Announcements:
- Fixed demo
- Grading
- Hong Kong contest
Outer-Loop/inner-Loop Vectorization

Contrast outer-loop vs inner-loop vectorization.

Side q: Would you consider GPUs outer- or inner-loop-vectorizing?
Alignment: How?

The old way:

```c
int __attribute__((aligned (8))) a_int;
```

Difference between these two?

```c
int __attribute__((aligned (8))) * ptr_t_1;
int *__attribute__((aligned (8))) ptr_t_2;
```

The ’new’ way (C/C++11):

```c
struct alignas(64) somestruct_t { /* ... */ };
struct alignas(sizeof(other_t))
    somestruct_t { /* ... */ };
struct
    alignas(
        std::hardware_destructive_interference_size)
    somestruct_t { /* ... */ };
```

What is constructive interference?
Alignment: Why?

What is the concrete impact of the constructs on the previous slide?

Allocating (Stack): allocated w/ non-alignment.

Passing pointers: promise about alignment.

Allocating aligned memory (Heap):
- post-manualign
- overallocate by alignment - 1
Pointers and Aliasing

Demo: machabstr/Pointer Aliasing
What if the register working set gets larger than the registers can hold? What is the performance impact?

Demo: machabstr/Register Pressure
Object-Oriented Programming

Object-oriented programming: The weapon of choice for encapsulation and separation of concerns!

Performance perspective on OOP?

- SOA vs AOS
- Runtime polymorphism

Demo: machabstr/Object Orientation vs Performance
Being Nice to Your Compiler

Some rules of thumb:

- Use indices rather than pointers
- Extract common subexpressions
- Make functions static
- Use const
- Avoid store-to-load dependencies

What are the concrete impacts of doing these things?
Outline

Introduction

Machine Abstractions
- C
- OpenCL/CUDA
  - Convergence, Differences in Machine Mapping
  - Lower-Level Abstractions: SPIR-V, PTX

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions

Program Representation and Transformation

Polyhedral Representation and Transformation
Chip Real Estate

65 nm, 4 SP ops at a time, 1 MiB L2.
“CPU-style” Cores

[Fetch/Decode]

[ALU (Execute)]

[Execution Context]

[Out-of-order control logic]

[Fancy branch predictor]

[Memory pre-fetcher]

[Data cache (A big one)]

[Fatahalian ‘08]
Slimming down

Idea #1:
Remove components that help a single instruction stream run fast

[Fatahalian ‘08]
More Space: Double the Number of Cores

[Fetch/Decode]

ALU (Execute)

Execution Context

[Fatahalian ‘08]
Even more

[Even more image]

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian '08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

[Fatahalian ‘08]
Latency Hiding

- Latency (mem, pipe) hurts non-OOO cores
- Do *something* while waiting

What is the unit in which work gets scheduled on a GPU?

- *SIMD vector* "warp" / "wavefront"

How can we keep busy?

- *SIMD*
- *LDP*

Change in architectural picture?
GPU Abstraction: Core Beliefs

- View concrete counts as an implementation detail
  - SIMD lane
  - Core
  - Scheduling slot
- Program as if there are infinitely many of them
- Hardware division is expensive
  Make $nD$ grids part of the model to avoid it
- Design the model to expose extremely fine-grain concurrency
  (e.g. between loop iterations!)
- Draw from the same pool of concurrency to hide latency
'SIMT'
Wrangling the Grid

get_local_id(axis)?/size(axis)?
get_group_id(axis)?/num_groups(axis)?
get_global_id(axis)?/size(axis)?
axis=0,1,2,...