facebook

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

1 Introduction

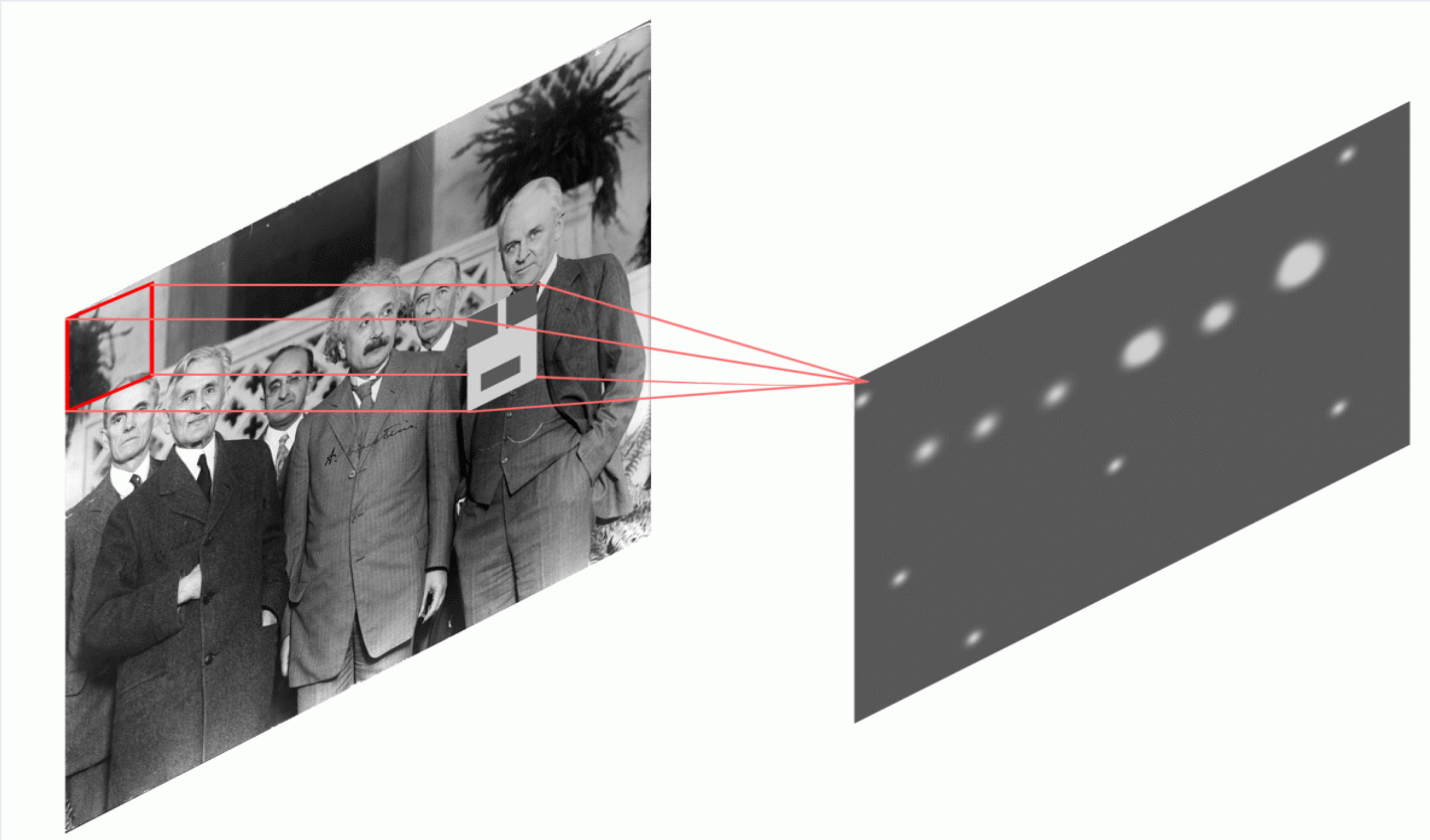
2 Contributions

3 Upcoming Work

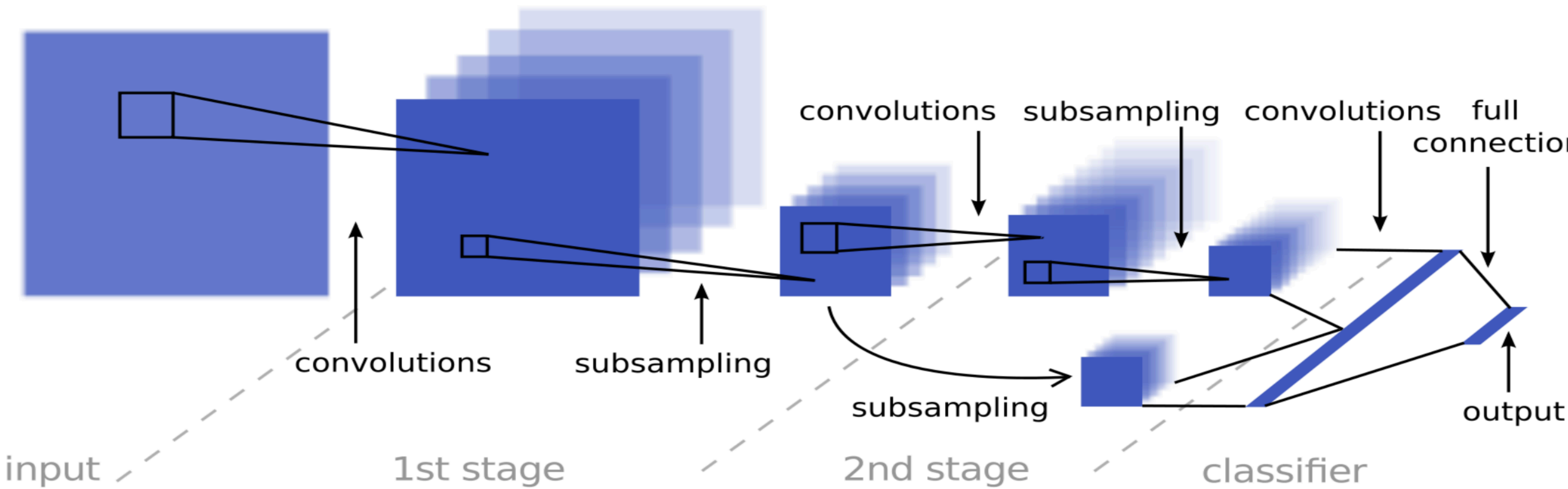
4 Numbers

Introduction

