Programming for Performance  
CS 378  
Andrew Lenharth  
8/27/2014

Administration

• Instructor:  
  – Andrew Lenharth (Research Associate in ICES and Lecturer in CS)  
  – Email: lenharth@ices.utexas.edu  
  – Office: ACES 4.124  
  – Office Hours: M,T 4-5

• TA:  
  – Lijia Liu (CS)  
  – Email:  
  – Office:  
  – Office Hours: 

Prerequisites

• Basic computer architecture knowledge:  
  – PC, pipelines, out-of-order, cache, memory

• Knowledge of C++/Java-like languages  
  – Indirect calls, virtual functions, stack, heap, iteration, recursion

• Software maturity  
  – Able to read APIs/documentation  
  – Able to write small programs from scratch (~1000 lines)  
  – Able to write and reason about recursive data structures

Coursework

• 5 programming projects  
  – About 3 weeks apart  
  – Written portion and programming portion

• Homeworks most other weeks  
  – Intended to familiarize you with thing you will need for the next project

• Final Presentation  
  – ~10 minute talk on the final project

Textbook

None

• Some useful references given in the syllabus

• Lots of material on the web  
  – You will have to look information up

What this course is not

• Tools/libraries  
  – Though we will use them  
  – We will need to understand what they do/measure

• Clever hacks  
  – There are many, see for example “Hackers Delight”

• Low-level performance tuning  
  – We have compilers, we will use them, except...
What this course is

• How to use hardware features to boost program performance
  – Understand why they exist
  – Understand the problems that motivate them
  – Understand why we have to think about them
  – Mental models

• Major features:
  – Caches – reducing memory latency
  – Vector units – doing more work at once
  – Multi-core – doing even more work

0th order performance model

• Data (and instructions) are fetched from memory
• Instructions are executed by the processor
• Run everything faster
• Compute more
• Bring data in faster

Moore’s laws

“That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.
I believe that such a large circuit can be built on a single wafer.”
--Moore

Moore’s laws

• Number of transistors on an IC double every 2 years
• Not:
  – Computers get twice as faster every whatever
  – Transistor density doubles every whatever
• So What?

Density does increase

Smaller Transistors Switch Faster

(Not the only thing going on in this graph)
So What?

- Don't use them!
- Same chip, but smaller
  - Micro-controllers on the legs of bees
  - Lightbulbs with WiFi
- Computers everywhere
- Use transistors to speed up program

Basic Uses of Transistors

- Add Functional Units -- Do more computation per cycle
  - Specialized instructions (e.g. AES)
  - Vectors (multiple data per instruction)
  - Parallel execution (multiple instructions per cycle)
    - SW (VLIW) v.s. HW OoO
  - Concurrent execution
    - Pipelining -- same thread
    - SMT -- different thread
- Wider units
  - More precise floating point
  - Larger integers

Basic Uses of Transistors

- Pipeline
  - Shorten critical path
  - Higher frequencies
  - Branches, caches, power limit benefit
- Out-of-order
  - Find more work to do each cycle
  - Construct dynamic dependency graphs between instructions and find independent sets
  - Speculation required, accuracy limits benefit
- More cache

Software’s Task

- Schedule for pipeline, OoO, VLIW, etc
  - Compilers do a good job
- Use specialized instructions
  - Libraries do a good job
- Operate on vectors
  - Compilers need a lot of help
- Place nice with speculation
  - Must understand how language features are implemented

“Memory” is Slower Than Processors

<table>
<thead>
<tr>
<th>Year</th>
<th>DRAM</th>
<th>SRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>0.1 ns</td>
<td>0.01 ns</td>
</tr>
<tr>
<td>1995</td>
<td>0.05 ns</td>
<td>0.005 ns</td>
</tr>
<tr>
<td>2000</td>
<td>0.02 ns</td>
<td>0.002 ns</td>
</tr>
<tr>
<td>2005</td>
<td>0.01 ns</td>
<td>0.001 ns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 transistor + 1 capacitor per bit</td>
</tr>
<tr>
<td>Very high density</td>
</tr>
<tr>
<td>Moderate power</td>
</tr>
<tr>
<td>Not so fast</td>
</tr>
<tr>
<td>Needs constant attention</td>
</tr>
<tr>
<td>Manufactured differently than processors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 transistors per bit</td>
</tr>
<tr>
<td>Low density</td>
</tr>
<tr>
<td>High power</td>
</tr>
<tr>
<td>Set-it-and-forget-it</td>
</tr>
<tr>
<td>Fast as the processor</td>
</tr>
<tr>
<td>Same basic building blocks as logic</td>
</tr>
</tbody>
</table>

