Simulation of ARM and X86 Micro Processors using In-order and Out-of-Order CPU Models with Gem5 Simulator

https://github.com/venkatsasank/Gem5-Simulations

Venkata Sasank Pamulapati
Department of Electrical and Computer Engineering,
University of North Carolina at Charlotte
Charlotte, NC, USA
vpamulap@uncc.edu

Anurag Kalagatoori
Department of Electrical and Computer Engineering,
University of North Carolina at Charlotte
Charlotte, NC, USA
akalagat@uncc.edu

Sindhu Bhavana Mandala
Department of Electrical and Computer Engineering,
University of North Carolina at Charlotte
Charlotte, NC, USA
smandal2@uncc.edu

Indrakiran Reddy Inapagundla
Department of Electrical and Computer Engineering,
University of North Carolina at Charlotte
Charlotte, NC, USA
iinapagu@uncc.edu

Abstract— **Previously, ARM and Intel x86 PC processors were pioneers in the distinctive specific territories with no contending interests. ARM, previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of_reduced instruction set computing (RISC)__architectures for__computer processors configured for various environments. x86 is a family of instruction set architecture initially developed by Intel based on the Intel 8086 microprocessor and its_8088 variants. While Intel was building processors for PCs and servers, ARM was focused on the handheld gadgets. These days, with no hard-breaking points in computing power between the various sizes of devices, ARM and Intel-based microprocessors are contenders. In this paper, a study is done to assess the performance of the two microprocessors. Both, the in-order and out-of-order, CPU models are utilized inside every architecture, and five performance metrics (total energy consumed, throughput, average cycles per instruction (CPI), and Instruction per cycle (IPC), power) are utilized to assess the work. The notable PC engineering test system, Gem5, is utilized to achieve the work. Results show that ARM microprocessors outperform x86 microprocessors in most cases.**

Keywords—x86, ARM, In-order, Out-of-Order, gem5.

I. Introduction and Motivation

Computers are computational machines executing different sets of instructions. Different architectures of these computational machines require different sets of instructions. For example, the binary code assigned to the opcode field varies based on its different implementations in different processors. Likewise, although a standard exists, the symbolic name given to instructions varies for different processors. Next to these minor differences, there are two major types of instruction set architectures that differ markedly in the relationship of hardware to software: complex instruction set computers (CISCs), and reduced instruction set computers (RISCs). CISCs (Intel x86) provide hardware support for high-level language operations and have compact programs, whereas, RISCs (ARM) emphasize simple instructions and flexibility. ARM is a family of RISC architectures, whereas x86 processor is of CISC architecture. ARM is a power efficient solution for supercomputers. ARM makes 32-bit and 64bit RISC multi-core processor. RISC processors are designed to perform a smaller number of types of computer_instructions so that they can operate at a higher speed, performing more millions of instructions per second (MIPS). The x86 architecture is a variable instruction length, primarily "CISC" design with emphasis on backward compatibility. The instruction set is not typical CISC, however, but basically an extended version of the simple eight-bit 8008 and 8080 architectures. Byte-addressing is enabled, and words are stored in memory with little-endian byte order.

The main aim of this paper is to distinguish between x86 and ARM microprocessors in both In-order and Out-of-order models. Here in this paper, the performance of ARM and x86 Microprocessor is evaluated using a renowned computer architecture simulator called Gem5. One of the main characteristics of Gem5 its Ability to simulate a complete system, including peripherals and network connections. Gem5 supports multiple architectures and has two modes of operation: system call emulation (SE) and full-system emulation (FS) emulation; the former acts as a functional simulator with a console interface as a means of communication to the user, the latter allows user interaction with the simulated operating system. Simulating is done by using a group of benchmark applications i.e. Mi bench. Mi bench majorly consists of 35 applications out of which we chose six applications namely Basic-Math, Dijkstra, Quicksort, Bit-count, Susan and Patricia. Evaluation has been performed by considering five major parameters i.e. total consumed energy(mJ), throughput(B/s), Cycles Per Instruction (CPI), Instructions per cycle (IPC), Power (mW). Simulation is done for both ARM as well as the x86 architecture for both In-order and Out-of-order models. All the six benchmarks are simulated for five different cases and the results are noted. During the evaluation process while doubling the level-1 and level-2 caches, the miss rate is increased which totally has neither rhyme nor reason. This is because the Gem5 simulator was working under System call Emulation mode, so it is better to use Full System emulation mode to evaluate the performance. In most of the cases ARM outperforms x86, but x86 works better when CPU model is out-of-order.