Convolution



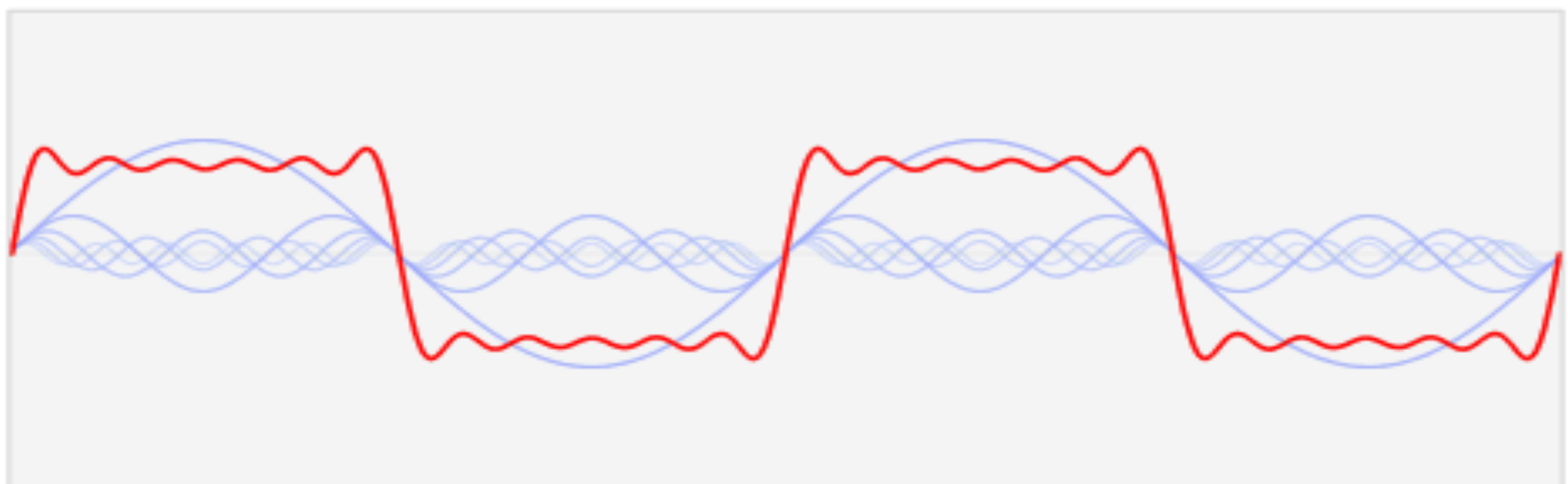
Convolutional Neural Networks



- 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) -> O (n . 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

