# CSE530: Assignment#3 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 CPU models, CPU frequencies, and memory configurations on 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) for python jupyter notebook for automation code and generation of graphs.

## Experiments and Discussions

### 1. Impact on varying CPU model

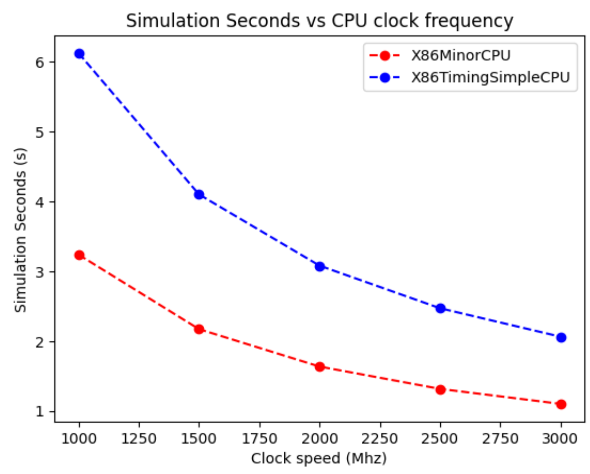
We experiment with two CPU models, namely, X86TimingSimpleCPU and X86MinorCPU, and observe how the benchmark behaves. For this experiment, we compare the metrics by fixing memory configuration to DDR3_2133_8x8 and CPU clock frequency to 2Ghz. We observe from Table 1a, that X86MinorCPU performs better on the metrics compared to X86TimingSimpleCPU. We hypothesize that this is due to the fact the bottleneck of TimingSimpleCPU is memory as the model stalls on memory writes/reads. However, MinorCPU utilizes instruction level parallelism to compensate for this by pipeling.

| **Metric** | **X86TimingSimpleCPU** | **X86MinorCPU** |
|---|---|---|
| Simulation Seconds (s) | 3.081702 | 1.102077 |
| Average system read bandwidth (Byte/s) | 2.47 x 10^6 | 68 x 10^6 |
| Average system write bandwidth (Byte/s) | 2.25 x 10^6 | 63 x 10^6 |
| Average L2 Miss latency (Tick/Count) | 8.19 x 10^4 | 7.4 x 10^4 |

Table 1a: Table showing few selected metrics which show high variation between the two CPU models

### 2. Impact on changing CPU clock frequency

We vary CPU clock frequencies from 1000Mhz to 3000Mhz in steps of 500Mhz on two CPU models from gem5: X86MinorCPU, X86TimingSimpleCPU and analyze the impact on few selected parameters. We notice that increasing clock frequency reduces simulation seconds/execution time. We believe this is because both the CPU models can execute the same instruction at a faster rate.



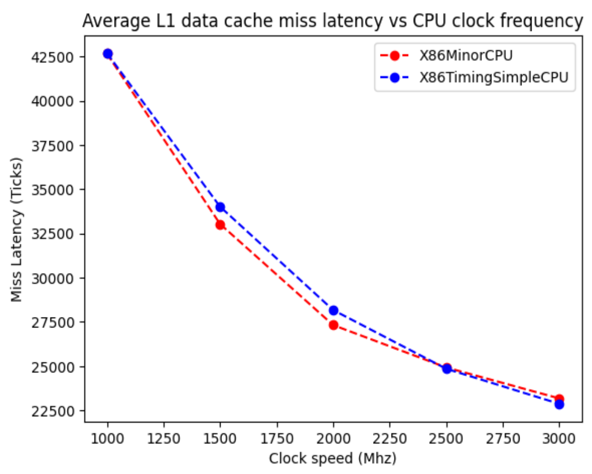
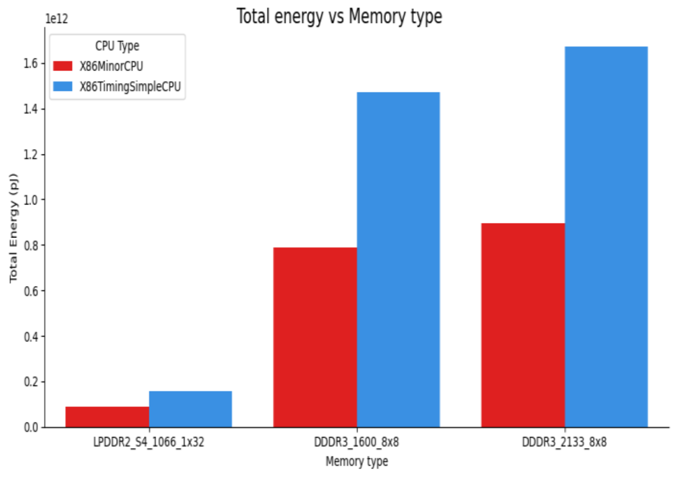


