Enhanced Secondary Bus Microarchitecture

by

John William O'Farrell

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 6, 2011

Keywords: real-time systems, computer architecture, low-power systems, simulation

Approved by

Abstract

In spite of advances to improving cache efficiency, memory access bottlenecks still prevent processors from executing at full speed. This research evaluates a fundamentally new concept of using a secondary bus, connecting the level-2 cache to memory, for committing cache write-backs. Simulations, using the Sim-Alpha version of the SimpleScalar tool set, demonstrate the feasibility and advantages of such a secondary bus. Based on simulation results, the added secondary bus can decrease queuing delays on the system bus by 6% to 99%, with an average of 87%, when sufficient write-backs are present. Such reduction in queuing delays leads to a decrease in worst-case execution times, and offers superior temporal determinacy in real-time environments. Real-time and near real-time embedded applications that depend on intense graphics processing and movement of large blocks of data, such as printer controllers and medical imaging, are prime candidates for applications of the secondary bus.

Simulations using small cache sizes serve as a basis that verifies that the microarchitecture is viable and that it produces interesting and significant results. Then, updates to large cache sizes comparable to those in current commercial processors and benchmarks taken from the industry-standard SPEC CPU2006 benchmark suite expand and validate the results. Decreases of 31% to 94%, with an average of 82%, in the maximum execution time of any instruction, and of 77% to 100%, with an average of 97%, in the number of instructions requiring more than 1000 cycles to execute point to a decrease in worst-case execution times in real-time systems. The improvements in instructions per cycle point to possible applications in low-power systems, where the clock frequency may be reduced while maintaining constant processing power. In addition to the real-time and low-power system benefits, overall system performance using the secondary bus microarchitecture is improved by up to 33% when I/O is present.

Acknowledgements

I would like to thank Dr. Sanjeev Baskiyar for his tireless encouragement and contribution to this work. He was always inspiring, and helped push me to keep making progress. Thank you also to my committee members, Dr. Kai H. Chang and Dr. James Cross. I appreciated the dedication of these men to the students of my beloved Auburn. I have been blessed with a wonderful family, and this dissertation would not have been possible without the loving help and encouragement of my wife, Joy. I would also like to thank my parents, Joan and Lamar, and my sons Ryan and Tim, for lots of love and moral support.

The simulations required significant time to set up and execute, and I am thankful for Rakshith Venkatesh's help executing the simulations of the SPEC benchmarks. Simulations were carried out together with Rakshith, and the code changes and results were unified. I am also grateful to the Alabama Supercomputing Authority for giving us access to the supercomputer clusters for running the simulations. This research was supported in part by a grant from DARPA/AFRL.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Problem Statement

As the speed of high performance processors continues to increase, there are a number of bottlenecks within computer system architectures that prevent those processors from achieving their full potential. Processor speeds continue to outpace memory access speeds. Because of this, cache memory has long been an important part of any computer system architecture. The faster access time provided by cache memory is essential for allowing high-speed processors to operate with slower and much more cost-efficient main memory. Cache memory is generally connected to main memory via a main system bus. The system bus that connects the cache memory to main memory represents a serious system bottleneck, one that can leave a high performance processor sitting idle while waiting for data to be transferred to and from main memory.

1.2. Research Objective

The secondary bus was proposed [2] and designed [4] to more effectively retire the cache write-back buffer entries to the main memory. The goal of this research is to study, via simulation, the impact and feasibility of a low bandwidth, low cost, non-intrusive secondary bus on the performance of a computer system. The study includes its impact on queuing delays on the main bus and the reduction in main bus access contention between processor and I/O accesses to main memory. Such bottleneck reduction is expected to result in a system with:

1) Increased determinacy, leading to a system with greater real-time performance,
2) Increased overall performance, and
3) Lowered power usage.

1.3. Background

When a cache miss occurs, the processor must delay to wait for data to be retrieved from main memory, placed into the cache, and transferred to the processor. When the miss occurs in a cache that employs a write-back scheme and is already full, a cache line must be evicted before the new data can be retrieved from main memory. When the line that is chosen for eviction is dirty, it must be written to memory before the new line can be read. This leads to several data transfers on the bus connecting the cache to the main memory. The processor must remain idle during those transfers and cannot continue until the needed data is available.

Various techniques have been developed in an attempt to decrease the time spent by the processor waiting for this cache processing to occur, but none has approached the problem from the viewpoint of using an additional, secondary bus for communication between the cache and main memory.

Many current cache configurations include some form of non-blocking mechanism. Miss-status holding registers [14], write buffers [7], and victim buffers [8] are all mechanisms that provide storage where dirty cache lines can be written efficiently without requiring the processor to stall waiting for the writes to complete. The eager write-back technique [15] attempts to address the problem of clustered bus accesses caused by graphics applications. The technique writes dirty cache blocks back to memory before they are evicted, using otherwise idle time on the bus and attempting to avoid clusters of bus usage.

This research represents a unique approach to the problem, and one that can be used to provide improvements in addition to other techniques. This research shows that a secondary bus is feasible, and that use of the bus will increase the performance of computer systems by reducing bus contention.

In addition, the research presents possible implementations for a secondary bus. Areas of investigation include a serial line and wireless communication. A reasonable bandwidth expectation for the secondary is presented, based on potential hardware and protocol support.

1.4. Outline

The remainder of this dissertation is organized as follows: Chapter 2 presents some background in relevant current literature on the subjects of cache performance enhancements, microprocessor simulation, and possible secondary bus implementations. Chapter 3 presents the results of the secondary bus microarchitecture in a small cache system. In Chapter 4, the secondary bus microarchitecture is demonstrated in a large cache system using the standard SPEC CPU2006 benchmarks. Chapter 5 presents the real-time system performance improvements with the serial bus, and Chapter 6 presents the low-power system improvements. Conclusions are presented in Chapter 7.

Chapter 2

Background

Relevant current literature explores possible methods of improving cache performance, microprocessor simulation, and performance measurement. Since wireless communication is a possible implementation of a secondary bus, literature describing wireless protocols and possible hardware support for a wireless bus is also relevant.

2.1. Cache performance enhancements

Cache memory has long been an important part of any computer system architecture. The faster access time supplied by cache memory is essential for allowing high-speed processors to operate with slower and much more cost-efficient main memory. The hierarchy formed by CPU registers, cache memory, main memory, and external disk storage represents an attempt by system designers to improve system performance while controlling costs. Each lower level in the hierarchy consists of slower, larger, and less expensive storage. The system depends on the principal of locality to maintain system performance.

When an entry in a cache is changed, the new value must be transferred into the lower-level memory. Two techniques, called write-back and write-through, may be used. In the write-back technique, when a memory location is written, it is marked as dirty. All entries are held in the cache until a read from the lower-level memory occurs when the cache is full. At that time, an entry that has been marked as dirty is removed, and the contests are written to the lower-level memory. This means that when a cache is full, a read miss in a write-back cache requires one memory accesses to retrieve the needed memory entry, and another access to write replaced data from the cache to the next memory level.

In the write-through technique, when a memory location is written, it is written into the cache and the appropriate lower-level memory location at the same time. This type of caching provides worse performance than write-back, but is simpler to implement and has the advantage of internal consistency, because the cache is never out of sync with the memory. The write-back cache technique has been shown to be superior to write-through for reducing bus usage and memory accesses [11].

Many current cache configurations include some form of non-blocking mechanism. Miss-status holding registers (MSHR) [14], write buffers [7], and victim buffers [8] are all mechanisms that provide storage where dirty cache lines can be written efficiently without requiring the processor to stall waiting for the writes to complete. The purpose of an MSHR is to merge pending cache misses to the same cache block into one transaction, therefore preventing subsequent references that occur during an in-process cache miss from generating additional reads to the memory hierarchy. When the line becomes available based on the initial cache miss, the MSHR responds to all pending loads. The write buffer is meant to hold cache lines that are being written to memory. It can hold a small number of lines, and it allows the processor to continue after a write without waiting for the memory operation to complete. The victim buffer holds a few recently evicted cache lines. If a read misses in the cache, but hits in the victim buffer, the line is moved back into the cache faster than with a normal miss. AMD refers to its write buffer as a victim buffer, so there is some conflict and confusion in terms. The eager write-back technique [15] attempts to address the problem of clustered bus accesses caused by graphics applications. The technique writes dirty cache blocks back to memory before they are evicted, using otherwise idle time on the bus and attempting to avoid clusters of bus usage.

2.2. Microprocessor simulation

The SimpleScalar Tool Set [1] is a widely used tool for microprocessor simulation and architectural modeling. The tool set consists of well-documented, open source tools for detailed modeling of microprocessors, and permits researchers to examine and analyze numerous aspects of processor execution. Using these tools, researchers can experiment with new architectures without incurring the expense and effort of developing new hardware. The base simulation system supports numerous configuration options to control aspects of the processor, execution cycles, and memory. In addition, the open source nature of the tools allows unlimited potential for enhancement by other researchers. The structure of the SimpleScalar simulator is show in Figure 1 [27].

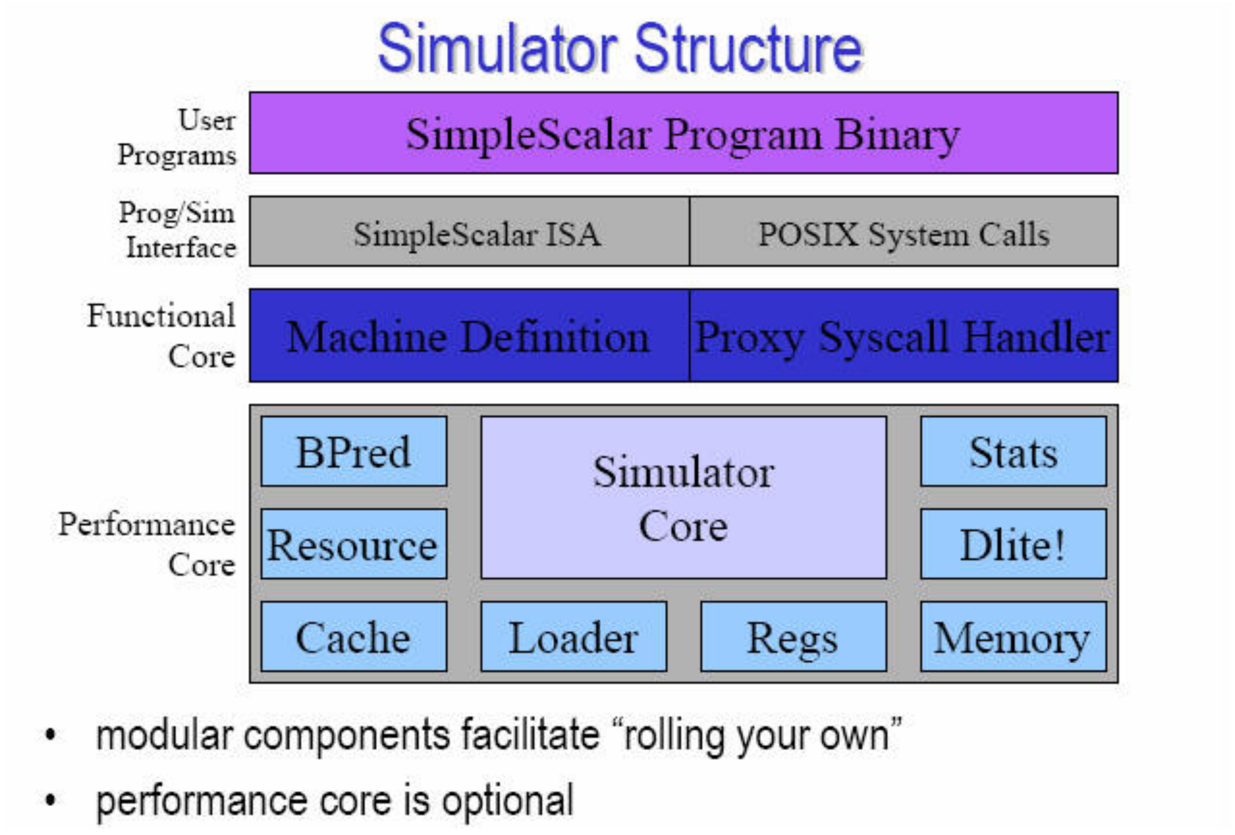


Figure 1: SimpleScalar Simulator Structure

Simplescalar has been verified using four approaches [22]: micro-benchmark validation, in which a number of small programs were executed to test various parts of the machine, correlation with independent simulators, regression correlation with previous simulator releases, and code inspections.

Memory hierarchy extensions [5] introduced into the base SimpleScalar tools support more extensive modeling and analysis of the cache and bus systems within a processor. These extensions support modeling of an arbitrary hierarchy of caches, associated buses, address translation, and translation look-aside buffers. A flexible configuration mechanism permits all aspects of the cache, bus, and memory parameters to be easily modified. The Sim-alpha [9] tools are based on SimpleScalar, and include the memory extensions. Sim-alpha provides an easily expandable simulator of the Alpha 21264 processor [8]. The architecture of the Alpha 21264 is show in Figure 2 [8].

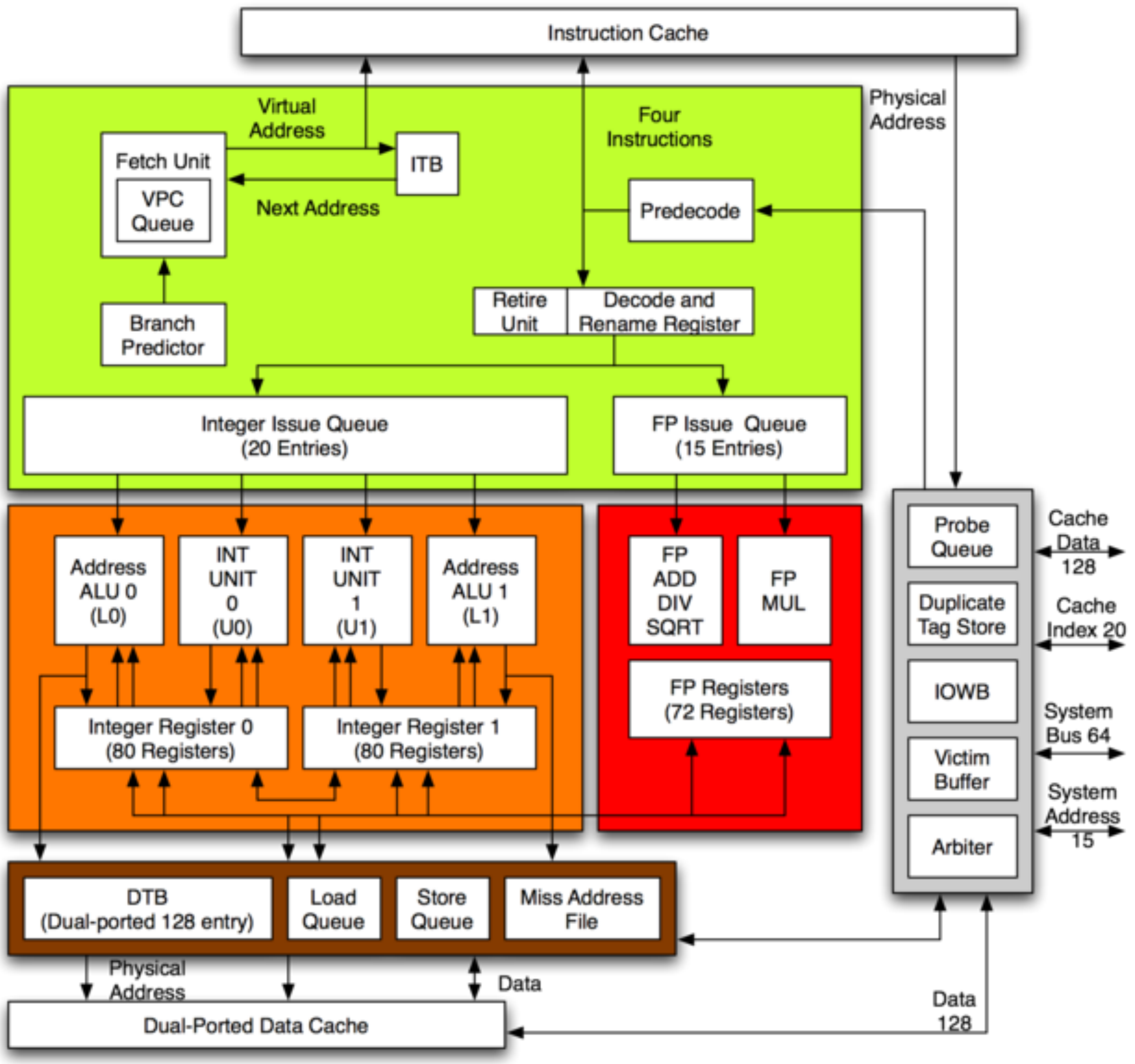
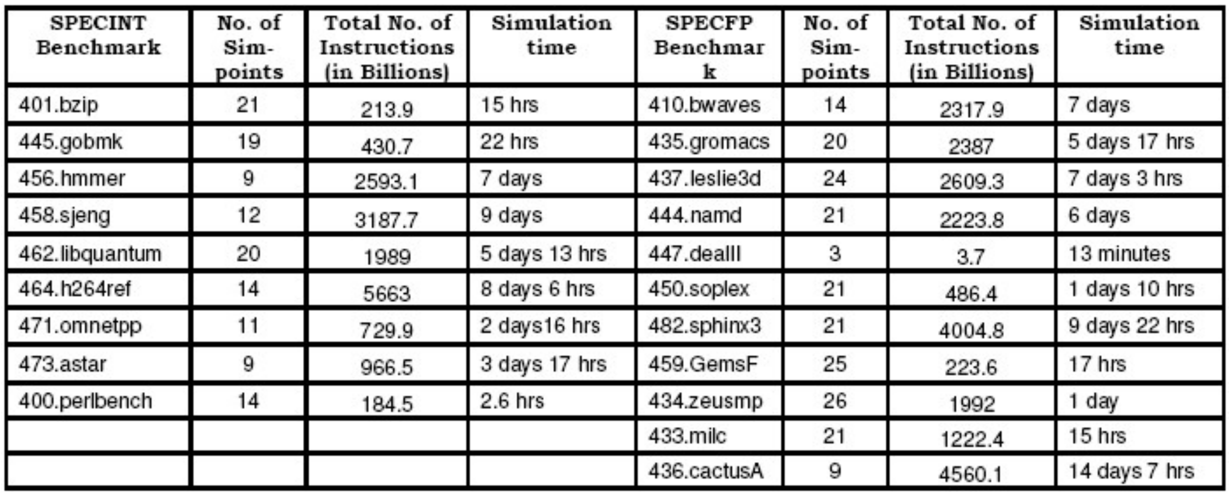


Figure 2: Alpha 21264 Architecture

2.3. Simulation using SPEC CPU2006 Benchmarks

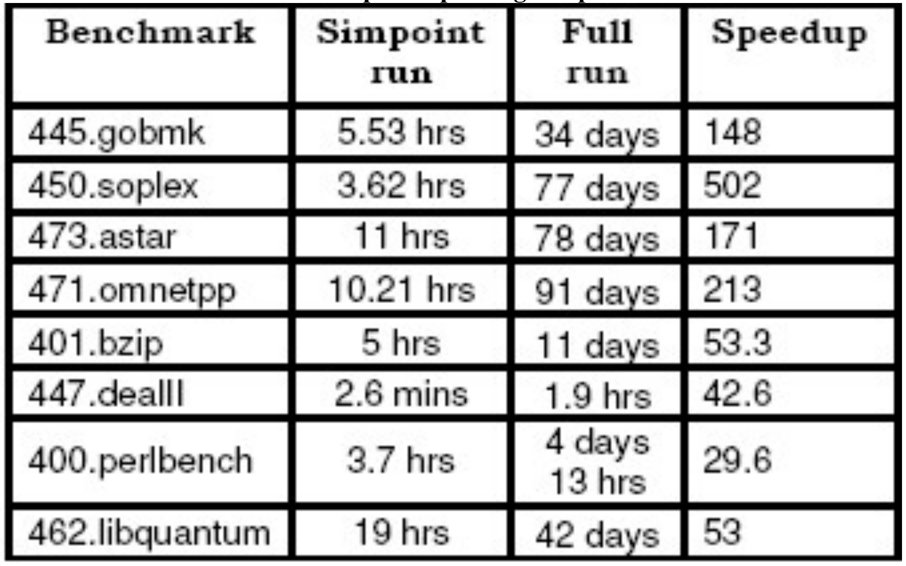
The SPEC CPU2006 benchmark suite [21] is an industry-standard, CPU-intensive group of benchmarks meant to stress a system's processor, memory subsystem, and compiler using workloads developed from real user applications. Simulation using the SPEC CPU2006 benchmarks on a cycle-accurate simulator such as SimpleScalar is difficult because of the time required to execute the simulations. One effective method of dealing with this problem is through the use of Simulation Points [19]. Simpoints permit execution of a representative subset of a program, and can be used to greatly reduce the time required to execute a simulation. Table 1 shows the time required to execute the benchmarks during generation of simpoints for the benchmark suite, and Table 2 shows the speed-up that was obtained by using simpoints [10].

