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

cuBLAS/NVBLAS

Intel CPUs

IBM Power

Kepler GPU

Pascal GPU

Volta GPU

DL COMPUTATION STACK

Low level primitives for DL

Data Scientists

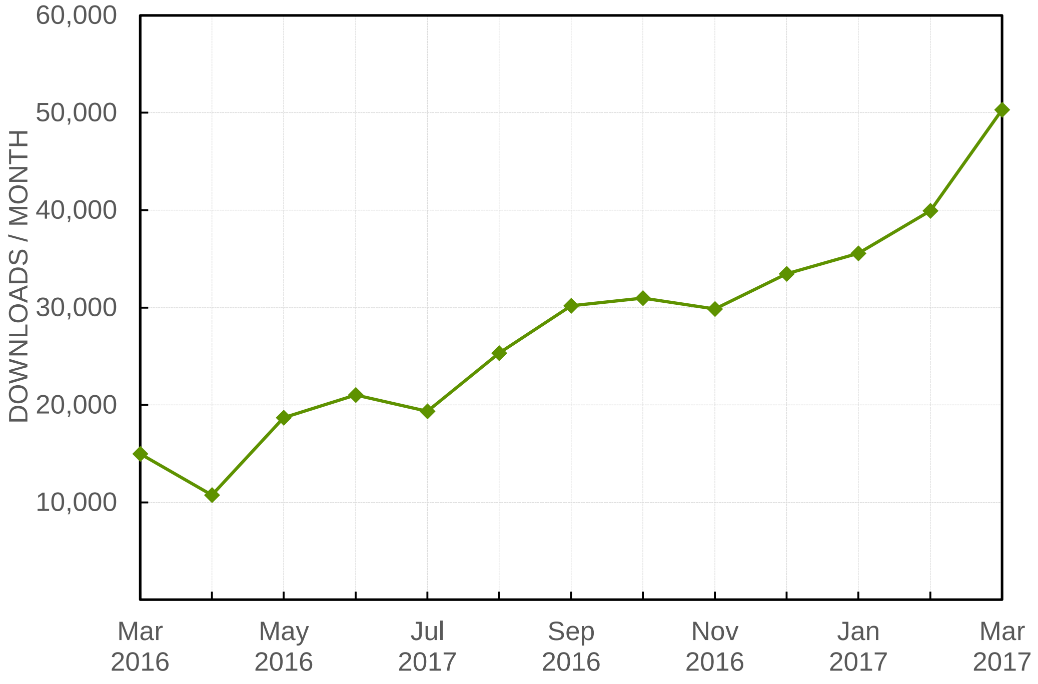
Frameworks

cuDNN | cuBLAS

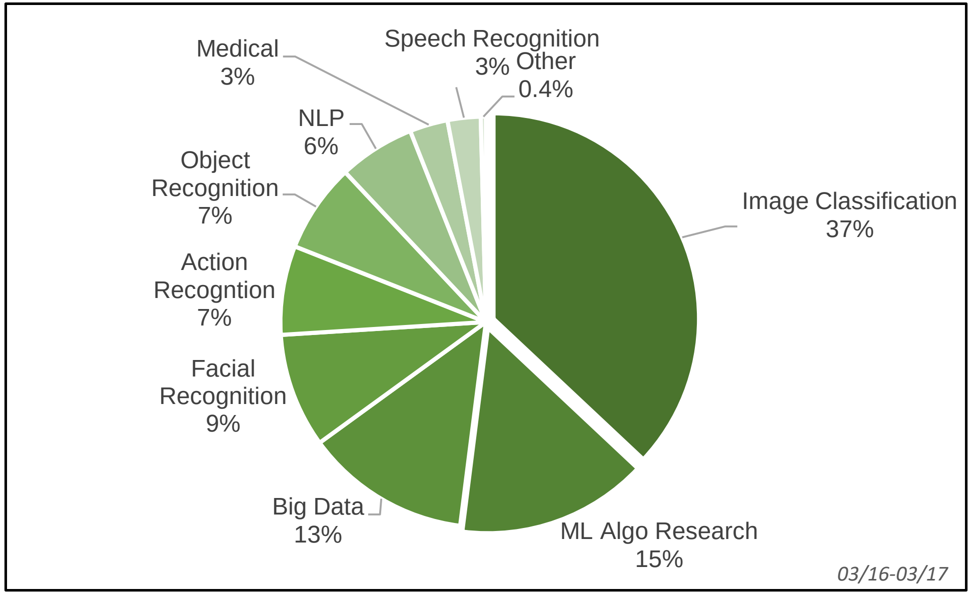
Kepler | Maxwell | Pascal | Volta | Future

