Project CS321/CS322

Simulating X86 on Gem5

Aim:- To use Gem5 to simulate an X86 processor in System Call Emulation Mode(SE) and conduct the following 3 experiments:-

- Observe how IPC(Instructions per Cycle), No. of CPU cycles, No. of L1 cache miss and L1i cache miss latency change as we vary the size of L1 cache keeping everything else constant.
- Observe how IPC and No. of CPU cycles change as we vary the number of physical integer registers keeping everything else constant.
- Observe how No. of CPU cycles, No. of L1 Miss and AVG. Miss latency of L1i cache changes as we vary L1 cache associativity keeping everything else constant.

Various Benchmarks Simulated:-

- SPEC CPU2000 (Bzip2)
- MiBench (BitCount)
- SciMark2.0 (Monte Carlo, LU, SOR, SparseMat_Mul)
- Splash2 (FFT)

Process:-

- **Step1:-** Set up the Gem5 Simulator on your system
  - Setup scons
    - Download from https://scons.org/pages/download.html
    - Execute “cd scons-3.1.0” in your terminal
    - Execute “python setup.py install”
  - Install zlib
    - Execute “apt-get install zlib1g-dev”
  - Install Boost, issues faced on some machine
    - Execute “apt-get install libboost-all-dev”
  - Installing m4 Macro Processor on Ubuntu Linux
    - Execute “wget ftp://ftp.gnu.org/gnu/m4/m4-latest.tar.gz”
    - Execute “tar -xvzf m4-latest.tar.gz”
    - Execute “cd m4-1.4.17”
    - Run “./configure”
    - Execute “make”
    - Execute “make install”
  - Execute “git clone https://github.com/gem5/gem5.git”
  - Execute “cd gem5”
  - Run “scons build/X86/gem5.opt”

- **Step 2:- Make scripts to run your benchmarks. These Scripts are made in python and will help in executing the benchmark on your simulated Processor. For Reference look at the examples given in Gem5 folder. You can even modify the given scripts to make them suitable for your use case. In this project, one of the scripts used is a modified version of SE.py already present in examples of Gem5 and another script used is made with the help of various online resources:-**
  - Modified SE.py:- https://github.com/VatsalSin/Gem5_project/blob/master/configs/example/se.py
  - Script.py:- https://github.com/VatsalSin/Gem5_project/blob/master/configs/src/script.py

- **Step 3:- Collect Appropriate benchmarks:-**
  - To use the benchmarks used in the project visit:-
    - https://github.com/VatsalSin/Gem5_project/tree/master/benchmark
  - You can also find other useful benchmarks by visiting:-
    - http://euler.slu.edu/~fritts/mediabench/
    - http://groups.csail.mit.edu/cag/streamit/shtml/benchmarks.shtml
    - http://axbench.org/
    - https://asc.llnl.gov/CORAL-benchmarks/
    - http://math.nist.gov/scimark2/

- **STEP 4:- Execute the benchmark on your simulated processor**
  - For running the program we can use various commands. We have to pass the configurations as parameters to the program. Some of the examples of commands used by me are as follows:-
    - Running SciMark2.0 Benchmark using script.py. The number of Physical Integer Register can be varied in the script.py file.

```
(base) useless2020@useless2020-Aspire-E5-572G:~/Documents/Development$ build/X86
/gem5.opt configs/src/script.py --caches --cmd=/home/useless2020/Documents/Devel
opment/gem5/benchmark/scimark/scimark2 -o '0' --directory=64
```

    - Running Splash2 Benchmark with various L1 cache sizes.

```
(base) useless2020@useless2020-Aspire-E5-572G:~/Documents/Development$ build/X86
/gem5.opt configs/example/se.py --caches --l1i_size=1024kB --l1d_size=1024kB --n
um-l2caches=1 --cpu-type=TimingSimpleCPU  -c /home/useless2020/Documents/Develop
ment/gem5/benchmark/splash2/codes/kernels/fft/FFT -o "-t -p1"
```

Observations:-

- **Experiment 1:-**

MiBench (BitCount):-

L1 Cache Size	Total CPU Cycles	Avg. IPC	Total L1 Miss	Overall L1i miss latency
16KB	1571912939	0.37289826138	1705	58948000
32KB	1571901515	0.37290099373	1232	53562500
128KB	1571893919	0.37290277347	918	49017500
1024KB (1MB)	1571893395	0.37290289778	899	48975500

