EEE4120F

High Performance Embedded Systems

Lecture 5:

Performance Benchmarking & Wall Clock Timing

Lecturer:
Simon Winberg

Planned as short recorded lecture

Outline for Lecture

- Towards performance benchmarking
- Simple benchmarking techniques

Performance Benchmarking

EEE4120F HPES

Performance Benchmarking

- "Don't loose sight of the forest for the trees…"
- Generally, the main objective is to make the system faster, use less power, use less resources…
- Most code doesn't need to be parallel.
- Important questions are…

Important questions

- Should you bother to design a parallel algorithm?
- Is your parallel solution better than a simpler approach, especially if that approach is easier to read and share?
- Major telling factor is:

Real-time performance measure

Or “wall clock time”

Benchmarking

- Process of measuring performance of a digital system
- A program (or systematic approach) that Quantitatively evaluates performance, cost and computer hardware and software resources (among other things) of a computing solution
- **Benchmark suites** – sets of benchmark programs designed to get a comprehensive view of the performance of a computer system for executing a variety of representative processing operations.
- Suitable benchmark
  - A meaningful representation of what the system can do
  - Helps select of an effective system
  - Indicates a measure of what one system can do compared to other options

Wall clock time

- Generally the most accurate: use a built-in timer, which is directly related to real time (e.g., if the timer measures 1s, then 1s elapsed in the real world)
- Technique:

See file:
Cycle.h
Cycles.c

Cycle.h
cycles.c

```
unsigned long long start; // store start time
unsigned long long end;   // store end time
start = read_the_timer(); // start timer / tic
 DO PROCESSNG
end = read_the_timer();   // end timer / toc
.. Output the time measurement (end-start), or save
it to an array if printing will interfere with the
times. Note: to avoid overflow, used unsigned vars.
```

StdC: gettimeofday

- **gettimeofday**
  - Very portable, part of the StdC library
  - Should be available on any Linux system
  - Returns time in seconds and microseconds since midnight 1 Jan 1970
  - Uses **struct timeval** comprising
    - tv_sec : number of seconds
    - tv_usec : number of microseconds*
  - Converting to microseconds will use huge numbers, rather work on differences

**** Word of caution: some implementations always return 0 for the usec field!!***
On Cygwin, the resolution is only in milliseconds, so tv_usec in multiples of 1000.
Not provided in DevC++.

See timing.c code file

gettimeofday example

```
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
struct timeval start_time, end_time; // variables to hold start and end time

int main()  {
   int tot_usecs;
   int i,j,sum=0;
   gettimeofday(&start_time, (struct timezone*)0); // starting timestamp

   /*   do some work */
   for (i=0; i<10000; i++)
      for (j=0; j<i; j++) sum += i*j;

   gettimeofday(&end_time, (struct timezone*)0);  // ending timestamp

   tot_usecs = (end_time.tv_sec-start_time.tv_sec) * 1000000 +
                 (end_time.tv_usec-start_time.tv_usec);
   printf("Total time: %d usec.\n", tot_usecs);
}
```

timing.c

What is wrong about using only wall clock time?

- It can provide a false impression of how effective your solution is – at least doesn’t give a ‘full picture’ …
  - Typically do tests after the system has ‘warmed up’* (cache loaded) by running the same data multiple times
  - May show the solution is quicker… but at what costs? e.g.:
    - Speed improved but accuracy sacrificed?
    - Development effort vs. execution speed improvement?
    - Resource costs for upgrading vs. costs saved by remaining with the old version?
    - Power usage? Does the new solution need more power (per execution, also on average including idle time)
    - Maintainability? (e.g. is the new version more complex?)
    - Environment impact? (Does the upgrade result in waste that could be environmentally detrimental)

* But this can be a very false impression too, e.g. cache etc pre-set with needed data.

Benchmarking: what to test

- What can be benchmarked? For DSP and HPC…
- Compiler
  - Converts High Level Language to Assembly language thus we benchmark compiler efficiency, such as how efficient is the generated assembly code?
- The Processor
  - Code in hand-crafted/inspected assembly (to make comparisons fair)
- Operating System
  - Interrupt latencies, overhead of operating system calls, limits on devices, kernel size, availability of services and facilities such as support for virtual memory and paged memory.
- Platform
  - Scalability of memory. Peripheral limits. Interfaces supported. Power use. Power saving features. OS's supported.
- Applications (e.g. representative operations for certain application domains – think 'DWARFS' as in Berkeley paper)

Next Lecture …

- We get more into depth of
  - Metrics for performance
  - Some specialized concepts (e.g. the ‘ACPI’ measure for a processor core)
  - Methods to summarising performance (commonly seen in performance reports)
  - SWAP
  - Profiling code designs*

* Only a brief flavour of profiling techniques, would need to be a course on its own to do properly.

closing remarks & reminders…

No quiz next week

Have a look at:

Valgrind About Page

Read About page for Valgrind (very useful): https://valgrind.org/info/about.html

End of Lecture

Disclaimers and copyright/licensing details

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particulate want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

Image sources:

Wikipedia (open commons)
https://www.vectorportal.com
http://www.flickr.com
http://pixabay.com/
Forrest of trees: Wikipedia (open commons)