Single Core Design Space Exploration

ECE 568: Advanced Microprocessor Architecture, Spring 2016

Anthony Gallotta
agallo4@uic.edu

ABSTRACT

This paper seeks to find the best single core superscalar architecture for a given benchmark application by experimenting with different architectures for branch prediction, memory systems, functional units, data path, and other areas. The metrics used to define the "best" architecture are *performance* as instructions per cycle (IPC) and *performance-energy* as the energy delay product (EDP).

1. INTRODUCTION

This study evaluates a single core superscalar architecture using the SimpleScalar simulator suite, seeking the best configuration for the `eeg` benchmark application. A base architecture was given, and design modifications were then made in four groups, branch prediction, memory system, functional units, and data path. In the first phase of design space exploration, tuning one parameter at a time. In the second stage, closely related parameters within the same groups are tuned together. Finally, the best configurations across groups are combined to formulate the best found architecture. For each configuration we observe performance (IPC) and performance-energy (EDP), calculated as $(CPI * energy/cycle) * CPI$. The "best" configuration seeks to maximize performance, and minimize performance-energy.

2. SIMULATION

Simulations of the `eeg` application benchmark were performed using `sim-outorder`. The base configuration had performance of 1.5226 IPC, and a performance-energy product of 219.68. While many parameter changes were tested, unless otherwise noted, all graphs that follow show only configurations that outperformed the base configuration in at least one of these metrics[1]. The most desirable configurations will lie in the lower right side of the graphs that follow.

[1]Results from all configurations, along with the configuration files themselves and scripts used in this simulation are available at https://github.com/tonygallotta/ece568-single-core-dse.git

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Table 1: Instruction Profile for `eeg`

Type	Count	Percent
load	171,182,879	20.30
store	84,069,724	9.97
uncond branch	60,740,822	7.20
cond branch	59,922,458	7.11
int computation	335,861,146	39.84
fp computation	131,306,086	15.57
trap	1,035	0.00

To gain some direction in performance tuning, the instruction profile for the benchmark was first evaluated using `sim-profile`. As shown in table 1, the benchmark is primarily an integer program. Given this instruction breakdown, most of the performance tuning was focused on branch prediction and the memory system. For each group, the initial focus was on one attribute at a time. Once good parameters to tune were found, some combinations of parameters within each group were attempted as well to further improve performance.

Simulations were executed by providing configuration files to `sim-outorder`. A sample configuration file and output are provided at the end of this report. Simulation results were then processed by a series of `bash` scripts to generate a CSV file of the results, which was processed using Python and the `matplotlib` library to generate the figures in this report[2].

2.1 Branch Prediction

The first branch prediction change attempted was to use static prediction, to see if the decreased power consumption would be significant enough to improve EDP. This strategy resulted in much worse performance by both metrics though, and was quickly abandoned. Next, some tuning was made with the default bimodal branch predictor, which uses a simple strategy of picking the most common direction. Sticking with this scheme, the branch target buffer (BTB) size and associativity was adjusted. In figure 1, the points labeled as `bp-btb-<sets>-<associativity>` indicate the results of these adjustments. The BTB configuration of 128 sets with 2 way associativity is able to achieve the same performance as the base configuration, with a much lower EDP.

Two level branch predictors use a combination of the last k historical branch outcomes, as well as the behavior for a specific pattern of previous branches [1]. While many of

[2]See footnote 1

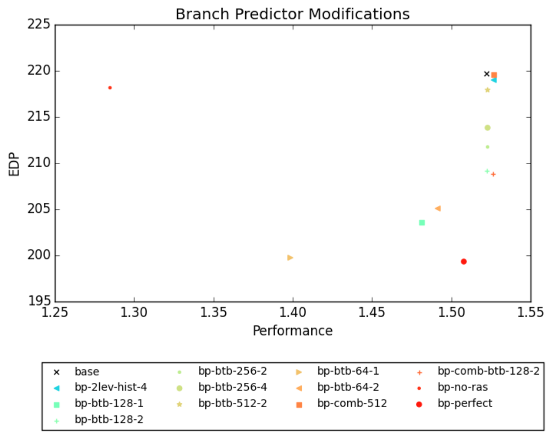

Figure 1: Branch Predictor Variation results

the 2 level configurations did not perform the base configuration, the 2 level Global Address (GA) branch predictor with a 4 wide history (bp-2level-hist-4) did achieve a better IPC. Combining these configuration changes (bp-comb-128-2) by keeping the best found BTB size and using a combined bimodal and 2-level branch predictor gives about a 5% performance-energy improvement over the base configuration.

A perfect branch prediction scheme is plotted for comparison. Suprisingly, the perfect branch predictor achieves a lower IPC than many of the experimental configurations. The simulator documentation does not describe in detail how the perfect branch predictor operates, so it may be that they are modeling a higher latency for the perfect predictor.

2.2 Memory System

The simulator has a 2 level cache hierarchy, and different sizes, associativity, block sizes, and replacement policies were experimented with at each level, and for both instruction and data caches. Cache parameter modifications were not chosen to align with the program in any way since it is difficult to predict the memory access patterns of a program without careful inspection of the source code. To start with, parameters were simply chosen to cast a wide net over the design space and see which modifications may show promise. For example, most parameters were increased and decreased by a power of 2 for an initial experiment. If that modification resulted in a performance improvement, another power of 2 increase or decrease was attempted, until performance started to degrade.

2.2.1 L1 Cache

As figure 2 shows, increasing the size of the L1 data cache (sets) did not result in significant IPC improvements relative to their increased power consumption. Increasing the block size to 64 while keeping the total L1 size the same (bsize-64) yields the best result in terms of both of our architecture goals.

Figure 3 shows similar results for the instruction cache - a block size of 64 performs best. These results indicate that there is some spatial locality in the instructions and data.