INPUT		OUTPUT
$In_{S\times f\times h\times w}$	$\xrightarrow{FFT2D}$	$InF_{S\times f\times(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)}$
$Wei_{f'\times f\times k_h\times k_w}$	$\xrightarrow{FFT2D}$	$WeiF_{f'\times f\times(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)}$
$InF_{S\times f\times(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)}$	$\xrightarrow{Trans2D}$	$InFT_{(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)\times S\times f}$
$WeiF_{f'\times f\times(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)}$	$\xrightarrow{Trans2D}$	$WeiFT_{(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)\times f'\times f}$
$\begin{cases} InFT_{(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)\times S\times f} \\ WeiFT^{*}_{(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)\times f'\times f} \end{cases}$	$\xrightarrow{\texttt{Cgemm}}$	$OutFT_{(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)\times S\times f'}$
$OutFT_{(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)\times S\times f'}$	$\xrightarrow{Trans2D}$	$OutF_{S\times f'\times(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)}$
$OutF_{S\times f'\times(h+p_h)\times(\lfloor\frac{w+p_w}{2}\rfloor+1)}$	$\xrightarrow{IFFT2D}$	$Out_{S\times f'\times(h-k_h+1)\times(w-k_w+1)}$

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 <= 15 x 15, you only need 16x16 or 32x32 FFTs
- Whatever the kernel size, cost is the same
  - True until you need a larger Fourier interpolation basis
    - Then tiling kicks in
- Algorithm ›› 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)

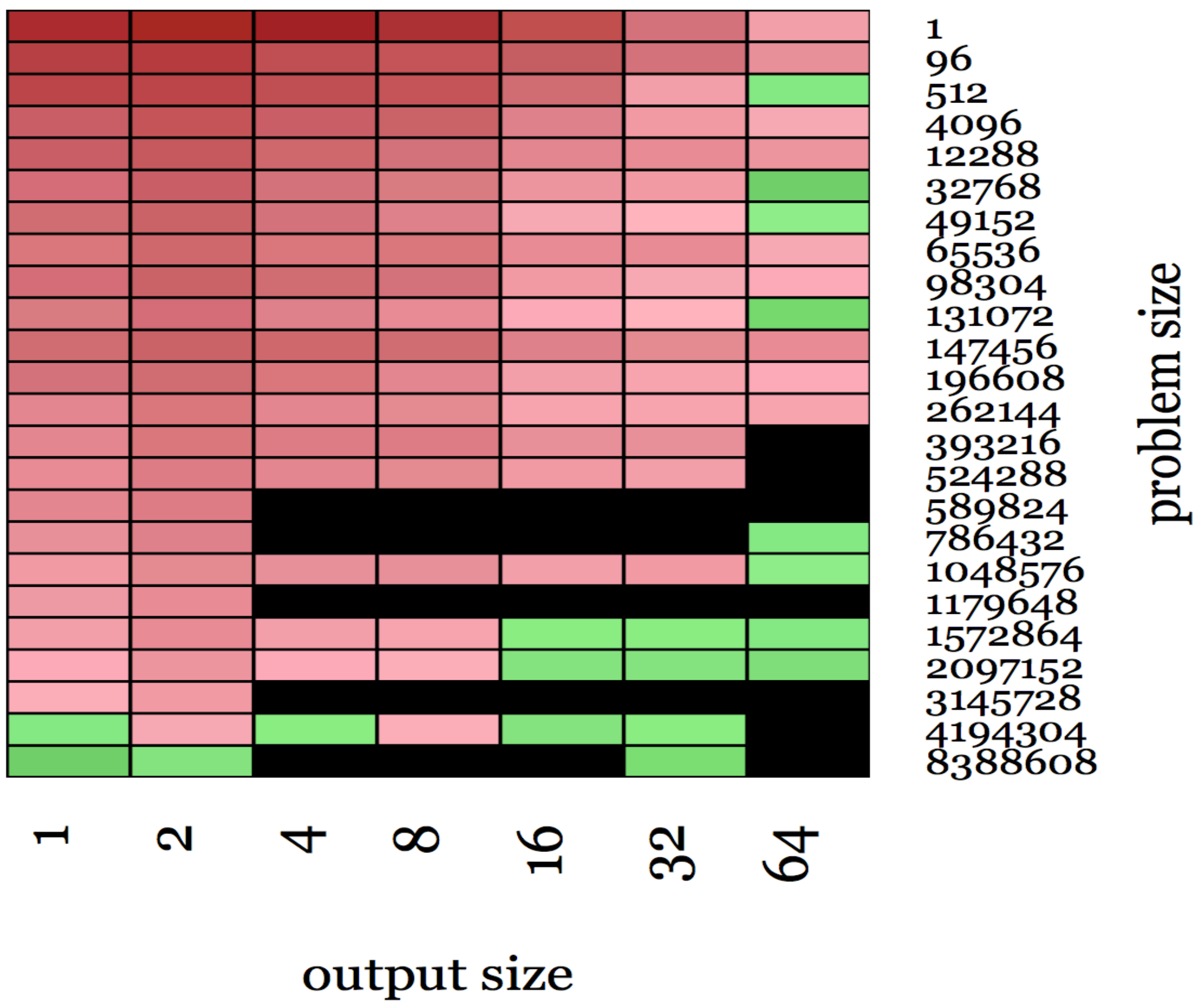


Figure 1: 3 × 3 kernel (K40m)

