# CSE530: Assignment#4 Report

Ajay Narayanan Sridhar, PSU ID: 970750943

## Introduction

In this report, we will be discussing and highlighting the results of ablation studies to understand the effect of changing Reorder buffer entries (ROB), pipeline width (PW), and branch predictor (BP) on an out-of-order core processor using the quicksort benchmark from Stanford LLVM[1]. We will be using gem5[2] as the simulation tool for our experiments. Kindly refer to this GitHub project (https://github.com/ajaynarayanan/cse_530) to find the python jupyter notebook for automation code and generation of graphs.

## Experiments and Discussions

Our search space for Reorder Buffer (ROB) entries, Branch predictor (BP) and pipeline width (PW) are the following,   ROB: [16, 32, 64, 128, 192]; BP: [2-bit, tournament, bimode, ltage]; PW: [2, 4, 6, 8]

We performed 5x4x4 = 80 experiments to figure out the parameters for a OoO core configuration which has the performance benefits of LargeCore  (PW=8, ROB=192, BP='ltage') with fewer resources, say we call this GoodCore. Since the analysis search space is huge, we label four OoO configurations (SmallCore, LowerCore, UpperCore, LargeCore) and hope to compare their relative performance to produce GoodCore configuration. We label the following OoO core processor configurations for ease of reference,

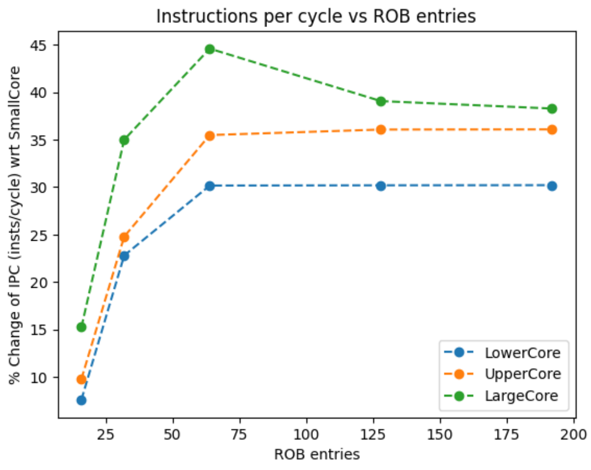
SmallCore: (PW=2, ROB=16, BP='2-bit'), LowerCore:  (PW=4, ROB=32, BP='tournament')

UpperCore: (PW=6, ROB=64, BP='bimode'), LargeCore: (PW=8, ROB=192, BP='ltage')

NOTE: We abuse the notation of the above labeling in our experiments. Although we change a parameter of a configuration, we still call it the same label. (Eg: In Fig 1a, we mark blue points as LowerCore even though we vary the ROB entries). This notation is justified for it results in legible and neat graphs.

### 1. Impact on varying ROB entries

We experiment with four labeled CPU configurations and observe how the benchmark behaves when ROB entries are changed while having branch predictor and pipeline width constant. We observe from Fig 1a and Fig 1b that after a point, increasing ROB entries saturates the perfomance improvement in IPC and execution time. We hypothesize that this is due to the fact the quicksort benchmark can't be parallelized more than a point to take advantage of register renaming. Based on this experiment, we find that ROB entries' values from 50 to 75 will be ideal. We choose the value of 64 for our GoodCore.



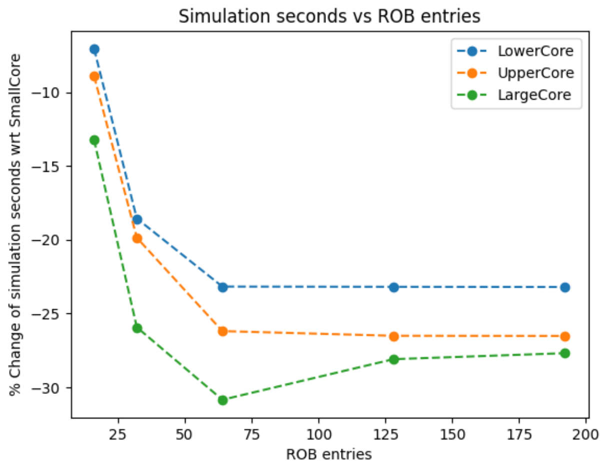
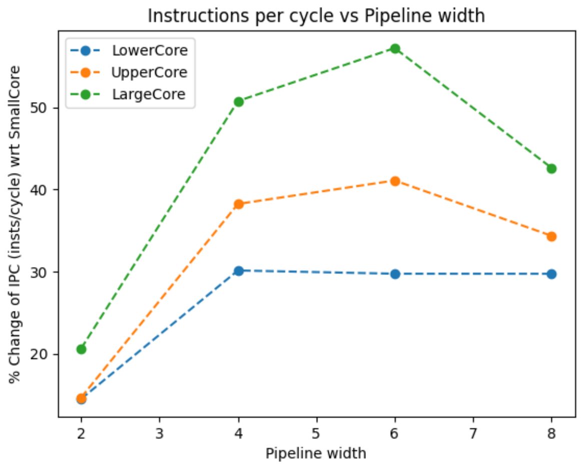