Table 1: Simulation Times of SPEC Benchmarks



SPECINT Benchmark	No. of Sim-points	Total No. of Instructions (in Billions)	Simulation time	SPECFP Benchmark	No. of Sim-points	Total No. of Instructions (in Billions)	Simulation time
401.bzip	21	213.9	15 hrs	410.bwaves	14	2317.9	7 days
445.gobmk	19	430.7	22 hrs	435.gromacs	20	2387	5 days 17 hrs
456.hmmer	9	2593.1	7 days	437.leslie3d	24	2609.3	7 days 3 hrs
458.sjeng	12	3187.7	9 days	444.namd	21	2223.8	6 days
462.libquantum	20	1989	5 days 13 hrs	447.dealII	3	3.7	13 minutes
464.h264ref	14	5663	8 days 6 hrs	450.soplex	21	486.4	1 days 10 hrs
471.omnetpp	11	729.9	2 days16 hrs	482.sphinx3	21	4004.8	9 days 22 hrs
473.astar	9	966.5	3 days 17 hrs	459.GemsF	25	223.6	17 hrs
400.perlbench	14	184.5	2.6 hrs	434.zeusmp	26	1992	1 day
				433.milc	21	1222.4	15 hrs
				436.cactusA	9	4560.1	14 days 7 hrs

Table 2:Speed-up Using Simpoints



Benchmark	Simpoint run	Full run	Speedup
445.gobmk	5.53 hrs	34 days	148
450.soplex	3.62 hrs	77 days	502
473.astar	11 hrs	78 days	171
471.omnetpp	10.21 hrs	91 days	213
401.bzip	5 hrs	11 days	53.3
447.dealII	2.6 mins	1.9 hrs	42.6
400.perlbench	3.7 hrs	4 days 13 hrs	29.6
462.libquantum	19 hrs	42 days	53

The simpoints method has been shown to product valid results. The cycles per instruction measured when using simpoints to simulate the SPEC CPU2006 benchmarks used here was within 5% of the full run values. Measurements of cache miss rates and DTLB miss rates were also within 5%.

2.4. Protocols and hardware for secondary bus implementation

Currently available serial line technologies may provide sufficient bandwidth for a secondary bus. The Inter-Chip USB specification [23] requires full speed operation of 12 Mb/sec, and contains future support for the high-speed mode of 480 Mb/sec. IEEE1394 [12] supports 400 Mb/sec. Proprietary versions of the 802.11g wireless standard support up to 108Mb/sec, and the proposed 802.11n standard includes support for up to 600Mb/sec [24]. Several emerging technologies [20] may also serve as possible implementations. Inter-chip communication [6] creates a miniature wireless LAN on a chip, and may reach speeds of 100Gb/sec. Using inductive coupling between chips, speeds of 1.25Gb/sec have been achieved [16]. Alternatively, bus frequency sharing techniques such as CDMA [13] may allow the secondary bus signal to share the primary bus.

Chapter 3

Analysis of Small Cache Systems with a Secondary Bus

3.1. Introduction

High performance processors use cache memory extensively to compensate for the speed difference between processors and main memory. Numerous advancements have improved the efficiency of cache memory usage, yet bottlenecks remain which prevent processors from executing at full speed. By introducing a secondary bus connecting the cache and main memories, and using that bus for cache write-backs, we may improve a fundamental bottleneck, namely the write-back latency [17]. The advantages of the additional bus are demonstrated using the Sim-alpha version of the SimpleScalar Tool Set. Simulations show that using a secondary bus for cache write-back decreases the queuing delay experienced on the primary bus, therefore increasing the temporal determinacy in real-time environments.

3.2. Background

When a cache miss occurs, the processor must wait for data to be retrieved from main memory, placed into the cache, and transferred to the processor. In a write-back cache, a line must be evicted before this can occur. When the line that is chosen for eviction is dirty, it must be written to memory before the new line can be read. Various techniques have been developed in an attempt to decrease the time spent by the processor waiting for such cache processing to occur.

A secondary bus used for cache write-back addresses the issue of bus contention caused by writes to memory to remove dirty cache lines and reads to retrieve new cache lines. The secondary bus can be used in addition to existing non-blocking mechanisms and along with eager write-back to provide even further performance improvements.

3.3. Secondary bus architecture

A modified version of the Alpha 21264 architecture was created using the Sim-alpha simulator. The base configuration was modified to create an architecture containing a secondary bus. Sim-alpha was chosen as a simulation tool because it contains the memory extensions necessary to accurately model cache and bus activity. The base configuration contained two buses. The onboard bus connected the L1 cache to the L2 cache, and the primary bus connected the L2 cache to main memory. In the modified configuration, an additional, secondary bus was added to connect the L2 cache to main memory for use only for cache write-backs (see Figure 3). Since the new configuration allows both buses to access memory simultaneously, simple dual-ported memory or some additional logic or buffer to allow two simultaneous accesses to memory is also required.

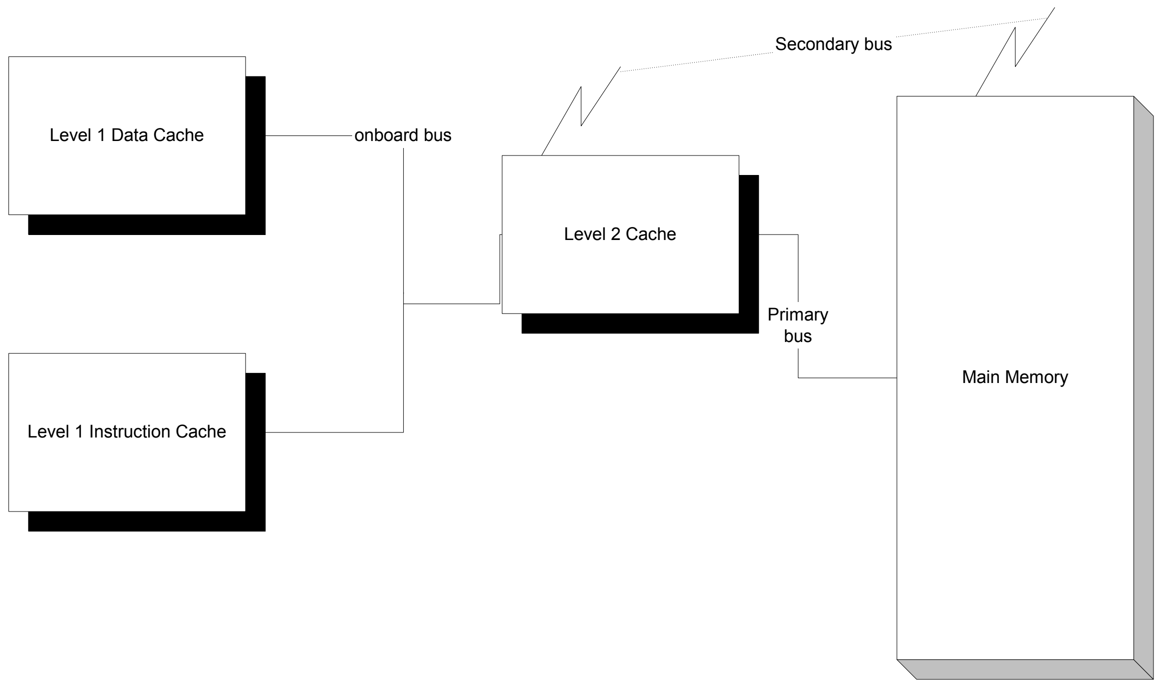


Figure 3: Secondary Bus Architecture

3.4. Simulation framework

The parameters used for the simulations are shown in Tables 1 and 2. The parameters are similar to those of a modern microprocessor, within the limitations of the simulation environment, with the exception of the secondary bus, which is only eight bits wide. The bandwidth of the secondary bus was chosen to be much less than that of the primary bus as the lower bandwidth is sufficient and allows for a variety of possible implementations of the bus, such as a serial line or a wireless link. Miss status holding registers and victim buffers were included in the simulated environment.

Table 3: Baseline Processor Configuration

Processor Parameter	**Specification**
Core Frequency	2 GHz
Level 1 Data Cache	4-way, 8192 sets, 4B line, virtually-indexed physically-

	tagged
Level 1 Instruction Cache	2-way, 8192 sets, 32B line, virtually-indexed physically-tagged
Level 2 Cache	4-way, 32768 sets, 32B line, physically-indexed physically-tagged
Onboard bus	64 bits wide,2 CPU cycles/bus cycle
Primary bus	64 bits wide 5 CPU cycles/bus cycle
Secondary bus	8 bits wide, 5 CPU cycles/bus cycle
Victim buffer	8 entries
MSHR	8 per cache

Table 4: Bus Latencies

Parameter	**Cycles in bus clocks**
Level 1 Instruction and Data cache	3
Level 2 Cache	18
Onboard bus arbitration	4
Primary bus arbitration	10
Secondary bus arbitration	10

3.5. Maximum speedup using free write-back

In order to determine the maximum possible speedup that may be obtained by using a secondary bus for cache write-back, modifications to the simulator allowed the entire write-back process to be bypassed for the L2 cache. The modifications created a situation in which write-backs from the L2 cache required no processor cycles and were therefore "free". This free write-back behavior represents an upper bound on the improvements possible when using the secondary bus. A comparison of performance when using a secondary bus with free write-back demonstrates the efficiency of the secondary bus.

3.6. Benchmarks

The three benchmarks detailed in Table 5 were used for the simulations. The benchmarks include two forms of matrix manipulation: matrix transpose and scalar multiplication, and a simulation of a graphics engine. These benchmarks represent a cross-section of applications for which cache and bus usage is high, and for which an additional bus might provide performance gains.

Table 5: Small Cache Benchmarks

Benchmark	Description
matscalar	multiplication of a scalar with a matrix
mattrans	matrix transpose
minigeo	the mini-geometry kernel described in [15]

3.7. Simulation results and analysis

The simulation results showed that the addition of a secondary bus for cache write-back could reduce the bus queuing delay and provide improved determinacy in real-time systems. An analysis of the bus usage and queuing delay improvement when using a secondary bus follows.

3.7.1. Bus queuing delay

Bus queuing delays can prove significant in real-time systems where unexpected latencies can cause hard deadlines to be missed. Significant changes in the usage of buses within the system were observed with the addition of the secondary bus. Decrease in the number of requests on the primary bus (see Table 6) is attributable to the use of the secondary bus for cache write-backs, since that was the only change. This decrease lead to a significant decrease in the number of cycles during which requests to the primary bus were queued (see Table 7) and decreased the queuing delay experienced by requests on the primary bus (see Table 8 and Figure 4).

Table 6: Decrease in requests on system bus - small cache

Benchmark	Requests w/o secondary bus	Requests with secondary bus	% decrease
matscalar	3,355,430	2,323,798	30.75%
mattrans	1,515,039	1,104,512	27.10%
minigeo	2,194,937	1,552,028	29.29%

Table 7: Decrease in queued cycles - small cache

Benchmark	Cycles queued w/o secondary	Cycles queued with secondary	% decrease
matscalar	58,109,468	883,571	98.48%
mattrans	37,005,612	4,208,484	88.63%
minigeo	62,802,508	9,534,934	84.82%

Table 8: Decrease in queuing delay on system bus - small cache

Benchmark	Improvement with secondary bus	Improvement free writeback
matscalar	98.22%	98.22%
matttans	84.40%	84.43%
minigeo	78.52%	83.22%

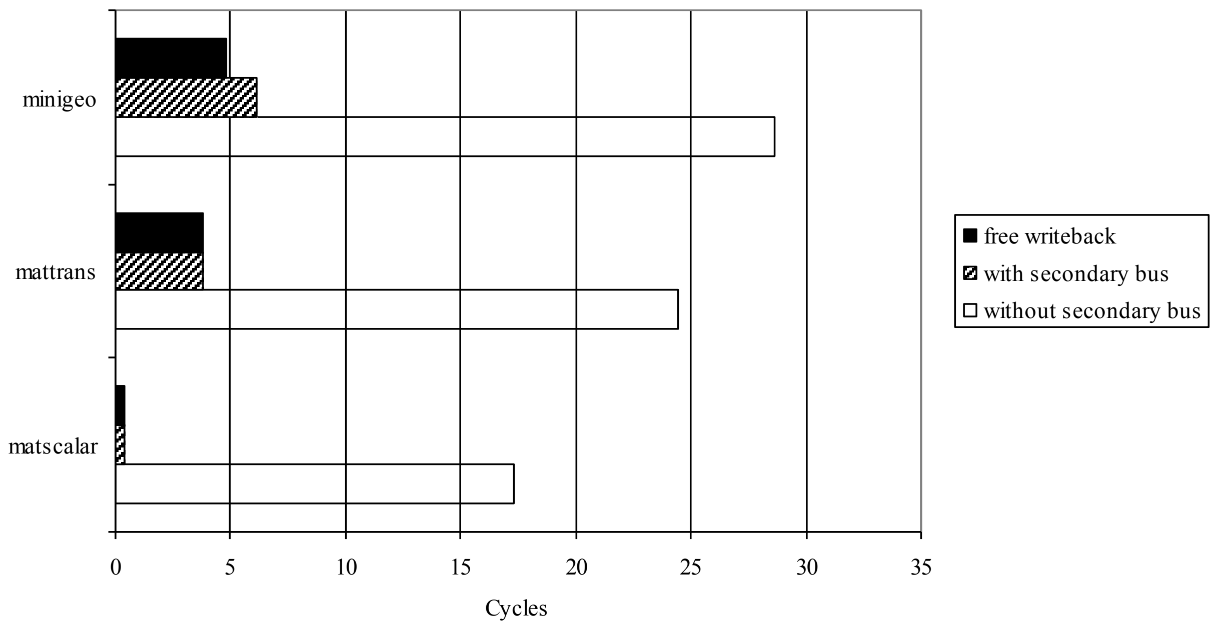


Figure 4: Queuing delay on primary bus - small cache

3.7.2. Secondary bus analysis

The queuing delay on the secondary bus is shown in Table 9. In all cases the delay is small, and is much smaller than the queuing delay on the primary bus. This shows that the limited bandwidth of the secondary bus is sufficient to support the write-back traffic.

Figure 5 illustrates the usage of the buses in the system during a cache miss that requires a write-back, with and without the secondary bus. When the secondary bus is not present, the write-back request must be queued until the read transfer is complete. If a subsequent request occurs during this time, it must also be queued, leaving the onboard bus idle waiting for the transaction to complete. When the secondary bus is present, the primary bus becomes free sooner, permitting a subsequent request to begin sooner and experience a shorter queuing delay, and also reducing the time during which the onboard bus must remain idle waiting for data to be returned.

Table 9: Queuing delay on secondary bus - small cache (cycles)

Benchmark	Queuing delay on secondary bus
matscalar	0.028
matttans	0.785
minigeo	0.139

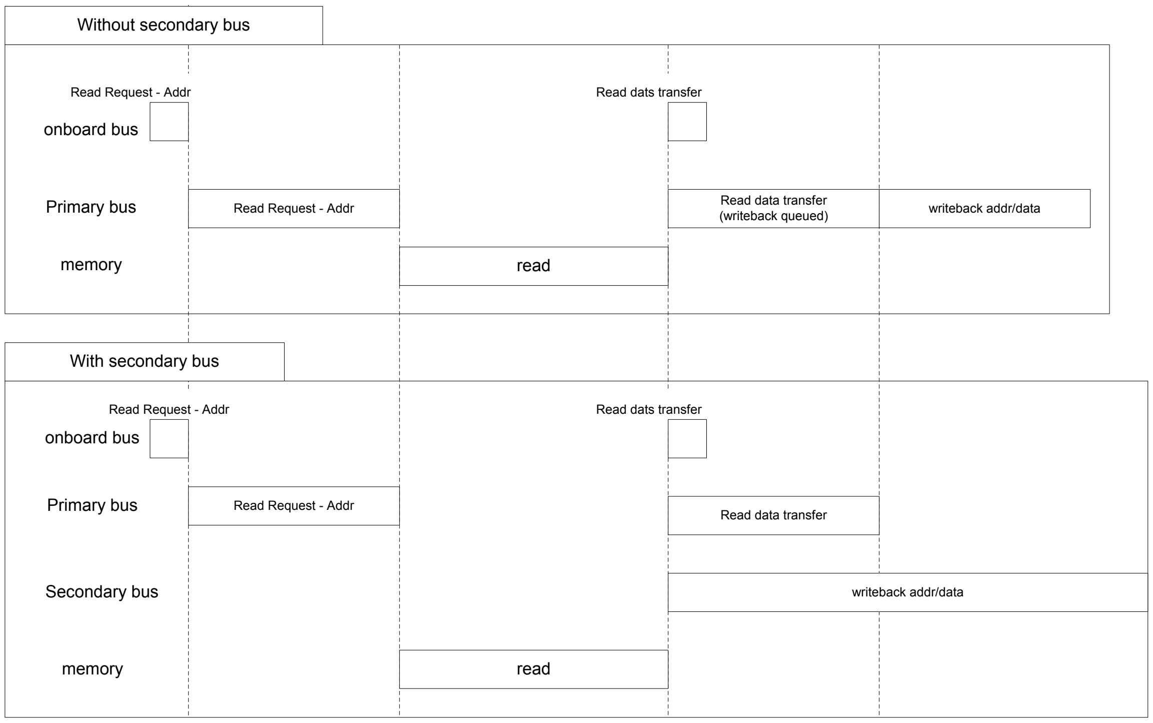


Figure 5: Bus timing

3.7.3. Write-back rates

As expected, the most important value that affects the potential usefulness of a secondary bus for cache write-back is the write-back rate. Table 10 shows the L2 cache write-back rates observed during the various benchmarks, where the write-back rate is defined as the ratio of the number of write-backs to the total number of accesses. The secondary bus provides significant decrease in bus queuing delay for programs that have a sufficient cache write-back rate.

