ECE 511 Assignment 1
Understanding Gem5

Vikram Sharma Mailthody

(NetID- vsm2, UIN - 663250535)

Contents

Goals of the Assignment:

1. Learn how to use gem5 and understand its various statistics.
2. Understand how to use se.py and how to make modifications for caches.
3. Understand how the system behaves when different levels of caches are modified.

More details of the assignment can be found here.

Software used:

Gem5 and its dependencies

Tasks:

The assignment requires to submit 3 results (baseline, intermediate and final) and its comparisons. Following are the main steps involved for completing the assignment.

1. Use se.py as the default script for running the system emulation
2. Section 2 - run se.py with dual core processors and with default L1,L2 cache sizes.
3. Make modifications as presented in tutorial for common/Caches.py common/CacheConfig.py files.
4. Run se.py with dual core processors and modified Caches. Run benchmarks fft and correlation medium. This is your baseline benchmarks. (1st submission)
5. Section 3 – make modifications to incorporate L3 shared cache for dual core system as given in the problem statement.
6. Run se.py with dual core processors and L3 Caches. Run benchmarks fft and correlation medium. This is your second benchmarks. (2nd submission or intermediate)
7. Create a system with 8 cores, L2 shared across 4 cores and L3 shared across 8 cores. Change L2 = 512kB and L3 = 2MB
8. Run se.py with 8 core processors and L3 Caches. Run benchmarks fft and correlation medium. This is your final benchmarks. (3rd submission - final)
9. Analyze the data obtained in baseline, intermediate and final reports.

Introduction:

Gem5 is a computer system simulator platform initially developed at University of Wisconsin-Madison Square (there are several other collaborators like MIT, UMich, ARM, etc). It's a modular simulator allowing the user to parameterize, extend and rearrange as required. The simulator supports both x86 and ARM ISA allowing the user the flexibility of choice. The simulator is primarily developed using C++ language and scripts to run are typically written in python. There are two main modes of operation when using the simulator – system call emulation mode and full system mode (se and fs respectively). The System call emulation mode is mainly used to simulate a binary file that are linked statically or dynamically. The full system mode runs complete system with choice of operating system to boot in the simulation environment. Further gem5 supports configurable CPU models, pluggable memory systems and device models providing flexibility, availably and enhancing collaboration for the computer system researcher.[1]

For this assignment, we will be using ARM ISA and System Call emulation mode. The primarily goal of the assignment is to understand how SE mode functions and how a user can modify the gem5 scripts to build and run benchmark binaries.

Report organization:

Next section describes a high level information on gem5 flow. This section is followed by three sections describing on baseline, intermediate and final architecture of the assignment. Goals, Modifications, commands and architecture results are discussed in each of these sections. The final section discusses the benchmarking results of FFT and Correlation_medium. This is followed by bottlenecks that individual faced during the assignment and references used.

Gem5 Flow:

Figure 1 shows the high level flow of how Gem5 runs the simulation [2]. The example python script (here in our case it is se.py) instantiates simulation objects based on parameters provided. In the next stage, python interpreter compiles it to gem5 readable format based on libraries of simulation of objects. Gem5 then runs the compiled binary in C++ simulations and run the binary provided in the command line.