Fig 1a: Line graph on the left showing the percentage change of IPC (insts/cycle) vs ROB entries for LowerCore, UpperCore and LargeCore with respect to SmallCore. The values of pipeline width and branch predictor are fixed while varying #ROB values.

Fig 1b: Line graph on the right showing the percentage change of simulation seconds (seconds) vs ROB entries for LowerCore, UpperCore and LargeCore with respect to SmallCore. The values of pipeline width and branch predictor are fixed while varying #ROB values.

## 2. Impact on changing pipeline width

We vary CPU pipeline width from 2 to 8 in steps of 2 for the four labeled CPU configurations while keeping ROB entries and branch predictor fixed. In Fig 2a and 2b, we notice that increasing pipeline width more than 6 results in decrease in performance of IPC and simulation seconds/execution time. We believe this is because issuing more than 6 insts at one instant will result into more instructions waiting for resources due to RAW dependenices for the quicksort benchmark. This leads to stalling and thereby decreasing IPC. Based on these observations, we choose a pipeline width of 6 for our GoodCore.



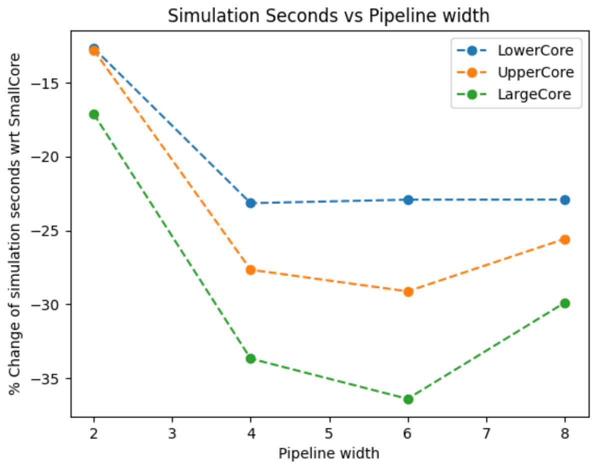
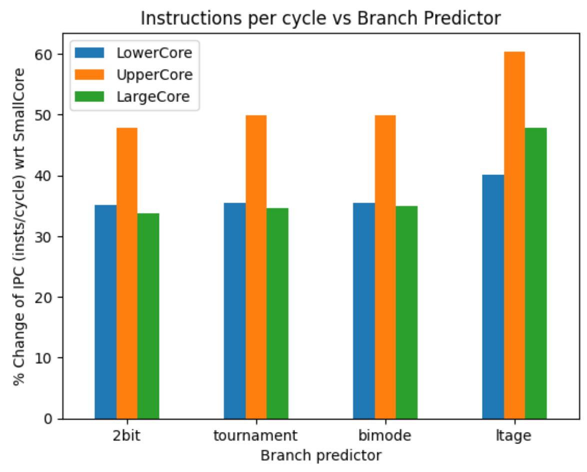


Fig 2a: Line graph on the left showing the percentage change of IPC (insts/cycle) vs Pipeline width for LowerCore, UpperCore and LargeCore with respect to SmallCore. The values of ROB entries and branch predictor are fixed while varying pipeline width.

Fig 2b: Line graph on the right showing the percentage change of simulation seconds (seconds) vs Pipeline width for LowerCore, UpperCore and LargeCore with respect to SmallCore. The values of ROB entries and branch predictor are fixed while varying pipeline width.

## 3. Impact on varying branch predictor

We experiment with four branch predictors for our four labeled CPU configurations. Fig 3a, 3b show the impact on IPC and execution time. We observe that percentage change for LTAGE is the highest compared to other branch predictors across all the CPU configurations. This observation is in-line with the benchmark's charactersitscs. Note that in quicksort algorithm, we iterate over the array multiple times, hence a loop predictor would be beneficial here. Thus, we choose L-TAGE as the branch predictor for our GoodCore.