Table 10: Write-back rates - small cache benchmarks

Benchmark	L2 write-back rate
matscalar	24.2%
mattrans	24.0%
minigeo	42.0%

3.8. Conclusions

The results obtained during these simulations show a large reduction in the queuing delay experienced by requests to the primary bus when using a secondary bus for cache write-back. However, significant speedup results can only be obtained for applications such as graphics-intensive applications in which the write-back rate is sufficient to require significant use of the secondary bus. The reduction in bus queuing delays could have a positive impact on real-time system temporal determinacy by reducing unexpected execution delays. Real-time and near real-time embedded applications that depend on intense graphics processing and movement of large blocks of data, such as printer controllers and medical imaging that involve significant I/O, are prime candidates for applications of the secondary bus.

Comparison of the secondary bus to the theoretical maximum speedup shown by the free write-back case show that the secondary bus provides decreases in queuing delay that are near the maximum possible.

Chapter 4

Analysis of Large Cache Systems with a Secondary Bus

4.1. Introduction

In order to further demonstrate the feasibility of the secondary bus microarchitecture, it is necessary to show its advantages in a system with larger caches while executing industry-standard benchmarks. The small cache secondary bus microarchitecture depended on dual-ported memory, which may add significant cost to the design. Adding a control mechanism in addition to the memory controller would eliminate the need for dual-ported memory and enable the secondary bus to snoop the main bus and initiate memory write-back when the later is not being used for memory operations or busy with I/O transactions between the CPU and the I/O devices. This additional control mechanism could be added for small cost.

4.2. Architecture of the Secondary Bus

The motivation behind the secondary bus architecture design is to first provide separate paths from write buffers to main memory so that stalls due to the buffer becoming full can be avoided [18]. Additionally, even during I/O transactions on the main bus, the secondary bus could be used as an alternate path to commit the dirty cache lines to the memory. The main memory is assumed to be a single port memory in this design, i.e. only one unit can access the memory at any point in time. The design of the secondary bus based architecture has been shown in Figure 4 [4].

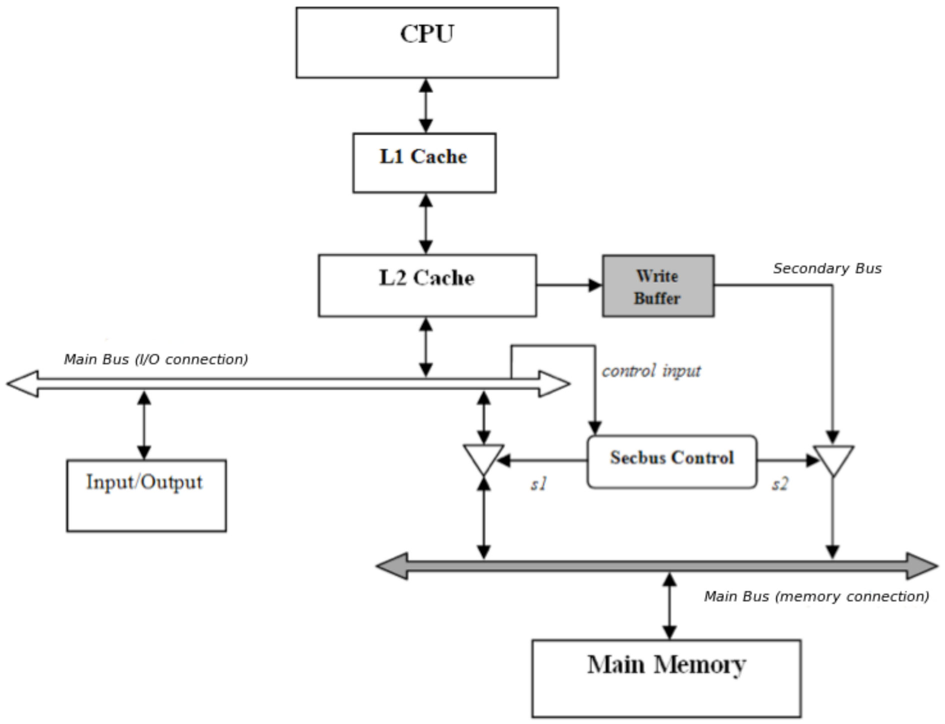


Figure 6: Large cache architecture with secondary bus

As seen in Figure 6 [18] [25], the secondary bus supports the main bus during write-backs and I/O transactions. This is made possible by the secondary bus controller, which snoops the main bus and identifies bus cycles that are not involved in memory accesses. These cycles will be used to retire the dirty cache lines to the memory over the secondary bus. The control inputs to the secondary bus controller are made up of the main bus control lines that give information about the type of transaction happening on the bus. The signals ‘s1’ and ‘s2’ are sent in accordance with the states of the main bus to arbitrate accesses. A simple main bus state diagram is shown in Figure 7 [18].

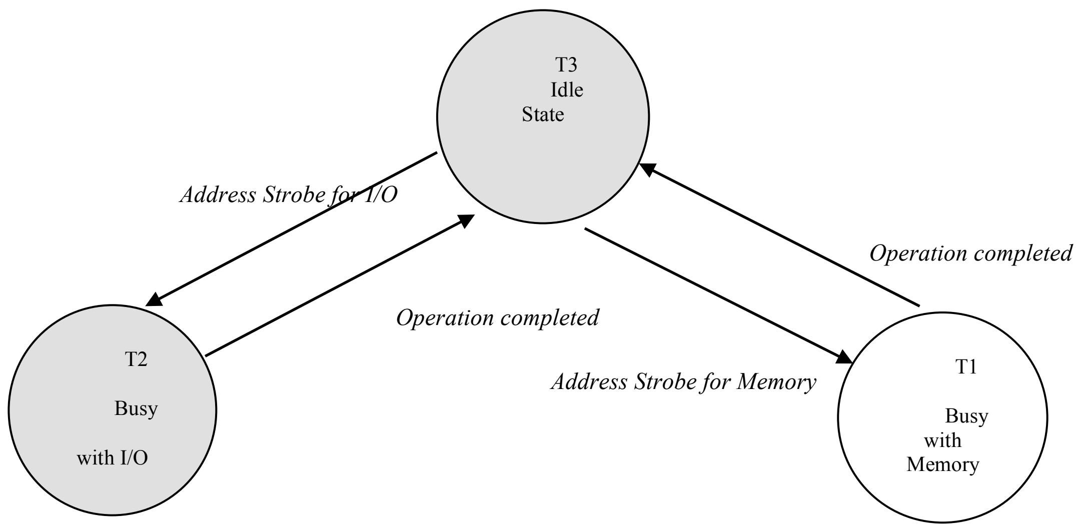


Figure 7: Simplified bus state diagram

In state T3 the main bus is idle and is waiting for an address strobe to move to the next state. If the address supplied is for an I/O operation, the main bus would move to T2. For the duration when the main bus is busy with I/O, it will be in T2 and upon completion of I/O will return to T3. In state T1 the bus is busy with memory accesses. In states T2 and T3 write-back entries can be retired through the secondary bus. The controller is designed so that s1 in state T1 is a connection-enabling signal and s2 is a disabling signal. In states T2 and T3 the reverse is true.

There can be situations where the main bus state would change from T3 to T1 or from T2 to T1 via T3 in the middle of a cache write-back commit via the secondary bus. To handle this situation, the first option is to queue the main bus request until the write operation completes and the write-back buffer is empty. The second option aborts the write-back and enables the main bus to access memory. This option of giving the main bus access priority was adopted in our design in order to not add to the queuing delay on the bus and for consistency with exiting architectures. Note that the state transitions of the secondary bus depend on the state of the main bus.

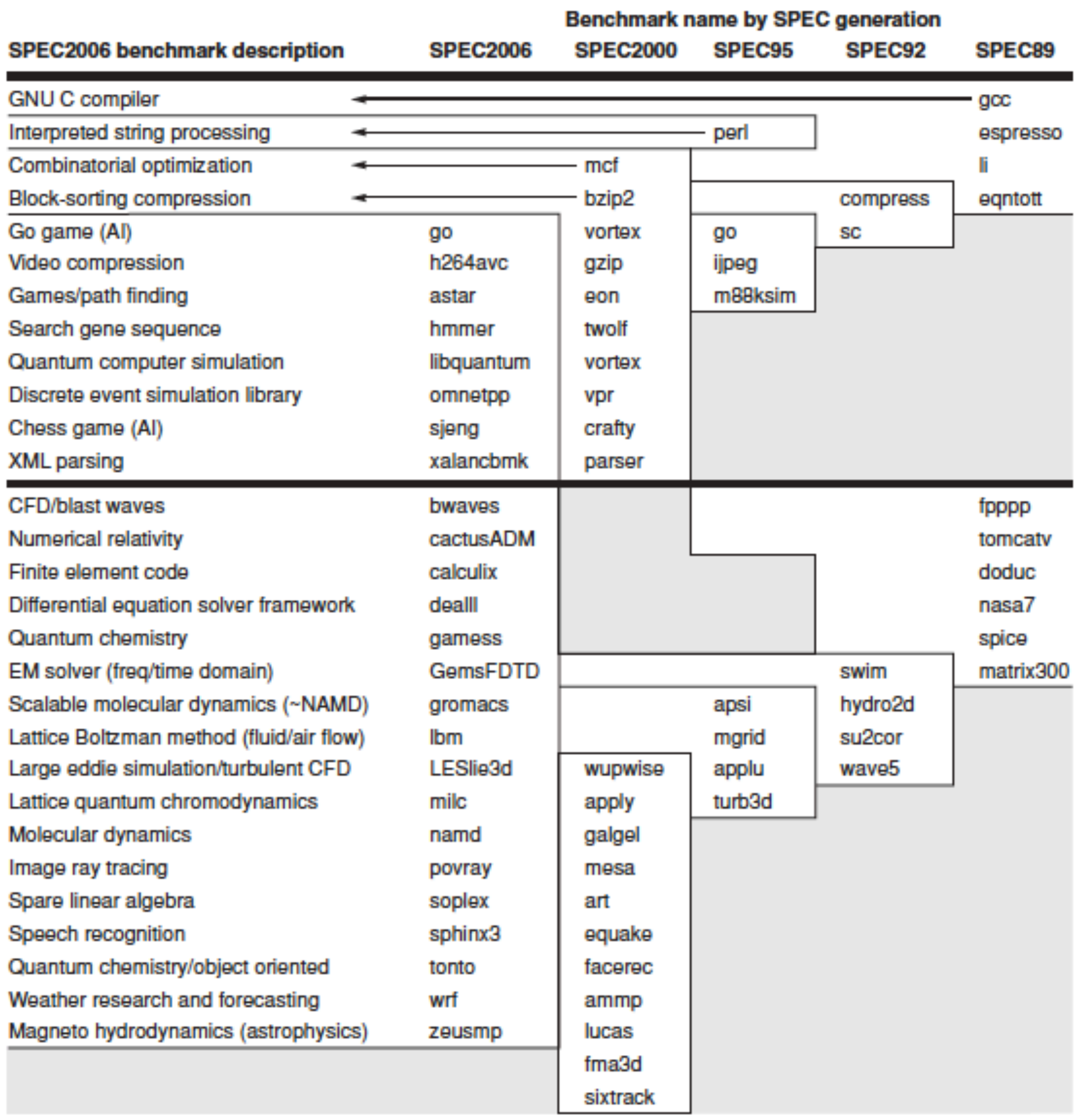
The addition of the secondary bus and its bus controller adds minimal overhead to the motherboard and the memory controller hub respectively. The secondary bus is narrow (8-bits), of a smaller bandwidth, and is only used for write-backs. On average fewer than 30% of all memory accesses in the SPEC CPU2006 benchmarks are writes, so a narrow bus could provide ample bandwidth. If the bandwidth of the secondary bus is adequate, our design ensures that the performance of a processor with the secondary bus will be no worse than one without. The usefulness of the secondary bus depends on the amount of time the main bus spends in the states of T2 and T3. Thus memory bound applications such as graphics-intensive applications and I/O bound applications such as database systems will see significant benefits.

4.3. Simulation Speedup Using Simpoints

The initial simulations using a small cache size were executed successfully on a Linux-based PC in a reasonable amount of time. When the cache size of the system was increased, and the SPEC CPU2006 benchmarks were used, the execution times became unacceptable. Execution of those benchmarks takes weeks or months of machine time [10], and there were several benchmarks to execute, each requiring multiple runs with varying system parameters. This problem was overcome by using the Simulation Points method [19] of only executing a small, representative subset of the entire benchmark. The SimPoint tool uses a profiling method called Basic Block Vectors to find blocks of code with similar execution behavior. The output of the SimPoint tool is a list of points to be simulated and their corresponding weights. Using those points, simulation time is greatly reduced. Errors with SimPoint have been shown to be always less than 8.4%. The use of simpoints allowed the execution time of the simulations to be reduced from weeks to days. In addition, the simulations were executed on the Alabama Supercomputer. With this combination, each of the simulations could be executed within about two days, and the entire suite of benchmarks with all combinations could then be executed in a reasonable amount of time.

4.4. SPEC CPU2006 Benchmarks

The SPEC CPU2006 benchmark suite consists of 29 benchmarks (12 integer and 17 floating-point). The SPEC benchmarks are real-world applications that have been modified to be easily portable and to minimize the effects of I/O on performance. The evolution of the SPEC benchmarks is shown in Figure 8 [11].



SPEC2006 benchmark description	Benchmark name by SPEC generation				
	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

Figure 8: Evolution of the SPEC Benchmarks

In order to execute the benchmarks on the Sim-alpha simulator, it was necessary to obtain Alpha binaries for the benchmarks, obtain the simpoints information for the benchmarks, and then successfully execute the tests on the simulator. In addition, those benchmarks that did not generate any cache write-backs did not exercise the secondary bus at all, and were therefore of no use in the evaluation. Alpha binaries for the benchmarks were obtained from Kenneth Hoste of Ghent University in Belgium. The following benchmarks were not executed because they would not execute on the Alpha simulator:

- 401.bzip
- 445.gobmk
- 453.povray
- 456.hmmer

The following benchmarks were not executed because simpoints were not available for those tests:

- 403.gcc
- 429.mfc
- 454.calculix
- 465.tonto
- 470.lbm
- 481.wrf
- 483.xalanbmk

The following benchmarks had no write-backs at all, and therefore were of no value to the evaluation:

- 416.gamess
- 444.namd

The following benchmarks had very small write-back rates, and therefore very little speedup:

- 400.perlbench
- 435.gromacs
- 458.jsend
- 464.h264ref
- 473.astar

The remaining benchmarks, with their respective speedup values using the secondary bus microarchitecture, are shown below:

- 410.bwaves: 18.83%
- 433.milc: 4.89%
- 434.zeusmp: 7.79%
- 436.cactusADM: 4.78%
- 437.leslie3D: 2.89%
- 447.dealII: 14.90%
- 450.soplex: 17.57%
- 459.gemsFTDT: 18.93%
- 462.libquantum: 18.02%
- 471.omnetp: 10.91%
- 482.sphinx3: 13.61%

Tables 9 and 10 summarize the simulation details for all of the SPEC CPU2006 benchmarks.

Table 11: SPEC CPU2006 Integer Benchmarks

400.perlbench: PERL programming language checkspam test	Small write-back rate (0.53%)	0.25% speedup
400.perlbench diffmail test	Small write-back rate (0.03%)	no speedup
401.bzip: Compression.	Would not execute on Alpha simulator.	
403.gcc: C compiler	Simpoints not available.	
429.mcf: Combinatorial optimization.	Simpoints not available.	
445.gobmk: Artificial intelligence: GO	Would not execute on Alpha simulator	
456.hmmer: Search gene sequence	Would not execute on Alpha simulator	
458.jseng: Artificial intelligence: chess	Small write-back rate (0.63%)	0.10% speedup
462.libquantum: Physics: quantum computing	36.42% write-back rate	18.02% speedup
464.h264ref: Video compression	Small write-back rate (0.27%)	0.08% speedup
471.omnetp Discrete event simulation	16.82% write-back rate	10.91% speedup
473.astar: Path-finding algorithm	Small write-back rate (0.88%)	0.42% speedup
483.xalancbmk: XML processing	Simpoints not available.	

Table 12: SPEC CPU2006 floating-point benchmarks

410.bwaves: Fluid dynamics	45.16% write-back rate	18.83% speedup
416.gamess: Quantum chemistry	No write-backs	
433.milc: Physics: Quantum Chromodynamics	44.84% write-back rate	4.89% speedup
434.zeusmp: Physics/CFD	11.56% write-back rate	7.79% speedup
435.gromacs: Biochemistry/Molecular dynamics	Small write-back rate (1.84%)	0.38% speedup
436.cactusADM: Physics/general relativity	26.85% write-back rate	4.78% speedup
437.leslie3d: Fluid dynamics	30.9% write-back rate	2.89% speedup
444.namd: Biology/Molecular dynamics	No write-backs	
447.dealII: Finite element analysis	20.2% write-back rate	14.90% speedup
450.soplex: Linear programming, optimization	22.12% write-back rate	17.57% speedup
453.povray: Image ray-tracing	Will not execute on Alpha simulator	
454.calculix: Structural mechanics	Simpoints not available	
459.GemsFDTD: Computational electromagnetics	36.35% write-back rate	18.93% speedup
465.tonto: Quantum chemistry	Simpoints not available	
470.lbm: Fluid dynamics	Simpoints not available	
481.wrf: Weather prediction	Simpoints not available	
482.sphinx3: Speech recognition	35.26% write-back rate	13.61% speedup

4.5. Simulation Setup

The usefulness of the secondary bus architecture was demonstrated using microprocessor simulation techniques. The simulation was again conducted using the SimAlpha simulator of the SimpleScalar tool set. The SPEC CPU 2006 benchmark suite was used for evaluating the design.

These benchmark programs were found to exercise the memory hierarchy better than SPEC 2000 [10]. SimAlpha was used with the simpoints to simulate the programs at those critical points and the results were later averaged using specific weights for each of those simpoints. The specifications used for the simulation are given in Table 13 [18] [25]. The SimAlpha simulator was modified to incorporate the bus controller mechanism and a write-back buffer capable of holding 256 cache lines. The L2 cache used the write back technique for cache coherency and hence a write-back buffer was required for smaller write back latencies. The secondary bus was connected between the write-back buffer and the main memory. The bus controller, as mentioned in the design, regulates when the secondary bus may gain access to the memory while giving higher priority to the main bus. The write-back buffer was implemented in the simulator as an additional cache connected to the L2 cache. In addition, the simulator code was modified so that writes from the write-buffer to memory via the secondary bus did not occur until the write buffer was full, at which point all of the entries were written to memory in burst mode. The functionality of the write-back buffer is similar to that of a cache, therefore simulating it as a cache is an elegant way to include its functionality in the SimAlpha simulator.

To further test the secondary bus architecture, an I/O injection mechanism was incorporated into the SimAlpha simulator code so that a given number of bytes could be injected into the bus at a user provided frequency. It was used for simulating the communication between the various devices connected to the chipset and the CPU. The bandwidth of the main bus used in the simulations was 4.8 GB/Second with quad data rate and hence I/O injection frequencies of 1.2 GB/Sec and 1.8 GB/Sec are a good estimate of the amount of bandwidth used by I/O on the main bus. The L2 cache sizes that were chosen are comparable to the latest embedded processor specifications.

The simulations were run in three different modes:

1. Zero I/O traffic, so that the main bus was used only by the processor to communicate with the memory via the controller for reading data and instruction blocks, and the secondary bus was used for the writing. Both the base and the secondary bus architectures were simulated.
2. With an I/O traffic injection into the main bus at an injection frequency of 200 Bytes/100 Cycles, which resulted in a bandwidth injection of 1.2 GB/Second.
3. An injection frequency of 300 Bytes/100 Cycles was used to provide a bandwidth of 1.8 GB/Second in the last case.