Splash2 (FFT):-

L1 Cache Size	Total CPU Cycles	Avg. IPC	Total L1 Miss	Overall L1i miss latency
16KB	6283776	0.2203504708	13993	139968000
32KB	5743338	0.24109098228	9227	115811500
128KB	5340448	0.25930315209	5786	97858000
1024KB (1MB)	5311612	0.26071162577	5587	96216000

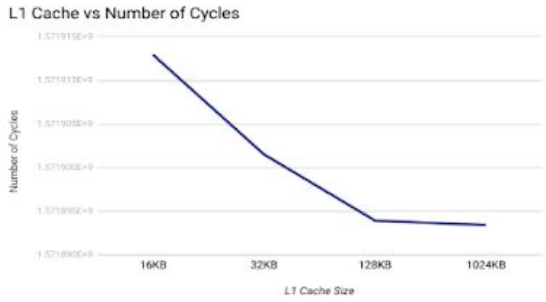
SciMark2.0 (Monte Carlo, LU, SOR, etc):-

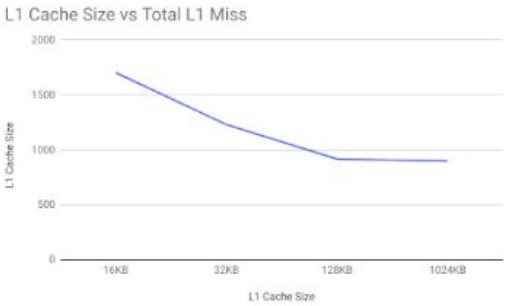
L1 Cache Size	Total CPU Cycles	Avg. IPC	Total L1 Miss	Overall L1i miss latency
16KB	63722293	0.2527567393	73777	131159500
32KB	63433867	0.25390662372	61694	124963500
128KB	62140827	0.2591882789	7879	120792000
1024KB (1MB)	62118713	0.25928083861	6838	121033500

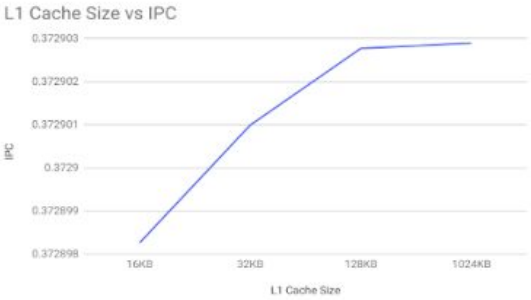
SPEC CPU2000 (bzip2):-

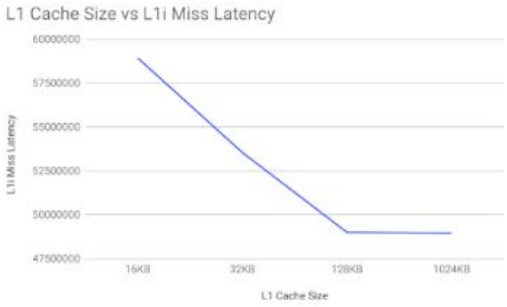
L1 Cache Size	Total CPU Cycles	Avg. IPC	Total L1 Miss	Overall L1i miss latency
16KB	2092784367	0.23891615776	3388867	85420000
32KB	2081007563	0.24026822818	2891178	82462000
128KB	2060465019	0.24266366834	2045337	79996500
1024KB (1MB)	2024103231	0.24702297409	647996	79875500

