Fast Deep Learning with Caffe and Tensorflow on multi-core CPUs

Victor Jakubiuk
victor@onspecta.com
About me

Victor Jakubiuk

- Research Scientist in Computational Neuroscience at MIT
- Co-founder & Chief Scientific Officer of OnSpecta

- Performance engineering researcher
- High-performance computing, graph theory, neural networks, concurrency & data structures

- B.S. & M.S. at MIT/CSAIL
- Co-founder & CTO of a fintech startup (Python + Excel)

victor@onspecta.com
http://jakubiuk.net
@victorjakubiuk
The GPU Story

“ImageNet Classification with Deep Convolutional Neural Networks”

- A. Krizhevsky, I. Sutskever, G. Hinton

2009
The GPU Story
## The GPU Story

<table>
<thead>
<tr>
<th>GPU</th>
<th>cuDNN</th>
<th>Forward (ms)</th>
<th>Backward (ms)</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 1080 Ti</td>
<td>5.1.10</td>
<td>41.23</td>
<td>86.91</td>
<td>128.14</td>
</tr>
<tr>
<td>Pascal Titan X</td>
<td>5.1.05</td>
<td>41.59</td>
<td>87.03</td>
<td>128.62</td>
</tr>
</tbody>
</table>

...  

<table>
<thead>
<tr>
<th>GPU</th>
<th>Forward (ms)</th>
<th>Backward (ms)</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 1080</td>
<td>None</td>
<td>143.73</td>
<td>379.09</td>
</tr>
<tr>
<td>Maxwell Titan X</td>
<td>None</td>
<td>172.61</td>
<td>415.87</td>
</tr>
<tr>
<td>CPU: Dual Xeon E5-2630 v3</td>
<td>None</td>
<td>3101.76</td>
<td>5393.72</td>
</tr>
</tbody>
</table>
# The GPU Story

## Tesla K80 Benchmark Results

<table>
<thead>
<tr>
<th></th>
<th>AlexNet</th>
<th>Overfeat</th>
<th>GoogLeNet</th>
<th>VGG (ver.a)</th>
<th>Speedup Over CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Caffe</strong></td>
<td>365</td>
<td>1,187</td>
<td>1,236</td>
<td>1,747</td>
<td>(9x – 15x speedups)</td>
</tr>
<tr>
<td><strong>TensorFlow</strong></td>
<td>181</td>
<td>622</td>
<td>979</td>
<td>1,104</td>
<td>(4x – 10x speedups)</td>
</tr>
<tr>
<td><strong>Elfin</strong></td>
<td>565</td>
<td>1,716</td>
<td>1,729</td>
<td></td>
<td>(4x – 1x speedups)</td>
</tr>
</tbody>
</table>
Really?

- Caffe
- TensorFlow

© 2017 OnSpecta
Really?

AlexNet (200MB) – first try

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Cores</th>
<th>Caffe (GPU)</th>
<th>Caffe</th>
<th>OnSpecta DLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU (K80)</td>
<td></td>
<td>0.54</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CPU (Haswell)</td>
<td>1</td>
<td>X</td>
<td>27.5</td>
<td>6.8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>X</td>
<td>28.1</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>X</td>
<td>26.6</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>X</td>
<td>28.3</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>X</td>
<td>28.3</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Inference (forward pass), 100x
Connectomics
Microscope

- 61-beams, high-resolution electron microscope
- 4nm x 4nm per pixel; 1mm^2 -> 62GB
- 27nm slice; 1mm^3 -> 2PB
- 1.22 Gpixel/s ~ 1TB/hr
Data processing pipeline

(1) Gray-scale EM images
(2) Pixel Classification
(3) Oversegmentation
(4) Agglomeration
(5) Skeletonization

© 2017 OnSpecta
Data processing pipeline
Bottlenecks

- **Pixel classification**
  - Random Forest classifier: faster, but less accurate
  - Convolutional Neural Network: more accurate, but slower

* Workload break-down after optimizations
Why not the GPU?

- Complex software pipeline
- At scale would require multiple GPUs
  - Resulting in data distribution issues
  - Expensive (GPU, storage, networking, space etc.)
- All neural network frameworks use cuDNN
  - Optimized matrix multiplications, but hard to modify
- GPUs are harder to program

Why the CPU?

