An FPGA Approach to Quantifying Coherence Traffic Efficiency on Multiprocessor Systems

Taeweon Suh
Platform Validation Architecture
Intel Corporation
taeweon.suh@intel.com

Shih-Lien Lu
Microarchitecture Research
Intel Labs
shih-lien.l.lu@intel.com

Hsien-Hsin S. Lee
Electrical and Computer Engineering
Georgia Tech
leehs@gatech.edu

ABSTRACT

Recently, there is a surge of interests in using FPGAs for computer architecture research including applications from emulating and analyzing a new platform to accelerating microarchitecural simulation speed for design space exploration. This paper proposes and demonstrates a novel usage of FPGAs for measuring the efficiency of coherent traffic of an actual computer system. Our approach employs an FPGA acting as a bus agent, interacting with a real CPU in a dual processor system to measure the intrinsic delay of coherence traffic. This technique eliminates non-deterministic factors in the measurement, such as the arbitration delay and stall in the pipelined bus. It completely isolates the impact of pure coherence traffic delay on system performance while executing workloads natively. Our experiments show that the overall execution time of the benchmark programs on a system with coherence traffic was actually increased over one without coherent traffic. It indicates that cache-to-cache transfers are less efficient in an Intel-based server system, and there exists room for further improvement such as the inclusion of the *O* state and cache line buffers in the memory controller.

1. INTRODUCTION

To maintain data consistency in multi-threaded applications, a symmetric multiprocessor (SMP) system often employs a cache coherence protocol to communicate among processors. The performance of the cache coherence protocols were previously evaluated in several literatures [1, 3, 7, 8]. Traditionally, the evaluations of coherence protocols focused on protocols themselves, and the system-wide performance impact of coherence protocols has not been explicitly investigated using off-the-shelf machines. When workloads are parallelized and run natively on an SMP system, the speedup is dependent on three factors: (1) how efficiently workloads are parallelized, (2) how much communication is involved among processors, and (3) how efficiently the communication mechanism manages communication traffic, e.g., cache-to-cache transfer between processors. While programmers make every effort to efficiently parallelize workloads, the underlying communication mechanism of the architectural implementation remains unmanageable in the software layer and becomes the limiting factor for speedup as the number of processors increases. Despite the importance of the communication, it has not been easy or feasible to completely isolate its impact directly from the speedup numbers collected on an SMP system. Oftentimes, due to the difficulty of the direct evaluation on real machines, software simulators [2, 4, 6, 9] were used to characterize performance of an SMP. Nevertheless, the software-based simulation is sometimes difficult to reach an unbiased conclusion since the exact real-world modeling such as I/O is difficult. In addition, it hinders the broad range measurement of the system behavior due to the intolerable simulation time.

By executing workloads on an off-the-shelf system natively, our case study evaluates and analyzes the impact of communication mechanism on the overall system performance based on a simplified model. Notably, our methodology completely isolates the impact of coherence traffic on the overall system performance. Also note that, our methodology did not attempt to *accurately* model the actual coherence traffic of a given parallel application. Rather, the objective is to analyze and understand how the inter-processor traffic itself, such as cache-to-cache transfer and invalidation traffic, affects the overall performance based on coherence traffic emulated with an implemented cache in a field-programmable gate arrays (FPGAs). In this work, we implemented coherent caches in the FPGA, then measured and evaluated the coherence traffic efficiency of the Pentium®-III's[1] (referred to as P-III hereafter) MESI protocol on the front-side bus (FSB). We then execute workloads directly on the Intel server system to obtain, measure and analyze a broad, precise spectrum of the system behavior. We found that coherence traffic on the P-III FSB is not as efficient as expected. The overall execution time of the benchmark programs with coherence traffic was actually increased compared to one without them.

2. BACKGROUND

Cache coherence protocols are widely used in shared-memory MP systems for maintaining data consistency. Depending on the scale of a shared-memory MP system, different protocols are employed. Large-scale systems that adopt the distributed shared memory architecture use a directory-based coherence scheme [3]. On the other hand, modest-sized MP systems based on a shared-bus architecture typically use a snoop-based coherence protocol [3]. Most of the commercial servers and high-end PC systems employ the snoop-based coherence protocols. Our work measures and evaluates the coherence traffic efficiency for such systems.

2.1 Coherence mechanism on the P-III FSB

The P-III-based SMP systems utilizes the FSB as a shared bus for communication. The FSB is a 7-stage pipelined bus, consisting of request1, request2, error1, error2, snoop, response, and data phases, as illustrated in Figure 1. Response and data phases are often overlapped. The FSB supports eight outstanding transactions. For cache coherence,