Table 13: Large cache simulation parameters

Processor Parameter	Specifications
Processor Speed	3 GHz
Level 1 Data Cache	8 way, 32KB, virtual-index virtual-tag
Level 1 Instruction Cache	8 way, 32KB, virtual-index virtual-tag
Level 2 Cache	8 way, 2MB, physical-index physical-tag
Number of MSHRs per Cache	8
Write Mechanism for Level 1 Cache	Victim Buffer, No Writeback Buffer
Write Mechanism for Level 2 Cache	Writeback Buffer, No Victim Buffer
Main Bus (Front Side Bus)	600MHz, 8B wide, 10 cycles of arbitration latency
Secondary Bus	600MHz, 1B wide, 10 cycles of arbitration latency

The results obtained during the simulations were based on simulation methods that have been shown to be valid. The comparison of the results obtained using the secondary bus with those of free write-back verify that the secondary bus microarchitecture delivers results that are close to but never exceed the maximum obtainable results of free write-back. In addition, the lack of any change in the results for tests with zero write-backs show that, as expected, the addition of the secondary bus has no affect in that case.

4.6. Simulation Results and Analysis

The results of the simulation demonstrate the ability of the secondary bus to improve performance. Most significantly, the queuing delay on the system bus is reduced. This reduction leads to improved performance in real-time systems, as well as improved overall system performance. In order to use the secondary bus, software programs must include write-backs from the level 2 cache to main memory. Table 14 shows the write-back rates of the SPEC CPU2006 benchmarks.

Table 14: SPEC CPU2006 benchmark write-back rates

Test	Write-back Rate
perlbench	0.53%
bwaves	45.16%
zeusmp	11.56%
milc	44.84%
gromacs	1.84%
cactusADM	26.85%
namd	0.00%
deal	20.20%
soplex	22.12%
sjeng	0.63%
GemsFDTD	36.35%
libquantum	36.42%
h264ref	0.27%
omnetpp	16.82%
astar_rivers	0.88%
sphinx	35.26%

In addition to the write-back rate, the overall usage of the cache by the benchmarks is important. Table 15 shows the number of accesses to the level-2 cache for each benchmark.

Table 15: SPEC CPU2006 benchmark level-2 cache usage

test	L2 cache accesses
perlbench	1295680

bwaves	209418624
milc	427381376
zeusmp	57956608
gromacs	1948608
cactusADM	861056000
leslie	183522560
deal	1759232
soplex	239954496
sjeng	1798656
GemsFDTD	147450688
libquantum	169901760
h264ref	2355328
omnetpp	133807744
astar_rivers	65588480
sphinx	132398080

4.6.1. Queuing Delay

When the bus is being used to service requests, any newly arriving requests must be queued. In real-time systems, these queuing delays can become significant, resulting in unexpected latencies and hard deadlines being missed. For instance, if an I/O device is requesting the use of the processor during a memory read/write it must wait until the main bus is free.

The main bus in the simulations has a bandwidth of 4.8 GBs that was shared among the I/O traffic of around 1.2 GBs, the write-back traffic, and the read traffic. Table 16 and Figure 9 show the percentage queuing delay reduction achieved with the secondary bus against the base architecture for each of the SPEC programs [25]. Almost all of the programs showed great reduction in the queuing delays, with an average reduction of nearly 87% with no I/O traffic on the main bus. The presence of the secondary bus helped in making the main bus less prone to bus contention as the write traffic was diverted. Observe that the percentage reduction starts to diminish as the I/O traffic becomes a larger portion of the main bus bandwidth. This is because the write traffic is now a smaller fraction of the total traffic between the CPU and the peripherals. Even in these conditions a reduction averaged to 80% and 77% during the two cases of I/O injected simulations.

Table 16: Decrease in queuing delay - large cache

test	No I/O	1.2 GB/s I/O	1.8 GB/s I/O
perlbench	8	7.90%	6.00%
bwaves	98.11%	85.12%	77.63%
milc	71.38%	68.10%	66.84%
zeusmp	99.68%	94.15%	92.28%
gromacs	95.34%	66.39%	61.95%
cactusADM	87.88%	65.05%	61.47%
leslie	88.80%	80.76%	79.50%
deal	6.56%	1.92%	1.60%
soplex	91.44%	69.74%	65.40%
sjeng	98.20%	79.59%	77.32%
GemsFDTD	97.03%	86.02%	82.47%
libquantum	99.19%	84.71%	81.63%
h264ref	94.96%	79.83%	75.39%
omnetpp	98.62%	89.26%	86.44%
astar_rivers	96.20%	86.60%	84.20%
sphinx	98.02%	89.85%	86.26%

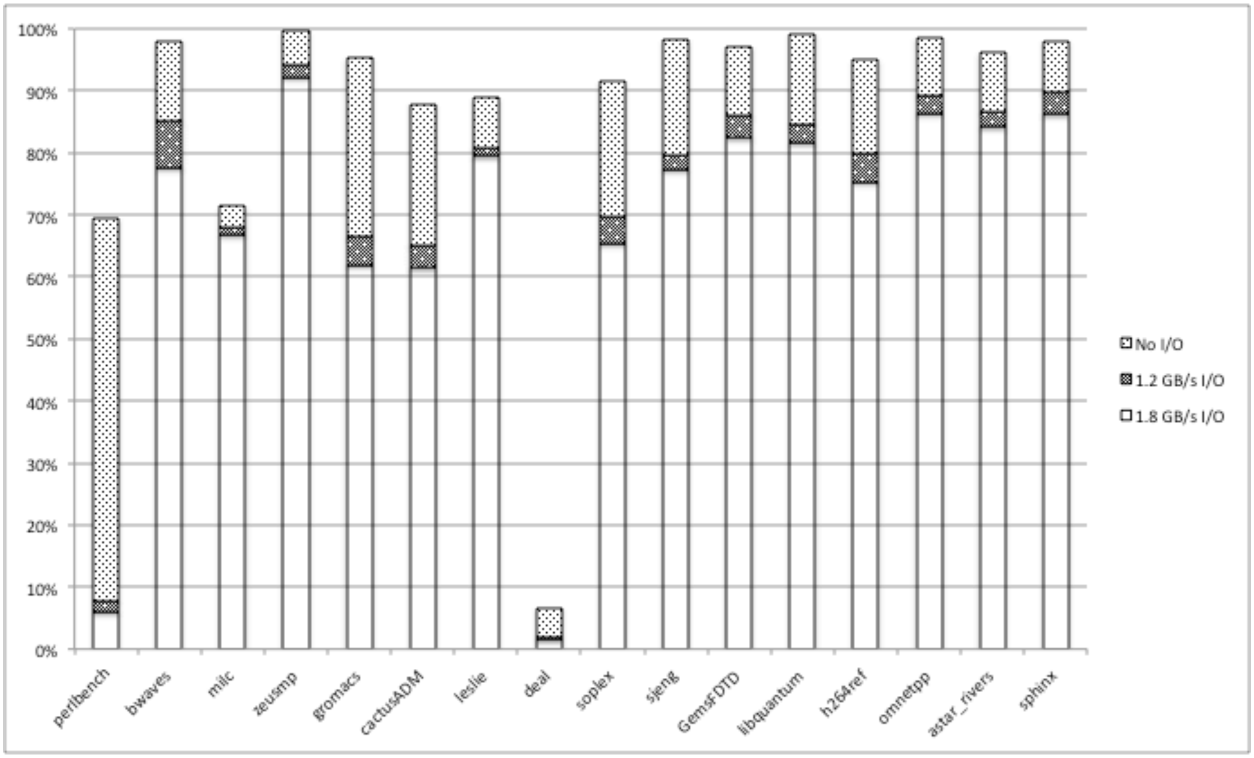


Figure 9: Decrease in queued cycles on system bus - large cache

4.6.2. Overall Speed-Up

A comparison of the ‘cycles per instruction’ between the secondary bus architecture and the base architecture gives us the speed-up achieved. Table 17 and Figure 10 show the percentage speed-up achieved across a range of programs from the SPEC CPU 2006 benchmark suite.

Table 17: Increase in cycles per instruction with secondary bus

test	No I/O	1.2GB/s I/O	1.8GB/s I/O
perlbench	0.25%	0.79%	0.94%
bwaves	18.83%	32.29%	32.81%
milc	4.89%	6.29%	6.81%
zeusmp	7.79%	18.77%	20.52%
gromacs	0.38%	1.20%	1.44%
cactusADM	4.78%	12.20%	13.83%
leslie	2.89%	3.83%	4.27%
deal	14.90%	18.47%	18.68%
soplex	17.57%	24.62%	25.01%
sjeng	0.10%	0.33%	0.32%

GemsFDTD	18.93%	30.12%	30.90%
libquantum	18.02%	19.99%	20.07%
h264ref	0.08%	0.19%	0.21%
omnetpp	10.91%	20.67%	21.79%
astar_rivers	0.42%	1.20%	1.44%
sphinx	13.61%	29.12%	30.11%

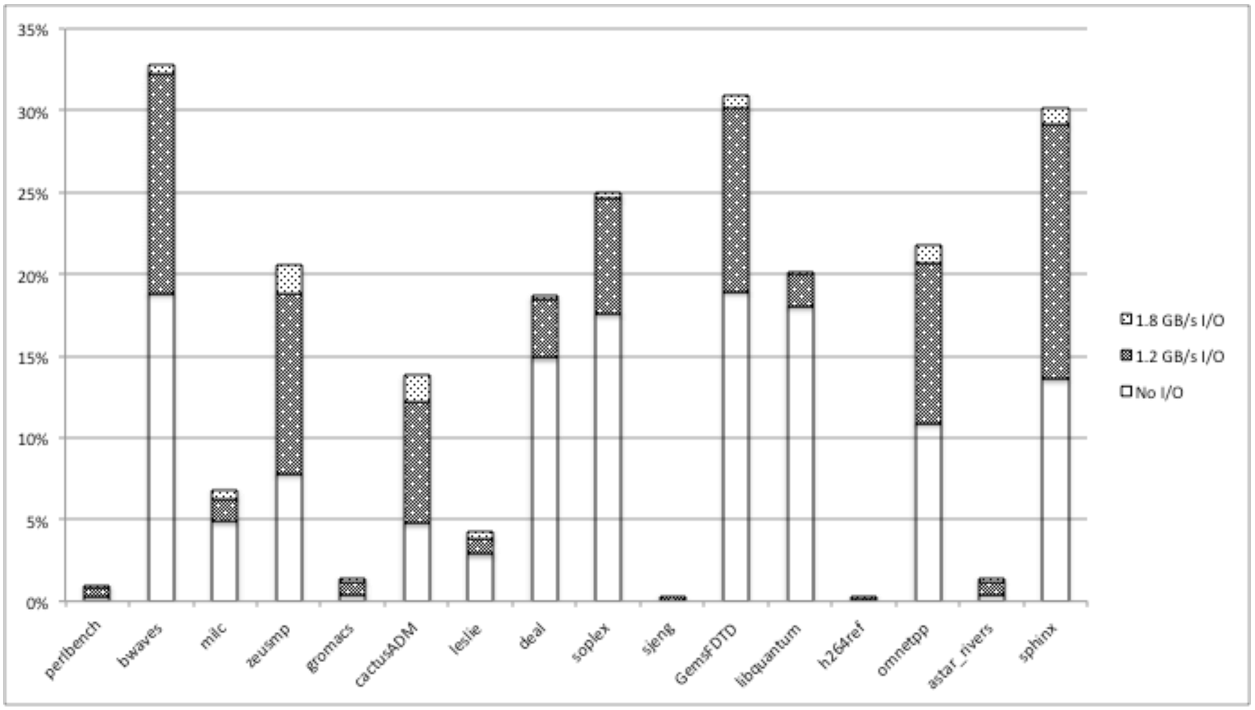


Figure 10: Percentage improvement with secondary bus

In the absence of I/O traffic on the main bus, speed-ups of up to 19% were achieved due to the addition of the secondary bus. The main reason for the performance improvement is the offloading of the write traffic on the main bus onto the secondary bus, as well as the timing of the memory access. With the presence of the secondary bus, the main bus never had to wait for the dirty write-back traffic to be written to the main memory whenever it requested data due to an L2 cache miss. In the presence of I/O traffic, further improvement was seen, with speed-ups of up to 33%. The secondary bus alleviates the performance degradation that normally happens on the main bus due to access contention by peripherals and memory.

Results also show that the speed-up depends on how much the program strains the memory hierarchy. Processors using smaller second level caches lead to higher number of cache misses and hence more write-backs. Thus programs having a very large working set could benefit compared to the ones using smaller caches and working sets such as gromacs, sjeng, and h264ref. The program namd did not have any write-backs to the memory and hence the architecture was never put to test during the simulations. Programs such as bwaves, zeusmp, gemsFDTD, and sphinx were highly write-back intensive with nearly 30% of the traffic on the main bus being write-back traffic [25].

Chapter 5

Analysis of Real-Time System Performance with a Secondary Bus

5.1. Introduction

Worst-case execution time (WCET) is a measure of the longest time that a program could take to execute on a target hardware system. Measurement of the WCET is necessary in safety-critical hard real-time systems to insure that critical system-dependent timing constraints are met. Examples of this type of system are brake control systems and flight controls. Additionally, numerous systems have soft real-time requirements that must meet real-time constraints during critical execution times, for example video processing systems and printers.

Accurately estimating the WECT of a real-time system is difficult. The two chief methods for determining WCET are either static analysis of the software and underlying hardware, or actual execution of the software on the target hardware with a variety of inputs. WCET estimates attempt to find measurements that reflect the actual worst-case behavior of the system without overestimating. Underestimating the WCET will lead to a system that fails, and overestimates lead to wasted resources and over-designed systems.

The underlying hardware architecture has a major impact on the WCET. Underlying architecture elements such as cache memory and instruction pipelining can lead to large variability in execution times. When the secondary bus microarchitecture is applied to real-time applications, it can deliver significant reductions in WCET.

5.2. New Measurement: Max Instruction Cycles

In addition to original SimpleScalar measurements, two additional metrics were added to the Sim-alpha simulations. First, measurements were taken of the maximum number of cycles used by any instruction during the entire benchmark. A reduction in this number has a direct correlation to a reduction of the WCET of the benchmark. Table 18 and Figure 11 show the significant decrease in the maximum cycles used by any instruction for the SPEC CPU2006 benchmarks. The reductions of up to 93% show that the addition of the secondary bus has a major impact on the WCET of the benchmark tests.

Table 18: Max Cycles Used by Any Instruction

Test	Original Max	Secondary Bus Max	Change
perlbench	14685	926	93.69%
bwaves	14575	2024	86.11%
milc	562	372	33.81%
zeusmp	15247	2090	86.29%
gromacs	14836	1098	92.60%
cactusADM	15370	3984	74.08%
leslie	674	464	31.16%
deal	14410	1047	92.73%
soplex	15732	1762	88.80%
sjeng	15192	1542	89.85%
GemsFDTD	15180	1880	87.62%
libquantum	14817	1002	93.24%
h264ref	15192	2080	86.31%
omnetpp	14860	1118	92.48%
astar_rivers	15080	1210	91.98%
sphinx	15190	1224	91.94%

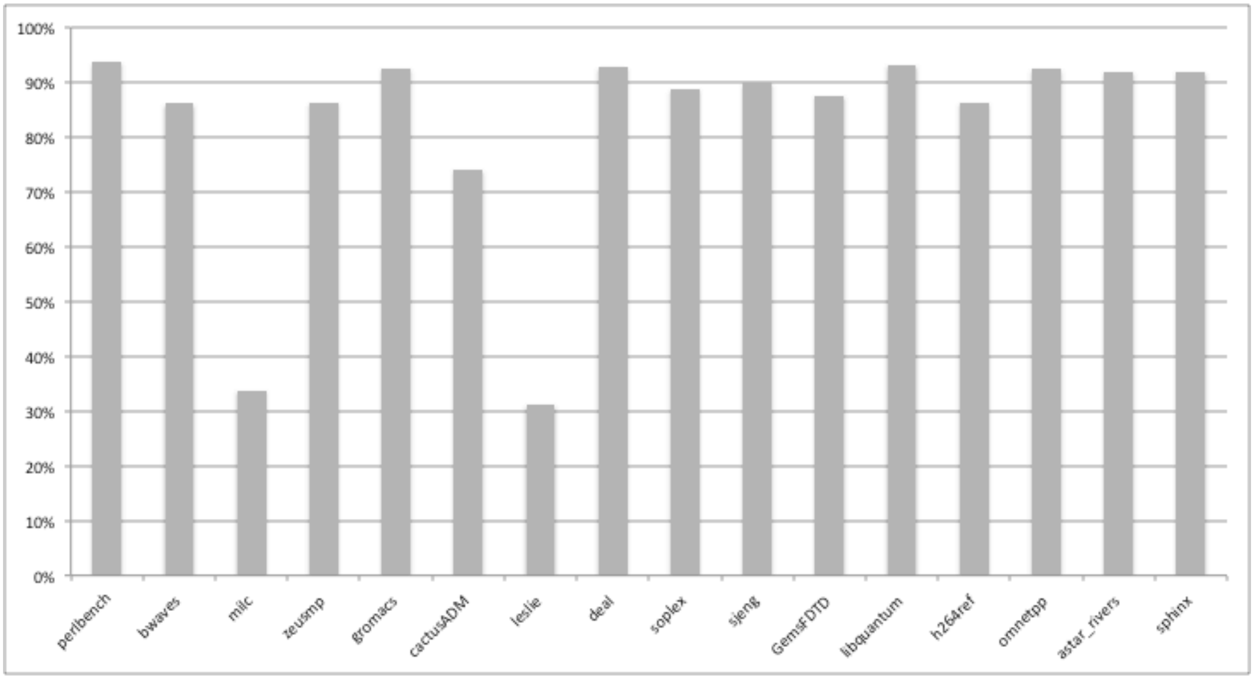


Figure 11: Change in Max Cycles

With I/O injection, the contention and hence the queuing delay on the main bus increased significantly and therefore instructions resulting in cache misses took more cycles than without I/O. As seen in Table 19, Table 20, Figure 12, and Figure 13, the reduction in the maximum cycles per instruction is still significant when I/O is present.

