ECE 568 Project 1 Report

Single-Core Design Space Exploration using SimpleScalar

Chunyang Hui
ECE Depart. College of Engineering
University of Illinois at Chicago
Chicago, IL
Chui21@uic.edu

***Abstract*—This report gives a presentation for single-core design space exploration(DSE) using SimpleScalar simulator.**

Keywords: single-core, DSE, simplescalar

I. INTRODUCTION

In this project, we are asked to do simulations using different parameters provided by simplescalar to see the effect caused by changing these parameters to get a better understanding in computer architecture design.

The main procedure of completing this project is as follows:

1. Get familiar with simplescalar simulator. Get known the basic instructions like how to adjust parameters.
2. Understand the meaning for different parameters and think about what kind of effect will they cause.
3. Do individual design space exploration simulations for all parameters listed. Change each parameter 3-5 times.
4. Find the related parameters of a group.
5. Do intra-group individual design space exploration simulations.
6. Do cross-group individual design space exploration simulations.
7. Try to understand the reason.

II. INDIVIDUAL DSE

In this section, I will talk about how I generate result for single parameter simulation.

A. Write a script to run a simulation

Change the parameter from the default configuration file and dump to a new config file. Use sim-outorder to read the new config file and run eeg file and print the result to a text file and only remain the result with the data we need.

```
###############
echo "1. Predictor Type" >> result_1.txt
#dump new configuration to file `new_cfg`
sim-outorder -bpred 2lev -dumpconfig new_cfg
echo "(1) 2lev" >> result_1.txt
#run simulation reading `new_cfg`
sim-outorder -config new_cfg eeg 2>&1 | grep -e \
"sim_IPC" -e "sim_CPI" -e "avg_total_power_cycle_cc3" | tee -a result_1.txt
```

Fig.1 Example of a simulation script file

The whole script file is attached in the *zip* file.

B. Plot the result

In the graph, the X-axis means performance, the bigger the better. And the Y-axis means EDP, energy-delay product, the smaller the better. Thus the dot which falls on the right corner of the graph means it might be a good parameter option.

1. Predictor Type:

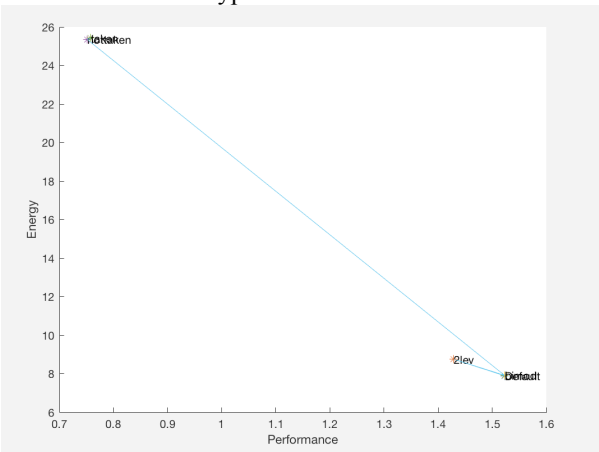

Fig.2 Simulation result for Predictor type.

We can see from the graph that 2lev predictor and bimod predictor have a better performance and lower EDP which are better choices as for the predictor type.

2. Bimodal Predictor Config

We can see that the table size doesn't really affect the performance based on the number we choose. Maybe because the smallest number we choose is big enough to handle.

3. 2-Lev Predictor Config

Similar with bimodal predictor simulation.

4. Return Address Stack Size

The return address stack shouldn't be too big or small. And when it is 16, we get the best result. they're fixed size stacks of return addresses, which only generate predictions for return instructions. they don't help at all for procedure call instructions

5. BTB config

BTB is Branch target buffer, which are fixed-size hash-tables that map from program counter to branch target addresses. The first parameter is the number of sets or size, and the second parameter is the associativity, which can be regarded as number of ways. Higher associativity will increase the hit rate but will consume more power. We can see from the graph that <512,4> is a good choice while <128,4> also is because although it has a poorer performance, its energy is less. And Too big size or associativity don't help in this case.

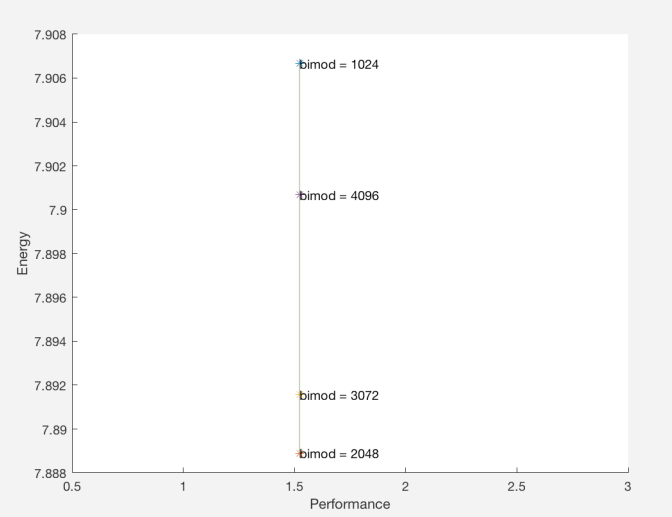

Fig.3 Simulation result for Bimodal Predictor Config.