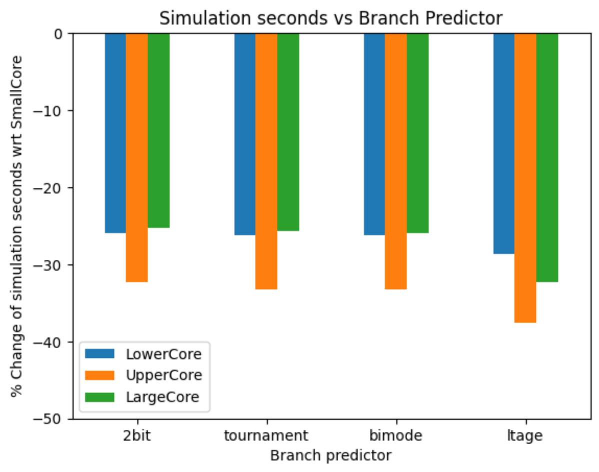


Fig 3a: Bar chart on the left showing the percentage change of IPC (insts/cycle) vs branch predictor for LowerCore, UpperCore and LargeCore with respect to SmallCore. The values of ROB entries and pipeline width are fixed while varying branch predictor.

Fig 3b: Bar chart on the right showing the percentage change of simulation seconds vs branch predictor for LowerCore, UpperCore and LargeCore with respect to SmallCore. The values of ROB entries and pipeline width are fixed while varying branch predictor.

## GoodCore Configuration

After the above experiments, we have arrive at a configuration for GoodCore: (PW=6, ROB=64, BP='ltage'). From Table Q4a, we notice that GoodCore has a speed of 1.68 compared to SmallCore in terms of execution time.

**Q1) What is the benchmark you picked? Also paste the link to the zip folder for relevant simulation outputs & scripts. (1 line)**

Quicksort benchmark.
https://drive.google.com/drive/folders/1EpE9J7OXvlaKZ83zio8NeC9yEHH1YoRf?usp=sharing

**Q2) Which core parameter(s) have the most impact on performance? Are there any dependencies between the parameters (e.g., do you need to change two or more parameters at once to have an impact on the performance)? (Max 150 words - 20 points)**

Branch predictor has the most impact on performance with almost 60% percent change when compared to SmallCore CPU configuration. We hypothesize that this because of the frequent number of branch instructions involved in the quicksort benchmark. Yes, from Fig. 3a, we note that changing branch predictor from 2-bit to tournament doesn't result in huge changes for LargeCore configuration. However, changing the pipeline width and rob entries result in better changes (I.e, going from orange to green in Fig. 3a)

**Q3) What are the parameters you chose? Let us call this, the GoodCore. (3 lines - 15 points)**

Reorder Buffer entries: 64 ; Branch predictor: L-TAGE ; Pipeline width: 6

**Q4) What are the IPC improvements and execution time speedups of LargeCore and GoodCore with respect to the SmallCore? (3 lines / 1 table - 15 points)**

| | GoodCore | LargeCore |
|---|---|---|
| Execution time speedup | 1.6849 | 1.5530 |
| IPC percentage change | 68.49 % | 55.31 % |

Table Q4a: Improvements in IPC and execution time of GoodCore and LargeCore with respect to SmallCore values.

**Q5) How did the benchmark characteristics affect your GoodCore design decisions? Look at the benchmark code (both the .c and .s files may be useful). (Max 150 words - 30 points)**

After going through Quicksort benchmark's assembly code and source code, we realize that there are a lot of branching statements in terms of while loop and if conditions. This hints us that we need a powerful branch predictor.  Also, the number of data dependencies are approximately less than 150 which hints us that ROB can be limited. Finally, quicksort's assembly code has some data dependency between every 4 to 6 insts approximately. Thus, issuing more than 6 insts will result in stalling and squashing.

**Q6) How did you reduce the search space? (Max 50 words - 20 points)**

We use the principle of divide and conquer. We split the search space into approx equal components (four levels of CPU configurations); the divide phase; and then we idenfity patterns on each component. Based on these patterns, we figure out the best core parameters.

## Conclusions

Based on the discussion and graphs, we understand that quicksort's benchmark is highly sensitive to the branch predictor on an OoO processor. We conclude that increasing resources after a point might not lead to optimal performance, which is verified by the better performance of GoodCore compared to LargeCore in terms of IPC and execution time.

## References

[1.] https://github.com/llvm/llvm-test-suite/blob/main/SingleSource/Benchmarks/Stanford

[2.] https://www.gem5.org/documentation/

[3.] https://jilp.org/vol8/v8paper1.pdf