Performance and Code Tuning Overview

CPSC 315 – Programming Studio

adapted from John Keyser's 315 slides

Is Performance Important?

- Performance tends to improve with time
  - Hardware Improves (SW tweak might not last?)
- Other things can be more important
  - Accuracy
  - Robustness
  - Code Readability
- Worrying about it can cause problems
  - "More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity." – *William A. Wulf*

Performance Increases without Code Tuning

- Lower your Standards/Requirements
  - Asking for more than is needed leads to trouble
  - Example: Return in 1 second
    - Always?
    - On Average?
    - 99% of the time?

Performance Increases without Code Tuning

- Lower your Standards/Requirements
- High Level Design
  - The overall program structure can play a huge role

Performance Increases without Code Tuning

- Lower your Standards/Requirements
- High Level Design
- Class/Routine Design
  - Algorithms used can have real differences
  - Can have largest effect, especially asymptotically

Performance Increases without Code Tuning

- Lower your Standards/Requirements
- High Level Design
- Class/Routine Design
- Interactions with Operating System
  - Hidden OS calls within libraries – their performance affects overall code

Performance Increases without Code Tuning

- Lower your Standards/Requirements
- High Level Design
- Class/Routine Design
- Interactions with Operating System
- Upgrade Hardware
  - Straightforward, if possible…

Performance Increases without Code Tuning

- Lower your Standards/Requirements
- High Level Design
- Class/Routine Design
- Interactions with Operating System
- Upgrade Hardware
- Compiler Optimizations
  - “Automatic” optimization,
  - Getting better and better, though not perfect
  - Different compilers work better/worse

Code Profiling

- Determine where code is spending time
  - No sense in optimizing where no time is spent
- Provide basis for measurement
  - Determine whether “improvement” really improved anything
- Need to take precise measurements

Profiling Techniques

- Profiler – compile with profiling options, and run through profiler
  - Gets list of functions/routines, and amount of time spent in each
- Use system timer
  - Less ideal
  - Might need test harness for functions
- Use system-supported real time
  - Only slightly better than wristwatch…
- Graph results for understanding
  - Multiple profile results: see how profile changes for different input types

What Is Tuning?

- Making small-scale adjustments to correct code in order to improve performance
  - After code is written and working
- Affects only small-scale: a few lines, or at most one routine
  - Examples: adjusting details of loops, expressions
- Code tuning can *sometimes* improve code efficiency *tremendously*

What Tuning is Not

- Reducing lines of code
  - Not an indicator of efficient code
- A guess at what might improve things
  - Know what you’re trying, and *measure* results
- Optimizing as you go
  - Wait until finished, then go back to improve,
  - as optimizing while programming is often a waste
- A “first choice” for improvement
  - Worry about other details/design first

Common Inefficiencies

- Unnecessary I/O operations
  - File access especially slow
- Paging/Memory issues
  - Can vary by system
- System Calls
- Interpreted Languages
  - C/C++/VB tend to be “best”
  - Java about 1.5 times slower
  - PHP/Python about 100 times slower

Operation Costs

- Different operations take different times
  - Integer division longer than other ops
  - Transcendental functions (sin, sqrt, etc.) even longer
  - Knowing this can help when tuning
- Vary by language
  - In C++, private routine calls take about twice the time of an integer op, and in Java about half the time.

Remember

- Code readability/maintainability/etc. is usually more important than efficiency
- Always start with well-written code, and only tune at the end
- Measure!