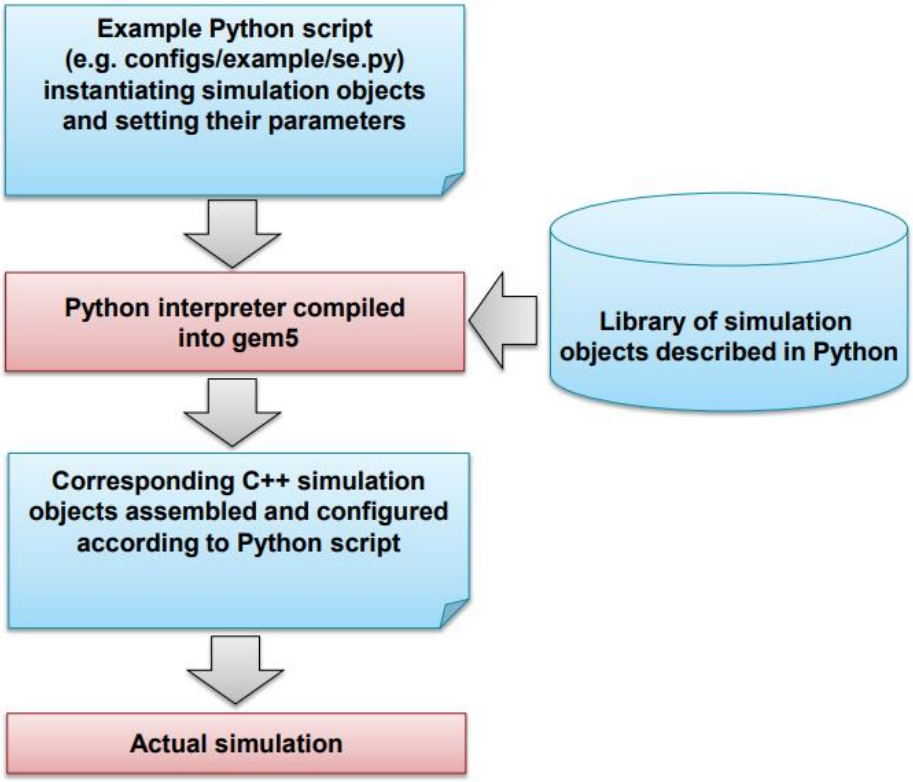

Figure 1 – High level flow of gem5

Baseline architecture:

In order to measure the default system performance while using se.py, baseline results are calculated. The high level architecture of baseline model is shown in figure 2:

- Develop dual core ARM ISA based system. Connect the system to membus interface and DDR3 memory.
- Create 2 Level private caches across individual CPU with default values

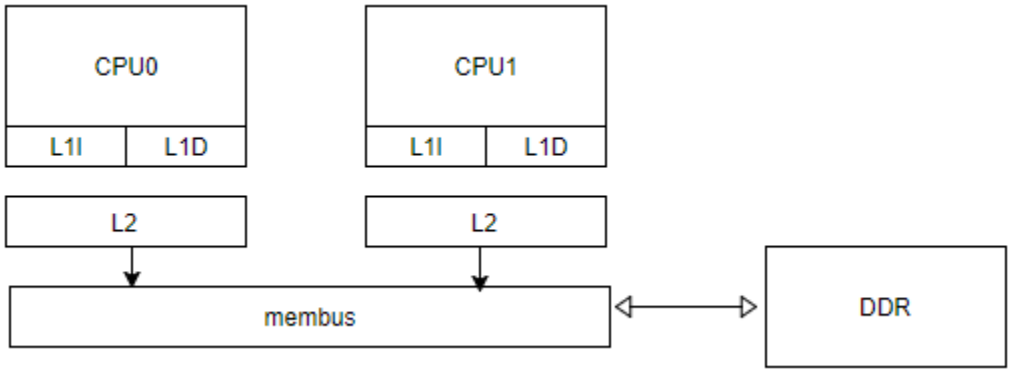

Figure 2: High level baseline architecture

Two benchmarks named FFT and Correlation medium are run to measure the performance of the system. FFT primarily does a matrix computation on both the cores assisting in finding the compute performance. The correlation medium benchmark runs correlation equation over the compute blocks. Both the benchmarks are well suited to measure memory and compute performance.

In the runs we will be using TimingSimpleCPU model for CPU. This model using timing path for memory access. The CPU waits until the memory data are returned. Two random delays – fetch delay and LD/ST delays are modeled inside the TImingSimpleCPU to provide some level of timing as real environment. A high level view of TimingSimpleCPU model is presented in Figure 3. [2]

