Proceedings of the 26th Annual

International Symposium on Microarchitecture

December 1–3, 1993 Austin, Texas

Sponsored by

IEEE Technical Committee on Microprogramming and Microarchitecture
and
Association for Computing Machinery SIGMICRO

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

A Microarchitectural Performance Evaluation of a 3.2 Gbyte/s Microprocessor Bus *

Tim Stanley, Michael Upton, Patrick Sherhart
Trevor Mudge, Richard Brown

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

Several architectural innovations intended to reduce access latency and improve overall throughput increase system bandwidth requirements. Bandwidth scales with clock speed, and can be regarded as an architectural resource to be applied to latency reduction. A properly designed bus provides low arbitration latency and delivers high sustained bandwidth.

This paper evaluates the performance of 3.2 Gbyte/s peak bandwidth, low-latency arbitration bus connecting a GaAs superscalar CPU to a GaAs memory management unit. A microarchitectural performance model was written in the Verilog hardware description language. Bus transactions characteristic of the SPECint92 benchmarks and other workloads were generated as input. Sustained bandwidths of 1.68 Gbytes/s were achieved with arbitration costs of less than 0.5 cycles per data transfer.

Keywords: *I/O Microarchitecture, Bandwidth, Latency, Hardware Description Language, Performance Modeling.*

1 Introduction

Processor clock speeds are increasing at 50 % per year [1], due to technology improvements and a better understanding of the interaction between technology and architecture. Superscalar CMOS microprocessors have reached clock speeds of 200 MHz [2]; an ECL microprocessor has achieved speeds of 300 MHz [3]; and GaAs promises to become a viable processor technology at speeds exceeding 200 MHz [4].

*This work was supported by the Advance Research Projects Agency under DARPA/ARO Contract DAAL03-90-C-0028.

Meanwhile, it has been widely observed that the speed of DRAM technologies is increasing at only about 7 % per year and is not keeping pace with microprocessor performance. For this reason, the impressive gains of microprocessor clock speeds may not directly translate into higher system performance. Necessarily, numerous architectural innovations have been developed to minimize the processor and memory hierarchy performance mismatch (see Table 1).

Some researchers suggest that a reduction in latency, rather than an increase in peak bandwidth might lead to faster machines [1]. We agree and would be pleased to find memory access latencies scaling with our clock speed. Until that time, we note that:

- Because memory speeds have not kept pace with clock speed, relative access latency appears to have increased from a system perspective.
- Prefetching and streaming techniques, intended to reduce latency, indeed require additional system bandwidth.
- Though latency does not scale with processor clock frequency, pin bandwidth potentially scales when using advanced packaging technologies.
- High peak bandwidths are necessary to support latency reduction techniques and multi-issue implementations. Few implementations have failed for an abundance of bandwidth and we expect that trend to continue.

For these reasons, we are interested in very high bandwidth inter-chip communication in GaAs microprocessor systems. While GaAs offers impressive on- and off-chip bandwidth, it does not offer dense, low-power, short-term storage so well exploited by recent

1072-4451/93 $3.00 © 1993 IEEE

Architectural Technique	Effect on latency and bandwidth
Latency Reduction	
Caches and write-buffers	use locality of reference and the size/speed characteristic of small memory structures to provide low-latency, high-bandwidth access on the frequent cache hit case, and long latency, low bandwidth access on the infrequent cache miss case.
Hardware prefetching [5], stream buffers [6], and software-directed prefetching [7]	effectively trade increased bandwidth for reduced latency.
Latency Tolerance	
Non-blocking and lockup-free caches [8]	lessen the impact of data dependencies by allowing execution to proceed while a cache miss is serviced.
Dynamic scheduling techniques	trade additional latency to achieve better latency tolerance and high throughput. This is an instance where latency is transformed to increased throughput (bandwidth).
Architecturally specified load and branch delay slots [9]	tolerate latency by filling empty pipeline stages.
Compiler loop unrolling and code reordering	tolerate latency by exposing instruction and data parallelism.
Increased Bandwidth Requirements	
Multi-issue processors	improve system performance by increasing the *actual* bandwidth built into the system but further exacerbate the impact of latency on overall system performance. In general, load/store implementations intended to perform *more than* 1 instruction per cycle are increasingly more sensitive to branch, load, and cache miss latency [10]. These architectures demand concurrent I- and D-stream access.

Table 1: Effect of Recent Architectural Trends on System Bandwidth and Latency.

Recent architectural trends can be grouped into three categories: techniques to reduce average access latency; techniques to tolerate latency; and techniques that use additional system bandwidth to increase throughput.

CMOS processors. Limited GaAs integration densities force us to consider multi-chip partitions with accompanying off-chip latencies that must be tolerated in the overall design. We believe that bandwidth is a resource to be applied to the latency reduction problem.

1.1 Contributions of this Work

To address these concerns, our research concentrates on the interaction of architectural innovations on bandwidth-latency requirements for high performance GaAs systems. As GaAs designers, we are compelled to consider architectural alternatives that trade advantages of our technology to compensate for its limitations; specifically, can we architecturally trade bandwidth to and from smaller and more specialized local storage to improve average access latency? This paper provides a detailed and accurate evaluation of the bandwidth and latency performance a microprocessor bus designed to provide 3.2 Gbyte/s bandwidth and near zero cycle arbitration overhead.

A system overview and the bus protocol are described in Section 2. Section 3 discusses the hardware description language (HDL) model of this bus interface unit (BIU) and the behavioral system models representing the design space of our GaAs multi-chip system. Section 4 describes the bus transaction mixes characteristic of the SPECint92 benchmarks and other workloads. Performance measurements are used to determine the bandwidth requirements, arbitration efficiency, usage patterns, and performance sensitivities of the CPU, MMU, and BIU system. In Section 5, conclusions are drawn regarding the ability of the bus and arbitration scheme to support a 300+ MHz GaAs chipset and the benefits of microarchitectural performance analysis using HDLs are discussed.

