Performance Tuning

Andrew Lenharth
CS 378 - UT Austin
9/24/2012

Outline

Day 1 - Theory

- Overview of Tuning
- Profiling Techniques
- HW Profiling Support

Day 2 - Practice

- Making Measurements
- Differential Profiling

Tuning

- Why are we tuning?
- What are we tuning?

Tuning

- Why are we tuning?
  - Application is not “good enough” by some metric
- What are we tuning?

Tuning

- Why are we tuning?
  - Application is not “good enough” by some metric
- What are we tuning?
  - Throughput
  - Response time
  - Memory footprint
  - Perceived performance
  - Power
  - Bandwidth

Monitoring v.s. Tuning

A difference of goal and methodology

Monitoring:
- Low-impact
- Automated
- In the field
- Ensures expected behavior

- Tuning
  - High effort
  - Intentional
  - Tool to drive development

Tuning Methodology

- Define the problem
  - Take measurements to isolate problems
  - Understand the problem and the measurements
  - Attempt a fix
  - Repeat until problem solved
- Repeat until no more problems

Defining the Problem

- The definition comes from the problems being solved and the system, not the code
  - AI + graphics must complete in 1/60 of a second
  - Don't use more than 16kb/s of bandwidth
  - Don't use more than 128MB ram
  - Complete as fast as possible subject to some error constraint
- The problem may be parametrized by input
  - Small v.s. Large
  - Hard v.s. Easy

Taking Measurements: Throughput

- Serial: Speed!
  - Algorithmic efficiency
  - Memory footprint
  - locality
  - How the code interacts with micro-architecture features
  - How the code exploits the cache
- Parallel: Scaling!
  - Communication
  - Contention
  - Sharing
  - Synchronization
  - + serial concerns

Things to Consider Fixing

- Simple:
  - Restructure loops
  - Remove redundancy
- Complex:
  - Bundling messages
  - Remove synchronization
  - Weaken global knowledge
- High level:
  - New algorithm
  - New data-structure
  - Solve a different problem

Types of Fixes (1)