MiBench (BitCount):-

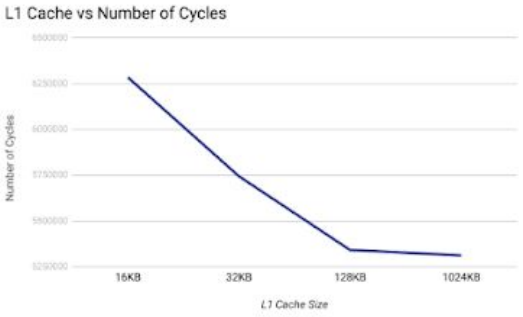


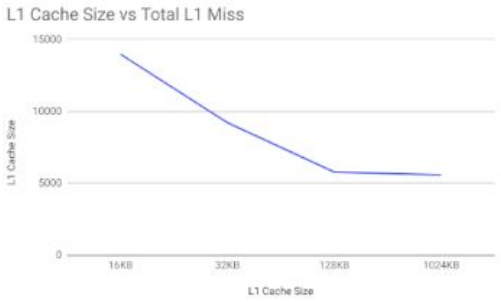


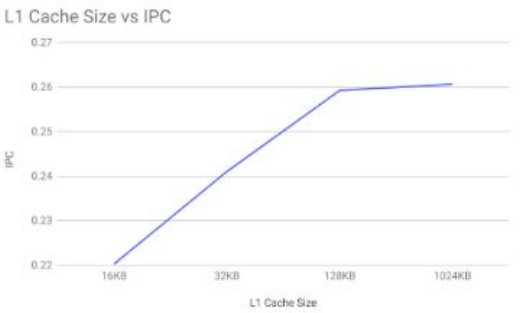


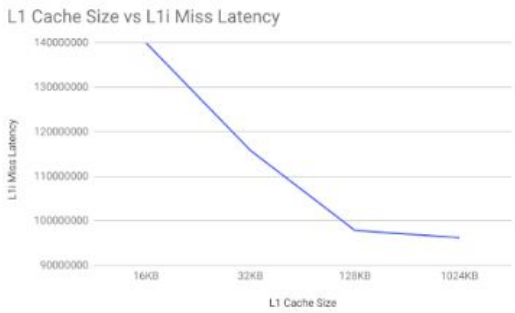


Splash2 (FFT):-

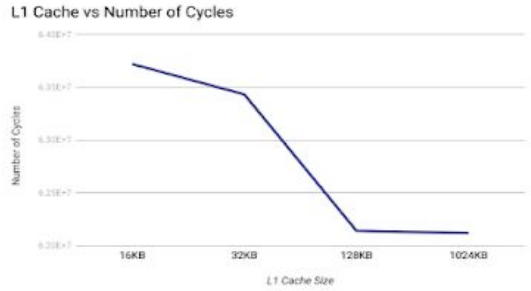


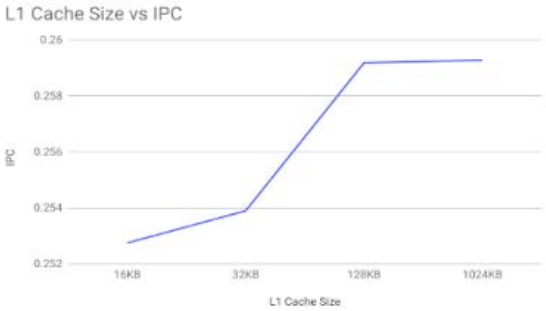


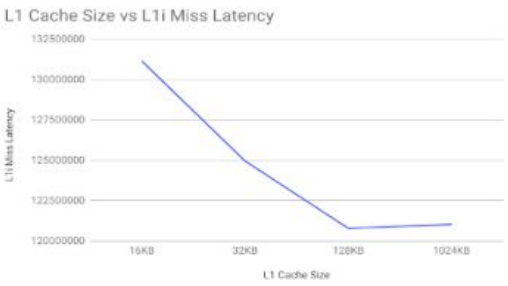




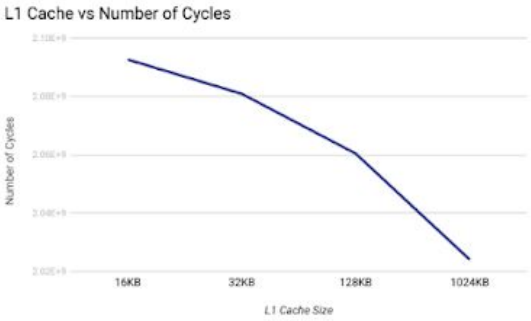
SciMark2.0 (Monte Carlo, LU, SOR, etc):-

