An Empirically Tuned 2D and 3D FFT Library on CUDA GPU

Liang Gu, Xiaoming Li, Jakob Siegel

University of Delaware

Fast Fourier Transform(FFT) Libraries on CUDA GPU

- *CUFFT*

-By NVIDIA, v-2.2 and 2.3 released in 2009
-Achieves 20-40G Flops on GTX280 for 1D 2D and 3D FFTs

- *High Performance Discrete Fourier Transforms on Graphics Processors (SC2008)*

- By Naga K. Govindaraju etc. (Microsoft) 2008
- Achieves up to 90 and 120 Gflops for 1D and 2D FFT

- *Bandwidth Intensive 3-D FFT kernel for GPUs using CUDA, SC2008*

-By Akira Nukada etc. (Tokyo Institution of Technology) 2008
-Achieves about 80GFlops on GTX280 for 3D FFT

Motivation

➢*Meet the following Challenges of multi-dimensional FFT*

- Generally describe multi-dimensional FFT solution
- Describe hardware constraints
- Overhead of global transposition
- Reduce memory access
- Better performance

Our 2D and 3D FFT Overview

➢ *Extended I/O tensor representation*

- Incorporate Cooley-Tukey decomposition and codelet into I/O tensor
  -Codelets are FFTW's compiler generated code for small FFTs (no branch!)
  -Compute high dimensional FFT without global transposition
- Express CUDA constraints in tensor format

➢ *Multi-dimensional FFT transforms*

- Grouping codelets across multi-dimensions
- Commutability and partial order of codelets

➢ *Trade-offs in search space*

- Choice of codelet size
- Codelet grouping and order exchange
- Pre-calculate twiddle factors / compute on the fly

Tensor Representation of FFT

➢ *Original I/O tesor representation in FFTW*

• I/O dimension $d_1 = d(n,i,o,I,O)$
n is the FFT transform size
i and *o* are the input and output strides
I and *O* are the pointers of the input and output arrays.

• I/O tensor $t = \{d_1, d_2, \ldots d_p\}$
This represents a p-dimensional FFT.

• E.g. A 3D array X*Y*Z can be denoted as
{d(Z,XY,XY,I,O),d(Y,X,X,O,I),d(X,1,1,I,O)}

It Only represent s an FFT problem setup, not a solution

Tensor Representation of Cooley-Tukey Algorithm and Codelet

- *Twiddle codelet extension of Cooley-Tukey algorithm*

Compute 1D FFT of size n=mr in three steps

(1) perform r column-wise FFTs of size m → direct FFT/codelet

(2) multiply twiddle factors and transpose

(3) compute m column-wise FFTs of size r → twiddle codelet (DIT)

- *I/O tensor rewrite rules*

(1) Decimation in time(DIT): $\{d(mr,i,o,I,O)\}=\{d(m,ir,o,I,O), t^m_r(r,mo,mo,O,O)\}$

(2) Decimation in frequency(DIF): $\{d(mr,i,o,I,O)\}=\{t^r_m(r,im,im,I,I), d(m,i,ro,I,O)\}$

The solution of an FFT is represented in extended I/O tensor format

CUDA Constraints in Tensor Form

➢ *Compute codelet sub-sequence in shared memory*

• Recursively applying Cooley-Tukey generates a sequence of codelets

• A sub-sequency is computed in one CUDA kernel with one pass of R/W of the whole data set from global memory

• Needs shared memory to store intermediate results

➢ *Shared memory constraint(16KByte)*

• X dimensional kernels, if codelets of size x_1, x_2 ... x_n are in the same kernel
Then, $2*2*4*x_1x_2...x_n<=16K \Leftrightarrow x_1x_2...x_n<=1024$

• Y dimensional kernels (or Z kernels), if codelets y_1, y_2 ... y_n are in the same kernel
Then, $16*2*2*4*y_1y_2...y_n<=16K \Leftrightarrow y_1y_2...y_n<=64$

CUDA Constraints in Tensor Form

- To reduce the possible search space represented in I/O tensor format

➢ *Condition of coalesced-access on global memory*

• For d(n,i,o,I,O), read is coalesced if $i=16*2^j$ where j>=0
I is 128Byte aligned (each thread access a float2 at a time)

➢ *Shared memory assisted coalesing on global memory*

• When the above condition is not satisfied but the overall accessed region is continuous, we can use shared memory to assist access.

• E.g: 2D FFT {d(y,x,x,I,O)d(x,1,1,O,O)} , where $x=16*2^j$ where j>=0, we can rewrite it as:
{d(y,x,x,I,S),{d(x,1,1,S,S)+cp(y,x,x,S,O)}} where S stands an address on shared memory

