Addis Ababa University

Addis Ababa institute of Technology

School of Electrical and Computer Engineering

Advanced Computer Architecture Project

Name:

1. Akinaw Mengesha GSR/3370/13
2. Meseret Dereje GSR/4274/13

Submited to: Dr. Fitsum

Date: May.3,2021

Table of Contents

1. Objective

1.1 General objectives

- To experience computer-system architecture research aspects and encompassing system-level architecture as well as processor micro-architecture.

1.2 Specific objective

- To compile and run gem5
- To implement Sieve of Eratosthenes using C++ and run using gem5
- To simulate gem5 on different architecture such as ARM cortex A15 and measure the performance of a system.

2. Problem of Statement

When you look at the computer engineering methodology you have technology trends that happen and various improvements that happen with respect to technology and this will give rise to newer and newer architectures. You have to evaluate the existing systems for bottlenecks and then try to come up with better architectures and this process continues. While evaluating the existing systems for bottlenecks, you will have to have certain metrics and certain benchmarks based on which you'll have the evaluation done.

You should basically be able to measure performance, report performance and summarize performance. While discussing about performance, you should be able to answer some questions like this:

- Why is some hardware better than others for different programs?

- What factors of system performance are hardware related? (e.g., Do we need a new machine, or a new operating system?)

- How does the machine's instruction set and architecture affect performance?

Performance is important both from the purchasing perspective and the designer's perspective. When you look at the purchasing perspective, given a collection of machines, you'll have to be able to decide which has the best performance, the least cost, and also the best cost per performance ratio. Similarly, from a designer's perspective, you are faced with several design options like which has the best performance improvement, least cost and best cost/performance. Unless you have some idea about the performance metrics, you will not be able to decide which will be the best performance improvement that you can think of and which will lead to least cost and which will give you the best cost performance ratio. Understanding what factors in the architecture contribute to the overall system performance and the relative importance and cost of these factors are the only ways of improvement in computer technology.
We use Sieve of Eratosthenes algorithm and measure the performance by varying the performance matrix to evaluate which architecture ARM CORTEX 15 or A9 and affected by factors explained as well will be shown in result & discussion section of the report.

3. Methodology

In mathematics, the sieve of Eratosthenes is an ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the first prime number, 2. The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime.

We will write a simple C++ program(sieve.cpp) to implement the Sieve of Eratosthenes. Considerations of this sieve, we want an output of one single integer at the end: the number of prime numbers <= the input argument, taken from the command line. Compiling program as a static binary. For the number of prime numbers <= 100 000 000, the output should be 5761455.

Compiling gem5

Gem5 is tool used which is a system simulator. It is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture. For usage of gem5 it should be installed and compiled using the following steps:

First of all, you need to install the dependencies by typing the commands 1-7 in your terminal

1. sudo apt-get install mercurial
2. sudo apt-get install build-essential
3. sudo apt-get install scons
4. sudo apt-get install python-dev
5. sudo apt-get install swig
6. sudo apt-get install libprotobuf-dev python-protobuf protobuf-compiler libgoogle-perftools-dev
7. sudo apt-get install swig gcc m4 python python-dev libgoogle-perftools-dev mercurial scons g++ build-essential
8. Now you need to clone the gem5 repository and build it onto your system by following steps 8-12.
9. hg clone http://repo.gem5.org/gem5
10. cd gem5/
11. scons build/ARM/gem5.opt jX, X for number of cores

gem5 on Linux gem5 will work the best on Linux. The Building gem5 page lists the dependency names on Ubuntu. If you are using a different distribution, you should be able to find the corresponding packages through your package manager, if the names aren't the same.

gem5 on OS X gem5 will run on OS X. However, in years past, we have had a lot of difficulty with certain tools used in the compilation process. In particular, we have had trouble with Python versions, SCons, the tool used to compile gem5, and LLVM, which Xcode uses as its back-end for gcc. See the Common Errors section for more information.

gem5 on Windows

There is limited support for Windows running the Windows Subsystem for Linux. If you use Windows, consider running a distribution of Linux in a VM or dual-booting. This will be a good idea for the rest of your time here as a student at Davis, as well.

gem5 on the CSIF

gem5 will run on the CSIF machines. Your regular CSIF home directory does not have enough space to store gem5. After your login, you will need to switch directories to /home2/<username>. This directory has enough space for you to work in gem5.
The contents of /home2 will be deleted shortly after the quarter ends. Make sure to back up anything you want to keep.