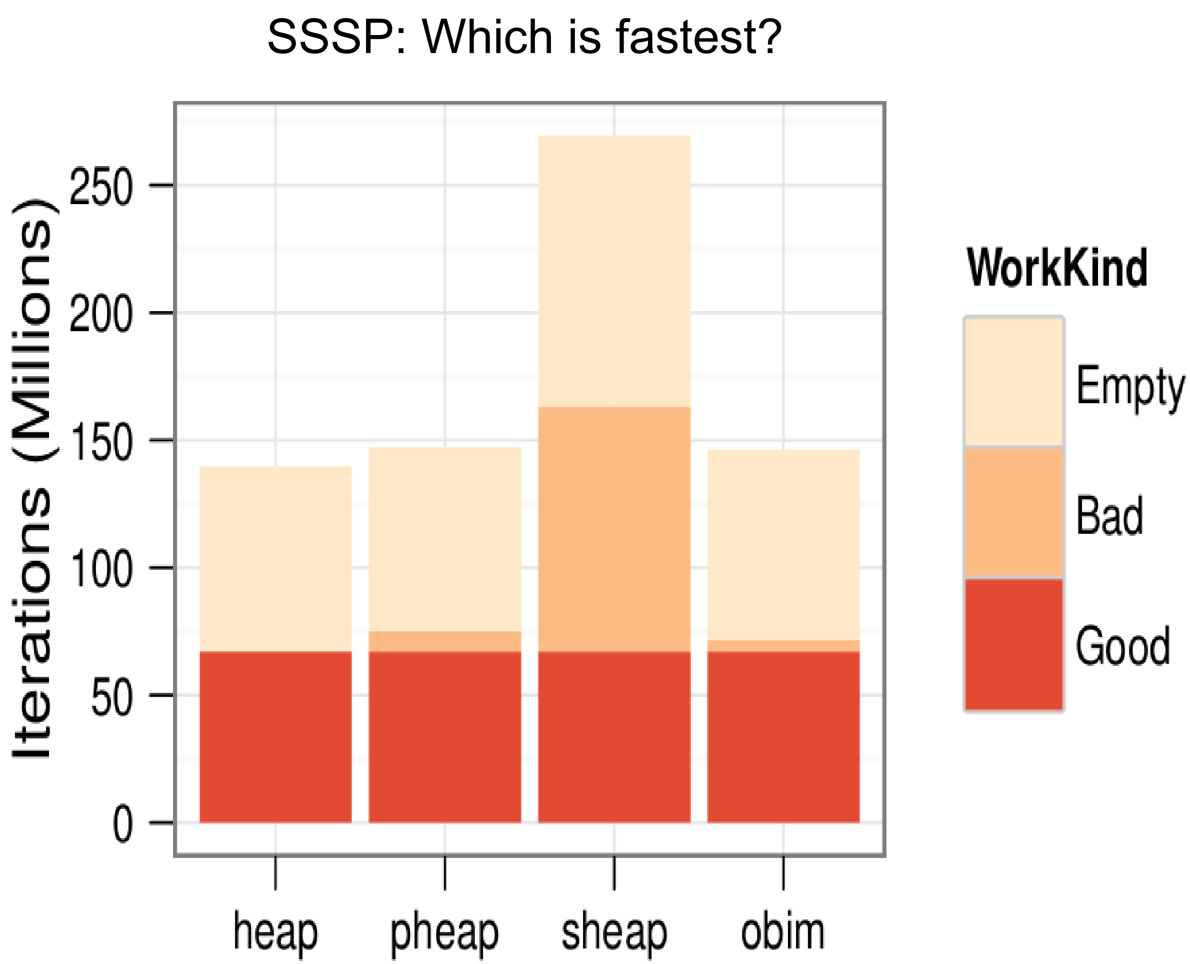
- Micro-optimizations
  - Address micro-architecture bottlenecks
    - Either not provably safe or not profitable for the compiler
  - Need detailed measurements
  - Very labor intensive
  - Not always portable to new architectures
- Compiler-assisting optimizations
  - Replace hard to optimize constructs with easier constructs
  - Assert knowledge the compiler can't generate
  - Requires extensive knowedge of compiler optimizations and analysis

Types of Fixes (2)

- Memory optimizations
  - Improve cache performance
  - Lower bandwidth
  - Reduce working set
- Algorithmic optimizations
  - Use a faster algorithm
  - Use a better data-structure
  - Dynamically select algorithm to runtime data

Unintended Consequences

- Some optimizations have surprising impact
  - Convergence rate
  - Contention
  - Cache usage
  - Total Work
  - Termination



Now Onto Profiling

We will talk about the space of profilers in terms of a set of mostly orthogonal design axis

- What
- How
- When
- Granularity
- Context
- Time
- Coverage
- Presentation

What is measured

- Time
- Control flow
  - Loop counts, Function calls
- Aliasing facts
- Cache stats
- Allocation information
  - Track allocation sites for objects
- Hardware stats

How are measurements taken

- Instrumentation
  - The code is modified to take the measurements
  - When?
    - At compile time
    - At runtime (JIT or dynamic patching)
- Interruption
  - An outside event triggers inspection and measurement
  - Who?
    - Hardware
    - Timer
    - Another thread

When are measurements taken

- All the time
  - Expensive
- Sampling
  - Cheaper
  - When?
    - Nth function call
    - Nth basic block
    - Timer
    - Some property of the hardware

Granularity of what is measured

- Instructions
- Basic blocks
- Line of code
- Function
- Modules

Context sensitivity of measurements

- Behavior of callees depend on the caller
- Flat profiles are cheap
  - Allocate a unique index for each measured location
  - Often track both self and total time
- Context sensitive profiles require more storage and overhead
  - Measurement ID is based on call stack
  - Computing ID requires walking call stack

Time Sensitivity

- Behaviors change over time (phases)
- Time sensitive profiling has similar problem to context-sensitive (but easier)

Presentation

- What models are the data fit to?
- What are the summaries computed?

