ECE571: Advanced Microprocessor Design – Homework 2

Due: Thursday 14 February 2013, 12:30PM

1. Background

   - For this assignment, use the ARM pandaboard machine just as in HW1.
     `ssh username@vincent-weaver-1.umelst.maine.edu`

   - We will use a benchmark from the MiBench suite:
     - MiBench is a free, embedded, benchmark suite released in 2000. More information, including a paper describing the benchmarks, can be found at the MiBench website:
       `http://www.eecs.umich.edu/mibench/`
     - For this assignment we will use the `susan.smoothing` benchmark with the large input set. This takes an image file and smooths the output. I've provided pre-compiled versions of the benchmark in the `/ece571/` directory on the pandaboard. (susan.static was compiled with gcc 4.6.3 and the `-static -O4` options)
   - Create a document that contains the data described in the Analysis sections below. A .pdf or .txt file is preferred but I can accept MS Office format if necessary.

2. Obtaining Aggregate Event Counts

   - Native Counts
     - Gather timing results for the susan benchmark on the pandaboard using the `time` tool:
       `time /ece571/susan.static /ece571/input_large.pgm output.pgm -s`
     - Run the benchmark 5 times and note the user times (as reported by `time`) and calculate the average user time taken.
     - Hint: you can usually use the keyboard up-arrow to repeat the last Linux shell command (so you don't have to type that long command multiple times).
   - perf tool
     - Gather timing results using perf:
       `time perf stat /ece571/susan.static /ece571/input_large.pgm output.pgm -s`
     - Run the benchmark 5 times and note the user time and total instructions, and calculate the average user time and average total instructions.
   - Valgrind DBI tool
     - Use the Valgrind exp-bbv tool to count instructions (command should be all on one line):
       `time valgrind --tool=exp-bbv --instr-count-only=yes /ece571/susan.static /ece571/input_large.pgm output.pgm -s`
     - Run the benchmark 5 times and note the user time and total instructions, and calculate the average user time and average total instructions.
   - PAPI
     - The `/ece571/susan.papi` file has been instrumented to setup and start the PAPI_TOT_INS event at the beginning of `main()` and to stop, read, and print the value at the end of `main()`.

- Run:
  `time /ece571/susan.papi /ece571/input_large.pgm output.pgm -s`
- Run the benchmark 5 times, reporting user time, total instructions, and the average user time and average total instructions.

- gem5 simulator
  - The gem5 simulator (http://www.m5sim.org/Main_Page) is a "cycle-accurate" simulator that can simulate ARM processors.
    The simulator is too resource-hungry to run on the pandaboard so I've run it on a different machine and provided the results for you.
    The results of susan.static being simulated with the fastest/least detailed simulation options on a 1.GHz Sandybridge Processor can be found here (also linked to on the HW assignment page):
    `http://www.eece.maine.edu/~vweaver/classes/ece571_2013s/gem5_results.txt`
  - Look at the above simulation log file and note the user time it took as well as the total instructions (found on the line `system.cpu.committedInst`)

- Analysis
  (a) Make a summary table of your results, containing four columns:
    i. the measurement methodology,
    ii. the average time for the measurement runs,
    iii. the slowdown compared to the native run,
    iv. and the average number of instructions measured (if applicable).
  (b) Questions to Answer
    i. Do the total instructions from each methodology match? Which do you think is most "correct"? Why?
    ii. Why do we use a statically-linked binary for these experiments?
    iii. How does the PAPI methodology differ from the other ones used, and how might this affect the results?

3. Obtaining Sampled Performance Results

- Native
  - Reuse the results from the previous section.

- gprof
  - The file `/ece571/susan.gprof` was compiled with the `-pg` option which enables profiling. When running this executable the file `gmon.out` is created which contains the profile data.
  - Run the following (once is fine) and note the user time:
    `time /ece571/susan.gprof /ece571/input_large.pgm output.pgm -s`
  - Run: `gprof /ece571/susan.gprof` to obtain the profile summary.
  - Report the top 3 routines listed.

- Valgrind Callgrind
  - The Valgrind tool Callgrind can generate sampled call results. It can also optionally run cache and branch simulators; we will not use this functionality.

  - Run the following and note the user time:
    `time valgrind --tool=callgrind /ece571/susan.static /ece571/input_large.pgm output.pgm -s`
  - A callgrind.PID file is generated
  - Use the following to print a summary: `callgrind_annotate --threshold=100`
  - Report the top 3 routines listed.
- perf
  - Run the following and note the user time:
    `time perf record /ece571/susan.static /ece571/input_large.pgm output.pgm -s`
  - This creates a perf.data file
  - Run `perf report`. Record the top 3 results. Q quits.
  - Run `perf annotate`.
    Note the instruction that took the most of the time.
- Analysis
  (a) Make a summary table of your results, containing three columns:
    i. The methodology used,
    ii. the user time,
    iii. and slowdown factor compared to the native run.
  (b) For each methodology where the top 3 routines were obtained, list them.
  (c) Answer the following questions:
    i. Did the top 3 routines from each methodology match up? Our benchmark is short-running enough that the results are not as good as they could be.
    ii. For the `perf annotate` results, which instruction is listed as using the most CPU time? Is this believable? Do you think the result is affected by skid?
    iii. Which tool did you find easiest to use? Why?

4. Submitting your work.

- Create the file as described in the Background and Analysis sections.
- Please make sure your name appears in the document.
- e-mail the file to me by the homework deadline.