2.2.2 L2 Cache

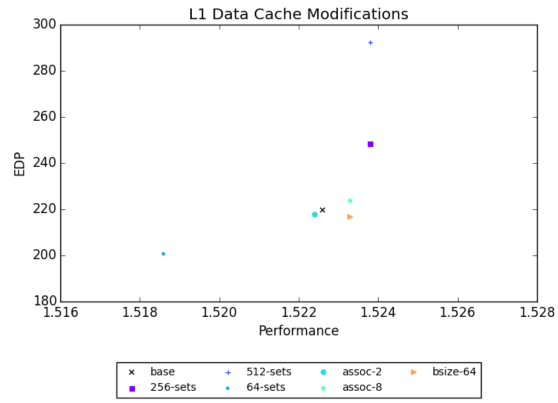

Figure 2: Modifications for L1 data cache

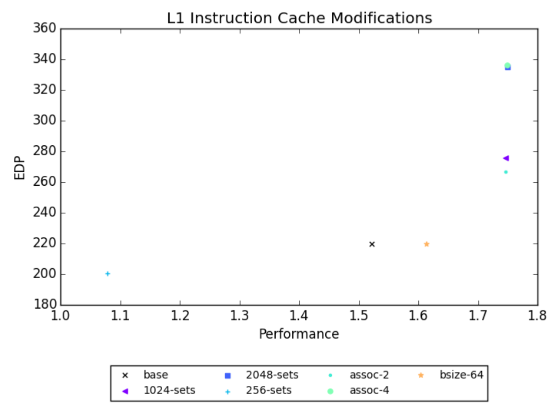

Figure 3: Modifications for L1 instruction cache

At the L2 cache, we see once again from figures 4, 5, and 8 that a unified cache with a decreased block size performs best. This is an interesting result since the L1 cache benefited from an increased block size - the performance improvement here seems to be primarily in aligning the L1 and L2 cache sizes, rather than one particular value. We also see a performance gain in *decreasing* the number of sets in the L1 cache, indicating that this program doesn't have working sets large enough to reap the benefits of a larger cache.

2.3 Functional Units

Since this program is integer focused, one of the first modifications made to increase performance-energy was to decrease the number of floating point units. As figure 7 shows (fpalu-2), the number of floating point ALUs can be decreased from 4 to 2 with no change in IPC. Interestingly, increasing the number of FP multipliers (from 1 to 2) does increase IPC, at a slight cost to EDP.

The integer functional units seem to already be at a sweet spot. Increasing the number of integer ALUs or multipliers does not result in significant IPC gains. The base configuration only has 1 integer multiplier, so this cannot be decreased, but decreasing the number of ALUs results in

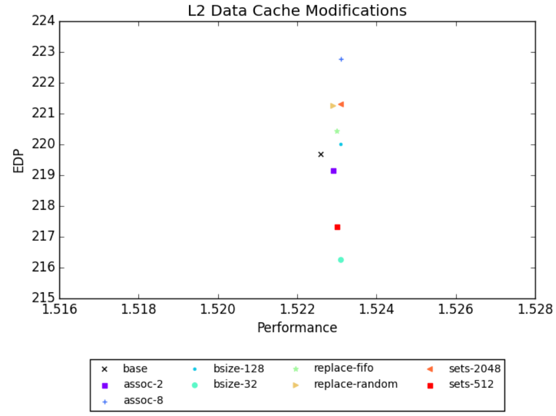

Figure 4: Modifications for L2 data cache

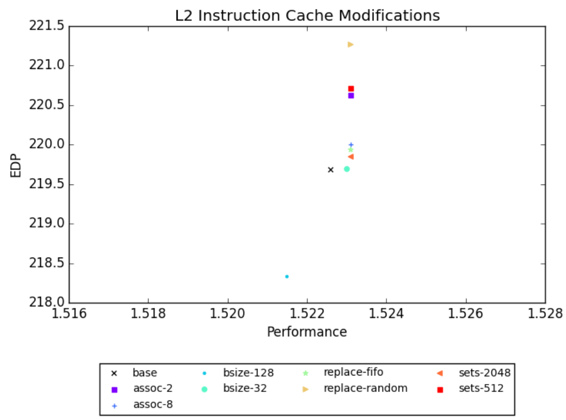

Figure 5: Modifications for L2 instruction cache

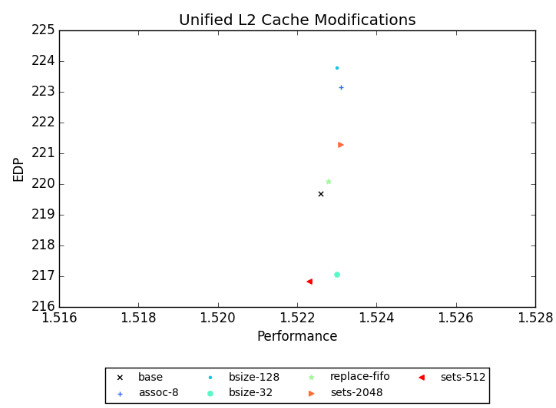

Figure 6: Unified L2 cache

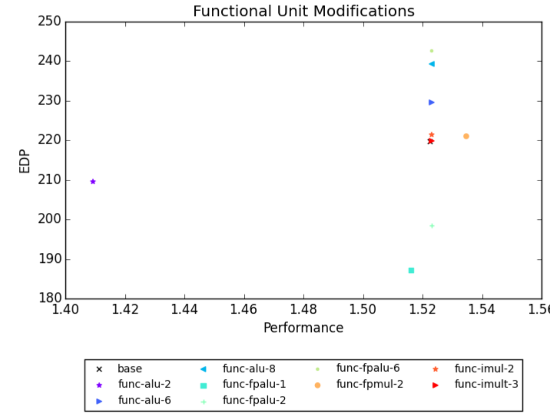

Figure 7: Functional unit modifications

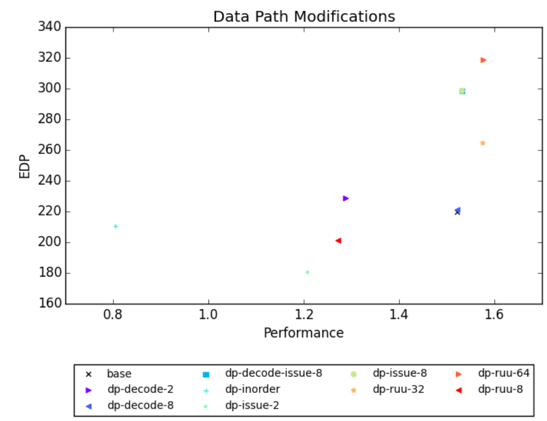

Figure 8: Data path modifications

significant IPC losses.

