Accelerators-I

Tushar Krishna
Associate Professor @ Georgia Tech
Visiting Professor @ MIT EECS and CSAIL
Outline

• Why do we need accelerators?
• Why now?
• How to design accelerators
Outline

• Why do we need accelerators?
• Why now?
• How to design accelerators
Power Constraints in Modern Computers

<< 1W  ~ 1W  ~ 15W  ~ 50W  ~ 100W  ~ 100W
Energy and Power Consumption

- Energy Consumption = $\alpha \times C \times V^2$

- Power Consumption = $\alpha \times C \times V^2 \times f$

Switching activity factor (between 0 to 1)
Capacitance
Voltage
Frequency
# CMOS Scaling (idealized)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Gen X</th>
<th>Gen X+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate Width (Moore’s Law)</td>
<td>1.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Device Area/Capacitance</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Voltage (Dennard’s)</td>
<td>1.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Energy</td>
<td>$\sim 1.0 \times 1.0^2 = 1.0$</td>
<td>$\sim 2 \times 0.7 \times 0.7^2 = 0.65$</td>
</tr>
<tr>
<td>Delay</td>
<td>1.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Frequency</td>
<td>$1/1.0 = 1.0$</td>
<td>$1/0.7 = 1.4$</td>
</tr>
<tr>
<td>Power</td>
<td>$\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$</td>
<td>$\sim 2 \times 0.7 \times 0.7^2 \times 1.4 = 1.0$</td>
</tr>
</tbody>
</table>

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]
CMOS Scaling (idealized)

- Moore’s Law (transistors) + Dennard’s Scaling (voltage)
  - 2.8X in chip capability per generation at constant power
  - ~5000x performance improvement in 20 years

Source: “Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012”
“Power Wall”

- Dennard’s Scaling has stopped
  - Why? *Already operating close to V_{threshold}*
  
  \( \Rightarrow \) Cannot increase operating frequency
  \( (P = \frac{1}{2} CV^2) \)

Source: “Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012"
Technology Trends

48 Years of Microprocessor Trend Data


First inflection: Multicore
Utilization Wall (“Dark Silicon”)

Spectrum of tradeoffs between # of cores and frequency

Example:
65 nm → 32 nm (S = 2)

4 cores @ 1.8 GHz

2x4 cores @ 1.8 GHz
(8 cores dark, 8 dim)
(Industry’s Choice)

4 cores @ 2x1.8 GHz
(12 cores dark)

75% dark after 2 generations;
93% dark after 4 generations

Power Management options?

- **Tackle power across all levels of the computing stack**
  - **Technology**
    - Cost of switching
  - **Circuits**
    - High-speed vs low-power implementation
    - Clock gating and power gating support
    - DVFS
  - **Microarchitecture and ISA**
    - Simplify design
    - Parallelism
    - Heterogeneity and Specialization
  - **Compiler**
    - Instruction footprint and Cache behavior
  - **OS**
    - Tune DVFS and power states
  - **Algorithm:**
    - Switching activity

<table>
<thead>
<tr>
<th>Application / Algorithm</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td>ISA</td>
</tr>
<tr>
<td></td>
<td>Microarchitecture</td>
</tr>
<tr>
<td></td>
<td>Circuits</td>
</tr>
<tr>
<td></td>
<td>Technology (Devices)</td>
</tr>
</tbody>
</table>
Power Management options?

- **Tackle power across all levels of the computing stack**
  - **Technology**
    - Cost of switching
  - **Circuits**
    - High-speed vs low-power implementation
    - Clock gating and power gating support
    - DVFS
  - **Microarchitecture and ISA**
    - Simplify design
    - Parallelism
    - Heterogeneity and Specialization
  - **Compiler**
    - Instruction footprint and Cache behavior
  - **OS**
    - Tune DVFS and power states
  - **Algorithm:**
    - Switching activity
Heterogeneity and Specialization

Improve Energy Efficiency via Customization!