Compilation Instructions
When you are compiling, you will need to pass the following option to SCons:

- Build or compile using steps specified above…1-7 steps.
- Create C++ binary file of the C++ program called sieve of Eratosthenes the steps put on the link https://jensd.be/1126/linux/cross-compiling-for-arm-or-aarch64-ondebian-or-ubuntu
- Then after the building process is over, develop a python script. The python script was updated from ARM cortex A9 based on the criteria given which are given below in appendix section.
- An Out-of-Order CPU capable of fetching, decoding, issuing, and dispatching 3 instructions per cycle. The CPU must have the following functional units:
    - ✦ 2 integer ALUs
    - ✦ 1 integer multiplier
    - ✦ 1 integer divider
    - ✦ 1 memory read/write port,
- Separate 32KB, two-way set-associative L1 caches for instructions and data with 64byte cache lines
- a 4MB, direct-mapped L2 unified cache with 64-byte cache lines next we place the modified script in configs path *gem5/configs/*.

Simulation based question
Finally, we will describe the changes in performance between our tests by answering basic questions below. Consider m5out/stats.txt output

- **The total number of instructions committed.** This total includes all the instructions that have completed all the phases of instruction execution, including writing their results to the register file. Why is this number not equal to the number of instructions executed as reported by the stats? Is this latter number the same as the one set-up in options. maxinsts (10^8)?
- **The total number of committed branches and their frequency with respect to the number of committed instructions.** Does this seem to be consistent with what the common knowledge about branch occurrences in integer programs is? (In computer architecture programs that mostly operate on integer types (of any size) are called "integer programs" to differentiate them from scientific

applications that operate mostly on floating-point types, and thus are referred to as "FP programs")

- **The number of accesses to the L2 cache.** If gem5 had not printed it out, how would you deduce it from other metrics generated by the simulator?

Answer the following questions:

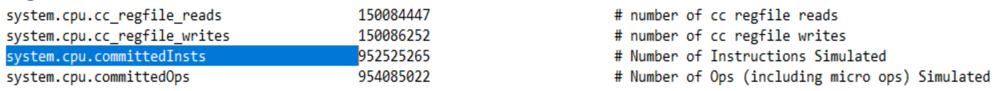
- In general, we say "L1 I-cache miss ratios are negligible while L1 D-cache miss ratios are not". Is this true for your experiment? The answer might vary depending on the benchmark or application run on the simulator. If you observed a non-negligible I-cache miss rate, what factors could have contributed to it?
- Accesses to the memory hierarchy are a major contributor to the increase of the CPI. You should attempt to quantify the contribution of cache misses to the CPI. For that, you should decompose it into the contribution of misses in each of the 3 caches (L1-I, L1-D and L2). Be careful that the hit ratios are related to accesses to the various caches and that you have to convert that into figures related to frequency of instructions. For this question, you should also note that latencies in stats.txt sometimes are printed in *ticks*, which are equivalent to the amount of *pico*seconds in the simulation. Therefore, for a 2GHz machine, for instance, 1 cycle == 500 ticks.
- List three other possible contributions to the increase of the CPI.
- The best time to be critical and to offer constructive suggestions about an assignment is soon after you complete it. Please include in your report the following:
  - A narrative description of any difficulties or misunderstandings that made ○ the assignment unnecessarily difficult, or that led you to waste time. ○ Suggestions of changes that could make this assignment more interesting,
  - more relevant or more challenging in future editions of this course.
- **Proper typesetting of the report and overall presentation style.**
  This includes the use of properly referenced figures, tables, and graphs (with descriptive axis titles and proper identification of units of measurement) where applicable.

4. Result and Discussion

From stat.txt file I answered the question asked above.

1. **The total number of instructions committed**. This total includes all the instructions that have completed all the phases of instruction execution, including writing their results to the register file. Why is this number not equal to the number of instructions executed as reported by the stats? Is this latter number the same as the one set-up in options? maxinsts (10^8)?

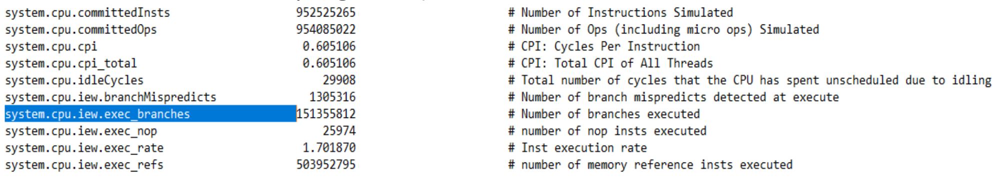
➢ From the stat.txt file **the total number of instructions committed** as shown in the figure below



```
system.cpu.cc_regfile_reads        150084447                 # number of cc regfile reads
system.cpu.cc_regfile_writes       150086252                 # number of cc regfile writes
system.cpu.committedInsts          952525265                 # Number of Instructions Simulated
system.cpu.committedOps            954085022                 # Number of Ops (including micro ops) Simulated
```