Table 19: Max Instruction Cycles with 1.2GB/s I/O

Test	Original Max	Secondary Bus Max	Max Cycles Change
perlbench	16404	4035	75.40%
bwaves	25300	4264	83.15%
milc	962	465	51.66%
zeusmp	28868	6563	77.27%
gromacs	16119	4359	72.96%
cactusADM	27758	9964	64.10%
leslie	760	505	33.55%
deal	14970	4426	70.43%
soplex	37097	7418	80.00%
sjeng	27652	5679	79.46%
GemsFDTD	27600	25090	9.09%

libquantum	26050	3312	87.29%
h264ref	27582	4820	82.52%
omnetpp	25191	4179	83.41%
astar_rivers	28175	4560	83.82%
sphinx	27640	5110	81.51%

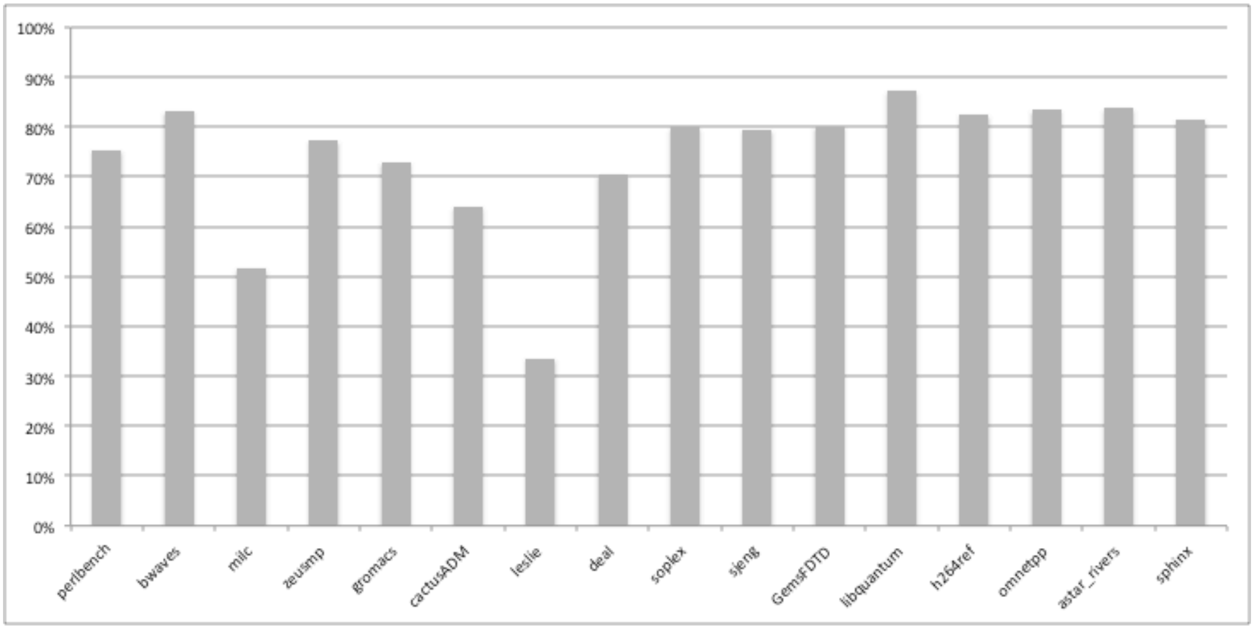


Figure 12: Change in max instructions with 1.2GB/s I/O

Table 20: Max Instructions with 1.8GB/s I/O

Test	Original Max	Secondary Bus Max	Max Cycles Change
perlbench	18295	4835	73.57%
bwaves	27002	4574	83.06%
milc	1127	425	62.29%
zeusmp	30238	7503	75.19%
gromacs	18969	5114	73.04%
cactusADM	29380	9279	68.42%
deal	15085	5446	63.90%
soplex	32432	9003	72.24%
sjeng	29067	7134	75.46%
GemsFDTD	29095	6660	77.11%
libquantum	27485	4077	85.17%
h264ref	29067	6041	79.22%

omnetpp	29139	5417	81.41%
astar_rivers	29865	5715	80.86%
sphinx	28645	6115	78.65%

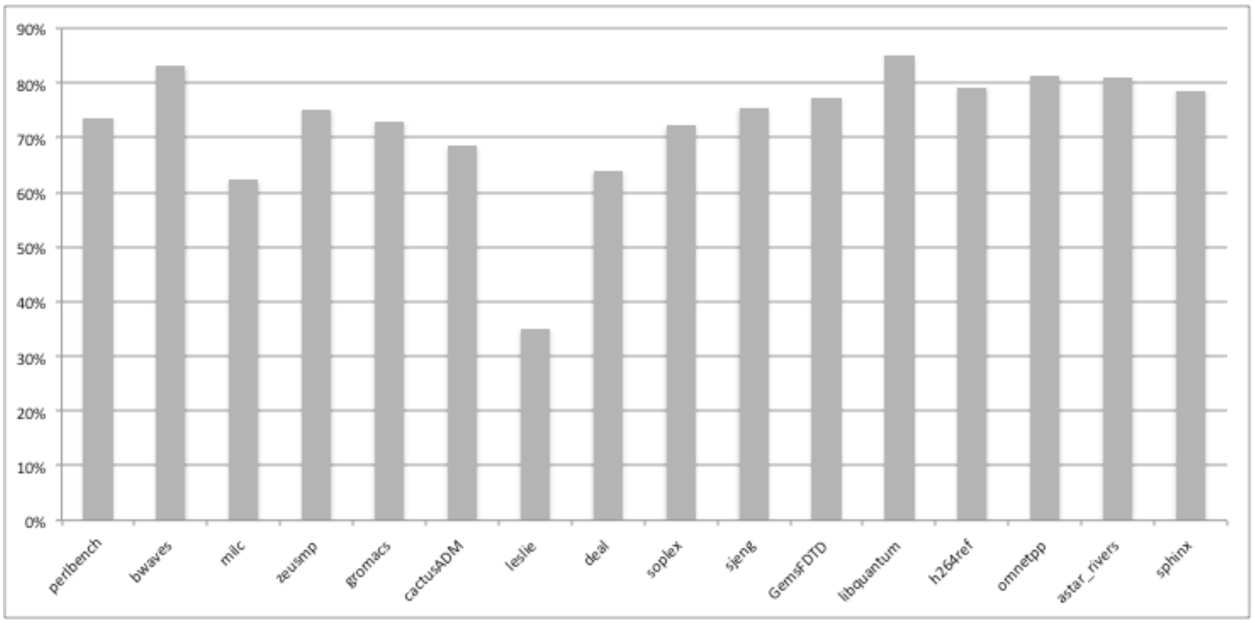


Figure 13: Change in Max Instructions with 1.8GB/s I/O

While there is no guarantee that the instruction requiring the maximum number of execution cycles is a part of the WCET path, a reduction in the max execution cycles is an obvious improvement in an attempt to reduce the WCET. The secondary bus delivers reductions of an order of magnitude in max execution cycles over the original system. This reduction, along with those measured in the next section, lead to a significant gain for hard real-time systems.

5.3. New Measurement: Instructions requiring more than 1000 cycles

In addition to the maximum number of cycles required by any instruction, the number of instructions per benchmark requiring more than 1000 cycles to execute was also measured. Although a majority of the simulated 100 million instructions took only 100 to 200 cycles to execute, there were instructions that took more than 1000 cycles due to cache misses. These additional cycles were mainly due to the latencies of memory accesses and main bus contention [25]. Even without I/O traffic, the base architecture was severely impacted by the contention between write-backs and reads on the main bus compared to the secondary bus architecture. With the secondary bus there was an average 97% decrease in the number of instructions taking more than 1000 cycles across the benchmark suite. This measure is also linked to the WCET, and the reduction in this metric shown with the addition of the secondary bus further demonstrates its ability to reduce the WCET in real-time systems. Table 21 and Figure 14 show the large decrease in the number of instructions requiring more than 1000 cycles in the SPEC benchmarks. Notice that the milc and lesie tests did not have any instructions that required more than 1000 cycles in the original setup, so they are not included in the table. When the secondary bus is present, the instructions requiring more than 1000 cycles is reduced by 100% in several of the benchmarks.

Table 21: Instruction using > 1000 cycles

Test	Original insn > 1000	Secondary Bus inst > 1000	Change
perlbench	64.93	0.00	100.00%
bwaves	6186.40	0.74	99.99%
zeusmp	1655.53	13.61	99.18%
gromacs	71.52	0.01	99.99%
cactusADM	1000.91	2.00	99.80%
deal	4946.94	3.56	99.93%
soplex	7177.96	639.41	91.09%
sjeng	23.99	2.92	87.85%
GemsFDTD	5417.60	7.52	99.86%
libquantum	6201.98	0.08	100.00%
h264ref	18.96	4.37	76.93%
omnetpp	3390.06	0.15	100.00%
astar_rivers	91.98	0.81	99.12%
sphinx	3591.36	16.25	99.55%

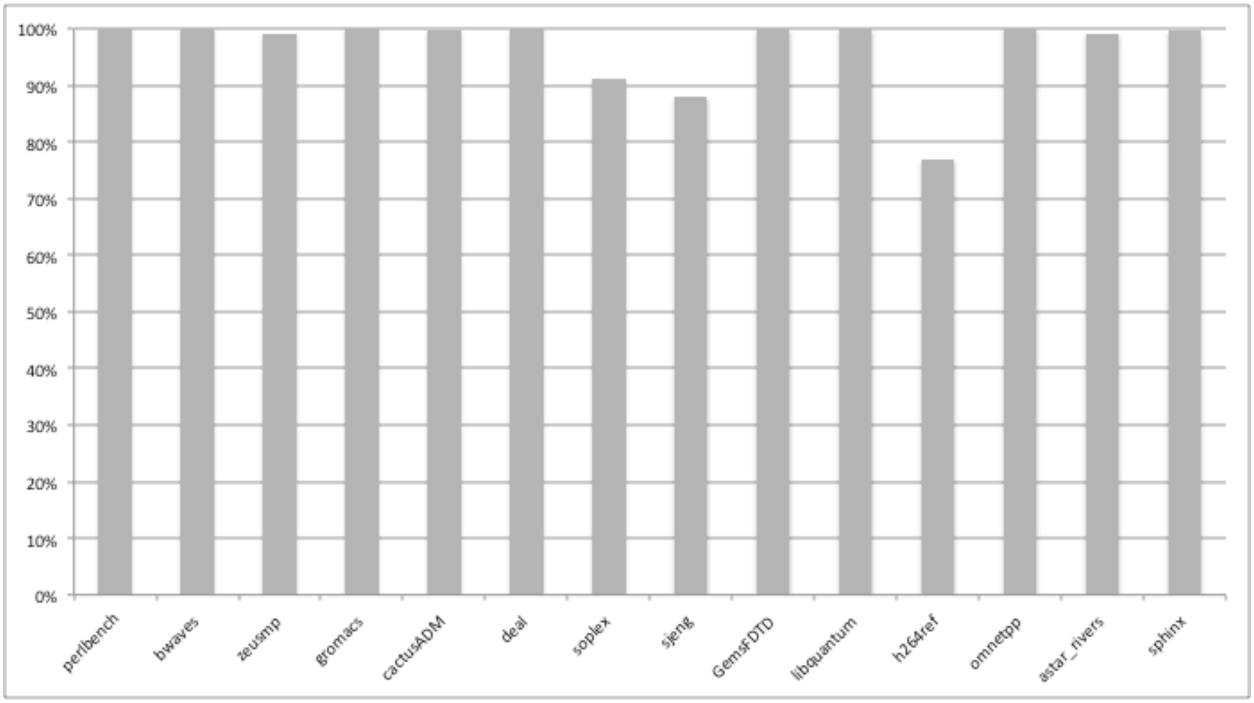


Figure 14: Change in instructions > 1000 cycles

When I/O is present, the reduction is also significant, as shown in Table 22, Table 23, Figure 15, and Figure 16. This decrease in the number of such time consuming instructions was more significant compared to the case where the I/O traffic was absent, which explains the increased speed-up with I/O injection as well. The results seen by measuring these two metrics indicate that the secondary bus microarchitecture is an excellent choice for real-time systems.

Table 22: Instructions > 1000 cycles with 1.2GB/s I/O

Test	Original insn > 1000	Secondary Bus inst > 1000	Change
perlbench	536.62	113.48	78.85%
bwaves	81290.82	17892.46	77.99%
zeusmp	22265.92	8565.09	61.53%
gromacs	546.07	23.76	95.65%
cactusADM	23282.93	19176.33	17.64%
deal	54567.71	38.94	99.93%
soplex	210101.78	158631.79	24.50%
sjeng	141.72	41.33	70.84%

GemsFDTD	105763.27	57626.26	45.51%
libquantum	50477.19	27.64	99.95%
h264ref	169.24	84.01	50.36%
omnetpp	43065.19	12517.53	70.93%
astar_rivers	656.21	146.00	77.75%
sphinx	32724.39	870.50	97.34%

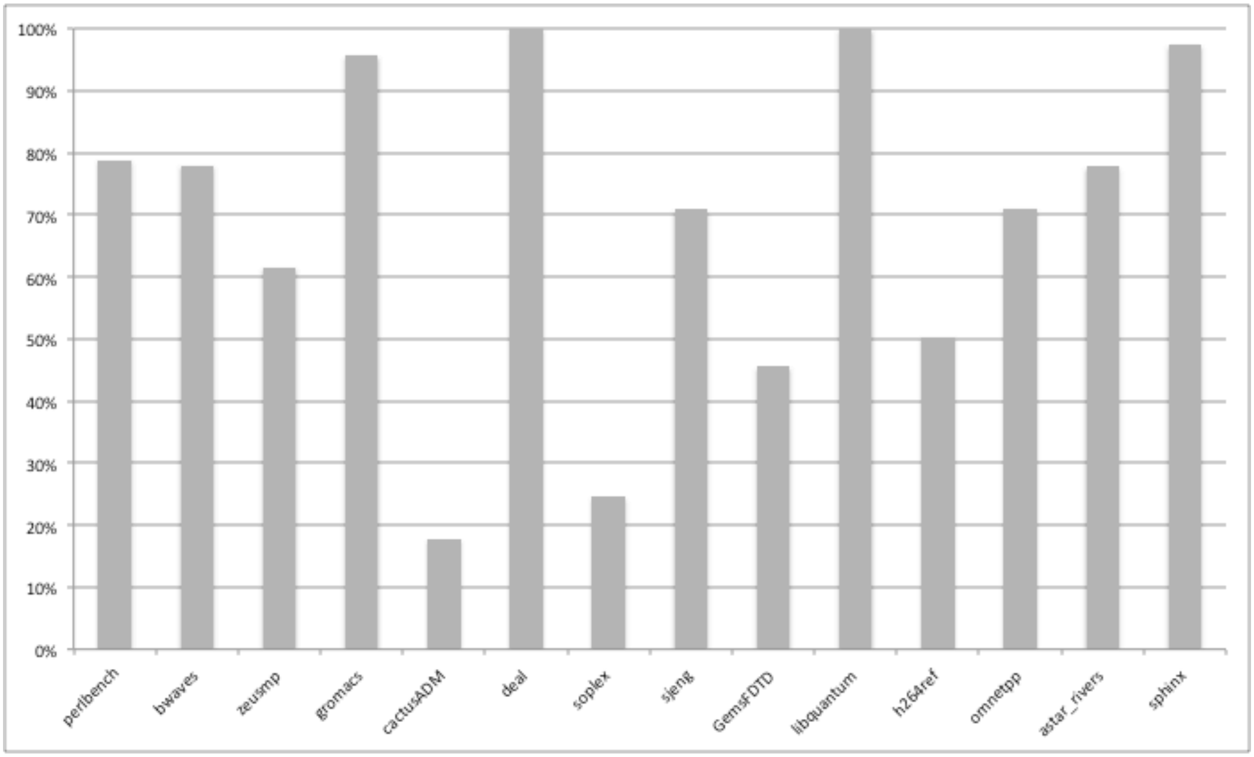


Figure 15: Instructions > 1000 cycles with 1.2GB/s I/O

Table 23: Instructions > 1000 cycles with 1.8GB/s I/O

Test	Original insn > 1000	Secondary Bus inst > 1000	Change
perlbench	729.09	202.87	72.17%
bwaves	165052.12	91615.23	44.49%
zeusmp	31779.34	15193.67	52.19%
gromacs	745.82	59.56	92.01%
cactusADM	27643.48	22485.62	18.66%
deal	80185.21	1249.17	98.44%
soplex	317720.40	263260.83	17.14%
sjeng	177.41	62.76	64.62%

GemsFDTD	158155.37	101660.01	35.72%
libquantum	66725.11	47.20	99.93%
h264ref	244.45	144.03	41.08%
omnetpp	61919.20	22989.39	62.87%
astar_rivers	817.24	183.42	77.56%
sphinx	42519.00	1457.06	96.57%

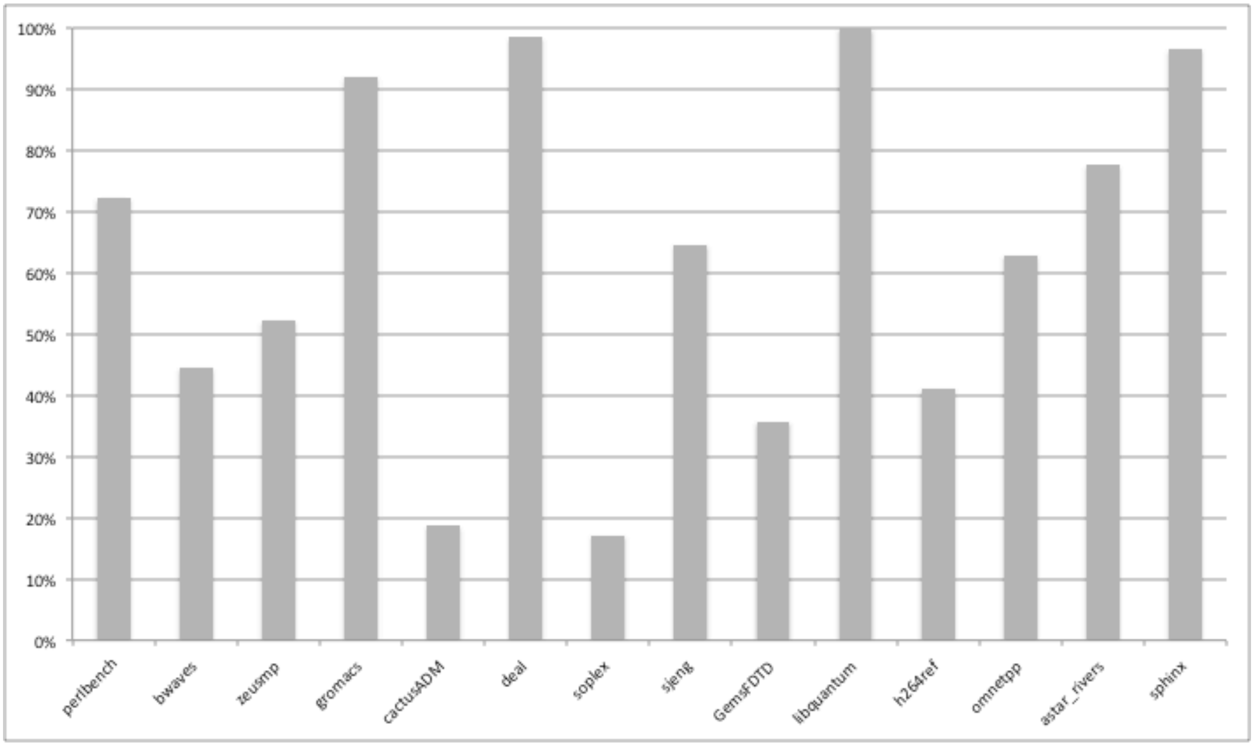


Figure 16: Instructions > 1000 cycles with 1.8GB/s I/O

The difficulty inherent in measuring WCET in any system, and the necessity to couple a real-time system with a specific microarchitecture in order to get accurate measurements mean that significant testing will be required to prove the advantages of the secondary bus microarchitecture. However, the significant reduction provided by the secondary bus microarchitecture in instructions requiring more that 1000 cycles to execute and in maximum cycles used by any instruction demonstrate its impact in reducing the cycles required for any sequence of instructions, including those that are included in the WCET path. Since the purpose of measuring the WCET is to guarantee that the timing constraints of the system are met, the most important measurement in a particular system is to guarantee that the absolute maximum execution times of the time-critical code segments are less that the required values. An additional advantage of the secondary bus architecture is to reduce the uncertainty and improve the predictability in execution times when using cache memories. This improvement can allow real-time system designers to take advantage of improved cache performance without risking missed timing constraints due to erratic cache-induce behavior.