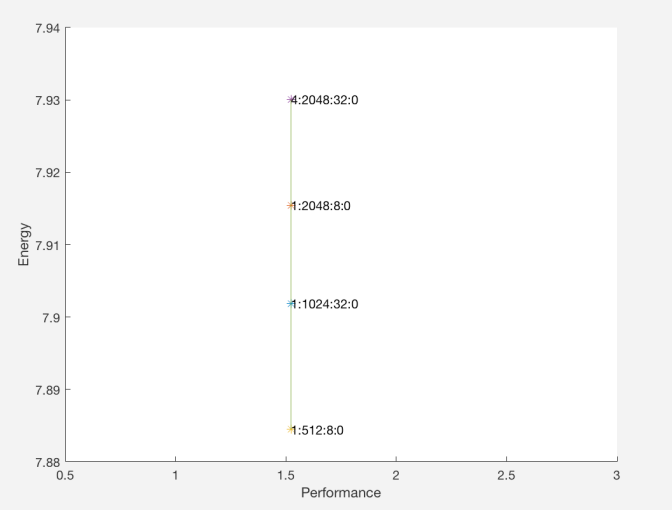

Fig.4 Simulation result for 2-Lev Predictor Config

6. L1 Data Cache Config dl1

Bigger size of cache will enhance the performance. But it will also cause the raise of power.

7. L2 Data Cache Config dl2

Similar with dl1.

