# Performance evaluation of Simultaneous Multi-Threading (SMT) of Single Core Processors in M-Sim

**Arun Das** @01521976**,** **Dr. Wei-Ming Lin**
Department of Electrical and Computer Engineering,
University of Texas at San Antonio,
San Antonio, TX 78249-0669, USA
Date: August 2015

**Abstract**-- *Simultaneous Multi-Threading (SMT) is an effective technique to increase the Instructions-Per-Cycle (IPC) of a Superscalar Processor without considerable hardware additions [1]. SMT utilizes the available resources efficiently thereby hiding the latencies in traditional Superscalar systems. It is achieved through Hardware Multi-Threading, by utilizing key data path components like issue queue and the physical register file and write buffer permitting multiple independent threads of executions at once [3], [4]. The variations in size of these key components will change the overall throughput/IPC of the system. The project studies these variations and their impact on IPC, concluding at a holistic understanding of SMT and the main parameters that determine the overall performance of it.*

***Index Terms*—Instruction level parallelism (ILP), Issue Queue, MSIM, Register File, Simultaneous Multi-Threading (SMT), Write Buffer.**

## 1. Introduction

Over the years, commercial microprocessor designers shifted their interest from Instruction level parallelism (ILP) towards Thread Level Parallelism (TLP). This shift came with the limited instruction-level parallelism inherent to most applications [1], [8]. Numerous hardware related techniques like register renaming, out of order execution, speculative execution and software techniques like loop unrolling, trace scheduling, software pipelining by compiler have been used to improve ILP. Superscalar architecture has the ability to fetch, decode and execute multiple executions concurrently in the same clock cycle, but as issue rates increased, extracting enough ILP from them got harder.

Multi-Threading in Superscalar processors enabled them to issue instructions from different threads. Multi-Threaded processors have been able to hide long memory latencies but it had its limits. With increased issue rates, the concept of Fine grain Multithreading where an instruction from a different thread is issued at each clock cycle and Coarse grain Multithreading where instructions from a particular thread is allowed to issue until a major problem like cache miss or interrupts arise proved to be inefficient because of the increasing context switching overhead.

Simultaneous Multi-Threading (SMT) is a technique which permits multiple instructions from different threads to issue to a superscalar's multiple functional units at the same clock cycle [1]. Unlike other multi-threading techniques, SMT does not require to make use of a single thread to fill in the issue slots, it can issue from multiple threads, making use of Thread level parallelism together with Instruction level parallelism. As stated in [1], the increase in throughput comes with a cost of increased complexity of design and competitiveness between threads.

The basic pipeline stages in a 4-Threaded SMT system is given in Fig. 1. SMT systems are provided with separate program counters, register files, branch predictors and subroutine stacks for each thread. Instructions from multiple threads are fetched and forwarded to corresponding Instruction Fetch Queue (IFQ). These instructions are then decoded, register renamed and forwarded to their respective Re-Order Buffer (ROB), then into dispatch stage and queues up in the Issue Queue. When the issuing conditions are met, i.e., operands are ready and the requested functional unit becomes available, the instructions are issued to corresponding functional units in Out-of-order fashion. As and when the results are generated, it is written back or forwarded back to IQ. All instruction are committed from the ROB in order [2, 5].

This project makes use of a simulated environment to study the basic behavior of the shared hardware resources in SMT architecture namely the Issue Queue, Register File and Write Buffer. Variation in size of these parameters are studied, simulated and analyzed, arriving at inferences about each of the result, concluding at the overall understanding of SMT and the main shared resources.

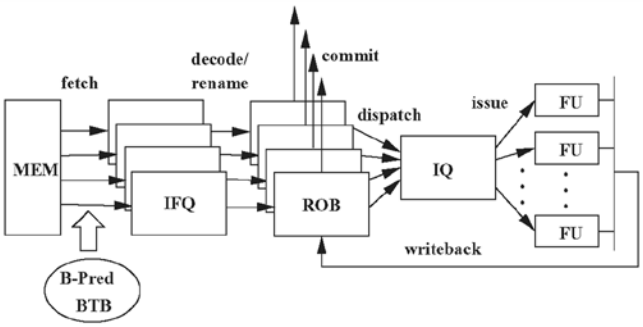


Fig. 1. Basic Pipeline Structure of a 4-Threaded SMT System [2]

## 2. Simulation Environment

This section provides information about the Simulator and the Workloads under consideration.

