**Mohammad Shughri 11051096**

**Results using the factorial of 6**

**Recursive factorial results:**

Task1

The total Count of macroinstructions and microinstructions is: 185245

Task2

Branch Instructions Count is: 40201

Memory Instructions is: 54826

Arithmetic and Logic Instructions Count is: 90218

Ratio of branch instructions: 0.217015

Ratio of memory instructions: 0.295965

Ratio of arithmetic instructions: 0.48702

**Iterative factorial results:**

Task1

The total Count of macroinstructions and microinstructions is: 185084

Task2

Branch Instructions Count is: 40182

Memory Instructions is: 54783

Arithmetic and Logic Instructions Count is: 90119

Ratio of branch instructions: 0.217101

Ratio of memory instructions: 0.29599

Ratio of arithmetic instructions: 0.486909

**Steps**

After downloading the PIN tool, we have to apply the following command in order to provide Ubuntu with the path of the PIN tool.

set INTEL\_JIT\_PROFILER /home/mohammad/Downloads/pin-3.17-98314-g0c048d619-gcc-linux/intel64/lib/libpinjitprofiling.so

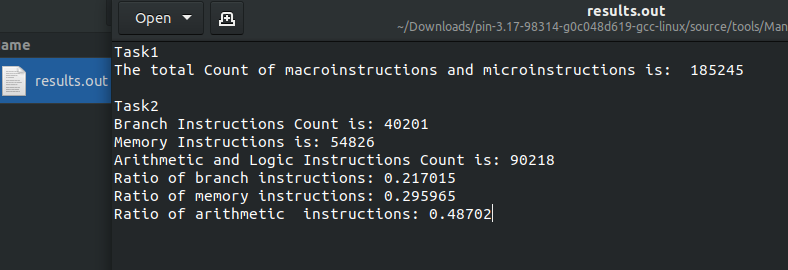
We change our directory to the tools file inside the PIN tool.

Cd /Downloads/pin-3.17-98314-g0c048d619-gcc-linux/source/tools/ManualExamples$

Inside the ManualExample, we put in our c++ code (sy.cpp) then, we create a folder obj-intel64 and put the factorial code inside it. Then, we run the c code using (gcc RecFact.c) then we execute the code using (./a.out). We put in a number and we get a new file (a.out). We move the file from obj-intel64 to the ManualExample file. After that, we will use the following command to execute our C++ code in order to count the instructions. (make sy.test).

Moreover, we will then execute our instruction c++ code on the a.out file that we got from the factorial using the command: (../../../pin -t obj-intel64/sy.so -- ./a.out)

Using this command will perform the count of the instructions and then it will give us a file that has the results of our calculations called results.out.



**Preview of the results.out file.**

**First task**

In first task, we find the number of the total instruction we have.

We start by defining total variable that will count each time it counts an instruction.

VOID findtotalcount() { total++; }

Our findtotalcount method will be called by this method.

INS\_InsertCall(ins, IPOINT\_BEFORE, (AFUNPTR)findtotalcount, IARG\_END);

**Second task**

1. We define if statements in order to find the requested instructions such branch instruction, memory instructions, and arithmetic instructions.

if (INS\_IsBranch(ins)) {

INS\_InsertCall(ins, IPOINT\_BEFORE, (AFUNPTR)dobranchcount, IARG\_END);

}

for (UINT32 memOp = 0; memOp < memOperands; memOp++)

{

if (INS\_MemoryOperandIsWritten(ins,memOp) || INS\_MemoryOperandIsRead(ins,memOp)) {

INS\_InsertCall(ins, IPOINT\_BEFORE, (AFUNPTR)domemcount, IARG\_END);

}

}

1. Define the required variables for each type of instructions.

static UINT64 memoryInstcount = 0;

static UINT64 total = 0;

static UINT64 branchInstcount = 0;

static UINT64 arithInstcount = 0;

1. After getting the branch count and memory count, we just subtract them from the total instruction count to get the value of the arithmetic instruction count.

arithInstcount = total - (branchInstcount + memoryInstcount);

We call our variables and print the results

OutFile << "\nTask2 " << endl;

OutFile << "Branch Instructions Count is: " << branchInstcount << endl;

OutFile << "Memory Instructions is: " << memoryInstcount << endl;

OutFile << "Arithmetic and Logic Instructions Count is: " << total - (branchInstcount + memoryInstcount) << endl;

OutFile << "Ratio of branch instructions: " << (double)branchInstcount /(double)total << endl;

OutFile << "Ratio of memory instructions: " << (double)memoryInstcount /(double)total << endl;

OutFile << "Ratio of arithmetic instructions: " << (double)arithInstcount /(double)total << endl;