Profiling
Andrew Lenharth
UT Austin CS378
Fall 2014

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
- What has higher error?

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 that are 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?