Fig 2a: Line graph on the left showing simulation seconds (s) vs CPU clock frequency (Mhz) for X86MinorCPU and X86TimingSimpleCPU models.

Fig 2b: Line graph on the right showing average L1 data cache miss latency (ticks) vs CPU clock frequency (Mhz) for X86MinorCPU and X86TimingSimpleCPU models.

## 3. Impact on varying memory configuration

We experimented with different memory configurations (DDR3_1600_8x8, DDR3_2133_8x8 and LPDDR2_S4_1066_1x3) on X86MinorCPU, X86TimingSimpleCPU models. Fig 3a, 3b show the impact on energy and power of rank0 in DRAM. We observe that LPDDR2 memory type consumes the lowest power and energy on both the models.



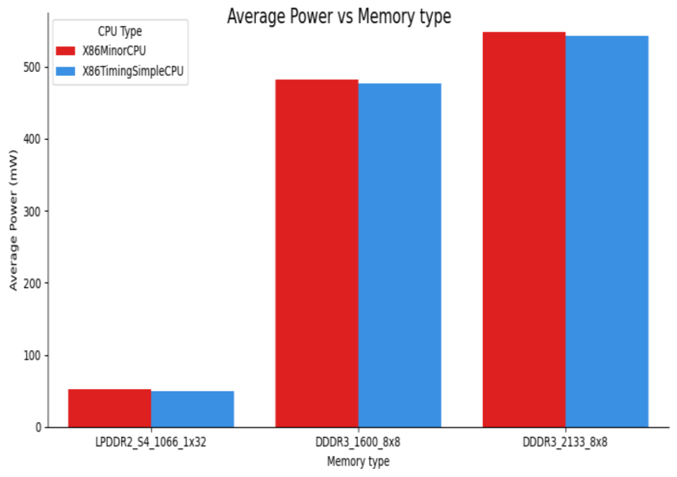
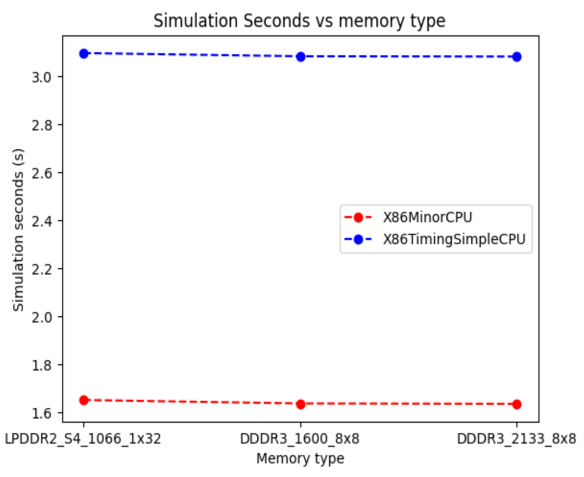


Fig 3a: Bar chart on the left showing total energy of rank0 in DRAM vs various memory type configurations for X86MinorCPU and X86TimingSimpleCPU models.

Fig 3b: Bar chart on the right showing average power of rank0 in DRAM vs various memory type configurations for X86MinorCPU and X86TimingSimpleCPU models.