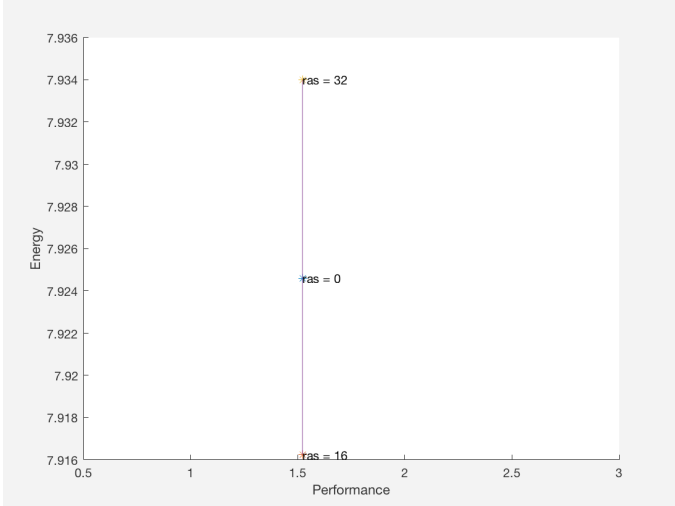

Fig.5 Simulation result for Return Address Stack Size

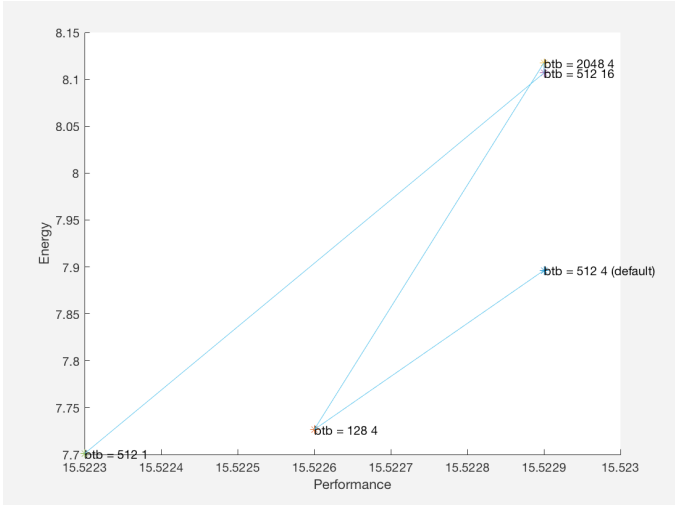

Fig.6 Simulation result for BTB config

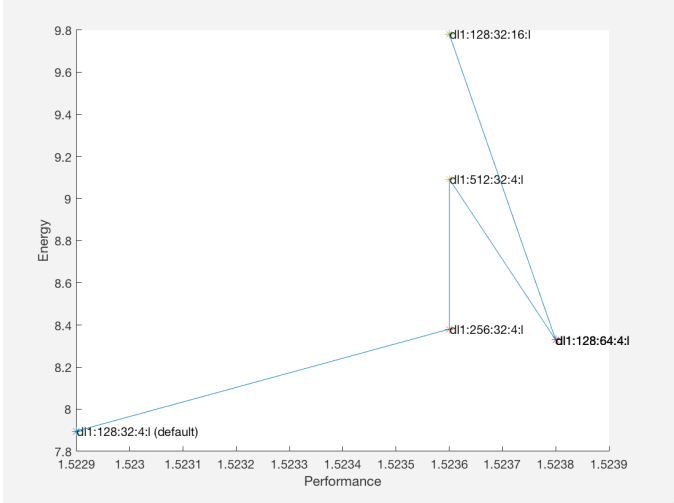

Fig.7 Simulation result for L1 data cache

8. L1 Instruction Cache Config il1

Similar with dl1.

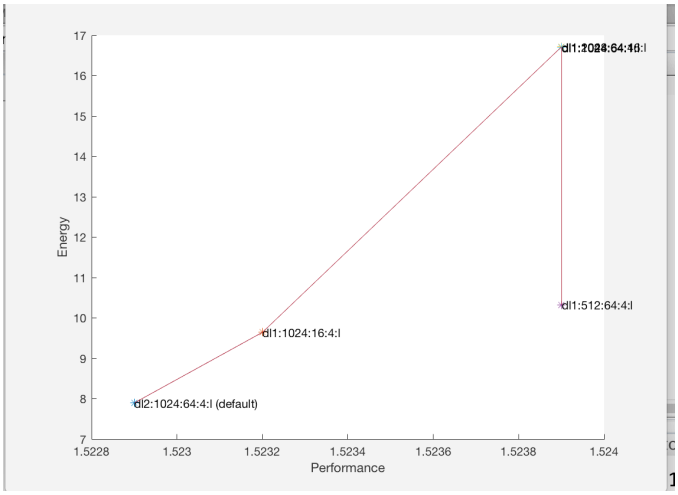

Fig.8 Simulation result for L2 data cache

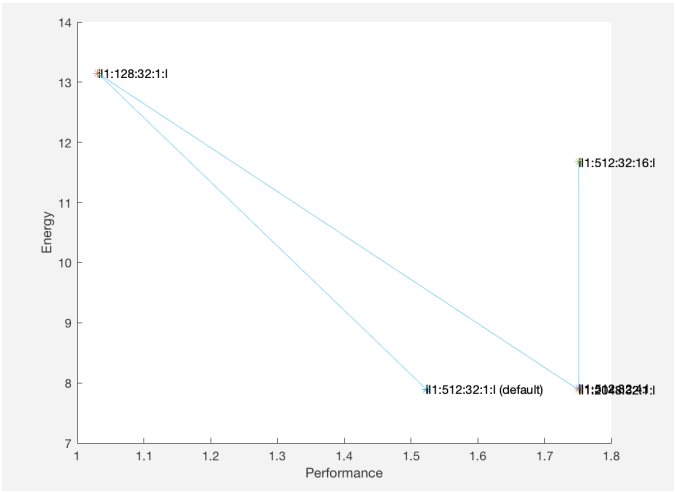

Fig.9 Simulation result for L1 instruction cache

9. Integer ALUs available

The increment of number of integer ALU doesn't bring any upgrade on performance. Maybe it is because the default number can offer enough ALU function.

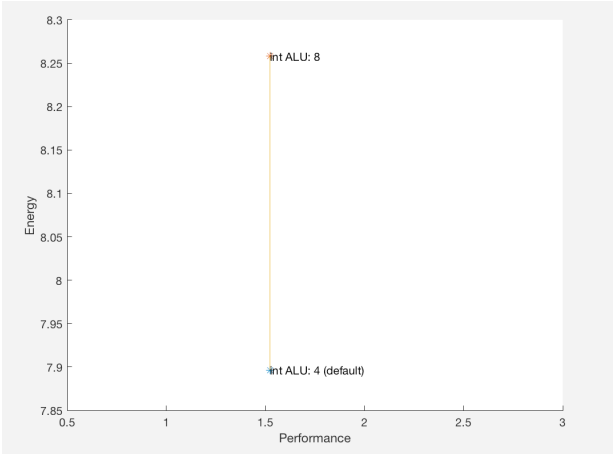

Fig.10 Simulation result for integer ALUs

10. Integer Multipliers/Divider

Similar with integer ALUs.