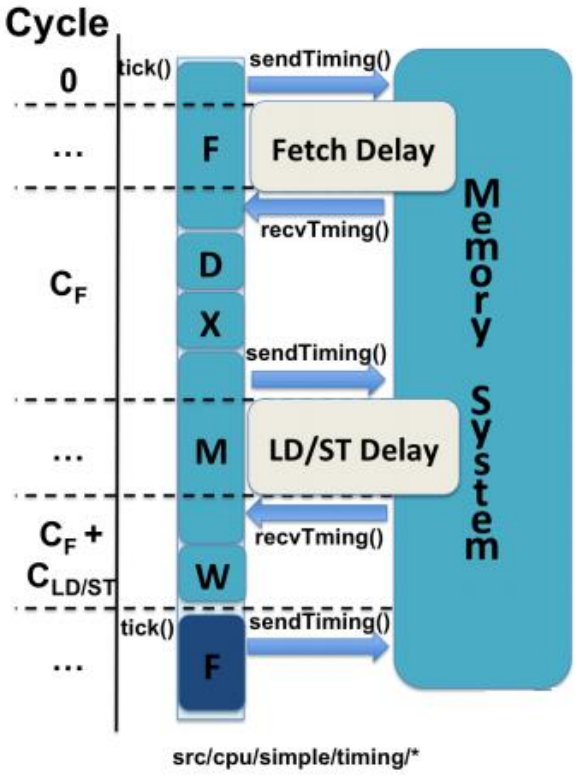

Figure 3 : TimingSimpleCPU High level model as described in [2]

Modifications in default repository:

1. Default caches sizes are modified as per [3] tutorial.

Commands:

1. FFT Benchmark :
```
build/ARM/gem5.fast --outdir=a1core2modcache_fft configs/example/se.py -n 2 --cpu-type=TimingSimpleCPU --num-cluster=1 --l2caches -c fft -o '-m16 -p2'
```
2. Correlation_medium Benchmark:
```
build/ARM/gem5.fast --outdir=a1core2modcache_correlation_medium configs/example/se.py -n 2 --cpu-type=TimingSimpleCPU --num-cluster=1 --l2caches -c correlation_medium
```

Results:

The high level architecture from the gem5 is shown in figure 4.