### 2.1 Simulator

Simulations are carried out in MSim v.3.0. MSim is an upgrade to Simplescalar 3.0d. It include accurate models of the pipeline structures, including explicit register renaming, and support for the concurrent execution of multiple threads to facilitate Simultaneous Multi-threading (SMT) [3]. It gives students and researchers a multi-threaded micro-architectural simulation environment where they can explore and exploit various performance scenarios in Simultaneous Multi-Threaded Microprocessors [11], [3]. In MSim internal architecture Register files, Issue queue and Write Buffer are shared by all the threads and MSim provide exclusive register file entries for each thread. Basic configuration of the SMT processor considered is shown in Table I.

TABLE I
BASIC CONFIGURATION OF THE SIMULATED PROCESSOR

| Parameter | Configuration |
|---|---|
| Machine Width | 8 wide fetch/dispatch/issue/commit |
| No. of Cores | 1, Single Core Processor |
| Load/Store Queue Size | 48 entry Load/Store Queue |
| IQ, RF and WB Size | As mentioned |
| L1 I-cache | 64KB, 2-way set-associative 128 byte line |
| L1 D-cache | 32 KB, 4-way set-associative 256 byte line |

TABLE II
DIFFERENT MODES OF OPERATION AND CORRESPONDING BENCHMARKS USED

| Mode of Operation | Benchmarks |
|---|---|
| Single Threaded | gcc |
| Two Threaded SMT | gcc, calculix |
| Four Threaded SMT | gcc, calculix, povray, HMMER |

TABLE III
DIFFERENT WORKLOADS AND CORRESPONDING BENCHMARKS USED

| Mix | Benchmarks |
|---|---|
| MIX 1 | gcc, calculix, povray, hmmer |
| MIX 2 | leslie3d, astar, bzip2, gobmk |
| MIX 3 | libquantum, sjeng, lbm, specrand |
| MIX 4 | perlbench, xalancbmk, gromacs, milc |

### 2.2 Workloads

Workload is gathered from SPEC2006 benchmark. Different mixes of benchmarks are created by choosing 2 integer and 2 floating point benchmarks [10]. Table II provides details about the workloads used for different modes of operation carried out in one of the simulations. Table III provides details about the workloads used for the main study. Four separate mixes are used for simulation, further details are available in Table III.

### 2.3 Tasks done in Simulation

Simulations are carried out, in M-Sim3.0 with Spec2006 benchmark, to understand change in throughput/IPC of a Single-Core Processor to variations in key data path components. Configuration of the processor under test is available in Table I. The common hardware resources that SMT and other Superscalar Processors share area Register Files, Issue Queue and Write Buffer.

#### 2.3.1 Number of threads

Typical Superscalar systems can issue instructions from a single thread at any given point in time. SMT extends this concept, giving access to issue instructions from multiple threads at any given time. Under default conditions of Simulator [see APPENDIX B] SMT operation under Single Issue, Dual Issue, Four Issue, Eight Issue and Sixteen Issue is carried out. The throughput/IPC of the system is logged from the simulation result. Fig. 3 shows the variation of IPC for changes in number of threads.

#### 2.3.2 Register File size

Register File (RF) size is an important parameter in SMT. It should always be greater than the total number of non-architectural registers required for all threads. The additional registers are required for Register Renaming. The in depth study carried out in [4], takes the number of register files required for Register renaming as 100; i.e., for single-threaded operation, total number of non-architectural registers required is 1*32 + 100 = 132 and for four-threaded operation is 4*32 + 100 = 228. In this project, the number of registers allocated for Register renaming is the same as the non-architectural registers allocated for all threads in total; i.e., for single-threaded operation, total number of non-architectural registers required is 1*32 + (1*32) = 64 and for four-threaded operation is 4*32 + (4*32) = 256. Any values lesser than specified will result in a CORE DUMP in the simulation; details are discussed in the following sections.