2 Aurora3 Organization

We are building a high performance GaAs chip set, named Aurora3 and shown in Figure 1. The 300+ MHz superscalar CPU chip implements a subset of the the MIPS ISA [9], has a non-blocking D-cache, and provides hardware I- and D-stream prefetching. The 300+ MHz Cache and Memory Management Unit (MMU) implements concurrent I- and D-stream datapaths, off-chip secondary caches, hardware prefetching, memory management support, and an I/O bus interface. GaAs integration levels do not yet permit a single-chip implementation of this system. The target bandwidths and latencies for the major

System Bandwidths			
Chip	Bus	Bandwidth	Latency
CPU (300 MHz)	Register File (4 read ports)	4.8 Gbyte/s	1 CPU cycle
	Register File (2 write ports)	2.4 Gbyte/s	1 CPU cycle
	L1-I cache	2.4 Gbyte/s	1 CPU cycle
	L1-Dcache/FPU read	2.4 Gbyte/s	3 CPU cycles
	L1-Dcache/FPU write	2.4 Gbyte/s	3 CPU cycles
CPU/MMU	BIU (400 MHz)	3.2 Gbyte/s	20-100 BIU cycles
MMU (300 MHz)	L2-Icache/Rambus	1 Gbyte/s	hit >72 ns miss > 246 ns
	L2-Dcache/Rambus	1 Gbyte/s	hit >72 ns miss > 246 ns
	I/O interface (non-GaAs)	100 Mbyte/s	
FPU (250 MHz)	FP Register file (4 64-bit read ports)	8 Gbyte/s	1 FPU cycle
	FP Register file (1 64-bit write port)	2 Gbyte/s	1 FPU cycle

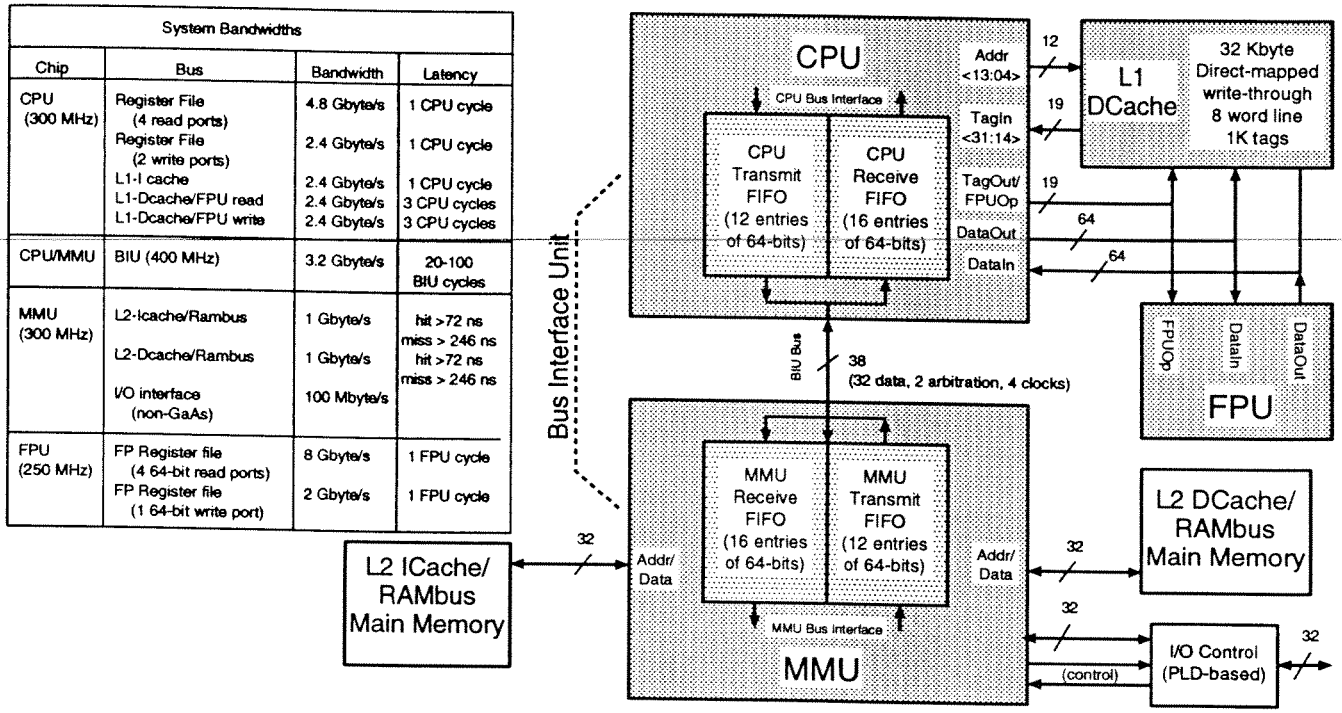

Figure 1: Aurora3 System Block Diagram and Bus Bandwidths.

Aurora3 system busses are tabulated in Figure 1.

Over 180 pins are dedicated to the data cache and FPU interface, providing the maximum bandwidth at the top of the memory hierarchy (refer to Figure 1). For our prototype implementation, the CPU package is constrained to 256 signal pins. A major design decision involved the most effective use of the remaining pins to achieve a high-bandwidth, low-latency connection between the CPU and MMU. More aggressive packaging schemes such as multi-chip modules allow higher off-chip I/O pin counts. However, power considerations still make a narrow, high bandwidth channel a more desirable alternative than simply building a wider bus.