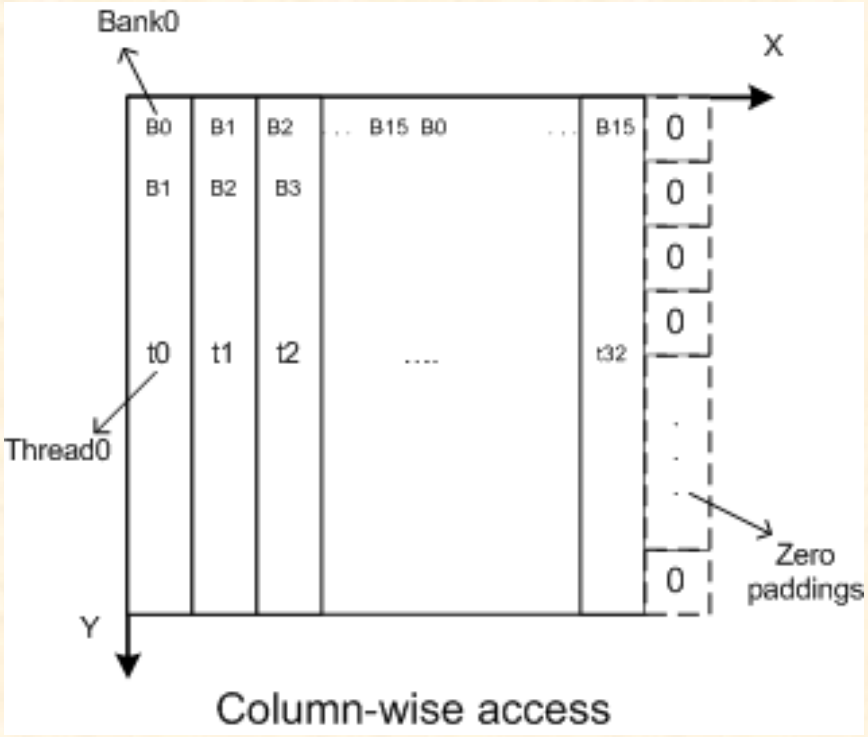
Padding on Shared Memory

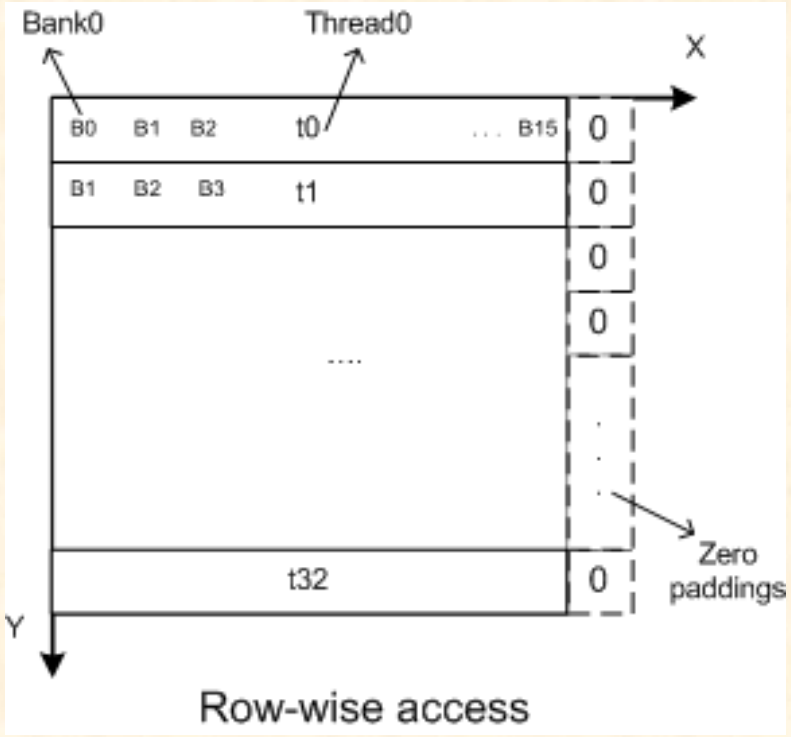
➢ *Bank-conflict free padding of 2D arrary*

• Shared memory is devided into 16 banks, a half warp(16 threads) will be serialized if any of them are accessing the same bank at the same time

• Constant stride access conflic-free condition: *The offset between consecutive threads is an odd number*

• Case (1): even number of columns





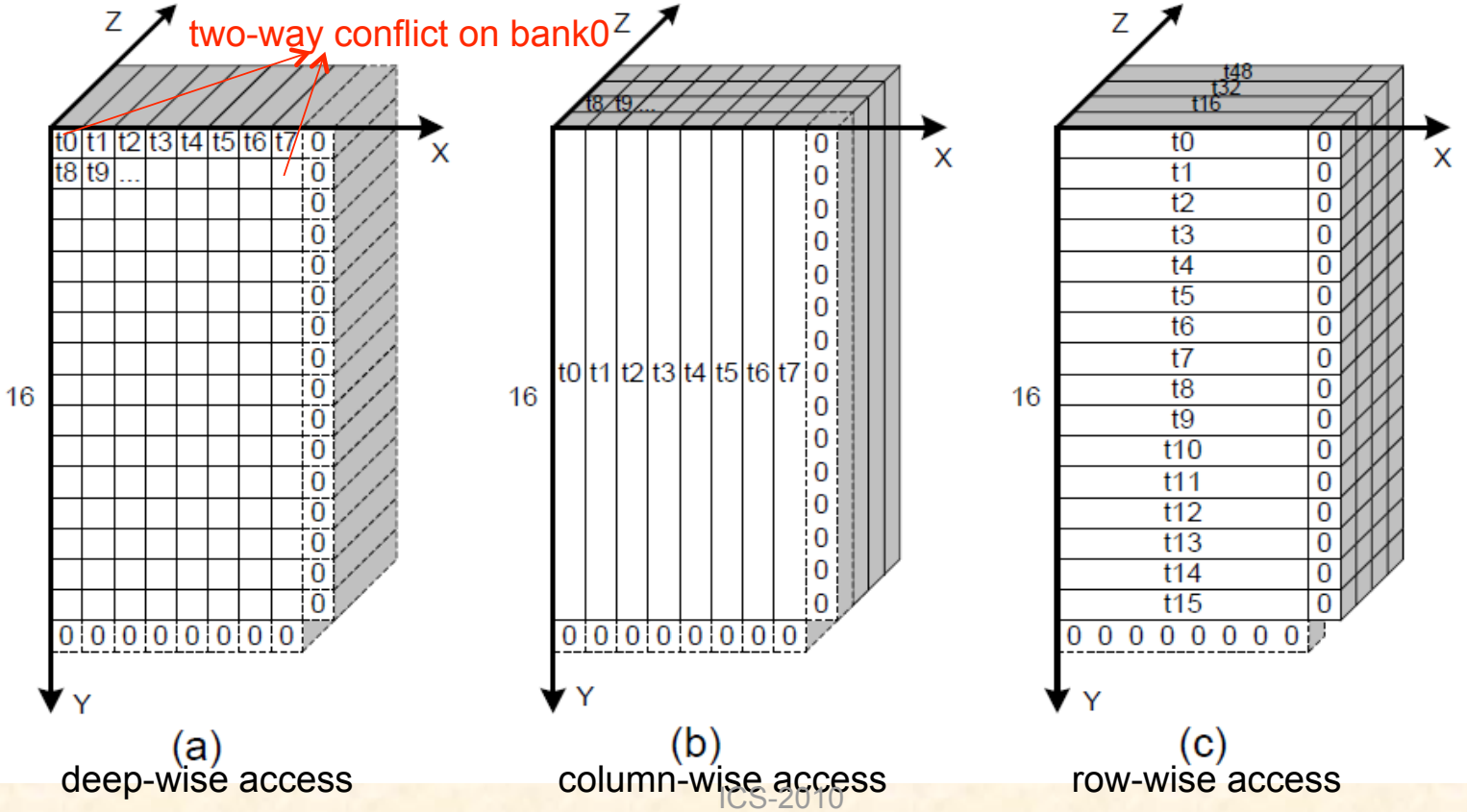
• Case (2): odd number of columns (no padding is necessary)