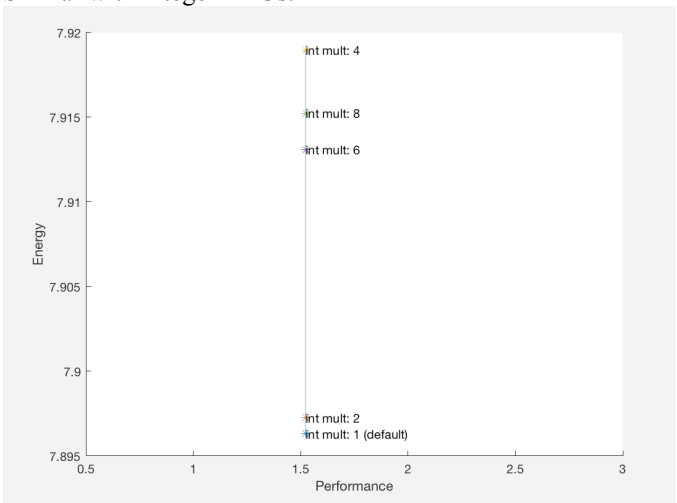

Fig.11 Simulation result for integer multipliers/divider

11. Floating Point ALUs

Similar with integer ALUs.

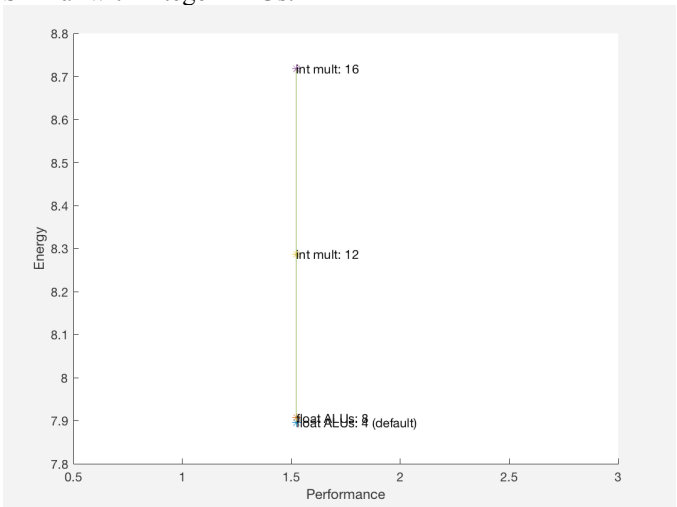

Fig.12 Simulation result for floating point ALUs.

12. Floating Point Multipliers/Divider

When increasing the number of floating point multiplier /divider, we get a better performance. But when there are too many, the only change is the power consumption.

13. Instruction Fetch Queue Size

We can see that we can get a better performance and EDP when the IFQ is big enough. Because if the IFQ is bigger, more instruction can be fetched at the same time.

14. Instruction Decode Width

We see that when the decode width is 4, it can already be able to handle the instruction decoding. However, if we change the instruction fetch queue size, things might be different.

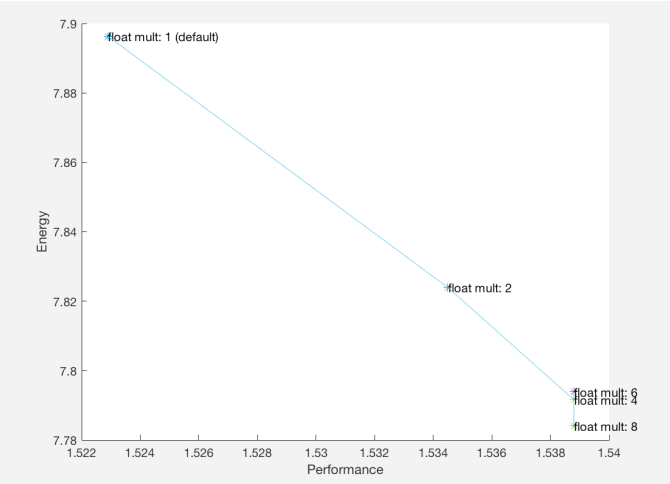

Fig.13 Simulation result for floating point multipliers/divider

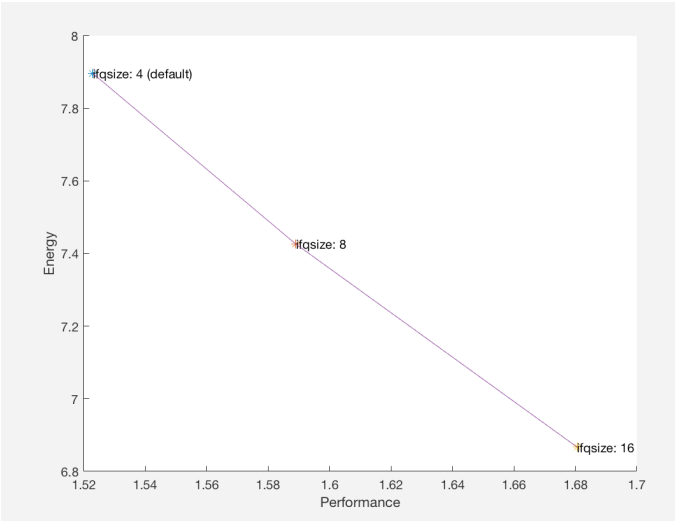

Fig.14 Simulation result for ifqsize

15. Instruction Issue Width

Similar with instruction decode width.

