Performance Analysis and Tuning

CS 501
October 18, 1999

Performance Tuning

- Profiling
- Code tuning
- Options tuning

Performance tuning

- Why tune? Won't processors be twice as fast next year?
  - Customers want it faster NOW
  - Processor speed isn't always the bottleneck
  - Algorithmic improvements can speed up your code far more than 2x
  - Embedded systems

When should I tune?

- Knuth: “Premature optimization is the root of all evil”
- Tune *after* you test and debug your code
  - No point being fast if it’s wrong
  - Bug fixes can de-tune code
  - Tuning often makes code more complicated, making it more difficult to debug
- Maintain/improve performance after you ship
  - Add performance tracking to regression suite to prevent degradation

The Tuning Process

- Don’t tune unless you really have to
- Iterative process
  - Profile, tune, profile, tune, ...
  - This continues until you reach the point of diminishing returns

Profiling

- Profiling will tell you where your program is spending its time
  - A typical program spends 90% of its time in 10% of the code
  - You want to speed up the hot code
- NEVER tune without profiling
  - With complex software difficult to tell where the program spends its time
- Profile with under realistic conditions with realistic data

Profilers

- All profilers are intrusive
  - They perturb the program being profiled
  - Want a profiler that minimizes intrusion

Do-it-yourself profilers

- Add timers to the source code
- Usually want time spent in your process, not real time
  - Unix: use user+sys time not real time
- Use HW counters
  - Often count cycles for all processes on the system, so you need to run on a quiescent machine

Function-level profilers

- Two major types of profilers
  - Instrumentation:
    - Automatically add code to the program to
      - count how often a function is called
      - record how much time is spent in a function
    - Usually requires recompiling or relinking the program
  - Stochastic:
    - Stops program every 10-100ms and check what function the program counter is in
    - Some work out of the box, other require a relink

Instruction-level profilers

- Good for tuning within a function (if you read assembly code)
- Usually stochastic profilers: requires longer run than function level since more fine-grained information
- Shade (Solaris) and Atom (Alpha) interpret the machine code and count the number of times a given instruction is executed
- CPU emulators can tell you anything you want to know (if you have the time)

Code Tuning Techniques

- Change algorithm
  - Most gain, but also most difficult
  - Example: set data structure
    - If sets are dense, bit vectors often better
    - If sets are sparse, hash tables, binary trees, or another sparse data structure might be better

Code Tuning Techniques (2)

- Make hot functions faster
  - Throw more compiler optimizations at it
  - Rewrite in assembly (usually not worth it)
  - Indirect calls → direct calls
    - C++: virtual functions → non-virtual
    - Java: non-static functions → static
      - Probably not worth it with latest JVMs
  - Move infrequently executed code out of the way
  - Eliminate unnecessary I/O, system calls, allocation

Code Tuning Techniques (3)

- Call hot functions less often
  - Cache previously computed values (memoization)
  - Inline: eliminates call overhead and allows compiler to do better job optimizing
    - Inline by hand if the compiler can‘t (ex: indirect calls)
- Java: less synchronization
  - Ex: ”a” + ”b” + ”c”
    - new StringBuffer(”a”).append(”b”).append(”c”).toString()
    - 3 monitorenter instructions: all unnecessary

Options tuning

- Don't optimize a program whose run time doesn't matter
- Start with –O
  - Typical speed up: 2x
  - Even local optimizations help 30-50%
  - YMMV
- Inlining: 10% gain if done blindly, 30% if done with profiling information
- Aliasing options: allow compiler to eliminate more memory references

Options tuning for Java

- Increase max heap size for less frequent GC
- Experiment with vendor-specific options
  - Often many options for improving synchronization performance