Padding on Shared Memory

➢ *Bank-conflict limited padding for 3D arrary*

• It is not always possible to achieve bank-conflict free for 3D array. We handle two frequently encountered cases of 3D array x*y*z (codelets 4,8,16,32 are often used):

• Case(1) y=16: column-wise and row-wise access (b,c) are bank-conflict free, (a) has at most 2-way conflict



• Case(2) x=16 (bank-conflict free, similar to 2D array)

Some FFT Transformations

- I/O tensor representaion enable us to easily explore some optimization opportunities

➢ *Grouping of codelets*

• We group different dimensions or codelets of different dimensions together whenever there is a benefit.

• E.g. 2D FFT of x=y=128 and $y_1=x_1=16$, $y_2=x_2=8$

Without grouping:

$\{\underline{d(y_1,xy_2,x,I,O)}, \underline{t(y_2,xy_1,xy_1,O,O)}, \underline{d(x_1,x_2,x_2,O,S)}, \underline{t(x_2,1,x_1,S,O)}\}$

kernel 1 kernel 2 kernel 3

If group Y_2 with X dimensional kernel:

$\{\underline{d(y_1,xy_2,x,I,O)}, \underline{t(y_2,xy_1,xy_1,O,S)}, \underline{d(x_1,x_2,x_2,S,S)}, \underline{t(x_2,1,x_1,S,O)}\}$

kernel 1 kernel 2

Some FFT Transformations

➢ *Commutability and partial order*

• The order of a codelet sequence can be changed
However, the relative order of codelets of the same dimension (generated by using Cooley-Tukey) must be preserved.

For DIT, a valid order is: $d(x_{1,}...)$, $d(y_1,...)$, $t(x_{2...})$, $t(y_{2...})$, $t(x_{3...})$, $t(y_{3...})$
an invalid order is: $t(y_2,...)$, $d(x_1,...)$, $d(y_1,...)$, $t(x_{2.},...)$

• E.g. 2D FFT of size 16*16 y=x=16, $y_1=y_2=4$
{d(y,x,x,I,S),d(x,1,1,S,O)}

Without order exchange:
$\{d(y_1,xy_2,x,I,S),t(y_2,xy_2,xy_2,S,S)$,{d(x,1,1,S,S) +cp(y_2,xy_2,xy_2,S,O)} }

With partial order exchange:
$\{d(y_1,xy_2,x,I,S),d(x,1,1,S,S),t(y_2,xy_2,xy_2,S,O)\}$

Trade-offs in Search Space

- Hand-tuning is performed for each FFT size on search space described above

➢ *Freedom in search space*

- Decomposition size and codelets grouping
- Codelets order exchange
- Thread block size
- CPU pre-calculated twiddle factor or compute on-the-fly
- Spill registers to increase occupancy

➢ *Codelet size trade-off*

- Smaller codelets: more paralellism, less register pressure and complexity
- Larger codelets: less passes of memory access, easy to ensure bank-conflict free
- Typically codelets of size 4,8,16 and 32 give good performance

Evaluation

➢ *Experiment Setup:*

•Our library supports single precision 2D up to 4K*4K, 3D up to 512*512
•We test on the following three GPU platforms:

GPU	Compute capability	Multi-processors	Bandwidth (GBps)	Driver	Nvcc & Cufft	Gcc
GeForce GTX280	1.3	30	141.7	185.18.08	2.2	4.3.2
Quadro FX5600	1.0	16	76.8	190.18	2.3	4.1.2-46
GeForce 9500GT	1.1	4	25.6	190.18	2.3	4.1.2