2.4 Data Path

The data path attributes modified were in-order vs. out of order (base) issue, instruction decode and issue width, and size of the reorder. On their own, none of these modifications result in a good trade off of performance vs. EDP. Even when tuning closely related parameters together, such as instruction decode and issue width, there are no clear performance improvements. These attributes will be revisited when tuning parameters are combined since other modifications may make a wider instruction issue or reorder buffer more useful.

3. CROSS GROUP SIMULATIONS

For the final group of simulations, the best parameter modifications from each group were selected, and combined. The only exception was between the L1 and L2 caches. While a block size of 64 performed best for the L1 cache, the best performance for the L2 cache was observed under a block size of 32. In practice, this doesn't really make sense since the L1 cache is typically a subest of the L2 cache, and the L1 cache would have to load 2 blocks to fill 1 of its

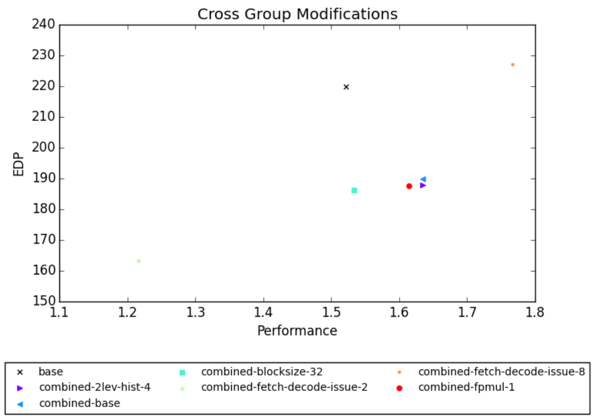

Figure 9: Cross group modifications

own. sim-outorder did not even allow this configuration, and crashed when attempting to use a smaller block size in the L2 cache than L1. Therefore, the block size was kept consistent between the 2 levels, and both 32 (combined-blocksize-32) and 64 (combined-base) were tried for the combined configuration. As seen in figure 9, the block size of 64 achieved a pretty large increase in IPC without a large EDP sacrifice, so this architecture was chosen to proceed with. Details for this configuration are listed at the end of this paper.

In this step, modifications to the instruction issue width were revisited, tuning all related parameters together (fetch, decode, issue, and commit width), however the results were fairly extreme. An issue width of 8 resulted in a 16% IPC increase, but also increased the EDP by 3%. Dropping the width down to 2 resulted in a significant EDP decrease (25%), but also dropped IPC by 20%. The architecture that achieved the best performance overall was comb-2lev-hist-4, as seen in figure 9, achieving roughly a 15% decrease in performance-energy, and 7% improvement in performance.

4. CONCLUSIONS

Simulation tools can be useful in tuning architectures for good performance for a specific application. Observing the instruction profile for the benchmark application can be helpful in making educated guesses when tuning architecture details, but some parameters like memory configurations can be difficult to estimate, and can be easier to tune by testing a wide range of parameters and observing their effects.

The design space, even for a single core processor, can be extremely large. Tuning parameters one at a time and combining the best results in each class can be an effective strategy for finding a good design with so many choices available. The experiments described in this paper seek to achieve a balance between performance and power consumption, without defining any quantitative targets for either. Given constraints on either aspect, other configurations found in this experiment could easily have been chosen, for example to optimize for performance or performance-energy alone.

APPENDIX

A. RESULTS OF ALL SIMULATIONS

Category	Config	IPC	EDP
base	base	1.5226	219.68
bp	bp-2lev-GAg	1.4142	223.2
bp	bp-2lev-GAp	1.4288	222.5
bp	bp-2lev-PAg-l1-16	1.5143	220.96
bp	bp-2lev-PAg-l1-2	1.4255	221.83
bp	bp-2lev-PAp-l1-2	1.5184	220.17
bp	bp-2lev-PAp-l1-4	1.5320	221.83
bp	bp-2lev-gshare	1.5259	220.15
bp	bp-2lev-hist-16	1.4142	223.8
bp	bp-2lev-hist-4	1.5266	218.98
bp	bp-2lev-l1-128	1.5378	218.41
bp	bp-2lev-l1-64	1.5380	219.7
bp	bp-2lev-l1-8	1.4287	220.19
bp	bp-2lev-l2-512	1.4108	223.55
bp	bp-2lev-xor	1.5235	220.29
bp	bp-2lev	1.4288	221.57
bp	bp-btb-1024-2	1.5230	219.7
bp	bp-btb-1024-4	1.5230	225.08
bp	bp-btb-128-1	1.4811	203.62
bp	bp-btb-128-2	1.5224	209.2
bp	bp-btb-256-2	1.5228	211.77
bp	bp-btb-256-4	1.5230	213.86
bp	bp-btb-512-2	1.5229	217.93
bp	bp-btb-64-1	1.3987	199.74
bp	bp-btb-64-2	1.4914	205.07
bp	bp-comb-2048	1.5270	220.73
bp	bp-comb-512	1.5269	219.54
bp	bp-comb-btb-128-2	1.5263	208.81
bp	bp-comb	1.5270	219.93
bp	bp-no-ras	1.2846	218.17
bp	bp-nottaken	0.7513	272.76
bp	bp-perfect	1.5078	199.34
bp	bp-ras-16	1.5231	220.8
bp	bp-taken	0.7578	281.58
combined	2lev-hist-4	1.6352	187.72
combined	base	1.6352	189.72
combined	blocksize-32	1.5339	186.33
combined	fetch-decode-issue-2	1.2162	163.13
combined	fetch-deodec-issue-8	1.7670	226.91
combined	fpmul-1	1.6152	187.55