cuDNN ADOPTION

cuDNN Downloads Over Time



cuDNN Downloads By Application



YEAR IN REVIEW

RNNs & Fast Convolutions

- LSTM / GRU
- 3D FFT Tiling
- Spatial Transformer Layer
- 3x3 Winograd convolution

v5 (Q2 2016)

Pascal Support

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

v5.1, cuBLAS 8 (Q3 2016)

Small Batch Training & Inference

- Fused Conv+Bias+ReLU
- Int8 convolutions
- Persistent RNNs
- Dilated convolutions
- Deterministic Max Pooling

v6 (Q1 2017)

CONVOLUTIONS

THE CONVOLUTION PROBLEM

Workhorse of CNNs

Circa 2014:

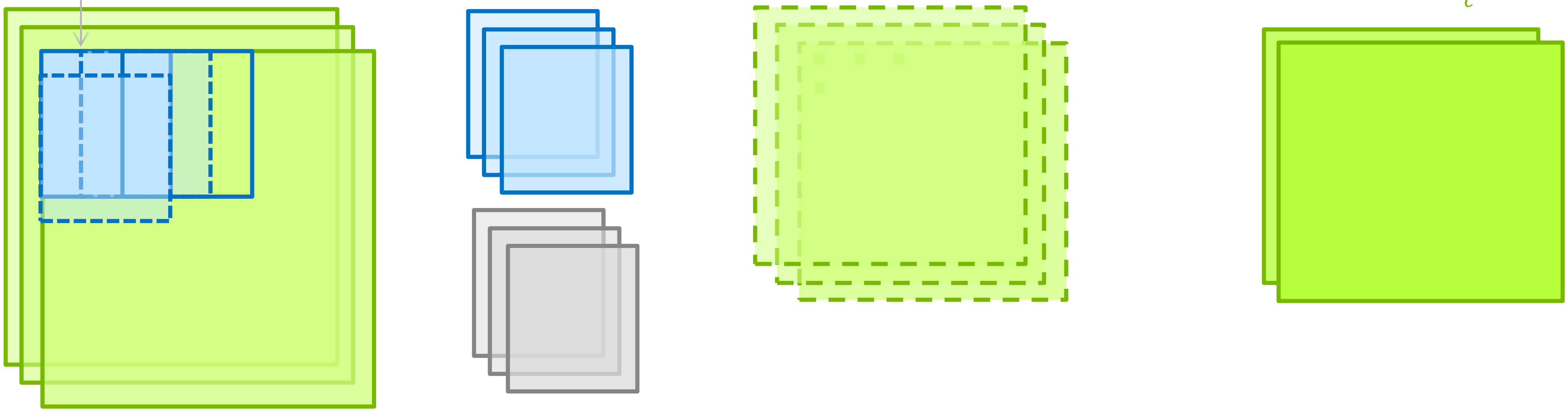
- Everybody is using GPUs to train CNNs
- Convolutions take up 80-90% of people's runtimes
- Can NVIDIA do something about this?

CONVOLUTION OPERATION

$$int[c,p,q] = \sum_{i,j \in filter} Im[c][istart + i, jstart + j]\,.\,Filt[c][i,j]$$