➢ *Both Gflops and runtime are compared with*

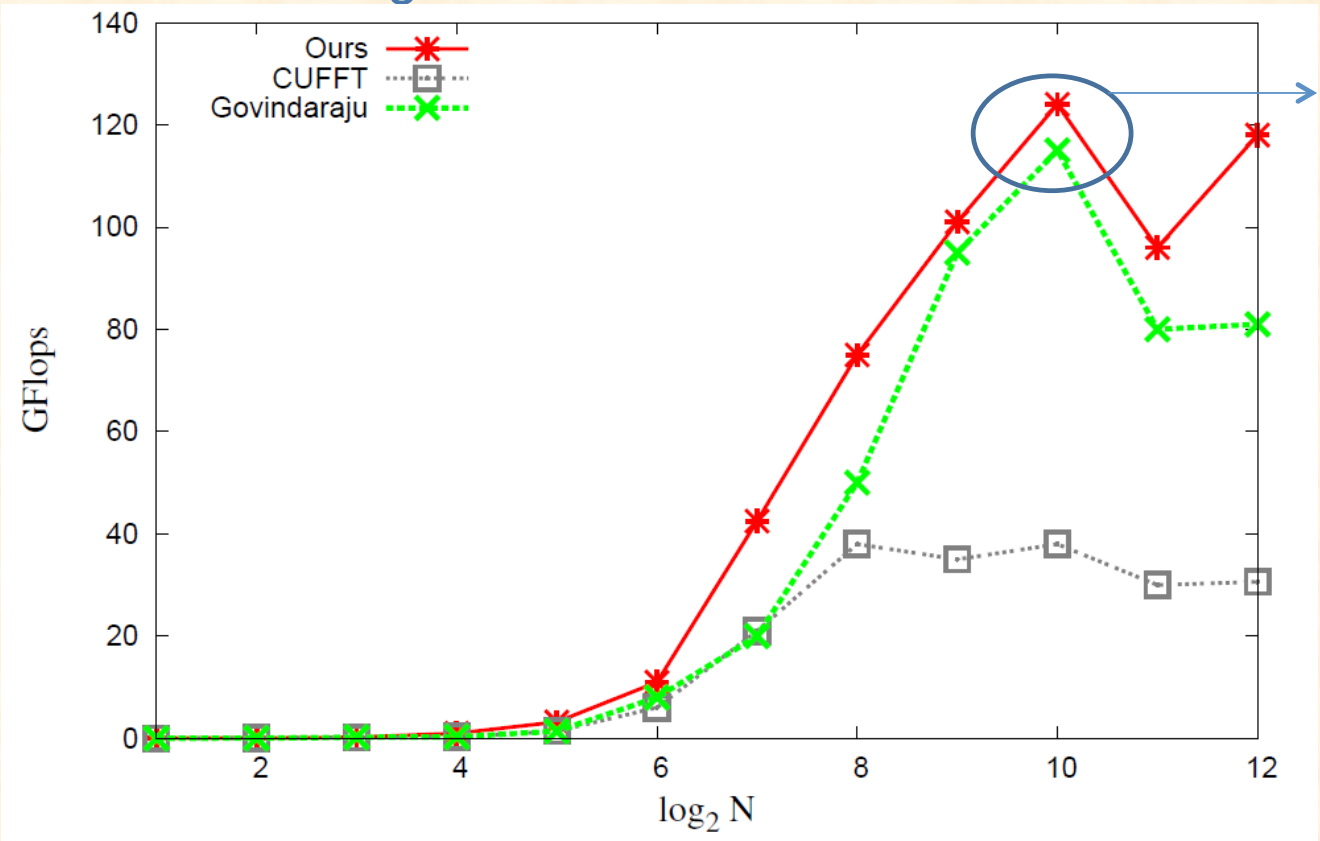
•Govindaraju's SC08 paper
•Nukada's SC09 paper (auto-tuning)
•CUFFT-2.2 and 2.3

Gflops are defined as the following for a D dimensional FFT($M=N_1N_2...N_D$)

$$GFlops = \frac{5M\sum_{d=1}^{D} log_2 N_d}{t} * 10^{-9}$$

Evaluation(GTX280-2D)

