Plan for today:
- DeVries survey
- Follow-ups
- Nuts & bolts
  machine details

 HW O
(Black-Box) Optimizing Compiler: Challenges

Why is black-box optimizing compilation so difficult?

- Application developer knowledge lost
  - Simple example: “Rough” matrix sizes
  - Data-dependent control flow
  - Data-dependent access patterns
  - Activities of other, possibly concurrent parts of the program
  - Profile-guided optimization can recover some knowledge

- Obtain proofs of required properties
- Size of the search space

Consider

Lies, Lies Everywhere

- Semantics form a contract between programmer and language/environment
- Within those bounds, the implementation is free to do as it chooses
- True at every level:
  - Assembly
  - “High-level” language (C)

Give examples of lies at these levels:

- Out-of-order execution
- Strength reduction

One approach: Lie to yourself

- “Domain-specific languages” ← A fresh language, I can do what I want!
- Consistent semantics are notoriously hard to develop
  - Especially as soon as you start allowing subsets of even (e.g.) C’s integers
Class Outline

High-level Sections:

- Intro, Armchair-level Computer Architecture
- Machine Abstractions
- Performance: Expectation, Experiment, Observation
- Programming Languages for Performance
- Program Representation and Optimization Strategies
- Code Generation/JIT
Survey: Class Makeup

- Compiler class: 11 no, 3 yes
- HPC class: 10 yes, 4 no
- C: very proficient on average
- Python: proficient on average
- Assembly: some have experience
- GPU: Half the class has experience, some substantial
- CPU perf: Very proficient
- 10 PhD, 4 Masters, mostly CS (plus physics, CEE, MechSE)
Survey: Learning Goals

- How to use hardware efficiently to write fast code (1 response)
- I want to learn about commonly encountered problems in HPC and efficient ways to approach and solve them. (1 response)
- about writing high performance code for large scale problems. (1 response)
- more (and more) about high-performance computing beyond parallel programming. (1 response)
- This summer (while interning at Sandia national labs), I got familiar with GPU programming using Kokkos as the back end. I enjoyed this work immensely, and hope to continue learning about it, especially so that I can become better at writing GPU code myself. I am also interested in the relationship between a higher level abstraction (Kokkos), the compiler, and the actual compute device (GPU/CPU) relate together, and what tricks we have to help fix issues regarding this. For example, Kokkos uses a small amount of template metaprogramming to convert the source code into actual code. (1 response)
- Some GPU stuff, course description sounded interesting for my research in HPC/Parallel Computing. Would be interesting to look at different programming models or abstractions for HPC. (1 response)
- Getting better at doing high performance computing. (1 response)
- become more familiar with abstractions (1 response)
- I want to be able to auto generate performance portable C++ code, specifically for small batched tensor contractions. (1 response)
- Languages and abstractions for high-performance scientific computing (1 response)
- Investigating problems in high performance computing and looking for solutions, especially large-scale and using GPUs. (1 response)
- Better ways to efficiently (in terms of human time) write high-performance code that may be useful to/Readable by others (1 response)
- about high-level languages and frameworks for high performance computing, the different interfaces they expose, compilation and runtime techniques they use, and the tradeoffs of these for an application developer. (1 response)
Outline

Introduction

About This Class
Why Bother with Parallel Computers?
Lowest Accessible Abstraction: Assembly
Architecture of an Execution Pipeline
Architecture of a Memory System

Machine Abstractions

Performance Measurement
Moore’s Law

Transistor count
doubling every two years

Date of introduction

Issue: More transistors = faster?

\[
\frac{\text{Work}}{s} = \text{Clock Frequency} \times \frac{\text{Work}}{\text{Clock}}
\]
Dennard Scaling of MOSFETs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Voltage</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Current</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Capacitance</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Delay Time</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Power dissipation/circuit</td>
<td>$1/\kappa^2$</td>
</tr>
<tr>
<td>Power density</td>
<td>1</td>
</tr>
</tbody>
</table>

[Dennard et al. ’74, via Bohr ’07]

- Frequency $= \text{Delay time}^{-1}$
MOSFETs ("CMOS" – "complementary" MOS): Schematic

[Dennard et al. ‘74]
MOSFETs: Scaling

- 'New' problem at small scale:
  - Sub-threshold leakage (due to low voltage, small structure)
  - Dennard scaling is over – and has been for a while.