$$output[p,q] = \sum_{c} int[c,p,q]$$

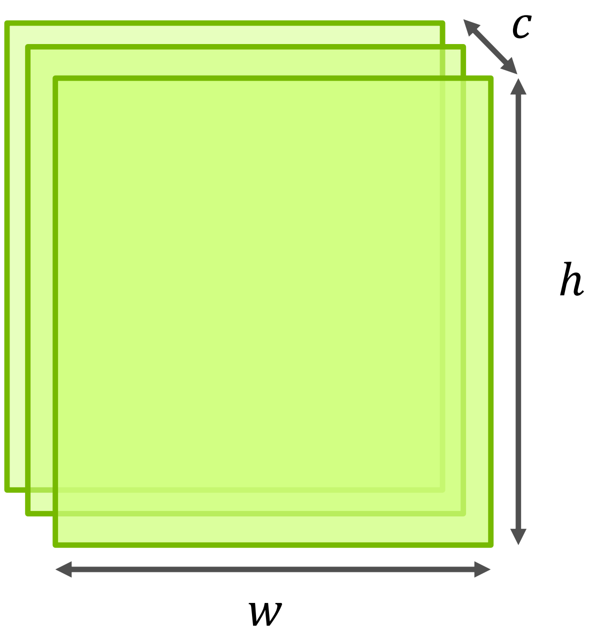
Pointwise multiply and sum, scalar output



Input Image | Input Filter | Intermediate output | Final Output

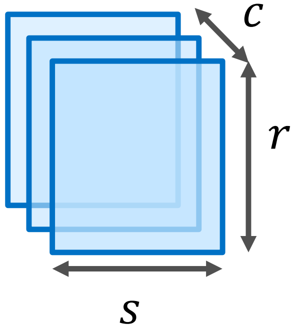
Why do it once if you can do it n times ? Batch the whole thing.

PARAMETER SPACE



Image

.... A batch of size n



Filter

.... A set of k

Attribute	Symbol
Batch size	n
Input Channels	c
Image height x Image width	$h\ x\ w$
Output Channels	k
Filter height x filter width	$r\ x\ s$
Zero Padding	$pad\_h\ x\ pad\_w$
Filter Striding	$u\ x\ v$