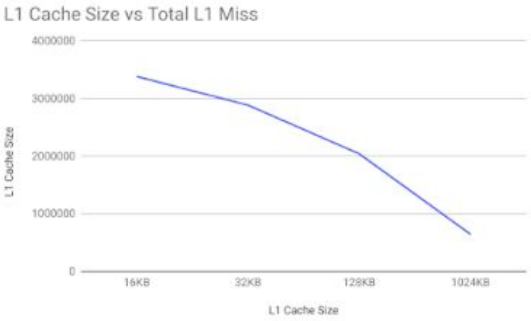


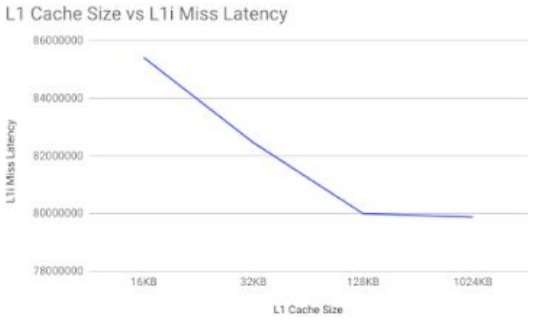




SPEC CPU2000 (bzip2):-







General Observations:-

Parameter Under Observation	Observed Change (with increase in L1 cache size)	Possible Reasons
Total CPU Cycles	Decrease	As size of L1 cache increases, it is more probable that data will be found in L1 cache and since extra cycles need not be spent accessing L2 Cache or main memory the number of total execution cycles decreases.
Average IPC	Increase	Since more memory operations can be completed through L1 cache, on average it takes less cycles to complete a single instruction.
L1(inst. + data) Miss Rate	Decrease	Since more data can be stored in the L1 cache, it is more likely that the data requested will already be present in the cache, hence chances of miss decreases
L1i miss latency (number of overall miss cycles)	Decrease	As size of L1 cache increases, number of miss decreases but cycles to search in cache increases. In general the increase in cycles is less than decrease in miss.

- **Experiment 2:-**

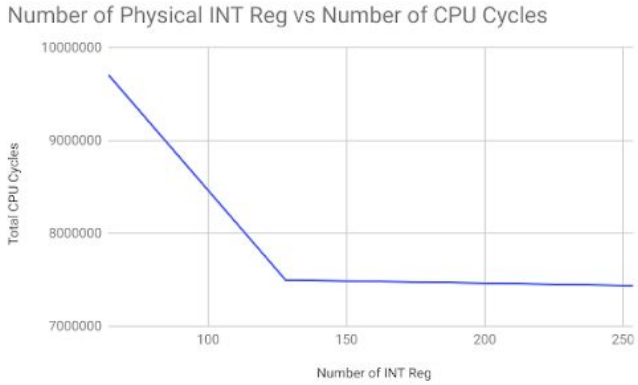
SciMark (Monte Carlo, LU, SOR, etc):-

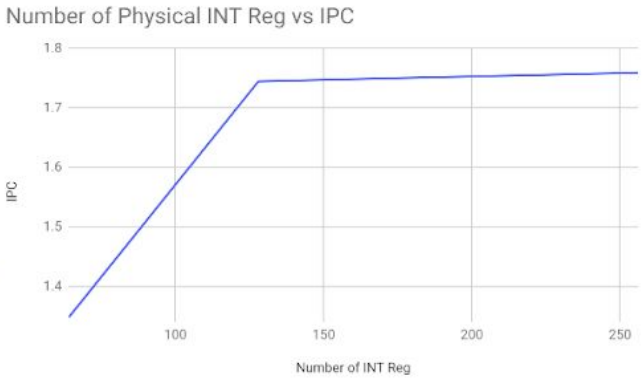
Number of Physical Int Registers	Total CPU Cycles	Avg. IPC
64	9710254	1.347429
128	7500782	1.744339
256	7438230	1.759003

Splash2 (FFT):-

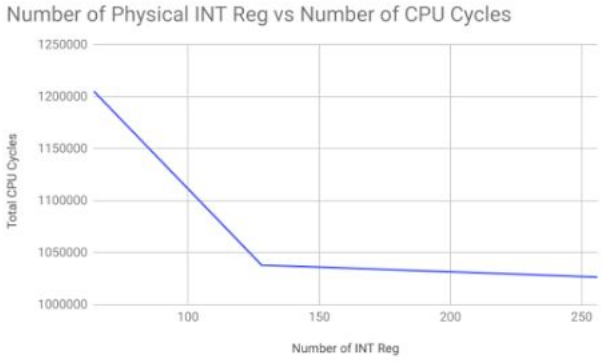
Number of Physical Int Registers	Total CPU Cycles	Avg. IPC
64	1205510	0.791711
128	1038216	0.919331
256	1026640	0.929697