[Intel Corp.]
Peak Architectural Instructions per Clock: Intel

<table>
<thead>
<tr>
<th>CPU</th>
<th>IPC</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 1</td>
<td>1.1</td>
<td>1993</td>
</tr>
<tr>
<td>Pentium MMX</td>
<td>1.2</td>
<td>1996</td>
</tr>
<tr>
<td>Pentium 3</td>
<td>1.9</td>
<td>1999</td>
</tr>
<tr>
<td>Pentium 4 (Willamette)</td>
<td>1.5</td>
<td>2003</td>
</tr>
<tr>
<td>Pentium 4 (Northwood)</td>
<td>1.6</td>
<td>2003</td>
</tr>
<tr>
<td>Pentium 4 (Prescott)</td>
<td>1.8</td>
<td>2003</td>
</tr>
<tr>
<td>Pentium 4 (Gallatin)</td>
<td>1.9</td>
<td>20</td>
</tr>
<tr>
<td>Pentium D</td>
<td>2</td>
<td>2005</td>
</tr>
<tr>
<td>Pentium M</td>
<td>2.5</td>
<td>2003</td>
</tr>
<tr>
<td>Core 2</td>
<td>3</td>
<td>2006</td>
</tr>
<tr>
<td>Sandy Bridge...</td>
<td>4ish</td>
<td>2011</td>
</tr>
</tbody>
</table>

[Charlie Brej http://brej.org/blog/?p=15](http://brej.org/blog/?p=15)

Discuss: How do we get out of this dilemma?
The Performance Dilemma

- IPC: Brick Wall
- Clock Frequency: Brick Wall

Ideas:
- Multiple copies of the same core (SPMD)
- Specialized HW
- SIMD

Question: What is the *conceptual* difference between those ideas?

- Lockstep execution
- Data vs. task parallel
The Performance Dilemma: Another Look

- **Really**: A crisis of the 'starts-at-the-top-ends-at-the-bottom' programming model
- **Tough luck**: Most of our codes are written that way
- **Even tougher luck**: Everybody on the planet is trained to write codes this way

So:

- **Need**: Different tools/abstractions to write those codes
Outline

Introduction
About This Class
Why Bother with Parallel Computers?
Lowest Accessible Abstraction: Assembly
Architecture of an Execution Pipeline
Architecture of a Memory System

Machine Abstractions

Performance Measurement
A Basic Processor: Closer to the Truth

- Memory Interface
- Address Bus
- Data Bus

- Internal Bus
  - Instruction fetch
  - PC
  - Control Unit
  - Data ALU

- Address ALU
- Register File
- Flags

- Loosely based on Intel 8086
- What's a [bus](#)?
A Very Simple Program

```c
int a = 5;
int b = 17;
int z = a * b;
```

Things to know:

- **Question:** Which is it?
  - `<opcode> <src>, <dest>`
  - `<opcode> <dest>, <src>`

- **Addressing modes** (Immediate, Register, Base plus Offset)

- **0xHexadecimal**
A Very Simple Program: Another Look

4: c7 45 f4 05 00 00 00 00 movl $0x5,-0xc(%rbp)
b: c7 45 f8 11 00 00 00 00 movl $0x11,-0x8(%rbp)
12: 8b 45 f4 mov -0xc(%rbp),%eax
15: 0f af 45 f8 imul -0x8(%rbp),%eax
19: 89 45 fc mov %eax,-0x4(%rbp)
1c: 8b 45 fc mov -0x4(%rbp),%eax
A Very Simple Program: Intel Form