Why?
A modern CPU

Instruction Fetch, Decode, Scheduling and Speculation for “programmability”

Actual Computation

Implicit data management via caches

AMD Zen (2016)
Quantifying this overhead

Embedded Processor Energy Breakdown

- Arithmetic: 24%
- Clock and control: 6%
- Data supply: 28%
- Instruction supply: 42%

Source: Dally et al. Efficient Embedded Computing, IEEE’08
# Performance/Area Benefits

<table>
<thead>
<tr>
<th></th>
<th>GFLOP/s</th>
<th>(GFLOP/s)/mm²</th>
<th>GFLOP/J</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FFT-2^10</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Core i7 (45nm)</td>
<td>67</td>
<td>0.35</td>
<td>0.71</td>
</tr>
<tr>
<td>Nvidia GTX285 (55nm)</td>
<td>250</td>
<td>1.41</td>
<td>4.2</td>
</tr>
<tr>
<td>Nvidia GTX480 (40nm)</td>
<td>453</td>
<td>1.08</td>
<td>4.3</td>
</tr>
<tr>
<td>ATI R5870 (40nm)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Xilinx V6-LX760 (40nm)</td>
<td>380</td>
<td>0.99</td>
<td>6.5</td>
</tr>
<tr>
<td>same RTL std cell (65nm)</td>
<td>952</td>
<td>239</td>
<td>90</td>
</tr>
<tr>
<td><strong>Black-Scholes</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Core i7 (45nm)</td>
<td>487</td>
<td>2.52</td>
<td>4.88</td>
</tr>
<tr>
<td>Nvidia GTX285 (55nm)</td>
<td>10756</td>
<td>60.72</td>
<td>189</td>
</tr>
<tr>
<td>Nvidia GTX480 (40nm)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATI R5870 (40nm)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Xilinx V6-LX760 (40nm)</td>
<td>7800</td>
<td>20.26</td>
<td>138</td>
</tr>
<tr>
<td>same RTL std cell (65nm)</td>
<td>25532</td>
<td>1719</td>
<td>642.5</td>
</tr>
</tbody>
</table>

*Source:* Chung et al., "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?”, MICRO 2010
Outline

• Why do we need accelerators?
• Why now?
• How to design accelerators
Domain-specific Accelerators in SoCs

Apple A8 SoC
The rise of AI

“AI is the new electricity” – Andrew Ng

Object Detection  Image Segmentation  Medical Imaging

Speech Recognition  Text to Speech  Recommendations  Games

Text  Speech
AI Compute Demands Growing Exponentially

AlexNet to AlphaGo Zero: A 300,000x Increase in Compute

Deep and steep

Computing power used in training AI systems
Days spent calculating at one petaflop per second*, log scale

By fundamentals
- Language
- Speech
- Vision
- Games
- Other

AlphaGo Zero becomes its own teacher of the game Go

AlexNet, image classification with deep convolutional neural networks

Two-year doubling (Moore’s Law)

Perceptron, a simple artificial neural network

First era

Modern era

Source: OpenAI
The Economist


December 4, 2023
Cloud SW Companies Building HW

Google’s dedicated TensorFlow processor, or TPU, crushes Intel, Nvidia in inference workloads

By Joel Hruska on April 6, 2017 at 9:48 am | 25 Comments

How Amazon is racing to catch Microsoft and Google in generative A.I. with custom AWS chips

PUBLISHED SAT. AUG 12 2023·9:00 AM EDT | UPDATED MON. AUG 21 2023·7:40 PM EDT

Meta announces AI training and inference chip project

By Katie Paul and Stephen Nellis
May 18, 2023 4:34 PM EDT · Updated 7 months ago

Microsoft announces custom AI chip that could compete with Nvidia

PUBLISHED WED, NOV 15 2023·11:00 AM EST | UPDATED WED, NOV 15 2023·3:05 PM EST
HW Beyond Cloud Computing

MUSK SAYS TESLA IS BUILDING ITS OWN CHIP FOR AUTOPILOT

Elon Musk disclosed plans for Tesla to design its own chip to power its self-driving function.