2.1 Bus Interface Unit

The Bus Interface Unit connects the CPU to the MMU via a unique bidirectional 32-bit bus, as shown in Figure 2. To tolerate transaction latency, multiple pending requests are buffered in the BIU queues (FIFOs). The FIFOs also allow the BIU to operate asynchronously with respect to the CPU and MMU. Control is passed between these 3 clock domains using synchronizers, allowing each domain to operate at its own maximum frequency. A consequence of this latency-tolerant organization is that a memory read transaction will suffer 4 synchronization penalties during the round trip.

The design philosophy is to stream data through the FIFOs at high-bandwidth (a GaAs advantage), rather than dedicate large amounts local memory for short-term storage (costly in GaAs). The bus master will *burst* the entire contents of its transmit FIFO (tFIFO) to the receiver before relinquishing the bus. Transmit FIFOs are 12 64-bit entries in size and receive FIFOs (rFIFO) are 16 64-bit entries in size. An rFIFO can hold an entire tFIFO burst.

2.1.1 BIU Timing

The maximum bandwidth available across a bus is limited by several factors including interconnect medium bandwidth, signal rise/fall times, signal setup/hold times, clock skew, and transmission delay. The ideal bus would be limited by the bandwidth of the interconnect medium, and failing this by the signal rise/fall times. We have adopted a signaling scheme similar to that used by Rambus [1] [11] that is limited by the signal rise/fall times rather than by clock skew or signal setup/hold time. It is suitable for board level and multi-chip module packaging, and matches the clock bandwidth with the data bandwidth.

Figure 3 shows a typical BIU transaction. Each chip generates its own transmit clock that is shifted

[1] Rambus is a trademark of Rambus Inc.

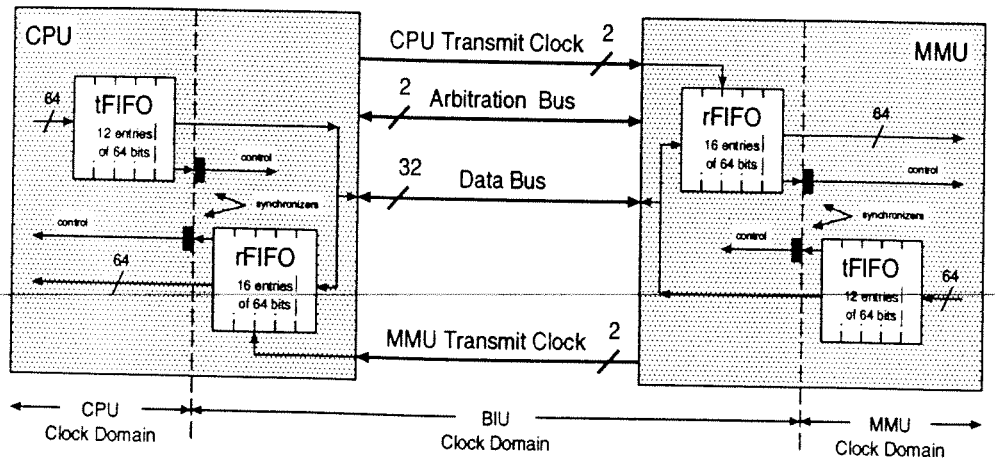

Figure 2: Bus Interface Unit Block Diagram.

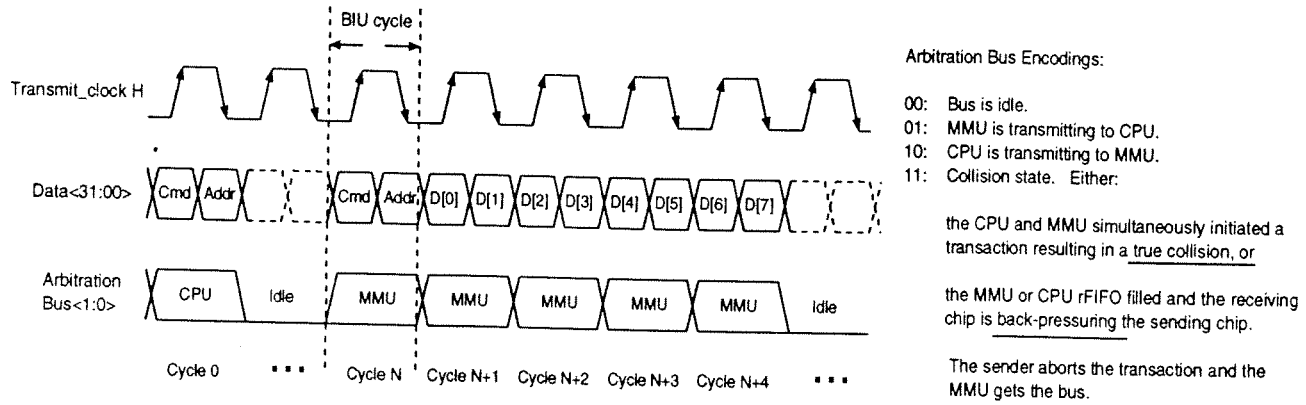

Figure 3: Bus Interface Unit Cycle Timing. CPU Load Transaction.

in phase by 90 degrees with respect to the data. By transmitting the clock and data to the receiver together, maximum setup and hold times are achieved. Clock skew is still a critical factor, but is minimized by uniformly routing the clock and data lines together so that clock skew is equal to data skew. Data is edge-triggered by the receiver on both edges of the clock, reducing the required clock bandwidth by a factor of two when compared to a system that transmits data only on a single clock edge. A BIU transmitting 4 bytes of data on each 400 MHz clock edge yields a 3.2 Gbyte/s peak bandwidth.