5.4. Details of SPEC CPU2006 benchmark results

The function of each of the SPEC CPU2006 benchmarks used in the simulations is described below [21], along with details of the simulation results obtained with each test. The results show that the overall system speed-up is highest when the write-back rate is high and the use of the L2 cache is moderate. When the L2 cache usage goes up, the low bandwidth of the secondary bus does not permit additional speed-up. Also, the addition of the secondary bus to the system decreased memory latencies even when the write-back rate was small, leading to improved real-time performance in all cases.

5.4.1. 400.pearlbench

400.perlbench is a slimmed-down version of the popular scripting language Perl v5.8.7 with most of the OS-specific features removed. In addition to the core interpreter, the following third-party modules are used: SpamAssassin v2.61, Digest-MD5 v2.33, HTML-Parser v3.35, MHonArc v2.6.8, IO-stringy v1.205, MailTools v1.60, and TimeDate v1.16. The primary component of the workload is the Open Source spam checking software SpamAssassin. SpamAssassin is used to score a couple of known corpora of both spam and ham (non-spam), as well as a sampling of mail generated from a set of random components. It was heavily patched to avoid doing file I/O.

The write-back rate of less than 1% for this benchmark lead to an overall speed-up of less than 1%. The small usage of the secondary bus was, however, enough to reduce the number of queued cycles on the system bus by 69% when no I/O was present, 8% with 1.2GB/s of I/O, and 6% with 1.8GB/s of I/O. The maximum number of cycles used was reduced by 95%, 75%, and 74%, and the number of instructions requiring more than 1000 cycles was reduced by 100%, 79%, and 72%. This test is a good example of the ability of the secondary bus microarchitecture to improve real-time performance even when overall system performance is not affected.

5.4.2. 410.bwaves

410.bwaves numerically simulates blast waves in three-dimensional transonic transient laminar viscous flow. The algorithm implemented is an un-factored solver for the implicit solution of the compressible Navier-Stokes equations using the Bi-CGstab algorithm, which solves systems of non-symmetric linear equations iteratively. The initial configuration of the blast waves problem consists of a high pressure and density region at the center of a cubic cell of a periodic lattice, with low pressure and density elsewhere. Periodic boundary conditions are applied to the array of cubic cells forming an infinite network. Initially, the high-pressure volume begins to expand in the radial direction as classical shock waves. At the same time, the expansion waves move to fill the void at the center of the cubic cell. When the expanding flow reaches the boundaries, it collides with its periodic images from other cells, thus creating a complex structure of interfering nonlinear waves. These processes create a nonlinear damped periodic system with energy being dissipated in time. Finally, the system will come to an equilibrium and steady state.

This test had the largest write-back rate at 45%, leading to an overall speed-up of 19% with no I/O, 32% with 1.2GB/s of I/O, and 33% with 18.GB/s of I/O. The queuing delay on the system bus was reduced by 98%, 85%, and 78% for the three cases. The maximum number of cycles used was reduced by 86%, 83%, and 83%, and the number of instructions requiring more than 1000 cycles was reduced by 100%, 78%, and 44%. This test demonstrated the usefulness of the secondary bus for applications with high write-back rates.

5.4.3. 433.milc

The MILC Code is a set of codes developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four-dimensional lattice gauge theory on MIMD parallel machines, and is used for millions of node hours at DOE and NSF supercomputer centers. 433.milc in SPEC CPU2006 uses the serial version of the su3imp program. The single processor version of this application is important and relevant, because parallel performance depends on good single processor performance. The program generates a gauge field, and is used in lattice gauge theory applications involving dynamical quarks. Lattice gauge theory involves the study of some of the fundamental constituents of matter, namely quarks and gluons.

This test has a 45% write-back rate, but the overall speed-up obtained with the secondary bus was only 5% with no I/O, 6% with 1.2GB/s of I/O, and 7% with 1.8GB/s of I/O. The small cycles per instruction of the base system… The queuing delay on the system bus was reduced by 71%, 68%, and 67% for the three cases. The maximum number of instructions used was reduced by 34%, 52%, and 62%. The number of instructions requiring more than 1000 cycles was not affected, since this test did not have any instructions requiring more than 1000 cycles in the base architecture.

5.4.4. 434.zeusmp

434.zeusmp is based on ZEUS-MP, a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, University of Illinois at Urbana-Champaign) for the simulation of astrophysical phenomena. ZEUS-MP solves problems in three spatial dimensions with a wide variety of boundary conditions. The program solves the equations of ideal (non-resistive), non-relativistic, hydrodynamics and magnetohydrodynamics, including externally applied gravitational fields and self-gravity. The physical problem solved in SPEC CPU2006 is a 3-D blastwave simulated with the presence of a uniform magnetic field along the x-direction. The original ZEUS-MP is based on ZEUS-3D and parallelized using the MPI message-passing library; for SPEC CPU2006, the MPI calls have been removed to create the single processor version 434.zeusmp.

This test has a 12% write-back rate, and the overall speed-up with the secondary bus was 8% with no I/O, 19% with 1.2GB/s of I/O, and 21% with 1.8GB/s of I/O. These speed-up values are consistent with the relatively small number of write-backs and subsequent use of the secondary bus. The queuing delay on the system bus was reduced by 99%, 94%, and 92% for the three I/O variations. The maximum number of cycles used was reduced by 86%, 77%, and 75%, and the number of instructions requiring more than 1000 cycles was reduced by 99%, 62%, and 52%.

5.4.5. 435.gromacs

435.gromacs is derived from GROMACS, a versatile package that performs molecular dynamics by simulating the Newtonian equations of motion for systems with hundreds to millions of particles. Although it is primarily designed for biochemical molecules such as proteins and lipids that have many complicated bonded interactions, GROMACS is also extremely fast at calculating the non-bonded interactions that usually dominate the simulation cost. Therefore, many groups are also using it for research on non-biological systems, such as polymers. The benchmark version performs a simulation of the protein Lysozyme in a solution of water and ions. The structure of a protein is normally determined by experimental techniques such as X-ray crystallography of NMR spectroscopy. By simulating the atomic motions of these structures, one can gain significant understanding of protein dynamics and function, and, in some cases, it might even be possible to predict the structure of new proteins. A dodecahedron-shaped box is used to reduce the amount of solvent water, but there are still 23179 atoms in the system. The simulation time is completely dominated by the inner loops where the non-bonded Lennard-Jones and Coulomb interactions are calculated between atoms that are closer than 1.4 nm in space.

This test has a write-back rate of less than 2%, leading to an overall speed-up of less that 1% with no I/O, and just over 1% when I/O was present. The queuing delay on the system bus was reduced by 95% when I/O was not present, 66% with 1.2GB/s of I/O, and 62% with 1.8GB/s of I/O. The maximum number of cycles was reduced by 93% with no I/O, and 73% when I/O was present. The number of instructions requiring more than 1000 cycles was reduced by 100%, 96%, and 92% for the three cases. This test indicates once again the improvements given by the secondary bus even when it not used extensively.

5.4.6. 436.cactusADM

CactusADM is a combination of Cactus, an open source problem-solving environment, and BenchADM, a computational kernel representative of many applications in numerical relativity. CactusADM solves the Einstein evolution equations, which describe how space-time curves in response to its matter content, and are a set of ten-coupled nonlinear partial differential equations, in their standard ADM 3+1 formulation.

This test has a 27% write-back rate. An overall speed-up of 5% with no I/O present, 12% with 1.2GB/s of I/O, and 14% with 1.8GB/s of I/O was seen with the secondary bus microarchitecture. The queuing delay on the system bus was reduced by 88%, 65%, and 61%. The maximum number of cycles used was reduced by 74%, 64%, and 68%. The number of instructions requiring more than 1000 cycles was reduced by 99%, 18%, and 19%.

5.4.7. 437.leslie3d

437.leslie3d is derived from LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D), a research-level Computational Fluid Dynamics (CFD) code. It is the primary solver used to investigate a wide array of turbulence phenomena such as mixing, combustion, acoustics and general fluid mechanics. For CPU2006, the program was set up to solve a test problem that represents a subset of such flows, namely the temporal mixing layer. This type of flow occurs in the mixing regions of all combustors that employ fuel injection. LESlie3d uses a strongly-conservative, finite-volume algorithm with the MacCormack Predictor-Corrector time integration scheme. The accuracy is fourth-order spatially and second-order temporally. The SPEC version performs limited file I/O to isolate the workload to CPU and memory subsystems.

The write-back rate for this test is 31%. The overall speed-up with the secondary bus is only 3% with no I/O and 4% when I/O is present. The queuing delay on the system bus was reduced by 89% when no I/O was present, 81% with 1.2GB/s of I/O, and 79% with 1.8 GB/s of I/O. The maximum number of cycles was reduced by 31%, 34%, and 35% for the three cases. The number of instructions requiring more than 1000 cycles was not affected, since this test did not have any instructions requiring more than 1000 cycles in the base architecture.

5.4.8. 447.dealII

447.dealII uses a C++ program library targeted at adaptive finite elements and error estimation. The library uses state-of-the-art programming techniques of the C++ programming language, including the Boost library. It offers a modern interface to the complex data structures and algorithms and enables use of a variety of finite elements in one, two, and three space dimensions, as well as time-dependent problems. The deal.II library provides the application programmer with grid handling and refinement, handling of degrees of freedom, input of meshes and output of results in graphics formats. Also, support for several space dimensions at once is included in a way that allows programs to be independent of the space dimension, without unreasonable penalties on run-time and memory consumption. The test case for the benchmark version, 447.dealII, solves an equation (a Helmholtz-type equation with non-constant coefficients) that is at the heart of solvers for a wide variety of applications. For example, solvers for incompressible fluid flow, static or time-harmonic electromagnetics, static and quasi-static elasto-plasticity, general relativity, and implicit time stepping schemes for seismic, acoustic, and electromagnetic applications spend most of their computing time solving one form or other of this equation. The code uses modern adaptive methods based on duality weighted error estimates to generate optimal meshes. The equation is solved in 3d.

This test has a 20% write-back rate, and the secondary bus provides an overall system speed-up of 15% when no I/O is present, 18% with 1.2GB/s of I/O, and 19% with 1.8 GB/s of I/O. The queuing delay on the secondary bus was reduced by 7% without I/O and 2% when I/O was present, giving this test the smallest change in that metric. The maximum number of cycles was reduced by 93%, 70%, and 64% for the three cases. The number of cycles requiring more than 1000 cycles to execute was reduced by 99%, 99%, and 98%.

5.4.9. 450.soplex

450.soplex is based on SoPlex Version 1.2.1, which solves a linear program using the Simplex algorithm. The LP is given as a sparse m by n matrix A, together with a right hand side vector b of dimension m and an objective function coefficient vector c of dimension n. In general, the problem is to find the vector x to:

minimize c'x

subject to Ax <= b

with   x >= 0 .

In practice, x may also have upper bounds and the A(i,.)x <= b(i) constraints could also be greater-than-or-equal-to constraints or equality constraints (where A(i,.) is row i of the matrix A). Note that the matrix A is rather sparse in practice. Therefore SoPlex, like most other implementations of the simplex algorithm, employs algorithms for sparse linear algebra, in particular a sparse LU-Factorization and appropriate solving routines for the resulting triangular equation systems.

This test has a 22% write-back rate. The secondary bus generated an 18% speed-up with no I/O and 25% when I/O was present. The system bus queuing delay was reduced by 91% when no I/O was present, 70% for 1.2GB/s of I/O, and 65% for 1.8GB/s of I/O. The maximum number of cycles used by any instruction was reduced by 89%, 80%, and 72% for the three cases. The number of instructions requiring more than 1000 cycles was reduced by 91%, 24%, and 17%.

5.4.10. 458.jseng

458.sjeng is based on Sjeng 11.2, a program that plays chess and several chess variants. It attempts to find the best move via a combination of alpha-beta or priority proof number tree searches, advanced move ordering, positional evaluation and heuristic forward pruning. Practically, it will explore the tree of variations resulting from a given position to a given base depth, extending interesting variations but discarding doubtful or irrelevant ones. From this tree the optimal line of play for both players is determined, as well as a score reflecting the balance of power between the two. The SPEC version is an enhanced version of the free Sjeng 11.2 program, modified to be more portable and more accurately reflect the workload of current professional programs.

The write-back rate for this test was very small (less than 1%), with a corresponding overall system speed-up of less than 1%. The queuing delay on the system bus was reduced by 98%, 80%, and 77%. The maximum number of cycles was reduced by 90%, 79%, and 75%. The number of cycles requiring more than 1000 cycles was reduced by 88%, 71%, and 65%.

5.4.11. 459.GemsFDTD

GemsFDTD solves the Maxwell equations in 3D in the time domain using the finite-difference time-domain (FDTD) method. The radar cross section (RCS) of a perfectly conducting (PEC) object is computed. GemsFDTD is a subset of the code GemsTD developed in the General ElectroMagnetic Solvers (GEMS) project. The code consists of three steps, initialization, time stepping and post-processing. More than 99% of the time is spent in the time stepping. The core of the FDTD method is second-order accurate central-difference approximations of the Faraday's and Ampere's laws. These central-differences are employed on a staggered Cartesian grid resulting in an explicit finite-difference method. An incident plane wave is generated using so-called Huygens' surfaces. This means that the computational domain is split into a total-field part and a scattered field part, where the scattered field part surrounds the total-field part. The computational domain is truncated by an absorbing layer in order to minimize the artificial reflections at the boundary. The Uni-axial perfectly matched layer (UPML) by Gedney is used here. A time-domain near-to-far-field transformation computes the RCS according to the Martin and Pettersson.

This test has a large write-back rate of 36%. The overall system speed-up was 19% with no I/O, 30% with 1.2GB/s of I/O, and 31% with 1.8GB/s of I/O. The system bus queuing delay was reduced by 97%, 86%, and 82% for the three cases. The maximum number of cycles was reduced by 88%, 80%, and 77%. The number of instructions requiring more than 1000 cycles was reduced by 99%, 46%, and 36%.

5.4.12. 462.libquantum

libquantum is a library for the simulation of a quantum computer. Quantum computers are based on the principles of quantum mechanics and can solve certain computationally hard tasks in polynomial time. In 1994, Peter Shor discovered a polynomial-time algorithm for the factorization of numbers, a problem of particular interest for cryptanalysis, as the widely used RSA cryptosystem depends on prime factorization being a problem only to be solvable in exponential time. An implementation of Shor's factorization algorithm is included in libquantum. libquantum provides a structure for representing a quantum register and some elementary gates. Measurements can be used to extract information from the system. Additionally, libquantum offers the simulation of decoherence, the most important obstacle in building practical quantum computers. It is thus not only possible to simulate any quantum algorithm, but also to develop quantum error correction algorithms. As libquantum allows you to add new gates, it can easily be extended to fit the ongoing research, e.g. it has been deployed to analyze quantum cryptography.

The write-back rate of 36% made good use of the secondary bus and lead to an overall speed-up of 18% with no I/O and 20% when I/O was present. The queuing delay on the system bus was reduced by 99%, 84%, and 82%. The maximum number of cycles used was reduced by 93% 87%, and 85%, and the number of instructions using more than 1000 cycles was reduced to nearly zero in all three cases.

5.4.13. 464.h264ref

464.h264ref is a reference implementation of H.264/AVC (Advanced Video Coding), the latest state-of-the-art video compression standard. This standard replaces the currently widely used MPEG-2 standard, and is being applied for applications such as the next-generation DVDs (Blu-ray and HD DVD) and video broadcasting. The 464.h264ref source code, which is part of SPEC CPU2006, is based on version 9.3 of the h264avc reference implementation downloaded from Karsten Sühring's website. The original sources have been modified to ensure portability, validation, and fairness across multiple hardware and software platforms.

The write-back rate of less than 1% lead to an overall speed-up of less than 1%. The queuing delay on the system bus was reduced by 95% with no I/O, 80% with 1.2GB/s of I/O, and 75% with 1.8GB/s of I/O. The maximum number of cycles used was reduced by 86%, 83%, and 79%. The number of instructions requiring more than 1000 cycles was reduced by 77%, 50%, and 41%.

5.4.14. 471.omnetp

This benchmark performs discrete event simulation of a large Ethernet network, based on the OMNeT++ discrete event simulation system (www.omnetpp.org). OMNeT++'s primary application area is the simulation of communication networks, but its generic and flexible architecture allows for its use in other areas such as the simulation of IT systems, queuing networks, hardware architectures or business processes as well. For the reference workload, the simulated network models a large Ethernet campus backbone, with several smaller LANs of various sizes hanging off each backbone switch. The model contains altogether about 8000 computers (hosts), and 900 switches and hubs. It mixes all kinds of Ethernet technology: Gigabit Ethernet, 100Mb full duplex, 100Mb half duplex, 10Mb UTP, 10Mb bus ("thin Ethernet"), switched hubs, repeating hubs.

This test's 17% write-back rate led to an 11% overall speed-up with no I/O, 21% with 1.2GB/s of I/O, and 22% with 1.8 GB/s of I/O. The system bus queuing delay was reduced by 99%, 89%, and 86% for the three cases. The maximum number of cycles used was reduced by 92%, 83%, and 81%, and the number of instructions requiring more than 1000 cycles was reduced by 100%, 71%, and 63%.

5.4.15. 471.astar

471.astar is derived from a portable 2D path-finding library that is used in game AI. This library implements three different path-finding algorithms: First is the well known A* algorithm for maps with passable and non-passable terrain types. Second is a modification of the A* path finding algorithm for maps with different terrain types and different move speed. Third is an implementation of A* algorithm for graphs that is formed by map regions with neighborhood relationship. The library also includes pseudo-intellectual functions for map region determination.

The write-back rate of this test was very small (less than 1%), leading to an overall speed-up with the secondary bus of less than 1% with no I/O, and just over 1% when I/O was present. The queuing delay on the system bus was reduced by 96%, 87%, and 84% for the three cases. The maximum number of cycles used was reduced by 92%, 84%, and 81%. The number of instructions requiring more than 1000 cycles was reduced by 99% with no I/O and 78% with I/O.

5.4.16. 482.sphinx3

Sphinx-3 is a widely known speech recognition system from Carnegie Mellon University. CMU supplies a program known as livepretend, which decodes utterances in batch mode, but otherwise operates as if it were decoding a live human. In particular, it starts from raw audio, not from an intermediate format. Although in real life I/O efficiency is important to any speech recognition system, the SPEC version concentrates on the CPU-intensive portions of the task.

This test has a large write-back rate of 35%. The overall system speed-up using the secondary bus is 14% when I/O is not present, 29% with 1.2GB/s of I/O, and 30% with 1.8GB/s of I/O. The queuing delay on the system bus was reduced by 98%, 90%, and 86% in the three cases. The maximum number of cycles used was reduced by 92%, 82%, and 79% for the three cases, and the number of instructions requiring more than 1000 cycles was reduced by 99% with no I/O and 97% when I/O was present.

Chapter 6

Analysis of Power Consumption with a Secondary Bus

6.1. Introduction

In embedded and real-time processing systems and increasingly in higher-end systems, power consumption is often as important as performance. Modern embedded devices require low power in order to improve battery life and increase product reliability. Assuming a goal of obtaining minimum power at a given performance level, the secondary bus microarchitecture has clear benefits when applied to systems that require low energy consumption. The single most efficient method of reducing the power consumption of a processor is to reduce the voltage [26]. A reduction in voltage leads to a reduction in frequency. The increase in instructions per cycle measured when the secondary bus is present shows that when the secondary bus is used, a low-power system may reduce its clock rate to reduce power while still maintaining a high level of processing power. By reducing the voltage along with the clock rate, energy consumption is reduced. This makes the secondary bus microarchitecture a good choice for low-power systems.

6.2. Instructions per Cycle Analysis

The instructions per cycle that were measured for the SPEC CPU2006 benchmarks are shown in Table 24 and Figure 17, and show an increase in the IPC of up to 18%.

