Advanced Computer Architecture
Autumn Semester 2025-26
End-Sem Project

Topic: Automated SimpleScaler IPC optimizer

[Github repository]

Name – Soham Chakraborty
Roll No – 22CS02002
Group No. - 4

Goal:

Producing the most optimal IPC for a user input C code by changing the default.cfg configuration file to an optimal configuration which produces the best possible IPC, by performing an exhaustive search on all possible combinations of varying range of parameters, which is given by user.

Approach:

- Write a C code (compatible with gcc2.7.2.3) for which the ideal configuration file is to be found.
- Write a parameter.txt file which refers to the set of parameters which are to be varied.
- The **optimizer.cpp** file now goes through all possible combinations of parameters in the parameter.txt file and gives the set of configurations which results in an optimal IPC.

Benchmarks:

Shows the comparison between the IPC produced when default.cfg is used, the best possible IPC, and the worst possible IPC (to show the worst case scenario if parameters in config.cfg had been chosen incorrectly) from the set of input parameters.

Program	Default IPC	Best IPC	Worst IPC
Matrix Multiplication	2.0848	2.1711	0.9015
Testcase	0.9078	2.8522	0.5558
FFT	1.3793	1.6319	0.9668
BFS	1.6889	1.8170	0.6485

Practical Uses:

- Improvement in IPC results in decrease in no. of Clock cycles required for a fixed number of instructions or in other words, the time required to run a code in reduced without doing any changes in the code, by just changing the configuration on which the code is run.
- **Neural Networks:**
    - Most of the computation time spent in Neural Networks is spent on Matrix Multiplication.
    - In neural network we know beforehand the dimensions of Matrix Multiplication.
    - So, we can create a C code for matrix multiplication with the required dimenions and find the optimal configuration which minimizes the time for multiplication and then use it ot run the actual Neural Network code.
- **Branch Predictor evaluation:**
    - We can test the performance of a newly developed branch predictor with respect to already existing ones.
    - All possible combination of the new predictor can be tested and we can find the optimal set of configuration which best correlates to the predictor to improve IPC.
- **Cache Replacement Policy:** Same concept as Branch Predictor evaluation.

Important Points

- Size of cache/TLB is always kept constant i.e. whatever is initially present in default.cfg. This is to ensure valid comparison.
- To change size of cache change values in default.cfg.
- Block sizes greater than 64 might not be supported in this version of SimpleScalar.
- The total simulation/computation time is = code running time $\times$ (product of no. of values each parameter can take).
- Hence, with each increase in additional parameter value, the search space and time required for search goes up exponentially.

Conclusion:

We can use this project to reduce a lot of manual work by testing out different possible sets of parameters. It can be used as a pair-up with another project which focuses on building another Branch-predictor or Cache-replacement-policy to test the optimalitiy for those policies with various combinations. This project pushes the limits for obtaining approximately the maximum achievable IPC for a given C code.