- *Performance of single 2D FFT of size NxN*

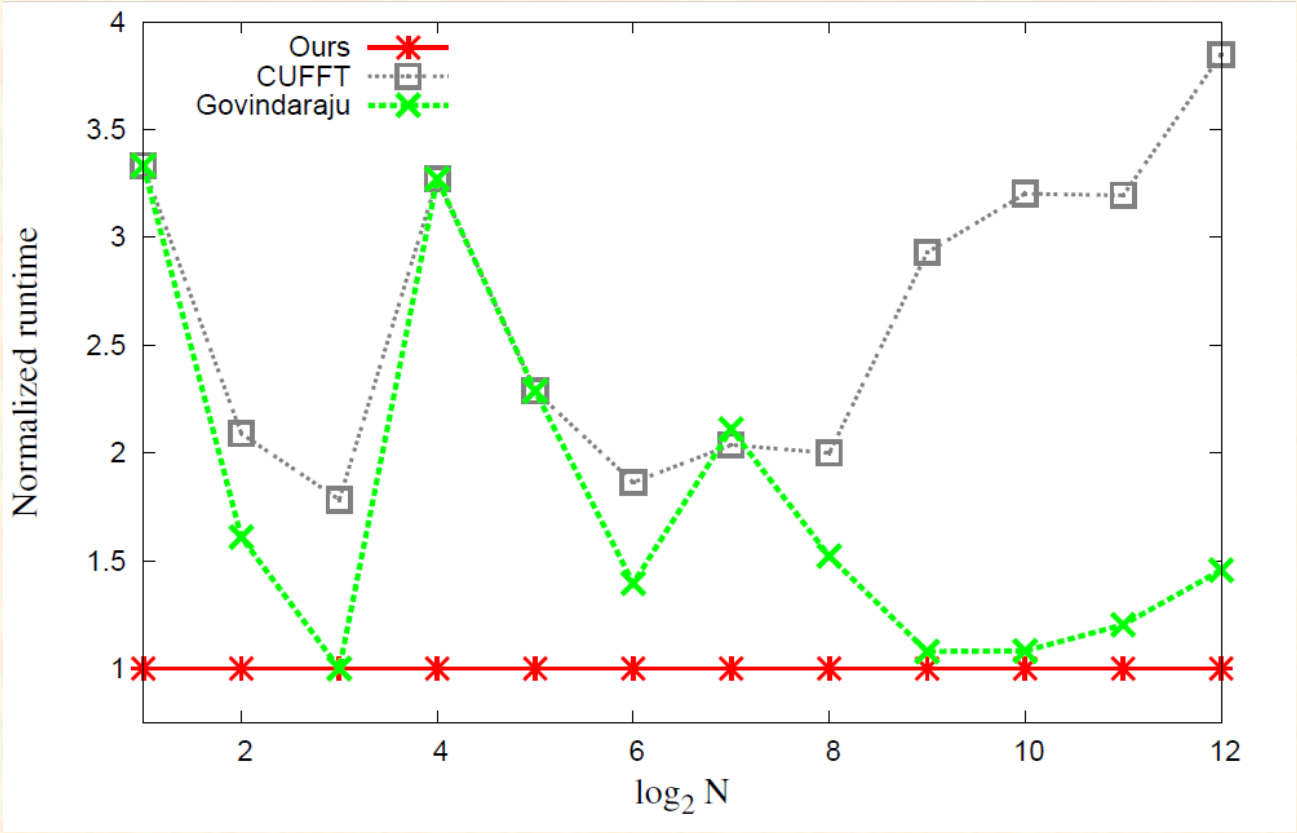


X dimensional kernel fits into shared memory

- Govindaraju's data is read from his SC08 paper ;
- CUFFT version is 2.2

Evaluation(GTX280-2D)

- *Normalized runtime of single 2D FFT NxN*

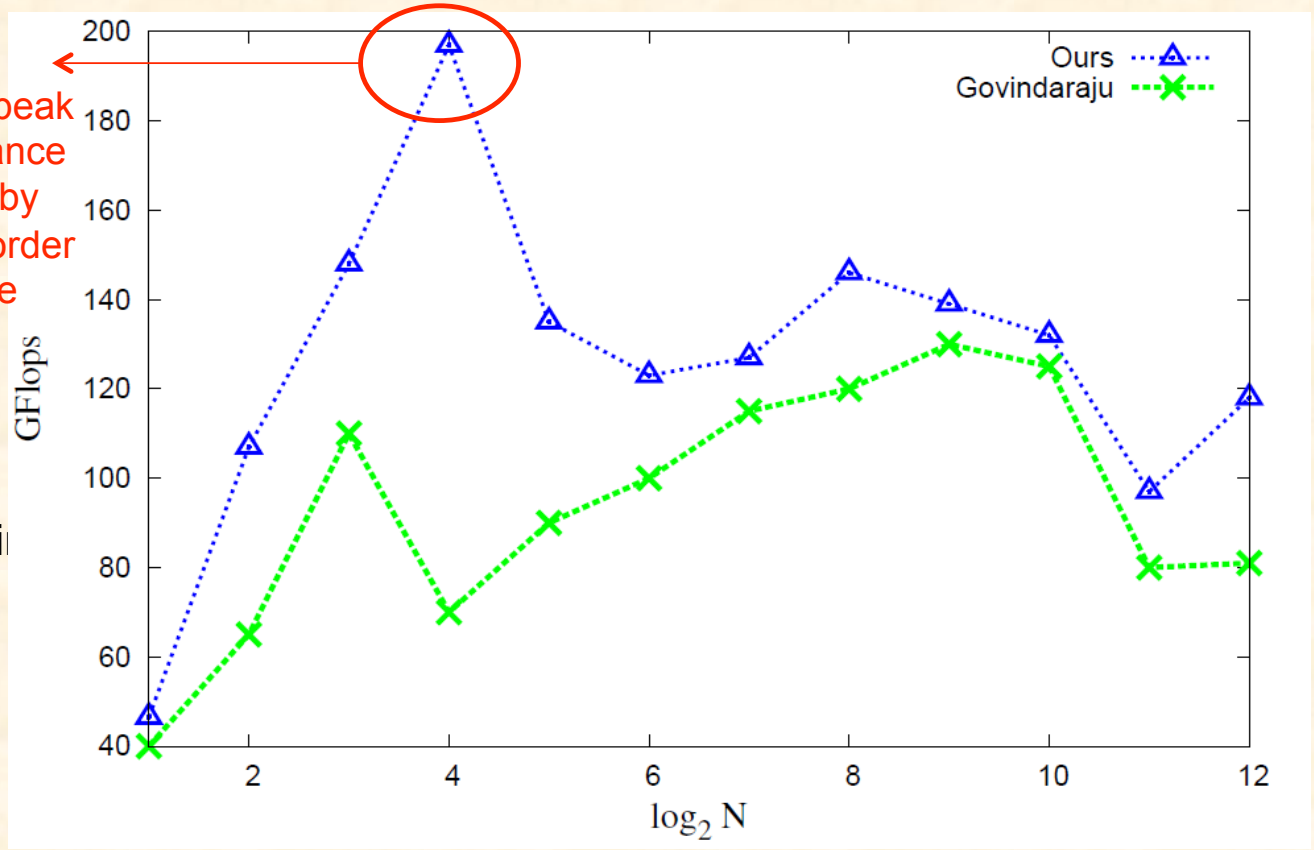


Average speedup:
2.65x over CUFFT
1.78x over Govidaraju

- Largest FFT size is limited by 1GByte global memory
- Codelet grouping technique greatly improves performance of small FFTs

