

Homework #5

cpe 631

Kyle Ray

February 26, 2018

Contents

[Purpose 1](#_Toc507963555)

[Execution 1](#_Toc507963556)

[Problem 1 2](#_Toc507963557)

[Problem 2 2](#_Toc507963558)

[Bonus 5](#_Toc507963559)

[Problem 3 6](#_Toc507963560)

[Appendix 8](#_Toc507963561)

# Purpose

To become familiar with the Intel PIN and learn more about measuring the performance of modern computer systems with specific benchmarks.

# Execution

To run the tests, travel in the working directory for homework 5 (hw5) and go into the SPEC folder, there you will find the 621 and 623 benchmarks that were tested. Inside of each folder will be a test and train input set. I have created runtest\_#.sh scripts for each problem inside the respective test folder. Just run this script from inside this directory to build the pin tool and execute it on the benchmark.

# Problem 1

The inscount0.cpp pin tool will count the total number of instructions executed by the application it is monitoring and write the result to an output file. In this situation the application being monitored is running a SPEC benchmark. The inscount0 PIN tool was ran against the 621\_wrf\_s and 623\_xalanchbmk\_s benchmarks both with the train input set. The data for the experiment can be found in the table below.

**Table 1. Instruction Count for SPEC Benchmarks via Intel PIN Tool**

|  |  |
| --- | --- |
| Benchmark | Total Instructions Executed |
| 621\_wrf\_s train | 583459817340.00 |
| 623\_xalanchbmk\_s train | 257156226730.00 |

# Problem 2

Modify the inscount0.cpp to count the number of basic blocks, number of memory reads, number of memory writes, and the total number of executed instructions. Refer to Figure 1 for the coding modifications to achieve the goal of this problem statement.



**Figure 1. Code Modifications for Problem 2**

This PIN tool works with single-threaded applications, in this case there were two simple applications and two SPEC benchmarks tested. The benchmarks consist of the following:

1. Serial Matrix Multiplication with (256x256) input
2. Accumulating Array Elements with (1x256) input
3. SPEC Benchmark 621.wrf\_s with train input
4. SPEC Benchmark 623.xalanchbmk\_s with train input

The results from the PIN tool executions can be found in the table below

**Table 2. Modified PIN Tool Results for SPEC and Simple Benchmarks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Benchmark | Basic Blocks | Memory Reads | Memory Writes | Total Instructions |
| mm\_mult\_serial 256x256 | 3104.00 | 3816.00 | 2559.00 | 482639296.00 |
| accumulate array 1x256 | 4619.00 | 5634.00 | 4632.00 | 1421192.00 |
| 621\_wrf\_s train | 104862.00 | 260269.00 | 251753.00 | 583459817334.00 |
| 623\_xalanchbmk\_s train | 37987.00 | 53881.00 | 39434.00 | 257156226729.00 |

# Bonus

Count and print the statistics for each thread in a separate column for a multithreaded program. This will require modifying the pin tool from problem 2 and making it work with multi-threaded applications. Note: that the pin tool will not generally work with pthreads or other multithreaded paradigms, there is a pin tool THREAD API that is provided, and the code written for this problem should make use of that.

I decided to experiment with more of the PIN API and this iteration became a hybrid using TRACE and INSTRUCTION instrumentation. The TRACE instrumentation was used to get an overall count of the basic blocks in the program and the INSTRUCTION instrumentation was used to look at every instruction and determine if it was reading or writing to memory.

Referring to the figures below it seems that the work was distributed amongst the threads considerably even; although, as is the case in most threaded applications thread 0, or the master thread, has a little bit more work than the rest.

Total number of threads = 4

Thread 0 Thread 1 Thread 2 Thread 3

Basic Blocks: 8423666 7953861 7928905 7927815

Memory Reads: 125063683 122049867 122037603 122036839

Memory Writes: 13353797 11695056 11695170 11695057

Total Instructions: 207567601 198639951 198565499 198561796

**Figure 2. Matrix Multiplication OMP 4 Threads PIN Profiler Output**

Total number of threads = 4

Thread 0 Thread 1 Thread 2 Thread 3

Basic Blocks: 3005730431 2993450999 2988057751 2954828943

Memory Reads: 6362837709 6131876411 6123255612 5710073130

Memory Writes: 1764395012 1581504720 1579269151 1470955178

Total Instructions: 23639249386 22870330015 22997384192 21724022559

**Figure 3. 621.wrf\_s Test Input OMP 4 Threads PIN Profiler Output**

Total number of threads = 4

Thread 0 Thread 1 Thread 2 Thread 3

Basic Blocks: 2414232269 2296752380 2399369944 2316503255

Memory Reads: 5054907334 4831556135 4881277019 4529772868

Memory Writes: 1385037562 1261135943 1260641327 1177204286

Total Instructions: 18884161621 17953346177 18387924631 17230065686

**Figure 4. 621.wrf\_s Train Input OMP 4 Threads PIN Profiler Output**

# Problem 3

Design and implement a PIN instrumentation tool for profiling dynamic basic blocks (or streams) in a program. A dynamic basic block is defined as a sequential run of instructions that starts with an instruction that is a target of a taken branch and ends with the first taken branch in a sequence.

The dynamic profiler works by instrumenting every instruction in the application. It will keep a stream table or map, with the stream index of execution as the key, of every stream that is executed. An entry in the map contains the following:



Refer to the figures below for output of the dynamic block profiler.



**Figure 5. Example Stream Output for MM Multiplication with PIN Dynamic Profiler**



**Figure 6. Matrix Multiplication with PIN Dynamic Profiler**

****

**Figure 7 Accumulate Array 256 with PIN Dynamic Profiler**

****

**Figure 8. 621.wrf\_s Test Input with PIN Dynamic Profiler**

****

**Figure 9. 623.xalanchbmk\_s Test Input with PIN Dynamic Profiler**

# Appendix