Maximizing Server Efficiency from μarch to ML accelerators

Michael Ferdman
Maximizing Server Efficiency from μarch to ML accelerators

Michael Ferdman
Maximizing Server Efficiency with ML accelerators

Michael Ferdman
STONY BROOK UNIVERSITY
Toward Efficiency of Cloud Servers

• Number of servers grows rapidly
  – User base increasing exponentially
  – New services appearing daily

• Constant need for more servers
  – Many costs: HW, space, power
  – Want max server efficiency
Achieving Server Efficiency

• Must target two classes of workloads
  – Server software
  – Machine learning
Talk Outline

• Machine Learning Accelerators
  – Fused-Layer CNN Accelerators [MICRO’16]
  – Flexible Buffering [FCCM’17]
  – Resource Partitioning [FPL’16][ISCA’17]

• Server Workloads
  – CloudSuite Benchmarks [ASPLOS’12][TopPicks’14][ISPASS’16]
  – Proactive Instruction Fetch [MICRO’08][MICRO’11]
  – Hardware Memoization [ASPLOS’15]
  – Scale-Out Processors [ISCA’12]
  – Reactive NUCA, Cuckoo Directories [ISCA’09][TopPicks’10][HPCA’10]
  – Cache Bursts, Spatio-Temporal Streaming [MICRO’08][HPCA’09][TopPicks’10]
  – ...
Deep Convolutional Neural Networks

• Revolutionizing machine learning
  – Need fast and efficient evaluation mechanisms

• Lots of computation, good target for accelerators
  – GPUs, FPGAs, ASICs
CNN Accelerator Efficiency

• What we want:
  – All compute units do useful work all the time

• Main challenges:
  – Off-chip data transfer
    • Starves compute units
    • Expensive energy cost
  – Mapping computation to compute units
    • An accelerator has thousands of compute units
    • How to make sure no compute unit is left idle?
Our Contributions

• Reduce off-chip data transfer:
  – Fused-Layer CNN Accelerators [MICRO’16]
  – Flexible Buffering [FCCM’17]

• Increase Compute Unit utilization:
  – Resource Partitioning [ISCA’17]
Feature Map Data in CNN Acceleration

- Layers are computed one after another
- Uses external memory at each layer
- Input and output small, inter-layer data are large
Feature Map Transfer Challenge

- Large amount of data transferred on and off chip
  - e.g., VGGNet-E (includes pooling layers)

Transfers MBs of data on and off chip per image
Fused-Layer CNNs

- Demonstrate existence of inter-layer data locality
- Re-order computation to cache intermediate data
  - Trade on-chip buffer for reduced off-chip transfer
Background: CNN Evaluation

External Memory

Weights

Input Feature Maps
Background: CNN Evaluation

External Memory

Weights

Input Feature Maps

Output Feature Maps
Background: CNN Evaluation

External Memory

Weights

Input Feature Maps

Output Feature Maps
Background: CNN Evaluation

Data goes to external memory in-between layers.
Layer Fusion

- Save memory bandwidth by processing multiple layers

- Challenges:
  - Intermediate data too large to store on chip
  - Layer’s output order different than next layer’s input order

- Approach: Re-arrange CNN evaluation operations to exploit inter-layer locality
Inter-Layer Locality: Sliding Pyramid

Input Feature Maps

Intermediate Feature Maps

Output Feature Maps

Layer 1

Layer 2
Inter-Layer Locality: Sliding Pyramid

Input Feature Maps: tile 1
Intermediate Feature Maps: Layer 1
Output Feature Maps: Layer 2

output pixel 1
Inter-Layer Locality: Sliding Pyramid

Input Feature Maps

Intermediate Feature Maps

Output Feature Maps

Layer 1

Layer 2

Output pixel 1

Output pixel 2
Inter-Layer Locality: Sliding Pyramid

Input Feature Maps

Intermediate Feature Maps

Output Feature Maps

new data
for tile 2

output
pixel 2

output
pixel 1
Inter-Layer Locality: Sliding Pyramid

Input Feature Maps

Intermediate Feature Maps

Output Feature Maps

Layer 1

Layer 2

Store intermediate values on chip
Layer Fusion Exploration Results

No fused layers:
0 KB storage
86 MB transfer

118 KB storage
25 MB transfer

All fused layers:
362 KB storage
3.6 MB transfer

Example: VGGNet-E Layers 1–5

Extra on-chip storage required [KB]

DRAM Transfer [MB]
Fused-Layer Benefits

• Validated concept with prototype
• FPGA: fused-layer vs. FPGA ‘15 [Zhang et al.]
  – 95% transfer reduction (on 5 layers of VGGNet-E)
  – Just 20% extra on-chip memory cost
• CPUs experience ~6x speedup from fusing layers
  – When targeting buffer size ~L1 cache size
• GPUs can similarly do fused-layer
  – BW-limited mobile/embedded GPUs
  – Implementing this can be challenging

Massive BW reduction, works on FPGA, GPU, CPU!
Our Contributions

• Reduce off-chip data transfer:
  – Fused-Layer CNN Accelerators [MICRO’16]
  – Flexible Buffering [FCCM’17]

• Increase Compute Unit utilization:
  – Resource Partitioning [ISCA’17]
Feature Map Transfer Challenge

- Large amount of data transferred on and off chip
  - e.g., VGGNet-E (includes pooling layers)

Transfers MBs of data on and off chip per image
Weight Data Transfer Challenge

- Large amount of data transferred on and off chip
  - e.g., VGGNet-E (includes pooling layers)