- Readily available
- Easy to program & integrate with the rest of the pipeline
- Easier to achieve higher practical utilization ("end-to-end")
This One Weird Trick…

- Will make your CPU run as fast as the GPU!
5 Performance Engineering “Tricks”

- Avoid repetitive computations
- Optimize network architecture for speed
- SIMD Parallelization (“Vectorized instructions”)
- Easy-to-use & efficient concurrency with CILK
- Memory optimization
#1 CNN – Sliding Window
#1: CNN – Sliding Window
#1 CNN – Sliding Window
#1 CNN – Sliding Window
#1 CNN – Theoretical Speedup

\[ C_l = s_l^2 \cdot |P_{l-1}| \cdot |P_l| \cdot w_l^2 \cdot k_l^2 \cdot 2 \]

\[ C_l = s_{x,l} \cdot s_{y,l} \cdot |P_{l-1}| \cdot |P_l| \cdot F_l \cdot k_l^2 \cdot 2 \]

| Layer (\( l \)) | \( s \) | \( s_{l-1} \) | \( |P_{l-1}| \) | \( |P_l| \) | \( w_l \) | \( k_l \) | \( F_l \) | FLOPS\(_l\)\text{\(^{\text{patch}} \)\( \cdot 10^9 \)} | FLOPS\(_l\)\text{\(^{\text{image}} \)\( \cdot 10^9 \)} | speedup |
|----------------|------|-------------|-----------------|-----------------|--------|--------|--------|-----------------|-----------------|--------|
| 1              | 512  | 559         | 1               | 48              | 92     | 4      | 1      | 3408            | 0.5             | 7114.8 |
| 3              | 512  | 279         | 48              | 48              | 42     | 5      | 4      | 53271           | 35.9            | 1485.1 |
| 5              | 512  | 139         | 48              | 48              | 18     | 4      | 16     | 6262            | 22.8            | 274.7  |
| 7              | 512  | 69          | 48              | 48              | 6      | 4      | 64     | 695             | 22.5            | 30.9   |
| Total          |      |             |                 |                 |        |        |        | 63636           | 81.6            | 779.8  |
#2: Network Architecture

| Layer $(l)$ | Type                | Maps ($|P_l|$) and neurons                          | Kernel $(k_l \times k_l)$ |
|-------------|---------------------|-------------------------------------------------|---------------------------|
| 0           | input               | 1 map of 95x95 neurons                          |                           |
| 1           | convolutional       | 48 maps of 92x92 neurons                        | 4x4                       |
| 2           | max pooling         | 48 maps of 46x46 neurons                        | 2x2                       |
| 3           | convolutional       | 48 maps of 42x42 neurons                        | 5x5                       |
| 4           | max pooling         | 48 maps of 21x21 neurons                        | 2x2                       |
| 5           | convolutional       | 48 maps of 18x18 neurons                        | 4x4                       |
| 6           | max pooling         | 48 maps of 9x9 neurons                          | 2x2                       |
| 7           | convolutional       | 48 maps of 6x6 neurons                          | 4x4                       |
| 8           | max pooling         | 48 maps of 3x3 neurons                          | 2x2                       |
| 9           | fully connected     | 200 neurons                                    | 1x1                       |
| 10          | fully connected     | 2 neurons                                      | 1x1                       |
N3

4-layer network ("ShortNet3")
#2: Network Architecture

Table 5.1: Network Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Training time</th>
<th>Size (MB)</th>
<th>Pixel Accuracy</th>
<th>GPU Latency (s)</th>
<th>CPU Latency (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>3h 2min</td>
<td>45</td>
<td>81%</td>
<td>3.27</td>
<td>41.53</td>
</tr>
<tr>
<td>AlexNet</td>
<td>14 min</td>
<td>80</td>
<td>81%</td>
<td>3.34</td>
<td>37.54</td>
</tr>
<tr>
<td>ShortNet1</td>
<td>11 min</td>
<td>225</td>
<td>81%</td>
<td>3.10</td>
<td>83.55</td>
</tr>
<tr>
<td>ShortNet2</td>
<td>22 min</td>
<td>150</td>
<td>78%</td>
<td>2.73</td>
<td>58.43</td>
</tr>
<tr>
<td>ShortNet3</td>
<td>38 min</td>
<td>630</td>
<td>77%</td>
<td>5.69</td>
<td>222.85</td>
</tr>
<tr>
<td>BN1</td>
<td>2 min</td>
<td>0.12</td>
<td>68%</td>
<td>1.79</td>
<td>1.98</td>
</tr>
<tr>
<td>BN2</td>
<td>2 min</td>
<td>0.12</td>
<td>50%</td>
<td>1.46</td>
<td>1.36</td>
</tr>
</tbody>
</table>
Not just matrix multiplication, look at the structure!
- AVX2 instructions need to have aligned memory addresses
- Hard to slide a convolution window in 2D, but...
- Most conv layers have channels (3D) -> vectorized over channel!
- Optimally re-arrange memory across the entire framework
- Naïve loop inefficient:
  - Fully utilize all registers (16) over 6 concurrent convolutions
  - Interleave register LOAD, FMA and STORE ops
- Specific algorithm (or, convolutional kernel) based on layer type & CPU type
- Results in 70-80% utilization
#4: Efficient concurrency

- CILK (Cilk Plus/Cilk++) = “nice threads”
- General purpose work-stealing scheduler, simple “fork-join” primitive
- Double-ended work queues
- Supported by GCC and the Intel Compiler
- Basic building blocks:
  - Hyperobjects
  - `cilk_spawn f(x)`
  - `cilk_sync;`
  - `cilk_for (int i = 0; i < N; i++)` ...