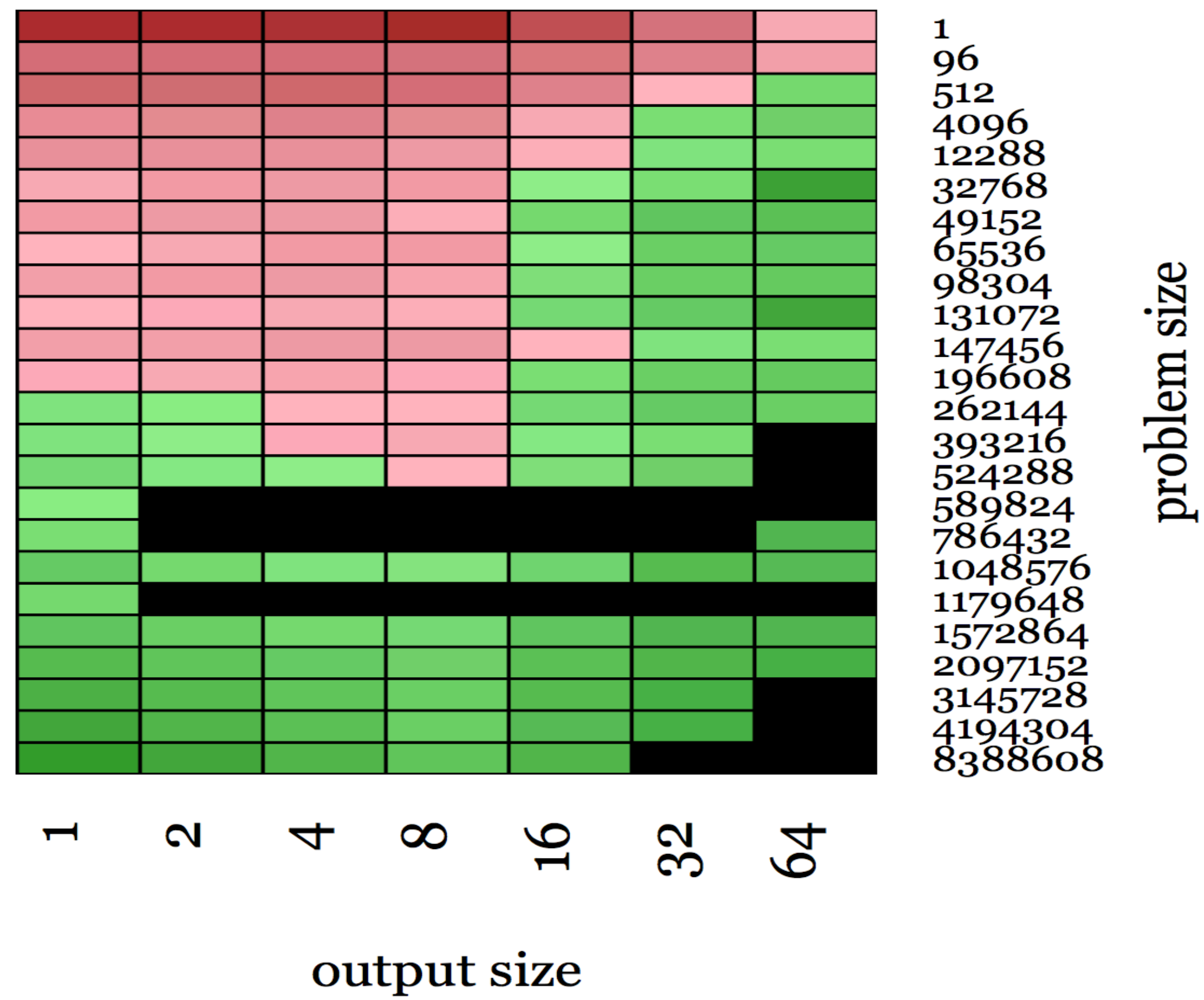


Figure 2: 5 × 5 kernel (K40m)

Speedup (CuFFT + CuBLAS)