The flags aren't mutually exclusive. int_insts is any instruction that accesses an integer register and fp_insts is any instruction that accesses an fp register. Most load instructions are going to be integer operations. As see from figure the value 952525265 which is some difference to that of maxinst (10^8)

2. **The total number of committed branches and their frequency with respect to the number of committed instructions**. Does this seem to be consistent with what the common knowledge about branch occurrences in integer programs is? (In computer architecture programs that mostly operate on integer types (of any size) are called "integer programs" to differentiate them from scientific applications that operate mostly on floating-point types, and thus are referred to as "FP programs")



```
system.cpu.committedInsts                952525265                 # Number of Instructions Simulated
system.cpu.committedOps                  954085022                 # Number of Ops (including micro ops) Simulated
system.cpu.cpi                            0.605106                 # CPI: Cycles Per Instruction
system.cpu.cpi_total                      0.605106                 # CPI: Total CPI of All Threads
system.cpu.idleCycles                        29908                 # Total number of cycles that the CPU has spent unscheduled due to idling
system.cpu.iew.branchMispredicts           1305316                 # Number of branch mispredicts detected at execute
system.cpu.iew.exec_branches             151355812                 # Number of branches executed
system.cpu.iew.exec_nop                      25974                 # number of nop insts executed
system.cpu.iew.exec_rate                  1.701870                 # Inst execution rate
system.cpu.iew.exec_refs                 503952795                 # number of memory reference insts executed
```

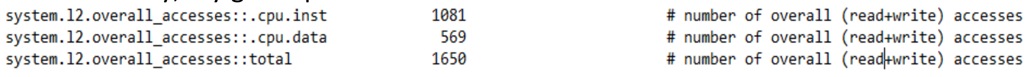
- As we see from graph the number of committed branches 151355812 and their frequency with respect to number of instructions executed is about 0.16 percent. I think it depends types of type of integer programs.

3. The total number of block replacements for the L1 data cache.

- The first level cache is composed of 4 separate instruction and data cache. Each with 64 kb. All caches use a common block size of 64 kb. All caches use a common block size of 64 kb (16,32 words). The l1 instruction and data cache are both 2 way set associative and unified (used both instruction and data). L2 cache is 16-way associative

4. The number of accesses to the L2 cache. If gem5 had not printed it out, how would you deduce it from other metrics generated by the simulator?

- Unfortunately, my gem5 printed out

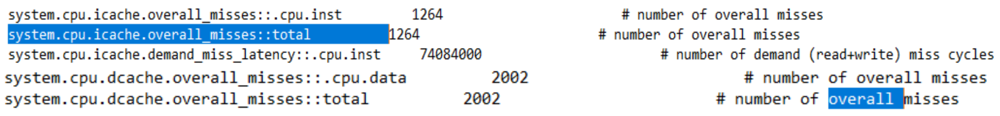


```
system.l2.overall_accesses::.cpu.inst           1081                      # number of overall (read+write) accesses
system.l2.overall_accesses::.cpu.data            569                      # number of overall (read+write) accesses
system.l2.overall_accesses::total               1650                      # number of overall (read+write) accesses
```

  I think I will calculate the overall number of accesses to the listed cache and will be subdivided according to rules stated in number 3.

5. In general we say "L1 I-cache miss ratios are negligible while L1 D-cache miss ratios are not". Is this true for your experiment? The answer might vary depending on the benchmark or application run on the simulator. If you observed a non-negligible I-cache miss rate, what factors could have contributed to it?

- From stat file overall l1 I cache miss and from overall l1 d cache miss



```
system.cpu.icache.overall_misses::.cpu.inst          1264                   # number of overall misses
system.cpu.icache.overall_misses::total          1264                   # number of overall misses
system.cpu.icache.demand_miss_latency::.cpu.inst     74084000                # number of demand (read+write) miss cycles
system.cpu.dcache.overall_misses::.cpu.data          2002                      # number of overall misses
system.cpu.dcache.overall_misses::total          2002                      # number of overall misses
```

  The number varies in but I can’t say the l1 I cache miss is negligible

6. List three other possible contributions to the increase of the CPI.

- By changing the reorder buffer entries
- Building the system with ALPHA ISA
- By Simulating system call emulation mode

5. Conclusion and Suggestion:

The purpose of experiment is dual-fold. First, it will expose us to gem5_Second, it will give you experience measuring performance on different systems, and comparing and contrasting those systems. Based on the analysis concluded, gem5 is modular platform for computer system architecture research, encompassing system-level architecture as well as processor micro-architecture. By varying parameters like processor architecture, we can measure the performance.