#### 2.3.3 Issue Queue size

Much like Register File size, Issue Queue (IQ) Size is also a crucial factor in determining the IPC of the single core processing system because it is shared by all the threads at real-time. Studies have found that thread behavior cannot be estimated precisely. A very active thread can suddenly become inactive and stay idle in pipeline for a longer time because of cache misses, branch miss-predictions, write port latencies etc [5]. Since Issue Queue acts like a basic waiting room for all instructions from various threads, increasing its size will give more room for each thread, ideally increasing the throughput/IPC of the system. In the study, Issue Queue value is changed from 8 entries to 512 entries to understand the nature of IPC and the bottleneck.

### 2.3.4 Write Buffer size

The write behavior of the system depends on the programs that are running and is highly unpredictable [7]. Study conducted in [6] show us the importance of write buffer in a memory sub-system. According to [6] adding a write buffer (WB) into the system will help to boost the IPC considerably. The major limitation in write buffer occurs when it encounters closely spaced writes. If, in this time, a read miss occurs in cache, the processor will not be able to access the memory since the write buffer will be writing to memory continuously. This creates a processor stall. Numerous methods are proposed [6], [7] to increase the write buffer performance and increase overall throughput.

This study concentrates on the write buffer size, its variation and its impact on IPC. The size of Write Buffer is increased from 4 entries to 16 in steps of 4. The results are logged and further analysis have been carried out.

## 3. Simulation Results

Results generated from the simulation is presented in this section. Respective analysis is done in the next section.