Surprise! The Pixel 2 is hiding a custom Google SoC for image processing

Google’s 8-core Image Processing Unit will be enabled with Android 8.1.
AI Chips ecosystem

AI Chip Landscape

Tech Giants/System:
- Google
- Microsoft
- Facebook
- AWS
- Apple
- IBM
- Alibaba Group
- Baidu
- Western Digital
- Nokia
- LG

IC Vendor/Fabless:
- Samsung
- NVIDIA
- Qualcomm
- AMD
- Xilinx
- NVIDIA
- MediaTek
- Intel
- Marvell
- Intel
- IBM
- NVIDIA
- Alibaba Cloud
- Baidu
- Western Digital
- Nokia
- LG

Startup in China:
- Canghuan
- BITMAIN
- Graphcore
- SambaNova
- Habana
- Hailo
- thinc
- Kalray
- Gaox
- Tachyum
- Esperanto
- PEZI Computing
- Chipletlli
- Eta Compute
- BrainOne
- Chipola
- Greenwaves
- CPUZ

Startup Worldwide:
- FPGA
- Achronix
- Xilinx
- Processing in Memory
- MYTHIC
- Syntiant
- Hynix
- Optical Computing
- Optimus
- Neurocomputing
- Eligo
- Brainchip
- AI - Benchmark
- AI Matrix
- MLPerf
- ONNX
- TensorFlow
- GLOW
- tvm

All information contained within this infographic is gathered from the internet and periodically updated, no guarantee is given that the information provided is correct, complete, and up-to-date.

More on https://basicmi.github.io/AI-Chip/

Design service with In-house IP:
- ARM
- Synopsys
- Cadence
- CEVA
- ARM
- Arteris
- Ilinx
- Eyeriss
- Project A
- Broadcom
- GUC
- nGraph
- nVidia
- Nervana
- ValleyLab
- DAWNBench
Opportunities

From EE Times – September 27, 2016

“Today the job of training machine learning models is limited by compute, if we had faster processors we’d run bigger models...in practice we train on a reasonable subset of data that can finish in a matter of months. We could use improvements of several orders of magnitude – 100x or greater.”

– Greg Diamos, Senior Researcher, SVAIL, Baidu

ACM’s Celebration of 50 Years of the ACM Turing Award (June 2017)

“Compute has been the oxygen of deep learning”

– Ilya Sutskever, Research Director of Open AI
Demand for Computer Architects

Meta is seeking an ASIC Engineer, Architecture to join their infrastructure organization. Our servers and data centers are the foundation upon which our rapidly scaling infrastructure efficiently operates and upon which our innovative services are delivered. By joining us, you will be an integral member of an ASIC team to build accelerators for some of our top workloads enabling our data centers to scale efficiently. You will have an opportunity to work with AI/Machine Learning (ML) and video codec experts in the company, help architect state-of-the art machine learning accelerators and video transcoders, and contribute to modeling these accelerators. Come work and learn alongside our expert engineers to build “Green” data center accelerators.

ASIC Design Engineer, Machine Learning Accelerator Cores

Job highlights
Identified by Google from the original job post

Responsibilities
• As an ASIC Design Engineer, you will be a part of a team developing ways to accelerate computation in data centers
• You’ll have dynamic, multi-faceted responsibilities in areas such as project definition, design, and implementation
• You will participate in the architecture, documentation, and implementation of data center accelerators
• Define architecture and micro-architecture specifications

Qualifications
• Bachelor’s degree in Electrical Engineering, Computer Science, or equivalent practical experience
• 2 years of industrial experience
• Experience in logic design and functional and Power Performance Area (PPA) closure
• Experience applying engineering practices (e.g., code review, testing, refactoring)

Embedded Software Development Engineer, Machine Learning Accelerators

Job highlights
Identified by Google from the original job post

Responsibilities
• Software / Hardware architecture and co-design
• Embedded software development, testing, and debug
• Test suite and infrastructure development
• Developing software which can be maintained, improved upon, documented, tested, and reused
• Close collaboration with RTL designers, design verification engineers, and other software teams

HW Development Manager, FPGA and ASIC IP design – CSI / Azure – Cloud Server Infrastructure

Microsoft
Bellevue, WA, US
Microsoft is seeking a highly motivated, FPGA and ASIC IP design engineer to build innovative FPGA-based computing systems within a large design team. The group will... careers.microsoft.com
13 connections work here
1 month ago

Physical Design Engineer

Microsoft
Redmond, WA, US
1-2 years of experience in ASIC physical design flows and methodologies. Job responsibilities will entail taking RTL logic through a full ASIC design flow. Worked with toolsets... careers.microsoft.com
13 connections work here
2 weeks ago
Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**
  - Billions of computations
  - Need lots of parallel compute