Category	Configuration	IPC	EDP
dp	dp-decode-2	1.2894	228.32
dp	dp-decode-8	1.5230	221.16
dp	dp-decode-issue-8	1.5323	298.64
dp	dp-inorder	0.8050	210.34
dp	dp-issue-2	1.2087	180.59
dp	dp-issue-8	1.5323	298.45
dp	dp-ruu-8	1.2721	201.05
func	func-alu-2	1.4094	209.63
func	func-alu-6	1.5230	229.5
func	func-alu-8	1.5230	239.39
func	func-fpalu-1	1.5160	187.15
func	func-fpalu-2	1.5230	198.54
func	func-fpmul-2	1.5347	221.12
func	func-imul-2	1.5230	221.43
func	func-imul-3	1.5230	219.94
mem	mem-dl1-128-64-2	1.5227	215.25
mem	mem-dl1-256-sets	1.5238	248.48
mem	mem-dl1-64-sets	1.5186	200.63
mem	mem-dl1-assoc-2	1.5224	217.52
mem	mem-dl1-assoc-8	1.5233	223.67
mem	mem-dl1-bsize-128	0.1143	529.49
mem	mem-dl1-bsize-16	1.5223	223.46
mem	mem-dl1-bsize-64	1.5233	216.74
mem	mem-dl1-replace-fifo	1.5225	221.42
mem	mem-dl1-replace-random	1.5217	219.78
mem	mem-dl2-assoc-2	1.5218	220.7
mem	mem-dl2-assoc-8	1.5231	223.16
mem	mem-dl2-bsize-128	1.5230	223.76
mem	mem-dl2-bsize-32	1.5230	217.05
mem	mem-dl2-replace-fifo	1.5228	220.07
mem	mem-dl2-replace-random	1.5226	220.31
mem	mem-dl2-separated-assoc-2	1.5229	219.15
mem	mem-dl2-separated-assoc-8	1.5231	222.78
mem	mem-dl2-separated-bsize-128	1.5231	219.99
mem	mem-dl2-separated-bsize-32	1.5231	216.26
mem	mem-dl2-separated-replace-fifo	1.5230	220.42
mem	mem-dl2-separated-replace-random	1.5229	221.26
mem	mem-dl2-separated-sets-2048	1.5231	221.29
mem	mem-dl2-separated-sets-512	1.5230	217.32
mem	mem-dl2-sets-2048	1.5231	221.27
mem	mem-dl2-sets-512	1.5223	216.82
mem	mem-il1-1024-sets	1.7466	275.48
mem	mem-il1-256-sets	1.0784	200.54
mem	mem-il1-assoc-2	1.7473	266.75
mem	mem-il1-assoc-4	1.7495	336.25
mem	mem-il1-bsize-16	1.4704	239.86
mem	mem-il1-bsize-64	1.6141	219.35
mem	mem-il1-fifo	1.5217	220.14
mem	mem-il1-random	1.5217	219.74
mem	mem-il2-separated-assoc-2	1.5231	220.62
mem	mem-il2-separated-assoc-8	1.5231	220.0
mem	mem-il2-separated-bsize-128	1.5215	218.33
mem	mem-il2-separated-bsize-32	1.5230	219.69
mem	mem-il2-separated-replace-fifo	1.5231	219.93
mem	mem-il2-separated-replace-random	1.5231	221.27
mem	mem-il2-separated-sets-2048	1.5231	219.85
mem	mem-il2-separated-sets-512	1.5231	220.71

B. CONFIGURATION FOR BEST FOUND ARCHITECTURE (COMB-2LEV-HIST-4)

```
# load configuration from a file
# -config

# dump configuration to a file
# -dumpconfig

# print help message
# -h                              false

# verbose operation
# -v                              false

# enable debug message
# -d                              false

# start in Dlite debugger
# -i                              false

# random number generator seed (0 for timer seed)
-seed                                 1

# initialize and terminate immediately
# -q                              false

# restore EIO trace execution from <fname>
# -chkpt                        <null>

# redirect simulator output to file (non-interactive only)
# -redir:sim                    <null>

# redirect simulated program output to file
# -redir:prog                   <null>

# simulator scheduling priority
-nice                                 0

# maximum number of inst's to execute
-max:inst                             0

# number of insts skipped before timing starts
-fastfwd                              0

# generate pipetrace, i.e., <fname|stdout|stderr> <range>
# -ptrace                       <null>

# instruction fetch queue size (in insts)
-fetch:ifqsize                        4

# extra branch mis-prediction latency
-fetch:mplat                          3

# speed of front-end of machine relative to execution core
-fetch:speed                          1

# branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred                            bimod

# bimodal predictor config (<table size>)
-bpred:bimod                2048

# 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
```

```
-bpred:2lev             1 1024 4 0

# combining predictor config (<meta_table_size>)
-bpred:comb             1024

# return address stack size (0 for no return stack)
-bpred:ras                          8

# BTB config (<num_sets> <associativity>)
-bpred:btb              128 2

# speculative predictors update in {ID|WB} (default non-spec)
# -bpred:spec_update          <null>

# instruction decode B/W (insts/cycle)
-decode:width                       4

# instruction issue B/W (insts/cycle)
-issue:width                        4

# run pipeline with in-order issue
-issue:inorder                  false

# issue instructions down wrong execution paths
-issue:wrongpath                 true

# instruction commit B/W (insts/cycle)
-commit:width                       4

# register update unit (RUU) size
-ruu:size                          16

# load/store queue (LSQ) size
-lsq:size                           8

# l1 data cache config, i.e., {<config>|none}
-cache:dl1              dl1:64:64:4:l

# l1 data cache hit latency (in cycles)
-cache:dl1lat                       1

# l2 data cache config, i.e., {<config>|none}
-cache:dl2              ul2:1024:64:4:l

# l2 data cache hit latency (in cycles)
-cache:dl2lat                       6

# l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1              il1:256:64:1:l

# l1 instruction cache hit latency (in cycles)
-cache:il1lat                       1

# l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2                        dl2

# l2 instruction cache hit latency (in cycles)
-cache:il2lat                       6

# flush caches on system calls
-cache:flush                    false

# convert 64-bit inst addresses to 32-bit inst equivalents
```

```
-cache:icompress                false

# memory access latency (<first_chunk> <inter_chunk>)
-mem:lat                18 2

# memory access bus width (in bytes)
-mem:width                              8

# instruction TLB config, i.e., {<config>|none}
-tlb:itlb               itlb:16:4096:4:l

# data TLB config, i.e., {<config>|none}
-tlb:dtlb               dtlb:32:4096:4:l

# inst/data TLB miss latency (in cycles)
-tlb:lat                                30

# total number of integer ALU's available
-res:ialu                               4

# total number of integer multiplier/dividers available
-res:imult                              1

# total number of memory system ports available (to CPU)
-res:memport                            2

# total number of floating point ALU's available
-res:fpalu                              2

# total number of floating point multiplier/dividers available
-res:fpmult                             2

# profile stat(s) against text addr's (mult uses ok)
# -pcstat                       <null>

# operate in backward-compatible bugs mode (for testing only)
-bugcompat                      false
```