Evaluation(GTX280-2D)

➢ *Performance of batched 2D FFT of size NxN*



Govi

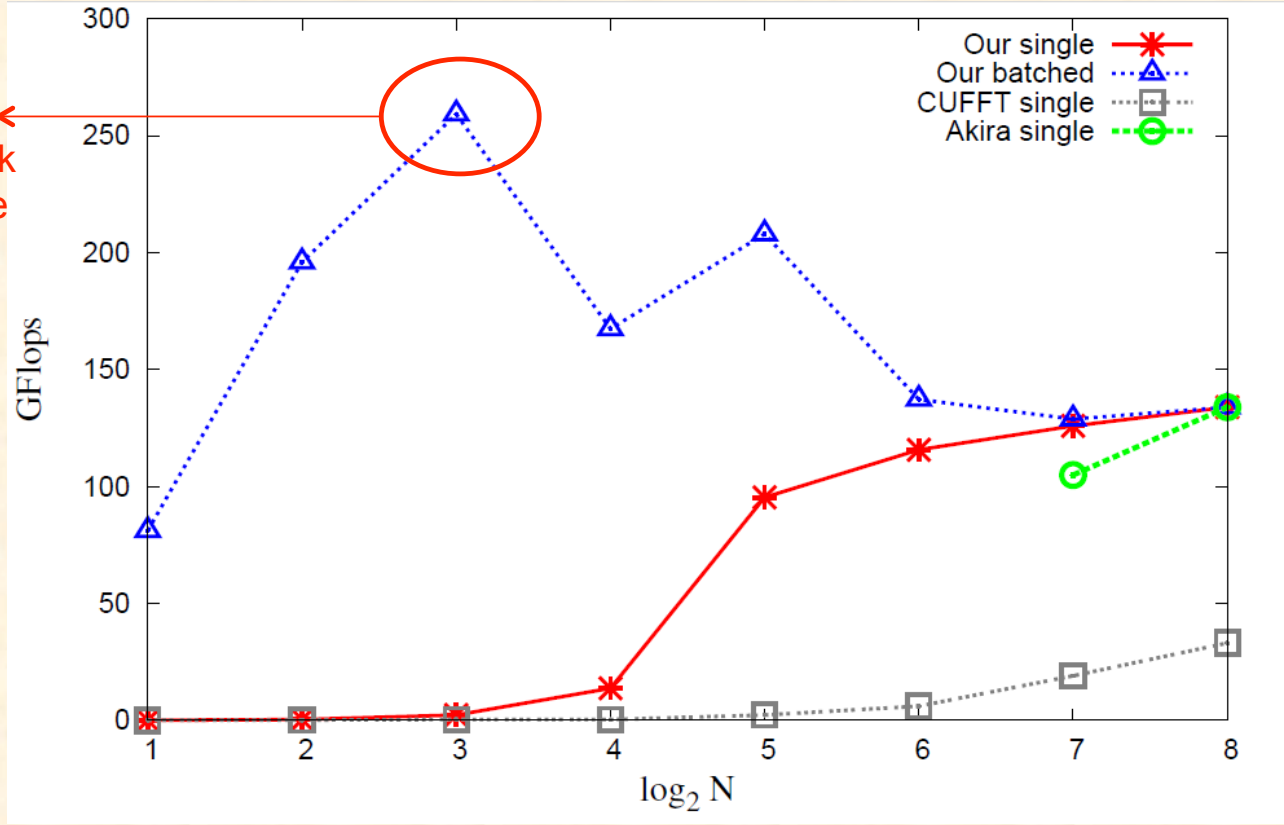
Average speedup: 2.8x over

- Tested batch size is $2^{24}/N^2$
- CUFFT-2.2 does not support batched execution
- Codelet order exchange enable the peak performance of 16X16

Evaluation(GTX280-3D)

