Chase McFarlane & Andre Mitrik

CMPEN 431 – Project 1 Report

1. **Describe in 100 words or less how the provided framework and its components enable a design space exploration.**

SimpleScalar provides a model allowing virtual testing of components (e.g., CPU, Caches, and Memory Hierarchies) within an architecture to maximize performance based on criteria. Design space exploration is facilitated by given constraints about component parameters (e.g., sizes and associativity for caches) being used in conjunction with logical decision parameters (e.g., scheduling format and replacement policies) to determine the workflow for an optimized framework. These parameters are tested and calculated in a defined exploration order to fit specified criteria (performance vs energy optimization). Ultimately, the goal is to maximize the efficiency ratio of cache sizes to latency.

**List the design points chosen by your DSE.**

\*DPI = Design Point Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **\*DPI** | **Value (unit)** | **Description** |
| Width | 0 | 1 (word) | Only one instruction is issued per clock cycle |
| Scheduling | 0 | -issue:inorder true -issue: wrongpath false | Instructions are issued in order which keeps execution simplier when compared to out of order |
| L1 Block Size | 2 | 32 (bytes) | The data cache blocks are 32 bytes in size (i.e., “wide”) |
| L1 Data Sets | 2 | 128 (sets) | The data cache is divided into 128 sets |
| L1 Data Associativity | 0 | 1 | The cache is 1-way associative which means each set only has one block of 32-bytes |
| L1 Instruction Sets | 6 | 2048 (sets) | The instruction cache is divided into 2048 sets |
| L1 Instruction Associativity | 0 | 1 | The instruction cache is 1-way associative, meaning each set has one block 32-bytes |
| Unified L2 Sets | 2 | 1024 (sets) | The unified L2 cache (instruction and data) is divided into 1024 sets |
| Unified L2 Block Size | 3 | 128 (bytes) | The unified L2 cache is 128 bytes in size |
| Unified L2 Associativity | 1 | 2 | The unified L2 cache is 2-way associative, meaning each set has 2 blocks of 128-bytes (1 set = 256 bytes) |
| Cache & TLB Replacement Policy | 0 | l | The cache and TLB’s blocks are replaced using a least recently used (LRU) policy on misses |
| Floating Point Unit Width | 0 | 1 | The floating point unit has a width of 1 |
| Branch Predictor Choice | 4 | -bpred comb -bpred:comb 1024 | The branch predictor is a size of 1024 entries with a comb policy |
| Return Address Stack Size | 3 | 8 (# entries) | The RAS contains 8 return address entries, meaning that 8 function calls’ return addresses can be put into the stack |
| Branch Target Buffer | 0 | 128 (# sets) 16 (associativity) | The BTB is divided into 128 sets with each set having 16 blocks of data (branch target addresses) |
| L1 D$ Latency | 1 | 2 (time) | The data cache is the smallest of the caches which usually correlates to the lowest latency |
| L1 I$ Latency | 5 | 6 (time) | The instruction cache is bigger than data cache |
| Unified L2 Latency | 4 | 9 (time) | The unified L2 cache is the largest of the caches which evident by the highest latency |

\*DPI = Design Point Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **\*DPI** | **Value (unit)** | **Description** |
| Width | 0 | 1 (word) | Only one instruction is issued per clock cycle |
| Scheduling | 0 | -issue:inorder true -issue: wrongpath false | Instructions are issued in order which keeps execution simplier when compared to out of order |
| L1 Block Size | 2 | 32 (bytes) | The data cache blocks are 32 bytes in size (i.e., “wide”) |
| L1 Data Sets | 2 | 128 (sets) | The data cache is divided into 128 sets |
| L1 Data Associativity | 0 | 1 | The cache is 1-way associative which means each set only has one block of 32-bytes |
| L1 Instruction Sets | 5 | 1024 (sets) | The instruction cache is divided into 2048 sets |
| L1 Instruction Associativity | 0 | 1 | The instruction cache is 1-way associative, meaning each set has one block 32-bytes |
| Unified L2 Sets | 1 | 512 (sets) | The unified L2 cache (instruction and data) is divided into 1024 sets |
| Unified L2 Block Size | 3 | 128 (bytes) | The unified L2 cache is 128 bytes in size |
| Unified L2 Associativity | 1 | 2 | The unified L2 cache is 2-way associative, meaning each set has 2 blocks of 128-bytes (1 set = 256 bytes) |
| Cache & TLB Replacement Policy | 0 | l | The cache and TLB’s blocks are replaced using a least recently used (LRU) policy on misses |
| Floating Point Unit Width | 0 | 1 | The floating point unit has a width of 1 |
| Branch Predictor Choice | 4 | -bpred comb -bpred:comb 1024 | The branch predictor is a size of 1024 entries with a comb policy |
| Return Address Stack Size | 3 | 8 (# entries) | The RAS contains 8 return address entries, meaning that 8 function calls’ return addresses can be put into the stack |
| Branch Target Buffer | 0 | 128 (# sets) 16 (associativity) | The BTB is divided into 128 sets with each set having 16 blocks of data (branch target addresses) |
| L1 D$ Latency | 1 | 2 (time) | The data cache is the smallest of the caches which usually correlates to the lowest latency |
| L1 I$ Latency | 4 | 5 (time) | The instruction cache is bigger than data cache |
| Unified L2 Latency | 3 | 8 (time) | The unified L2 cache is the largest of the caches which evident by the highest latency |

