Bachelor’s Seminar in Comp Arch

- Fall 2020
- 2 credit units

- **Rigorous seminar on fundamental and cutting-edge topics in computer architecture**
- Critical presentation, review, and discussion of seminal works in computer architecture
  - We will cover many ideas & issues, analyze their tradeoffs, perform **critical thinking** and brainstorming

- Participation, presentation, synthesis report
- You can register for the course online
  - [https://safari.ethz.ch/architecture_seminar/spring2020/doku.php](https://safari.ethz.ch/architecture_seminar/spring2020/doku.php)
Announcement

- If you are interested in **learning more and doing research** in Computer Architecture, three suggestions:
  - Email me with your interest (CC: Juan)
  - Take the seminar course and the “Computer Architecture” course
  - Do readings and assignments on your own

- There are **many exciting projects and research positions** available, spanning:
  - Memory systems
  - Hardware security
  - GPUs, FPGAs, heterogeneous systems, ...
  - New execution paradigms (e.g., in-memory computing)
  - Security-architecture-reliability-energy-performance interactions
  - Architectures for medical/health/genomics
Broader Agenda

- Single-cycle Microarchitectures
- Multi-cycle and Microprogrammed Microarchitectures
- Pipelining
  - Issues in Pipelining: Control & Data Dependence Handling, State Maintenance and Recovery, ...
- Out-of-Order Execution
- Other Execution Paradigms
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- Fine-Grained Multithreading
- SIMD Processing (Vector and array processors, GPUs)
Readings for Today

- **Required**

- **Recommended**
Readings for Next Week

- **Required**

- **Recommended**
Systolic Arrays
Systolic Arrays: Motivation

- **Goal:** design an accelerator that has
  - **Simple, regular design** (keep # unique parts small and regular)
  - **High concurrency** → high performance
  - **Balanced computation and I/O (memory) bandwidth**

- **Idea:** Replace a single processing element (PE) with a regular array of PEs and carefully orchestrate flow of data between the PEs
  - such that they collectively transform a piece of input data before outputting it to memory

- **Benefit:** Maximizes computation done on a single piece of data element brought from memory
Why Systolic Architectures?

- Idea: Data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory

- Similar to blood flow: heart $\rightarrow$ many cells $\rightarrow$ heart
  - Different cells “process” the blood
  - Many veins operate simultaneously
  - Can be many-dimensional

- Why? Special purpose accelerators/architectures need
  - Simple, regular design (keep # unique parts small and regular)
  - High concurrency $\rightarrow$ high performance
  - Balanced computation and I/O (memory) bandwidth
Systolic Architectures

- Basic principle: Replace a single PE with a regular array of PEs and carefully orchestrate flow of data between the PEs
  - Balance computation and memory bandwidth

- Differences from pipelining:
  - These are individual PEs
  - Array structure can be non-linear and multi-dimensional
  - PE connections can be multidirectional (and different speed)
  - PEs can have local memory and execute kernels (rather than a piece of the instruction)

Figure 1. Basic principle of a systolic system.
Systolic Computation Example

- **Convolution**
  - Used in filtering, pattern matching, correlation, polynomial evaluation, etc ...
  - Many image processing tasks
  - Machine learning: up to hundreds of convolutional layers in Convolutional Neural Networks (CNN)

\[
\text{Given} \text{ the sequence of weights } \{w_1, w_2, \ldots, w_k\} \\
\text{and the input sequence } \{x_1, x_2, \ldots, x_n\}, \\
\text{compute} \text{ the result sequence } \{y_1, y_2, \ldots, y_{n+1-k}\} \\
\text{defined by} \\
y_i = w_1x_i + w_2x_{i+1} + \cdots + w_kx_{i+k-1}
\]
LeNet-5, a Convolutional Neural Network for Hand-Written Digit Recognition

This is a 1024*8 bit input, which will have a truth table of $2^{8196}$ entries.
Convolutional Neural Networks: Demo

LeNet-5, convolutional neural networks

Convolutional Neural Networks are a special kind of multi-layer neural networks. Like almost every other neural networks they are trained with a version of the back-propagation algorithm. Where they differ is in the architecture.

Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing. They can recognize patterns with extreme variability (such as handwritten characters), and with robustness to distortions and simple geometric transformations.

LeNet-5 is our latest convolutional network designed for handwritten and machine-printed character recognition. Here is an example of LeNet-5 in action.

Many more examples are available in the column on the left:

Several papers on LeNet and convolutional networks are available on my publication page:

[LeCun et al., 1998]  
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.
[page]

[Bottou et al., 1997]  
L. Bottou, Y. LeCun, and Y. Bengio. Global training of

http://yann.lecun.com/exdb/lenet/index.html
Implementing a Convolutional Layer with Matrix Multiplication

Slide credit: Hwu & Kirk
In 2010, Prof. Andreas Moshovos adopted Professor Hwu’s ECE498AL Programming Massively Parallel Processors Class

Several of Prof. Geoffrey Hinton’s graduate students took the course

These students developed the GPU implementation of the Deep CNN that was trained with 1.2M images to win the ImageNet competition
Example: AlexNet (2012)

- AlexNet won ImageNet with more than 10.8% points ahead of the runner up


---

**ImageNet Classification with Deep Convolutional Neural Networks**

Alex Krizhevsky  
University of Toronto  
kriz@cs.utoronto.ca

Ilya Sutskever  
University of Toronto  
ilya@cs.utoronto.ca

Geoffrey E. Hinton  
University of Toronto  
hinton@cs.utoronto.ca

**Abstract**

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.
Example: GoogLeNet (2014)

- Google improves accuracy by **adding more layers**
  - From 8 in AlexNet to 22 in GoogLeNet
  - Szegedy et al., "**Going Deeper with Convolutions**", CVPR 2015.

---

**Going Deeper with Convolutions**

Christian Szegedy\(^1\), Wei Liu\(^2\), Yangqing Jia\(^1\), Pierre Sermanet\(^1\), Scott Reed\(^3\), Dragomir Anguelov\(^1\), Dumitru Erhan\(^1\), Vincent Vanhoucke\(^1\), Andrew Rabinovich\(^4\)

\(^1\)Google Inc. \(^2\)University of North Carolina, Chapel Hill
\(^3\)University of Michigan, Ann Arbor \(^4\)Magic Leap Inc.

\(^1\){szegedy, jiayq, sermanet, dragomir, dmitru, vanhoucke}@google.com
\(^2\)wliu@cs.unc.edu, \(^3\)reedscott@umich.edu, \(^4\)arabinovich@magicleap.com


Deep Residual Learning for Image Recognition

Kaiming He  Xiangyu Zhang  Shaoqing Ren  Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

ImageNet experiments

First CNN

Human: 5.1%
Systolic Computation Example: Convolution (I)

- Convolution
  - Used in filtering, pattern matching, correlation, polynomial evaluation, etc ...
  - Many image processing tasks
  - Machine learning: up to hundreds of convolutional layers in Convolutional Neural Networks (CNN)

Given the sequence of weights \( \{ w_1, w_2, \ldots, w_k \} \) and the input sequence \( \{ x_1, x_2, \ldots, x_n \} \), compute the result sequence \( \{ y_1, y_2, \ldots, y_{n+1-k} \} \) defined by

\[
y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1}
\]
Systolic Computation Example: Convolution (II)

- \( y_1 = w_1x_1 + w_2x_2 + w_3x_3 \)
- \( y_2 = w_1x_2 + w_2x_3 + w_3x_4 \)
- \( y_3 = w_1x_3 + w_2x_4 + w_3x_5 \)

Figure 8. Design W1: systolic convolution array (a) and cell (b) where \( w_i \)'s stay and \( x_i \)'s and \( y_i \)'s move systolically in opposite directions.
Worthwhile to implement adder and multiplier separately to allow overlapping of add/mul executions
Systolic Computation Example: Convolution (IV)