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

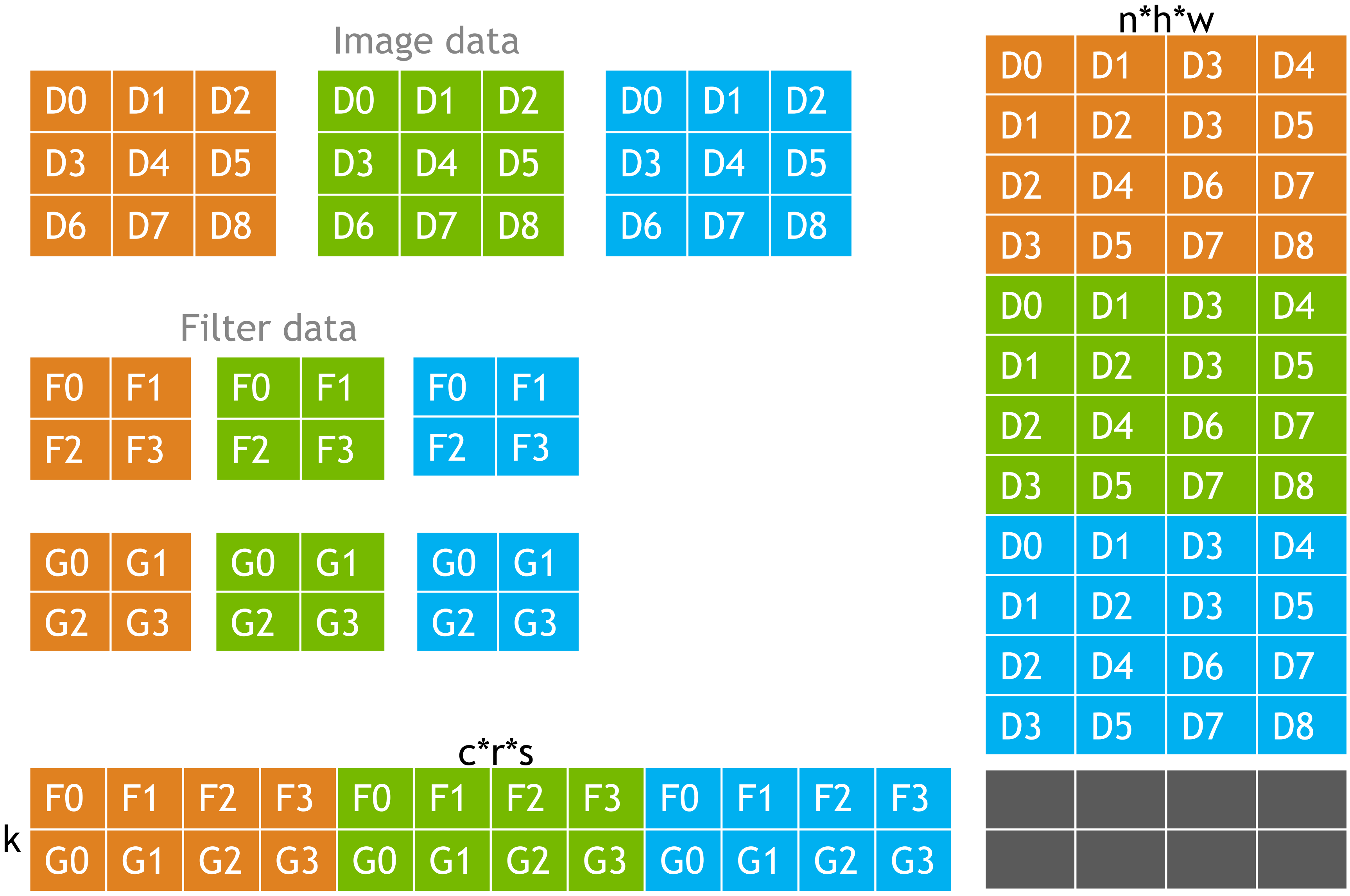
EFFICIENT DIRECT CONVOLUTIONS

AN ALTERNATIVE APPROACH

A Header file library

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

```
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);
```

