# LAB 1 - Performance in Computer Architecture

**Assessment:** 4% of the total course mark.

---

## 1 General Instructions

- ◇ start by accepting the assignment link, which will create the repo for you on your github. Then clone this repo into the VM.
- ◇ Make sure you use the provided VM for developing and running the labs. We will not help debugging any other setup.
- ◇ In the lab room machines, you can login in to the VM with your group number. You will be asked to enter a password for the first time. Feel free to copy the VM into your own laptop to work on the labs at your time.
- ◇ clone the repository into a folder named macsim-4dm4; all instructions in this document assumes this folder as the main repo folder.
- ◇ All the output files you are asked to produce should be placed under macsim-4dm4/labs/lab1.
- ◇ The submission of the code, output files, and the lab report should be done by pushing to the repository before the deadline time.
- ◇ In addition to the code modifications and output files, you are asked to prepare a pdf report of all the steps and answering the questions within the lab document. The first page of the report should be a declaration of contribution where each of the two members of the group lists clearly what they have done.
- ◇ In addition to the declaration of contribution, we will be using github commit history to track the contributions of the group. Therefore, it is very critical to commit and push updates one at a time and do not wait until you finish the full lab to push once. This will be considered a red flag.

## 2 Submission Deadline

The deadline for lab1 is Friday **Sep 22nd, 11:59PM**. Please note that this is a programmed deadline in the environment, so make sure you submit in time since you will not be able to submit after that dictated deadline.

### 2.1 Importing the starter code of the lab

You should follow exactly the same process you have been following in past labs to import the starter code from the following invitation link and to create the project:

https://classroom.github.com/a/JIH0gvv9

# 3 Environment Setup and Get Familiar with the Used Tools

In the labs, we are using a variety of tools to offer a seamless and rich learning experience for you about computer architecture.

## 3.1 Github Classroom

First of all, we are using Github classroom as a mean to 1) distribute you starter codes, 2) facilitate collaboration between group members working on the same repository similar to professional/industry experience, and 3) a history tracking mechanism to assess the contributions of the group as a whole and each group member individually. We created a new organization and class in github classroom named COMPENG-4DM4-Fall2023 and added all your McMaster Email addresses there in the class roaster. **For this first lab, you need to do the following:**

1. You have to make sure you are signing to github using your McMaster Email address.
2. Accept the assignment by clicking into this link: Since we have two team members and we are forming the teams for the first time, you need to do the following:
   (a) One of the two group members should accept the assignment first. In this process, you will be asked to create a team. Create the team.
   (b) The second group member, on the other hand, when accepting the assignment must join the same team that the first one created.

   You can see the illustration in the Form student groups section in this tutorial: https://github.blog/2018-03-06-how-to-use-group-assignments-in-github-classroom/.
3. Once accepted the assignment and joined the team, your McMaster Email in the roster should be linked by your github account.

To refresh your minds about using git, we added the same short git-guide we used in 2SH4 on Avenue here: https://avenue.cllmcmaster.ca/d2l/le/content/570077/viewContent/4333846/View.

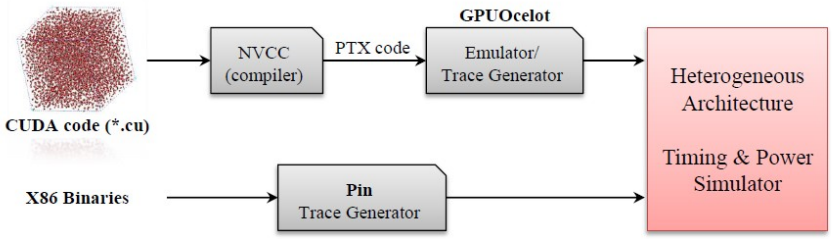


Figure 1: Pin instruments the program while running and creates a trace that MacSim uses in simulation. Do not worry about the CUDA path for now.

### 3.2 Intel Pintools

We are going to use MacSim as our architectural cycle-accurate simulator for the labs. Macsim accepts programs represented as instruction traces with a particular format. x86 traces are generated using Pin. Internally, MacSim converts both x86 and PTX trace instructions into RISC style microops (uop) for simulation. Figure 1 shows a high-level overview of this procedure.

- ◇ Pin can be freely downloaded from Intel's website: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html. The current version we are using (that is compatible with MacSim) is: Pin 3.13.
- ◇ Documentation of Pintool is here: https://software.intel.com/sites/landingpage/pintool/docs/98749/Pin/doc/html/index.html. The Examples section of the documentation is particularly very useful.
- ◇ Pin team also conducted several tutorials that can be found on the website. You should go through this tutorial: https://www.intel.com/content/dam/develop/external/us/en/documents/cgo2013-256675.pdf and understand how does Pin really instruments an application. **Since we will be using the inscount (instruction count tool) in next step, use this tutorial to understand how does this simple tool work.**
- ◇ To make sure you are running the correct pintool, open the .bashrc file at /.bashrc and add the following lines at the end of the file and replace [USER] with our home directory name.

```
export PIN_HOME=/home/[USER]/macsim-4dm4/tools/x86_trace_generator/pin-3.13-98189-g60a6ef199-gcc-linux
export PATH=$PIN_HOME:$PATH
export PIN_ROOT=$PIN_HOME
```