2.1.2 BIU Arbitration

The BIU interface uses a carrier-sense, distributed and low-latency arbitration scheme. A 2-bit arbitration bus connects the CPU and MMU using ECL wire-OR signals (see Figure 3 for bus encodings). When the bus is idle these lines are pulled low by the ECL terminators and either unit is free to immediately begin transmitting data. As soon as the receiver recognizes the transmission, it inhibits its own transmissions to avoid bus contention.

This 2-party, collision-based arbitration scheme is designed to fall into a favorable ping-pong pattern where each chip alternates *streaming* the entire contents of its tFIFO to the other chip with no collisions, and therefore, zero arbitration latency overhead.

When the CPU sends data, it drives a binary $< 10 >$ onto the arbitration bus. When the MMU sends data it drives $< 01 >$. If both units transmit simultaneously, the bus will be $< 11 >$. This is called a *true collision* and is similar to a collision in the Ethernet protocol.

If either rFIFO controller detects that its rFIFO is full while receiving a transmission, it forces the arbitration bus into the $< 11 >$ collision state to abort the transaction. Partial bursts in the rFIFO are discarded and must be retransmitted again later.

After either collision type, both units stop transmission, and the MMU takes control of the bus even if its tFIFO is not yet ready to transfer. The MMU has priority because it is supplying data previously requested by the CPU and this avoids any possibility of deadlock. Also, the MMU to CPU bandwidth requirement exceeds the CPU to MMU requirement for all workloads. This method of deadlock avoidance has the favorable side-effect of dynamically balancing the system by giving the more demanding MMU priority under heavy loads (see Section 4).

2.1.3 BIU Transactions and Transfers

Each BIU *transaction* begins with a CPU command-address pair. Referring to cycle 0 of Figure 3, the command is removed from the CPU tFIFO and transmitted to the MMU 90 degrees prior to the rising transmit clock edge. The address is transmitted 90 degrees prior to the falling edge to complete the pair. If the transaction is a CPU store request, this command-address pair is *immediately* followed in cycles 1-4 by 8 words of write data (organized as 4 pairs) from the CPU tFIFO. The MMU does not send any return status back to the CPU once it accepts the entire request. Thus, a CPU to MMU store transaction is composed of a single *transfer* of information. A burst is composed of multiple transfers.

Because the latency to the MMU and secondary cache is many cycles, it is undesirable to hold the bus for the duration of a load or prefetch transaction. If the CPU command-address during cycle 0 is a load or prefetch request, the BIU uses a split-transaction protocol composed of two individual *transfers*: the CPU request and the MMU response. The MMU holds the request in its rFIFO until it can be processed. The CPU uses non-blocking caches, and is free to initiate more requests if possible.

The MMU moves the transaction from its rFIFO to a small staging FIFO in either the I- or D-stream functional unit. This keeps the head of the MMU rFIFO exposed for further I- and D-stream concurrency. When the data is fetched from memory, the command-address pair and 8 words of read data (organized as 4 pairs) are loaded into the MMU tFIFO. This transfer is then sent back to the CPU as shown in cycles N through N+4 of Figure 3, completing the load or prefetch split-transaction.

3 Simulation Method

Architectures are often evaluated using performance models written in C or C++. These models describe the computer structures at a level of abstraction as low as the register-transfer level (RTL) but typically much higher. The input to these models is an address trace of a workload and a simulator configuration. The output is the corresponding predicted performance. This prediction is accomplished by counting the occurrence of specific events (e.g., cache misses) during the simulation, assigning an average cost to each event (e.g., cycles to service the cache miss), and summing the products of counts and costs. This method is sufficient for high-level performance analysis.

Performance evaluation at the microarchitecture level, i.e., the RTL level of abstraction, must be more accurate. Designers often notice that cost assumptions used in the high-level performance model are dramatically oversimplified once low-level and complex implementation details are considered. For example, the time to arbitrate for a shared bus is longer than predicted; the time to turn a tristate bus from driving to receiving adds unanticipated latency to the cache fill control flow; latency-sensitive data spends more time in a queue when the system is under a heavy load than first predicted; etc. Moreover, a microarchitect needs to explore the design space toward identifying an implementation with the appropriate cost/performance. Many design methodologies preclude this option by presenting a detail-intensive level of abstraction to the microarchitect that is too low.

3.1 BIU Structural Model

We have overcome this limitation in the design and evaluation of the BIU by structurally modeling with the Verilog [2] HDL. This high-level language description is then compiled and synthesized to VLSI layout using the EPOCH toolsuite [3]. Thus, the models are implementation-accurate and design quality is tested by reviewing the output of the physical design tools.

The BIU is a complex and performance-critical subsystem of Aurora3. The bidirectional bus collisions and rFIFO back-pressures yield complex and dynamic behavior. BIU datapath and controller design explorations are readily accomplished by simply editing the Verilog language source code description, running the BIU under various workloads and monitoring its performance.

[2] Verilog is a trademark of Cadence Design Systems, Inc.
[3] EPOCH is a trademark of Cascade Design Automation Corporation

Input Parameters & Description		Experimental Values	
		SPECint92	DAXPY
Workload	Mean rate at which the CPU generates (i.e., places command/address in CPU tFIFO) new transactions.	8 CPU cycles	2 CPU cycles
	Instruction-stream / Data-stream request ratio.	70/30	0/100
	Load/Store/Prefetch % request mix.	20/60/20	0/34/66
CPU	Clock speed.	298 MHz	
	Number of pending transactions.	1 to 16	
BIU	Clock speed.	397 MHz	
MMU	Clock speed.	298 MHz	
	Mean latency for MMU to retire (i.e., begin placing return data in MMU tFIFO) a load or store request.	8 MMU cycles	2 MMU cycles
	Mean latency for MMU to retire a prefetch request.	2 MMU cycles	2 MMU cycles

Table 2: Aurora3 SPECint92 and DAXPY Workload and System Configurations.