16. Instruction Issue In-order

When issue in-order is true, all the instructions can only be fetched in the order it comes, which will will consume a lot of time waiting for the end of the previous instructions. So out-order is a better choice.

17. Instruction Commit Width

Similar with instruction decode width. But it seems a bigger commit width gives a better performance. Of course along with the increment of the power consumption.

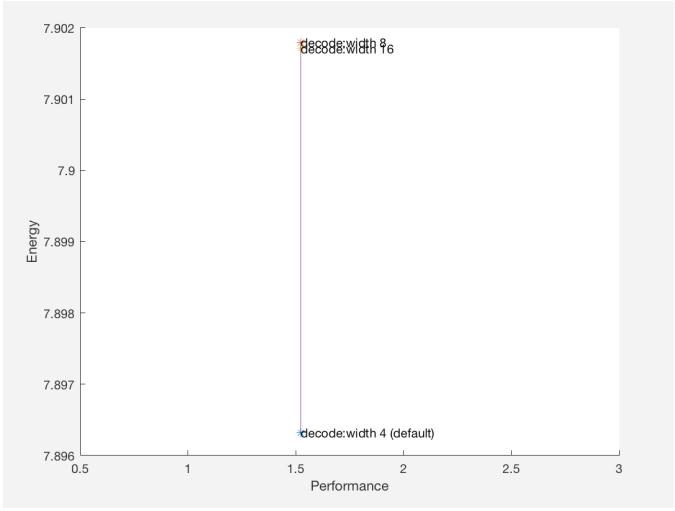

Fig.15 Simulation result for decode width

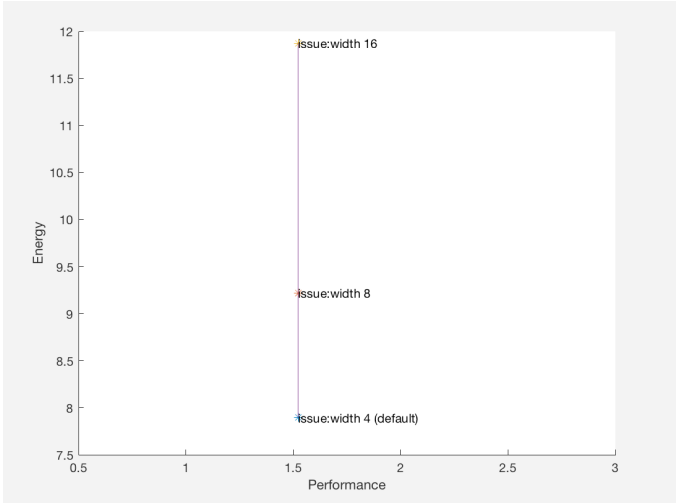

Fig.16 Simulation result for issue width

18. Register Update Unit Size

When the size becomes bigger, the performance and the power raise almost the same.

19. Load/Store Queue Size

Similar with commit width.