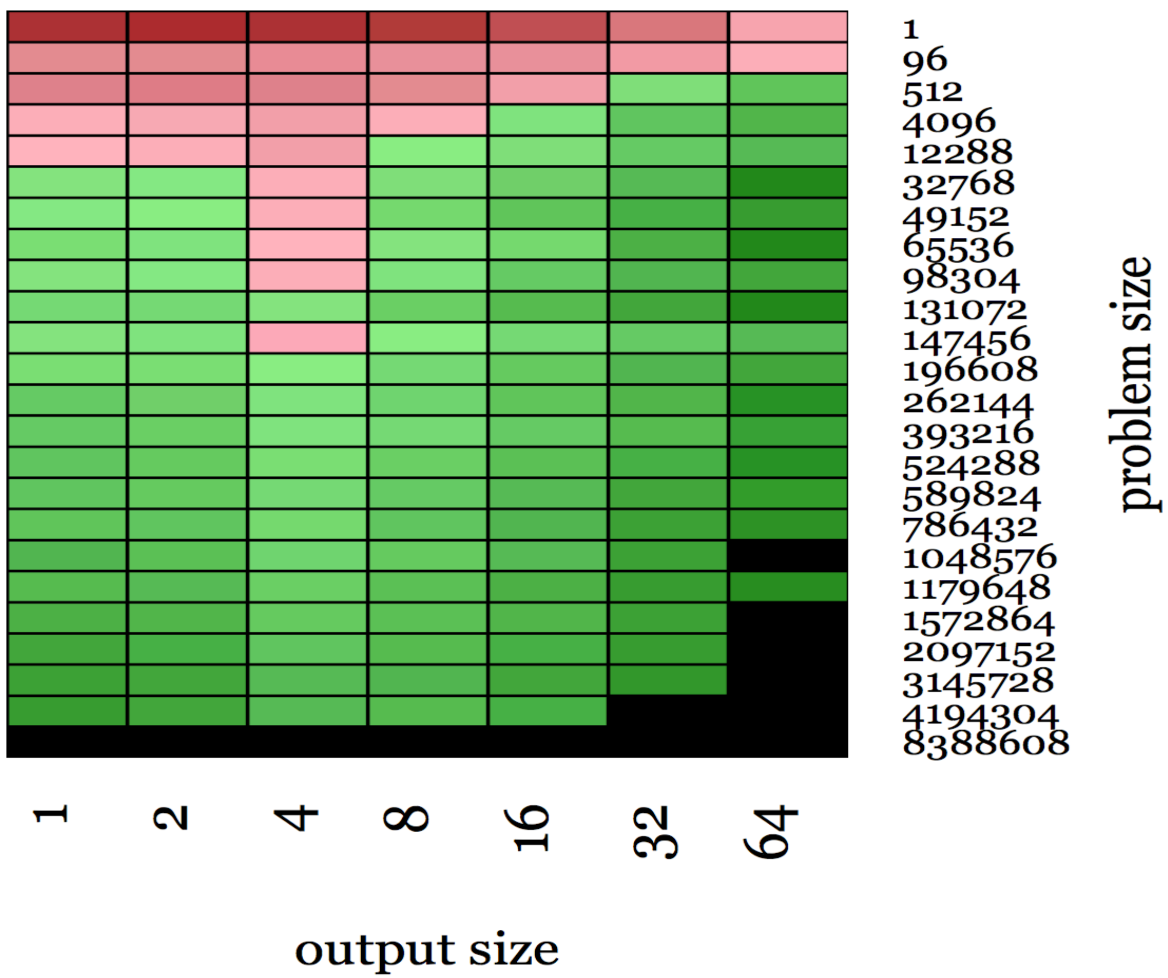


Figure 3: 7 × 7 kernel (K40m)

