EFFICIENT PRIMITIVES FOR DL
Sharan Chetlur,
Tech Lead - cuDNN and cuBLAS

NVIDIA
HPC COMPUTATION STACK

BLAS interface

Applications (Fluid Dynamics, Computational Physics ...)

BLAS standard interface

Various CPU BLAS implementations

- Intel CPUs
- IBM Power

cuBLAS/NVBLAS

- Kepler GPU
- Pascal GPU
- Volta GPU
DL COMPUTATION STACK
Low level primitives for DL

Data Scientists

Frameworks

cuDNN

cuBLAS

Kepler
Maxwell
Pascal
Volta
Future
cuDNN ADOPTION

cuDNN Downloads Over Time

cuDNN Downloads By Application

03/16-03/17
YEAR IN REVIEW

RNNs & Fast Convolutions
- LSTM / GRU
- 3D FFT Tiling
- Spatial Transformer Layer
- 3x3 Winograd convolution

Pascal Support
- 5x5 and faster 3x3 Winograd convolutions
- Mixed precision GEMMs
- Improved Winograd accuracy
- Faster RNN GEMMs

Small Batch Training & Inference
- Fused Conv+Bias+ReLU
- Int8 convolutions
- Persistent RNNs
- Dilated convolutions
- Deterministic Max Pooling

v5 (Q2 2016)
v5.1, cuBLAS 8 (Q3 2016)
v6 (Q1 2017)
CONVOLUTIONS
THE CONVOLUTION PROBLEM
Workhorse of CNNs

Circa 2014:

• Everybody is using GPUs to train CNNs
• Convolutions take up 80-90% of people’s runtimes
• Can NVIDIA do something about this?
CONVOLUTION OPERATION

Input Image  Input Filter  Intermediate output  Final Output

Why do it once if you can do it n times? Batch the whole thing.
PARAMETER SPACE

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>$n$</td>
</tr>
<tr>
<td>Input Channels</td>
<td>$c$</td>
</tr>
<tr>
<td>Image height x Image width</td>
<td>$h \times w$</td>
</tr>
<tr>
<td>Output Channels</td>
<td>$k$</td>
</tr>
<tr>
<td>Filter height x filter width</td>
<td>$r \times s$</td>
</tr>
<tr>
<td>Zero Padding</td>
<td>$pad_h \times pad_w$</td>
</tr>
<tr>
<td>Filter Striding</td>
<td>$u \times v$</td>
</tr>
</tbody>
</table>
ISSUES
Why is this difficult?

- Large parameter space; highly variable shapes and sizes
- Computation-bound
- User cares about a small subset, run repeatedly
- GPU arch changes often
EFFICIENT DIRECT CONVOLUTIONS
AN ALTERNATIVE APPROACH

A Header file library

- An idea we considered and rejected - similar to Thrust
- Specialized for each filter size
- Code like:

```c++
int r = 11;  int s = 11;
thrust::device_vector<float> kernel_store(k * c * s * r);
typedef texture_stack<float>::type tx_stk;
tx_stk kern(thrust::raw_pointer_cast(kernel_store.data()), k, c, s, r);
convolve<11, 11>(src, kern, dest);
```
EFFICIENT GEMM ON GPUs

A two-stage pipeline

A (gmem) → A tile → FP unit → B tile → B (gmem)

A tile

B tile

C tile (eventually written to gmem)
CONVOLUTION AS AN IMPLICIT GEMM

Modified GEMM pipeline

Filters (gmem) → FP unit → B tile → C tile (eventually written to gmem)

A tile → FP unit

Image (gmem)

Gather logic
CONCLUSIONS ON IMPLICIT GEMM

Costs/Benefits

• Benefits:
  • Deliver perf in an scalable manner across parameter space
  • Leverage a lot of painstakingly optimized code from GEMM
  • Basic idea has served us well, across 4 generations of HW
  • Low memory overhead