[1]Pentium® is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

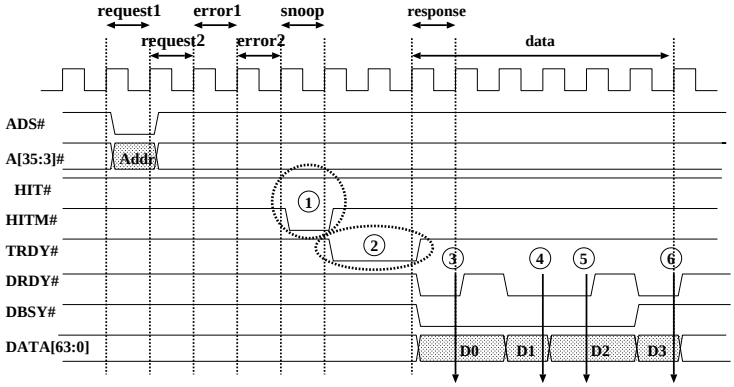


Figure 1: Simplified timing diagram of cache-to-cache transfer on the FSB

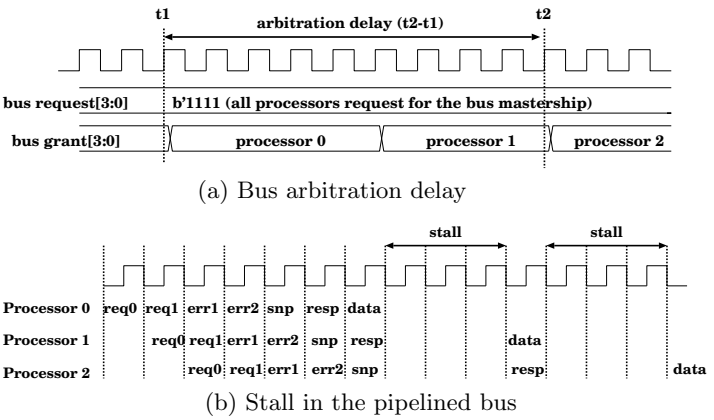
FSB data traffic type	State change in requesting processor	State change in remote processor	FSB signal assertion		Cache-to-cache transfer
			HIT#	HITM#	
Read	**I → S**	**M → S**		√	√
		E,S → S	√		
	I → E	**I → I**			
Write	**I → M**	**M → I**		√	√
		E,S,I → I			

Table 1: Cache line state changes and snoop phase bus signal behavior on the P-III FSB

the P-III uses the MESI protocol. Two active-low bus signals (HIT# and HITM#) are dedicated for the snooping purpose. The HIT# assertion indicates that one or more processors have the requested line in one of the clean states (*E* or *S*). The HITM# is asserted when the remote processor has the requested line in the *M* state, as depicted in ① of Figure 1. The snoop result of every memory transaction is driven in the snoop phase of the pipeline. Depending on the cache line status of the remote processors, the bus signal behavior and state transitions are different as shown in Table 2. Table 2 assumes that an FSB transaction hits the snooping processor's cache whose line is in the *M*, *E*, or *S* state. There are three kinds of coherence traffic on the P-III FSB

- Cache-to-cache transfer
- Read-for-ownership
- Invalidation traffic for upgrade miss

In the P-III, cache-to-cache transfer is limited to snoop-hits on the *M* state line. As shown in Table 2, it occurs with HITM# asserted, when memory read or write transactions hit on the *M* state line. Because of the lack of the *O* state, the P-III updates main memory simultaneously when cache-to-cache transfer occurs. To update memory, the memory controller (MC) should be ready to accept data. It represents its readiness by asserting the TRDY# (target ready) FSB signal, as depicted in ② of Figure 1. Afterwards, eight words (32B) of data are transferred as shown in ③, ④, ⑤, and ⑥. The number of bus cycles taken depends on the readiness of data from the processor or the main memory. Figure 1 shows six cycles to transfer one cache line. The read-for-ownership transaction takes place when a write operation misses its own cache. It generates a *full-line memory read with invalidation* on the P-III FSB. The upgrade miss takes place when a write operation hits on the *S* state line in its own cache. It initiates *0-byte memory read with invalidation* on the P-III FSB. All remote processors' caches, which may have the same line in the *S* state, invalidate the corresponding cache lines when invalidation traffic is observed.



(a) Bus arbitration delay

(b) Stall in the pipelined bus

Figure 2: Non-deterministic factors in SMP systems

3. METHODOLOGY

We first introduce the issues in measuring the intrinsic delay of coherence traffic with MP configurations. Then we present our methodology, and justify that our experimental configuration, i.e. one processor and an FPGA, is one feasible way of measuring and evaluating the intrinsic delay of coherence traffic.

3.1 Shortcomings in MP Environment

The pipelined bus architecture in the MP systems makes it difficult to measure communication efficiency because of the arbitration delay and the pipeline stall incurred by multiple outstanding requests from processors. We now discuss the problems using a four-way SMP system as an example.

3.1.1 Bus Arbitration Delay

In shared-bus-based MP systems, an arbiter mediates the bus mastership one at a time. Typically, a priority-based or round-robin-based arbitration is used to grant bus accesses for processors. Therefore, a processor with a low-priority or a processor recently accessed the bus will have to wait until its next turn to access the bus again. In Figure 2(a), we assume that all four processors are requesting for the bus mastership at time *t1* and the processor 2's transaction incurs cache-to-cache transfer. Processor 0 is first granted for the bus access, followed by Processor 1. Processor 2 is granted for the bus mastership at time *t2*. This arbitration delay elongates processor 2's transaction by *t2* - *t1*, causing an effectively longer cache-to-cache transfer time. Such a non-deterministic arbitration delay highly depends on how the workloads are parallelized and when cache misses occur.