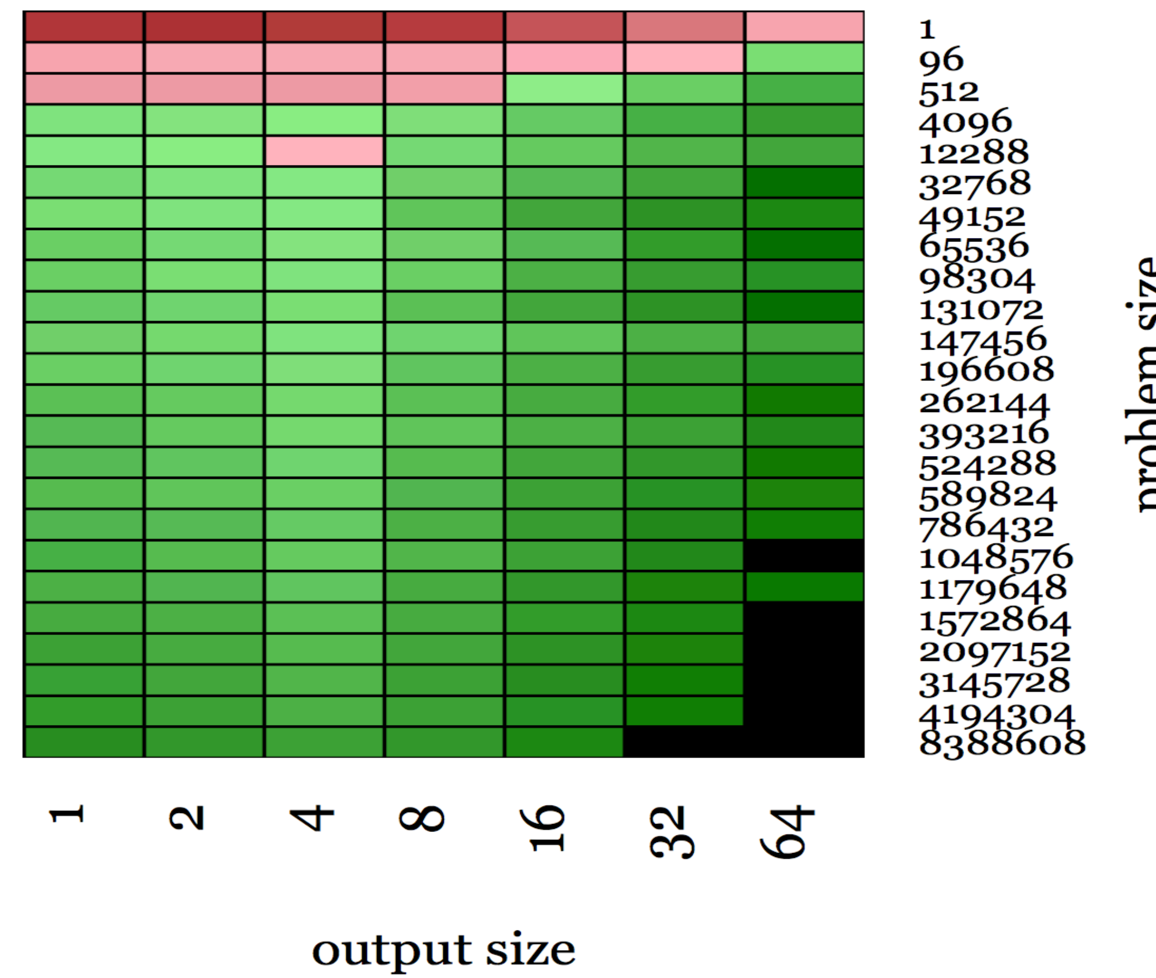


Figure 4: 9 × 9 kernel (K40m)

Speedup (CuFFT + CuBLAS)