Table 24: Instructions per Cycle

Test	Original IPC	Secondary Bus IPC	IPC Change

perlbench	0.32	0.32	0.25%
bwaves	0.22	0.27	18.61%
milc	0.37	0.39	4.80%
zeusmp	0.35	0.38	6.96%
gromacs	0.42	0.42	0.36%
cactusADM	0.38	0.40	4.77%
leslie	0.40	0.41	2.87%
deal	0.27	0.29	7.82%
soplex	0.20	0.24	16.47%
sjeng	0.38	0.38	0.05%
GemsFDTD	0.25	0.31	18.54%
libquantum	0.23	0.27	16.27%
h264ref	0.34	0.35	0.08%
omnetpp	0.24	0.26	10.83%
astar_rivers	0.39	0.39	0.31%
sphinx	0.28	0.32	13.59%

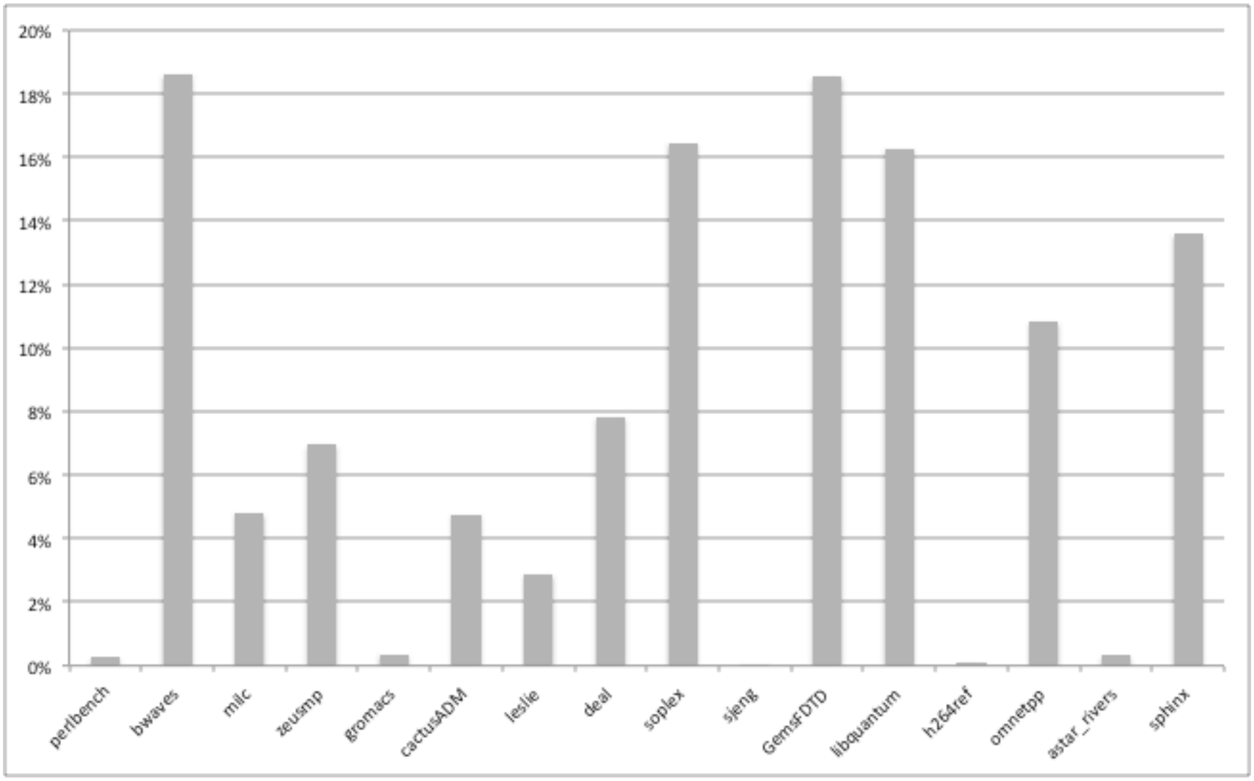


Figure 17: Change in Instructions per Cycle

When I/O is included, the secondary bus microarchitecture once again shows improvements. Table 25 and Figure 18 show the change in instructions per cycle when 1.2GB/s of I/O are present of up to 31%. When the I/O is increased to 1.8GB/s, increases in IPC of up to 32% are seen, as show in Table 26 and Figure 19.

Table 25: Instructions per Cycle w/ 1.2GB/s I/O

Test	Original IPC	Secondary Bus IPC	IPC Change
perlbench	0.31	0.32	0.81%
bwaves	0.11	0.17	31.86%
milc	0.35	0.38	6.12%
zeusmp	0.28	0.33	15.22%
gromacs	0.41	0.42	1.09%
cactusADM	0.30	0.34	12.15%
leslie	0.39	0.40	3.79%
deal	0.20	0.21	3.87%
soplex	0.09	0.12	21.13%
sjeng	0.38	0.38	0.12%
GemsFDTD	0.13	0.18	28.05%
libquantum	0.10	0.12	16.25%
h264ref	0.34	0.34	0.20%
omnetpp	0.14	0.17	21.32%
astar_rivers	0.39	0.39	0.81%
sphinx	0.18	0.25	29.09%

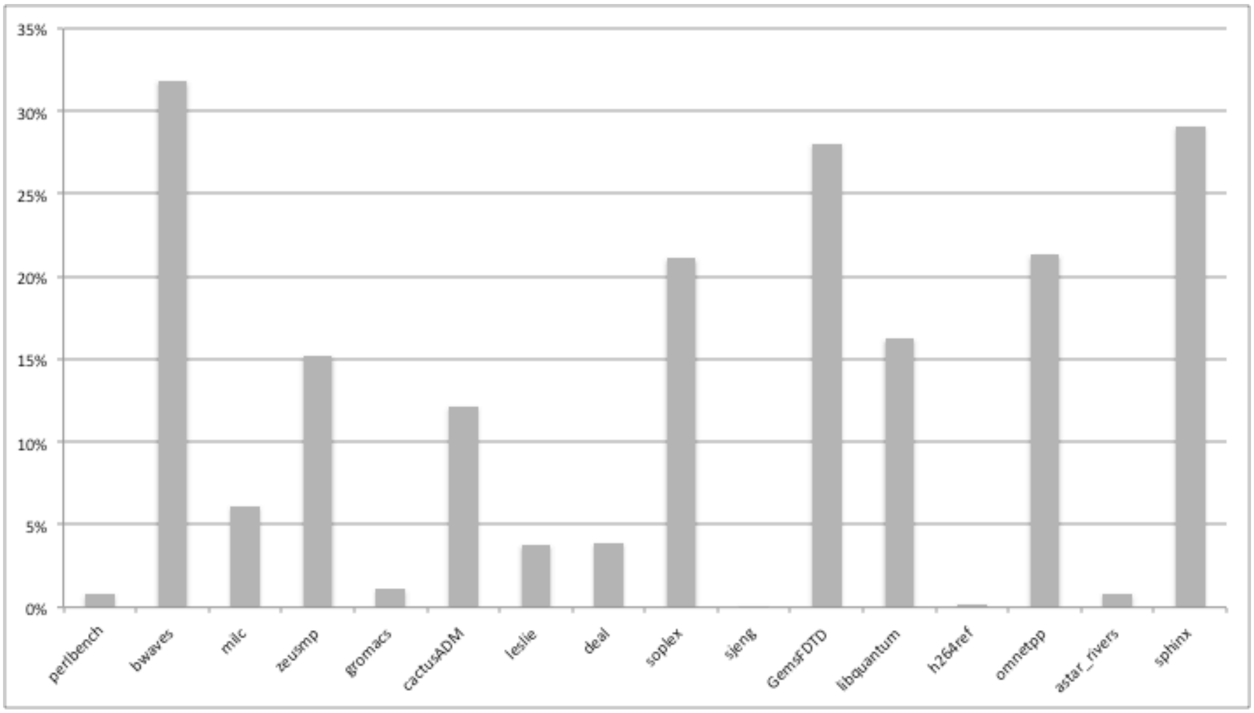


Figure 18: IPC Change with 1.2GB/s I/O

Table 26: Instructions per Cycle with 1.8GB/s I/O

Test	Original IPC	Secondary Bus IPC	Change
perlbench	0.31	0.31	0.96%
bwaves	0.10	0.14	32.39%
milc	0.35	0.37	6.59%
zeusmp	0.26	0.31	16.55%
gromacs	0.41	0.42	1.29%
cactusADM	0.28	0.32	13.77%
leslie	0.38	0.40	4.22%
deal	0.19	0.20	3.36%
soplex	0.08	0.10	20.90%
sjeng	0.38	0.38	0.14%
GemsFDTD	0.12	0.16	29.05%
libquantum	0.09	0.10	16.25%
h264ref	0.34	0.34	0.22%
omnetpp	0.12	0.15	22.53%
astar_rivers	0.38	0.39	0.96%
sphinx	0.15	0.22	30.10%

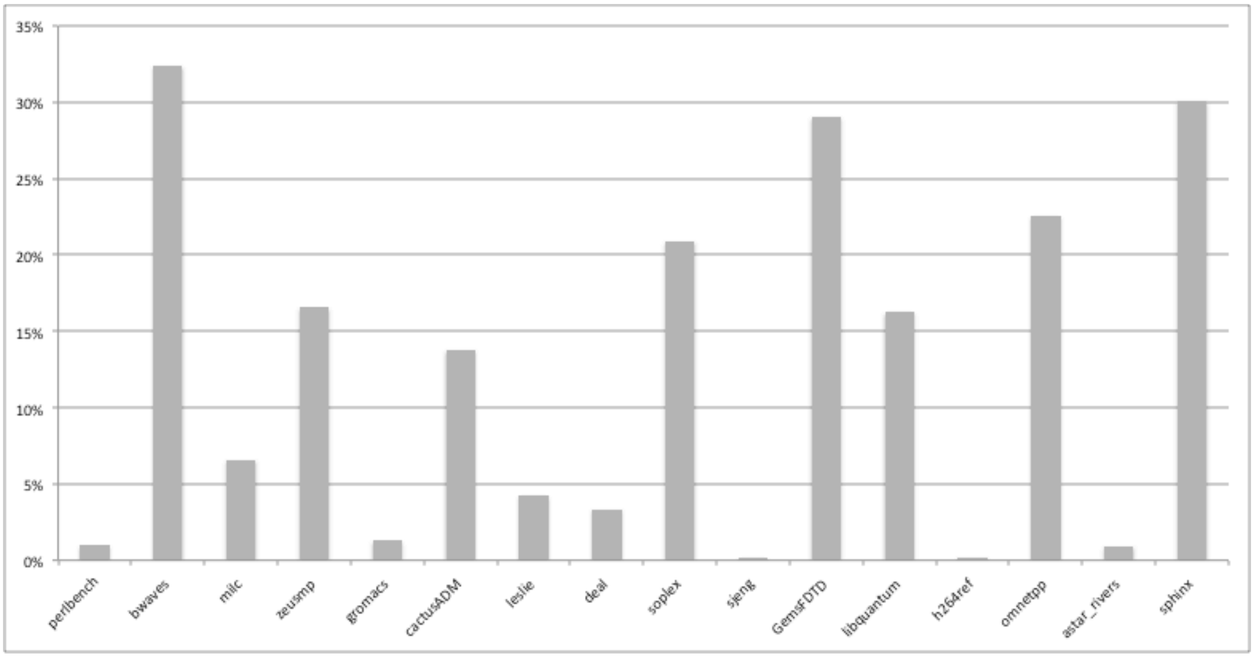


Figure 19: IPC Change with 1.8GB/s I/O

Energy efficiency in processors is typically measured using the energy-delay product (EDP) or energy-delay-square product (ED^2P), and expressed in terms of MIPS/W or $MIPS^2/W$, respectively. These metrics combine the total energy consumed to perform a unit of work with the execution time. Since the execution time is squared in ED^2P, it puts more emphasis on the execution time. The measurements quantify energy efficiency in a single value, and take into account the loss of processing power that generally accompanies a reduction in power usage. In a system using the secondary bus architecture, it is possible to reduce the frequency, and therefore the power consumption, without reducing the system performance. This leads to an increase in the MIPS/W or $MIPS^2/W$ values of the system.

6.3. Reduced Power Consumption

The runtime of a program is computed as:

*Runtime = Instructions * Cycles/Instruction * Seconds/Cycle*

When the secondary bus is present, the cycles/instruction value is decreased (see Table 17). Because of the decrease in CPI, the runtime of the program or the system throughput in MIPS can be kept constant by reducing the clock frequency. Table 27 shows the reduced clock frequency that can be used for the SPEC benchmarks when the secondary bus is present without changing the program runtimes.

Table 27: Frequency change with secondary bus

Test	Frequency with Secondary Bus (GHz)	Decrease
perlbench	2.00	0.25%
bwaves	1.62	18.83%
milc	1.90	4.89%
zeusmp	1.84	7.79%
gromacs	1.99	0.38%
cactusADM	1.90	4.78%
leslie	1.94	2.89%
deal	1.70	14.90%
soplex	1.65	17.57%
sjeng	2.00	0.10%
GemsFDTD	1.62	18.93%
libquantum	1.64	18.02%
h264ref	2.00	0.08%
omnetpp	1.78	10.91%
astar_rivers	1.99	0.42%
sphinx	1.73	13.61%

The dynamic power consumption in a processor is given by

$$P = C * V^2 * F$$

Where C is the capacitance, V is the voltage, and F is the clock frequency. Consider an example processor with a dynamic power of about 65 watts, which is comparable to that of an Intel Pentium Core 2 processor. If the system clock frequency is 2 GHz, and the voltage is 2.8 volts, then a capacitance is 4.2×10^{-9} coulombs/volt would lead to power consumption of 65.86 watts. By adding a secondary bus to that example system, the frequency may be reduced as shown in Table 27 without reducing program runtimes. This reduction in frequency leads to the power usage reduction show in Table 28.

Table 28: Decrease in power usage with secondary bus

Test	Power with secondary bus (watts)	Decrease
perlbench	65.69	0.25%
bwaves	53.46	18.83%
milc	62.63	4.89%
zeusmp	60.73	7.79%
gromacs	65.60	0.38%
cactusADM	62.71	4.78%
leslie	63.95	2.89%
deal	56.05	14.90%
soplex	54.29	17.57%
sjeng	65.79	0.10%
GemsFDTD	53.39	18.93%
libquantum	53.99	18.02%
h264ref	65.80	0.08%
omnetpp	58.67	10.91%
astar_rivers	65.58	0.42%
sphinx	56.90	13.61%

This reduction in power consumption leads to a corresponding energy savings when the secondary bus is present.

Chapter 7

Conclusion

A novel idea to reduce latencies due to bus contentions on the main system bus between the CPU and memory has been evaluated. The technique of introducing an additional, low-bandwidth, secondary bus into a microarchitecture, and using that bus specifically for cache write-backs has been shown to give significant performance improvements compared to existing architectures. Simulations carried out using the widely accepted SimpleScalar toolset demonstrate the feasibility and advantages of the secondary bus. A microarchitecture using a small cache size and simple benchmarks showed that the ability of the secondary bus to reduce queuing delays on the system bus could significantly improve worst-case execution times in real-time systems. The small cache system also demonstrated the ability of the secondary bus microarchitecture to approach the optimal improvement level represented by free write-backs. Additional simulations with large cache sizes, comparable to those in current commercial processors, and the industry-standard SPEC CPU2006 benchmark suite demonstrated the further advantages of the secondary bus microarchitecture to improve determinacy and worst-case execution time in real-time systems, decrease power consumption, as well as increase overall system performance. Additional simulations with I/O injections to replicate a real world scenario where there will be a number of communications between the peripherals and the processor not involving the memory indicate that the secondary bus microarchitecture advantages are viable in that instance as well.

The major advantage offered by the secondary bus microarchitecture derives from the reduction in queuing delay on the system bus. The queuing delay reduction of 6% to 99%, with an average of 87%, leads to a corresponding reduction in the maximum execution time of any instruction, and in the number of instructions that require more than 1000 cycles to execute. These results indicate that the secondary bus microarchitecture is an excellent choice for real-time, embedded system applications. In addition, the secondary bus microarchitecture generates an overall system speed-up of up to 19% when no I/O is present, and of up to 33% in the presence of external I/O. The speed-up achieved suggests an additional application, in which the new microarchitecture is used in low-power systems where the energy consumption may be reduced while maintaining consistent processing throughput.

Current commercial microprocessors contain various mechanisms designed to reduce cache access latencies and minimize processor delays caused by cache misses. None of those mechanisms includes a secondary path to allow dirty cache lines to be more efficiently moved back into memory. The secondary bus method can therefore be used in addition to those mechanisms to present even further improvements. Since current multi-core processors tend to contain separate level-1 caches but share the level-2 cache among all cores, the secondary bus method, which provides a connection between the L2 cache and memory, is applicable to multi-core systems as well.

The secondary bus can be implemented in many ways. The simulated system used an 8-bit-wide bus to evaluate the design for different traffic intensities. As future work, various other implementations of the secondary bus can be evaluated. One such design could be of a serial line using a high speed signaling mechanism for fast data transfer. Split bus or pipelined transactions can be tried on the secondary bus with multiple bit lines [25]. The number of bit lines that can be used for the secondary bus depends on the L2 cache write-back rate. A smaller cache can result in more write-backs and may require a wider bus for transferring data faster to the memory. The secondary bus provides benefits for single ported memories at a cost of a small hardware addition for controlling the bus access and a smaller bus compared to the main bus. A truly novel implementation of the secondary bus is through the use of wireless communication [3]. Implementation of the secondary bus with a wireless interconnect could lead to a large number of additional applications that would be permitted by monitoring the wireless signal using external devices.

References

[1] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, pp. 59-67, February 2002.

[2] S. Baskiyar, "A New Technique to Solve Cache Coherence", Tech Report TR CSSE03-06, Dept. of Comp. Science and Software Engineering, June 25, 2003.

[3] S. Baskiyar, "Wireless Techniques in Architecture and Fault Tolerance," National Science Foundation Proposal, 2004.

[4] S. Baskiyar and C. Wang, "A secondary channel between cache and memory for decreasing queuing delay," US Provisional patent application filed no. 61/003,542 on Nov 17, 2007, Auburn University, AL.

[5] D. Burger, A. Kägi, and M. Hrishikesh, "Memory Hierarchy Extensions to the SimpleScalar Tool Set," Technical Report TR-99-25, Department of Computer Sciences, University of Texas at Austin, December 1999.

[6] M. F. Chang, V.P. Roychowdhury, L. Zhang, H. Shin, and Y. Qian. "RF/Wireless Interconnect for Inter- and Intra-chip Communications," *Proceedings of the IEEE*, Volume 89, Issue 4, pp. 456-466, April 2001.

[7] P.P. Chu and R. Gottipati, "Write Buffer Design for On-Chip Cache," *Proceedings, IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 311-316,October 1994.

[8] Compaq Computer Corporation, *Alpha 21264 Microprocessor Hardware Reference Manual*, ftp://ftp.digital.com/pub/Digital/info/semiconductor/literature/dsc-library.html, July 1999.

[9] R. Desikan, D. Burger, S.W. Keckler, and T.M. Austin, "Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator," Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.

[10] K. Ganesan, D. Panwar and L. K. John, "Generation, Validation and Analysis of SPEC CPU2006 Simulation Points Based on Branch, Memory and TLB Characteristics", *Proceedings of the SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, Section: Modeling and Sampling Techniques, pp. 121-137, Jan. 2009.