1. **Fill out the following table as detailed below.**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Performance** | **EDP** |
| Width | Value = 0  Why = 1 intr issue is simpler and faster | Value = 0  Why = 1 instr issue is simpler and requires less power ( 1 nW vs 2/4/8 mW) |
| Scheduling | Value = 0  Why = issuing in order reduces complexity thus increases issuing performance | Value = 0  Why = issuing in order reduces complexity and requires less power ( 1 mW vs 1.5 mW) |
| L1 Block Size | Value = 2  Why = larger block sizes reduce miss rate thus increase performance | Value = 2  Why = larger block sizes reduce miss rate thus decrease miss penalty (delay) |
| L1 Data Sets | Value = 2  Why = | Value = 2  Why = fewer sets decrease overall cache size thus decrease power draw |
| L1 Data Associativity | Value = 0  Why = direct mapping only requires one comparator which proved to be more impactful than increasing hit rate | Value = 0  Why = 1-way only needs one comparator which requires less hardware compared to 2/4/8-way thus less power consumption |
| L1 Instruction Sets | Value = 6  Why = larger sets increase the cache size and hit rate thus increasing performance | Value = 5  Why = slightly smaller (compared to DSE perf.) because less |
| L1 Instruction Associativity | Value = 0  Why = | Value = 0  Why = |
| Unified L2 Sets | Value = 2  Why = | Value = 1  Why = |
| Unified L2 Block Size | Value = 3  Why = | Value = 3  Why = |
| Unified L2 Associativity | Value = 1  Why = | Value = 1  Why = |
| Cache & TLB Replacement Policy | Value = 0  Why = | Value = 0  Why = |
| Floating Point Unit Width | Value = 0  Why = | Value = 0  Why = |
| Branch Predictor Choice | Value = 4  Why = | Value = 4  Why = |
| Return Address Stack Size | Value = 3  Why = | Value = 3  Why = |
| Branch Target Buffer | Value = 0  Why = | Value = 0  Why = |
| L1 D$ Latency | Value = 1  Why = | Value = 1  Why = |
| L1 I$ Latency | Value = 5  Why = | Value = 4  Why = |
| Unified L2 Latency | Value = 4  Why = | Value = 3  Why = |

1. **Plots as defined below.**

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

1. **Describe a more sophisticated heuristic which you expect will perform design space exploration (limited by 1000 design points) more effectively to find a better performing design (with respect to execution time).**
2. **Elaborate on any 2 new insights you gained while working on this project.**

SimpleScalar is a simulator for computer architectures that can emulate the performance of an architecture without having to build the physical chips or components. This virtualization facilitates the host machine to keep operating under its current architecture (e.g., x86) while simulating another ISA (e.g., MIPS). It can be similarly compared to how virtual machines allow different operating systems to run on top of another unrelated operating system. Given this description of the overbearing architecture, two insights our group gained working on this project include:

1. The importance of virtualized simulator hardware components to save on R&D costs
2. How to develop an effective way to approach a potential architecture’s component design given a list of input parameters/constraints. Additionally, valuable insight was provided depicting the importance of the sequence of how these constraints are used and the effect the sequence has on the final output.
3. **List of additional resources used (optional).**

* TA Office Hours
* C++ math library (for log2 function)
* <https://www.geeksforgeeks.org/c-plus-plus/>
* http://www.simplescalar.com/
* https://en.wikipedia.org/wiki/Microarchitecture\_simulation
* http://www.ecs.umass.edu/ece/koren/architecture/Simplescalar/Simulators.pdf

1. **Additional information or comments (optional)**

N/A