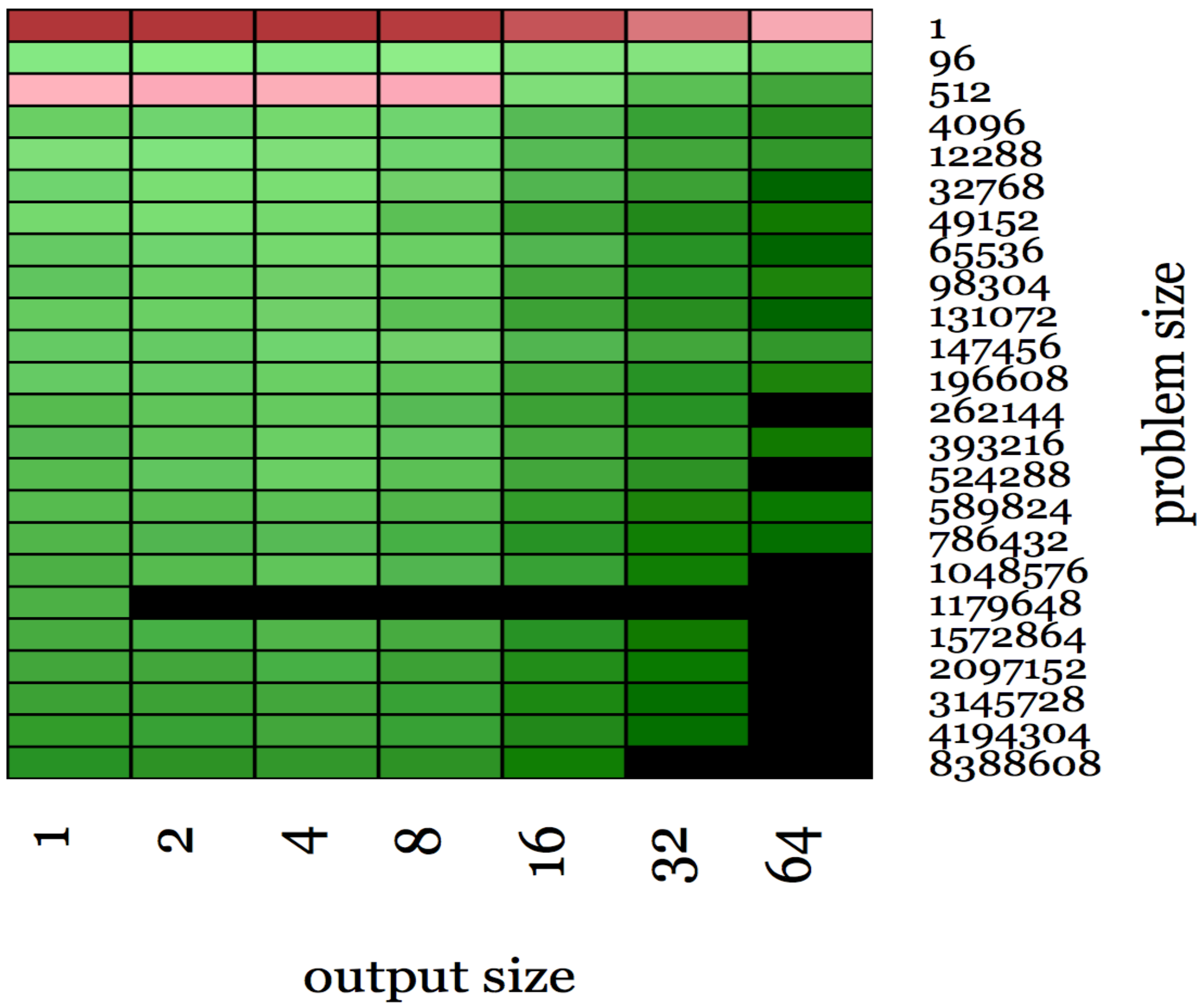


Figure 5: 11 × 11 kernel (K40m)

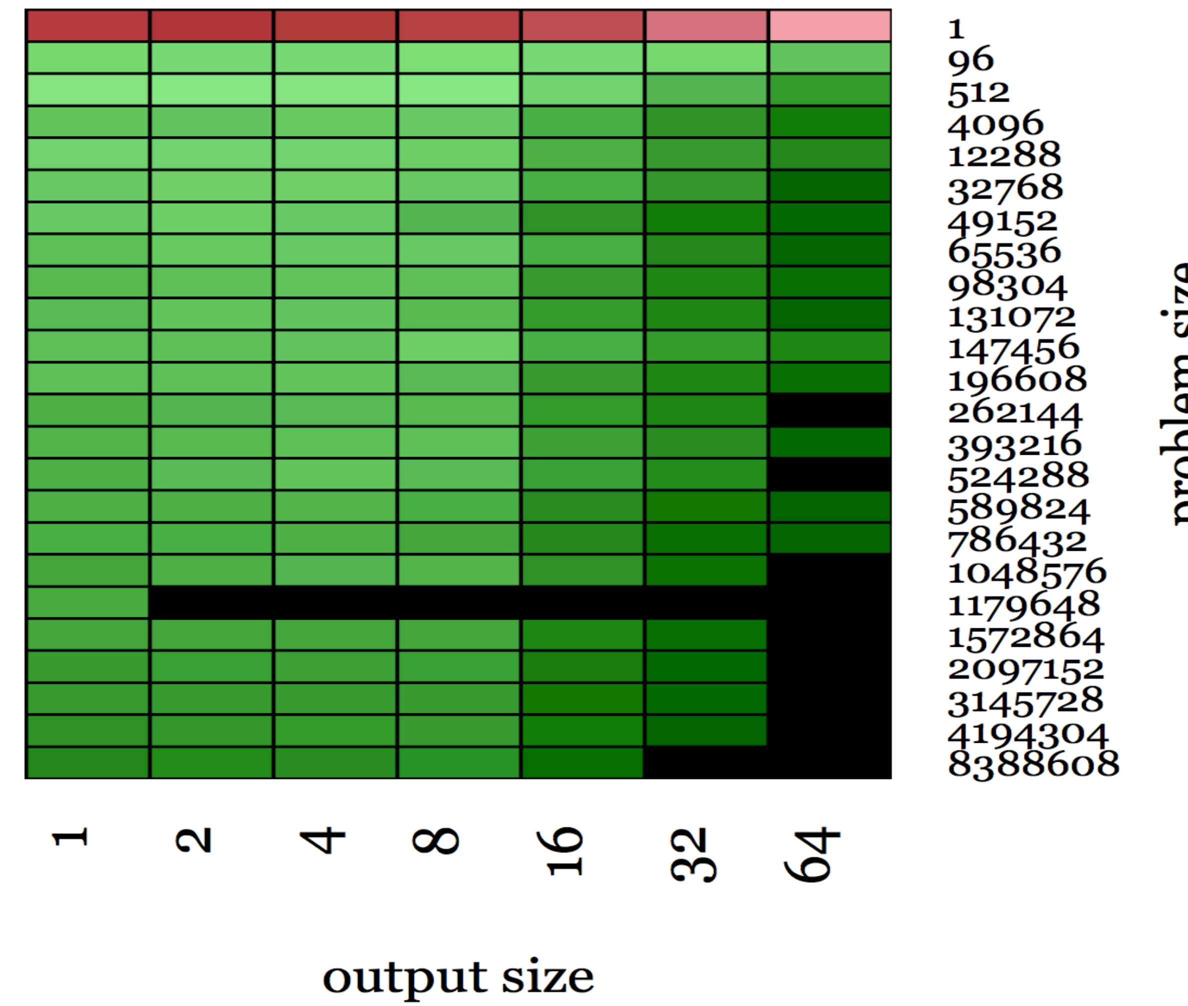


Figure 6: 13 × 13 kernel (K40m)