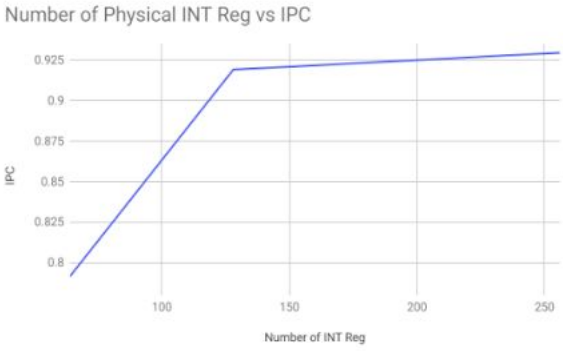
SciMark (Monte Carlo, LU, SOR, etc):-





Splash2 (FFT):-





General Observations:-

Parameter Under Observation	Observed Change (with increase in number of physical Registers)	Possible Reasons
Total CPU Cycles	Decrease	As the number of integer registers increases, the number of writes to cache, and writes to main memory to free register space is lower. Hence there are lower number of execution cycles.
Average IPC	Increase	As the number of integer registers increases, more data can be maintained in registers causing less long writes and reads to cache and main memory. This increases the IPC.

Interesting Observation:-

As we increase the number of Physical Integer Register, the effect of increase diminishes. This is mainly due to the process of register renaming. Hence increasing the number of registers beyond a particular number doesn't give any significant advantage.

- **Experiments 3:-**

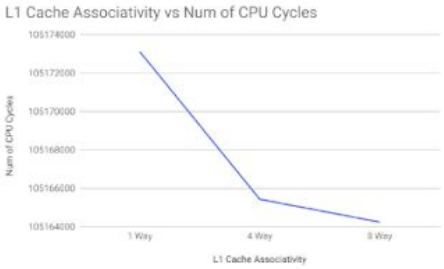
MiBench (BitCount):-

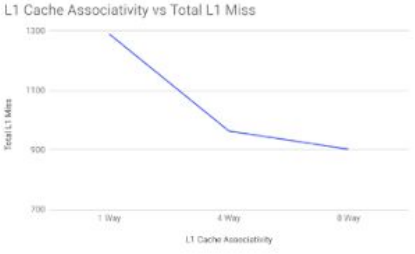
Associativity of Cache	Total CPU Cycles	Total L1 Miss	Average Overall L1i miss latency (Per Miss)
1 Way	105173137	1290	53467.124632
4 Way	105165451	964	72038.239538
8 Way	105164259	903	77915.348101

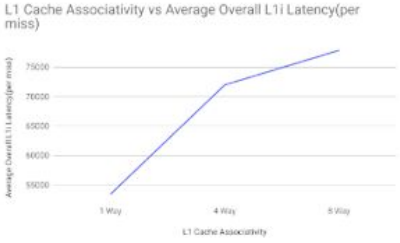
Splash2 (FFT):-

L1 Associativity of Cache	Total CPU Cycles	Total L1 Miss	Average Overall L1i miss latency (Per Miss)
1 Way	4151259	7492	61206.954436
4 Way	4120601	6292	75264.017252
8 Way	4116413	6112	77258.349086