C. SIMULATION OUTPUT FOR BEST FOUND ARCHITECTURE (COMB-2LEV-HIST-4)

```
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).


Processor Parameters:
Issue Width: 4
Window Size: 16
Number of Virtual Registers: 32
Number of Physical Registers: 16
Datapath Width: 64
Total Power Consumption: 58.5672
Branch Predictor Power Consumption: 1.25629  (2.22%)
 branch target buffer power (W): 0.902794
 local predict power (W): 0.0867068
 global predict power (W): 0.0996078
 chooser power (W): 0.0702439
 RAS power (W): 0.0969383
Rename Logic Power Consumption: 0.417965  (0.739%)
 Instruction Decode Power (W): 0.0159915
```

```
 RAT decode_power (W): 0.113514
 RAT wordline_power (W): 0.0307815
 RAT bitline_power (W): 0.246771
 DCL Comparators (W): 0.0109075
Instruction Window Power Consumption: 2.20432  (3.9%)
 tagdrive (W): 0.0943488
 tagmatch (W): 0.033695
 Selection Logic (W): 0.0134217
 decode_power (W): 0.0532503
 wordline_power (W): 0.0844643
 bitline_power (W): 1.92514
Load/Store Queue Power Consumption: 0.96318  (1.7%)
 tagdrive (W): 0.457232
 tagmatch (W): 0.100774
 decode_power (W): 0.00799577
 wordline_power (W): 0.0150119
 bitline_power (W): 0.382166
Arch. Register File Power Consumption: 3.57247  (6.32%)
 decode_power (W): 0.113514
 wordline_power (W): 0.0844643
 bitline_power (W): 3.37449
Result Bus Power Consumption: 2.29754  (4.06%)
Total Clock Power: 21.9259  (38.8%)
Int ALU Power: 4.66013  (8.24%)
FP ALU Power: 7.14052  (12.6%)
Instruction Cache Power Consumption: 1.99903  (3.53%)
 decode_power (W): 0.362444
 wordline_power (W): 0.0491838
 bitline_power (W): 1.14691
 senseamp_power (W): 0.192
 tagarray_power (W): 0.248491
Itlb_power (W): 0.263317 (0.465%)
Data Cache Power Consumption: 4.75559  (8.41%)
 decode_power (W): 0.651055
 wordline_power (W): 0.196735
 bitline_power (W): 2.71235
 senseamp_power (W): 0.768
 tagarray_power (W): 0.427443
Dtlb_power (W): 0.901877 (1.59%)
Level 2 Cache Power Consumption: 4.2091 (7.44%)
 decode_power (W): 0.41817
 wordline_power (W): 0.0430878
 bitline_power (W): 3.0244
 senseamp_power (W): 0.192
 tagarray_power (W): 0.531433
sim: command line: sim-outorder -redir:sim /home/ubuntu/ece568-single-core-dse/results/combined-2lev-

sim: simulation started @ Mon Mar  7 14:13:39 2016, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

# -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
# -i                    false # start in Dlite debugger
-seed                       1 # random number generator seed (0 for timer seed)
# -q                    false # initialize and terminate immediately
# -chkpt               <null> # restore EIO trace execution from <fname>
```

```
# -redir:sim      /home/ubuntu/ece568-single-core-dse/results/combined-2lev-hist-4.out # redirect simu
# -redir:prog           <null> # redirect simulated program output to file
-nice                        0 # simulator scheduling priority
-max:inst                    0 # maximum number of inst's to execute
-fastfwd                     0 # number of insts skipped before timing starts
# -ptrace               <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
-fetch:ifqsize               4 # instruction fetch queue size (in insts)
-fetch:mplat                 3 # extra branch mis-prediction latency
-fetch:speed                 1 # speed of front-end of machine relative to execution core
-bpred                   bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod      2048 # bimodal predictor config (<table size>)
-bpred:2lev       1 1024 4 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb       1024 # combining predictor config (<meta_table_size>)
-bpred:ras                   8 # return address stack size (0 for no return stack)
-bpred:btb        128 2 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update        <null> # speculative predictors update in {ID|WB} (default non-spec)
-decode:width                4 # instruction decode B/W (insts/cycle)
-issue:width                 4 # 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                4 # instruction commit B/W (insts/cycle)
-ruu:size                   16 # register update unit (RUU) size
-lsq:size                    8 # load/store queue (LSQ) size
-cache:dl1        dl1:64:64:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat                1 # l1 data cache hit latency (in cycles)
-cache:dl2        ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat                6 # l2 data cache hit latency (in cycles)
-cache:il1        il1:256:64:1: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                6 # l2 instruction cache hit latency (in cycles)
-cache:flush             false # flush caches on system calls
-cache:icompress         false # convert 64-bit inst addresses to 32-bit inst equivalents
-mem:lat          18 2 # memory access latency (<first_chunk> <inter_chunk>)
-mem:width                   8 # memory access bus width (in bytes)
-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)
-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                   2 # total number of floating point ALU's available
-res:fpmult                  2 # total number of floating point multiplier/dividers available
# -pcstat               <null> # profile stat(s) against text addr's (mult uses ok)
-bugcompat               false # operate in backward-compatible bugs mode (for testing only)

  Pipetrace range arguments are formatted as follows:

    {{@|#}<start>}:{{@|#|+}<end>}

  Both ends of the range are optional, if neither are specified, the entire
  execution is traced.  Ranges that start with a `@' designate an address
  range to be traced, those that start with an `#' designate a cycle count
  range.  All other range values represent an instruction count range.  The
  second argument, if specified with a `+', indicates a value relative
  to the first argument, e.g., 1000:+100 == 1000:1100.  Program symbols may
  be used in all contexts.

    Examples:   -ptrace FOO.trc #0:#1000
                -ptrace BAR.trc @2000:
                -ptrace BLAH.trc :1500
                -ptrace UXXE.trc :
                -ptrace FOOBAR.trc @main:+278
```

```
  Branch predictor configuration examples for 2-level predictor:
    Configurations:   N, M, W, X
      N   # entries in first level (# of shift register(s))
      W   width of shift register(s)
      M   # entries in 2nd level (# of counters, or other FSM)
      X   (yes-1/no-0) xor history and address for 2nd level index
    Sample predictors:
      GAg     : 1, W, 2^W, 0
      GAp     : 1, W, M (M > 2^W), 0
      PAg     : N, W, 2^W, 0
      PAp     : N, W, M (M == 2^(N+W)), 0
      gshare  : 1, W, 2^W, 1
  Predictor `comb' combines a bimodal and a 2-level predictor.

  The cache config parameter <config> has the following format:

    <name>:<nsets>:<bsize>:<assoc>:<repl>