<table>
<thead>
<tr>
<th>DNN Topology</th>
<th>Number of Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet (2012)</td>
<td>3.98M</td>
</tr>
<tr>
<td>VGGnet-16 (2014)</td>
<td>28.25M</td>
</tr>
<tr>
<td>GoogleNet (2015)</td>
<td>6.77M</td>
</tr>
<tr>
<td>Resnet-50 (2016)</td>
<td>23M</td>
</tr>
<tr>
<td>DLRM (2019)</td>
<td>540M</td>
</tr>
<tr>
<td>Megatron (2019)</td>
<td>8.3B</td>
</tr>
</tbody>
</table>

- Heavy data movement
  - Need to reduce energy

This makes CPUs inefficient

This makes GPUs inefficient
Outline

• Why do we need accelerators?
• Why now?
• How to design accelerators
HW-SW Co-Design

- Target Domain (e.g., AI)
- Key Computation Kernels
- Compute and Memory Behavior
- Hardware Structures
- Design-space Exploration

Constraints (e.g., Area and Power)

Accelerator
Input: Image

Output: "Volvo XC90"

Modified Image Source: [Lee, CACM 2011]
Convolution (CONV2D) Layer

Filter Weights

Input fmaps (N)  Output fmaps (N)

Activations
Convolution (CONV2D) Layer

- Filters
- Input fmap
- Output fmap
- Filter overlay
- Incomplete partial sum
Convolution (CONV2D) Layer
Cycle through input fmap and weights (hold psum of output fmap)

filters

input fmap

output fmap

Filter overlay

Incomplete partial sum
Convolution (CONV2D) Layer
Cycle through input fmap and weights (hold psum of output fmap)
Convolution (CONV2D) Layer

Cycle through input fmap and weights (hold psum of output fmap)
Convolution (CONV2D) Layer

Cycle through input fmap and weights (hold psum of output fmap)

Filters

Input fmap

Output fmap

Filter overlay

Incomplete partial sum
Convolution (CONV2D) Layer

Start processing next output feature activations

filters

input fmap

output fmap

filters

Incomplete partial sum
Convolution (CONV2D) Layer

Cycle through input fmap and weights (hold psum of output fmap)
Convolution (CONV2D) Layer

Cycle through input fmap and weights (hold psum of output fmap)
Convolution (CONV2D) Layer

Cycle through input fmap and weights (hold psum of output fmap)
Representation: Tensors

Rank-0 - Scalar

Rank-1 - Vector

Rank-2 - Matrix

Rank-3 - Cube
Example: Input Act/Fmap Tensor

The compiler “lowers” high-rank tensors into appropriate HW structures
What to accelerate?

Transformer (Language Understanding)

Matrix multiplications (GEMMs) consume around 70% of the total runtime on modern deep learning workloads.

GNMT (Machine Translation)

Prime candidate for acceleration

Possible Pitfall?
Beware of Amdahl’s Law
GEMMs in Deep Learning

**Forward Pass**

Input Activation of Layer $i$ + 1

Model Weight

Output Activation

$(Input \ Activation \ of \ Layer \ i + 1)$

**Backward Pass**

Input Activation (Transpose)

Input Gradient of Layer $i$ + 1

Weight Gradient of Layer $i$

Input Gradient of Layer (i + 1) (Transpose)

Model Weight

Input Gradient of Layer $i$
Spatial (or Dataflow) Accelerators

- **Millions of Parameters (i.e., weights)**
  - Billions of computations

  - Spread computations across thousands of PEs (i.e., parallelism)

  - Heavy data movement

  - Reuse data within the array via dataflow.

- **Features**
  - Thousands of Processing Elements (PEs)
  - Custom Memory Hierarchy (typically scratchpads, no caches)
  - Custom NoCs
What is Reuse?