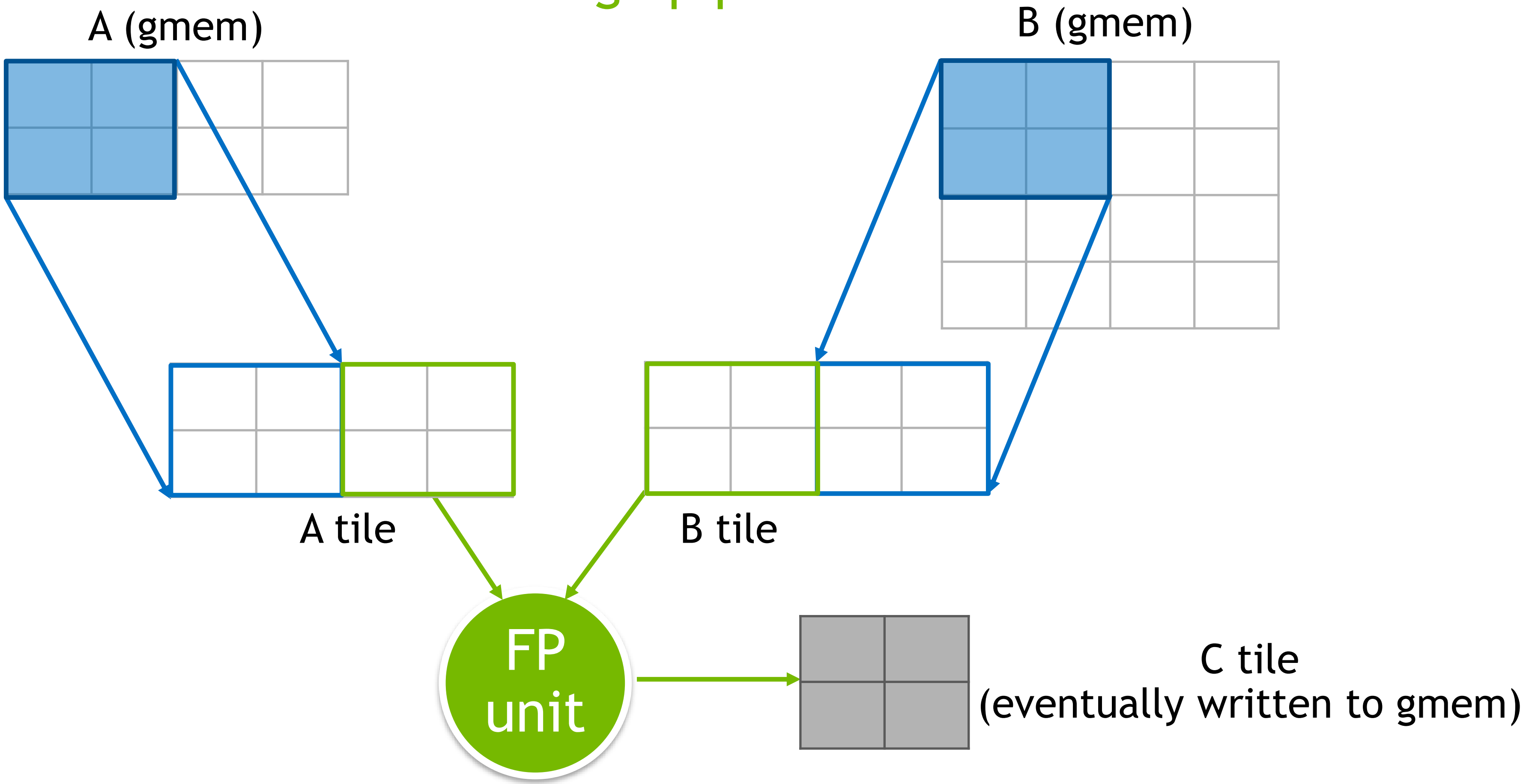


'High Performance Convolutional Neural Networks for Document Processing'