- Linear scaling across multi-core CPUs...
- As long as the data iterated over fits into L3 cache!
  - Large matrix multiplication must be executed over sub-matrices
Benchmarks

Image data throughput

Skylake (4 cores)

Haswell (18 cores)

Throughput (MB/s)

Threads

8 channels

32 channels

(a) XNN  (b) Caffe  (c) ZNN
#5: 2-layer Memory Buffer

- Throughput, only forward pass
- Statically allocate two large matrix buffers (input & output)
  - Before the network’s execution
- Swap them in between layers
- Memory bound by the largest layer
- No dynamic allocation, reduced cache trashing and OS paging

Other:
- Bind threads to cores (eliminates NUMA issues)
## Benchmarks

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>8-channel MaxoutNet Throughput (MB/s)</th>
<th>32-channel MaxoutNet Throughput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XNN</td>
<td>CPU (72-core Haswell)</td>
<td>111.1</td>
<td>16.67</td>
</tr>
<tr>
<td>gpuZNN</td>
<td>GPU (Titan X)</td>
<td>67.61</td>
<td>25.28</td>
</tr>
<tr>
<td>Neon</td>
<td>GPU (Titan X)</td>
<td>37.06</td>
<td>(exceeds memory)</td>
</tr>
<tr>
<td>XNN</td>
<td>CPU (4-core Skylake)</td>
<td>11.49</td>
<td>1.74</td>
</tr>
</tbody>
</table>
Summary

- Writing multi-core code is hard: “one weird trick” doesn’t exist.
- Combining algorithmic speed-up and “piercing” thru logical abstraction layers saves computations
- General purpose frameworks don’t fully utilize peak FLOPS
- Not polluting L3 cache is crucial
- SIMD/vectorization is hard
- CILK simplifies parallelization
- With careful engineering, performance is on par!
- These techniques apply to training as well

- We’ll have Connectome in our lifetime!
  - Due to software improvements, not just Moore’s/hardware
Caffe Framework Optimization

- General purpose optimizations, work across variety of networks
- Initial focus on computer vision applications
# Intel Haswell CPU

**AlexNet (200MB)**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Cores</th>
<th>Caffe (GPU)</th>
<th>Caffe</th>
<th>OnSpecta DLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU (K80)</td>
<td></td>
<td>0.54</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CPU (Haswell)</td>
<td>1</td>
<td>X</td>
<td>27.5</td>
<td>6.8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>X</td>
<td>28.1</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>X</td>
<td>26.6</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>X</td>
<td>28.3</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>X</td>
<td>28.3</td>
<td>0.6</td>
</tr>
</tbody>
</table>

*Inference (forward pass), 100x*
# Intel Haswell CPU

**GoogleNet (41MB)**

Inference (forward pass), 100x

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Cores</th>
<th>Caffe</th>
<th>OnSpecta DLS</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-A53</td>
<td>1</td>
<td>73.9</td>
<td>32.5</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>61.3</td>
<td>14.2</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>59.3</td>
<td>9.6</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>55.4</td>
<td>6.8</td>
<td>8.2</td>
</tr>
</tbody>
</table>
Server Side Applications

- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform

- Automotive
- Healthcare
- Finance
- HPC Simulations
“You’ve got cars, you’ve got drones, you’ve got microphones; in the future almost every electronic device will have deep learning inferencing within it. We call that AI at the edge,” he said “And eventually there’ll be a trillion devices out there; Vending machines, every camera, every house will have deep learning capability.”
Hardware approach – TPU

- Various ASICs, Google's: Tensorflow Processing Unit; Nervana; Movidius
OnSpecta

Software Deep Learning Optimization Across Chips

DLS Platform – Inference Initially
Performance Acceleration, Hardware Virtualization

x86 CPUs  
TPUs & Deep Learning Chipsets  
GPUs  
ARM CPUs
References

- “A Multicore Path to Connectomics-on-Demand”
- “A Multi-Pass Approach to Large-Scale Connectomics”
- “High Performance Data Processing Pipeline for Connectome Segmentation”
- “Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU”
- https://www.intelnervana.com/framework-optimizations/
Questions?

Victor Jakubiuk
victor@onspecta.com
Backup slides
Toolset

- Profiler
- Network visualization
- Dynamic parameter adjustment