3.1.2 Stall in Pipelined Bus

Modern bus protocols provide pipeline to increase the overall throughput. For example, the P-III FSB has a 7-stage pipeline, as illustrated in Figure 1. Although the pipelined bus increases the throughput, it also incurs the non-deterministic behavior of data transfer for bus transactions. For example, suppose that there are three outstanding transactions from different processors on the FSB as shown in Figure 2(b). We assume again that the processor 2's transaction incurs cache-to-cache transfer. Processor 0 finishes its transaction in the shortest time because there are

no previous transactions. However, processor 1's transaction is stalled by three cycles, as illustrated in Figure 2(b), because its snoop phase is overlapped with the data phase of the previous transaction. Note that the data phase requires at least four bus cycles to transfer one cache line (32B). Even worse, the processor 2's transaction is stalled by six cycles because of the overlaps with the processor 0 and processor 1's transactions. This six-cycle delay is effectively reflected as part of the cache-to-cache transfer time. The stall in the pipelined bus is also non-deterministic because it again depends on how workloads are parallelized and consequently how processors manage cache misses.

3.1.3 Discussion

Measuring and evaluating the intrinsic delay of coherence traffic requires eliminating non-deterministic factors such as arbitration delay and stalls in the pipelined bus. Unfortunately, these problems persist as long as parallel workloads are running on an MP system. In the next section, we introduce our FPGA-based methodology, which is capable of eliminating these interferences and isolating the impact of coherence traffic on system performance. Note that, our methodology did not attempt to *accurately* model the coherence traffic of a given parallel workload running on a shared-memory MP machine. Rather, our goal is aimed to analyze and understand how the inter-processor traffic itself (cache-to-cache transfer and invalidation traffic) affects the overall performance based on coherence traffic emulated by the use of an FPGA.

3.2 Proposed Methodology

We use an Intel dual processor system, which features two P-III[2] processors on the FSB. To remove the non-deterministic factors, one P-III was replaced with an FPGA board and a cache is implemented in the FPGA. From the P-III's point of view, the FPGA functions like a virtual processor with a cache. Then, our strategy to measure the efficiency of coherence traffic is shown in Figure 3. Whenever the P-III evicts a modified cache line to the main memory, the FPGA seizes the line from the bus and saves it into the cache implemented in Block RAM (BRAM). This is shown in ① of Figure 3. When the P-III requests the same line later, the FPGA indicates a snoop-hit by asserting HITM# and provides the requested line through cache-to-cache transfer, as shown in ② of Figure 3. In other words, the FPGA is helping the P-III to run workloads by supplying data via cache-to-cache transfer. By comparing execution times with and without the FPGA, we can evaluate the effectiveness of the intrinsic delay of coherence traffic.

This configuration *completely eliminates* the bus arbitration delay because only one P-III is requesting for the bus mastership. In other words, the FSB is always granted for the remaining P-III. The pipeline stalls incurred by multiple processors' requests are also completely eliminated because again only one processor is monopolizing the bus. Even though one processor may initiate multiple transactions on the bus, it does not disturb our measurement because the same behavior will take place in our *baseline*. The baseline is to measure the execution times of benchmark programs on one P-III without the FPGA. Consequently, this evaluation scheme is able to isolate the impact of the intrinsic delay of coherence traffic on system performance, and enables its efficiency evaluation by measuring and comparing the execution times of the benchmark programs.

[2]Note that the P-III used contains an 8KB L1 and a 256KB L2 cache, each with a line size of 32 bytes.

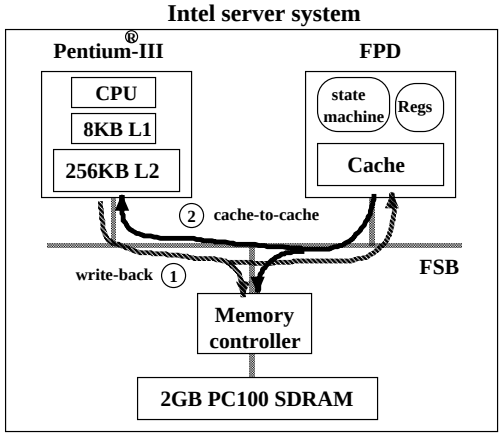


Figure 3: Proposed evaluation methodology.