Transfers MBs of data on and off chip per image
Weight Bandwidth and Batching

• Weight transfer a limiter of throughput

  Throughput VS. Weight Bandwidth Requirement

  - Throughput(Images/s)
  - Bandwidth(GB/s)

  ![Graph showing Throughput vs. Bandwidth for DDR3(800MHz) and AlexNet]

  - VGGNet-E
  - DDR3(800MHz)
  - AlexNet

• Batching reduces weight transfer
  – Read a weight on chip once, reuse for a batch of images

• Batching is limited in existing accelerators
  – On-chip buffer capacity = # images * buffer size

Need optimized batching to minimize bandwidth
Fully-Connected Layers

Input Vector

Weight Vectors

Output Vector

Input Vector • Weight Vector = Output
Output Buffer Size and Input Retransfer

- Buffer **whole** Output Vector $\rightarrow$ Input Vector read **once**
- Buffer **half an** Output Vector $\rightarrow$ Input Vector read **twice**

Smaller output buffer $\rightarrow$ More Input Vector retransfer
Input vs. Weight Transfer

- Different budgets have different optimal points
- Different layers also have different optimal points

Flexible buffering is needed to be near optimal points
Optimized Batching Contributions

• Batch size affects Input Feature Map retransfer
  – An important trade-off overlooked by existing designs

• Conv layers also need batching
  – Existing designs only consider batching FC layers

• New accelerator design w/flexible buffering: Escher
  – Different layers can use different optimized batch sizes
  – Support batching in both Conv and FC layers

• Escher vs. existing batching: reduce bw. up to 2.4x
• Batching in Conv layers: reduce bw. up to 1.7x
Our Contributions

• Reduce off-chip data transfer:
  – Fused-Layer CNN Accelerators [MICRO’16]
  – Flexible Buffering [FCCM’17]

• Increase Compute Unit utilization:
  – Resource Partitioning [ISCA’17]
CNN Accelerator Underutilization

- State-of-the-art: all compute → one processor
  - Single-CLP (Convolutional Layer Processor)
- Dimension mismatch causes underutilization
  - CNN layers each have different “dimensions” \((N \times R \times C \times M)\)
  - Traditional approach: one fixed-dimension CLP

“One size fits all” → poor compute utilization
Compute-unit Underutilization Problem

• Average utilization 56%

Dimension mismatch $\rightarrow$ poor dynamic utilization
Our Multi-CLP Solution

- Single large CLP → Multiple smaller CLPs
- Each CLP optimized for a subset of layers
- CLP dimensions fit CNN layers

Use specialization to increase dynamic utilization
Throughput Improvement

- Virtex-7 690T, 16-bit fixed-point
- Throughput is proportional to compute utilization

Multi-CLP significantly increases throughput
Scalability Projection

- AlexNet, 32-bit floating point, 100MHz

Multi-CLP benefit scales with FPGA size
CNN Accelerator Conclusions

• Fusing Convolutional Layers
  – Demonstrated new locality in CNN evaluation
  – Use small on-chip memory to save off-chip data transfer

• Batch size $\rightarrow$ Input vs. Weight trade-off
  – Find optimal batch size per layer
  – Benefits batching of Conv and FC layers

• Single-CLP has dynamic underutilization
  – Multi-CLP has more flexibility
  – Each CLP optimized for a subset of layers
Debugging Systems with FPGAs is Difficult

- Involve HW, SW, and OS at the same time
- Painfully slow FPGA compilation process
- Tedious host rebooting due to freezes
- Poor visibility for debugging
State Of The Art

• Testbenches for HW excludes SW and OS
• Existing full-system simulation is targeting SoC ASIC
• HW-SW Co-Simulation excludes OS
• Debug on hardware (synth, place, route, reboot)
The VM-HDL Co-Simulation Framework

Host System
- Software
- Operating System
- Virtual Hardware

FPGA
- Hardware Design
- PCIe Block

Queue

Attach GDB

Save Waveforms
## Run Time Comparison

<table>
<thead>
<tr>
<th></th>
<th>Real System (sec.)</th>
<th>Co-Simulation (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation</td>
<td>-</td>
<td>167</td>
</tr>
<tr>
<td>Synthesis</td>
<td>1617</td>
<td>-</td>
</tr>
<tr>
<td>Place and Route</td>
<td>2672</td>
<td>-</td>
</tr>
<tr>
<td>Reboot</td>
<td>120</td>
<td>-</td>
</tr>
<tr>
<td>Execution</td>
<td>0.000032</td>
<td>6.02</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>≈ 4409 (1h:13m:29s)</td>
<td>≈ 173 (2m:53s)</td>
</tr>
</tbody>
</table>
Talk Outline

• Machine Learning Accelerators
  – Fused-Layer CNN Accelerators [MICRO’16]
  – Flexible Buffering [FCCM’17]
  – Resource Partitioning [FPL’16][ISCA’17]

• Server Workloads
  – CloudSuite Benchmarks [ASPLOS’12][TopPicks’14][ISPASS’16]
  – Proactive Instruction Fetch [MICRO’08][MICRO’11]
  – Hardware Memoization [ASPLOS’15]
  – Scale-Out Processors [ISCA’12]
  – Reactive NUCA, Cuckoo Directories [ISCA’09][TopPicks’10][HPCA’10]
  – Cache Bursts, Spatio-Temporal Streaming [MICRO’08][HPCA’09][TopPicks’10]
  – ...
Thanks!

• Happy to take any questions and comments
  – Now, after the talk, or via email

mferdman@cs.stonybrook.edu