    <name>   - name of the cache being defined
    <nsets>  - number of sets in the cache
    <bsize>  - block size of the cache
    <assoc>  - associativity of the cache
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

    Examples:   -cache:dl1 dl1:4096:32:1:l
                -dtlb dtlb:128:4096:32:r

  Cache levels can be unified by pointing a level of the instruction cache
  hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache
  configuration arguments.  Most sensible combinations are supported, e.g.,

    A unified l2 cache (il2 is pointed at dl2):
      -cache:il1 il1:128:64:1:l -cache:il2 dl2
      -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

    Or, a fully unified cache hierarchy (il1 pointed at dl1):
      -cache:il1 dl1
      -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l



sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn                842550792 # total number of instructions committed
sim_num_refs                255032800 # total number of loads and stores committed
sim_num_loads               171016145 # total number of loads committed
sim_num_stores          84016655.0000 # total number of stores committed
sim_num_branches            120522975 # total number of branches committed
sim_elapsed_time                 1198 # total simulation time in seconds
sim_inst_rate             703297.8230 # simulation speed (in insts/sec)
sim_total_insn              859672798 # total number of instructions executed
sim_total_refs              263260804 # total number of loads and stores executed
sim_total_loads             179138000 # total number of loads executed
sim_total_stores        84122804.0000 # total number of stores executed
sim_total_branches          121719684 # total number of branches executed
sim_cycle                   515261573 # total simulation time in cycles
sim_IPC                        1.6352 # instructions per cycle
sim_CPI                        0.6115 # cycles per instruction
sim_exec_BW                    1.6684 # total instructions (mis-spec + committed) per cycle
sim_IPB                        6.9908 # instruction per branch
IFQ_count                  1641662544 # cumulative IFQ occupancy
```

```
IFQ_fcount                353961031 # cumulative IFQ full count
ifq_occupancy                3.1861 # avg IFQ occupancy (insn's)
ifq_rate                     1.6684 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  1.9096 # avg IFQ occupant latency (cycle's)
ifq_full                     0.6870 # fraction of time (cycle's) IFQ was full
RUU_count                6716942746 # cumulative RUU occupancy
RUU_fcount                237602608 # cumulative RUU full count
ruu_occupancy               13.0360 # avg RUU occupancy (insn's)
ruu_rate                     1.6684 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  7.8134 # avg RUU occupant latency (cycle's)
ruu_full                     0.4611 # fraction of time (cycle's) RUU was full
LSQ_count                2242723383 # cumulative LSQ occupancy
LSQ_fcount                135153363 # cumulative LSQ full count
lsq_occupancy                4.3526 # avg LSQ occupancy (insn's)
lsq_rate                     1.6684 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  2.6088 # avg LSQ occupant latency (cycle's)
lsq_full                     0.2623 # fraction of time (cycle's) LSQ was full
sim_slip                 9945067008 # total number of slip cycles
avg_sim_slip                11.8035 # the average slip between issue and retirement
bpred_bimod.lookups       122325017 # total number of bpred lookups
bpred_bimod.updates       120522975 # total number of updates
bpred_bimod.addr_hits     118590139 # total number of address-predicted hits
bpred_bimod.dir_hits      118682809 # total number of direction-predicted hits (includes addr-hits)
bpred_bimod.misses          1840166 # total number of misses
bpred_bimod.jr_hits        28422318 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen        28430346 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP           39 # total number of address-predicted hits for non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP         1276 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate    0.9840 # branch address-prediction rate (i.e., addr-hits/updates)
bpred_bimod.bpred_dir_rate    0.9847 # branch direction-prediction rate (i.e., all-hits/updates)
bpred_bimod.bpred_jr_rate    0.9997 # JR address-prediction rate (i.e., JR addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP    0.0306 # non-RAS JR addr-pred rate (ie, non-RAS JR hits/JRs s
bpred_bimod.retstack_pushes     29022762 # total number of address pushed onto ret-addr stack
bpred_bimod.retstack_pops     28632363 # total number of address popped off of ret-addr stack
bpred_bimod.used_ras.PP     28429070 # total number of RAS predictions used
bpred_bimod.ras_hits.PP     28422279 # total number of RAS hits
bpred_bimod.ras_rate.PP    0.9998 # RAS prediction rate (i.e., RAS hits/used RAS)
il1.accesses              877454212 # total number of accesses
il1.hits                  866364421 # total number of hits
il1.misses                 11089791 # total number of misses
il1.replacements           11089535 # total number of replacements
il1.writebacks                    0 # total number of writebacks
il1.invalidations                 0 # total number of invalidations
il1.miss_rate                0.0126 # miss rate (i.e., misses/ref)
il1.repl_rate                0.0126 # replacement rate (i.e., repls/ref)
il1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses              261741591 # total number of accesses
dl1.hits                  261423865 # total number of hits
dl1.misses                   317726 # total number of misses
dl1.replacements             317470 # total number of replacements
dl1.writebacks               195736 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0012 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0012 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0007 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses               11603253 # total number of accesses
ul2.hits                   11541038 # total number of hits
ul2.misses                    62215 # total number of misses
ul2.replacements              58119 # total number of replacements
ul2.writebacks                57080 # total number of writebacks
ul2.invalidations                 0 # total number of invalidations
```

```
ul2.miss_rate                0.0054 # miss rate (i.e., misses/ref)
ul2.repl_rate                0.0050 # replacement rate (i.e., repls/ref)
ul2.wb_rate                  0.0049 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
itlb.accesses             877454212 # total number of accesses
itlb.hits                 877454181 # total number of hits
itlb.misses                      31 # total number of misses