Kumar Chellapilla, Sidd Puri, Patrice Simard

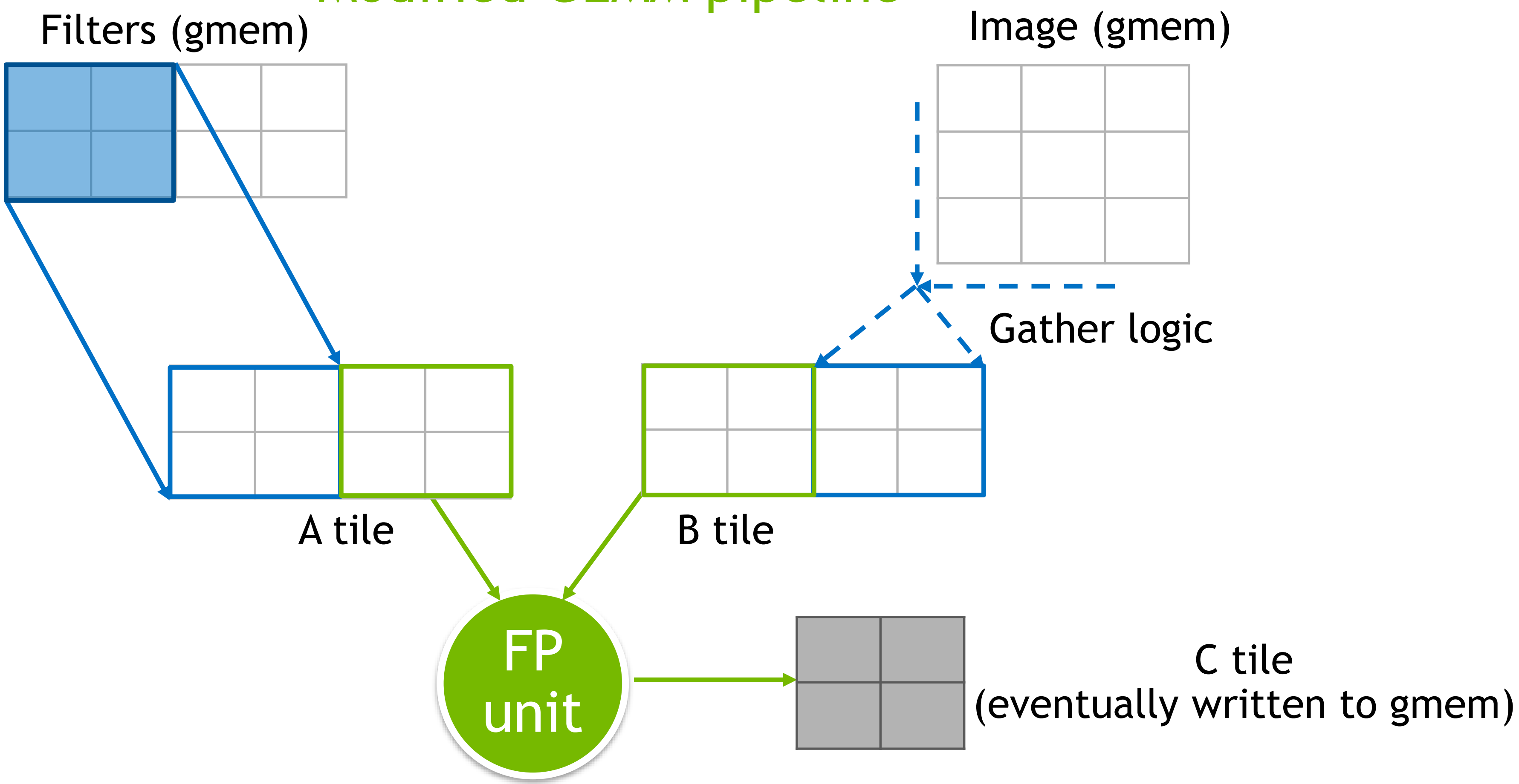
EFFICIENT GEMM ON GPUs

A two-stage pipeline



CONVOLUTION AS AN IMPLICIT GEMM

Modified GEMM pipeline



CONCLUSIONS ON IMPLICIT GEMM

Costs/Benefits

- Benefits:
  - Deliver perf in an scalable manner across parameter space
  - Leverage a lot of panistakingly 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:

- ‘Fast Convolutional Nets With fbfft: A GPU Performance Evaluation’ (2014) – Vasilache N, Johnson J et al
- ‘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 $Image * Filter = Output$ as described before ("$*$" is convolution)

$$FFT(Image * Filter) = FFT(Output)$$

$$FFT(Image * Filter) = FFT(Image) *. FFT(Filter)$$; "$*.$" is a point-wise multiply

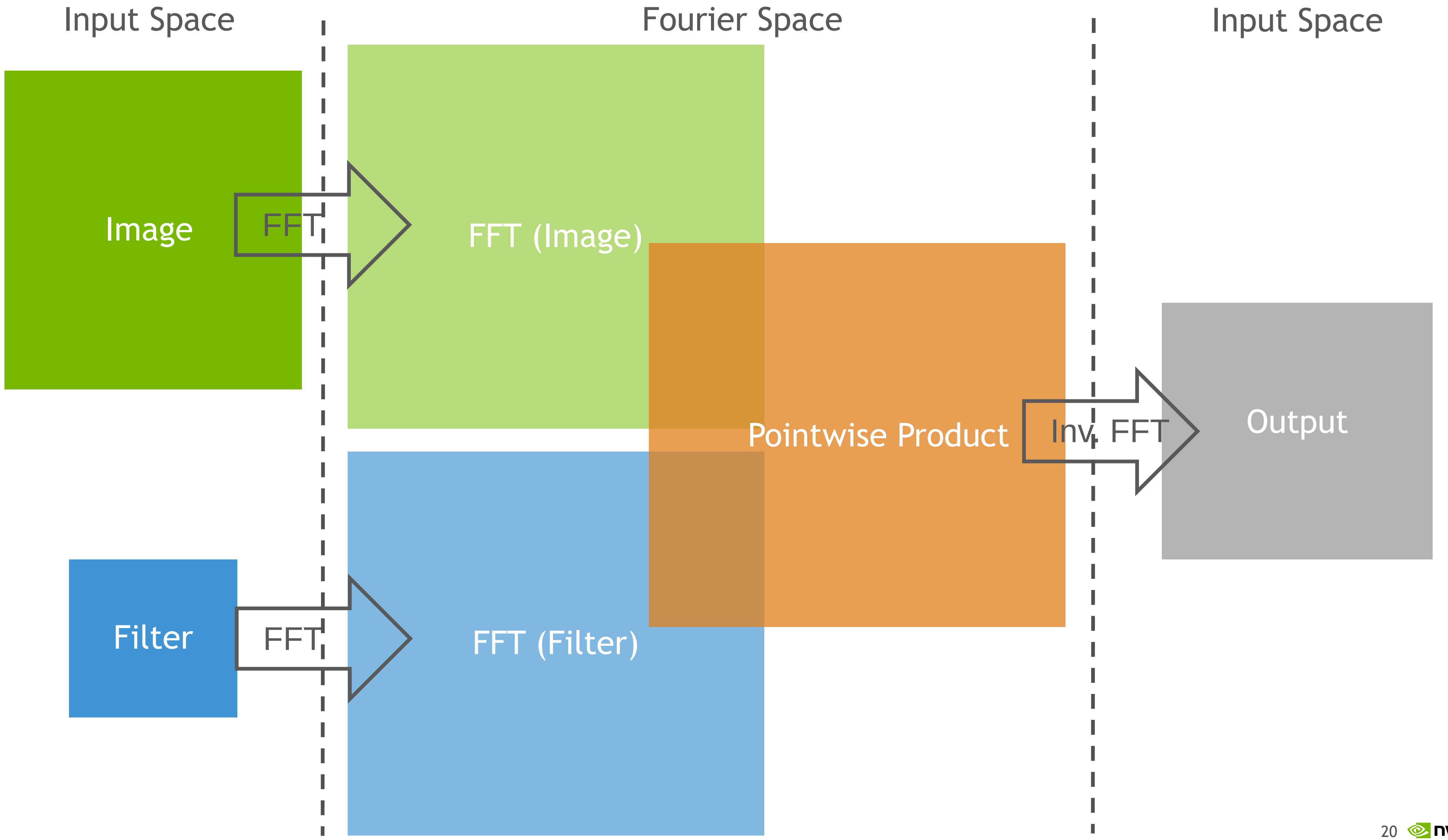
$$\therefore Output = FFT^{-1}(FFT(Image) *. FFT(Filter))$$