What is “Memory”

- Hard to put on the same chip
- Trivial to put on the same chip
Memory Hierarchy

- Lots of the cheap, dense, slow memory
- Some of expensive, less-dense, fast memory
- Haswell (min hit times)
  - 168 registers – 1 cycle access time (well...)
  - 32KB L1 – 4 cycles
  - 256KB L2 – 11 cycles
  - 8MB L3 – 40-80 cycles (local)
  - 128 L4 (EDram) – ~100 cycles
  - 4-64GB RAM – 200+ cycles

Software's Task

- Keep the stuff we need in the fast memory
- But hardware manages the cache
- Design algorithms and data-structures to work with the hardware
  - Compilers can’t do too much here

Locality

- Caches help programs with locality of reference
  - Temporal locality: access a recently accessed memory location
  - Spacial locality: access a location near a recently accessed location
  - Predictable strides: hardware can figure out regular traversals and pre-fill the cache
- Easy to lose 2-10x performance by poor cache behavior

Not a New Problem

"...The CPU chip industry has now reached the point that instructions can be executed more quickly than the chips can be fed with code and data. Future chip design is memory design. Future software design is also memory design. .... Controlling memory access patterns will drive hardware and software designs for the foreseeable future.”

-- Richard Sites, DEC

Power Dissipation

- Dynamic power is proportional to clock speed and is incurred when gates switch
- Static power is background loss and gets worse at smaller transistor sizes

Power is more than just frequency

* Patrick Gelsinger’s most famous figure ever

* Hikkel Electronics Asia
This Decade’s Problem

- ILP extraction limited
  - Increased speculation of limited help and high cost
- Vector units also limited
  - Great when they can be used, but not usable for all software

What to do with the transistors now?

- Make special purpose hardware
  - Mp3 decoder, video decoder, image compositor, encryption unit, audio codecs, etc.
  - Bonus: orders of magnitude less power usage
  - Hard to use for tasks other than as designed

What about making general purpose programs faster?

Multi-Core

- Replicate CPUs on a die
  - SMP is old and well understood
  - No change to micro-architecture
  - No need to extract more ILP from programs
- Don’t increase clock speed (controls power)

Multi-Core

- More aggregate performance for:
  - Multi-threaded apps
  - Transactions (many instances of same app)
  - Multi-tasking
- No performance (or negative) for:
  - Legacy apps
  - Single-threaded apps

Multi-Core Programability

- Writing multi-threaded code is costly
  - 3x for Unreal (Tim Sweeney)
- Writing correct multi-threaded code is (currently) hard
  - Good abstractions are active research problems
- Writing scalable multi-threaded code is hard
  - Amdahl’s law
  - New overheads
  - New hardware considerations

“We are the cusp of a transition to multicore, multithreaded architectures, and we still have not demonstrated the ease of programming the move will require... I have talked with a few people at Microsoft Research who say this is also at or near the top of their list [of critical CS research problems].” Justin Ratt

Software’s Task

- Split programs up so that multiple computations can happen concurrently
  - How big a computation?
  - When can we even do this and when is it profitable?
- Synchronize to enforce required order and non-interference
- Deal with caches and memory
  - Now even more complex
Amdahl’s Law

- Simple observation that shows that unless most of the program can be speed up (say by executing in parallel, the benefits are limited
  - Remainder of program becomes bottleneck
- Example: I drive from Austin to San Antonio, which is 90 miles, at 60 mph. What is my travel time if:
  - I return at 60 mph
  - I return at 120 mph
  - I return infinitely fast

Amdahl’s Law

- Give a program with a serial portion and a parallel portion with $N$ operations:
  - $r \times N$ operations in parallel section
  - $(1-r) \times N$ operations in serial portion
- Assume operations take unit time with infinite cores
- Speed-up:
  \[
  \frac{\text{single core}}{\text{multicore}} = \frac{N}{(1-r) \times N} = \frac{1}{1-r}
  \]
- Generalizes to speedup given speedup of accelerated portion

Course Content

- Using caches effectively
  - Locality, algorithm choice, data-structures
- Using vector units
- Using mutli-core
  - Shared memory parallel programming
  - Synchronization, memory models, abstractions
- Understanding the cost of abstractions and compilers
- Measurement