The most interesting performance results are observed when the bus load is a function of the number of pending transactions, i.e., given a non-blocking cache, the number of outstanding load and prefetch requests that can be active at one time without stalling the CPU.

The issue and retire rates are randomly generated using a normal distribution function. The mean values are listed above.

3.2 System Behavioral Model

Behavioral Verilog CPU and MMU models exercise the structural Verilog BIU model. These models have several configurable parameters to generate the desired workload and system characteristics. The values assigned to each parameter for two workloads of interest to this study are shown in Table 2.

The rate at which the CPU issues bus transactions, the ratio of I- to D-stream transactions, and the mix of load/store/prefetch transactions were determined by traditional trace-driven simulations of the Aurora3 system using the SPECint92 benchmarks.

We assume that the CPU can dispatch *up to* 4 non-blocking D-stream loads at once, and that our hardware prefetch strategy will issue one prefetch request for every load request for a total of 8 outstanding D-stream transactions. An I-stream load request also results in a prefetch for 2 more additional outstanding requests. As such, we are interested in the BIU performance when the range of outstanding transactions is varied from 1 to 16.

The DAXPY workload configuration represents the CPU running a DAXPY inner-loop having 2 load/1 store sweeping through the small primary D-cache, and all I-stream references are captured by the primary on-chip I-cache. Due to high spatial D-stream locality of a DAXPY inner-loop, we expect aggressive hardware prefetching to be successful through the memory hierarchy and assign lower MMU response latencies accordingly. We use this workload to expose the BIU's bandwidth, latency, and arbitration performance limits as bus load increases from a realistic number of outstanding prefetches (1-4), to a very heavy bus load (8-16).

The behavioral CPU and MMU models comprise a self-checking system. Two copies of memory are maintained. One copy is used by the MMU to service CPU requests; the other copy is used by the random transaction generating procedures of the behavioral CPU to hold expected answers. All transactions are verified for correctness. Also, an error is indicated if a transaction does not return within a maximum time.

4 Experiments and Analysis

All graphs represent the mean of 20 simulation runs per data point at 10,000 CPU cycles per simulation run.

4.1 SPECint92 Workload Results

Figure 4 shows that the average bandwidth achieved as a function of the number of pending transactions plateaus at 1359 Mbytes/s (of nearly 3.2 Gbyte/s raw available bandwidth) between 8 to 16 outstanding transactions. This bandwidth represents cumulative Command/Address *and* Data communication across the bus. The MMU transmits about 71 % of the bandwidth, and the largest component of BIU traffic for this workload is for the MMU I-stream. For

this workload, 31 % of the total traffic is dedicated to Command/Address transmission.

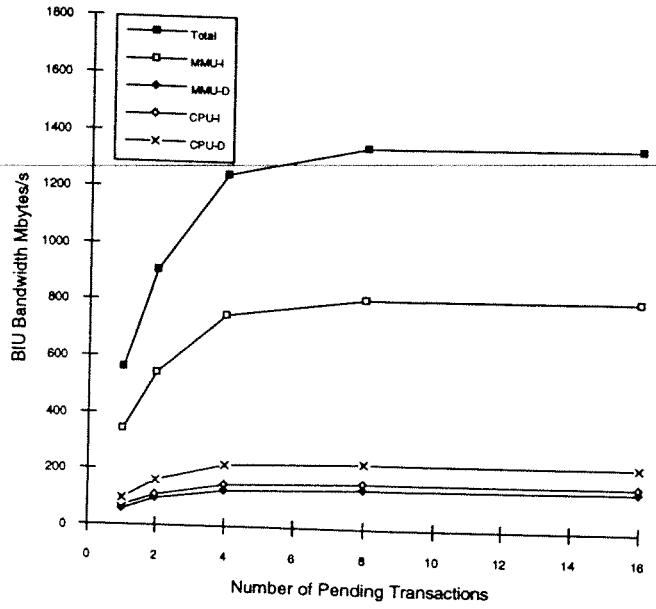

Figure 4: Bandwidth for SPECint92.

Bandwidth nearly levels off at 1359 Mbyte/s at 8 pending transactions.

To understand why average bandwidth nearly levels off when there are 8-16 pending transactions, refer to the load latency breakdown shown in Figure 5. Total load latency (in CPU cycles) can be broken down into 4 legs of the round-trip:

Latency-1 time from when command-address enters the CPU tFIFO until it is removed from the head of the MMU rFIFO and dispatched to either the I- or D-stream MMU functional unit staging FIFO,

Latency-2 time from Latency-1 until the MMU I- or D-stream functional unit removes the command-address from its local staging FIFO,

Latency-3 time from Latency-2 until the MMU begins loading the response into its BIU tFIFO,

Latency-4 time from Latency-3 until the command-address is removed from the head of the CPU rFIFO.

As the number of pending transactions increase, total load latency increases from 26 cycles to 47 CPU cycles at 16 pending transactions. The Latency-2 component of total latency, i.e., the time a transaction spends in the MMU I- or D-stream functional unit staging FIFO waiting to be serviced, strongly increases from 1 to 8 pending transactions. It levels off at about 15.9 cycles above 8 pending transactions and is the largest contributor to total latency. Latency-3, the programmable time for the MMU to retire a request, is 8 cycles. For this configuration, between 16 and 39 cycles of memory access latency are spent on the BIU FIFO communication mechanism.