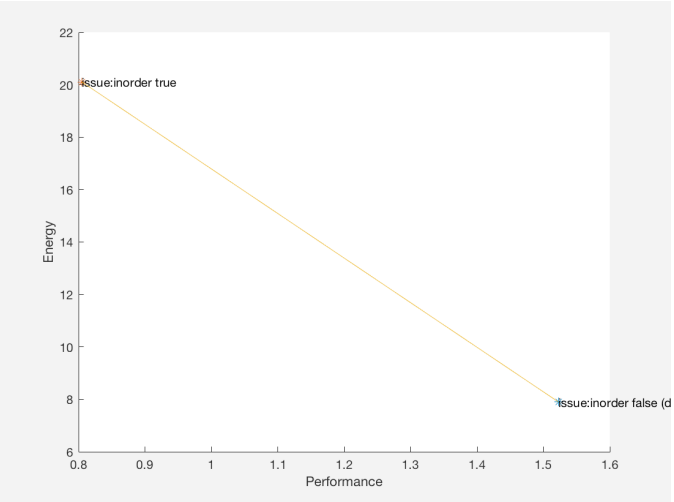

Fig.17 Simulation result for issue inorder or outorder

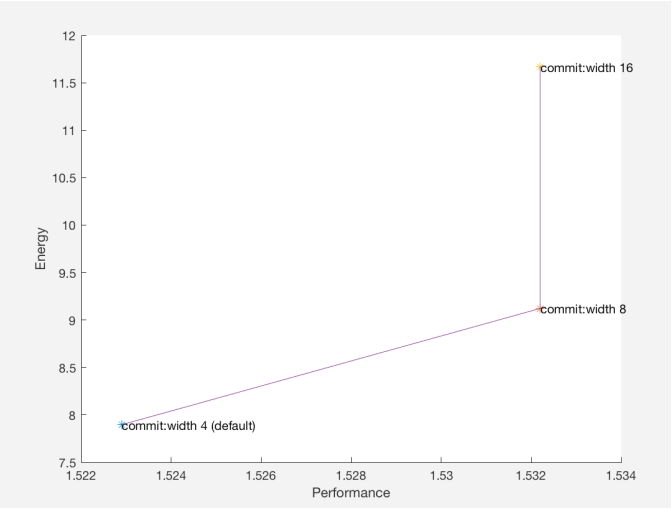

Fig.18 Simulation result for commit width

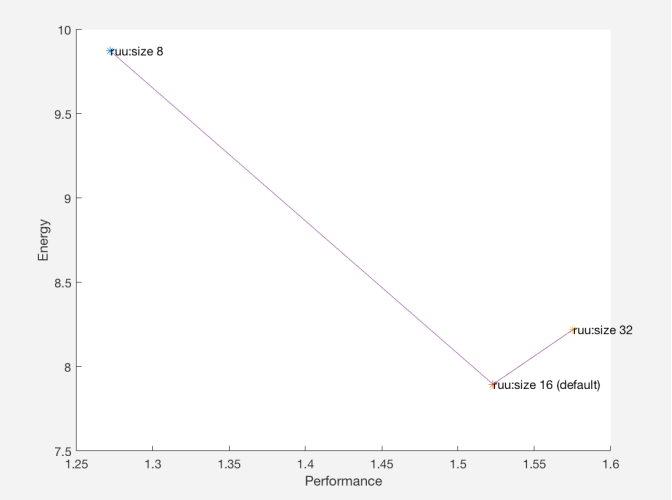

Fig.19 Simulation result for register update unit size

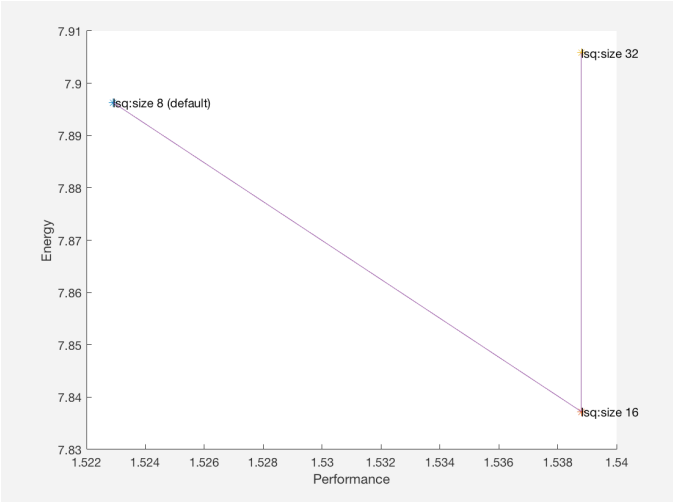

Fig.20 Simulation result for load/store queue size

III. INTRA-GROUP DSE

In this section, I will change a number of parameters of a same group to get a better view of the effect cause by these parameters.

A. Branch Prediction Group

To choose parameters to change together, I will divide it into two directions, one is low-power, the other is good-performance.

(1) Low power:

- bpred

2lev, taken, nottaken will give us a lower power based on the result we got from last section.

- bpred:2lev

We can also change the size when we use 2 level predictor.

- bpred: ras, bpred:btb

We can decrease the size of return address stack (RAS) and branch target buffers to get smaller power consumption.

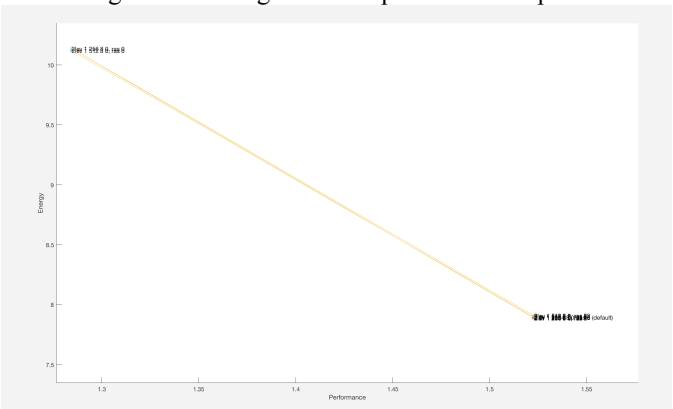
Fig.21 Simulation result for 2lev and RAS

(2) Good performance:

- bpred

bimod can giveas a good performance.

- bpred:bimod

We can change the bimod table size.

- bpred: ras, bpred:btb

We can try increasing the size of these two buffers to see whether we get a better performance.

And then we can get result. I use low power as an example.