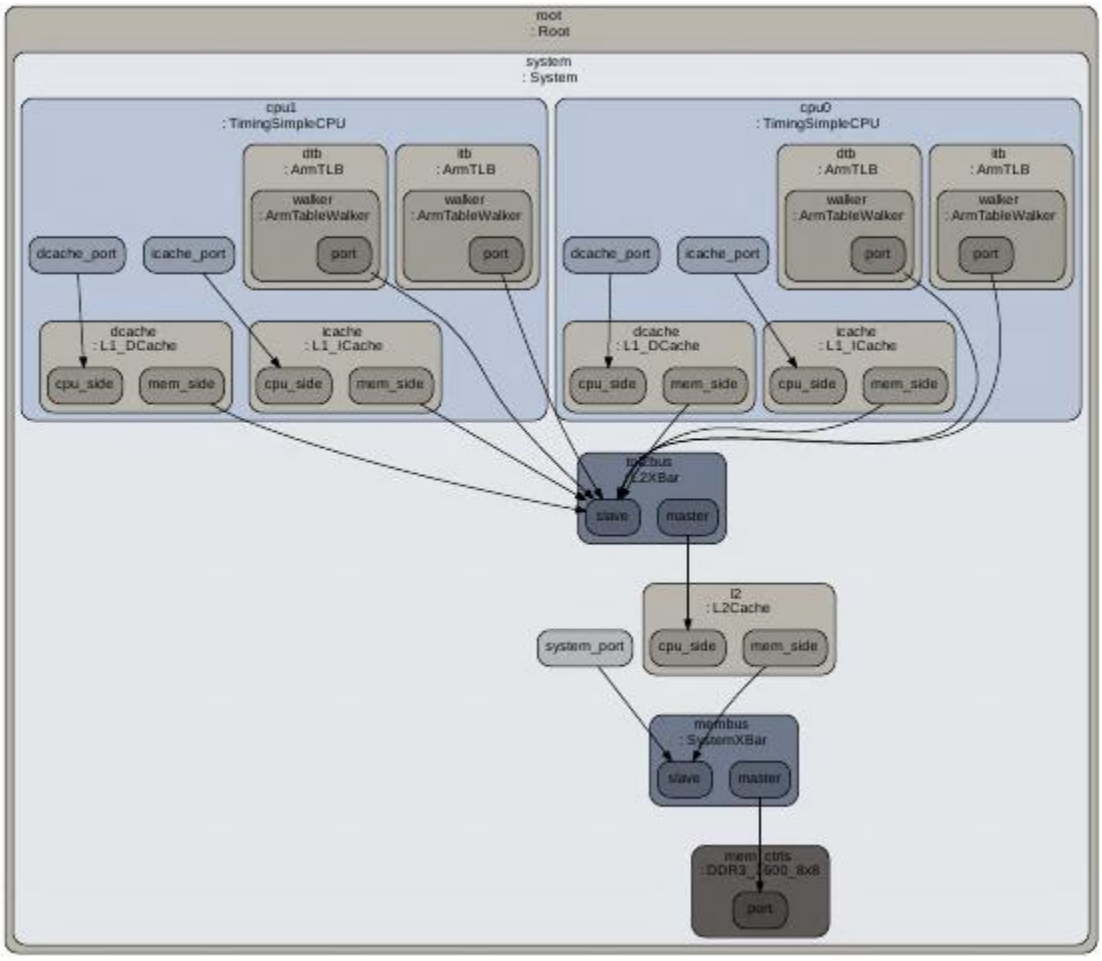

Figure 4: config.dot.pdf from gem5 results showing the complete architecture.

Intermediate Architecture

Intermediate architecture requirement is to have L3 cache enabled over dual core private L2 caches with default sizes. The block diagram shown in figure 4 summarizes the architecture.

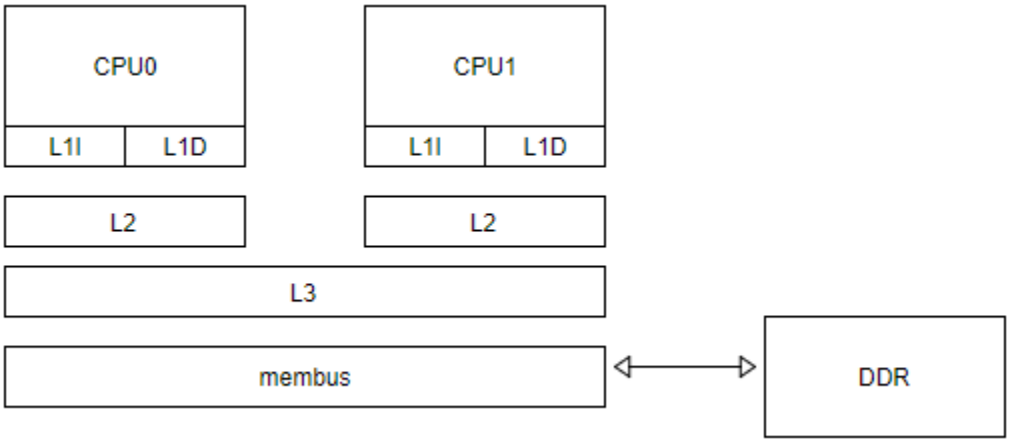

Figure 4. Intermediate architecture having dual core with L1,L2 and L3 caches.

Modifications in default repository:

1. L3 cache is added in Caches.py as follow:

```
#tag and data latency are required. Keeping same as L2-20. Although
tag and data latency can be removed, however I have specifically
added the default values for consistency across the code. Further
please note that write_buffer default values were 8, however they
have been modified to 256 to handled shared memory access. 8 is
too less to handle at the L3 level when 8 CPU starts requests for
write.

class L3Cache(Cache):
      assoc = 16
      tag_latency = 20
      data_latency = 20
      response_latency = 20
      mshrs = 512
      tgts_per_mshr = 20
      write_buffers = 256
```

2. The Options.py and se.py files are modified to handle two additional parameters – --Num-clusters and –l3caches. The –num-cluster option can be used to create CPU clusters or L2 division. For example if L2 needs to be shared across 2 CPUs in 4 core system, then value of 2 needs to be specified to in the command line. Specifying 1 will result in private L2 for all 4 cores. Similarly L3 cache addition is option and primarily used for adding L3 cache. The detailed flow chart of this will be discussed in final result.

Commands:

1. FFT Benchmark :

```
   build/ARM/gem5.fast          --outdir=a1f8core2L512cacheL32MB_fft
   configs/example/se.py   -n   2   --cpu-type=TimingSimpleCPU   --
   num_cluster=2    --l2cache    --l3cache         --caches    -c
   /home/vsm2/Downloads/fft -o '-m16 -p2'
2. Correlation_medium Benchmark:
   build/ARM/gem5.fast                                          --
   outdir=a1f8core2L512cacheL32MB_correlation_medium
   configs/example/se.py   -n   2   --cpu-type=TimingSimpleCPU   --
   num_cluster=2    --l2cache    --l3cache         --caches    -c
   /home/vsm2/Downloads/correlation_medium
```