### 3.1 Number of threads

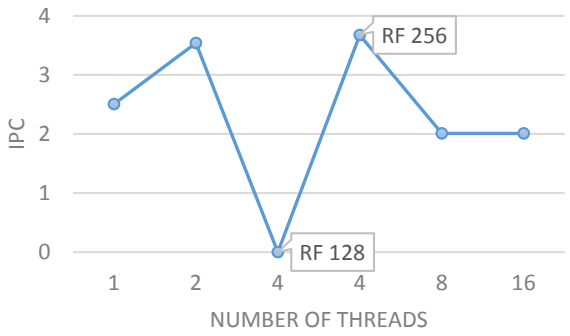


Fig. 2. Number of threads versus IPC

### 3.2 Register File size

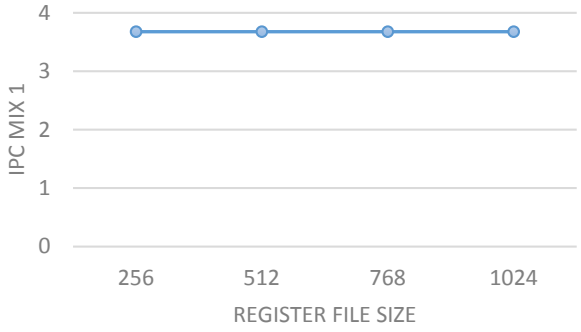


Fig. 3. Register File size vs IPC: MIX 1

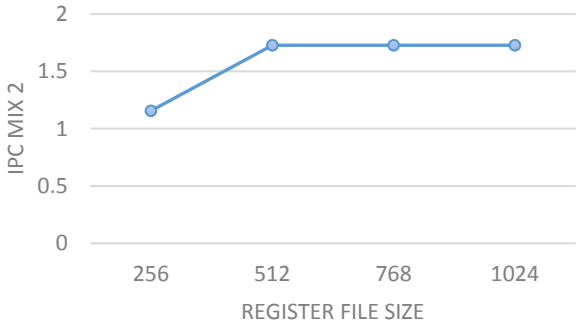


Fig. 4. Register File size vs IPC: MIX 2


Fig. 5. Register File size vs IPC: MIX 3

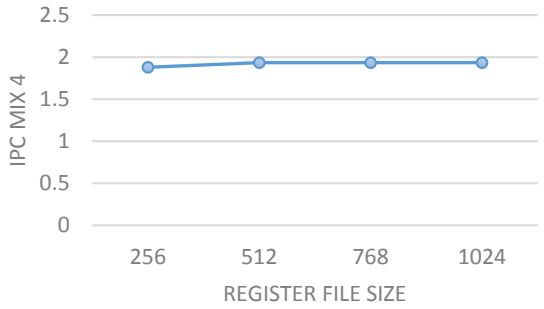


Fig. 6. Register File size vs IPC: MIX 4

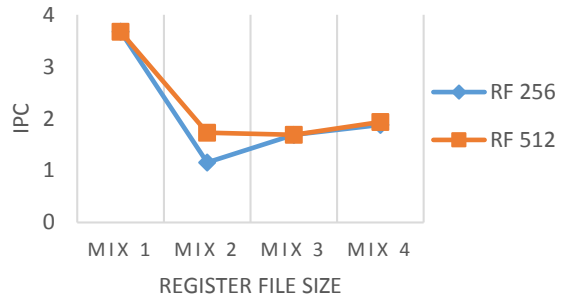


Fig. 7. Register File size vs IPC: Summary

### 3.3 Issue Queue size

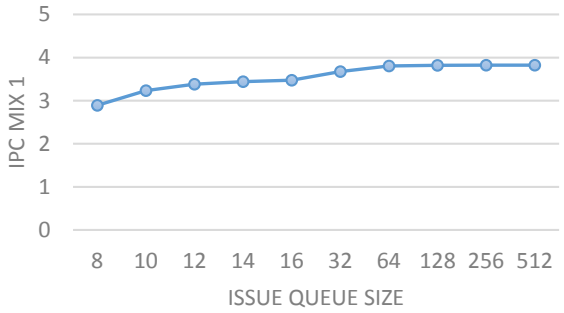


Fig. 8. Issue Queue Size vs IPC: MIX 1

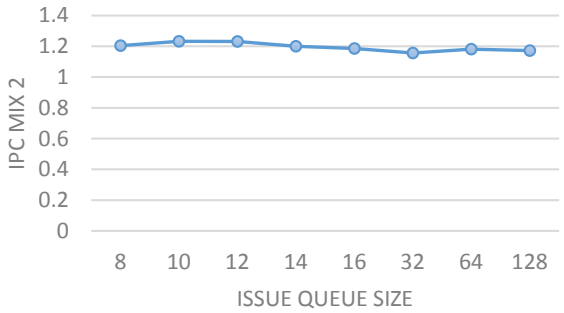


Fig. 9. Issue Queue Size vs IPC: MIX 2

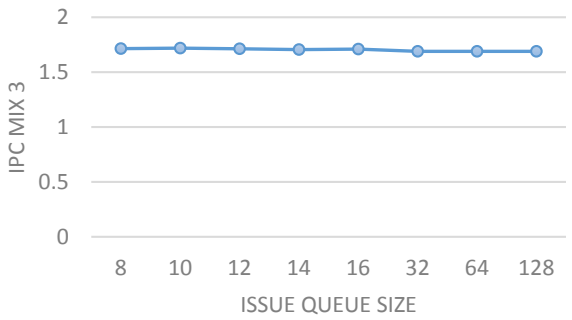


Fig. 10. Issue Queue Size vs IPC: MIX 3

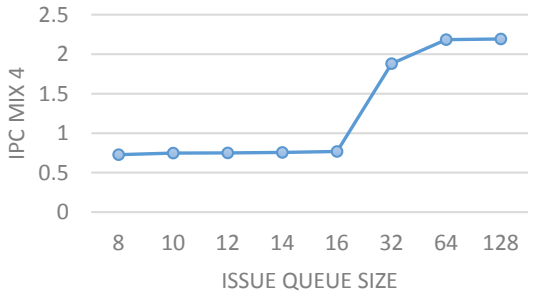


Fig. 11. Issue Queue Size vs IPC: MIX 4

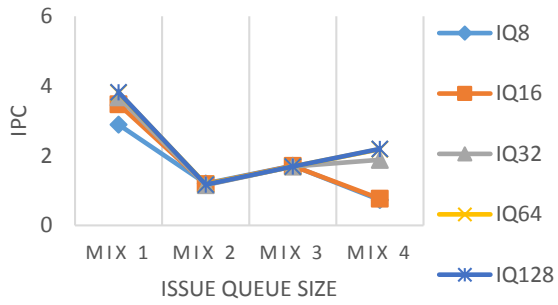


Fig. 12. Issue Queue Size vs IPC: Summary

### 3.4 Write Buffer size

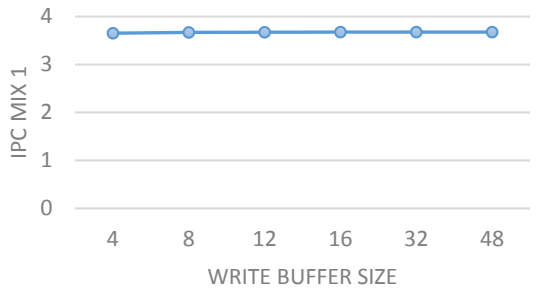


Fig. 13. Write Buffer Size vs IPC: MIX 1

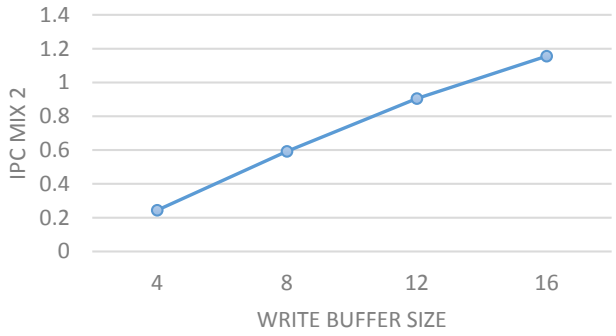


Fig. 14. Write Buffer Size vs IPC: MIX 2

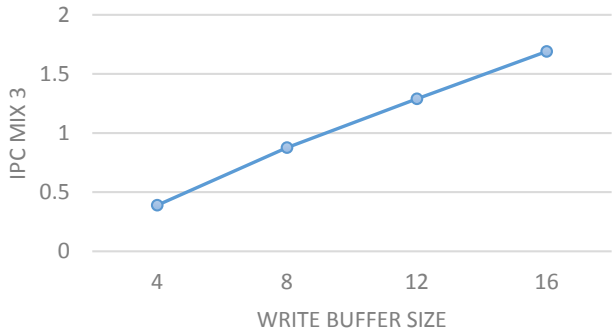


Fig. 15. Write Buffer Size vs IPC: MIX 3

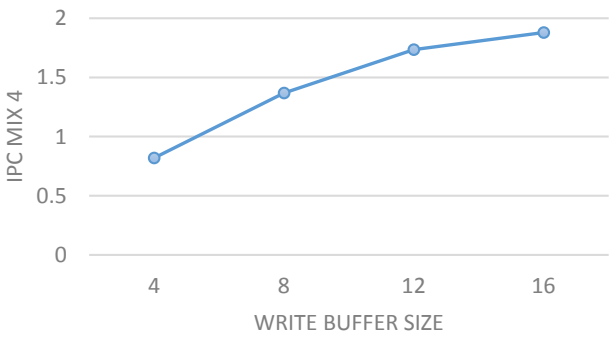


Fig. 16. Write Buffer Size vs IPC: MIX 4

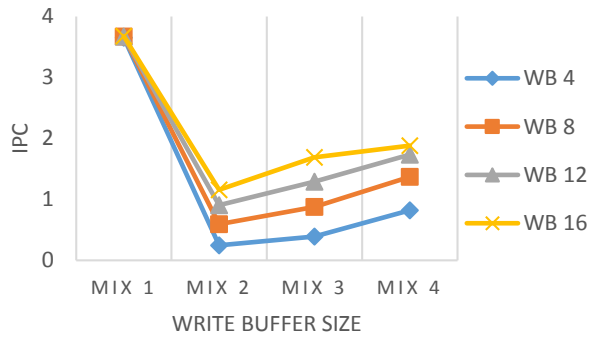


Fig. 17. Write Buffer Size vs IPC: MIX 5

## 4. Inference

The logged results provide insight into the role and working of each common resources. The inferences reached, through analysis of the data, is listed below.