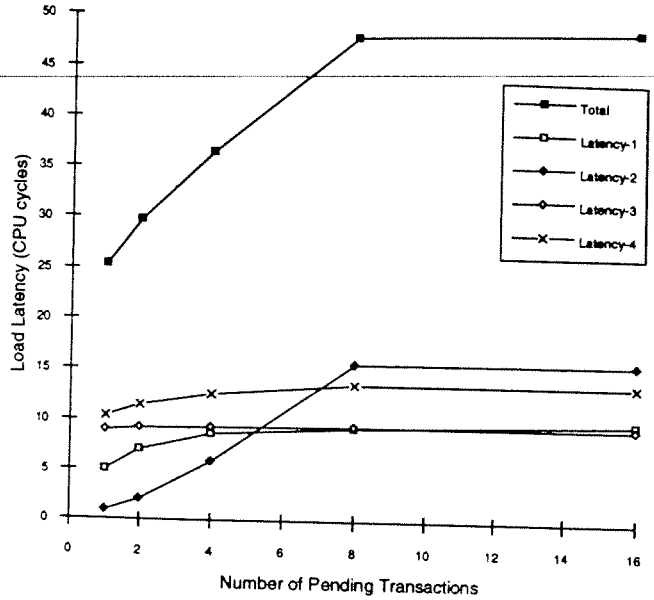

Figure 5: Load Latency for SPECint92.

The major latency component is due to Latency-2, i.e., time spent in the MMU staging FIFOs.

When the number of pending transactions is between 8 and 16, the BIU bandwidth is sufficient to support this system. The MMU rFIFO buffers the CPU requests. The MMU I- and D-stream controllers operate concurrently and effectively keep up with the CPU requests, though latency increases. The CPU can not keep 16 outstanding transactions in flight because the MMU I- and D-stream datapaths have enough bandwidth to retire requests quickly. Traffic is light enough that neither the CPU, nor the MMU rFIFO back-pressures the system. Bandwidth reaches a plateau as each chip takes turns bursting its tFIFO contents to the other chip's rFIFO. The average CPU burst length about 3.4 fifo entries in length, and the average MMU burst length is 7.4 fifo entries in length.

The 12-entry tFIFOs are full less than 10 % of the time, and the collision overhead, our measure of bus arbitration cost, is quite low. About 8 % of bursts result in a true bus collision. The SPECint92 benchmarks are not particularly bandwidth intensive compared to scientific code having a high degree of data parallelism. Therefore, the remainder of this analysis will focus on a different workload and system configur-

ation that better explores the BIU performance limits.

4.2 Scientific Workload Results

To further stress the system and bus load, a workload configuration representing the CPU running a scientific benchmark was developed. This configuration represents, e.g., a DAXPY inner-loop having 2 load/1 store sweeping through the small primary D-cache, and all I-stream references captured by the primary on-chip I-cache.

Referring to Table 2, the mean rate at which the CPU initiates transactions was reduced to 2 cycles and the rate at which the MMU services loads and stores was reduced to two cycles. The real CPU running a DAXPY inner-loop will aggressively and accurately prefetch all load references. Prefetch requests account for 66 % of transactions and 34 % are store requests. This optimistic and aggressive configuration is intended to be quite bandwidth intensive.

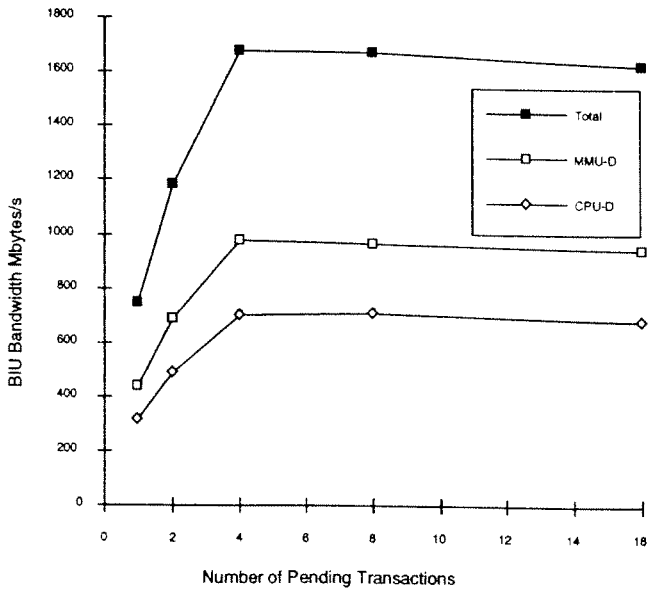

Figure 6: Bandwidth for DAXPY.

Bandwidth reaches 1681 Mbyte/s for 4 pending transactions before decreasing slightly to 1628 Mbytes/s at 16 pending transactions.

The average bandwidth required for this experiment was as high as 1681 Mbytes/s for 4 pending transactions, and dropped slightly to 1628 Mbytes/s for 16 pending transactions (see Figure 6). In this configuration, the system uses over 50 % of the 3.2 Gbyte/s peak bandwidth.

Figure 7 shows the average prefetch transaction latency breakdown. Latency starts as low as 20 CPU cycles, but climbs dramatically to 84 CPU cycles as the bus load increases. Only 2 cycles are due to the actual MMU (Latency-3), the remainder are attributed to the FIFO-based communication. This dramatic latency increase is due to the lack of I- and D-stream concurrency in the MMU. Though bus bandwidth remains quite high, the MMU D-stream functional unit can not keep pace with the CPU transaction rate. Latency-1 (latency in the MMU rFIFO) is the largest contributor to overall latency for this workload and does not level off to a steady value.