- ◇ To run any pintool, you need to go to macsim-4dm4/tools/x86_trace_generator/pin-3.13-98189-g60a6ef199-gcc-linux and run a command like this:

```
# cd  /macsim-4dm4/tools/x86_trace_generator/pin-3.13-98189-g60a6ef199-gcc-linux

# ./pin -t [path to tool]/toolname.so -- [path to binary]
```

For example:

```
# ./pin -t source/tools/ManualExamples/obj-intel64/inscount0.so -o ../mergesort_inscount -- ../mergesort
```

runs the inscount0 tool on the mergesort binary and places the output in a file called mergesort_inscount. Try it out and confirm that you are able to get it working.

## Exercise 1

### Part 1: Instruction Count Pintool

In this exercise, you will use the inscount pintool to collect the number of instructions for a set of benchmarks. The benchmarks are located in the following path: macsim-4dm4/tools/benchmarks. We have a total of 13 benchmarks for lab1. To first compile the benchmarks and create the binaries, run the provided script compile.sh as follows:

```
# cd  /macsim-4dm4/tools/benchmarks
```

```
# source compile.sh
```

Afterwards, using the inscount0 pintool to collect the instruction count for all the benchmarks and put the results in a csv file with the following format, one row per benchmark:

```
benchmark name; instruction count
```

and name it lab1-exercise1-inscount.csv under macsim-4dm4/labs/lab1. Note that to run pintool, you have to be in the pin folder:

```
# cd /macsim-4dm4/tools/x86_trace_generator/pin-3.13-98189-g60a6ef199-gcc-linux
```

You can automoate this step by scripting it. You can use the script from Part 2 as an example.

### Part 2: MacSim Trace Generator

As we stated eariler, MacSim uses its own pintool to create a trace of a particular format for the programs to run them through MacSim later on. This pintool is located at: macsim-4dm4/tools/x86_trace_generator/obj-intel64/trace_generator.so. Go through Chapter 3: Traces of the documentation located at macsim-4dm4/doc/macsim.pdf to understand the trace format and how does it really work. Ignore the discussions about the GPU traces and focus on the x86 ones.

Afterwards, go to the pin folder again and run the following example (As one line) of the trace_generator tool to make sure you have it working:

```
# ./pin -t ../obj-intel64/trace_generator.so -tracename ../mergesort -dump
-dump_file ../mergesort -- ../mergesort
```

In addition to the path to the binary we are providing at the end, we are adding three extra parameters: tracename, dump, and dumpfile. Understand their meaning by searching for them in the trace_generator.cpp code. Now, you are required to create the MacSim traces for our 13 benchmarks.

The trace raw file is not readible but the txt file and the dump files are. Open the mergesort.txt config file and understand its format as explained in the documentation in Chapter 3. Open the mergesort_0.dump file and match the instruction data structures with the explanation in 3.4.2 trace_xx.raw in documentation.

Now, we want to create the MacSim traces for our 13 benchmarks. To do so, we are providing a script at: pin-3.13-98189-g60a6ef199-gcc-linux named createTraces.sh. The trace takes as input the path to the benchmarks directory and creates the traces in each of the benchmark folder. Open the trace to see how it works. To create the traces, invoke the trace by passing the benchmarks folder as input, for example:

```
# source createTraces.sh  /macsim-4dm4/tools/benchmarks
```

Check that the traces have been successfully created for each benchmark in that directory.

## 3.3 Editor and IDE

You can use any IDE you prefer. In fact, you can even run all the labs without the need for any IDE since macsim buidling and running is simply invoked from commandline. However, to facilitate code navigation, we suggest that you use vscode. VScode is already installed in

the VM you are using and can be found by searching for it. Open the folder of your repository in VScode. Good news is that you can also easily deal with git within the IDE. You can also install the 'C/C++ Extension Pack' extension from Microsoft for the C++ language.

# 4 MacSim

MacSim is a heterogeneous architecture simulator, which is trace-driven and cycle-level. It thoroughly models architectural behaviors, including detailed pipeline stages, multi-threading, and memory systems. Currently, MacSim support x86 and NVIDIA PTX instruction set architectures (ISA). MacSim is capable of simulating a variety of architectures, such as Intel's Sandy Bridge [6] and NVIDIA's Fermi [8]. It can simulate homogeneous ISA multi-core simulations as well as heterogeneous ISA multi-core simulations. MacSim is a micro-architecture simulator that simulates detailed pipeline stages (in-order and out-of-order) and the memory system components including caches, NoC, and memory controllers. It supports asymmetric multicore configurations as well as SMT or MT architectures.