II. Approach

Simulation of the architecture does not just mean running with certain constraints and getting the output of that program. It should be able to generate enough data in order to provide enough information to evaluate the architectures. For this project, we looked for simulators to fulfill the above needs. QEMU emulator is good enough to build the target architecture but requires additional work to generate statistics. Simulators like Gem5 is widely appreciated as it provides System Call emulation mode and Full system emulation mode. Gem5 also supports a wide range of ISAs including ARM, X86, ALPHA, SPARC, MIPS, POWER. Running a benchmark on Gem5 is easy with the available scripts. It generates a lot of important data from running a simulation.

Some of the information it generates help to determine the performance of a configuration in terms of energy, average power and even cache coherency. Some other information like IPC, page hits, throughput are useful in case of comparing different architectures. As this project involves two different architectures being compared for different CPU models, it makes sense to take as many parameters as possible into consideration. Some of the very important parameters considered for this project like Average power, energy, Instructions per cycle, throughput are easy to obtain from a Gem5 simulation. Gem5 simulator is also capable of simulating a complete system, full of network stack and peripherals. Also, it has two different approaches to the memory system. Classic, for a good pipeline simulation and ruby for a good memory hierarchy simulation. Gem5 is also easy to install in Unix based operating systems. Due to all the above reasons, we chose to use the Gem5 simulator to build our microprocessor models. In-order and out-of-order CPU models simulations are done in this project. This helps to analyze the performance of the architectures for both CPU models and determine which model is better for which application.

For the benchmarks, Mi bench benchmark suite is chosen for this project. Mi bench is the publicly available set of benchmarks. It has 35 different applications divided into six categories. Benchmarks should be able to stress on all the configuration parameters of the simulated architecture. From Mi bench, we considered Basic math, Dijkstra's benchmark, Qsort benchmark, Bitcount benchmark, Patricia benchmark and Susan benchmark. Dijkstra's, Patricia benchmarks emphasize on cache performance as they are sparse like applications. Dijkstra's, Qsort, Susan benchmarks taken input files. These are linked at the build time.

All the simulations are done for two CPU models and two cache configurations each. Cache configurations taken in this project are L1, L2 as a) 32kB, 512kB b) 64kB, 1MB. In-order and Out-of-order CPU models are given using the "MinorCPU" and "DerivO3" options during the build.

III. Simulation setup

Gem5 Installation:

Gem5 can be installed in Unix like operating systems, in this project we have used ubuntu 16.04 and the simulator can be downloaded from the git repository link provided in the website. The hardware required for the simulator is 64bit. The following are the steps to install the gem5 simulator:

1.sudo apt-get install update
2.sudo apt-get install upgrade
3.sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python-dev python
Then we need to install few dependencies before proceeding with the gem5 simulator here are the following dependencies that need to be installed.

a. **sudo apt-get install git**

This command is used to install the Git in ubuntu. Git is the distributed version control system, the gem5 uses Git for version control.

b. **sudo apt-get install build-essential**

This is used to install gcc 4.8+ and is used for the environment variables to point to a non-default version of gcc.

c. **sudo apt-get install scons**

This is used get scons, gem5 uses scons as its build environment. scons is like make on steroids and uses Python scripts for all aspects of the build process. This allows for a very flexible (if slow) build system.

d. **sudo apt-get install python-dev**

gem5 relies on the Python development libraries. To install these on Ubuntu use.

e. **sudo apt-get install libprotobuf-dev python-protobuf protobuf-compiler libgoogle-perftools-dev**

"Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data." In gem5, the protobuf library is used for trace generation and playback. Protobuf is not a required package, unless you plan on using it for trace generation and playback.
After installing all the dependencies, we then continue with the installation process.

4. git clone https://gem5.googlesource.com/public/gem5
Change directories to where you want to download the gem5 source. Then, to clone the repository, use the git clone command.
5. cd gem5
This will open the gem5 directory.
6. scons/build /ARM/gem5.opt (ARM); scons/build/X86/gem5.opt (X86)
This will build the gem5.out binary file for the given ISA.
BENCHMARKS COMPILATIONS:
Now the installation part is done for the gem5 simulation. Let us see the building of the benchmarks and how to compile them using the commands.

