EE382M – 15: Assignment 2

Professor: Lizy K. John
TA: Jee Ho Ryoo
Department of Electrical and Computer Engineering
University of Texas, Austin

Due: 11:59PM September 28, 2014

1. Introduction

The goal of this assignment is to evaluate workloads on real systems using performance monitoring counters (PMCs). You will gain experience working directly with complex vendor provided specifications, in this case, Intel's PMC specification. You will also gain experience using PMCs to evaluate workload performance on real systems.

1.1 Setup and Requirements

- You will need access to a Linux machine with the perf utility installed to complete Section 3. Due to the nature of this lab, your machine should be lightly loaded or complete dedicated to you. This lab was written using Ubuntu 14.04 LTS and perf version 3.13.11.6.

- While we don't think that a difference in perf version or linux distribution will have a significant impact on this lab, we cannot guarantee success with all possible machine combinations. We will be using the above tools as the gold standard. If you have a doubt as to the suitability of your setup, please ask!

- Once you do choose a machine, please use this same machine for all experiments, as we ask for a description of the machine as one of the deliverables.

- Some virtual machines (like VMWare player) can virtualize the CPU performance counters. This should be sufficient for this lab if you choose to go down this route. I do not believe that VirtualBox supports counter virtualization however. If youa re not sure, please consult the TA.

- To install perf on an Ubuntu 14.04 machine, execute the following:
  ```
  sudo apt-get install linux-tools-common
  sudo apt-get install linux-tools-3.13.0-35-generic
  ```
  Different systems will require different packages.

1.2 Deliverables

The following deliverables must be completed for this assignment

- Problems in Section 2
- Problems in Section 3
- Any scripts used to complete the problems

- Description of the profiled machine, detailing
    - CPU [cores/freq/sockets/vendor/model]
    - [L1/L2/L3] [I/D] cache [size/assoc]
    - Main Memory Size
- **HINT: Much can be gleaned from `cat /proc/cpuinfo` and your good friend Google.**

2. Performance Monitoring Counters

2.1 Overview

For this section, you will read and answer questions about the performance monitoring facilities for Intel processors. Our goal in this endeavor is to exercise your ability to find and interpret information in professional documentation.

All major CPU hardware vendors expose performance monitoring counters, mostly for their own debugging and analysis efforts. However, these counters are documented publicly and can be read by anyone with the appropriate hardware. Before we dive too deeply into the high level Linux tools, let us first look at what documentation the CPU vendors provide. It is this documentation that is used by tools such as perf to query the hardware.

For this example, let's look at what documentation Intel offers in its Software Developer's Manual. The appropriate section is as follows:

http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

Don't be daunted by the size of this document, it is very modular. Sections 18 and 19 contains documentation for all publically available performance counters, from the Pentium Pro (P6) to Haswell. For this section, your focus will be much more specific. While the x86 instruction set maintains backwards compatibility across microarchitectures, the performance counters can and do change drastically from one generation to another.

2.2 Problems

1. Intel defines ***architectural performance events*** as hardware performance counters that do not change across microarchitecture generations. Please list all architectural performance events by name.

2. For the following questions, let's assume we are working with an Intel® Core™2 Duo Processor (T7700).
    a. What architecture Performance Monitoring Version does this processor support? What is the underlying microarchitecture according to the manual? (Be careful here!)
    b. I need to count L2 cache loads of dirty cache lines for core 0 and not core 1 with all prefetches. What are the Umask Values and Event Numbers?

c. Let's say I want to count multiply and divide operations for a given workload. Can they both be active at the same time? How can I get around this? (You may not find an answer in the document, so please search Google).
d. I have a very specific need to count the number of L2 dirty cache line evictions triggered by all cores including the contribution of hardware prefetches. What are my Umask Values and Event Number in this case?

3. Using Performance Counters for Analysis

3.1 Overview

In practice, few people use the MSRs (machine status registers) directly for monitoring the counters (but it's still important to know how it's done!). We use higher level interfaces that abstract that icky complexity away from us. For this lab, we will use the Linux perf tool to extract hardware performance counter information. Take a look through this tutorial:

https://perf.wiki.kernel.org/index.php/Tutorial

As you can see, there is a lot more that perf can do besides reading PMCs! But for this assignment, we will only focus on this functionality. Pay particular interest to Section 2 on the wiki. To see what performance counters are available on your machine, type `` `perf list` ``. The performance counters from the CPU are listed as [Hardware Events] or [Hardware cache events]. Here are a few on my machine:

```
L1-dcache-loads                          [Hardware cache event]
L1-dcache-load-misses                    [Hardware cache event]
L1-dcache-stores                         [Hardware cache event]
L1-dcache-store-misses                   [Hardware cache event]
```

If I want to see the number of branches and the number of cycles executed during a program, use the following command:

```
perf stat -e <events> <program>
```

On my machine the following output is produced from the set of default events running a test program:

```
     2,010,911,722 cycles                    #    3.236 GHz
     4,001,043,136 instructions              #    1.99  insns per cycle
     1,000,187,034 branches                  #    1609.276 M/sec
            12,764 branch-misses             #    0.00% of all branches

       0.623228772 seconds time elapsed
```

Using the above tutorial and wiki link, please answer the questions in the following section.