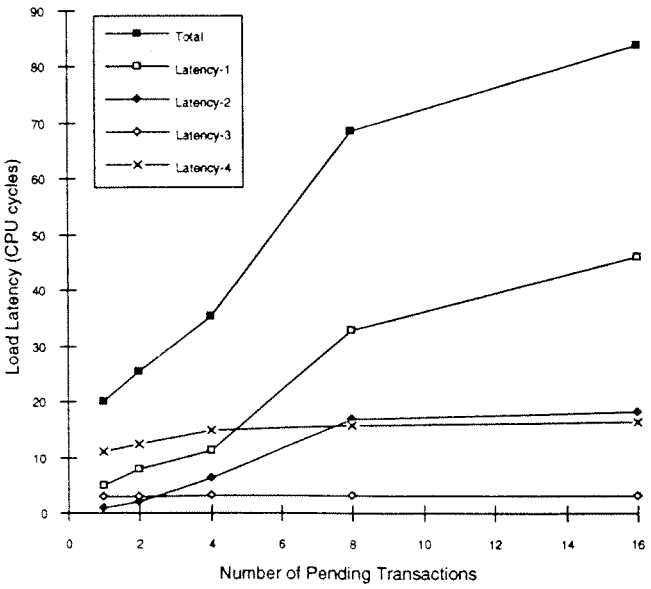

Figure 7: Prefetch Latency for DAXPY.

As bus load increases, the MMU rFIFO back-pressures the system. This drives total latency from 20 to 84 CPU cycles.

An indirect cost of the FIFO-based communication is the latency increase as bursts are aborted and reattempted due to rFIFO back-pressure. The impact of the MMU rFIFO overload can be readily seen on the average collisions per successful burst graph and the BIU FIFO utilization graph (Figures 8 and 9. Recall that a *burst* is defined as a transmission of the entire tFIFO contents by the sender. The total collisions/burst curve never levels-off and 2 regions of behavior are observed.

From 1 to 4 pending transactions, the BIU and the MMU have sufficient bandwidth to support the chipset. All bus collisions are due to true bus contention, i.e., both chips attempt to transmit at nearly the same time. The total traffic requirements are quite low, and

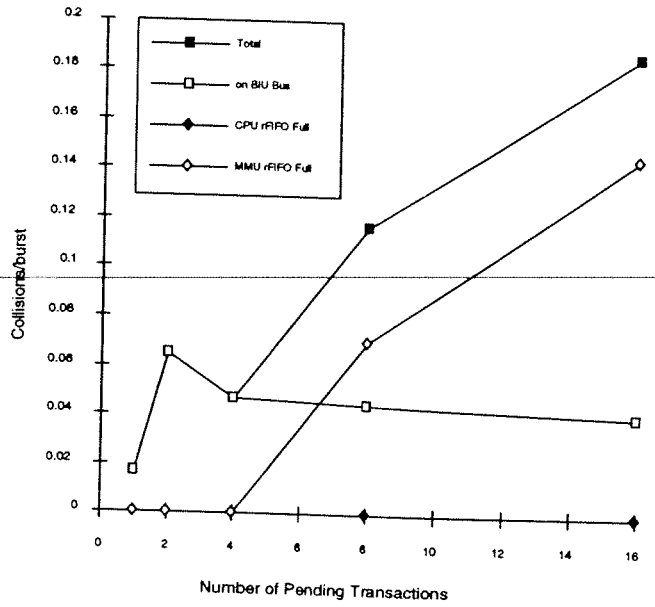

Figure 8: Collisions/burst for DAXPY.

MMU rFIFO back-pressures dramatically increase with bus load. True arbitration collisions level off as the back-pressure dominates performance.

neither chip's rFIFO affects the collision rate. Because of the limited traffic, the average CPU and MMU burst lengths are 4.0 and 6.5 fifo entries in length respectively. In fact, because of the limited traffic and small burst length, the FIFO controllers do not fall into the favorable ping-pong behavior and a significant peak in true collisions is seen when the number of outstanding transactions is 2. The desirable ping-pong effect between the two chips does not take effect until 4 pending transactions are allowed, and the average burst size increases to 4.8 for the CPU, and 7.5 for the MMU.

True collision recovery has a cost of 3 BIU cycles wasted before the collision occurred and 7 BIU cycles to turn the bus over to the MMU. These cycles are regarded as arbitration overhead brought on by the collision. In this region, the arbitration overhead reaches about 0.65 BIU cycles/burst at 2 pending transactions, and falls to 0.47 BIU cycles/burst at 4 pending transactions.

From 8 to 16 pending transactions, the effects of the full MMU rFIFO back-pressuring the system are seen. True collisions drop slightly and level off at about 0.04 collisions/burst with an arbitration cost of 0.4 BIU cycles/burst. However, MMU rFIFO back-pressures increase dramatically from 0 collisions/burst at 4 pending transactions to 0.15 collisions/burst. CPU rFIFO back-pressures remain at 0. In this region, bandwidth remains high but latency strongly increases.

The utilization of the four BIU FIFOs is shown in Figure 9. The MMU rFIFO is rarely empty after 8 pending transactions are allowed. This correlates with the collisions/burst graph showing that this rFIFO was often full and back-pressuring the bus as a strong function of number of pending transactions. It follows that the CPU tFIFO is full 43 % of the time, no new transactions are initiated, and the CPU is stalled. The MMU tFIFO, and the CPU rFIFO are not under such stress, though they are increasingly utilized.