➢ *Performance of 3D FFT of size NxNxN*

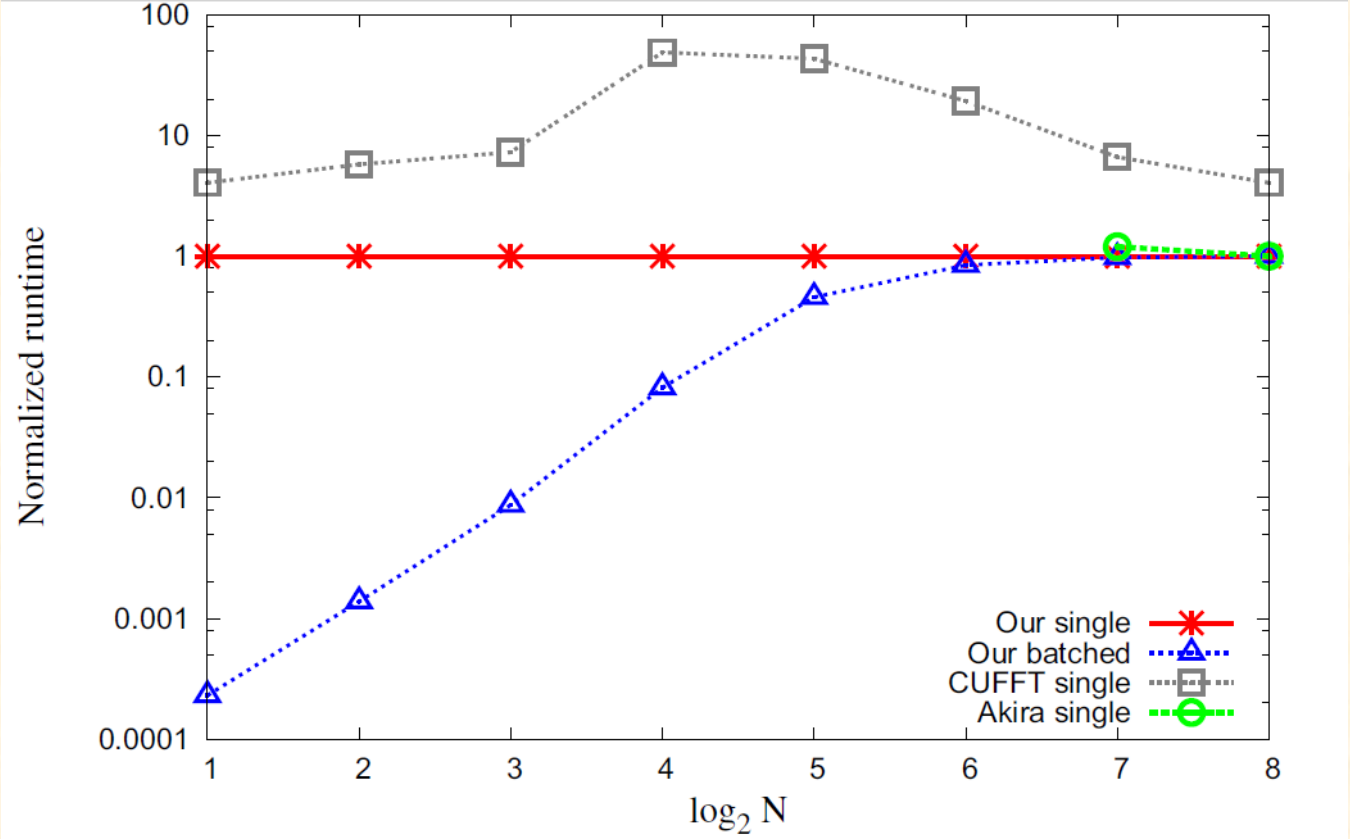
3D FFT peak performance 259GFlops



•Nukada's data is read from his SC09 paper ;
•CUFFT version is 2.2

Evaluation(GTX280-3D)

