ECE 5775
High-Level Digital Design Automation
Fall 2022

Hardware Specialization
Announcements

- First reading assignment
  - Complete reading **before Thursday 9/1**

- Hands-on HLS tutorial next Tuesday
  - Bring your laptop
Agenda

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

- Essential specialization techniques

- Roofline-based performance modeling
A Golden Age of Hardware Specialization

▸ **Higher demand** on efficient compute acceleration, esp. for machine learning (ML) workloads

▸ **Lower barrier** with cloud FPGAs & open-source hardware coming of age
Modern ML Models are Computationally Demanding

- Deep neural networks (DNNs) require enormous amount of compute
  - For example, ResNet50 (70 layers) performs 7.7 billion operations to classify one image

On Crash Course with the End of “Cheap” Technology Scaling

› End of Dennard scaling: power becomes a key constraint
  – Amdahl’s Law and dark silicon prevent “easy” multicore scaling

End of Dennard Scaling and Dark Silicon

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

- Leakage limited scaling
  - $V_{\text{th}}$ virtually stopped scaling due to exponentially increasing leakage power
  - $V_{\text{DD}}$ 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>Transistor (trans.) #</th>
<th>$S^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitance / trans.</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Voltage ($V_{\text{dd}}$)</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Frequency</td>
<td>$S$</td>
</tr>
<tr>
<td><strong>Total power</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

### Leakage limited scaling

<table>
<thead>
<tr>
<th>Transistor (trans.) #</th>
<th>$S^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacitance / trans.</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Voltage ($V_{\text{dd}}$)</td>
<td>$\sim1$</td>
</tr>
<tr>
<td>Frequency</td>
<td>$\sim1$</td>
</tr>
<tr>
<td><strong>Total power</strong></td>
<td><strong>$S$</strong></td>
</tr>
</tbody>
</table>
Ending of Moore’s Law? A Reality Check of Density Scaling

[Figure credit: David Brooks, Harvard]
Trade-off Between Compute Efficiency and Flexibility

General-purpose CPU is less energy efficient
WHY?
Rough Energy Breakdown for an Instruction

Estimated energy costs for various integer, floating-point (FP), and memory operations on CPU (target node: 45nm at 0.9V)

<table>
<thead>
<tr>
<th>Integer</th>
<th>FP</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>FAdd</td>
<td>Cache</td>
</tr>
<tr>
<td>8 bit</td>
<td>16 bit</td>
<td>8KB</td>
</tr>
<tr>
<td>0.03pJ</td>
<td>0.4pJ</td>
<td>10pJ</td>
</tr>
<tr>
<td>32 bit</td>
<td>32 bit</td>
<td>32KB</td>
</tr>
<tr>
<td>0.1pJ</td>
<td>0.9pJ</td>
<td>20pJ</td>
</tr>
<tr>
<td>Mult</td>
<td>FMult</td>
<td>1MB</td>
</tr>
<tr>
<td>8 bit</td>
<td>16 bit</td>
<td>100pJ</td>
</tr>
<tr>
<td>0.2pJ</td>
<td>1.1pJ</td>
<td></td>
</tr>
<tr>
<td>32 bit</td>
<td>32 bit</td>
<td>DRAM</td>
</tr>
<tr>
<td>3.1pJ</td>
<td>3.7pJ</td>
<td>1.3-2.6nJ</td>
</tr>
</tbody>
</table>

M. Horowitz, Computing’s energy problem (and what we can do about it), ISSCC’2014.
Reducing Compute Energy Overhead

A sequence of energy-inefficient instructions

<table>
<thead>
<tr>
<th>I-Cache</th>
<th>RF</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... Arithmetic

Single instruction multiple Data (SIMD): tens of operations per instruction

<table>
<thead>
<tr>
<th>I-Cache</th>
<th>RF</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Further specialization (what we achieve using accelerators)

<table>
<thead>
<tr>
<th>I-Cache</th>
<th>RF</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... hundreds or more

[Figure credit] W. Qadder, et al., Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing, ISCA'2013.
Additional Energy Savings from Specialization

- **Customized data types**
  - Exploit accuracy-efficiency trade-off to simplify arithmetic operations and reduce memory accesses

- **Customized memory hierarchy**
  - Exploit regular memory access patterns to minimize energy per memory read/write

- **Customized 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 general-purpose processors**
Customized Data Types

- Using custom numeric types tailored for a given application/domain improves performance & efficiency

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half float (fp16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bfloat16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>block-fp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixed&lt;9,4&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Covered in lectures & labs
Binary Representation – Positional Encoding

**Unsigned number**
- MSB has a place value (weight) of $2^{n-1}$

**Two’s complement**
- MSB weight = $-2^{n-1}$

<table>
<thead>
<tr>
<th></th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th><strong>unsigned</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>= 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$-2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th><strong>2’c</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>= -5</td>
</tr>
</tbody>
</table>
Fixed-Point Representation of Fractional Numbers

- The positional binary encoding can also represent fractional values, by using a **fixed** position of the binary point and place values with negative exponents
  - Less convenient to use in software, compared to floating point
  - Much more efficient in hardware

<table>
<thead>
<tr>
<th>Unsigned fixed-point number</th>
<th>Integer part (4 bits)</th>
<th>Fractional part (2 bits)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 1</td>
<td>$2^3$  $2^2$  $2^1$  $2^0$  $2^{-1}$  $2^{-2}$</td>
<td>0 1</td>
<td>$11.25$</td>
</tr>
</tbody>
</table>

| Signed fixed-point number | 1 0 1 1 0 1 | $2^{’c}$ | $??$ |
Overflow and Underflow