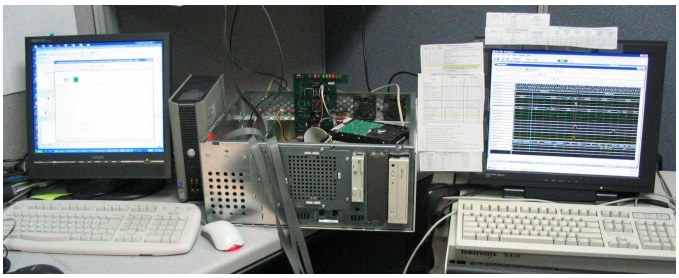
In this configuration, three types of coherence traffic are generated on the P-III FSB, as if there are two P-IIIs in the system. First, cache-to-cache transfer is generated when the FPGA finds the requested block in its own cache. Second, the read-for-ownership transaction, which is known as the *full-line (32B) memory read with invalidation* on the P-III FSB, is generated upon a write miss in the P-III. The requested block is supplied either from a cache-to-cache transfer (if found in the FPGA) or from the main memory (if missing the FPGA). Third, invalidation traffic, which is known as the *0-byte memory read with invalidation* on the FSB, is generated by a P-III's write to an S state line.

By changing the cache size in the FPGA and measuring native execution times of workloads, we study the sensitivity of system performance on the intrinsic delay of coherence traffic. In this experiment, the more the P-III evicts replaced cache lines onto the FSB, there are better chances for the FPGA to incur coherence traffic, resulting in a more accurate evaluation of coherence traffic efficiency. Therefore, as long as the reasonable number of evictions occurs, the selection of the benchmark programs running on the P-III does not make any difference in the evaluation.

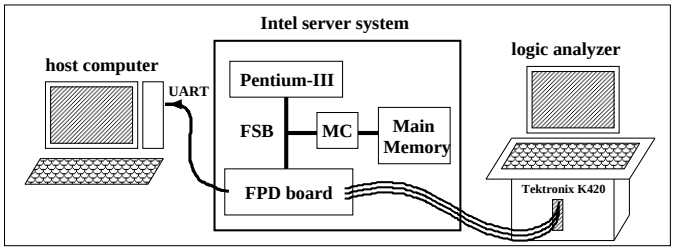
4. EXPERIMENT INFRASTRUCTURE

Figure 4 shows the equipment setup for our experiments. There are three major components — an Intel server system, a host computer, and a logic analyzer. The Intel server system originally features two P-III processors. For this work, one processor was replaced with the FPGA board as depicted in Figure 4(b). Therefore, the P-III and the FPGA board are connected through the FSB, and the MC intermediates the main memory accesses. The FPGA board contains a Xilinx' Virtex-II (XC2V6000) [10], logic analyzer ports, and LEDs for debugging purposes. Each FSB signal is mapped to one Virtex-II pin. The FSB operates at 66MHz while the P-III runs at 500MHz. The host computer is used for synthesizing our hardware design and programming the FPGA with the generated bitstreams. It also collects statistics from the FPGA board, which sends the number of events occurred every second through UART for post-processing. The logic analyzer (K420) from Tektronix is used for debugging our hardware design. It is connected onto the FPGA board to probe the FSB and internal hardware signals.

5. HARDWARE DESIGN



(a) Equipment picture



(b) Equipment schematic

Figure 4: Experiment equipment

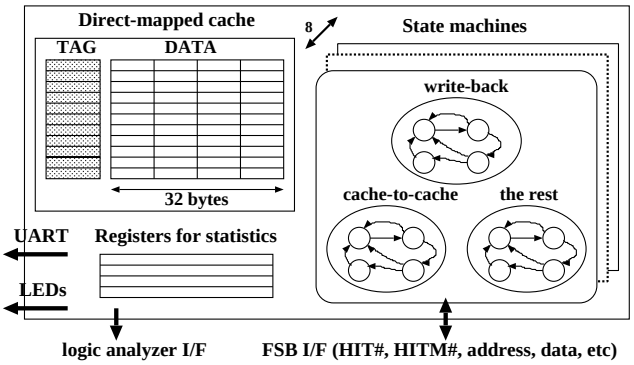


Figure 5: Hardware design schematic in Virtex-II

Table 2: Metrics for efficiency evaluation.

Metric	Unit
Number of cache-to-cache transfers	#/sec
Number of increased invalidation traffic	#/sec
Hit rate of coherence caches in the FPGA	Percent (%)
Execution time difference compared to baseline	sec

Figure 5 demonstrates the hardware schematic designed in the Virtex-II FPGA. It consists of several state machines, a direct-mapped cache, statistics registers, and the FSB interface. Now we detail each component designed in the FPGA.

State Machine: The main state machines keep track of all bus transactions on the FSB and manage all internal and external operations. As shown in Figure 5, it is composed of three paths to perform the followings operations:

- To seize evicted cache lines from the FSB and store those into the cache implemented in the FPGA.
- To initiate a cache-to-cache transfer when the requested block is found in the FPGA's cache.
- The rest, which follows all other transactions on the FSB, including code read, I/O transactions, etc.

Since the P-III allows up to eight outstanding transactions on the bus, the FPGA should be able to track all eight transactions concurrently. Thus, the same state machine is instantiated eight times as shown in Figure 5.