Results:

The high level architecture from gem5 simulation is shown in figure 5

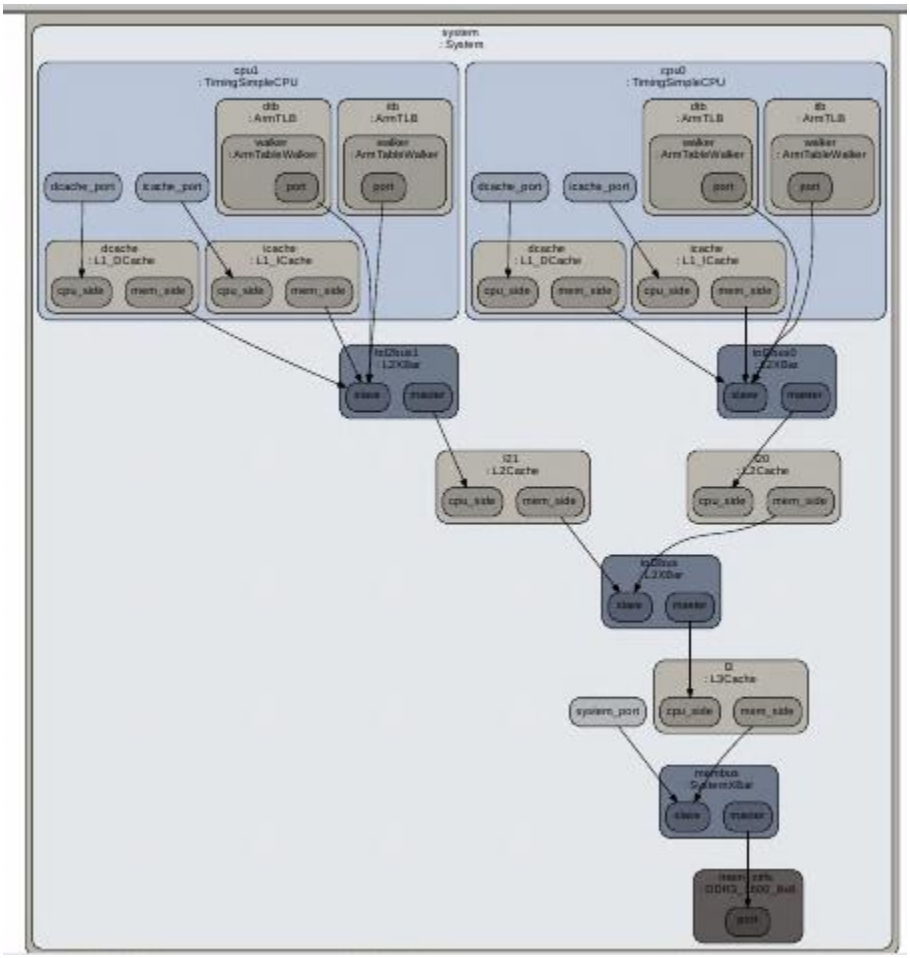

Figure 5. L3 cache addition as intermediate architecture

Final Architecture

Final architecture requirement is to have L3 cache enabled with 2MB size over 8 core shared L2 caches with 512kB size. The block diagram shown in figure 6 summarizes the architecture.

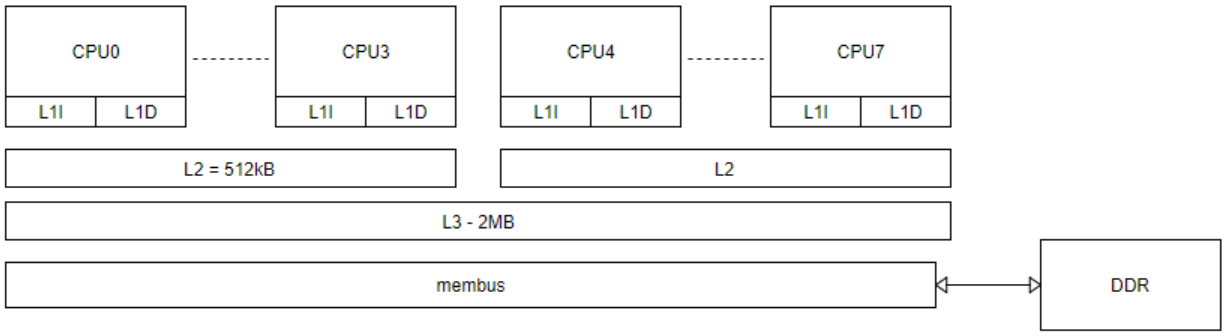

Figure 6. Intermediate architecture having dual core with L1,L2 and L3 caches.

