ECE 5775
High-Level Digital Design Automation
Fall 2018

Specialized Computing
Announcements

- All students enrolled in CMS & Piazza
  - Email instructor otherwise

- Lab 1 to be released next Monday
Outline

- Motivation for specialized computing
  - Key driving forces from applications and technology
  - Main sources of inefficiency in general-purpose computing

- FPGA introduction
  - Basic building blocks
  - Classical homogeneous FPGA architectures
  - Modern heterogeneous FPGA architectures
30 Years of Hot Chips Symposium (www.hotchips.org)

Hot Chips 1 (1989)

Hot Chips 10 (1998)

Hot Chips 20 (2008)

Hot Chips 30 (2018)
The resurgence of AI is revolutionizing many crucial aspects of the world

- Self-driving vehicles
- Image recognition and scene modeling
- Game AI (e.g. AlphaGo)

A key method is the deep neural networks (DNNs)
Modern DNNs are Computationally “Expensive”

- DNNs require enormous amount of compute
  - For example, ResNet50 (70 layers) performs 7.7 billion operations required to classify one image
On Crash Course with the End of “Cheap” Technology Scaling

End of Dennard scaling: power becomes the key constraint
- Amdahl’s Law and dark silicon prevent “easy” multicore scaling
End of Dennard Scaling and Dark Silicon

- Classical scaling
  - Frequency increases at constant power profiles
    - Performance improves “for free”!

- Leakage limited scaling
  - $V_{th}$ virtually stopped scaling due to exponentially increasing leakage power
  - $V_{DD}$ scaling nearly stopped as well to maintain performance

- Dark silicon
  - Power constraints limit how much of the chip can be activated at any one time (not 100% anymore)

**Classical Dennard scaling**

<table>
<thead>
<tr>
<th>Device (transistor) #</th>
<th>$S^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitance / device</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Voltage ($V_{dd}$)</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Frequency</td>
<td>$S$</td>
</tr>
<tr>
<td><strong>Total power</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

**Leakage limited scaling**

<table>
<thead>
<tr>
<th>Device (transistor) #</th>
<th>$S^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitance / device</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Voltage ($V_{dd}$)</td>
<td>$\sim 1$</td>
</tr>
<tr>
<td>Frequency</td>
<td>$\sim 1$</td>
</tr>
<tr>
<td><strong>Total power</strong></td>
<td>$S$</td>
</tr>
</tbody>
</table>
Tradeoff between Compute Efficiency and Flexibility

Why is general-purpose CPU less energy efficient?
Understanding Energy Inefficiency of General-Purpose Processors (GPPs)

Typical Superscalar OoO Pipeline

Parameter | Value
--- | ---
Fetch/issue/retire width | 4
# Integer ALUs | 3
# FP ALUs | 2
# ROB entries | 96
# Reservation station entries | 64
L1 I-cache | 32 KB, 8-way set associative
L1 D-cache | 32 KB, 8-way set associative
L2 cache | 6 MB, 8-way set associative

[source: Jason Cong, ISLPED’14 keynote]
Energy Breakdown of Pipeline Components

- Fetch: 9%
- Decode: 6%
- Rename: 12%
- Register files: 3%
- Scheduler: 11%
- Int ALU: 14%
- Mul/div: 4%
- FPU: 8%
- Memory: 10%
- Misc: 23%

Diagram:

- L1-I$ (Fetch, Branch Predictor)
- Decode
- Rename
- Free list
- Int RF
- FP RF
- Register Read/write
- LSQ + TLB
- D-cache
- ALU
- FPU
- ROB
- Commit
- Control

Legend:
- Fetch unit: 9%
- Rename: 12%
- Register files: 3%
- Scheduler: 11%
- Int ALU: 14%
- Mul/div: 4%
- FPU: 8%
- Memory: 10%
- Misc: 23%
Removing “Non-Computing” Portions

“Computing” portion: 10% (memory) + 26% (compute) = 36%
Energy Comparison of Processor ALUs and Dedicated Units

<table>
<thead>
<tr>
<th>Operation</th>
<th>Processor ALU</th>
<th>45 nm TSMC standard cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit add</td>
<td>0.122 nJ @ 2 GHz</td>
<td>0.002 nJ @ 1 GHz</td>
</tr>
<tr>
<td>32-bit multiply</td>
<td>0.120 nJ @ 2 GHz</td>
<td>0.007 nJ @ 1 GHz</td>
</tr>
<tr>
<td>Single-precision FP operation</td>
<td>0.150 nJ @ 2 GHz</td>
<td>0.008 nJ @ 500 MHz</td>
</tr>
</tbody>
</table>

Why are processor units so expensive?
- ALU can perform multiple operations
  - Add/sub/bitwise XOR/OR/AND
- 64-bit ALU
- Dynamic/domino logic used to run at high frequency
  - Higher power dissipation
Energy Breakdown with Standard-Cell ASICS

- Fetch unit: 9%
- Decode: 6%
- Rename: 12%
- Register files: 3%
- Scheduler: 11%
- Int ALU: 0.2%
- Mul/div: 0.2%
- FPU: 0.4%
- ALU/FPU savings: 25.0%
- Misc: 23%
- Memory: 10%

“Computing” portion: 10% (memory) + ~1% (compute) = 11%

Still, only 10X gain is attainable?
Additional Energy Savings from Specialization

- **Customized data types**
  - Exploit data range information to reduce bitwidth/precision and simply arithmetic operations

- **Specialized memory architecture**
  - Exploit regular memory access patterns to minimize energy per memory read/write

- **Specialized communication architecture**
  - Exploit data movement patterns to optimize the structure/topology of on-chip interconnection network

These techniques combined can lead to another 10-100X energy efficiency improvement over GPPs
Tradeoff between Compute Efficiency and Flexibility

What makes FPGA an interesting compute substrate?