Cache: To retain evicted cache lines, we implemented a direct-mapped cache. For the experiments, several versions varying from 1KB (32 cache lines) to 256KB (8192 cache lines) were designed. The cache's TAG, data, and valid bits were implemented with the dual-port BRAM inside the FPGA. One port is configured as the read port, and the other one is configured as the write port. In our implementation, the critical path is from the TAG lookup to driving the snoop result on the bus. All FSB signals are latched inside the FPGA prior to their use. Even though it exacerbated the timing budget, it is an inevitable choice for stable data processing.

Statistics Registers: Statistics registers were implemented to keep track of statistics such as the number of cache-to-cache transfers, invalidation traffic, cache line evictions, and data read transactions on the FSB. Whenever those events occur, the appropriate counter is incremented, and is reset to zero after sending it to the host computer. The statistics is sent every second to the host computer via UART. The UART is configured with 9600 baudrate.

FSB interface: As explained in the cache design, the FSB signals are latched before being processed. The state machines change states depending on the latched FSB signals. Especially, when the state machine goes through the "cache-to-cache transfer" path, the FPGA actively participates in the bus transactions. Cache-to-cache transfer involves assertion of several FSB signals by the FPGA such as HITM#, 64-bit data bus, Data Ready (DRDY#), and Data Busy (DBSY#). The DRDY# and DBSY# signals are used to inform the right time to latch data by the P-III and/or by the MC. Figure 1 illustrates the usage of those signals. The DBSY# signal is kept asserted until all 4 quadwords are transferred. Then, when the DRDY# signal is asserted, data is available as indicated ③, ④, ⑤ in Figure 1. The last data is available when the DRDY# signal is asserted and the DBSY# signal is de-asserted, as shown in ⑥ of Figure 1.

6. COHERENCE TRAFFIC EVALUATION

To measure and analyze the coherence traffic efficiency, we ran the SPEC2000 integer benchmark suite natively under Redhat Linux 2.4.20-8 on the remaining P-III. Eight benchmark programs from SPECint2000 were executed for 5 times. Then, the average is calculated from the statistics gathered. By changing the cache size from 1KB to 256KB in the FPGA, we report and analyze the behavior of coherence traffic. The *baseline* system has a single P-III without the FPGA. As a result, all the memory transactions initiated by the P-III are serviced from the main memory. In other words, cache-to-cache transfers and associated invalidation traffic never occur in the baseline. As discussed in Section 3, because of the nature of the experiment methodology, the benchmark selection does not affect our evaluation as long as the reasonable number of eviction transactions is generated on the bus. Table 2 summarizes the metrics used to report measured coherence traffic and evaluate its efficiency.

6.1 Cache-to-cache transfer

Figure 6 shows the average cache-to-cache transfers occurred every second, and Figure 7 shows the hit rates in the FPGA's caches. The hit rate is calculated based on Eq (1) that denotes the number of times the FPGA can supply data when the P-III requests memory blocks.

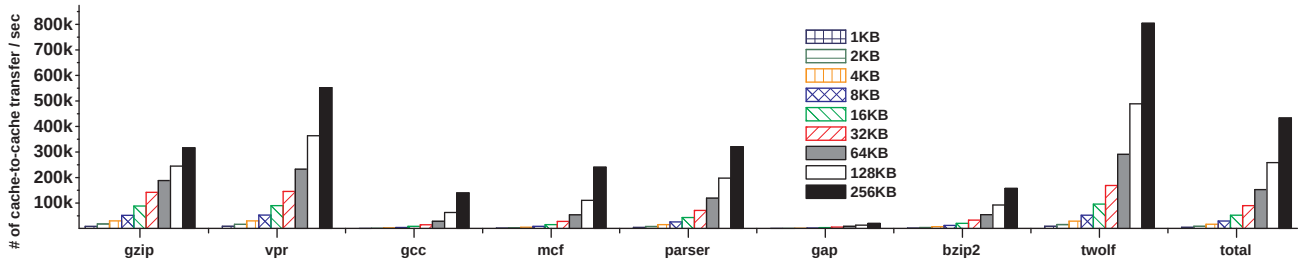


Figure 6: Average cache-to-cache transfer per second

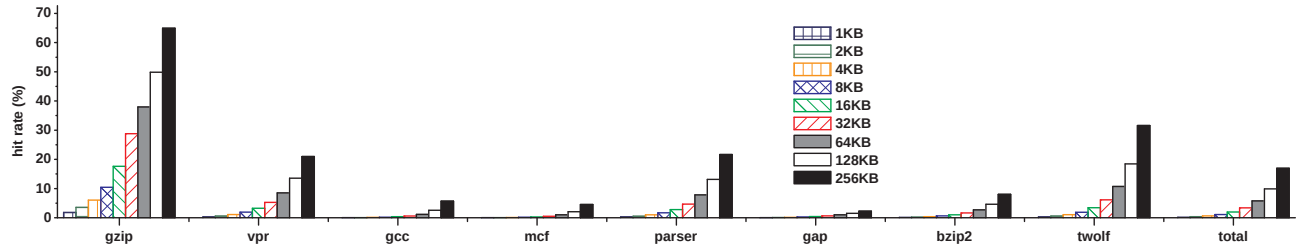


Figure 7: Hit rate of coherence caches in the FPGA

$$hit\ rate\ (\%) = \frac{\#\ cache\text{-}to\text{-}cache\ tranfer}{\#\ data\ read\ (full\ cache\ line)\ on\ the\ FSB} \times 100 \quad (1)$$

