

COMPENG 4DM4: Computer Communication Networks

Lab 1: Performance in Computer Architecture

Shanzeb Immad - 400382928 - immads

Josh Umansky - 400234265 - umanskyj

Exercise 1:

Part 1:

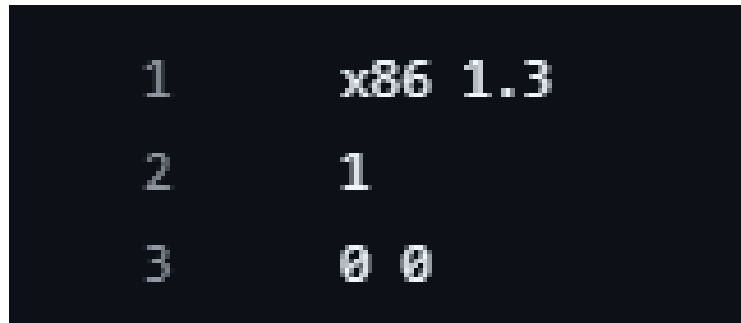
In part 1 for the exercise the task was to obtain the instruction count for each of the benchmarks compiled. To ease the process, within the pin folder a new script was created under the name 'exercise1_part1_script.sh'. This script, when given the benchmark directory as an input, goes through each of the raw files and applies the inscount0.so command and saves the value in a .txt file. The values of the txt files were then manually placed in a .csv file called 'lab1-exercise1-inscount.csv' placed under the main directory of the github repo.

1	Benchmark Name	Instruction Count
2	st-matmul	238014
3	st-wvad	244195
4	wvadd	142018
5	multiply	163521
6	memcpy	221844
7	qsort	480859
8	memBw	27881864
9	spmv	234746
10	mergesort	396804
11	towers	143274
12	rsort	673284
13	median	154848
14	memlatency	22897838

Figure 1: Instruction Count for each benchmark

Part 2:

Within this part of the lab, the goal was to create traces for each of the benchmarks. This part was tested with the mergesort file running the trace generator pin command given, this created a mergesort.txt. Once confirmed that it works, the createTraces script was run to automate the retrieval of each benchmark trace. Once the script finishes, within each benchmark folder there is a respective BM.txt, which all contains identical data when compared to each other.



```
1 x86 1.3
2 1
3 0 0
```

Figure 2: BM.txt for benchmark memcopy

When compared to the information given on the macsim.pdf, the trace files created follow the same format. The first line shows the Trace Format which is x86 and the trace version being 1.3. The second line being the value of 1 indicates the number of threads being used. The last line indicates the first threads starting instruction count, which is always zeroes.

Exercise 2:

Part 1:

To start, the build python file was run to MacSim within the main repo. From this the parameter file was edited to make the requested change of giving the debug_print_trace a value of 0. The trace file list was changed to keep one trace, and the directory to the mergesort.txt, this allows MacSim to be run on the trace of the Mergesort benchmark. By running the command of ./macsim , the total number of instructions are displayed, since this was done with the trace and macsim, the possibility of being different then the intel pin was a potential thought. That thought seemed to be true yet the difference between the two is fairly minimal with the MacSim result being 414,708 instructions while the inscount from the pin command was 396,804. This process was repeated for all the benchmark traces created in part 2 of exercise 1 and the results were fairly similar when compared to exercise 1 results.

Part 2:

To complete the program analysis section of lab 1, our group enabled the parameter to allow execution stage debug to occur for MacSim. The main objective of this part was to construct a csv file of all the UOP counts for each benchmark. This was done by utilizing a simple bash script using `grep -c` command line to obtain the total counts. The result was then placed into another .csv file named `UOP_No.csv`. By observing the csv the following trends were identified, around 28 of the 40 UOPs have 0 counts for all benchmarks. Another identified fact is that the `UOP_IADD` is the UOP with the most average count through all the benchmarks and the `memBW` has the largest UOP summed count of all the benchmarks.

Part 3:

For this part, a copy of the `UOP_No.csv` file was made named `avg_CPI.csv` to implement more rows for CPI calculations. To get the column CPI we used the equation from figure 3.

$$= \frac{\Sigma(UOP\ Count * UOP\ Latency)}{Total\ UOP\ Count}$$

Figure 3: Equation to obtain CPI for all Benchmarks

The latencies were obtained from the `uoplatency_x86.def` file. The next rows added were obtained when MacSim from part 1 was run on the traces, the total instruction and total cycles were added. The cycles were then divided by the instruction count to get the `CPI_macsim` row. Figure 5 shows the newly added rows to differ `UOP_No` and `avg_CPI`. When comparing both CPI's calculated majority of them are fairly close in values representing accurate simulations. A few benchmarks had completely different CPI's, the following table shows which benchmarks they are and the associated CPIs.

Benchmark Name	CPI	MacSim CPI	Difference +/-
MemBw	17.60281	166.972687	149.3699
qsort	9.982647213	21.39735281	11.4147056
Memcpy	9.962915678	83.76504735	73.802131672

Figure 4: Large CPI difference table

CPI;17.60281
Total_Instructions;3000000
Total_cycles;500918061
CPI_macsim;166.972687

Figure 5: New rows from avg_CPI.csv

Part 4:

With this part, a copy of the avg_CPI.csv file was made but the data is different due to changes in some latencies. The latency to be changed for each benchmark depended on the UOP with the highest count. When compared to the statement made regarding the highest average, the majority of the benchmarks except 2 had the latency of UOP_IADD changed. The only 2 that had a different UOP latency changed was both qsort and rsort where the UOP with the most counts is UOP_IMEM. Below is a table of each benchmark along with the CPI's to compare with the latency changed calculations.

Benchmark	UOP	CPI (old latency)	MacSim CPI (old latency)	CPI (New latency)	MacSim CPI (New latency)
median	UOP_IADD	9.950130095	10.21129712	9.948636171	10.25619044
memBW	UOP_IADD	17.60281	166.972687	9.995709417	16.69726867
memcpy	UOP_IADD	9.962915678	83.76504735	9.962045191	8.742261644
memlatency	UOP_IADD	9.992794827	7.735603609	9.992794827	8.275189
mergesort	UOP_IADD	9.979139347	11.84682958	9.978844962	11.87915037
multiply	UOP_IADD	9.956894394	13.44193468	9.955667216	13.67642372
qsort	UOP_IMEM	9.982647213	21.39735281	9.98241325	21.5163912
rsort	UOP_IMEM	9.985578006	8.979166456	9.985428606	9.008427985
spmv	UOP_IADD	9.76229418	9.228273433	9.757332214	9.356113035
st-matmul	UOP_IADD	9.966518497	10.14251049	9.965810509	10.26767801

st-vvadd	UOP_IADD	9.818937995	9.821662807	9.815327264	9.901420587
towers	UOP_IADD	9.946808716	9.741078076	9.944976303	9.76781373
vvadd	UOP_IADD	9.946285441	9.673711666	9.944483249	9.523784699

Figure 6: CPI table of different latencies

By observing the table it is clear that there are minimal changes to the CPI. Our group believes that this is due to only one out of the 40+ UOPs being changed and has minimal effect on the calculated values. Though it is not visible on the csv file or the table, the applied knowledge of the CPI of those UOPs must have significantly decreased due to the learned definition of latency. Latency allows the same amount of instructions to be done in a lower time. By decreasing the latency the clock cycles also deplete lowering the overall cpi.