- Example: gprof
  - Sample execution time
  - Exact call counts
  - Model for assigning execution time to call context
- Example: vtune
  - Multiplexes types of measurements
  - Statistical model to tie measurement to code
  - Skew correction (more on this later)

Example Profilers

- Gprof
  - Samples time and counts calls
  - Statistical model to assign time to callgraph (pseudo-context sensitivity)
  - Compiler instrumented code
- Valgrind – Cachegrind
  - JITs instrumented code
  - counts calls and monitors memory accesses
- Vtune
  - Samples hardware stats, context-free, whole machine, time filtering

A nice compromise

- Interrupt based Sampling
  - No change to code
  - Low impact
- Measure Instructions
  - Higher level entities can be build from summaries
- Entire machine coverage
  - OS and Libraries can be a bottleneck
- Context Sensitivity
- Machine stats
  - Support both highlevel and lowlevel tuning
  - Already collected asynchronously

Open Question?

- Who can sample everything?
  - Including OS and privileged code
- Who can sample at clock cycle boundaries?

Hardware can.

Hardware support for profiling

Major Types of HW Support

- Performance Counters
  - Region granularity
- Event based sampling
  - Imprecise usually
  - Limited Information per run
  - Instruction granularity
- Instruction based sampling
  - Instruction granularity
  - Precise
  - Lots of Information per run
    - Can capture address traces

Performance Counters

- We need to count the events
- HW provides special programmable counters
- Program a counter to count a specific Event
- Software (ring-0) can read and write the counters

Performance Counters - Use

- Very fine-grained detail
  - e.g. Number of times processor stalled because register file couldn't allocate enough registers for all issuable instructions
  - e.g. number of Prefetches generated by the prefetch engine (v.s. Number of prefetches issued (not the same thing!))
- Course-grained regions
  - We can count over a region of code
  - Fairly expensive to read counter, so finer granularity causes increased perturbation

Using Performance Counters

Much like gettimeofday() and printf

Instrument region to read the counter before and after

Print the value

What's wrong with performance counters?

Event Based Sampling

- Let's solve the course-granularity problem
- Extend performance counters with a programmable limit value
- Cause an interrupt when the counter overflows (exceeds limit value)
- That's it (from the hardware side)

EBS - Software

- Interrupt handler in the OS
  - Records PC (and process id) from interrupt
  - Records event type
  - Resets counter
- Produces a log of <PC,event> pairs
- Map PCs back to code
- Now we have a sampled profile of any event we want
  - Which branches have bad miss rates?
  - Where are the high latency memory accesses?

EBS - Inaccuracies

- Interrupt is generated after event
- Pipeline drains instructions
- Skew: latency from PC triggering event to interrupt
- Masking of events by other events
- Events missed during sampling interrupt
- Events miss counted during SMI
- Other platform-specific problems

EBS – In Practice (HW)

- HW has many types of events and sub-events it can monitor
  - Nehaleum ~100 documented
    - Only 7 are defined as stable across future processors
- HW has a few programmable counters
  - Various restrictions on what can be programmed
    - e.g. Nehaleum: 3 fixed, 4 programmable, only 2 can count any event
- Usually fairly flexable overflow programming

EBS – In Practice (SW)

- Usually a flat profile (why?)
- Sampling can be used to collect more events than HW supports simultaneously
  - Rotate which events are currently programmed
- Extremely useful for monitoring OS overhead
- Low overhead
  - Unless long runs
  - Unless short sampling period
- Software usually corrects for skew as best it can

Instruction Based Sampling - Motivation

- Let's eliminate Skew and the need for multiple runs
- Let's also collect more information
  - e.g. Virtual and physical address trace
  - e.g. branch history trace

IBS - Mechanism

- HW selects instruction to monitor and tags it
- Record all events and stats as tagged instructions execute
  - Completion time, execution time, branch stats and address, ld/st stats and addresses, cache stats, latencies, etc
- At retire, record tagged instruction info to a log