Measuring Computer Performance

CS 471/Math 471
Brian T. Smith, UNM
Andrew C. Pineda, UNM

Computer Performance

- Machines are used for many different purposes and have very different architectures
  - number crunching, visualization, data processing, …
- How do we tell whether we have the correct machine for the job?
  - the application is scientific computing
  - the focus on number crunching and visualization
  - the emphasis is high performance
- Review Chapter 11 of Intro. To HP Sc. Comp.

Measuring Performance

- How do we measure the performance?
  - Measure basic operations
    - how long does it take the machine to multiply?
  - Representative applications
    - benchmarks, kernel codes (netlib benchmarks)
  - Your application codes
    - the measurements of the kernels and representative codes permit the modeling of the performance of your application codes

Why Do You Want Go To This Effort

- Buying your next machine
  - deciding what you can believe from the vendor's propaganda
- Understanding why your code performs the way it does
- Understanding how to tune your code's performance -- tuning or a new algorithm

Performance Measurement -- An Art Or A Science

- More like an art than a science
- Many factors effect performance
  - many factors are unknown
    - what did the compiler do to my code?
    - what else is running on the machine and how does that effect my performance
- The goal should be to bound the performance, not to determine it exactly (20% rule)
- What 10% of the code takes 90% of the time?

Motivation And Outline

- Select the first measurement approach to study
  - elementary operations
  - demonstrates many of the difficulties in performance measurement
  - topics:
    - what effects performance?
    - how to measure CPU performance
    - clock granularity
    - removing overhead (necessary for elementary operations)
    - compiler optimizations

Factors That Effect CPU Performance

- CPU execution time of:
  - – register operations
  - – integer operations
  - – floating point operations
  - – string operations
  - – memory
    - » in cache (level 1 and level 2)
    - » in main memory
  - – systems and file service
  - – input/output
  - – communication, especially parallel machines

Performance -- What Units?

- – Milliseconds -- 1/1000-th of a second
  - – typical disk access time
- – Microseconds -- 1/1,000,000-th of a second
  - – CPUs of the 60s to mid 80s
- – Nanoseconds -- 1/1,000,000,000-th of a second
  - – CPUs of 80s to 90s
- – Picoseconds -- 1/1,000,000,000,000-th of a second
  - – transistor switching times, CPUs of the future?

Clock Rates

- – Hertz
  - • cycle per second
    - – PNM AC power supply
- – Megahertz
  - • 1,000,000 cycles per second
    - – current PC CPU chips (200 MHz Pentium)
- – Gigahertz
  - • 1,000,000,000 cycles per second
    - – future (??) CPU chips, a few GHz

Unix System Timing Commands

- The **time** command

  time <*your_command*>

  - • returns

    5.3u 0.5s 0:31 18% 0+1124k 97+18io 129pf+0w

  - – try man time
    - • user-time the amount of time the CPU is busy with <*your_command*> (5.3 seconds)
    - • system-time the amount of time the system is active with <*your_command* (0.5 seconds)

Unix Timing Command Cont'd

- • Elapsed-time the wall-clock time taken from command initiate to command completion (0 mins and 31 secs)
- • Utilization ratio of user-time to elapsed time (18%)

Unix Date Command

- The **date** command

  date

  - – returns the current time and date (correct to the current second). The form is shown below:

```
Tue Sep 30 13:52:28 MDT 1997
```

Fortran Date and Time Procedures

– Fortran 90 uses:
  - SYSTEM_CLOCK(CNT, CNT_RATE, CNT_MAX)
    - » CNT returns an integer counter giving the number of clock ticks from some beginning time
    - » CNT_RATE returns the number of clock ticks per second
    - » CNT_MAX returns the maximum value of CNT
  - – DATE_AND_TIME(DATE, TIME)
    - » DATE returns a character string of length 8of the form CCYYMMDD
    - » TIME returns a character string of length 10 of the form hhmmss.ddd

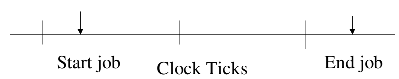
Elapsed Timers For Programs

- Fortran 95 has a procedure CPU_TIME
- There is a POSIX standard timer which has been included in ~acpineda/Lab3/utilities
- There are many vendor-specific timers
  - ETIME -- DEC systems
  - SECONDS -- Cray systems
  - MCLOCK -- IBM AIX systems
    - – returns an integer counter measuring 1/100-th second

Determining Clock Granularity

- The integer function mclock returns an integer where each unit measures 1/100-th of a second -- what is the smallest interval you can measure? (not necessarily 1/100)
- Formats and units vary from vendor to vendor -- seconds to 1/60-th second to 1/256-th second to milliseconds to microseconds to nanoseconds

Granularity Continued




```
do i = 1, 100
  call  mytime( t_start ); sum = i
  do j = 1, 10000
    sum = sum + i
   enddo
   call  mytime( t_end)
   print *, t_end - t_start, sum
enddo
```


Removing The Loop Overhead

- There is overhead related to establishing the loops of repeated code.
  - – Part of that overhead is constant and part of it is linear in cost with the number of repetitions
  - – We want to get rid of the constant time part
  - – The linear part is considered part of the operation time

Removing The Constant Overhead

– Set up the computation as a number of repetitions, say R1, in a first loop of NREP repetitions

– Set up a second computation with a different number of internal repetitions, say R2, in a second loop of NREP repetitions

– Subtract the two elapsed times and divide by the number of repetitions NREP*(R1-R2)

– What values for R1 and R2 -- selected 16 and 8

Form Of Repetitions

- For the addition example, use

  x = x + y; x = x + y; … x = x + y   ! 16 times, NREP reps
  x = x + y; x = x + y; … x = x + y   ! 8 times, NREP reps

  – in the body of the code of do I = 1, NREP loops

  – the operation times then are measured as:

    OPTIME = ((T2-T1) - (T3-T2))/(NREP*(R1-R2))

    - where T1 (T2) is the time before first (second) loop,
    - and T2(T3) is the time at the end of the first (second) loop

Compiler Issues

- The numbers 8 and 16 have to be large enough that optimizer does not replace the repetitions by x = x + 8*y or x = x + 16*y
- Must make sure that there are data dependencies (e.g. x = x …) in the NREP loops so that the optimizer does not remove the loop with direct code (in general, this is not possible).

Granularity Continued

- Obtain results such as:

  0.0
  0.0
  0.01
  0.0
  0.0
  0.0
  0.0
  0.01
  0.0

Laboratory 3 Instructions

- Copy directory ~acpineda/CS471/LabManual/Lab3 to your current directory
  – study the **make** file Makefile
  – change f77 to xlf
  – set the BIN macro to point to your copy of the utilities directory

    BIN = ./utilities

  – modify the make file to create mytime.o in directory utilities
- xlf compiler has three optimization levels
  – nothing,, -O, -O2 (the same as -O), -O3