4: c7 45 f4 05 00 00 00 mov DWORD PTR [rbp-0xc],0x5
b: c7 45 f8 11 00 00 00 mov DWORD PTR [rbp-0x8],0x11
12: 8b 45 f4 mov eax,DWORD PTR [rbp-0xc]
15: 0f af 45 f8 imul eax,DWORD PTR [rbp-0x8]
19: 89 45 fc mov DWORD PTR [rbp-0x4],eax
1c: 8b 45 fc mov eax,DWORD PTR [rbp-0x4]

▶ “Intel Form”: (you might see this on the net)
  <opcode> <sized dest>, <sized source>
▶ Previous: “AT&T Form”: (we’ll use this)
▶ Goal: Reading comprehension.
▶ Don’t understand an opcode?
Assembly Loops

```assembly
int main()
{
    int y = 0, i;
    for (i = 0; y < 10; ++i)
        y += i;
    return y;
}
```

Things to know:
- **Condition Codes (Flags)**: Zero, Sign, Carry, etc.
- **Call Stack**: Stack frame, stack pointer, base pointer
- **ABI**: Calling conventions
Demos

Demo: intro/Assembly Reading Comprehension

Demo: Source-to-assembly mapping
Code to try:

```c
int main()
{
    int y = 0, i;
    for (i = 0; y < 10; ++i)
        y += i;
    return y;
}
```
Outline

Introduction
- About This Class
- Why Bother with Parallel Computers?
- Lowest Accessible Abstraction: Assembly

Architecture of an Execution Pipeline
- Architecture of a Memory System

Machine Abstractions

Performance Measurement
Modern Processors?

All of this can be built in about 4000 transistors.
(e.g. MOS 6502 in Apple II, Commodore 64, Atari 2600)

So what exactly are Intel/ARM/AMD/Nvidia doing with the other billions of transistors?
Execution in a Simple Processor

- **[IF]** Instruction fetch
- **[ID]** Instruction Decode
- **[EX]** Execution
- **[MEM]** Memory Read/Write
- **[WB]** Result Writeback

[Wikipedia ☛]
Solution: Pipelining
MIPS Pipeline: 110,000 transistors
Hazards and Bubbles

Q: Types of Pipeline Hazards? (aka: what can go wrong?)

- Structural
- Data
- Control

[Wikipedia ©]
Demo: intro/Pipeline Performance Mystery
A Glimpse of a More Modern Processor

[Sandy Bridge]

Instruction Fetch Unit

Branch Predictors

144 Entry L1 ITLB (4 way)

32KB L1 I-Cache (8 way)

16B Predecode, Fetch Buffer

18+ Entry Instruction Queue

Code Engine

Complex Decodes

Simple Decodes

Simple Decodes

1.5K μop Cache (8 way)

28 μop Decoder Queue

168 Entry Reorder Buffer (ROB)

144 Entry FP Physical Register File

160 Entry Physical Register File

54 Entry Unified Scheduler

Port 0

ALU LEA Shift

SIMD MUL Shift

ALU LEA MUL

ALU Shift Branch

SIMD ALU Shuffle

64-bit FMUL Blend

256-bit FADD

256-bit ALU Shuffle

256-bit Shuffle Blend

64-bit AGU

64-bit AGU

Store Data

512 Entry L2 TLB (4 way)

100 Entry L1 DTLB (fully)

32KB L1 D-Cache (8 way)

256KB L2 Cache (8 way)

[David Kanter / Realworldtech.com]
A Glimpse of a More Modern Processor: Frontend

[Sandy Bridge diagram]

[David Kanter / Realworldtech.com]
A Glimpse of a More Modern Processor: Backend

- New concept: Instruction-level parallelism ("ILP", "superscalar")

- Where does the IPC number from earlier come from?

[David Kanter / Realworldtech.com]
Demo: intro/More Pipeline Mysteries