In this experiment sieve of Eratosthenes algorithm which calculates the number of prime numbers <= 100 000 000, the output expected to be 5761455. You should use something large enough that the application is interesting, but not too large that gem5 takes more than 10 minutes to execute a simulation.

The simulation using sieve program which calculates specified number is simulated using armA15.py python script which is available at appendix section of this report and the simulation output is discussed at Result and Discussion section of this document.

6. Appendix

6.1 Seive.cpp implements Sieve of Eratosthenes algorithm

```
//////////////////////////////////////////////////////////////////////////////////////////////
#include<iostream>

using namespace std;
#ifdef _WIN32
#define USE_WINDOWS_TIMER
#endif


                                   // number of primes between 2 and x:
                                   //            10 =>          4
                                   //           100 =>         25
                                   //         1,000 =>        168
                                   //        10,000 =>      1,229
                                   //       100,000 =>      9,592
                                   //     1,000,000 =>     78,498
                                   //    10,000,000 =>    664,579
                                   //   100,000,000 =>  5,761,455
                                   // 1,000,000,000 => 50,847,534
                                   // 10,000,000,000 => 455,052,511

int lastNumber; #include <stdio.h>
#include <math.h>

// simple serial sieve of Eratosthenes
int eratosthenes(int lastNumber)
{
    // initialize
        char* isPrime = new char[lastNumber + 1];
for (int i = 0; i <= lastNumber; i++)
isPrime[i] = 1;

        // find all non-primes   for (int i
= 2; i*i <= lastNumber; i++)          if
(isPrime[i])
                          for (int j = i*i; j <= lastNumber; j += i)
                isPrime[j] = 0;

 // sieve is complete, count primes   int found
= 0;
         for (int i = 2; i <= lastNumber; i++)
        found += isPrime[i];

    delete[] isPrime;
```

```
        return found;
}
#ifdef USE_WINDOWS_TIMER
#include <windows.h>
#else
#include <sys/time.h>
#endif

double seconds()
{
#ifdef USE_WINDOWS_TIMER
        LARGE_INTEGER frequency, now;
             QueryPerformanceFrequency(&frequency);
QueryPerformanceCounter(&now);        return
now.QuadPart / double(frequency.QuadPart);
#else
             timeval now;   gettimeofday(&now, NULL);
        return now.tv_sec + now.tv_usec / 1000000.0;
#endif
}
int main(int argc, char *argv[])
{
  char *a = argv[1];    int n =
atoi(a);
             printf("Primes between 2 and
%d\n\n", n);      printf("Simple Sieve\n");  int
startTime = seconds();   int found =
eratosthenes(n);            double duration = seconds()
- startTime;
             printf("--> %d primes found in %.3fs\n\n", found, duration);   getchar();
        return 0;
}

////////////////////////////////////////////////////////////////////////////////////////
```

