Comp 528 Computer Systems Performance Analysis
Spring 2005

Homework Assignment 1
Due: February 10

Performance Measurement with Hardware Counters

For this assignment, you will use `191.fma3d`, a Fortran 90 code from the SPEC CFP2000 benchmark suite, as the basis for experiments on the Rice Terascale cluster (rtc.rice.edu). *Be sure to log in using your class account when performing experiments for assignments.* See the Appendix at the end of this assignment for more information about where to get the code, how to find other information you will need to know, and what *different* compiler options to use when compiling applications for study with HPCToolkit and `gprof`.

1. **Sampling rate.** The sampling rate for sampling-based profilers can affect the accuracy of performance measurements. Sampling too frequently can perturb the execution you are examining; sampling too infrequently can lead to imprecise results. When using hardware performance counters, a sampling rate should be carefully chosen after considering the frequency of the events you aim to measure. Investigate the impact of the sampling rate on the accuracy of profiles collected using `hpcrun` out of HPCToolkit. Consider sampling using hardware counters to measure cycles (event PAPI_TOT_CYC) and primary cache misses (event PAPI_L1_DCM). Is the default sampling rate used by `hpcrun` for these events appropriate? If not, what is an appropriate sampling frequency for each of these events. Justify your answers with data.

   Be sure to exploit your findings in this exercise as you tackle the rest of the exercises in this assignment.

2. **Profiling with hardware counters vs. interval timer.** Traditionally, Unix profilers have used the interval timer[1] to drive statistical sampling instead of hardware performance counters. Use `hpcrun` to collect profiles of `fma3d` using each of wall clock time (event WALL_CLK) and the hardware cycle counter (event PAPI_TOT_CYC) as the basis for sampling. (Note: samples collected based on event WALL_CLK are reported in milliseconds whereas samples based on event PAPI_TOT_CYC are reported in cycles.) What did you find? `hpcrun` provides process-based sampling. Suppose it used node-wide sampling instead. Discuss the strengths and weaknesses of each approach when used on a dual-processor node dedicated exclusively to your application.

3. **Evaluating the accuracy of `gprof`'s call-graph profiles.** Call graph profiling, as supported by `gprof`, is a technique commonly used for analyzing application performance. Call graph profiles report the amount of time spent in each function and

[1] See "man setitimer" on the RTC for information about the interval timer.

attribute the time to calling contexts as well. An interesting question is the extent to which collecting information about calling context distorts application performance. To answer this question, you will compare performance measurements gathered using call-graph profiling and those gathered with `hpcrun`, a flat profiler.

Measure an execution of `fma3d` with `hpcrun` and collect samples based on wall clock time or cycles. Run `fma3d-gprof`, which has support for `gprof` profiling compiled in, and it will produce a *gmon.out* file for `gprof`. Use `hpcprof` to analyze the profile data collected with `hpcrun` and `gprof` to analyze the data in gmon.out. Consider the set of procedures that account for 95% of the execution time in each of the programs. Is the set of procedures the same for the two executions? For this set of procedures, are the differences times spent in corresponding procedures in the two executions substantially different? Correlate the measurements you collect with the different profilers. Interpret your findings.

4. **Understanding memory hierarchy utilization.** A strength of hardware performance counters is that they can measure phenomena that cannot be measured directly otherwise. Use `hpcrun` to gather data to enable you to compute the bandwidth utilized by `fma3d` at each level of the memory hierarchy: between primary (L1) data cache and the processor, between L1 and secondary (L2) cache, between L2 and tertiary (L3) cache, and finally between L3 cache and main memory. (To minimize the number of experiments you run, you will undoubtedly want to use `hpcrun`'s ability to collect multiple events in a single execution.) Look up information about the processors used in the RTC nodes. What fraction of available bandwidth is used by the application at each level of the hierarchy? Does the application appear bandwidth bound at any level of the hierarchy? Are there substantially different bandwidth rates used by the three most time consuming routines?

   You will probably want to use `hpcview`*'s support for computing derived metrics to answer these questions. See the HPCToolkit SC04 tutorial slides for how to compute derived metrics.*

Appendix

Below are several useful facts you will need to know to complete this assignment.

- The Rice Terascale Cluster is based on 900MHz Itanium2 processors with 1.5MB L3 cache. The processor's cycle time can be computed from its clock rate.
- You can find the SPEC `191.fma3d` code on the RTC in directory /users/comp528/assignment1/191.fma3d/src
- Input data for the code can be found in directory /users/comp528/assignment1/191.fma3d/run.
- A Makefile in /users/comp528/assignment1/191.fma3d/src will build two executables: `fma3d` for use with HPCToolkit or unmonitored runs, and `fma3d-gprof`, which will collect call graph profile data.
- When you run the `fma3d` or `fma3d-gprof` programs, launch them while in the "run" directory. The run directory contains input files the application expects to find in the current working directory.
- See /users/comp528/assignment1/README for information about using HPCToolkit, including how to configure your paths and where to find documentation.
- Running `hpcrun`, `hpcquick` and `hpcview` without any arguments will cause each tool to print information about its options and proper usage. `hpcrun -L` will print information about all of the hardware counter events it can use to collect data.
- You can find information about `gprof` using "man gprof". You will not need to use any options with gprof for this assignment. The command "gprof *your-executable* gmon.out" will suffice.
- Executions of the `fma3d` and `fma3d-gprof` codes take 4-5 minutes.