itlb.replacements                 0 # total number of replacements
itlb.writebacks                   0 # total number of writebacks
itlb.invalidations                0 # total number of invalidations
itlb.miss_rate               0.0000 # miss rate (i.e., misses/ref)
itlb.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
dtlb.accesses             261755961 # total number of accesses
dtlb.hits                 261754579 # total number of hits
dtlb.misses                    1382 # total number of misses
dtlb.replacements              1254 # total number of replacements
dtlb.writebacks                   0 # total number of writebacks
dtlb.invalidations                0 # total number of invalidations
dtlb.miss_rate               0.0000 # miss rate (i.e., misses/ref)
dtlb.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
rename_power           215361557.0026 # total power usage of rename unit
bpred_power            647318162.2106 # total power usage of bpred unit
window_power           1135801106.6498 # total power usage of instruction window
lsq_power              496289638.3890 # total power usage of load/store queue
regfile_power          1840757591.8683 # total power usage of arch. regfile
icache_power           1165701571.4586 # total power usage of icache
dcache_power           2915073390.5187 # total power usage of dcache
dcache2_power          2168785151.1401 # total power usage of dcache2
alu_power              6080422942.5858 # total power usage of alu
falu_power             3679236845.0226 # total power usage of falu
resultbus_power        1183835566.9784 # total power usage of resultbus
clock_power            11297579083.8019 # total power usage of clock
avg_rename_power             0.4180 # avg power usage of rename unit
avg_bpred_power              1.2563 # avg power usage of bpred unit
avg_window_power             2.2043 # avg power usage of instruction window
avg_lsq_power                0.9632 # avg power usage of lsq
avg_regfile_power            3.5725 # avg power usage of arch. regfile
avg_icache_power             2.2623 # avg power usage of icache
avg_dcache_power             5.6575 # avg power usage of dcache
avg_dcache2_power            4.2091 # avg power usage of dcache2
avg_alu_power               11.8007 # avg power usage of alu
avg_falu_power               7.1405 # avg power usage of falu
avg_resultbus_power          2.2975 # avg power usage of resultbus
avg_clock_power             21.9259 # avg power usage of clock
fetch_stage_power      1813019733.6693 # total power usage of fetch stage
dispatch_stage_power   215361557.0026 # total power usage of dispatch stage
issue_stage_power      13980207796.2619 # total power usage of issue stage
avg_fetch_power              3.5186 # average power of fetch unit per cycle
avg_dispatch_power           0.4180 # average power of dispatch unit per cycle
avg_issue_power             27.1323 # average power of issue unit per cycle
total_power            29146925762.6040 # total power per cycle
avg_total_power_cycle       56.5672 # average total power per cycle
avg_total_power_cycle_nofp_nod2      45.2176 # average total power per cycle
avg_total_power_insn        33.9047 # average total power per insn
avg_total_power_insn_nofp_nod2      27.1021 # average total power per insn
rename_power_cc1       126192532.6166 # total power usage of rename unit_cc1
bpred_power_cc1        123687653.3334 # total power usage of bpred unit_cc1
window_power_cc1       889766746.9270 # total power usage of instruction window_cc1
lsq_power_cc1          79796652.8497 # total power usage of lsq_cc1
```

```
regfile_power_cc1       991751122.0569 # total power usage of arch. regfile_cc1
icache_power_cc1        705392491.0598 # total power usage of icache_cc1
dcache_power_cc1        936024640.5427 # total power usage of dcache_cc1
dcache2_power_cc1       48009536.1099 # total power usage of dcache2_cc1
alu_power_cc1           2354937174.4183 # total power usage of alu_cc1
resultbus_power_cc1     627846504.7025 # total power usage of resultbus_cc1
clock_power_cc1         5032067057.3145 # total power usage of clock_cc1
avg_rename_power_cc1          0.2449 # avg power usage of rename unit_cc1
avg_bpred_power_cc1           0.2400 # avg power usage of bpred unit_cc1
avg_window_power_cc1          1.7268 # avg power usage of instruction window_cc1
avg_lsq_power_cc1             0.1549 # avg power usage of lsq_cc1
avg_regfile_power_cc1         1.9248 # avg power usage of arch. regfile_cc1
avg_icache_power_cc1          1.3690 # avg power usage of icache_cc1
avg_dcache_power_cc1          1.8166 # avg power usage of dcache_cc1
avg_dcache2_power_cc1         0.0932 # avg power usage of dcache2_cc1
avg_alu_power_cc1             4.5704 # avg power usage of alu_cc1
avg_resultbus_power_cc1        1.2185 # avg power usage of resultbus_cc1
avg_clock_power_cc1           9.7660 # avg power usage of clock_cc1
fetch_stage_power_cc1   829080144.3933 # total power usage of fetch stage_cc1
dispatch_stage_power_cc1 126192532.6166 # total power usage of dispatch stage_cc1
issue_stage_power_cc1   4936381255.5501 # total power usage of issue stage_cc1
avg_fetch_power_cc1           1.6090 # average power of fetch unit per cycle_cc1
avg_dispatch_power_cc1        0.2449 # average power of dispatch unit per cycle_cc1
avg_issue_power_cc1           9.5803 # average power of issue unit per cycle_cc1
total_power_cycle_cc1   11915472111.9314 # total power per cycle_cc1
avg_total_power_cycle_cc1       23.1251 # average total power per cycle_cc1
avg_total_power_insn_cc1       13.8605 # average total power per insn_cc1
rename_power_cc2        89821908.6050 # total power usage of rename unit_cc2
bpred_power_cc2         75705927.6269 # total power usage of bpred unit_cc2
window_power_cc2        638654916.6765 # total power usage of instruction window_cc2
lsq_power_cc2           61461045.5775 # total power usage of lsq_cc2
regfile_power_cc2       305427328.4438 # total power usage of arch. regfile_cc2
icache_power_cc2        705392491.0598 # total power usage of icache_cc2
dcache_power_cc2        740396709.7564 # total power usage of dcache_cc2
dcache2_power_cc2       24419599.6654 # total power usage of dcache2_cc2
alu_power_cc2           1277331962.5571 # total power usage of alu_cc2
resultbus_power_cc2     371253747.9957 # total power usage of resultbus_cc2
clock_power_cc2         3111510595.9035 # total power usage of clock_cc2
avg_rename_power_cc2          0.1743 # avg power usage of rename unit_cc2
avg_bpred_power_cc2           0.1469 # avg power usage of bpred unit_cc2
avg_window_power_cc2          1.2395 # avg power usage of instruction window_cc2
avg_lsq_power_cc2             0.1193 # avg power usage of instruction lsq_cc2
avg_regfile_power_cc2         0.5928 # avg power usage of arch. regfile_cc2