Matrix A

\[
\begin{array}{cccc}
A1 & A2 & A3 & A4 \\
A5 & A6 & A7 & A8 \\
A9 & A10 & A11 & A12 \\
A13 & A14 & A15 & A16 \\
\end{array}
\]

Matrix B

\[
\begin{array}{cccc}
B1 & B2 & B3 & B4 \\
B5 & B6 & B7 & B8 \\
B9 & B10 & B11 & B12 \\
B13 & B14 & B15 & B16 \\
\end{array}
\]

Matrix C

\[
\begin{array}{cccc}
C1 & C2 & C3 & C4 \\
C5 & C6 & C7 & C8 \\
C9 & C10 & C11 & C12 \\
C13 & C14 & C15 & C16 \\
\end{array}
\]

Example Operations

\[
C1 = A1 \cdot B1 + A2 \cdot B5 + A3 \cdot B9 + A4 \cdot B13
\]

\[
C11 = A9 \cdot B3 + A10 \cdot B7 + A11 \cdot B11 + A12 \cdot B15
\]

\[
C5 = A5 \cdot B1 + A6 \cdot B5 + A7 \cdot B9 + A8 \cdot B13
\]

\[
C11 = A9 \cdot B3 + A10 \cdot B7 + A11 \cdot B11 + A12 \cdot B15
\]
Examples of Data Reuse in DNN

**Convolutional Reuse**

CONV layers only (sliding window)

Reuse:
- Activations
- Filter weights
Examples of Data Reuse in DNN

**Convolutional Reuse**
CONV layers only (sliding window)

**Fmap Reuse**
CONV and FC layers

---

Reuse: Activations
Filter weights
Examples of Data Reuse in DNN

**Convolutional Reuse**
CONV layers only (sliding window)

**Fmap Reuse**
CONV and FC layers

**Filter Reuse**
CONV and FC layers (batch size > 1)

---

**Reuse:**
- **Activations**
- **Filter weights**

---
Why does reuse help?

Suppose
- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

Simple Accelerator:
BW requirement: 3 bytes/cycle => 300 GBps
--> Mem Bound!

Suppose we have data reuse
- weight reused completely
- input reused for 10 cycles
- psums reused for 10 cycles

BW requirement: 20 GBps
→ Compute Bound

Realistic?
How to exploit Reuse?

Weights *reused* across multiple input activations (called “weight stationary” mapping)

*What HW structures would you need?*
Building a DNN Accelerator

<table>
<thead>
<tr>
<th>Memory Read</th>
<th>MAC*</th>
<th>Memory Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>ALU</td>
<td>updated partial sum</td>
</tr>
<tr>
<td>weight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fmap activation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>partial sum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* multiply-and-accumulate
Building a DNN Accelerator

Worst Case: all memory R/W are DRAM accesses

- Example: AlexNet [NeurIPS 2012] has 724M MACs → 2896M DRAM accesses required
Building a DNN Accelerator

Extra levels of local memory hierarchy
Building a DNN Accelerator

Memory Read

DRAM → Mem → ALU → Mem → DRAM

MAC

ALU

Memory Write

DRAM → Mem → DRAM

Extra levels of local memory hierarchy

Opportunities: 1 data reuse
Building a DNN Accelerator

Extra levels of local memory hierarchy

Opportunities: 1 data reuse

1 Can reduce DRAM reads of filter/fmap by up to 500×**

** AlexNet CONV layers
Building a DNN Accelerator

Opportunities: ① data reuse  ② local accumulation

① Can reduce DRAM reads of filter/fmap by up to 500×
② Partial sum accumulation does NOT have to access DRAM
Building a DNN Accelerator

Opportunities:

1. **Data Reuse**
   - Can reduce DRAM reads of filter/fmap by up to **500×**

2. **Local Accumulation**
   - Partial sum accumulation does **NOT** have to access DRAM

   • Example: DRAM access in AlexNet can be reduced from **2896M** to **61M** (best case)
Building a DNN Accelerator

Leverage Parallelism for Throughput!
Building a DNN Accelerator

Leverage Parallelism for \textit{spatial} data reuse!
Spatial DNN Accelerator

Local Memory Hierarchy
- Global Buffer
- Direct inter-PE network
- PE-local memory (RF)

Processing Element (PE)
- Reg File: 0.5 – 1.0 kB
- Control
Hardware structures to exploit reuse

Temporal Reuse

Spatial Reuse

Spatio-Temporal Reuse

Memory Hierarchy / Staging Buffers

Multicasting-support NoCs

Neighbor-to-Neighbor Connections

The availability of the hardware structure limits the “mapping-space”
Design-space of a DNN Accelerator