Contr

Control Unit (CU)

Registers

Arithmetic Logic Unit (ALU)

CPUs

GPUs

FPGAs

ASICs
What is an FPGA?

▸ FPGA: Field-Programmable Gate Array
  – An integrated circuit designed to be configured by a customer or a designer after manufacturing (wikipedia)

▸ Components in an FPGA Chip
  – Programmable logic blocks
  – Programmable interconnects
  – Programmable I/Os
Three Important Pieces

- SRAM-based implementation is popular
  - Non-standard technology means older technology generation

 Lookup table (LUT, formed by SRAM bits)

 Pass transistor (controlled by an SRAM bit)

 Multiplexer (controlled by SRAM bits)
Multiplexer as a Universal Gate

- Any function of k variables can be implemented with a $2^k:1$ multiplexer

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$8:1$ MUX
How Many Functions?

- How many distinct 3-input 1-output Boolean functions exist?

- What about K inputs?
A k-input LUT (k-LUT) can be configured to implement any k-input 1-output combinational logic
- \(2^k\) SRAM bits
- Delay is independent of logic function
Exercise: Implementing Logic with LUTs

Implement a 2:1 MUX using a network of 2-input LUTs. Use the minimum number of LUTs.

Building block: 2-input LUT
Example: Implementing 6-LUT with 5-LUTs

- A 6-input LUT can be two 5-input LUTs with common inputs (used in Xilinx 7-series FPGAs)
  - Any function of six variables or two independent functions of five variables
A Logic Element

- A k-input LUT is usually followed by a flip-flop (FF) that can be bypassed
- The LUT and FF combined form a logic element
A Logic Block

- A logic block clusters multiple logic elements

- Example: In Xilinx 7-series FPGAs, each configurable logic block (CLB) has two slices
  - Two independent carry chains per CLB for implementing adders
  - Each slice contains four LUTs
Traditional Homogeneous FPGA Architecture

Switch block

Logic block

Routing track
Modern Heterogeneous Field-Programmable System-on-Chip

- Island-style configurable mesh routing
- Lots of dedicated components
  - Memories/multipliers, I/Os, processors
  - Specialization leads to higher performance and lower power

[Figure credit: embeddedrelated.com]
Dedicated DSP Blocks

- Built-in components for fast arithmetic operation optimized for DSP applications
  - Essentially a multiply-accumulate core with many other features
  - Fixed logic and connections, functionality may be configured using control signals at run time
  - Much faster than LUT-based implementation (ASIC vs. LUT)
Example: Xilinx DSP48E Slice

- 25x18 signed multiplier
- 48-bit add/subtract/accumulate
- 48-bit logic operations
- SIMD operations (12/24 bit)
- Pipeline registers for high speed

[source: Xilinx Inc.]
Dedicated Block RAMs (BRAMs)

- Example: Xilinx 18K/36K block RAMs
  - 32k x 1 to 512 x 72 in one 36K block
  - Simple dual-port and true dual-port configurations
  - Built-in FIFO logic
  - 64-bit error correction coding per 36K block

[source: Xilinx Inc.]
Embedded FPGA System-on-Chip

Dual ARM Cortex-A9 + NEON SIMD extension @600MHz~1GHz

Up to 350K logic cells
2MB Block RAM
900 DSP48s

Xilinx Zynq All Programmable System-on-Chip
[Source: Xilinx Inc.]
FPGA as an Accelerator for Cloud Computing

Amazon EC2 F1 instances

Launch Instance and Load AFI

CPU Application on F1

PCIe

DDR Controllers

DDR-4 Attached Memory

FPGA Link

EC2 F1 Instance

Amazon Machine Image (AMI)

~2 Million Logic Blocks

~5000 DSP Blocks

~300Mb Block RAM

Block RAM

Block RAM

~2 Million Logic Blocks

~5000 DSP Blocks

~300Mb Block RAM

Xilinx UltraScale+ VU9P

[Figure source: David Pellerin, AWS]
FPGA Deployment in Datacenter

- FPGAs deployed in Microsoft datacenters to accelerate various web, database, and AI services
  - e.g., project BrainWave claimed ~40Teraflops on large recurrent neural networks using Intel Stratix 10 FPGAs

[source: Microsoft, BrainWave, HotChips’2017]
Summary: FPGA as a Programmable Accelerator

- Massive amount of fine-grained parallelism
  - Highly parallel and/or deeply pipelined to achieve maximum parallelism
  - Distributed data/control dispatch

- Silicon configurable to fit algorithm
  - Compute the exact algorithm at the desired level of numerical accuracy
    - Bit-level sizing and sub-cycle chaining
  - Customized memory hierarchy

- Performance/watt advantage
  - Low power consumption compared to CPU and GPGPUs
    - Low clock speed
    - Specialized architecture blocks
Next Class

- Fixed-point data types
- Basics of algorithm analysis
Acknowledgements

- These slides contain/adapt materials developed by
  - Prof. Jason Cong (UCLA)