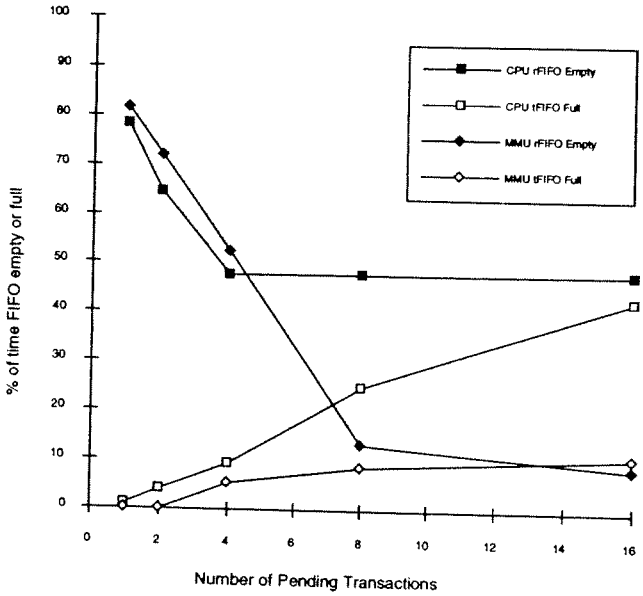

Figure 9: FIFO Utilization for DAXPY.

The MMU rFIFO is seldom empty and back-pressures the system.

At this point, the reasons for the slight decrease in average bandwidth seen in Figure 6 become clear. When the MMU rFIFO fills and back-pressures the bus, the CPU relinquishes the bus and the MMU gains bus ownership and stalls until it has something to return to the CPU. If the CPU tFIFO is full, or if the maximum number of pending transactions has been reached, the CPU stalls. Of course this back-pressure strongly suppresses bandwidth.

In addition to the lack of bus activity in this circumstance, the BIU system falls out of the favorable ping-pong behavior. This limits the burst length to less than the size of a tFIFO. In this instance, the BIU will send back the first complete transfer that is

in the MMU tFIFO, i.e., a single transfer of 5 MMU tFIFO entries (1 Command/Address, 4 Data pairs).

Because the MMU rFIFO has a dominant damping effect on total bandwidth in this configuration, we do not see any Ethernet-like effect where a high frequency of true contention collisions causes a precipitous bandwidth drop-off when the bus is under heavy load. In fact, the bus has not approached the limits of its arbitration mechanism, it has reached the D-stream bandwidth performance limits of the behavioral MMU model. Increasing the MMU rFIFO size might filter out some back-pressures during peak request periods, but the MMU D-stream controller alone can not keep pace when more than 4 outstanding CPU D-stream requests are allowed at an issue rate of 2 CPU cycles.

5 Conclusions

Bandwidth scales with clock speed, and is regarded as an architectural resource that can be applied to latency reduction. Several architectural innovations intended to reduce access latency and improve overall throughput require increased system bandwidth.

Using realistic bus workloads, sustained bus bandwidths of over 1.6 Gbyte/s have been simulated on an HDL model of a CPU to MMU bus having a peak bandwidth of 3.2 Gbyte/s. This bandwidth is sufficient to support our 300 MHz GaAs chipset. However, at very high bus loads, bandwidth remains high but latency increases become intolerable.

The distributed collision-based bus arbitration is not the limiting performance effect. The bus falls into a favorable ping-pong pattern with alternating chips transmitting a full tFIFO to an available rFIFO. When this occurs, the arbitration overhead is between 0.4 and 0.7 cycles/burst. At high bus loads, the MMU rFIFO back-pressures dominate performance.

Predictably, latency tolerance costs additional latency in this highly partitioned, FIFO-based system. Communication latency attributed to the BIU FIFOs is as low as 18 cycles for light bus loads, but climbs rapidly under heavy bus loads. This directly affects secondary cache latency and might not exist if the CPU and MMU were implemented in a technology that facilitated a single-chip implementation.

The performance results gained by using HDLs as modeling languages are extremely accurate - the system under test captures the true dynamic complexity of the microarchitecture. The results of these low-level simulations provide a justified cost basis for higher level architectural simulations. Hardware description languages coupled with compilation and synthesis tools, allow design exploration from the RTL level through physical implementation from a high-level of abstraction. These are powerful microarchitectural design capabilities.

References

[1] J. L. Hennessy and N. P. Jouppi, "Computer technology and architecture: An evolving interaction," *IEEE Computer*, vol. 24, pp. 18–29, September 1991.

[2] D. Dobberpuhl, et al., "A 200 MHz 64b dual-issue CMOS microprocessor," in *1992 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 254–256. IEEE, Feb 1992.

[3] N. Jouppi, et al., "A 300 MHz 115W 32b bipolar ECL microprocessor with on-chip caches," in *1993 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 84–85. IEEE, Feb 1993.

[4] M. Upton, T. Huff, P. Sherhart, P. Barker, R. McVay, T. Stanley, R. Brown, R. Lomax, T. Mudge, and K. Sakallah, "A 160,000-transistor GaAs microprocessor," in *1993 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 92–93. IEEE, Feb 1993.

[5] A. Smith, "Sequential program prefetching in memory hierarchies," *IEEE Computer*, vol. 11, no. 12, pp. 7–21, December 1978.

[6] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proceedings of the* 17^{th} *Annual International Symposium on Computer Architecture*, pp. 364–373, New York NY (USA), May 1990, IEEE.

[7] D. Callahan, K. Kennedy, and D. Porterfield, "Software prefetching," in *Proceedings of the* 4^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 40–52. ACM, 1991.

[8] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proceedings of the* 8^{th} *Annual International Symposium on Computer Architecture*, pp. 81–87, 1981.

[9] G. Kane and J. Heinrich, *MIPS RISC Architecture*, Prentice Hall, 1992.

[10] M. Johnson, *Superscalar Microprocessor Design*, Prentice-Hall Inc., 1991.

[11] N. Kushiyama, et al., "A 500-megabyte/s data-rate 4.5m DRAM," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 4, pp. 490–498, April 1993.