### 4.1 Number of threads

Simulation begins with a single threaded operation, increasing it to sixteen threads with default values for all common hardware like Register File, Issue Queue and Write Buffer. From Fig. 2, the variation in IPC with changes in number of threads can be studied. Single threaded operation gave the lowest IPC because of the limited ILP and TLP extracted from it. Also, Single threaded operation is highly influenced by branch miss-predictions, cache misses and long memory stalls. It is seen that by shifting from single threaded to multi-threaded operation, the IPC increases considerably up to a four threaded operation. This increase can be linked to the increase of TLP that a SMT provides. With multiple threads issuing at the same time, resources like Issue Queue, Register File, Write Buffer and the functional units can be better utilized, thus increasing the efficiency and overall throughput of the system. Contrary to the performance gains in case of double and four threaded SMT, increasing the number of threads further to eight threads for default values of common hardware resources does not reflect on an improvement in IPC. Moreover, the IPC, in this study, decreased showing the limitation of other supporting hardware. In order to issue more instructions from multiple threads, like eight threads, proper increments in hardware resources should be provided. Register File size should be increased to account for the per-thread register requirement and registers for register renaming purpose. Similarly Issue Queue size should also be increased to make use of the eight threads available, so that each thread will get a considerable window to issue instructions. Also proper Write Buffer size should be allowed to give the retiring instructions a proper ‘waiting station’ in the write stage to lower level memory.