- One needs to *carefully orchestrate* when *data elements are input to the array*
- And when *output is buffered*

- This gets more involved when
  - Array dimensionality increases
  - PEs are less predictable in terms of latency
To a given problem there could be both one- and two-dimensional systolic array solutions. For example, two-dimensional convolution can be performed by a one-dimensional systolic array\textsuperscript{24,25} or a two-dimensional systolic array.\textsuperscript{6} When the memory speed is more than cell speed, two-dimensional systolic arrays such as those depicted in Figure 11 should be used. At each cell cycle, all the I/O ports on the array boundaries can input or output data items to or from the memory; as a result, the available memory bandwidth can be fully utilized. Thus, the choice of a one- or two-dimensional scheme is very dependent on how cells and memories will be implemented.
Combinations

- Systolic arrays can be chained together to form powerful systems.
- This systolic array is capable of producing on-the-fly least-squares fit to all the data that has arrived up to any given moment.

Figure 12. On-the-fly least-squares solutions using one- and two-dimensional systolic arrays, with \( p = 4 \).
Systolic Arrays: Pros and Cons

- **Advantages:**
  - Principled: Efficiently makes use of limited memory bandwidth, balances computation to I/O bandwidth availability
  - Specialized (computation needs to fit PE organization/functions)
    - improved efficiency, simple design, high concurrency/performance
    - good to do more with less memory bandwidth requirement

- **Downside:**
  - Specialized
    - not generally applicable because computation needs to fit the PE functions/organization
More Programmability in Systolic Arrays

- Each PE in a systolic array
  - Can store multiple “weights”
  - Weights can be selected on the fly
  - Eases implementation of, e.g., adaptive filtering

- Taken further
  - Each PE can have its own data and instruction memory
  - Data memory → to store partial/temporary results, constants
  - Leads to stream processing, pipeline parallelism
    - More generally, staged execution
Pipeline-Parallel (Pipelined) Programs

Figure 1. (a) The code of a loop, (b) Each iteration is split into 3 pipeline stages: A, B, and C. Iteration i comprises Ai, Bi, Ci. (c) Sequential execution of 4 iterations. (d) Parallel execution of 6 iterations using pipeline parallelism on a three-core machine. Each stage executes on one core.
Stages of Pipelined Programs

- Loop iterations are divided into code segments called **stages**
- Threads execute stages on different cores

```
loop {
    Compute1
    Compute2
    Compute3
}
```
Figure 3. File compression algorithm executed using pipeline parallelism
Systolic Array: Advantages & Disadvantages

- **Advantages**
  - Makes multiple uses of each data item → reduced need for fetching/refetching → better use of memory bandwidth
  - High concurrency
  - Regular design (both data and control flow)

- **Disadvantages**
  - Not good at exploiting irregular parallelism
  - Relatively special purpose → need software, programmer support to be a general purpose model
Example Systolic Array: The WARP Computer

- HT Kung, CMU, 1984-1988
- Linear array of 10 cells, each cell a 10 Mflop programmable processor
- Attached to a general purpose host machine
- HLL and optimizing compiler to program the systolic array
- Used extensively to accelerate vision and robotics tasks

The WARP Computer

Figure 1: Warp system overview
The WARP Cell

Figure 2: Warp cell data path
**Figure 3.** TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

**Figure 4.** Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

An Example Modern Systolic Array: TPU (II)

As reading a large SRAM uses much more power than arithmetic, the matrix unit uses systolic execution to save energy by reducing reads and writes of the Unified Buffer [Kun80][Ram91][Ovt15b]. Figure 4 shows that data flows in from the left, and the weights are loaded from the top. A given 256-element multiply-accumulate operation moves through the matrix as a diagonal wavefront. The weights are preloaded, and take effect with the advancing wave alongside the first data of a new block. Control and data are pipelined to give the illusion that the 256 inputs are read at once, and that they instantly update one location of each of 256 accumulators. From a correctness perspective, software is unaware of the systolic nature of the matrix unit, but for performance, it does worry about the latency of the unit.