As the cache size in the FPGA increases, the number of cache-to-cache transfers for all the benchmark programs also increases. With a 256KB cache in the FPGA, twolf shows the highest transfer frequency (804.2K/sec), whereas gap shows the lowest (20.2K/sec). With a 256KB cache, the hit rate can go as high as 64.89% in gzip and as low as 2.24% in gap. On average, as the cache size increases, the overall cache-to-cache transfer increases from 5.4K/sec to 433.3K/sec, and the hit rate varies from 0.2% to 16.9%. Memory-bound programs do not necessarily show the highest cache-to-cache transfer frequency because they might not use the evicted cache lines later and/or because conflict or capacity misses occur in the FPGA's cache. For example, mcf shows a relatively smaller number of cache-to-cache transfers. The low hit rate 4.5% even with a 256KB cache indicates that most of data requests in mcf were serviced from main memory.

6.2 Invalidation traffic

Figure 8 shows the "increased" amount of invalidation traffic per second, compared to the baseline. With a 256KB cache in the FPGA, twolf again shows the highest peak (306.8K/sec). On average, as the cache size increases, overall invalidation traffic increases from 1.7K/sec to 157.5K/sec. As explained in Section 2.1, Invalidation traffic is incurred by two scenarios: ⓐ *0-byte memory read with invalidation*, ⓑ *full-line (32B) memory read with invalidation*. Figure 8 includes both traffic even though we observed that type ⓐ accounts for more than 99% of the activity. This indicates that the SPECint2000 benchmark programs read data first and subsequently write to the same cache line, generating type ⓐ traffic.

In general, Figure 8 shows a similar pattern to the average cache-to-cache transfers shown in Figure 6. This is explained as follows. When a memory read hits the cache in the FPGA, the FPGA initiates a cache-to-cache transfer to supply data, causing the $I \rightarrow S$ transition in the line of the P-III's cache. A subsequent write by the P-III to the same cache line generates type ⓐ traffic because of an upgrade miss, as the cache line is in the S state. As measured, SPECint2000 benchmark programs tend to read data first and subsequently write to the same line. Therefore, the more cache-to-cache transfer occurs, the more likely invalidation traffic is to be generated.

The baseline system also generates invalidation traffic even though cache-to-cache transfer never occurs. This is due to cache flush instructions. When a page fault occurs, the P-III internally executes a cache flush instruction, which in turn, appears on the FSB as invalidation traffic. Depending on the Linux system services running in the background, the amount of invalidation traffic varies over time. In Figure 8, invalidation traffic sometimes decreases in the cases when the cache size in the FPGA is small enough, e.g. 1KB or 2KB. With small caches, the hit rate and the corresponding frequency of cache-to-cache transfer decrease significantly. For this reason, overall invalidation traffic is more sensitive to the system noise generated by the Linux system services. Especially, gcc is rather susceptible to the Linux system perturbation. The page faults caused by the large number of *malloc()* calls in gcc induce inconsistent patterns.

6.3 Execution time

Figure 9 shows the increase of the overall execution time as the cache size increases. It shows all the collected data from five runs. On average, the total execution of the baseline takes 5,635 seconds (93.9 minutes). As shown in Figure 9, the execution time increases compared to the base-

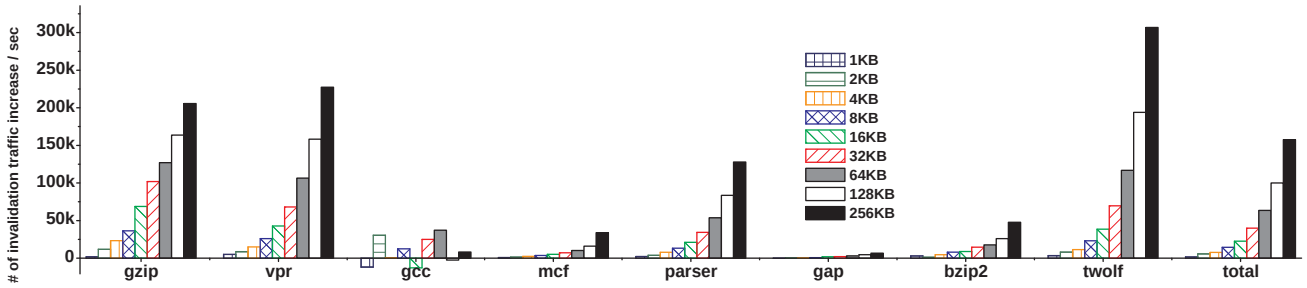


Figure 8: Average increase of invalidation traffic per second (99% type @ traffic)