6.2 armA15.py: gem5 configuration script

```
……………………………………………………………………………………………………………………………………………
from __future__ import print_function
from __future__ import absolute_import

import os
import optparse
import sys

import m5
from m5.defines import buildEnv
from m5.objects import *
from m5.params import NULL
from m5.util import addToPath, fatal

addToPath('../')

from common import Options
from common import Simulation
from common import CacheConfig
from common import CpuConfig
from common import ObjectList
from common import MemConfig
from common.FileSystemConfig import config_filesystem
from common.Caches import *
from common.cpu2000 import *

### Create the Options Parser
parser = optparse.OptionParser()
Options.addCommonOptions(parser)
Options.addSEOptions(parser)

### Parse for command line options
(options, args) = parser.parse_args()

### Override some options values to match the ARM Cortex A9

options.cpu_type = "DerivO3CPU"   # The A15 is an OutOfOrder CPU
options.cpu_clock = "2GHz"
options.num_cpus = 1

options.caches = 1            # Symmetric, L1 caches
options.cacheline_size = 64

options.l1i_size = "32kB"
options.l1i_assoc = 2

options.l1d_size = "32kB"
options.l1d_assoc = 2

options.l2cache = 1
options.l2_size = "4MB"
options.l2_assoc = 4

if args:
   print ("Error: script doesn't take any positional arguments")
   sys.exit(1)

### Setup the workload to execute on the CPUs
multiprocesses = []
apps = []
```

```
if options.cmd:
    process = Process()
    process.executable = options.cmd
    process.cmd = [options.cmd] + options.options.split()
    multiprocesses.append(process)
else:
    print >> sys.stderr, ("No workload specified. Exiting!\n")
    sys.exit(1)

### Optionally pipe output to a file

if options.input != "":
    process.input = options.input
if options.output != "":
    process.output = options.output
if options.errout != "":
    process.errout = options.errout




# By default, set workload to path of user-specified binary
workloads = options.cmd

numThreads = 1

if options.cpu_type == "DerivO3CPU" or options.cpu_type == "inorder":
    #check for Simultaneous Multithreaded workload
    workloads = options.cmd.split(';')
    if len(workloads) > 1:
        process = []
        smt_idx = 0
        inputs = []
        outputs = []
        errouts = []

        if options.input != "":
            inputs = options.input.split(';')
        if options.output != "":
            outputs = options.output.split(';')
        if options.errout != "":
            errouts = options.errout.split(';')

        for wrkld in workloads:
            smt_process = Process()
            smt_process.executable = wrkld
            smt_process.cmd = wrkld + " " + options.options
            if inputs and inputs[smt_idx]:
                smt_process.input = inputs[smt_idx]
        if outputs and outputs[smt_idx]:
            smt_process.output = outputs[smt_idx]
        if errouts and errouts[smt_idx]:
            smt_process.errout = errouts[smt_idx]
        process += [smt_process, ]
        smt_idx += 1
    numThreads = len(workloads)

### Using the provided options, setup the CPU and cache configuration

(CPUClass, test_mem_mode, FutureClass) = Simulation.setCPUClass(options)
CPUClass.numThreads = numThreads;


### Select the CPU count

np = 1 #options.num_cpus

### Assemble the system

system = System(cpu = [CPUClass(cpu_id=i) for i in range(np)],
                #xrange(np)]
                physmem = SimpleMemory(range=AddrRange("512MB")),
                membus = SystemXBar(), mem_mode = test_mem_mode)

# Create a top-level voltage domain
system.voltage_domain = VoltageDomain(voltage = options.sys_voltage)

# Create a source clock for the system and set the clock period
system.clk_domain = SrcClockDomain(clock =  options.sys_clock,
                                   voltage_domain = system.voltage_domain)

# Create a CPU voltage domain
system.cpu_voltage_domain = VoltageDomain()

# Create a separate clock domain for the CPUs
system.cpu_clk_domain = SrcClockDomain(clock = options.cpu_clock,
                                       voltage_domain =
                                       system.cpu_voltage_domain)

### Reconfigure the CPU to match the Cortex A9 specs

cpu = system.cpu[0]
cpu.decodeWidth = 3
cpu.fetchWidth = 3
cpu.issueWidth = 3
cpu.dispatchWidth = 3

#Functional Unit definitions taken from FuncUnitConfig.py, modified to match A9
```

```
class IntALU(FUDesc):
   opList = [ OpDesc(opClass='IntAlu') ]
   count = 2

class IntMultDiv(FUDesc):
   opList = [ OpDesc(opClass='IntMult', opLat=3),
            OpDesc(opClass='IntDiv', opLat=20) ]
   count = 1

class RdWrPort(FUDesc):
   opList = [ OpDesc(opClass='MemRead'), OpDesc(opClass='MemWrite') ]
   count = 1

# Attach the Functional units
cpu.fuPool = FUPool(FUList=[IntALU(), IntMultDiv(), RdWrPort()])

### Sanity checks

#if options.fastmem and (options.caches or options.l2cache):
   #fatal("You cannot use fastmem in combination with caches!")

for i in range(np): #xrange(np):

   if len(multiprocesses) == 1:
      system.cpu[i].workload = multiprocesses[0]
   else:
      system.cpu[i].workload = multiprocesses[i]

   #if options.fastmem:
      #system.cpu[i].fastmem = True

   if options.checker:
      system.cpu[i].addCheckerCpu()

   system.cpu[i].createThreads()

### Connect up system memory

system.system_port = system.membus.slave
system.physmem.port = system.membus.master
CacheConfig.config_cache(options, system)

### Create the root, instantiate the system, and run the simulation

root = Root(full_system = False, system = system)
#m5.instantiate()

#print("Beginning simulation!")
```

```
#exit_event = m5.simulate()

Simulation.run(options, root, system, FutureClass)
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```