• Costs:
  • More memory traffic
  • ‘Unfriendly’ layouts can throttle perf
FAST CONVOLUTIONS
ALGORITHMICALLY FASTER CONVOLUTIONS

FFT and Winograd

Circa 2015:

• Direct methods for convolutions tapped out
• Convolutions still most significant percent of workload
• Algorithmic improvements long known in Signal Processing

Key references:

• ‘Arithmetic Complexity of Computations’ (1980) - Winograd S
• ‘Fast Algorithms for Convolutional Neural Networks’ (2015) - Lavin A, Gray S
CONVOLUTION THEOREM AND FFTS

How it works

Say \( \text{Image} \ast \text{Filter} = \text{Output} \) as described before (“\( \ast \)” is convolution)

\[
\text{FFT(Image} \ast \text{Filter}) = \text{FFT(Output)}
\]

\[
\text{FFT(Image} \ast \text{Filter}) = \text{FFT(Image)} \ast \text{FFT(Filter)} ; \text{“\( \ast \)” is a point-wise multiply}
\]

\[
\therefore \text{Output} = \text{FFT}^{-1}(\text{FFT(Image)} \ast \text{FFT(Filter)})
\]

For a signal of length \( n \) in 1D, FFT evaluation takes \( O(n \log n) \) time

\[
\text{Output} = \text{FFT}^{-1}(\text{FFT(Image)} \ast \text{FFT(Filter)})
\]

FFTs take \( O(n \log n) \) time, and the point-wise multiply takes \( O(n) \) time

Reuse cost of transform across batch and output feature maps
WINOGRAD CONVOLUTIONS

A different transform

Follows a similar scheme

Output = Inverse Transform (Forward Transform (Image) * Forward Transform (Filter))

Asymptotic complexity same as FFT-convolution, but the transforms are real valued, so total FLOP count smaller

Cost of smaller FLOP count is reduced numeric accuracy

PERSISTENT RNNs
RECURRENT NEURAL NETWORKS
A different breed of DNN

• RNNs have been in use for several decades
• Conventional wisdom: GEMMs are the main workload, already well optimized
• Can we do better?
• 2016, cuDNN 5 introduced RNN support
• Substantial speedups from intelligent scheduling and simple fusion
INTRO TO RNNs
First pass optimizations

ISSUES

The small-batch problem

For large batches, GEMMs are compute bound

Still account for major fraction of the execution time

Increasing preference for small batches, distributed training

Large batches not always an option for inference

As batch size decreases, GEMMs pushed toward memory bound regime
PERSISTENT RNNs

Exploiting reuse of weight matrix

- Weight matrix size: Hidden Size * Hidden Size
- Activations size: Hidden Size * Batch
- Weight matrix accounts for majority of memory traffic per timestep
- Load once into register, reuse across timesteps
- Allows us to move from memory bound, to compute bound
PERSISTENT RNN EXECUTION

Block diagram

- All time steps of a layer done in one kernel
- Need to guarantee that all blocks launched are scheduled and executing
- Impossible pre-Pascal. Tricky on Pascal. Volta onwards, CG allows this formally.
CHALLENGES FOR THE FUTURE
COMING UP

cuDNN 7, cuBLAS 9 and beyond

• Volta Convolutions and GEMMs - Tensor cores and legacy
  • 4-5x speedup on convolutions over Pascal, up to 9x on large GEMMs
  • Using FP16 storage and FP32 math; Volta using tensor cores

• Faster Softmax, Batch Normalization, Pooling

• Convolution Groups

• Faster Deconvolution
CHALLENGES FOR THE FUTURE
Memory, memory, memory!

Convolutions/GEMMs diminishing in % of compute time; no longer low hanging fruit for perf improvement

Several commonly used memory-bound routines

Batch Norm, for instance, needs 9 passes over memory (for forward+backward)

Can we invent new algorithms, keeping execution efficiency in mind?

Can we fuse operations together in a generic manner?

Can we use sparse operators?
QUESTIONS?