### 4.2 Register File size

Register File size (-rf:size) is varied from 256 entries to 1024 entries to find out whether there are any changes in IPC. . Default value of Register File size is 128 entries. According to MSim Architecture, each thread needs at least 32 non-architectural registers for operation. This is excluding the register requirement for renaming purpose. From the simulation result, it can be seen that for a Register File size of 128, a Four Threaded operation was aborted, resulting in a CORE DUMP error. This is due to the fact that a four threaded operation require 32*4 = 128 non-architectural registers and extra registers for register renaming. As discussed in the previous section, we assign the value of register file size as double than of actual requirement to account for the register renaming. It can be seen that for a Register File size of 256, i.e., 4*32 for each thread + (4*32) for register renaming, the Four Threaded simulation works flawlessly, generating desired result. It is found that further increase in Register File size will not change the IPC of any single thread or the overall Throughput/IPC as the required number of Non-Architectural Registers and entries required for Register Renaming are already available with 256 entries for a 4 threaded SMT operation. Register File size is fixed at 256 entries for all Mixes considered.

### 4.3 Issue Queue size

Fig. 1 shows the dependence of Issue Queue size on IPC for MIX 1. IPC shows considerable improvement up to an Issue Queue size (-iq:size) of 64 entries. Further increase in Issue Queue size does not result in an improvement of throughput/IPC. For MIX 2, however, IPC increases up to Issue Queue size of 12 and begins to drop. While MIX 3 does not show any considerable change in IPC, MIX 4 shows an abrupt change in IPC from Issue Queue size of 16 to 64 entries. From this, we can find that there is no general trend for the impact of Issue Queue on IPC for the Mixes considered. As discussed earlier, the studies conducted throw light into the issue by reminding about the unpredictable thread behavior. From the results, it can be concluded that increased size of Issue Queue not always help in improving IPC. Concepts like Capping, Instruction Recalling [5] must be applied to Issue Queue to increase it’s the effectiveness under varying conditions. IPC degradation in mixes considered may be because of high latency instructions, long processor stalls due to cache misses or increased dependence on a particular thread while issuing instructions from different threads. It can thus be inferred that the IPC of the system is deeply tied with the programs to which it is subjected to.

### 4.4 Write Buffer size

Write Buffer effectively reduces the waiting time between the CPU and lower-level memory in case of write-through policy [6]. With a decent size write buffer, even if there are many read follows a write operation, the entire write-through process can be done in the background without affecting the CPU performance since both cache and memory are updated together [6], [7]. Studies presented here shows that IPC of the system increase considerably with an increase in write buffer size. Similar performance gains have been found in all Workloads. From analysis of the data generated, it is concluded that the increase in number of slots in write buffer provides more waiting room for finished instructions, frees the processor to work on other instructions, thereby increasing the throughput of the system.

## 5. Conclusion

Simulation results and respective inferences provide insight into the impact of variation in shared parameters on overall system IPC. Variation of Write Buffer size showed a positive effect on IPC. When variation in Register File size alone showed very little or no change in IPC, variation in Issue Queue size broke general trends and showed the complexity of system under consideration. From the project, it can be concluded that performance changes in a SMT system not only depends upon the physical components, but also on the type of programs that it is being subjected into.

## 6. Acknowledgment

I gratefully acknowledge the support Dr. Wei-Min Lin extended towards me and this project. Also, I would like to thank my friend Avinash Anand for inspiring me to take up on this challenge which eventually led me to understand so many concepts which I would have easily neglected otherwise.

## 7. References

[1] D.M. Tullsen, S.Eggers, H. Levy, “Simultaneous Multithreading: Maximizing On-Chip Parallelism”, in the Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995.