3.2 Problems

1. I have a program "foo" that is multithreaded and contains both user space and kernel protected code that I am interested in profiling. Please provide the command line(s) **AND** identify any post-processing you would need to:
    a. Collect user-space IPC aggregated across all CPUs.
    b. Collect kernel branch miss rate on core 2.
    c. Collect system wide dTLB MPKI.

2. One nice thing about perf is that it allows you program the MSRs directly using what we learned in Section 2.0. This allows you to get at specific features hardware counters not exposed by perf. For this problem, identify a hardware counter on your microarchitecture that is not exposed by perf. Show how you can use perf's raw hardware descriptors to monitor that counter on your system. Give a sample command.

3. Try running perf with as many counters as you can. You will see statistics related to multiplexing.
    a. How does perf deal with multiplexing the performance counters? Can you think of a better way? (it's ok to look in the literature!)
    b. How many generic and fixed counters does your microarchitecture have? Cite your source for this.
    c. What are the rules concerning event assignment to the generic and fixed counters for your microprocessor?

4. In the lab handouts, you will find a program, little_program1, which is compiled from around 10 lines of C. Use perf to analyze the microarchitectural performance of this program. What can you say about this program? Can you make a reasonable guess as to what it is doing? Please include all relevant data to support your argument.
   NOTE: Yes, I know you can disassemble it and figure this out. Please don't.

5. Compiler optimizations can have a huge impact on program performance. In the lab packet, you will see a simple implementation of the inner product written in C (inner_product.c). Compile this code with –O0 and –O2 optimization flags.
    a. How do the branch behavior, cache behavior, IPC, and floating point ops change between the two optimization levels?
    b. Generate assembly with these optimization flags. What are the major differences that you see in code structure? Do these observations agree with your counter data?

6. In the lab handouts you were provided with a two sample binaries and inputs from the SPEC CPU2006 suite, named program1 and program2. I want you to analyze the microarchitectural performance of these programs. You can run them simply by typing `` `./program[1/2] program[1/2].in` ``.

   NOTE: Try to minimize multiplexing and scaling of the counters by perf. You will need to perform multiple runs.

a) Please collect and turn the following aggregate statistics for these programs. Show the perf commands issued and any post-processing needed to derive these.

   Instructions, Cycles, CPI, L1 Miss Rate, LLC MPKI, Branch Miss Rate

b) Please collect micro op mixes for these programs. Specifically, I want % Load, % Store, % Branches, %Floating Point, % Other.

c) Please create a graphs showing how the following statistics change over the execution of the programs. The x-access should be time, and the y-axis is the statistic. Sample the program counters every 1 second. If you write a script, please include it in the submission along with the raw data and graphs. How do the aggregate statistics for CPI, cache miss rates, and branch miss rates compare with the time-variant statistics?

   CPI, Load ops, Store ops, Branch ops, L1 Miss Rate, LLC MPKI, Branch Miss Rate

d) Using all of these statistics, analyze the performance of these two programs. What can you say about their characteristics?

3.3 Hints and Tips

1. As was previously mentioned, performance counters are attached to the microprocessor generation. It is possible (and highly probable) that you and your friends will have different performance counters on your machines. Don't worry, this assignment should only need the basics that should be available on all vendors/models. But if this assumption proves false, please ask!

2. As you have no doubt noticed, there are many different types of cycles that can be counted. For this assignment, please always use the generic "cycles/cpu-cycles" counter in perf, not "ref-cycles".

4. Submission Instructions

4.1 Submission instructions

Your assignment must be compressed in tar.gz format. Please use the following command to tar your submission:

```
% tar –zcvf <your last name>-a2.tar.gz <your working directory>
```

There will be a submission link available on the course website, so please submit your tarball by deadline. Your submission directory must have the same directory structure as the template directory structure. Please remove all your temporary and input trace files. You are only required to submit your code and makefile. Make sure that the name of your working directory is named <your last name>-a2.