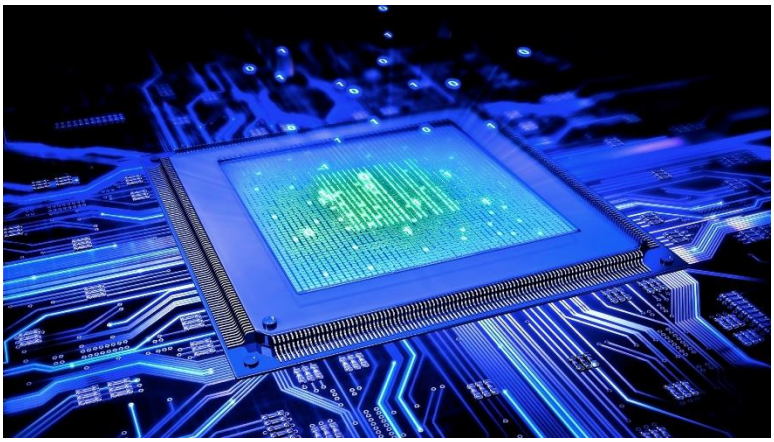

Homework #5

CPE 631

KYLE RAY

February 26, 2018

Contents

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.

```
// This function is called before every instruction is executed
VOID PIN_FAST_ANALYSIS_CALL docount() { ++icount; }
VOID PIN_FAST_ANALYSIS_CALL doCountMemReads() { ++memReadCount; }
VOID PIN_FAST_ANALYSIS_CALL doCountMemWrites() { ++memWriteCount; }
VOID PIN_FAST_ANALYSIS_CALL block_count() { ++blockCount; }

// Pin calls this function every time a new instruction is encountered
VOID Instruction(INS ins, VOID *v)
{
    // Grab the mem operands
    UINT32 memOperands = INS_MemoryOperandCount(ins);

    // Iterate over each memory operand of the instruction
    for (UINT32 memOp = 0; memOp < memOperands; ++memOp)
    {
        // Check for a read
        if (INS_MemoryOperandIsRead(ins, memOp))
        {
            INS_InsertPredicatedCall(
                ins, IPOINT_BEFORE, (AFUNPTR)doCountMemReads,
                IARG_FAST_ANALYSIS_CALL,
                IARG_END);
        }

        // Check for a read
        if (INS_MemoryOperandIsWritten(ins, memOp))
        {
            INS_InsertPredicatedCall(
            ins, IPOINT_BEFORE, (AFUNPTR)doCountMemWrites,
            IARG_FAST_ANALYSIS_CALL,
            IARG_END);
        }
    }

    // Check if instruction is branch
    if (INS_IsBranch(ins))
    {
      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)block_count,
        IARG_FAST_ANALYSIS_CALL, IARG_END);
    }

    // Count the instruction towards the total
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount,
        IARG_FAST_ANALYSIS_CALL, IARG_END);
}

KNOB<string> KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool",
    "o", "inscount_serial.out", "specify output file name");

// This function is called when the application exits
VOID Fini(INT32 code, VOID *v)
{
    // Write to a file since cout and cerr maybe closed by the application
    OutFile.setf(ios::showbase);
    OutFile << "Basic Blocks: " << blockCount << endl;
    OutFile << "Memory Reads: " << memReadCount << endl;
    OutFile << "Memory Writes: " << memWriteCount << endl;
    OutFile << "Total Instructions " << icount << endl;
    OutFile.close();
}
```

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:

```
struct stream_table_entry
{
  ADDRINT sa;                 // stream starting address
  UINT32  sl;                 // stream length
  UINT32  scount;             // stream count -- how many times it has been executed
  UINT32  lscount;            // number of memory referencing instructions
  UINT32  nstream;            // number of unique next streams
  vector<UINT32> nstream_id;  // next stream id (index in the stream table)
  vector<UINT32> ncount;      // counter how many times the next stream is encountered
  stream_table_entry()
  {
    sa = 0x00000000;
    sl = 0;
    scount = 0;
    lscount = 0;
    nstream = 0;
  }
};
```

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

```
1017:  (0x000000301980efdc,    10,     5),  2; { (991, 1) (1018, 1) }
1018:  (0x0000003019c358c8,    18,     3),  1; { (1019, 1) }
1019:  (0x0000003019c358dd,    40,    18),  1; { (1020, 1) }
1020:  (0x0000003019c73a88,     8,     2),  1; { (1021, 1) }
1021:  (0x0000003019c73c2f,     4,     2),  1; { (1022, 1) }
1022:  (0x0000003019c73c88,     4,     2),  3; { (1023, 3) }
1023:  (0x0000003019c73c6e,     5,     2),  3; { (1024, 3) }
1024:  (0x0000003019c73c27,     7,     2),  3; { (1022, 2) (1025, 1) }
1025:  (0x0000003019c73be8,     2,     0),  1; { (1026, 1) }
1026:  (0x0000003019c73c08,    19,    10),  1; { (1027, 1) }
1027:  (0x0000003019c74bc0,     4,     2),  2; { (1028, 1) (1036, 1) }
1028:  (0x0000003019c74b30,    16,     5),  2; { (1029, 1) (1027, 1) }
1029:  (0x0000003019c74b76,    12,     7),  1; { (1030, 1) }
1030:  (0x0000003019c74c50,    11,     8),  1; { (1031, 1) }
1031:  (0x0000003019c74ba5,    34,    19),  1; { (1032, 1) }
1032:  (0x0000003019c736c9,    51,    33),  1; { (1033, 1) }
1033:  (0x0000003019c74c90,     9,     5),  1; { (1034, 1) }
1034:  (0x0000003019c74cc4,     3,     2),  1; { (1027, 1) }
1035:  (0x0000003019c74bc0,     7,     2),  1; { (1028, 1) }
1036:  (0x0000003019cacfa0,     3,     0),  1; { }
```

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

```
Number of unique streams:           1037
Number of program streams executed: 17274371
Number of memory ref instructions:  274611188
Number of instructions executed:    500671308
Max Stream Length:                  423
Average Stream Length:              28
```

Figure 6. Matrix Multiplication with PIN Dynamic Profiler

```
Number of unique streams:           1455
Number of program streams executed: 113350
Number of memory ref instructions:  458596
Number of instructions executed:    1421245
Max Stream Length:                  423
Average Stream Length:              12
```

Figure 7 Accumulate Array 256 with PIN Dynamic Profiler

```
Number of unique streams:           20413
Number of program streams executed: 80967922
Number of memory ref instructions:  228698706
Number of instructions executed:    798801113
Max Stream Length:                  1259
Average Stream Length:              9
```

Figure 8. 621.wrf_s Test Input with PIN Dynamic Profiler

```
Number of unique streams:           3397
Number of program streams executed: 244643
Number of memory ref instructions:  1024232
Number of instructions executed:    2854137
Max Stream Length:                  178
Average Stream Length:              11
```

Figure 9. 623.xalanchbmk_s Test Input with PIN Dynamic Profiler

Appendix