# Exercise 2

## Part 1: First MacSim Experiment

- ◇ Spend sometime with the documentation of Macsim located at: macsim-4dm4/doc/macsim.pdf. Most importantly, go through the steps in the **Getting Started** Chapter.
- ◇ navigate in high-level the code structure of Macsim under src/ folder. MacSim is nicely structured where you can find that in most cases each component has its own source and header files, which faciliates learning.
- ◇ now assuming that you created the traces trace correctly from Exercise 1 Part 2, we will use MacSim to simulate the hardware while executing these application traces.
- ◇ Perform the following steps to run one MacSim experiment as an example to make sure that everything is working in your side:
  - build MacSim. In the main repository folder, run:
    ```
    # ./build.py --debug
    ```
    The passed debug option is to enable debugging capabilities of the simulator
  - configure macsim setup through the params.in file under the /bin folder. In particular, we want to simulate a single x86 core system. Make sure that the params.in has the following parameter values:
    ```
    num_sim_cores 1
    num_sim_small_cores 0
    num_sim_medium_cores 0
    num_sim_large_cores 1
    large_core_type x86
    sim_cycle_count 0
    debug_exec_stage 0
    ```

```
debug_print_trace 0
```

- update the trace_file_list file with the correct pointer the mergesort benchmark to run.
  `../tools/x86_trace_generator/pin-3.13-98189-g60a6ef199-gcc-linux/mergesort.txt`
  This is assuming you have the mergesort trace under the pin folder from Exercise 1 Part 2.
- run macsim
  `# ./macsim`
  Once done, macsim prints the total number of executed instructions. Write this done and compare it to the one you obtained from Pintools in Exercise 1 Part 1 for the mergesort benchmark. Do they match? why this is the case?

## Part 2: Program Analysis

◇ use the same MacSim setup as before, and change only the following parameter:
`debug_print_trace 1`
This will enable the execution stage debug for MacSim.

◇ For each of the benchmarks:

- update the trace_file_list to run this benchmark
- run macsim as follows:
  `# ./macsim &> benchmarkName_execution.debug`
  replace the benchmarkName with the actual benchmark name. Here we are casting the output to a file to keep the log of the execution stage.
- **IMPORTATNT HINT: you are highly recommended to automate these steps by writing a simple script since you will be doing these steps for all the benchmarks.**
- We are interested into analyzing all the UOPs of each benchmark. To do so, you can further filter the output to another file that only has the DEBUG message that is printed when a UOP finishes executing:
  `# grep done_exec benchmarkName_execution.debug &> benchmarkName_done_execution.debug`
  To understand this message, you need to open the src/exec.cpp file and search for done_exec.

◇ Now, we have the debug info we need for each benchmark, we will analyze it. As you might have already figured out from the DEBUG message, uop_type_name gives us the name of the UOP finished executing. In our analysis of the programs, we want to create an analysis table for each program that gives the total number of each UOP type.

- First, you need to know what are the available UOP types. This is defined in the uop.h header file. You should ignore all the UOP_GPU_* UOPs as we are not simulating GPUs.
- Second, you need to obtain the count of each of the UOPs in the benchmarkName_done_execution.debug file for each program. You can do it using whatever

way is better; possible suggestions are: grep commandline command (which you can also script to apply for all benchmarks if you want), taking the file into csv and analyze in Excel or Python. This is left for your choice of what works best for you. The final outcome of this should be a csv file named UOP_No.csv that looks like the following, where first line is the header, columns represent the benchmarks, rows represent the UOPs, and final line is for total number of UOPs.

```
UOP; benchmark1, ..., benchmark13
UOP_IADD; number; ....; number
....; ....; ....; ....
total_UOPS; number;....;number
```

## Part 3: Performance Calculations

In this part, we are going to calculate the average CPI as we have done in lectures. While we can calculate each of the UOP type percentages from Part 1, we are yet to figure out the cycles/instruction part.

- ◇ Each of the UOPs you identified in Part 1 has an associated latency with it. You need to figure out where this is being defined.
- ◇ Afterwards, using the number of occurrences of each UOP from Part 1 and its latency, calculate the average CPI of each benchmark.
- ◇ MacSim also reports at the end of execution the total number of instructions as well as total number of cycles. Use this to calculate the CPI from MaCsim. Compare with your own calculated CPI and comment on the comparison in your report. The output of this step is a csv file named avg_CPI.csv that is an extension of the UOP_No.csv file by adding one extra line for the average CPI as follows:

```
UOP; benchmark1, ..., benchmark13
UOP_IADD; number; ....; number
....; ....; ....; ....
total_UOPS; number;....;number
CPI; number;....;number
Total_Instructions; number;....;number
Total_cycles; number;....;number
CPI_macsim; number;....;number
```

## Part 4: Performance Improvement

In this part, we are going to study the impact of improving the latency of UOPs on the overall execution time of the program.

- ◇ For each program, pick the top most UOP type in terms of percentage of the program and reduce their latency by a factor equal to $(G + 55) * 10/110$, where $G$ is your lab group number. Since this entails modifying the code, you need to recompile.

- ◇ recalculate your average CPI and report also the total number of cycles from MacSim. The output of this step is an updated version of the avg_CPI.csv file that you should name: avg_CPI_improved.csv that has the same format as the former but with the new numbers after the modification.