avg_icache_power_cc2          1.3690 # avg power usage of icache_cc2
avg_dcache_power_cc2          1.4369 # avg power usage of dcache_cc2
avg_dcache2_power_cc2         0.0474 # avg power usage of dcache2_cc2
avg_alu_power_cc2             2.4790 # avg power usage of alu_cc2
avg_resultbus_power_cc2        0.7205 # avg power usage of resultbus_cc2
avg_clock_power_cc2           6.0387 # avg power usage of clock_cc2
fetch_stage_power_cc2   781098418.6868 # total power usage of fetch stage_cc2
dispatch_stage_power_cc2 89821908.6050 # total power usage of dispatch stage_cc2
issue_stage_power_cc2   3113517982.2287 # total power usage of issue stage_cc2
avg_fetch_power_cc2           1.5159 # average power of fetch unit per cycle_cc2
avg_dispatch_power_cc2        0.1743 # average power of dispatch unit per cycle_cc2
avg_issue_power_cc2           6.0426 # average power of issue unit per cycle_cc2
total_power_cycle_cc2   7401376233.8678 # total power per cycle_cc2
avg_total_power_cycle_cc2       14.3643 # average total power per cycle_cc2
avg_total_power_insn_cc2        8.6095 # average total power per insn_cc2
rename_power_cc3        98738811.0230 # total power usage of rename unit_cc3
bpred_power_cc3         128655718.5853 # total power usage of bpred unit_cc3
window_power_cc3        654052383.3097 # total power usage of instruction window_cc3
lsq_power_cc3           102587557.4351 # total power usage of lsq_cc3
regfile_power_cc3       367654081.6172 # total power usage of arch. regfile_cc3
```

```
icache_power_cc3        751423399.0601 # total power usage of icache_cc3
dcache_power_cc3        939008703.8529 # total power usage of dcache_cc3
dcache2_power_cc3       236497644.4692 # total power usage of dcache2_cc3
alu_power_cc3           1649880542.4448 # total power usage of alu_cc3
resultbus_power_cc3     402496435.3149 # total power usage of resultbus_cc3
clock_power_cc3         3696763866.1950 # total power usage of clock_cc3
avg_rename_power_cc3            0.1916 # avg power usage of rename unit_cc3
avg_bpred_power_cc3             0.2497 # avg power usage of bpred unit_cc3
avg_window_power_cc3            1.2694 # avg power usage of instruction window_cc3
avg_lsq_power_cc3               0.1991 # avg power usage of instruction lsq_cc3
avg_regfile_power_cc3           0.7135 # avg power usage of arch. regfile_cc3
avg_icache_power_cc3            1.4583 # avg power usage of icache_cc3
avg_dcache_power_cc3            1.8224 # avg power usage of dcache_cc3
avg_dcache2_power_cc3           0.4590 # avg power usage of dcache2_cc3
avg_alu_power_cc3               3.2020 # avg power usage of alu_cc3
avg_resultbus_power_cc3          0.7811 # avg power usage of resultbus_cc3
avg_clock_power_cc3             7.1745 # avg power usage of clock_cc3
fetch_stage_power_cc3   880079117.6454 # total power usage of fetch stage_cc3
dispatch_stage_power_cc3 98738811.0230 # total power usage of dispatch stage_cc3
issue_stage_power_cc3   3984523266.8266 # total power usage of issue stage_cc3
avg_fetch_power_cc3             1.7080 # average power of fetch unit per cycle_cc3
avg_dispatch_power_cc3          0.1916 # average power of dispatch unit per cycle_cc3
avg_issue_power_cc3             7.7330 # average power of issue unit per cycle_cc3
total_power_cycle_cc3   9027759143.3071 # total power per cycle_cc3
avg_total_power_cycle_cc3         17.5207 # average total power per cycle_cc3
avg_total_power_insn_cc3         10.5014 # average total power per insn_cc3
total_rename_access         859610769 # total number accesses of rename unit
total_bpred_access          120522975 # total number accesses of bpred unit
total_window_access        3190835736 # total number accesses of instruction window
total_lsq_access            261771603 # total number accesses of load/store queue
total_regfile_access       1257292845 # total number accesses of arch. regfile
total_icache_access         877552572 # total number accesses of icache
total_dcache_access         261741591 # total number accesses of dcache
total_dcache2_access         11603253 # total number accesses of dcache2
total_alu_access            821106635 # total number accesses of alu
total_resultbus_access      906768006 # total number accesses of resultbus
avg_rename_access              1.6683 # avg number accesses of rename unit
avg_bpred_access               0.2339 # avg number accesses of bpred unit
avg_window_access              6.1927 # avg number accesses of instruction window
avg_lsq_access                 0.5080 # avg number accesses of lsq
avg_regfile_access             2.4401 # avg number accesses of arch. regfile
avg_icache_access              1.7031 # avg number accesses of icache
avg_dcache_access              0.5080 # avg number accesses of dcache
avg_dcache2_access             0.0225 # avg number accesses of dcache2
avg_alu_access                 1.5936 # avg number accesses of alu
avg_resultbus_access           1.7598 # avg number accesses of resultbus
max_rename_access                   4 # max number accesses of rename unit
max_bpred_access                    4 # max number accesses of bpred unit
max_window_access                  16 # max number accesses of instruction window
max_lsq_access                      6 # max number accesses of load/store queue
max_regfile_access                 12 # max number accesses of arch. regfile
max_icache_access                   4 # max number accesses of icache
max_dcache_access                   4 # max number accesses of dcache
max_dcache2_access                  4 # max number accesses of dcache2
max_alu_access                      4 # max number accesses of alu
max_resultbus_access                9 # max number accesses of resultbus
max_cycle_power_cc1           46.3333 # maximum cycle power usage of cc1
max_cycle_power_cc2           30.7981 # maximum cycle power usage of cc2
max_cycle_power_cc3           31.7760 # maximum cycle power usage of cc3
sim_invalid_addrs                   2 # total non-speculative bogus addresses seen (debug var)
ld_text_base               0x00400000 # program text (code) segment base
ld_text_size                   124464 # program text (code) size in bytes
ld_data_base               0x10000000 # program initialized data segment base
```

```
ld_data_size                  13588 # program init'ed '.data' and uninit'ed '.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count                  962 # total number of pages allocated
mem.page_mem                  3848k # total size of memory pages allocated
mem.ptab_misses                1275 # total first level page table misses
mem.ptab_accesses        5573662405 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

D. REFERENCES

[1] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *ACM SIGARCH Computer Architecture News*, volume 21, pages 257–266. ACM, 1993.