[2] Yilin Zhang, “Management Of Shared Resources In Multi-Threading / Multi-Core Systems,” Ph.D. dissertation, Dept. ECE, UTSA.,SanAntonio, TX, 2014.

[3] Joseph J. Sharkey, Dmitry Ponomarev and Kanad Ghose, " M-SIM: A Flexible, Multithreaded Architectural Simulation Environment," [Online]. Available: http://www.cs.binghamton.edu/~jsharke/m-sim/documentation/msim_tr.pdf

[4] D.M. Tullsen, S.Eggers, J.S.Emer, H. Levy, J.L. Lo, R.L. Stamm, “Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor”, in the Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996.

[5] Y. Zhang, C. Douglas, Wei-Ming Lin - "On Maximizing Resource Utilization for Simultaneous Multi-Threading (SMT) Processors by Instruction Recalling".

[6] P.P.Chu, R.Gottipati, “Write Buffer Design for On-Chip Cache” in the Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors, October 1994.

[7] F.M.-Toussi, D.J.Lilja, “Write Buffer Design for Cache-Coherent Shared-Memory Multiprocessors” in the Proceedings of International Conference on Computer Design: VLSI in Computers and Processors, October 1995.

[8] D. Wall, Limits of instruction-level parallelism, Technical Report WRLTR-93.6, Western Research Laboratory, Digital, 1993.

[9] K.S.Loh, W.F.Wong, “Multiple context multithreaded superscalar processor architecture” in the Journal of Systems Architecture, vol. 46, pp243-258, 2000.

[10] J.Henning, “SPEC CPU2006 Benchmark Descriptions”, [Online] Available: http://spec.org/cpu2006/publications/CPU2006benchmarks.pdf

[11] ECE.UMASS.EDU, "Introduction to M-Sim," [Online]. Available: http://www.ecs.umass.edu/ece/koren/architecture/MSim/IntroductionToMSim.htm

# 8. Appendix

## 8.1 Nomenclature

| | |
|---|---|
| *SMT* | Simultaneous Multi-Threading |
| *RF* | Register File |
| *IQ* | Issue Queue |
| *ILP* | Instruction Level Parallelism |
| *IPC* | Instructions per Cycle |
| *TLP* | Thread Level Parallelism |
| *WB* | Write Buffer |

## 8.2 Installation Instructions for MSim3.0 and Spec 2006 Benchmarks

1. Prepare files, do not copy libraries or files in spec2006 folder to MSim folder yet.
2. Open Terminal.
3. Update GCC. (Instructions available ONLINE: http://charette.no-ip.com:81/programming/2011-12-24_GCCv47/ )
4. In the MSim folder, open memory.c and #include unistd.h into the headers.
5. Open Terminal, go to MSim folder, type and execute the following :

make clean
make all
make

6. Close terminal, Copy the contents in benchmark Spec2006 folder into MSim folder.
7. Test the simulator by executing ./sim-outorder in Terminal.