Modifications in default repository:

Modifications are done over intermediate architecture. In this assignment, L2 is shared cache memory across 4 cores. L3 caches are of 2MB in sizes. Figure 7 shows the high level modifications done on CacheConfig.py. The modifications are such that it can handle intermediate/baseline run depending on what is provided in the command line.

Commands:

FFT Benchmark :

1. build/ARM/gem5.fast --outdir=a1f8core2L512cacheL32MB_fft_1core configs/example/se.py -n 8 --cpu-type=TimingSimpleCPU --num_cluster=2 --l2cache --l2_size=512kB --l3cache --l3_size=2MB --caches -c /home/vsm2/Downloads/fft -o '-m16 -p1'
2. build/ARM/gem5.fast --outdir=a1f8core2L512cacheL32MB_fft_2core configs/example/se.py -n 8 --cpu-type=TimingSimpleCPU --num_cluster=2 --l2cache --l2_size=512kB --l3cache --l3_size=2MB --caches -c /home/vsm2/Downloads/fft -o '-m16 -p2'
3. build/ARM/gem5.fast --outdir=a1f8core2L512cacheL32MB_fft_4core configs/example/se.py -n 8 --cpu-type=TimingSimpleCPU --num_cluster=2 --l2cache --l2_size=512kB --l3cache --l3_size=2MB --caches -c /home/vsm2/Downloads/fft -o '-m16 -p4'
4. build/ARM/gem5.fast --outdir=a1f8core2L512cacheL32MB_fft_8core configs/example/se.py -n 8 --cpu-type=TimingSimpleCPU --num_cluster=2 --l2cache --l2_size=512kB --l3cache --l3_size=2MB --caches -c /home/vsm2/Downloads/fft -o '-m16 -p8'

Correlation_medium Benchmark:

5. build/ARM/gem5.fast --outdir=a1f8core2L512cacheL32MB_correlation_medium configs/example/se.py -n 8 --cpu-type=TimingSimpleCPU --num_cluster=2 --l2cache --l2_size=512kB --l3cache --l3_size=2MB --caches -c /home/vsm2/Downloads/correlation_medium