We have downloaded the benchmarks from the Mi Benchmark. It is a collection of 35 applications divided into 6 different categories, and their source codes are available publicly for free. The applications that are used in this project are mentioned.

1. Basicmath
2. Bitcount
3. Qsort
4. Dijkstra
5. Patricia
6. Susan

After downloading the benchmarks, extract all the benchmarks to the gem5 folders and follow the steps as shown.

The steps to run the benchmarks are given:

Generate the executable files by compiling the c files, or else use the make file given.

Now using the command, run the simulation

```
./build/X86/gem5.opt -d ~/basicmath ./configs/example/se.py -
-ruby -c ./automotive/basicmath/basicmath_small --cpu-
type=DerivO3CPU --num-cpus=1 --caches --l2cache --
l1i_size=32kB --l1d_size=32kB --l2_size=512kB --
l1d_assoc=2 --l1i_assoc=2 --l2_assoc=8 --cacheline_size=64 -
-options=10
```

This is the sample command to run the benchmarks and the following output stats are generated at stats.txt file and their configurations can be verified in the configs.ini file. The option -d is used to generate the output file at the user defined directory else the output is generated at m5out directory. This creates a problem because it erases the previously generated output file whenever a benchmark is run.

For few benchmarks we also need to include the data file to retrieve the data for the benchmarks, for that we use -o /"file path" option to link the data file to the benchmark. If there are any command Line arguments, give it along with the path. For ARM we must use the cross compilation before running the benchmarks, we use the following command for the cross compilation

arm-linux-gnueabihf-gcc

Follow these steps to enable the In-order CPU model for both ARM and X86:

- Under the gem5 folder open the build_opts folder.
- Open the text file using gedit editor. (X86 or ARM)
- The default text in the line 2 statement is CPU_MODELS='AtomicSimpleCPU, O3CPU, TimingSimpleCPU'.
- Add "MinorCPU" at the end of its statement CPU_MODELS='AtomicSimpleCPU, O3CPU, TimingSimpleCPU, MinorCPU'.
- Save the file and close it.
- Under the gem5 folder open build and then open variables folder.
- Open the text file using gedit program. (X86 or ARM)
- Be sure that the second line contains "MinorCPU" CPU_MODELS='AtomicSimpleCPU, O3CPU, TimingSimpleCPU, MinorCPU'.
- Save the file and close it.
- Open the terminal, type cd gem5 and press enter.
- Type scons/build/X86/gem5.opt or scons/build/ARM/gem5.opt and then press enter.
- Wait until the build operation is completed.
- Proceed with the benchmarks compilations as mentioned in the above steps for running the "benchmarks compilations".

IV. CONFIGURATIONS

To evaluate the performance of both the architectures, we use two different cache configurations for both the cpu models and measure five performance metrics i.e. CPI(Cycles per Instruction), IPC (Instructions per cycle), Total Energy (Total energy per rank), Average power (Power consumed per core) and Throughput. System-call emulation mode is used. We used two level cache with L1 being 32kb and L2 being 512kB, and the other configuration with L1 being 64kb and L2 being 1MB.

The configurations are clearly depicted below:

CPU MODEL	:	In-order("MinorCPU") Out-of-Order("DerivO3CPU)
Number of Cores	:	1
L1I Cache	:	32kb,64kb
L1I Associativity	:	2
L1D Cache	:	32kb,64kb
L1D Associativity	:	2
L2 Cache	:	512kB, 1MB
L2 Associativity	:	8

V. RESULTS AND ANALYSIS

Simulations results are plotted below, and the analysis of the results is done in this section.

1.Average Cycles Per Instruction (CPI): CPI is the number of cycles required to execute a single instruction. From the graphs plotted we can observe that Out-of-Order model has less CPI than In-order Model. This is because Out of order model doesn't execute instructions in sequence. It removes the dependent instructions out of the path and executes the independent instructions. This reduces the stalls that is created by memory access latency i.e. cache misses. This reduces the average CPI. When cache is doubled, the number of misses is reduced and hence reduces the latency caused by cache misses. This reduces the average cycles per instruction. We can also observe that ARM has lower CPI when compared to X86.