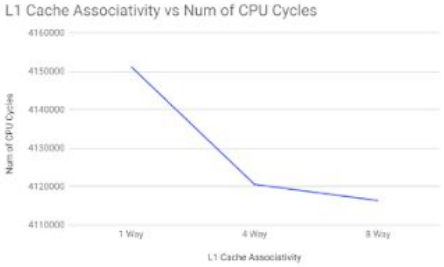
MiBench (BitCount):-

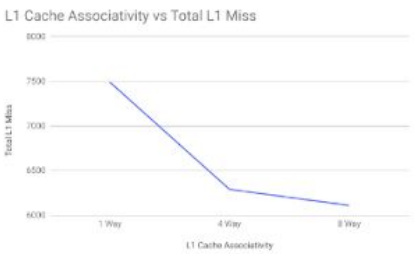


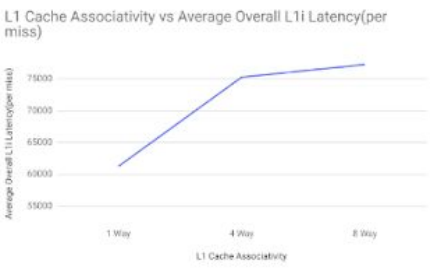




Splash2 (FFT):-

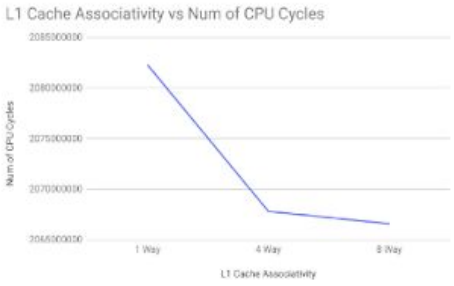


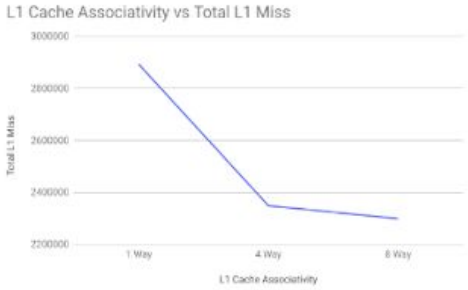


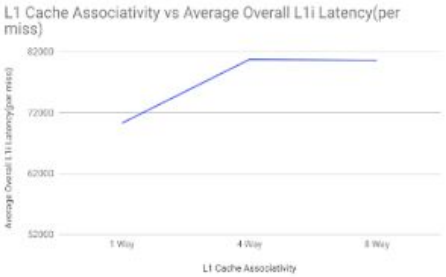


SPEC CPU2000 (bzip2):-

Associativity of Cache	Total CPU Cycles	Total L1 Miss	Overall L1i miss latency (Per Miss)
1 Way	2082291049	2893644	70336.683417
4 Way	2067816051	2349446	80786.422200
8 Way	2066593677	2299558	80648.851149







General Observations:-

Parameter Under Observation	Observed Change (with increase in L1 cache Associativity)	Possible Reasons
Total CPU Cycles	Decrease	As associativity of L1 cache increases, number of conflicts decreases and the probability that the data will be found in L1 cache also increases and since extra cycles need not be spent accessing L2 Cache or main memory the number of total execution cycles decreases.
L1(inst. + data) Miss Rate	Decrease	As associativity of L1 cache increases, number of conflicts decreases and the probability that the data will be found in L1 cache also increases, hence chances of miss decreases.
Average Overall L1i miss latency (Cycle/Miss)	Increase	set associative caches are usually slower and somewhat more expensive to build because of the output multiplexer and additional comparators which adds some overhead cycles.

Observation on Benchmarks(workloads):-

- The average number of cycles required to run a SPEC CPU2000 and MiBech are significantly large as compared to Splash2.0 and SciMark2.0. This shows that MiBench and SPEC CPU2000 are heavier to process.
- The L1 Cache miss shows a drastic decrease when the cache size was increased in case of SPEC CPU2000 and SciMark2.0. This shows that these two benchmarks were more dependent on L1 cache size.
- SciMark2.0 shows a more drastic decrease in the number of CPU cycles as compared to Splash2.0 benchmark. This shows that SciMark2.0 is more sensitive to the number of Physical Integer Registers.

Overall Conclusion:-

The Gem5 simulator is a modular platform for computer system architecture research. In the above experiments, we have built an X86 system architecture and simulated its performance after making changes to various of its key parameters. This helps us to study how the CPU performance is affected by these key parameters and also helps us to find the optimal value of these parameters, which in turn will help us to design highly optimised CPU architectures.

***All the results(Stats.txt files) can be found at**
https://github.com/VatsalSin/Gem5_project/tree/master/Results

References:-

- **http://gem5.org**
- **https://developer.arm.com/solutions/research/research-enablement-kits**
- **https://research.cs.wisc.edu/multifacet/papers/can11_gem5.pdf**
- **https://ieeexplore.ieee.org/document/8391354**
- **https://github.com/estwings57/CasHMC/issues/1**