Fast ConvNets with fbfft
A GPU Performance Evaluation

Facebook AI Research

Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, Yann LeCun
Facebook AI Research
7th May, 2015
## Agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td><strong>Introduction</strong></td>
</tr>
<tr>
<td><strong>2</strong></td>
<td><strong>Contributions</strong></td>
</tr>
<tr>
<td><strong>3</strong></td>
<td><strong>Upcoming Work</strong></td>
</tr>
<tr>
<td><strong>4</strong></td>
<td><strong>Numbers</strong></td>
</tr>
</tbody>
</table>
Introduction
Convolution
Convolutional Neural Networks

- Convolutional layers computationally expensive
- Main reason for justifying GPUs

Figure from Sermanet et. al., ICPR-12
Fourier Transform

Public Domain animation from Wikipedia
Convolution using Fourier Transform

\[ y(s,j) = \sum_{i \in f} x(s,i) \star w(j,i) = \sum_{i \in f} \mathcal{F}^{-1} \left( \mathcal{F}(x(s,i)) \circ \mathcal{F}(w(j,i))^* \right) \]

- Convolution Theorem
  - In Fourier basis, pointwise multiplications
- FFT with Cooley-Tuckey: \( O(n^2) \rightarrow O(n \cdot \log n) \)
Contributions
Contributions

- Convolutions as composition of FFT, transpose and GEMM
  - Implementation based on NVIDIA libraries + Auto-Tuner
- High Performance FBFFT and FBMM for our domain
- Bandwidth-bound (at least on GPUs)
  - Unlike convolutions in spatial domain
  - We increase the memory BW requirements
    - Tiling moves communication from main memory to caches
- Moved the ceiling of achievable performance
  - Now focus on optimization
### Convolutions as composition of operations

<table>
<thead>
<tr>
<th><strong>INPUT</strong></th>
<th><strong>OUTPUT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>InS ( f \times h \times w )</td>
<td>FFT2D ( S \times f \times (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) )</td>
</tr>
<tr>
<td>Wei ( f' \times f \times k_h \times k_w )</td>
<td>FFT2D ( f' \times f \times (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) )</td>
</tr>
<tr>
<td>( InFT \ (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) \times S \times f )</td>
<td>Trans2D ( InFT \ (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) \times S \times f' )</td>
</tr>
<tr>
<td>( WeiFT^* \ (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) \times f' \times f )</td>
<td>Cgemm ( OutFT \ (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) \times S \times f' )</td>
</tr>
<tr>
<td>( OutFT \ (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) \times S \times f' )</td>
<td>Trans2D ( OutFT \ (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) \times S \times f' )</td>
</tr>
<tr>
<td>( Out \ S \times f' \times (h+p_h) \times (\lfloor \frac{w+p_w}{2} \rfloor + 1) )</td>
<td>IFFT2D ( Out \ S \times f' \times (h-k_h+1) \times (w-k_w+1) )</td>
</tr>
</tbody>
</table>
Fast convolutions using cuFFT + cuBLAS

- **Choosing between**
  - Batched vs iterated cuBLAS calls
  - Best FFT interpolation sizes (cuFFT only) vs FBFFT
    - Efficiency vs additional multiplications
  - FBMM vs cuBLAS transpose + cublas GEMM
    - Efficiency vs additional memory consumption

- **Auto-tuning**
  - Construct small search space, traverse exhaustively
  - Enough for our purposes
The need for specialized FFT implementation

- cuFFT not suited for ConvNet regimes
  - Tuned for HPC and DSP applications, large FFTs
  - Convolutional nets need many small FFTs
- cuFFT needs explicit zero-padding
- cuFFT / cuBLAS are closed-source
  - Cannot try new ideas or even implicit zero-padding
- Extra time / memory wasted on data layout transpose
**FBFFT**

- Implementation views a GPU as a wide vector
  - Exchanges data using shuffles
  - Avoids shared memory
  - Heavy use of registers

- Compute twiddle factors using trigonometric symmetries

- Actually limited by numbers of shuffle operations
  - Not by memory BW
  - Not by compute
Memory Consumption

- **Tradeoff: parallelism / efficiency / reuse and memory bloat**
  - We can make them arbitrary small
  - Given a memory budget, get the best performance, across layers

- **Single layer problem: all buffers must fit in memory**
  - Reuse buffers across all layers, no reuse of FT values
    - ~9x the largest layer with cuBLAS / cuFFT, 3x with FBFFT / FBMM
  - Large inputs problematic (common Fourier interpolation basis) -> tiling

- **Multi-layer problem**
  - Exploit reuse between FT, dependences are long (2 long, 1 short)
Key insights

• For kernels $\leq 15 \times 15$, you only need $16 \times 16$ or $32 \times 32$ FFTs

• Whatever the kernel size, cost is the same
  ▪ True until you need a larger Fourier interpolation basis
    ▪ Then tiling kicks in

• Algorithm $\gg$ Optimization

• Main memory BW limited
  ▪ Work towards cache BW limited
  ▪ Significant room for improvement (float16)
Numbers
(as of December 2014)
Speedup (CuFFT + CuBLAS) over CuDNN (R1)