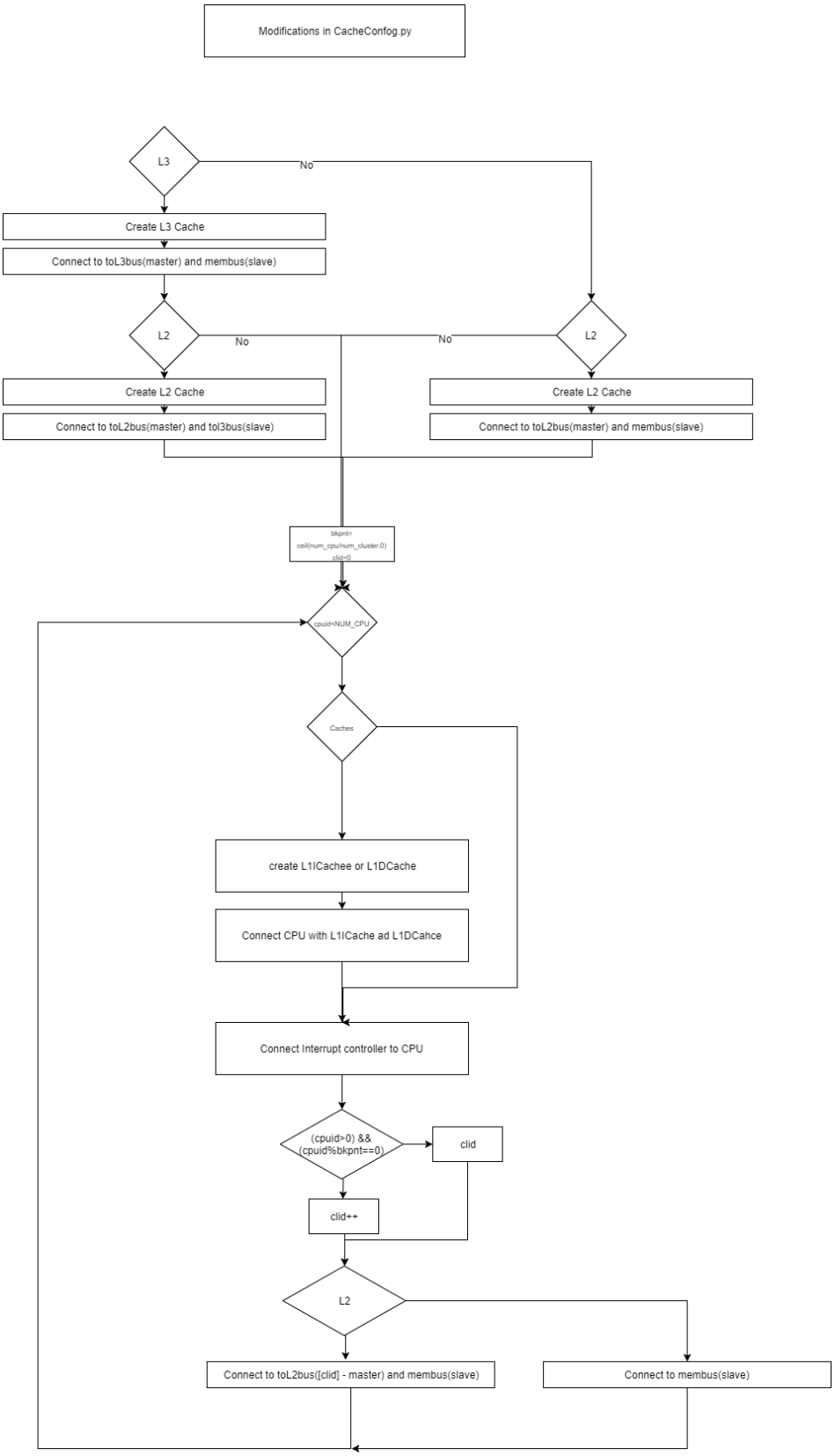

Figure 7: modifications in CacheConfig.py

Results:

The high level architecture from gem5 simulation is shown in figure 8

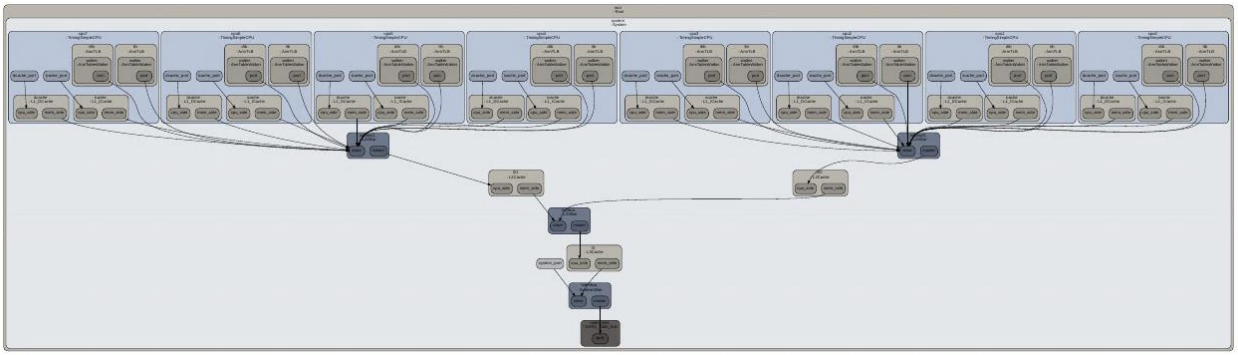

Figure 8 : Final architecture of gem5 assignment

Analysis of Results:

The results of various runs are averaged to get table 1.