- **Overflow** occurs when a number is larger than the largest number that can be represented using a given number of bits.

```
2^3 2^2 2^1 2^0 2^-1 2^-2 unsigned
```

- Overflow with 3 integer bits

```
1 0 1 1 0 1 = 11.25
0 1 1 0 1 = 3.25
```

- **Underflow** occurs when a number is smaller than the smallest number that can be represented.
Handling Overflow

- One common & efficient way of handling overflow is to drop the MSB(s) of the original number
  - This is commonly called wrapping

\[
\begin{array}{ccccccc}
-2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2\text{'}c \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & = -4.75 \\
\end{array}
\]

Dropping MSB when integer width is reduced

\[
\begin{array}{ccccccc}
2\text{'}c \\
\hline
0 & 1 & 1 & 0 & 1 & = ?? \\
\end{array}
\]

Wrapping can cause a negative number to become positive, or a positive to negative
Custom Memory Hierarchy: A Case Study on Convolution

- **Convolution** is pervasive in image/video processing and ML – performed over overlapping windows (aka stencils)

\[
(Img \otimes f)[n+\frac{k-1}{2},m+\frac{k-1}{2}] = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} Img[n+i][m+j] \cdot f[i,j]
\]

Input image frame  
KxK convolution (K=3 here)  
Output image frame
Example Application: Edge Detection

- Identifies discontinuities in an image where brightness (or image intensity) changes sharply
  - Very useful for feature extractions in computer vision

![Example Image]

Sobel operator
\[ G = (G_x, G_y) \]

\[
G_x = \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{bmatrix}
\]

\[
G_y = \begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
\]

Figures: Pilho Kim, GaTech
CPU Implementation of a 3x3 Convolution

\[
\text{for} \ (r = 1; \ r < H; \ r++) \\
\quad \text{for} \ (c = 1; \ c < W; \ c++) \\
\quad \quad \text{for} \ (i = 0; \ i < K; \ i++) \\
\quad \quad \quad \text{for} \ (j = 0; \ j < K; \ j++) \\
\quad \quad \quad \quad \text{out}[r][c] += \text{img}[r+i-1][c+j-1] \ast f[i][j];
\]
General-Purpose Cache for Convolution

- A general-purpose cache can effectively reduce external memory accesses
  - but is expensive in cost and incurs nontrivial energy overhead
Line Buffer: Customized “Cache” for Convolution

▸ “Cache” the input pixels in a line buffer
  – Each time we move the KxK window to the right and push in a new pixel to “cache”

```
-3 -2 -1 0 1 2 3 4
5 6 7 8 9 A B C
D E F
```

**Input image frame**

**(K-1)*W+K pixels in flight, K=3 here**
A More Complete Picture: Customized On-Chip Memory Hierarchy

Line Buffer + Shift Registers:
- a custom “cache” + a custom “register file”

Pixels in **line buffer**
- (stores 2 lines using on-chip SRAM)

Push 3 pixels into shift registers – 1 new pixel plus 2 from line buffer

New pixel read from frame buffer in main memory (DRAM)
Custom Communication Architecture
Example: Systolic Arrays

- An array of processing elements (PEs) that process data in a systolic manner using nearest-neighbor communication

> Simple & regular design
> Massive parallelism
> Short nearest-neighbor interconnection
> Balancing compute with I/O
Matrix Multiplication on a Systolic Array

- An array of processing elements that process data in a systolic manner

\[ C = A \times B \]
Accelerator Performance Modeling

**Accelerator (often called a “device”)**

- **Computation Resource**
  - PE-1
  - PE-2
  - ... (PE-n)

- **Interconnect**
  - Buffer 1
  - Buffer 2

- **On-chip memory**

- **Off-chip Bus**

- **External Memory**

---

**Computational Throughput**

\[
\text{Computational Throughput} = \frac{\text{Total number of operations}}{\text{Total execution time}} \quad \text{(OPs / Sec)}
\]

**Operational Intensity**

\[
\text{Operational Intensity}^* = \frac{\text{Total number of operations}}{\text{Total external memory access}} \quad \text{(OPs / Byte Accessed)}
\]

**Memory Bandwidth**

\[
\text{Memory Bandwidth} = \frac{\text{Bytes}}{\text{Sec}}
\]

* OI is also known as computation to communication ratio (CTC) or arithmetic intensity (AI)
Roofline Model \[1\]

Computational Throughput

\[\text{OPs/Sec} = \frac{\text{Computational Throughput}}{\text{OI}}\]

\[\frac{\text{OPs/Sec}}{\text{OPs/Byte Accessed}} = \frac{\text{Bytes}}{\text{Sec}}\]

Roofline Model [1]

Design Space Exploration with Roofline

Design points A & B achieve same throughput

But which one would you prefer?
Exercise: OI Analysis of 2D Convolution

Without line buffer (no cache): OI = ?

With line buffer: OI = ?
Summary

- End of Dennard scaling leads to increasing **hardware specialization** to sustain improvement in hardware performance and energy efficiency.

- Special-purpose hardware **accelerators** commonly leverage customized (1) processing engines, (2) data types, (3) memory hierarchy, and (4) communication architectures.

- **Roofline modeling** is a useful tool for first-order analysis of the accelerator performance.
Reading Assignment

- Before next Thursday (9/1)
Acknowledgements

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