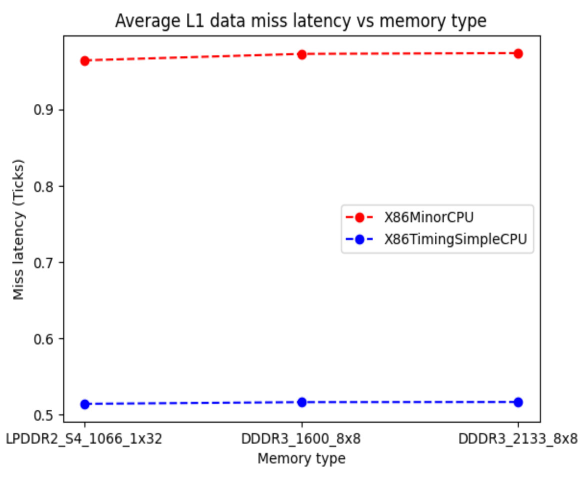


Fig 3c: Graph on the left showing simulation seconds vs various memory type configurations for X86MinorCPU and X86TimingSimpleCPU models.

Fig 3d: Graph on the right showing average l1 data cache miss latency vs various memory type configurations for X86MinorCPU and X86TimingSimpleCPU models.

**Q1) Which CPU model is more sensitive to changing the CPU frequency? Why do you think this is? (Max 200 Words)**

If we consider sensitivity with respect to the average L1 data cache miss latency metric. We observe from Fig 2b that **MinorCPU is more sensitive** compared to TimingSimpleCPU. Note that the slope of the curve is much steeper for MinorCPU. We postulate this is because MinorCPU can take advantage of operator forwarding in memory instructions with increasing clock frequency.

**Q2) Which CPU model is more sensitive to memory technology? Why? (Max 200 Words)**

For this question, we consider sensitivity with respect to total energy metric. We observe from Fig 3a that **Timing Simple CPU is more sensitive** to memory type. We hypothesize that this trend is due to timing simple CPU's bottleneck being on the memory read/write instructions. However, MinorCPU can exploit instruction level parallelism to compensate for this bottleneck through pipelining.

**Q3) Is the application more sensitive to the CPU frequency or the memory technology? Why? (Max 200 Words)**

The application is **more sensitive to CPU frequency** because the quicksort benchmark is CPU intensive. After the cache reaches the warmup state, most of the accesses are hits; and the miss rate of l2 cache is very low. Thus, improving memory technology doesn't lead to significant improvement when compared to increasing CPU clock frequency.

**Q4) If you were to use a different application, do you think your conclusions would change? Why? (Max 200 Words)**

Yes, if we choose a different benchmark, we might get divergent conclusions compared to the ones above. Suppose we choose a memory intensive benchmark, say, big data processing benchmark, then the application will be more sensitive to the memory technology compared to CPU frequency. This is because the CPU is likely to stall on memory reads and writes.

**Q5) Specify the benchmark name and working set size of the application input (in KB) you are using for the benchmark. Specify cache hierarchy parameters & warmup tick/total ticks simulated. (Max 50 words)**

We used the benchmark of Quicksort from Stanford LLVM[1]. Our input integer array size is 3x10^6, which roughly translates to 12 MB.

| Cache | Configurations |
|---|---|
| L1 instruction | 32 kB, 8-way, 64 B line size, latency 3, per core |
| L1 data | 32 kB, 8-way, 64 B line size, latency 3, per core |
| L2 | 2 MB, 16-way, 64 B line size, latency 15, shared |

Table Q5a: Configurations of different caches in Intel Core 2. Data adapted from Table 8.3 of [3].

In our experiments, we use the above cache configurations, as mentioned in Table Q5, with single CPU core. We also use StridePrefetcher for the l2 cache.

| Metrics | Value |
|---|---|
| system.cpu.dcache.tags.warmupTick | 106147500 |
| system.l2.tags.warmupTick | 45014845000 |
| simTicks | 3081702328000 |

Table Q5b: Warmup Ticks and total ticks for X86TimingCPU with 2Ghz and DDR3_2133_8x8 memory type.

## Conclusions

Based on the discussion and graphs, we understand that quicksort's performance is highly dependent on the metric we choose to compare. For instance, X86MinorCPU works better than X86TimingSimpleCPU if we compare simulation seconds. However, if we compare metrics like average L1 data cache miss latency there is no clear distinction on which model performs better.

## References

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

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

[3.] https://www.agner.org/optimize/microarchitecture.pdf