Table 1: Results of various run

		Baseline		Intermidiate		Final 2	Final 4	Final 8		SpeedxFFT_8core	SpeedxFFT_4core
		FFT	CM	FFT	CM	FFT	FFT	FFT	CM		
sim_seconds		0.526624	1.774765	0.525735	1.789733	0.529262	0.424883	0.367026	1.775601	1.434841128	1.239456509
Host Inst Rate		1830936	1803064	1756827	1809624	1745912	1734034	1350870	1691837	1.355375425	1.055882411
IPC (avg)		0.587989	0.419624	0.589056	0.416115	0.585131	0.728843	0.599478	0.419427	1.019538987	1.23955249
Cache Miss (av	L1D	108836	1332710	74787	1339423	119862	76223	33702	1332873	3.229363243	1.427862981
	L1I	67362	392	67357	392	67362	67362	66964	392	1.005943492	1
	L2	108939	15602	65307	708941	182073	179553	34396	39709	3.167199674	0.606723363
	L3	NA	NA	51208	15602	112308	110336	34364	15602		
Cache Hit (avg)	L1D	22594924	2.63E+08	28737061	2.63E+08	28727943	53365847	57044018	2.63E+08	2.524638631	2.36185114
	L1I	50217	1.49E+09	4.45E+08	1.49E+09	1.75E+08	87399980	2.99E+08	1.49E+09	5947.29038	1740.446064
	L2	142087	1317500	31117	624161	69019	70552	66566	1293393	0.468487617	0.496540852
	L3	NA	NA	12267	693339	69755	69063	172	24107		
Cache WB (avg	L1D	92796	1331059	93115	662703	74092	53125	32177	1309098	0.34674986	0.572492349
	L1I	66851	52	66846	52	66851	66851	66458	52	0.994121255	1
	L2	66793	-	93115	704290	114714	137134	25224	31090	0.377644364	2.053119339
	L3	NA	NA	21116	704290	48574	47273	1531	31090		
ReadBW MiBps		13.18	0.56	6.23	0.56	13.54	16.6	5.99	0.56		
WriteBW MiBps		5.61	0	0	0	5.87	7.12	0.26	0		

Following points can be derived from the table:

1. FFT is a multithread-able algorithm. Improvements is clearly visible when number of cores are changed. However Correlation Medium (CM) is a single threaded algorithm. No significant improvement is observed in terms of simulation time. Further, CM is primarily integer based compute problem where as FFT has lot of integer and FP operations.
2. Simulation time is 1.43x faster in final run when compared to baseline. This is mainly because the number of compute elements (cores) available to run parallel is more in final architecture. Further, with 2MB of cache memory, amount of data required to read and write is drastically reduced. This is clearly visible looking at the Read/Write average bandwidth. The bandwidth is shared across multiple instances and different times causing reduction in total requirement.
3. IPC is slightly more compared to baseline. I believe this is mainly because of addition of parallel thread execution handler instructions. Thread handling between two cores consumes additional sets of instructions and hence average IPC is about 1.02x.

4. Its noteworthy to observe that the L1Dcache and L1Icache hits rates are significantly improved. This significant change is primarily because of 2 levels of large shared caches present in the final architecture (L2 and L3). Since hit rates have improved, the misses have similarly decreased. However not significantly. The number of data/instruction misses in L1 cache is directly proportional to amount of data handshakes required between two or more threads. Based on the table I concur that there could have been multiple handshakes between the two threads and shared L2 and L3 were not sufficiently large to reduce this miss rate. This may be primarily because of coherency issues.
5. About 1.35x time improvement is found on host_instr_rate. This can be attributed to different machine runs. Baseline was run in i7 processor with 16GB of RAM whereas Final FFT 8 core was run with 24 core, 128GB RAM Xeon 5th gen core.
6. In the intermediate architecture, addition of L3 shared memory assisted in improving the cache hits and reducing the miss rate. Further, this addition significantly reduced the data bandwidth required between the membus and DDR interface.

Concerns/Additional Learning:

1. Gem5 is very slow is running. Laptops and desktops are not effective way to run these environments! Gem5 parallelization or distributed computing probably a good PhD thesis all together!
2. Final FFT 8 core implementation took 1.5 days run where 4 core it was about 30mins. Further, FFT 8 core implementation crashed twice a server and also a desktop machine before getting this result. The result does seem promising, however, the average bandwidth reduction needs to be analyzed further.
3. The assignment consumed a lot more time than expected. Time was spent on runs and not system crashes.

References:

1. Gem5.org
2. Sascha Bischoff, Andreas Hansson – Session on Gem5.
3. www.learning.gem5.org/book/index.html