Speedup (FBFFT vs CuFFT)

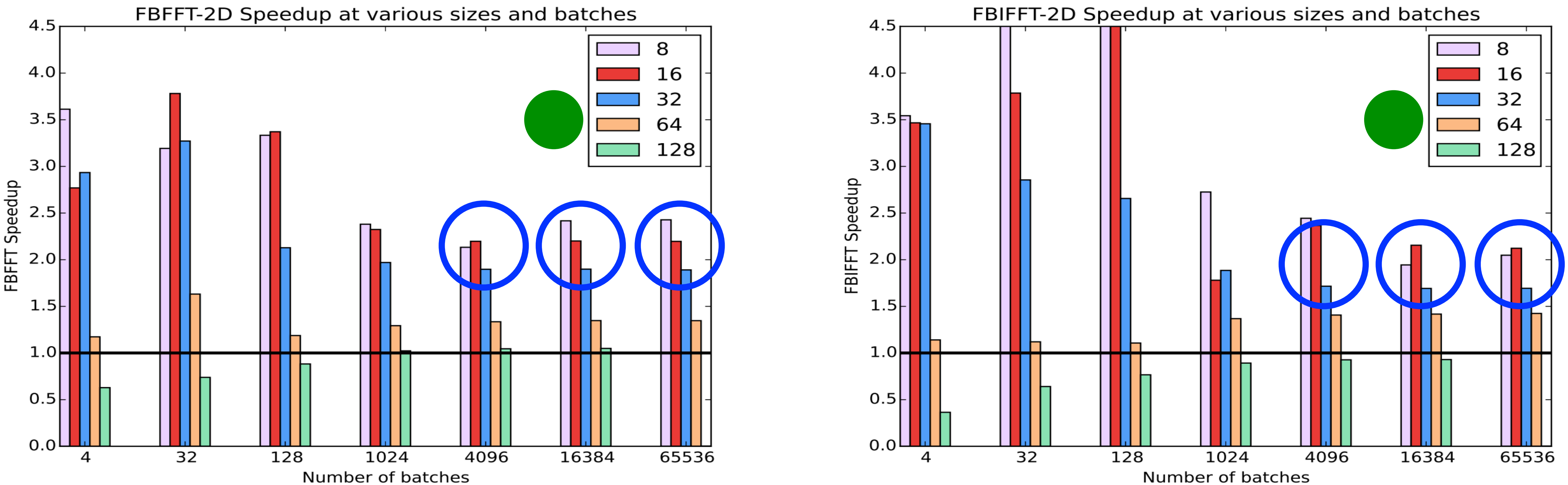


Figure 8: `fbfft`-2D FFT and IFFT (K40m, cuFFT 6.5 @ 1x)

Comparison on Imagenet Networks

AlexNet (One Weird Trick paper) - Input 128x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
NervanaSys-16	ConvLayer	**97**	**30**	**67**
NervanaSys-32	ConvLayer	109	31	78
fbfft	SpatialConvolutionCuFFT	136	45	91
cudaconvnet2*	ConvLayer	177	42	135
CuDNN (R2) *	cudnn.SpatialConvolution	231	70	161
Caffe (native)	ConvolutionLayer	324	121	203
Torch-7 (native)	SpatialConvolutionMM	342	132	210

Comparison on Imagenet Networks

Overfeat [fast] - Input 128x3x231x231

Library	Class	Time (ms)	forward (ms)	backward (ms)
NervanaSys-16	ConvLayer	**364**	**119**	**245**
NervanaSys-32	ConvLayer	410	126	284
fbfft	SpatialConvolutionCuFFT	407	139	268
cudaconvnet2*	ConvLayer	723	176	547
CuDNN (R2) *	cudnn.SpatialConvolution	810	234	576
Caffe	ConvolutionLayer	823	355	468
Torch-7 (native)	SpatialConvolutionMM	878	379	499

Comparison on Imagenet Networks

OxfordNet [Model-A] - Input 64x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
NervanaSys-16	ConvLayer	**530**	**166**	**364**
NervanaSys-32	ConvLayer	629	173	456
fbfft	SpatialConvolutionCuFFT	1092	355	737
cudaconvnet2*	ConvLayer	1229	408	821
CuDNN (R2) *	cudnn.SpatialConvolution	1099	342	757
Caffe	ConvolutionLayer	1068	323	745
Torch-7 (native)	SpatialConvolutionMM	1105	350	755

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 9th, 10:30am – 1:30pm

Questions?