[11] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th Edition, Morgan Kaufmann Publishing Co., Menlo Park, CA. 2007.

[12] IEEE Standard for a High Performance Serial Bus, http://standards.ieee.org/reading/ieee/std/busarch/1394-1995.pdf, 1995.

[13] J. Kim, Z. Xu, and M. F. Chang, "Reconfigurable Memory Bus Systems Using Multi-Gbps/pin CDMA I/O Transceivers," *Proceedings of the 2003 International Symposium on Circuits and Systems*, Volume 2, pp. II-33 - II-36, May 2003.

[14] D. Kroft, "Lockup-Free Instruction Fetch/Prefetch Cache Organization," *Proceedings of the 8th Annual Symposium on Computer Architecture*, Minneapolis, Minnesota, United States, pp. 81-87, 1981.

[15] H.H.S. Lee, G. Tyson, and M. Farrens, "Eager Writeback - A Technique for Improving Bandwidth Utilization," *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, Monterey, California, United States, p.11-21, December 2000.

[16] N. Miura, D. Mizoguchi,, T. Sakurai, and T. Kuroda, "Analysis and Design of Inductive Coupling and Transceiver Circuit for Inductive Inter-chip Wireless Superconnect," *IEEE Journal of Solid-State Circuits*, Volume 40 , Issue 4, pp. 829 – 837, April 2005.

[17] J. O'Farrell and S. Baskiyar. "Improved Real-Time Performance Using a Secondary Bus," *Proceedings of the ISCA 25th International Conference on Computers and Their Applications*, CATA 2010, pp. 31-36, Honolulu, Hawaii, USA.

[18] J. O'Farrell, R. Venkatesh, and S. Baskiyar, "Secondary Bus Performance in Retiring Cache Write-backs to Memory", 26th International Symposium on Computer and Information Sciences, London, UK, 26-28 September 2011.

[19] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder. 2003. "Using SimPoint for accurate and efficient simulation," *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '03)*. ACM, pp. 318-319, New York, NY, USA.

[20] V. Raghunathan, M.B. Srivastava, and R.K. Gupta, "A Survey of Techniques for Energy Efficient On-chip Communication," *Proceedings Design Automation Conference, 2003*, pp. 900 – 905, June 2003.

[21] SPEC CPU2006, http://www.spec.org/cpu2006.

[22] SimpleScalar, LLC, http://simplescalar.com.

[23] USB 2.0 Specification, http://www.usb.org/developers/docs, April 2000.

[24] R. Van Nee, V.K. Jones, G. Awater, A. Van Zelst, J. Gardner, G. Steele, "The 802.11n MIMO-OFDM Standard for Wireless LAN and Beyond," *Wireless Personal Communications*, Volume 37, Numbers 3-4, pp. 445-453, May, 2006.

[25] R. Venkatesh, *Secondary Bus Performance in Reducing Cache Writeback Latency*, Master's thesis, Auburn University, 2011.

[26] M. Weiser, B. Welch, A. Demer, and S. Shenker, :Scheduling for Reduced CPU Energy," *The First USENIX Symp. on Operating Systems Design and Implementation (OSDI'94)*, pp. 13–23, 1994.

[27] *What is SimpleScalar*, http://www.ecs.umass.edu/ece/koren/architecture/Simplescalar/SimpleScalar_introduction.ht m

Appendix A

Simulation Configuration Values

A.1Sim-alpha Configuration for Base System

```
-fetch:ifqsize 4
-fetch:width 4
-fetch:speed 1
-slot:width 4
-map:width 4
-issue:intwidth 4
-issue:fpwidth 2
-commit:width 11
-res:iclus 2
-res:ialu 4
-res:imult 4
-res:fpclus 1
-res:fpalu 1
-res:fpmult 1
-res:delay 1
-mach:freq 3000000000
-bpred:ras 32
-line_pred:ini_value 0
-reg:int_p_regs 41
-reg:fp_p_regs 41

# cache configuration
-cache:define       DL1:64:64:0:8:F:3:vipt:0:1:0:Onbus
-cache:define       IL1:64:64:0:8:l:3:vipt:0:1:0:Onbus
-cache:define       L2:4096:64:0:8:l:10:pipt:0:2:0:Membus:Wirelessbus
-cache:define       WBB:4096:64:0:8:f:10:pipt:0:1:0:Secbus

# flush caches on system calls
-cache:flush        false

-cache:writeback    true


# defines name of first-level data cache
-cache:dcache       DL1

# defines name of first-level instruction cache
```

```
-cache:icache       IL1

# number of regular mshrs for each cache
-cache:mshrs        8

# number of prefetch mshrs for each cache
-cache:prefetch_mshrs     4

# number of targets for each cache
-cache:mshr_targets 8

# bus configuration
# For original long timing runs, Onbus was set at 2 GHz, now at 4
-bus:define         Onbus:64:2:1:0:1:0:L2
-bus:define         Membus:64:5:2:0:1:0:SDRAM
-bus:define         Wirelessbus:8:5:1:0:1:0:WBB
-bus:define         Secbus:8:5:1:0:1:0:SDRAM

# memory bank configuration
-mem:define         SDRAM

# define tlbs
-tlb:define         DTLB:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:define         ITLB:1:32:0:128:l:1:vivt:0:1:0:Onbus

# data TLB config, i.e., {<config>|none}
-tlb:dtlb           DTLB

# instruction TLB config, i.e., {<config>|none}
-tlb:itlb           ITLB

-cache:addr_trap 0
-wb:diffsize_trap 0
-cache:target_trap 0
-cache:mshrfull_trap 0
-prefetch:dist 0
```

A.2Sim-alpha configuration including secondary bus

```
-fetch:ifqsize 4
-fetch:width 4
-fetch:speed 1
-slot:width 4
-map:width 4
-issue:intwidth 4
-issue:fpwidth 2
-commit:width 11
-res:iclus 2
-res:ialu 4
-res:imult 4
```

```
-res:fpclus 1
-res:fpalu 1
-res:fpmult 1
-res:delay 1
-mach:freq 3000000000
-bpred:ras 32
-line_pred:ini_value 0
-reg:int_p_regs 41
-reg:fp_p_regs 41

# cache configuration
-cache:define       DL1:64:64:0:8:F:3:vipt:0:1:0:Onbus
-cache:define       IL1:64:64:0:8:l:3:vipt:0:1:0:Onbus
-cache:define       L2:4096:64:0:8:l:10:pipt:0:2:0:Membus:Wirelessbus
-cache:define       WBB:4096:64:0:8:f:10:pipt:0:1:0:Secbus

# flush caches on system calls
-cache:flush        false

-cache:writeback    true


# defines name of first-level data cache
-cache:dcache       DL1

# defines name of first-level instruction cache
-cache:icache       IL1

# number of regular mshrs for each cache
-cache:mshrs        8

# number of prefetch mshrs for each cache
-cache:prefetch_mshrs     4

# number of targets for each cache
-cache:mshr_targets 8

# bus configuration
# For original long timing runs, Onbus was set at 2 GHz, now at 4
-bus:define         Onbus:64:2:1:0:1:0:L2
-bus:define         Membus:64:5:2:0:1:0:SDRAM
-bus:define         Wirelessbus:8:5:1:0:1:0:WBB
-bus:define         Secbus:8:5:1:0:1:0:SDRAM

# memory bank configuration
-mem:define         SDRAM

# define tlbs
-tlb:define         DTLB:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:define         ITLB:1:32:0:128:l:1:vivt:0:1:0:Onbus

# data TLB config, i.e., {<config>|none}
-tlb:dtlb           DTLB
```

```
# instruction TLB config, i.e., {<config>|none}
-tlb:itlb           ITLB

-cache:addr_trap 0
-wb:diffsize_trap 0
-cache:target_trap 0
-cache:mshrfull_trap 0
-prefetch:dist 0
```

A.3Sim-alpha configuration for free write-back

```
-fetch:ifqsize 4
-fetch:width 4
-fetch:speed 1
-slot:width 4
-map:width 4
-issue:intwidth 4
-issue:fpwidth 2
-commit:width 11
-res:iclus 2
-res:ialu 4
-res:imult 4
-res:fpclus 1
-res:fpalu 1
-res:fpmult 1
-res:delay 1
-mach:freq 3000000000
-bpred:ras 32
-line_pred:ini_value 0
-reg:int_p_regs 41
-reg:fp_p_regs 41

# cache configuration
-cache:define       DL1:64:64:0:8:F:3:vipt:0:1:0:Onbus
-cache:define       L2:4096:64:0:8:l:10:pipt:0:2:0:Membus:Wirelessbus
-cache:define       IL1:64:64:0:8:l:3:vipt:0:1:0:Onbus
-cache:define       WBB:64:64:0:8:f:10:pipt:0:1:0:Membus

# flush caches on system calls
-cache:flush        false

-cache:writeback    false

# defines name of first-level data cache
-cache:dcache       DL1

# defines name of first-level instruction cache
-cache:icache       IL1

# number of regular mshrs for each cache
```

```
-cache:mshrs        8

# number of prefetch mshrs for each cache
-cache:prefetch_mshrs     4

# number of targets for each cache
-cache:mshr_targets 8

# bus configuration
# For original long timing runs, Onbus was set at 2 GHz, now at 4
-bus:define         Onbus:64:2:4:0:1:0:L2
-bus:define         Membus:64:5:10:0:1:0:SDRAM
-bus:define         Wirelessbus:64:2:0:0:1:0:WBB

# memory bank configuration
-mem:define         SDRAM

# define tlbs
-tlb:define         DTLB:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:define         ITLB:1:32:0:128:l:1:vivt:0:1:0:Onbus

# data TLB config, i.e., {<config>|none}
-tlb:dtlb           DTLB

# instruction TLB config, i.e., {<config>|none}
-tlb:itlb           ITLB

-cache:addr_trap 0
-wb:diffsize_trap 0
-cache:target_trap 0
-cache:mshrfull_trap 0
-prefetch:dist 0
```

Appendix B

SPEC CPU2006 Simulation SimPoints

Test	Index	Point	Weight
perlbench	0	1194	0.15122
perlbench	1	1417	0.00325203
perlbench	2	236	0.131165
perlbench	3	3	0.00325203
perlbench	4	355	0.0937669
perlbench	5	692	0.148509
perlbench	6	18	0.0124661
perlbench	7	331	0.135501
perlbench	8	1245	0.00813008
perlbench	9	878	0.0861789
perlbench	10	606	0.142005
perlbench	11	53	0.0157182
bwaves	0	11211	0.03089
bwaves	1	20179	0.622676
bwaves	2	10729	0.0787782
bwaves	3	15057	0.0250226
bwaves	4	5928	0.0257129
bwaves	5	18870	0.032098
bwaves	6	6099	0.0330472
bwaves	7	2634	0.033608
bwaves	8	5275	0.0330472
bwaves	9	7183	0.0275681
bwaves	10	1219	0.030588
bwaves	11	15571	0.00664394
milc	0	1213	0.0683082
milc	1	2494	0.0463842
milc	2	2141	0.0197971
milc	3	608	0.0147251
milc	4	3927	0.0013089
milc	5	6502	0.0237238

milc	6	9397	0.0203698
milc	7	3714	0.0429483
milc	8	7689	0.0430301
milc	9	5715	0.07518
milc	10	11904	0.100867
milc	11	10398	0.0634817
milc	12	1134	0.0307592
milc	13	1713	0.125245
milc	14	824	0.0135798
milc	15	9405	0.0251145
milc	16	594	0.00466296
milc	17	11647	0.0521106
milc	18	2864	0.0463842
zeusmp	0	2323	0.026255
zeusmp	1	12021	0.0656627
zeusmp	2	4404	0.0453313
zeusmp	3	241	0.123845
zeusmp	4	18408	0.00672691
zeusmp	5	8129	0.025251
zeusmp	6	2143	0.0203815
zeusmp	7	9629	0.0942771
zeusmp	8	6085	0.013253
zeusmp	9	8524	0.0597892
zeusmp	10	13947	0.0249498
zeusmp	11	9181	0.0476406
zeusmp	12	15831	0.0345884
zeusmp	13	6180	0.0405622
zeusmp	14	17760	0.0242972
zeusmp	15	17993	0.00758032
zeusmp	16	11615	0.0684237
zeusmp	17	4233	0.0197289
zeusmp	18	9353	0.0253012
zeusmp	19	6487	0.0673193
zeusmp	20	15846	0.0381024
zeusmp	21	17154	0.0338353
zeusmp	22	812	0.0320281
zeusmp	23	9927	0.0062751
gromacs	0	11510	0.0248848
gromacs	1	21209	0.00640972
gromacs	2	5917	0.0239631
gromacs	3	7093	0.0147884

gromacs	4	14122	0.00712191
gromacs	5	17073	0.0229996
gromacs	6	8658	0.0314621
gromacs	7	1185	0.169711
gromacs	8	1684	0.0159615
gromacs	9	11106	0.0201927
gromacs	10	8285	0.00519481
gromacs	11	18060	0.0158777
gromacs	12	9077	0.105069
gromacs	13	16094	0.020863
gromacs	14	17719	0.0254713
gromacs	15	7018	0.297863
gromacs	16	2882	0.0201089
gromacs	17	6926	0.117679
cactusADM	0	36939	0.00109647
cactusADM	1	18590	0.386834
cactusADM	2	9583	0.00166663
cactusADM	3	13068	0.563957
cactusADM	4	26680	0.0202189
cactusADM	5	2	0.000109647
cactusADM	6	29070	0.0260301
leslie	0	26086	0.044533
leslie	1	23328	0.0329974
leslie	2	22804	0.0377879
leslie	3	5390	0.0541908
leslie	4	21741	0.0265588
leslie	5	6016	0.0580232
leslie	6	10878	0.0462959
leslie	7	5863	0.0220366
leslie	8	25975	0.0534626
leslie	9	12307	0.0378262
leslie	10	17230	0.0384778
leslie	11	2926	0.0456828
leslie	12	18793	0.0416587
leslie	13	12319	0.0528494
leslie	14	9987	0.0461426
leslie	15	612	0.0250642
leslie	16	4082	0.0375963
leslie	17	1594	0.0613958
leslie	18	22908	0.0264439
leslie	19	20177	0.0508566

leslie	20	4242	0.0362549
leslie	21	9534	0.0395125
namd	0	722	0.0672273
namd	1	4395	0.097221
namd	2	4909	0.0252271
namd	3	4206	0.0135804
namd	4	8674	0.0517583
namd	5	7690	0.0571544
namd	6	9974	0.04933
namd	7	10851	0.0149294
namd	8	14609	0.101808
namd	9	21538	0.0259016
namd	10	18067	0.0423599
namd	11	8	0.00620559
namd	12	2137	0.0884522
namd	13	22043	0.0125911
namd	14	19771	0.0294091
namd	15	10858	0.0420901
namd	16	17753	0.112015
namd	17	9593	0.0502293
namd	18	11040	0.0148395
deal	0	15	0.611111
soplex	0	2741	0.0444079
soplex	1	320	0.0641447
soplex	2	1705	0.0633224
soplex	3	2984	0.0565378
soplex	4	190	0.00349507
soplex	5	1954	0.045847
soplex	6	2478	0.0540707
soplex	7	1098	0.0493421
soplex	8	2831	0.0509868
soplex	9	3012	0.0550987
soplex	10	2543	0.0620888
soplex	11	18	0.0462582
soplex	12	2457	0.0310444
soplex	13	1525	0.0676398
soplex	14	1472	0.0532484
soplex	15	554	0.000616776
soplex	16	654	0.0234375
soplex	17	1205	0.0331003
soplex	18	1807	0.0904605

sjeng	0	17175	0.0940804
sjeng	1	15619	0.155222
sjeng	2	13137	0.0527653
sjeng	3	6086	0.10415
sjeng	4	10548	0.0625529
sjeng	5	30639	0.0451109
sjeng	6	4818	0.000501929
sjeng	7	21707	0.0955234
sjeng	8	5736	0.124918
sjeng	9	1805	0.06569
GemsFDTD	0	463	0.0250559
GemsFDTD	1	1835	0.0402685
GemsFDTD	2	1598	0.0205817
GemsFDTD	3	1032	0.052349
GemsFDTD	4	1118	0.0219239
GemsFDTD	5	1907	0.0161074
GemsFDTD	6	1444	0.0223714
GemsFDTD	7	61	0.0456376
GemsFDTD	8	1856	0.0178971
GemsFDTD	9	1063	0.0362416
GemsFDTD	10	1069	0.165101
GemsFDTD	11	471	0.0192394
GemsFDTD	12	185	0.0143177
GemsFDTD	13	1239	0.0286353
GemsFDTD	14	2197	0.0447427
GemsFDTD	15	552	0.0183445
GemsFDTD	16	2155	0.0326622
GemsFDTD	17	693	0.0281879
GemsFDTD	18	1842	0.177629
GemsFDTD	19	1662	0.0223714
GemsFDTD	20	984	0.0510067
GemsFDTD	21	0	0.0299776
GemsFDTD	22	651	0.0255034
libquantum	0	12424	0.0999497
libquantum	1	19813	0.0092006
libquantum	2	291	0.0152338
libquantum	3	19874	0.0025641
libquantum	4	19824	0.00191051
libquantum	5	9974	0.139367
libquantum	6	5803	0.0581699
libquantum	7	13400	0.0692308

libquantum	8	12732	0.0289593
libquantum	9	18153	0.0335847
libquantum	10	6917	0.0831574
libquantum	11	1106	0.0499246
libquantum	12	15532	0.0463047
libquantum	13	13574	0.0933132
libquantum	14	16856	0.120362
libquantum	15	15642	0.071091
libquantum	16	13674	0.0275013
h264ref	0	26559	0.068356
h264ref	1	36588	0.00321384
h264ref	2	1517	0.0918418
h264ref	3	53508	0.0906587
h264ref	4	45541	0.00722232
h264ref	5	732	0.00734593
h264ref	6	9115	0.138337
h264ref	7	29684	0.0536641
h264ref	8	13267	0.0734593
h264ref	9	44147	0.117376
h264ref	10	49954	0.156419
h264ref	11	37984	0.0593855
omnetpp	0	2872	0.115221
omnetpp	1	981	0.177559
omnetpp	2	17	0.00630223
omnetpp	3	2094	0.0293191
omnetpp	4	204	0.0287711
omnetpp	5	3093	0.0786409
omnetpp	6	4892	0.0486368
omnetpp	7	289	0.0224688
omnetpp	8	2197	0.052884
omnetpp	9	3315	0.147143
omnetpp	10	7296	0.000685025
omnetpp	11	6465	0.110015
omnetpp	12	452	0.0165776
astar_rivers	0	4	0.00186239
astar_rivers	1	6223	0.0601138
astar_rivers	2	2828	0.338024
astar_rivers	3	2068	0.0973616
astar_rivers	4	432	0.159027
astar_rivers	5	8615	0.0973616
astar_rivers	6	2452	0.168236

sphinx	0	34086	0.0760587
sphinx	1	38691	0.0986067
sphinx	2	30144	0.0412755
sphinx	3	4727	0.0461946
sphinx	4	37454	0.0282411
sphinx	5	18411	0.0563324
sphinx	6	7772	0.0563574
sphinx	7	3008	0.0257441
sphinx	8	23787	0.0443218
sphinx	9	12598	0.0504395
sphinx	10	4561	0.0163054
sphinx	11	19771	0.0477177
sphinx	12	0	0.000898921
sphinx	13	13822	0.0453206
sphinx	14	33472	0.0223482
sphinx	15	3423	0.0356073
sphinx	16	19390	0.0593038
sphinx	17	8232	0.0462445
sphinx	18	20352	0.0579555