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

$$Output = FFT^{-1}(FFT(Image) *. FFT(Filter))$$

FFTs take $O(nlogn)$ time, and the point-wise multiply takes $O(n)$ time

Reuse cost of transform across batch and output feature maps


Input Space
Fourier Space
Input Space
Image
FFT
FFT (Image)
Pointwise Product
Inv. FFT
Output
Filter
FFT
FFT (Filter)

WINOGRAD CONVOLUTIONS

A different transform

Follows a similar scheme

$$Output = Inverse\ Tranform\ (Forward\ Transform\ (Image) *. Forward\ Transform\ (Filter))$$

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

'On Improving the Numerical Stability of Winograd Convolutions' (2017) – Vincent K, Stephano K, Frumkin M et al

PERSISTENT RNNs

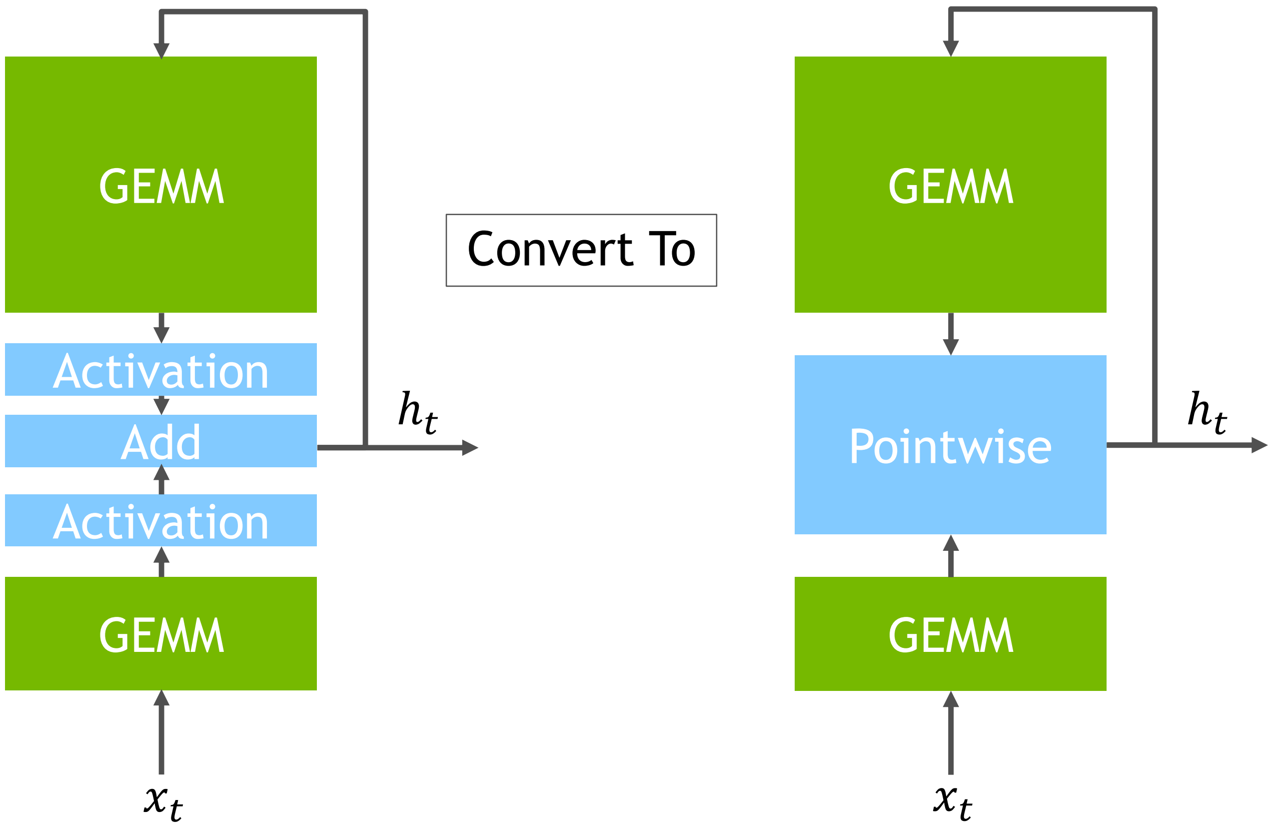
RECURRENT NEURAL NETWORKS

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



https://devblogs.nvidia.com/parallelforall/optimizing-recurrent-neural-networks-cudnn-5

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

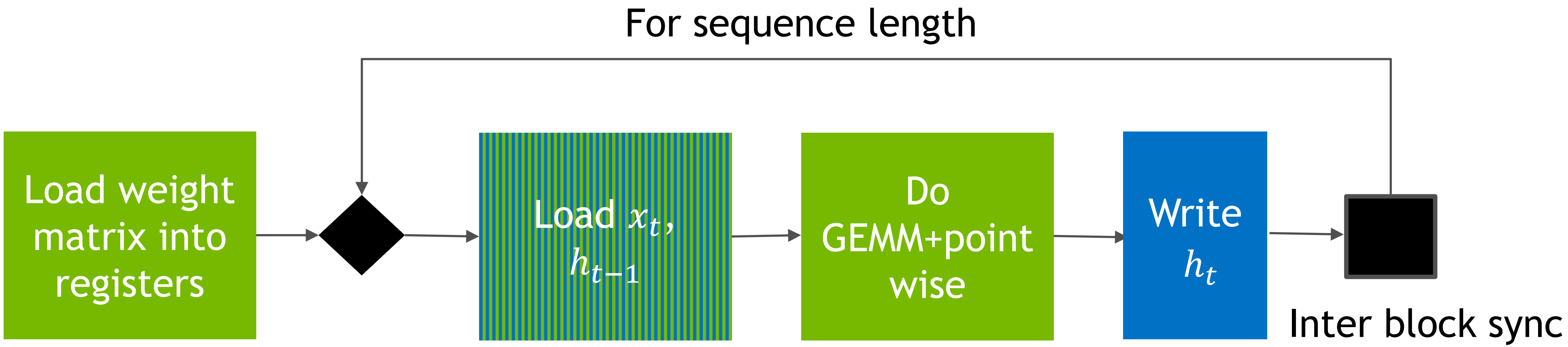
PERSISTENT RNNs

Exploiting reuse of weight matrix

- “Persistent RNNs: Stashing Recurrent Weights On-Chip” (2015) – Diamos G, Sengupta S, Catanzaro B et al
- 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?