## 8.3 Basic Commands and Default Values

**# -config** # load configuration from a file
**# -dumpconfig** # dump configuration to a file
**# -h** false # print help message
**# -v** false # verbose operation
**# -d** false # enable debug message
**-seed** 1 # random number generator seed (0 for timer seed)
**# -q** false # initialize and terminate immediately
**# -redir:sim** <null> # redirect simulator output to file (non-interactive only)
**# -redir:prog** <null> # redirect simulated program output to file (all benchmarks)
**# -redir:err** <null> # redirect simulated program output (stderr) to file (all benchmarks)
**-nice** 0 # simulator scheduling priority
**-max:inst** 1000000 # maximum number of inst's to execute
**-max:cycles** -1 # maximum number of cycles to execute
**-fastfwd** 1000000 # number of insts skipped before timing starts (1 to use value in .arg file)
**# -ptrace** <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
**# -pcstat** <null> # profile stat(s) against text addr's (mult uses ok)
**-power:print_stats** false # print power statistics collected by wattch?
**-num_cores** 1 # Number of processor cores
**-max_contexts_per_core** 1 # Number of contexts allowed per core (-1 == limit is number of contexts, 0 is invalid)
**-fetch:speed** 1 # speed of front-end of machine relative to execution core
**-decode:width** 8 # instruction decode B/W (insts/cycle)
**-issue:width** 8 # instruction issue B/W (insts/cycle)
**-issue:inorder** false # run pipeline with in-order issue
**-issue:wrongpath** true # issue instructions down wrong execution paths
**-commit:width** 8 # instruction commit B/W (insts/cycle)
**-iq:issue_exec_delay** 1 # minimum cycles between issue and execution
**-fetch_rename_delay** 4 # number of cycles between fetch and rename stages
**-rename_dispatch_delay** 1 # number cycles between rename and dispatch stages
**-lsq:size** 48 # load/store queue (LSQ) size
**-rob:size** 128 # reorder buffer (ROB) size
**-fetch:policy** icount # fetch policy, icount, round_robin, dcra
**-recovery:model** squash # Alpha squash recovery or perfect predition: |squash|perfect|
**-iq:size** 32 # issue queue (IQ) size
**-rf:size** 128 # register file (RF) size for each the INT and FP physical register file)
**-res:ialu** 4 # total number of integer ALU's available
**-res:imult** 1 # total number of integer multiplier/dividers available
**-res:memport** 2 # total number of memory system ports available (to CPU)
**-res:fpalu** 4 # total number of floating point ALU's available
**-res:fpmult** 1 # total number of floating point multiplier/dividers available
**-write_buf:size** 16 # write buffer size (for stores to L1, not for writeback)
**-cache:dl1** dl1:256:64:4:l # l1 data cache config, i.e., {<config>|none}
**-cache:dl1lat** 1 # l1 data cache hit latency (in cycles) -cache:dl2 ul2:512:64:16:l # l2 data cache config, i.e., {<config>|none}
**-cache:dl2lat** 10 # l2 data cache hit latency (in cycles)
**-cache:il1** il1:512:64:2:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
**-cache:il1lat** 1 # l1 instruction cache hit latency (in cycles)
**-cache:il2** dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
**-cache:il2lat** 10 # l2 instruction cache hit latency (in cycles)
**-tlb:itlb** itlb:16:4096:4:l # instruction TLB config, i.e.,

{<config>|none}
**-tlb:dtlb** dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
**-tlb:lat** 30 # inst/data TLB miss latency (in cycles)
**-bpred** bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
**-bpred:ras** 16 # return address stack size (0 for no return stack)
**# -bpred:spec_update** <null> # speculative predictors update in {ID|WB} (default non-spec)
**-bpred:bimod** 2048 # bimodal predictor config (<table size>)
**-bpred:2lev** [ 1 1024 12 0 ]# 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
**-bpred:comb** 1024 # combining predictor config (<meta_table_size>)
**-bpred:btb** [ 512 4 ]# BTB config (<num_sets> <associativity>)
**-bpred:penalty** 6 # penalty (in cycles) for a branch misprediction
**-cpred** bimod # cache load-latency predictor type {nottaken|taken|perfect|bimod|2lev|comb}
**-cpred:ras** 0 # return address stack size (0 for no return stack)
**-cpred:bimod** 2048 # cache load-latency bimodal predictor config (<table size>)
**-cpred:2lev** [ 1 1024 12 0 ]# cache load-latency 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
**-cpred:comb** 1024 # cache load-latency combining predictor config (<meta_table_size>)
**-cpred:btb** [ 512 4 ]# cache load-latency BTB config (<num_sets> <associativity>)
**-mem:config** chunk:4:300:2 # Main memory configuration
**-cache:dl3** none # l3 data cache config, i.e., {<config>|none}
**-cache:dl3lat** 30 # l3 data cache hit latency (in cycles)
**-cache:il3** none # l3 instruction cache config, i.e., {<config>|dl3|none}
**-cache:il3lat** 30 # l3 instruction cache hit latency (in cycles)

### 8.4 Method of Simulation

1. In Linux OS, open Terminal and direct it to MSim install folder.
2. Simulation is executed by calling sim-outorder, specifying configuration of the processor and the workload.
   E.g.: For a simulation of MIX 1, with Register File size of 256 entries and Write Buffer size of 16 entries, use the following code: *./sim-outorder –rf:size 256 –write_buf:size 16 gccNS.1.arg calculixNS.1.arg povrayNS.1.arg hmmerNS.1.arg*
3. The simulation results will be displayed in Terminal or can be directed to external files according to users wish.