Figure 1. TPU Block Diagram. The main computation part is the yellow Matrix Multiply unit in the upper right hand corner. Its inputs are the blue Weight FIFO and the blue Unified Buffer (UB) and its output is the blue Accumulators (Acc). The yellow Activation Unit performs the nonlinear functions on the Acc, which go to the UB.
TPU: Second Generation

4 TPU chips
vs 1 chip in TPU1

High Bandwidth Memory
vs DDR3

Floating point operations
vs FP16

45 TFLOPS per chip
vs 23 TOPS

Designed for training
and inference
vs only inference
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- Fine-Grained Multithreading
- SIMD Processing (Vector and array processors, GPUs)
Decoupled Access/Execute (DAE)
Decoupled Access/Execute (DAE)

- **Motivation:** Tomasulo’s algorithm too complex to implement
  - 1980s before Pentium Pro

- **Idea:** Decouple operand access and execution via two separate instruction streams that communicate via ISA-visible queues.

Decoupled Access/Execute (II)

- **Compiler generates two instruction streams** (A and E)
  - Synchronizes the two upon control flow instructions (using branch queues)

\[
q = 0.0 \\
\text{Do } 1 \ k = 1, 400 \\
x(k) = q + y(k) * (r * z(k+10) + t * z(k+11))
\]

---

**Access**

- A7 + -400 . negative loop count
- A2 + 0 . initialize index
- A3 + 1 . index increment
- X2 + r . load loop invariants
- X5 + t . into registers

**Execute**

- AEQ + z + 10, A2 .
- AEQ + z + 11, A2 .
- AEQ + y, A2 .
- A7 + A7 + 1 .
- EAQ + AEQ *f X6 .

**Loop:**

- X3 + z + 10, A2 . load z(k+10)
- X7 + z + 11, A2 . load z(k+11)
- X4 + X2 *f X3 . r*z(k+10)-flt. mult.
- X3 + X5 *f X7 . t * z(k+11)
- X7 + y, A2 . load y(k)
- X6 + X3 +f X4 . r*z(x+10)+t*z(k+11))
- X4 + X7 *f X6 . y(k) * (above)
- A7 + A7 + 1 . increment loop counter
- x, A2 + X4 . store into x(k)
- A2 + A2 + A3 . increment index
- JAM Loop . Branch if A7 < 0

---

**Fig. 2a.** Lawrence Livermore Loop 1 (HYDRO EXCERPT)

**Fig. 2b.** Compilation onto CRAY-1-like architecture

**Fig. 2c.** Access and execute programs for straight-line section of loop
Decoupled Access/Execute (III)

- **Advantages:**
  + Execute stream can run ahead of the access stream and vice versa
    + If A is waiting for memory, E can perform useful work
    + If A hits in cache, it supplies data to lagging E
    + Queues reduce the number of required registers
  + Limited out-of-order execution without wakeup/select complexity

- **Disadvantages:**
  -- **Compiler support** to partition the program and manage queues
    -- Determines the amount of decoupling
  -- Branch instructions require synchronization between A and E
  -- Multiple instruction streams (can be done with a single one, though)
Astronautics ZS-1

- Single stream steered into A and X pipelines
- Each pipeline in-order


Loop Unrolling to Eliminate Branches

**Idea:** Replicate loop body multiple times within an iteration

+ Reduces loop maintenance overhead
  - Induction variable increment or loop condition test
+ Enlarges basic block (and analysis scope)
  - Enables code optimization and scheduling opportunities

-- What if iteration count not a multiple of unroll factor? (need extra code to detect this)
-- Increases code size

```c
for (int i = 0; i < N; i++){
}
```
Intel Pentium 4 Simplified

Approaches to (Instruction-Level) Concurrency

- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- Fine-Grained Multithreading
- SIMD Processing (Vector and array processors, GPUs)