Announcements:

- HW 3 solutions posted (discuss?)
- HW 4: tonight (-ish)

Today:

- Profile
- Program Representation
Timing Experiments: Pitfalls

What are potential issues in timing experiments? (What can you do about them?)

- Noise from users
- Timing noise
  - Know clock sources
  - Knowing your overheads
- TSC
- Wait for the right things
- NUMA (pin mem & execution)
- Frequency scaling: RAPL
Timing Experiments: Pitfalls (part 2)

What are potential issues in timing experiments? (What can you do about them?)

- uninitialized mem.
- alloc memory
- run a stream benchmark
- bandwidth > peak
Outline

Introduction

Machine Abstractions

Performance: Expectation, Experiment, Observation
  Forming Expectations of Performance
  Timing Experiments and Potential Issues
  Profiling and Observable Quantities
  Practical Tools: perf, toplev, likwid

Performance-Oriented Languages and Abstractions

Polyhedral Representation and Transformation

Code Generation and Just-in-Time Compilation
Profiling: Basic Approaches

Measurement of “quantities” relating to performance

- **Exact:** Through binary instrumentation (valgrind/Intel Pin/...)
- **Sampling:** At *some* interval, examine the program state

We will focus on profiling by *sampling*.

Big questions:

- What to measure?
- At what intervals?
Defining Intervals: Performance Counters

A *performance counter* is a counter that increments every time a given *event* occurs.
What events?

- Demo: perf/Using Performance Counters
- see also Intel SDM, Volume 3

Interaction with performance counters:

- Read repeatedly from user code
- Interrupt program execution when a threshold is reached
- Limited resource!
  - Only a few available: 4-8 per core
  - Each can be configured to count one type of event
  - *Idea*: Alternate counter programming at some rate (requires steady-state execution!)
Profiling: What to Measure

- Raw counts are hard to interpret
- Often much more helpful to look at ratios of counts per core/subroutine/loop/…

What ratios should one look at?
Demo: perf/Using Performance Counters
Profiling: Useful Ratios

Basic examples:

- \( \frac{\text{Events in Routine 1}}{\text{Events in Routine 2}} \)
- \( \frac{\text{Events in Line 1}}{\text{Events in Line 2}} \)
- \( \frac{\text{Count of Event 1 in X}}{\text{Count of Event 2 in X}} \)

Architectural examples:

- IPC
- dTLB misses / (hits + misses)
- cache misses / instructions
- \( \frac{\text{rops}}{\text{clock}} \)

Issue with 'instructions' as a metric?

'Instructions ~ work?'
“Top-Down” Performance Analysis

Idea: Account for useful work per available issue slot

What is an issue slot?

[Yasin ‘14]
Issue Slots: Recap
What can happen to an issue slot: at a high level?

[Image of a decision tree]

[Yasin ‘14]
What can happen to an issue slot: in detail?

[Fig. 4](Yasin '14)
Outline

Introduction

Machine Abstractions

Performance: Expectation, Experiment, Observation
  Forming Expectations of Performance
  Timing Experiments and Potential Issues
  Profiling and Observable Quantities
  Practical Tools: perf, toplev, likwid

Performance-Oriented Languages and Abstractions

Polyhedral Representation and Transformation

Code Generation and Just-in-Time Compilation
Demo: Performance Counters

Show the rest of:

Demo: perf/Using Performance Counters
Outline

Introduction

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions
  Expression Trees
  Parallel Patterns and Array Languages
  Internal Representations
  The Importance of Semantics: Defined and Undefined Behavior
  The Importance of Batches: kernels and traffic cops
  Functional and not
  Lazy and eager

Polyhedral Representation and Transformation
Outline

Introduction

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions
  Expression Trees
  Parallel Patterns and Array Languages
  Internal Representations
  The Importance of Semantics: Defined and Undefined Behavior
  The Importance of Batches: kernels and traffic cops
  Functional and not
  Lazy and eager

Polyhedral Representation and Transformation