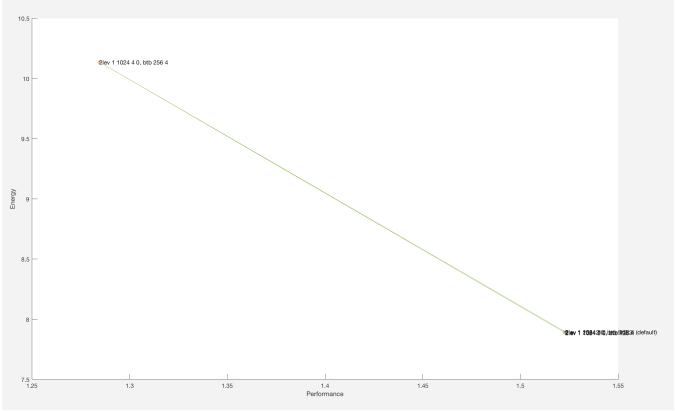

Fig.22 Simulation result for 2lev and BTB.

From the result we get, the default configuration is good enough.

B. *Memory System*

From the result we get, if we increase the size of the memory we will get a slightly better performance but not explicit enough but will extremely increase the power consumption.

I will choose to combine:

(1) level-1 data cache and level-2 data cache

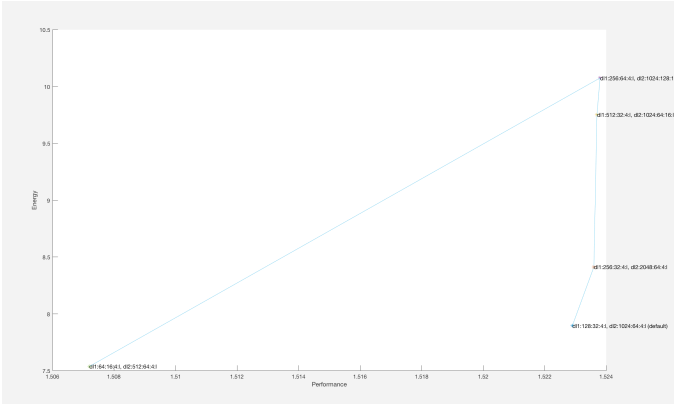

Fig.23 Simulation result for dl1 and dl2

(2) level-1 data cache and level-1 instruction cache

And get the result:

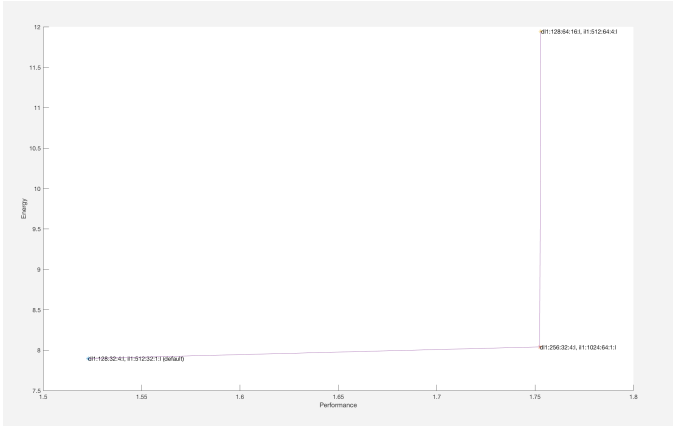

Fig.24 Simulation result for dl1 and il1

C. *Function Units*

Function units are easy to understand. From the result we got previously, the number of integer ALUs doesn't really affect the performance and only causing a big raise on power consumption. I believe it's because the speed of ALU is fast enough and the number of ALU can provide enough performance. And the same for integer multiplier and floating point ALUs. However, when I use more floating point multipliers, the performance became much better. The reason might be for simulation we do IFFT which has a lot of floating point multiplication.

Thus, in this simulation, the only parameter that will upgrade performance is the number of floating point multipliers.

D. *Data Path & others*

From the result we get, instruction fetch queue (IFQ) size will increase the performance and if choose a bigger decoding width ,we might be able to get a better performance. Thus, first set we can choose -fetch:ifqsize and -decode:width. And we get this:

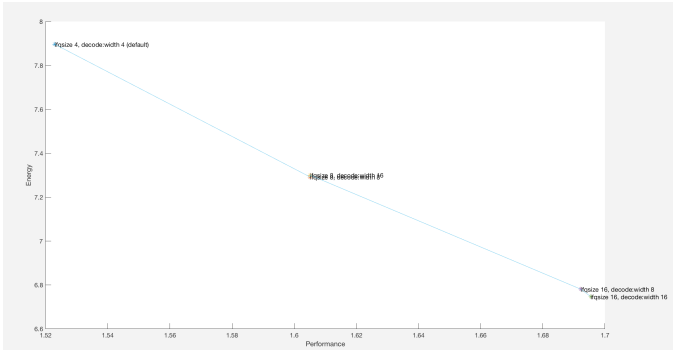

Fig.25 Simulation result for ifqsize and decode width

And We can see with bigger ifqsize and decode width simultaneously, we can get big gain on the performance.