line, as the number of coherence traffic increases. In other words, the benchmark execution assisted by coherence traffic is more time-consuming than the one without it. With a 256KB cache, the execution time increased up to 191 seconds. There are two reasons that cause the inefficiency of coherence traffic. First, as explained in Section 2.1, main memory is simultaneously updated with each cache-to-cache transfer. This means that even when the P-III is ready to drive the FSB for the data transfer, it has to wait until the MC is ready to accept data. Because of the busy schedule of the pipelined FSB, the MC would not promptly respond to cache-to-cache transfer requests. The second reason comes from invalidation traffic. As explained in Section 6.2, the increase of invalidation traffic follows a similar pattern to the number of cache-to-cache transfers. With a 256KB cache, overall invalidation traffic increased by 157.5K/sec on average. Even though such invalidation involves no data transfer, it still takes in-negligible amount of time since one FSB slot is needed for each invalidation.

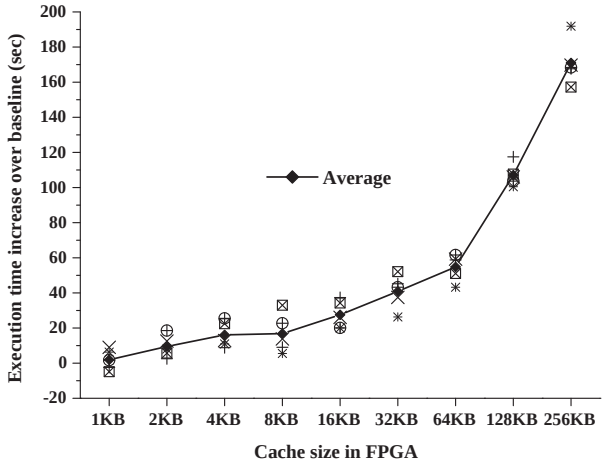


Figure 9: Execution time increase (5 runs, baseline = 5635 sec)

6.4 Intrinsic Delay Estimation

The pipelined nature of the FSB makes it difficult to break down the contribution of each coherence traffic to the execution times. In some cases, coherence traffic may be delayed by long snoop and/or data phases of previous transactions when pipelined with other traffic. On the other hand, some coherence traffic might be injected to the FSB when the bus is idle. Therefore, our estimation is done by closely observing the FSB waveforms of coherence traffic in the logic analyzer. By roughly estimating $5 \sim 10$ FSB cycles for each invalidation traffic and $10 \sim 20$ FSB cycles for each cache-to-cache transfer, the time spent for each coherence traffic is calculated using Eq (2).

$$\begin{aligned} &Execution\ time\ (sec) = \\ &\quad (average\ occurrences/sec) \times (total\ execution\ time) \\ &\quad \times (clock\ period/cycle) \times (latency\ for\ each\ traffic) \end{aligned} \tag{2}$$

$$\begin{aligned} &Average\ occurrences/sec\ =\ 157.5K/sec \\ &Total\ execution\ time\ = 5806\ (= 5635 + 171)\ seconds \\ &Clock\ period/cycle\ =\ 15.15\ ns/cycle \\ &Latency\ for\ each\ invalidation\ traffic\ =\ 5\ cycles \end{aligned} \tag{3}$$

For example, the time spent for invalidation traffic with a 5-cycle latency and a 256KB cache in the FPGA is calculated by plugging the numbers of Eq (3) into Eq (2), and Table 3 summarizes estimated times according to different latencies. Even with a 10-cycle latency for each invalidation traffic, it requires only 138 seconds, which is less than the average increase (171 seconds) of the total execution time. Therefore, the difference (33 seconds) in this calculation comes from cache-to-cache transfers unless each invalidation traffic requires more cycles. Based on this calculation, it is not unreasonable to say that cache-to-cache transfer in this Intel sever system, which takes roughly $6.5\% \sim 13\%$ of the total execution time, is not as efficient as expected. In other words, cache-to-cache transfer on the P-III FSB is slower than getting data directly from main memory.[3] Even though clearly shown in Figure 9, this trend would be more conspicuous with a bigger cache size and/or high associativity cache implemented in the FPGA, as more coherence traffic is generated.

6.5 Opportunities for Performance Enhancement

[3] **It does not mean that cache-to-cache transfer is not good. Note that without cache-to-cache transfer, two memory transactions are necessary when a snoop hits on the *M* state line; one for writing back modified block to main memory, the other for reading the same block from memory. Cache-to-cache transfer reduces the number of memory transactions from two to one. Therefore, it clearly has the advantage over non-cache-to-cache transfer.**

Table 3: Times spent for each coherence traffic according to latencies (256KB cache in the FPGA)

	Coherence traffic	
	Invalidation traffic	Cache-to-cache transfer
Latencies	5 $\sim$ 10 cycles	10 $\sim$ 20 cycles
Times spent	69 $\sim$ 138 seconds	381 $\sim$ 762 seconds