Figure 1: 3 × 3 kernel (K40m)

Figure 2: 5 × 5 kernel (K40m)
Speedup (CuFFT + CuBLAS)

Figure 3: 7 × 7 kernel (K40m)

Figure 4: 9 × 9 kernel (K40m)
Speedup (CuFFT + CuBLAS)

Figure 5: $11 \times 11$ kernel (K40m)

Figure 6: $13 \times 13$ kernel (K40m)
Speedup (FBFFT vs CuFFT)

**Figure 8:** `fbfft-2D FFT and IFFT (K40m, cuFFT 6.5 @ 1x)`
# Comparison on Imagenet Networks

**AlexNet (One Weird Trick paper)** - Input 128x3x224x224

<table>
<thead>
<tr>
<th>Library</th>
<th>Class</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NervanaSys-16</td>
<td>ConvLayer</td>
<td>97</td>
<td>30</td>
<td>67</td>
</tr>
<tr>
<td>NervanaSys-32</td>
<td>ConvLayer</td>
<td>109</td>
<td>31</td>
<td>78</td>
</tr>
<tr>
<td>fbfft</td>
<td>SpatialConvolutionCuFFT</td>
<td>136</td>
<td>45</td>
<td>91</td>
</tr>
<tr>
<td>cudaconvnet2*</td>
<td>ConvLayer</td>
<td>177</td>
<td>42</td>
<td>135</td>
</tr>
<tr>
<td>CuDNN (R2) *</td>
<td>cudnn.SpatialConvolution</td>
<td>231</td>
<td>70</td>
<td>161</td>
</tr>
<tr>
<td>Caffe (native)</td>
<td>ConvolutionLayer</td>
<td>324</td>
<td>121</td>
<td>203</td>
</tr>
<tr>
<td>Torch-7 (native)</td>
<td>SpatialConvolutionMM</td>
<td>342</td>
<td>132</td>
<td>210</td>
</tr>
</tbody>
</table>
## Comparison on Imagenet Networks

**Overfeat [fast] - Input 128x3x231x231**

<table>
<thead>
<tr>
<th>Library</th>
<th>Class</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NervanaSys-16</td>
<td>ConvLayer</td>
<td>364</td>
<td>119</td>
<td>245</td>
</tr>
<tr>
<td>NervanaSys-32</td>
<td>ConvLayer</td>
<td>410</td>
<td>126</td>
<td>284</td>
</tr>
<tr>
<td>fbfft</td>
<td>SpatialConvolutionCuFFT</td>
<td>407</td>
<td>139</td>
<td>268</td>
</tr>
<tr>
<td>cudaconvnet2*</td>
<td>ConvLayer</td>
<td>723</td>
<td>176</td>
<td>547</td>
</tr>
<tr>
<td>CuDNN (R2) *</td>
<td>cudnn.SpatialConvolution</td>
<td>810</td>
<td>234</td>
<td>576</td>
</tr>
<tr>
<td>Caffe</td>
<td>ConvolutionLayer</td>
<td>823</td>
<td>355</td>
<td>468</td>
</tr>
<tr>
<td>Torch-7 (native)</td>
<td>SpatialConvolutionMM</td>
<td>878</td>
<td>379</td>
<td>499</td>
</tr>
</tbody>
</table>
## Comparison on Imagenet Networks

**OxfordNet [Model-A] - Input 64x3x224x224**

<table>
<thead>
<tr>
<th>Library</th>
<th>Class</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NervanaSys-16</td>
<td><strong>ConvLayer</strong></td>
<td>530</td>
<td>166</td>
<td>364</td>
</tr>
<tr>
<td>NervanaSys-32</td>
<td><strong>ConvLayer</strong></td>
<td>629</td>
<td>173</td>
<td>456</td>
</tr>
<tr>
<td>fbfft</td>
<td><strong>SpatialConvolutionCuFFT</strong></td>
<td>1092</td>
<td>355</td>
<td>737</td>
</tr>
<tr>
<td>cudaconvnet2*</td>
<td><strong>ConvLayer</strong></td>
<td>1229</td>
<td>408</td>
<td>821</td>
</tr>
<tr>
<td>CuDNN (R2) *</td>
<td><strong>cudnn.SpatialConvolution</strong></td>
<td>1099</td>
<td>342</td>
<td>757</td>
</tr>
<tr>
<td>Caffe</td>
<td><strong>ConvolutionLayer</strong></td>
<td>1068</td>
<td>323</td>
<td>745</td>
</tr>
<tr>
<td>Torch-7 (native)</td>
<td><strong>SpatialConvolutionMM</strong></td>
<td>1105</td>
<td>350</td>
<td>755</td>
</tr>
</tbody>
</table>
Hot From The Press

• **Updated numbers:**
  - Tiled FFT
  - Implicit padding
  - Buffer reuse and memory management strategies
  - Asynchrony for better utilization
  - Faster FFT (precomputed coefficients)

• **Discuss at our poster session on Saturday**
  - Saturday May 9\textsuperscript{th}, 10:30am – 1:30pm
Questions?