Dataflow

Mapping Design-Space aka Map Space

HW Design-Space

Number of PEs

Buffer sizes (global/local)

NoCs

Mapping on entire accelerator at time = 1

Spatial Partitioning

Tile Scheduling

Number: Tile IDs

Ordering (aka “stationary”)

Parallelism Dimension

HW or SW depending on flexibility
Putting it all together: Case Study of Google TPUv1 (2016)

“Systolic Array”
Systolic Array Structure

Schematic of MAC PE

MAC Processing Element (PE)

Systolic Array
Difference from CPU Architecture

• Registers distributed across PEs
  – One per operand per PE
    • Can me more than one as well

• Operands “forwarded” from one register to the other
  – More energy-efficient than reading large register file

• Stage data through array in deterministic manner
  – No need for hazard checks etc
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Walkthrough Example
Performance/watt

~200X incremental perf/W of Haswell CPU
~70X incremental perf/W of K80 GPU
Google TPU – large tensor engine vs NVIDIA: multiple tiny tensor engines.
Trade-off?
Design-space of a DNN Accelerator
Dataflow Choices

- **Weight Stationary (WS) Dataflow**
  - Minimize *weight* read energy consumption
  - Maximize convolutional and filter reuse of weights
  - Broadcast *activations* and accumulate *psums* spatially across the PE array

![Diagram of Dataflow Choices](image-url)
Dataflow Choices

- Output Stationary (WS) Dataflow
  - Minimize *partial sum* R/W energy consumption
    - maximize local accumulation

- Broadcast/Multicast *filter weights* and *reuse activations spatially* across the PE array
Dataflow Choices

- Input Stationary (WS) Dataflow
  - Minimize *activation* read energy consumption
  - Maximize convolutional and fmap reuse of activations
  
  - *Unicast weights* and *accumulate psums spatially* across the PE array
Impact of Dataflow and Mappings

480,000 mappings shown

Spread: 19x in energy efficiency

Only 1 is optimal, 9 others within 1%

6,582 mappings have min. DRAM accesses but vary 11x in energy efficiency

VGG conv3 2 Layer. Source: Timeloop

1-level par. 2-level par. 3-level par.

Immense Search space $O(10^{12}) + O(10^{24}) + O(10^{36})$
How to precisely represent dataflows

Loop Nests are the most popular

```
Input Fmaps:  I[G][M][C][H][W]
Filter Weights: W[G][M][C][R][S]
Output Fmaps:  O[G][N][M][E][F]

// DRAM levels
for (g3=0; g3<N3; g3++) {
    for (n3=0; n3<N3; n3++) {
        for (m3=0; m3<M3; m3++) {
            for (f3=0; f3<F3; f3++) {
                // Global buffer levels
                for (g2=0; g2<G2; g2++) {
                    for (n2=0; n2<N2; n2++) {
                        for (m2=0; m2<M2; m2++) {
                            for (f2=0; f2<F2; f2++) {
                                for (c2=0; c2<C2; c2++) {
                                    for (s2=0; s2<S2; s2++) {
                                        // NoC levels
                                        parallel-for (g1=0; g1<G1; g1++) {
                                            parallel-for (n1=0; n1<N1; n1++) {
                                                parallel-for (m1=0; m1<M1; m1++) {
                                                    parallel-for (f1=0; f1<F1; f1++) {
                                                        parallel-for (c1=0; c1<C1; c1++) {
                                                            parallel-for (s1=0; s1<S1; s1++) {
                                                                // SPad levels
                                                                for (e0=0; e0<E0; e0++) {
                                                                    for (n0=0; n0<N0; n0++) {
                                                                        for (e0=0; e0<E0; e0++) {
                                                                            for (r0=0; r0<R0; r0++) {
                                                                                for (c0=0; c0<C0; c0++) {
                                                                                    for (m0=0; m0<M0; m0++) {
                                                                                        O += I * W;
                                                                                    }
                                                                                }
                                                                            }
                                                                        }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```
Summary

• High Throughput requirements and Energy costs of Data Movement are key drivers towards the trend towards custom accelerators

• Heavy HW-SW Co-Design is used in practice to design accelerators

• Open questions: how to avoid “over”-specialization
Thank you!

Next Lecture: Accelerators-II