And if we can have a bigger issue width, we should be able to handle more instructions, so for second set, we add a -issue:width to the parameters we choose before.

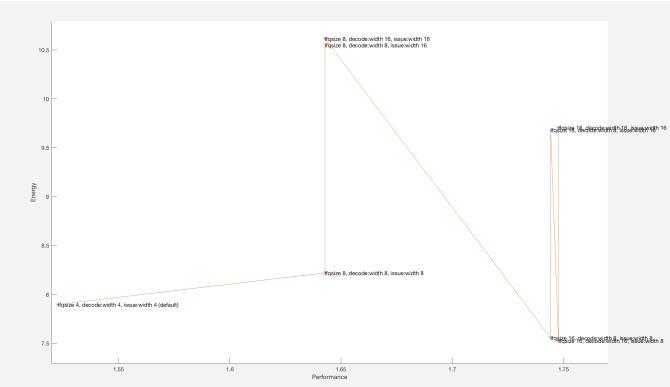

Fig.26 Simulation result for ifqsize and decode width and issue width

And We can see that increase the IFQ size is always a better choice. Because all instructions need to be fetched first and then operated.

And also if we increase the commit width, we can commit more instructions in fetch queue order so we should also have a better performance. So we add this parameter too.

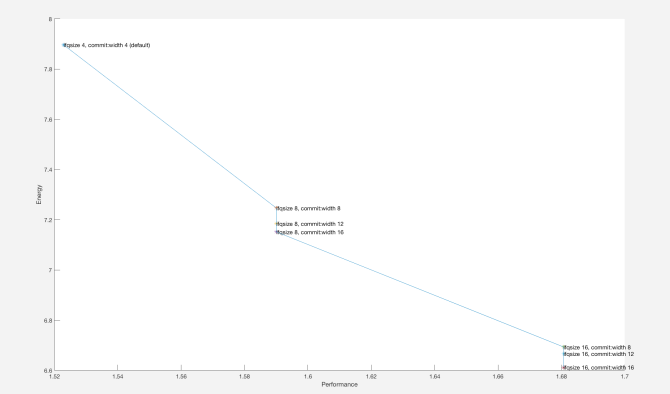

Fig.27 Simulation result for ifqsize and commit width

We can see from the graph increase the commit width will reduce the power. So we need bigger commit width.

IV. Cross-Group DSE

In this section, I will try to find a combination of parameters which might give us the best performance and the lowest energy consumption based on the results we obtain in the former simulations.

A. High Performance

For branch prediction I will use bimod which is default and also 2lev. And when using bimod, I will change the table size to 2048 and when using 2lev, I will change the 2-lev predictor to 1, 512, 8, 0. And for the BTB size, I will change it to 512, 4 to get higher performance.

For memory system, the default configuration has a good balance on performance and power. Nothing will change on this.

As for function unit, the total number of floating point multiplier/divider is changed to 4. And Others keep the default.

And for other parameters, increase the instruction fetch queue size to 16, and the instruction commit width to 8 and 16. And decode width to 16 and issue width to 8. For register update unit, change the size to 32 for performance and for load/store queue, change to 16.

B. Low Power

For branch prediction, the difference from the high performance is that for BTB size, 128, 4 will be used. And Register Unit size will be change to 16.

And we get result.

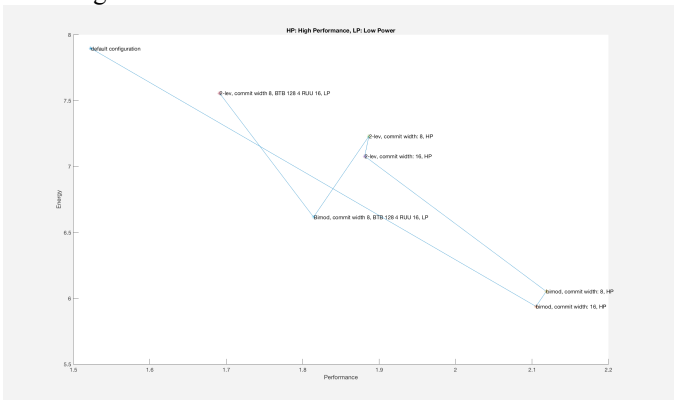

Fig.27 Simulation result for cross group DSE.

Here we can see that bimod has a better predictor performance. And bigger commit width will decrease the power.

References

[1] http://www.simplescalar.com/

[2] http://www.cse.chalmers.se/edu/year/2009/course/DAT105/old_project_2008/DAT105_Tools.pdf

[3] https://courses.cs.washington.edu/courses/csep548/06au/lectures/multipleIssue.pdf

[4] http://www.eecs.umich.edu/courses/eecs470/lectures/470L11W14.pdf