2. Instructions Per Cycle (IPC): IPC is the number of instructions that is executed per cycle. For a better system, the Instructions per cycle should be large. Instruction per cycle is the inverse of cycles per instruction(CPI) and a major factor that affects the performance of any processor. From the graph we can depict that IPC value for In-order is less than Out-of-order for the same reasons mentioned for the CPI i.e. by removing the dependent instructions out of the path, a greater number of instructions is executed per cycle reducing the stalls created by dependent instructions. ARM has better IPC than compared to x86 and hence we can decide that ARM has better performance than x86.

3. **Total Energy:** This is the total energy consumed by the processor per rank. ARM Processors are designed in such a

fashion that it consumes less energy when compared to X86. From the graphs we can see that total energy consumed by arm is less when compared to the X86. As the cache size is doubled, the number of cache misses is reduced and hence the energy consumed is also less because of reduction of energy that is consumed by the memory access. ARM outperforms X86 in terms of energy consumption.

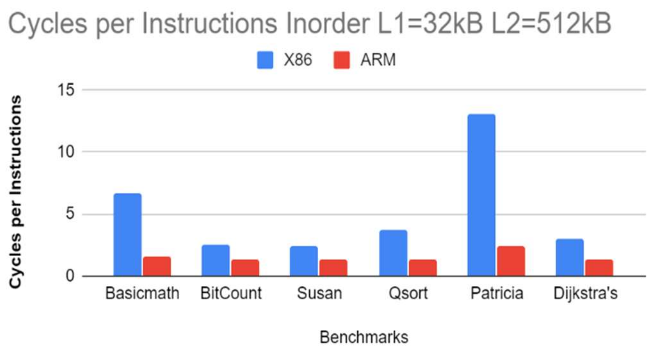
4. **Total Power:** Total power consumed by core is measured and we can clearly see that ARM consumes less Power when compared to x86. Out-of-order model consumes more power than the In-order model. This is because in the Out-of-order Model, the number of resources is increased, and the increased resources consumes more power than the In-order model. Though the consumed power is little high, the performance benefits of using out-of-order model overcomes this disadvantage.

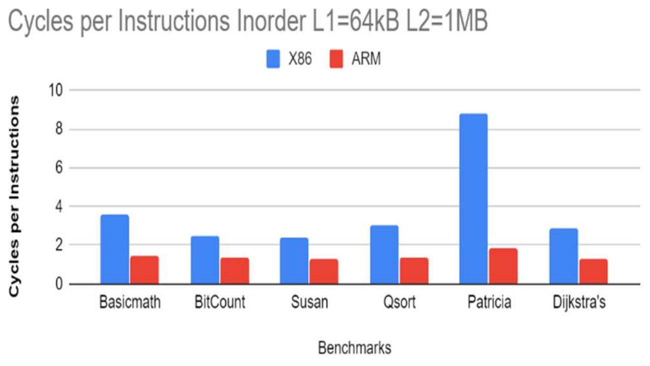
5. Throughput: Throughput is the total number that is transmitted to and from the memory per time. As the CPU model changes from In-order to Out-of-model, the instructions are fetched parallelly and the cache miss rate is increased because the spatial locality is reduced. If the instructions are executed sequentially, and the cache fetches the data as blocks there by the data around the required byte is also fetched. Because of increase in cache misses, throughput increases.

VI. Figures

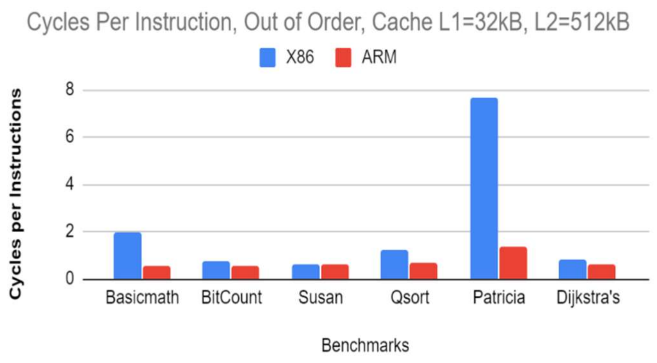
AVERAGE CPI:

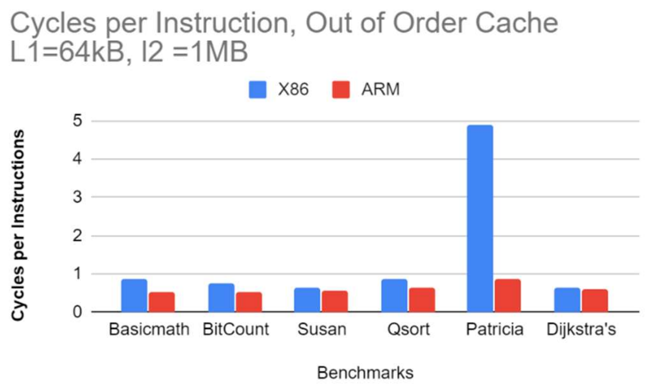
In-Order:





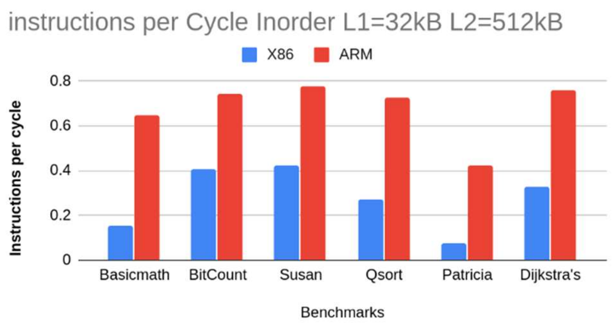
Out-of-Order:

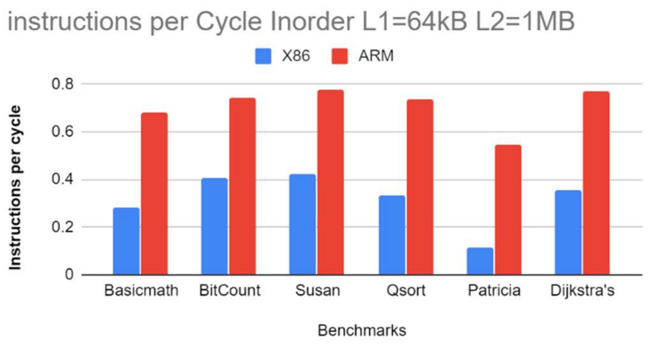




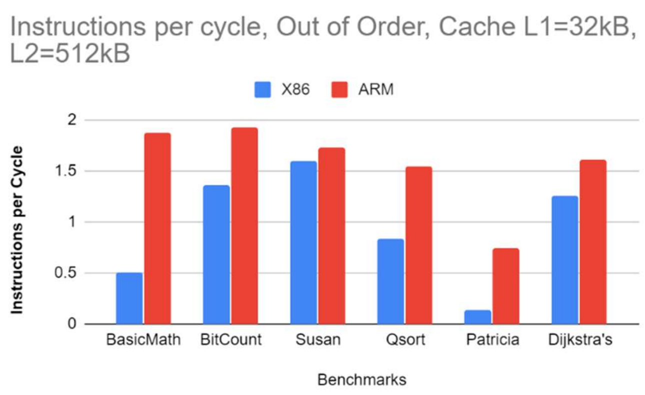
AVERAGE IPC:

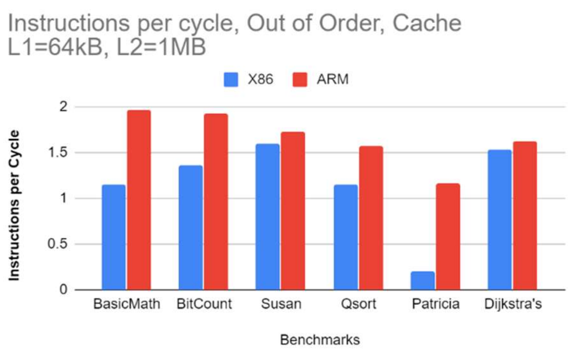
In-Order:



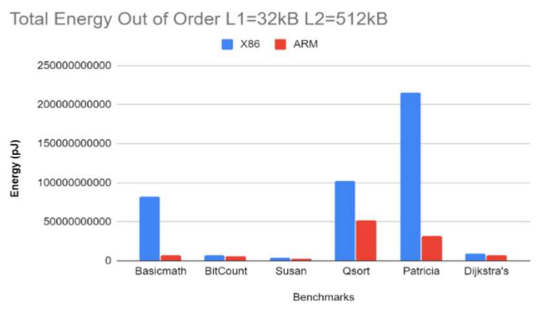


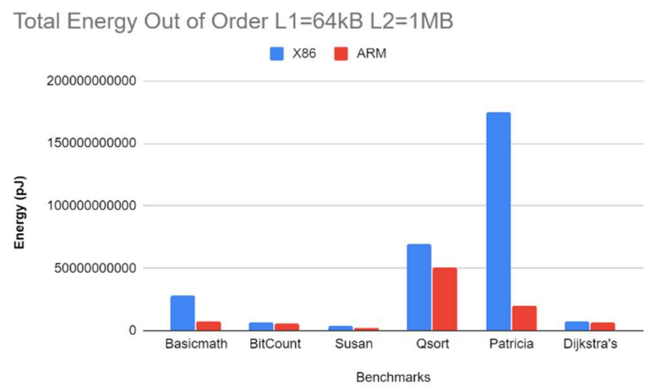
Out of Order:





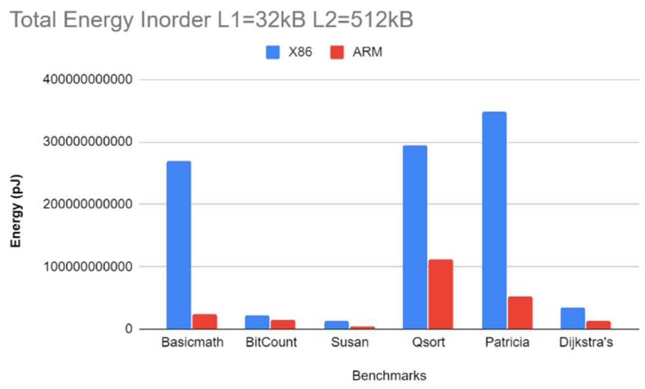
Out-of-order:

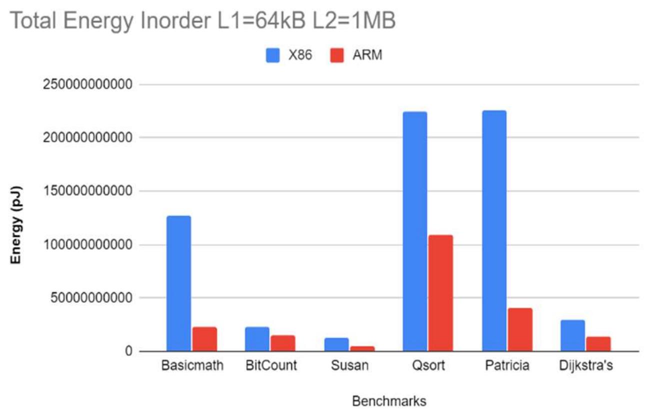




Total Energy Per Rank:

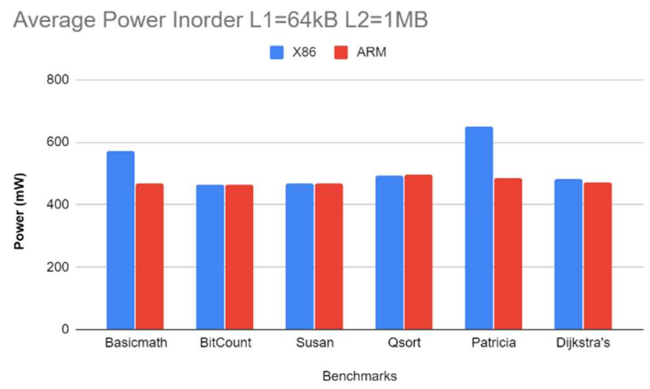
In-order:

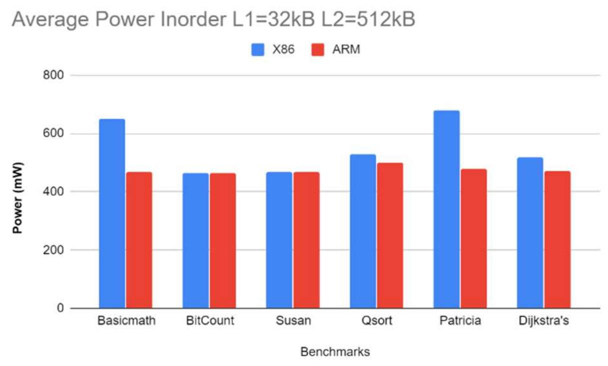




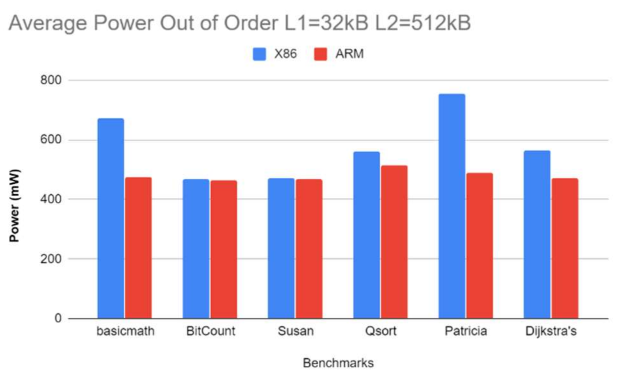
Average Power Per Core:

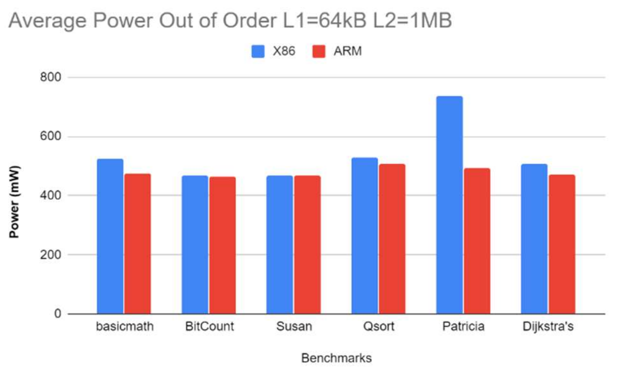
In-order:





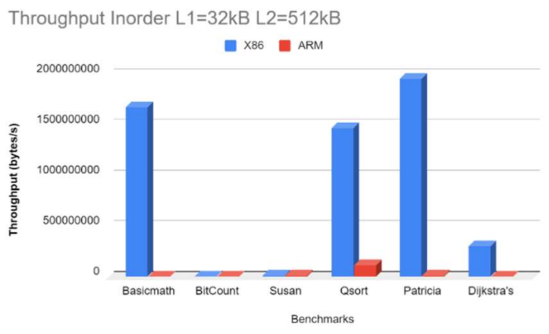
Out-of-Order:

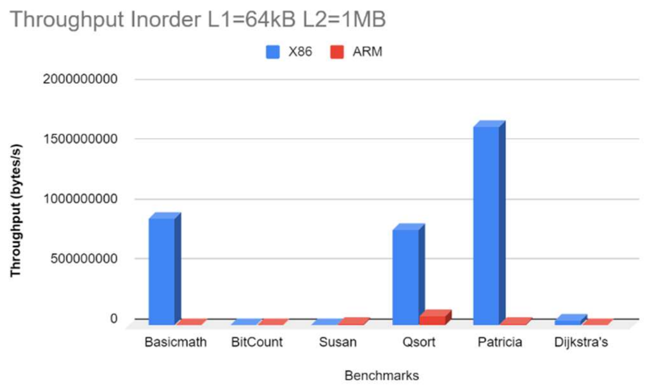




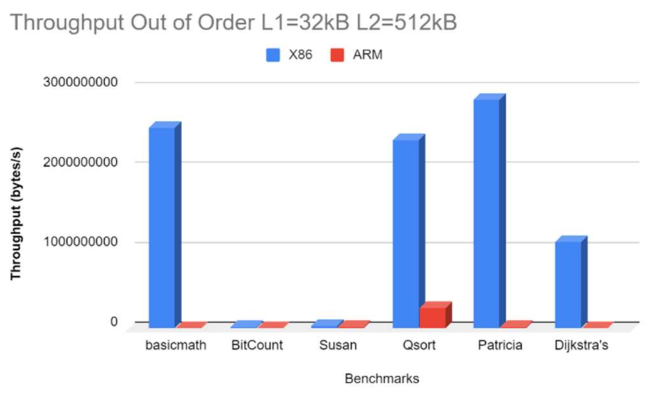
Throughput:

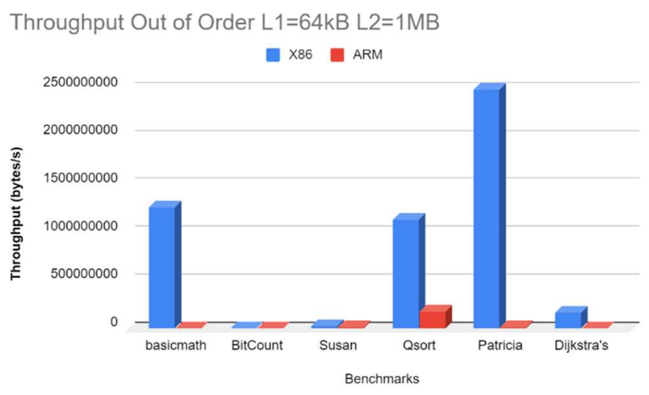
In-order:





Out-of-Order:





VII. Conclusion and Future Work

In this project, we compare the two major competitors in processors field, i.e. ARM and x86 using both In-order and out-of-order models by using Gem5 simulator. For simulation we use a set of benchmarks called Mibench benchmark suite which consists of 35 applications. we chose 6 benchmarks depending on the complexity and nature of benchmarks i.e. Basic math, Dijkstra's, Qsort, Susan, Patricia and Bitcount. For comparing the two processors we selected few performance metrics like CPI, IPC, Total Energy per rank, Average Power per core and throughput. Since cache results are not reliable in system emulation mode, we decided to ignore results of cache. By analyzing the results of simulations, we can clearly say that ARM outpowers x86 in most of the cases. Out-of-Order performs well when compared to In-order CPU model.

Few results of the System-call-emulation mode is not reliable and hence it is advised to use Full-system emulation mode for better results. Moreover, running larger set of input data can give you better performance in the results. The obtained stats.txt file which contains statics have large set of performance metrics that can be used to compare both architectures in a better way.

The stats file contains many metrics like write latency, read latency and many more. Analyzing more metrics yields better results for the comparison.

VIII. CHALLENGES FACED

1. Setting and building the environment for gem5 simulator and solving the errors with installing various dependencies.
2. Simulating benchmarks with larger data sets takes a huge time to generate the output file.
3. While generating the output files for few benchmarks, the linking of the input file is required, and few files needed command line arguments. Figuring out how to pass it to the simulator was a major challenge.
4. Memory and performance issues while building on a virtual machine.
5. Running so many commands is a problem and hence we created a script file to run the simulations and this can be found in the repository.

IX. APPENDIX A

In order to build the full system emulation mode, we need to create an emulator using gem5 and kernel image files and run the script file. The procedure to build a full system emulator is given below.

- First Build the gem5 simulator as per the instructions given above.
- Download the kernel image files required for either ARM/X86 from the website http://m5sim.org/dist/current/arm/ for ARM
- And for X86 http://m5sim.org/dist/current/arm/.
- Once you download the image file, extract the image file and place it in a folder.
- Now set M5_Path variable using the command echo export " M5_Path="******Path to the folder****" > ~/bashrc
- Use source ~/.bashrc to refresh the bashrc file and use echo $M5_PATH to verify the path
- Once the path is set, we need to copy the benchmark to the image file and to do it follow the below instructions.
- mount -o loop,offset=32256 linux-parsec-2-1-m5.img /mnt/m5_disk
- mkdir -p /mnt/m5_disk/benchmarks/
- cp qsort /mnt/m5_disk/benchmarks/
- umount /mnt/m5_disk/
- Now add the benchmark to the benchmarks.py file
- 'qsort': [SysConfig('qsort.rcS', '512MB')],
- Once unmounted write the script file that launches the emulator.

```
#!/bin/sh
cd benchmarks
echo "Running qsort now..."
./qsort -t -p1
#Gracefully exit M5
/sbin/m5 exit
```

- Now launch the emulator using the following command: ./ build/ARMgem5.opt configs/example/fs.py –mem-size=256MB
- use telnet localhost 3456 in a new terminal to interact with the console

X. REFERENCES

[1] Anas Ahmad Abudaqa ; Talal M. Al-Kharoubi ; Muhamed F. Mudawar and Armin Kobilica, "Simulation of ARM and x86 microprocessors using in-order and out-of-order CPU models with Gem5 simulator". In 2018 5th International Conference on Electrical and Electronic Engineering (ICEEE) (pp. 317-322). Istanbul, Turkey, May 2018, IEEE.

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood," The gem5 Simulator", May 2011, ACM SIGARCH Computer Architecture News.

[3] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on, 2001, pp. 3–14.

[4] J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach. Elsevier, 2011.

[5] http://learning.gem5.org/book/index.html

[6] https://bt.nitk.ac.in/r/GEM5-FS.pdf