Coherence traffic plays an important role in the performance of MP systems. In the P-III FSB, the fact that main memory should be updated simultaneously upon cache-to-cache transfer would be the main reason for the slowdown. The *O* state in the MOESI protocol is specially designed for this purpose. It allows cache-to-cache transfer without updating main memory. However, a processor with the *O* state line is responsible for updating main memory when the line is displaced. With the *O* state, the MC need not represent its readiness like the P-III FSB. Another alternative is to include cache line buffers in the MC. With the buffers, the MC receives cache-to-cache transfer data temporarily before updating main memory. As long as the buffer space is available, the MC is ready to accept data from snoop-hit processors, which would enable the prompt response for faster cache-to-cache transfer. If the MC is designed to have an ability to compare addresses on the FSB and supply data to a processor when hit on the buffered lines, it would further reduce the memory access latency.

Invalidation traffic is also inevitable in MP systems. In the P-III FSB, remote processors inform a master processor of the snoop results in the snoop phase, which is the 5^{th}-stage in the pipeline. Advancing the snoop phase to an earlier stage bear a potential of reducing the latency. It could reduce the effective cache-to-cache transfer latency, too. However, it requires the faster TAG-lookup in data caches. Deep pipelined-bus and faster bus speed would also help relieve the impact of invalidation traffic even though it complicates the hardware to process requests in shorter time and to accommodate more outstanding transactions.

7. RELATED WORK

MemorIES [5] is a passive emulator developed by IBM T.J. Watson, for evaluating large caches and SMP cache coherence protocols for future server systems. The emulation board can be directly plugged into the 6xx bus of RS/6000 SMP server. Therefore, it is able to perform on-line emulation of several cache configurations, structures, and protocols while the system is running real-life workloads, without any slowdown in application execution speed. Even though our proposed method looks similar to the MemorIES, it has a critical difference from the MemorIES. The MemorIES is a passive emulator, whereas our proposal is a active one. Since the MemorIES is passively monitoring 6xx bus transactions, it is not able to inject transactions on the bus such as cache-to-cache transfer and invalidation traffic. Thus, MemorIES cannot emulate a fully-inclusive cache and cannot perform the latency studies.

8. CONCLUSION

In this paper, we measured the intrinsic delay of coherence traffic, and analyzed its efficiency using a novel FPGA approach on a P-III-based server system. The proposed approach eliminates non-deterministic factors in the measurement, such as the arbitration delay and stall in the pipelined bus. Therefore, it completely isolates the impact of coherence traffic on the system performance. Our case study implements coherence caches in the FPGA and shows that the performance of the SPECint2000 benchmark with coherence traffic was actually degraded. The overall execution time was increased by up to 191 seconds over 5635 seconds of the baseline. With a 256KB cache implemented in the FPGA, the cache-to-cache transfer and the invalidation traffic occurred 433.3K/sec and 157.5K/sec respectively, on average. Performance degradation is attributed to the following reasons. First, cache-to-cache transfer in the MESI protocol requires main memory update simultaneously. It often delays cache-to-cache transfer since the MC would not respond promptly for the update requests because of the busy schedule of the pipelined FSB. Second, invalidation traffic also increased in proportion to the number of cache-to-cache transfers. Even though invalidation traffic involves no data transfer, it still takes in-negligible amount of time since one FSB slot is needed for each invalidation.

To reduce the latency of coherence traffic, we discussed architectural possibilities; the inclusion of the *O* state, cache-line buffers in the MC, advancing the snoop phase to an earlier stage, and deep pipelined-bus and faster bus speed. However, all these potential architectural enhancements come at the expense of additional hardware. Thus, thorough investigations are necessary to measure the trade-offs.

9. ACKNOWLEDGMENTS

This research was sponsored in part by the National Science Foundation under award CNS-0325536.

10. REFERENCES

[1] J. Archibald and J.-L. Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4(4):273–298, Nov. 1986.

[2] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented full-system simulation using m5. In *Proceedings of the 6th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, February 2003.

[3] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.

[4] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, Feb. 2002.

[5] A. Nanda, K.-K. Mak, K. Sugavanam, R. K. Sahoo, V. Soundararajan, and T. B. Smith. MemorIES: a programmable, real-time hardware emulation tool for multiprocessor server design. In *Proc. of ASPLOS-9*, pages 37–48, November 2000.

[6] V. Pai, P. Ranganathan, and S. Adve. RSIM reference manual, version 1.0. *IEEE Technical Committee on Computer Architecture Newsletter*, Fall 1997.

[7] K. Petersen and K. Li. An evaluation of multiprocessor cache coherence based on virtual memory support. In *Proceedings of the 8th Int'l Parallel Processing Symposium*, pages 158–164, 1994.

[8] C. A. Prete, G. Prina, and L. Ricciardi. A trace-driven simulator for performance evaluation of cache-based multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):915–929, 1995.

[9] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The SimOS approach. *IEEE Parallel and Distributed Technology*, 3(4):34–43, 1995.

[10] Xilinx. Virtex-II Platform FPGAs. http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_platform_fpgas/index.htm.