] R. R. P. Bohrer and H. Shafi. Method and apparatus for accelerating input/output processing using cache injections. *In US Patent* No. US 6,711,650 B1, 2004.

[12] E. A. Leon, K. B. Ferreira, and A. B. Maccabe. Reducing the impact of the memorywall for I/O using cache injection. *In HOTI '07: Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects*, pages 143–150, Washington, DC, USA, 2007. IEEE Computer Society.