➢ *Normalized runtime of 3D FFT of size NxNxN*



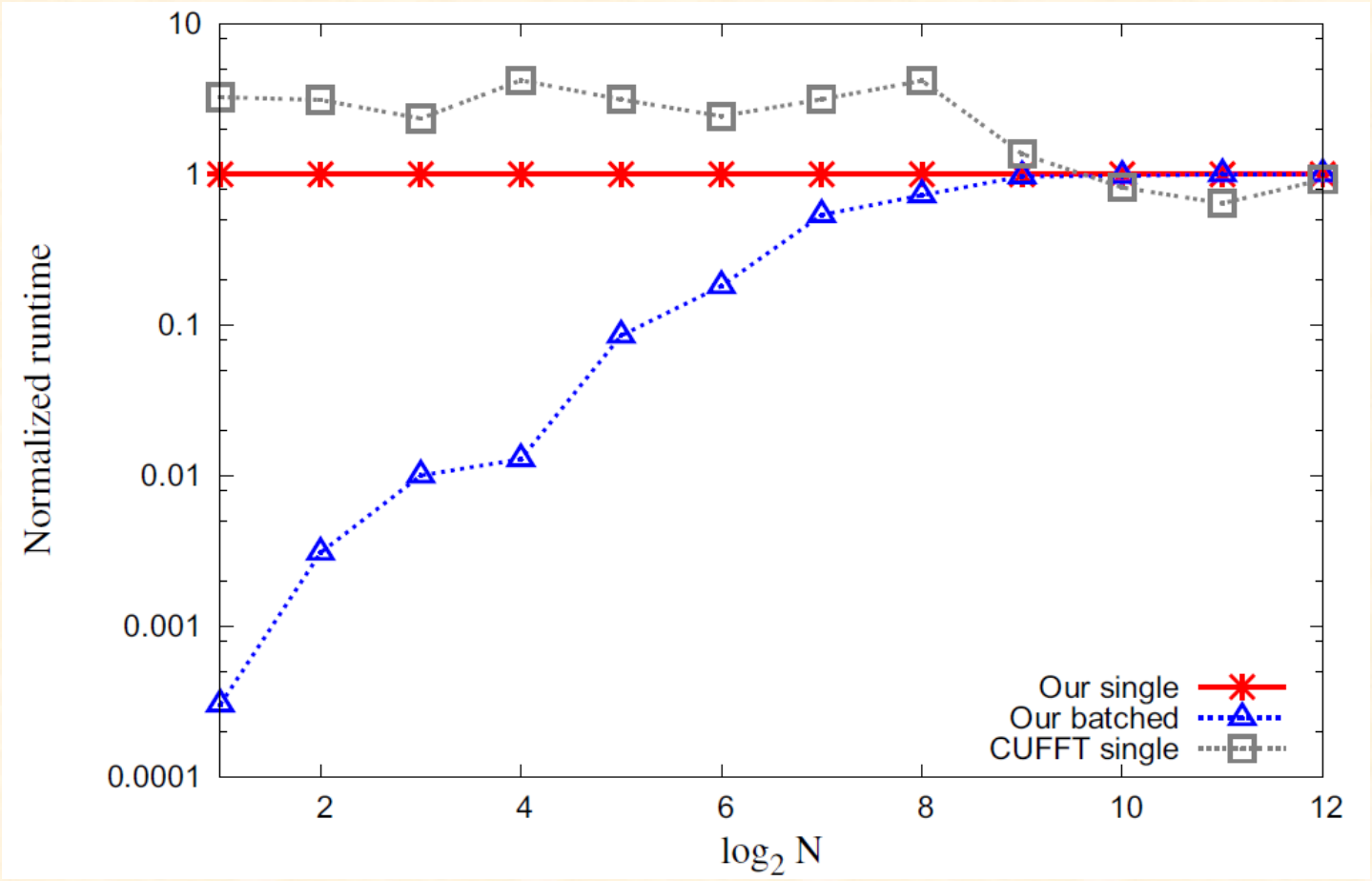
Average speedup:
17.4x over CUFFT

- Our 3D FFT has much greater speedup over CUFFT than 2D FFT
- Nukada's work shows good performance

Evaluation(FX5600-2D)

- 16 multiprocessors, compute capability 1.0 and CUFFT-2.3

➢ *Normalized runtime of 2D FFT of size NxN*

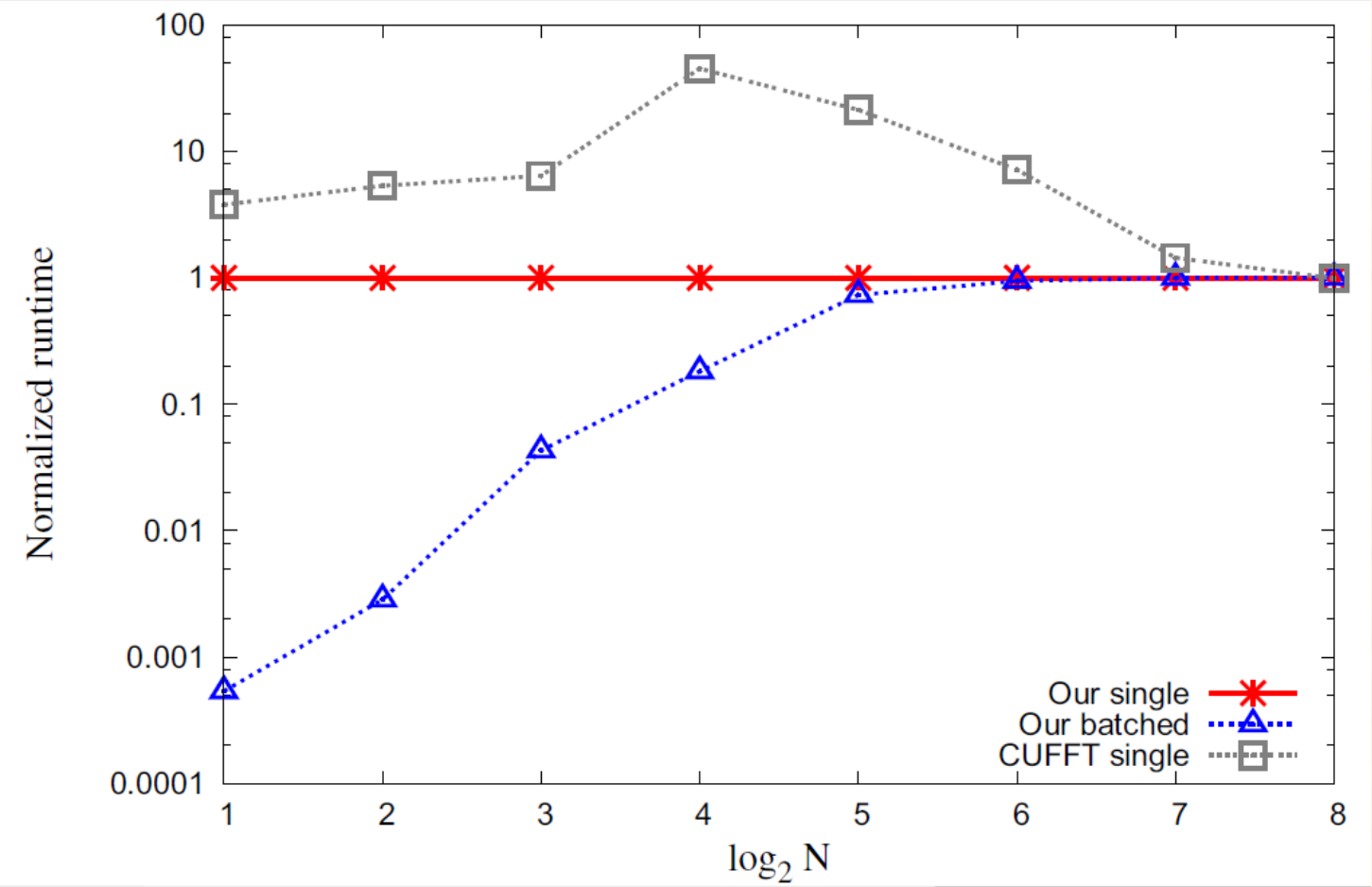


Average speedup:
2.5x over CUFFT

- *CUFFT-2.3 performs better than our work on FX 5600 for large sizes*

Evaluation(FX5600-3D)

- *Normalized runtime of 3D FFT of size NxNxN*



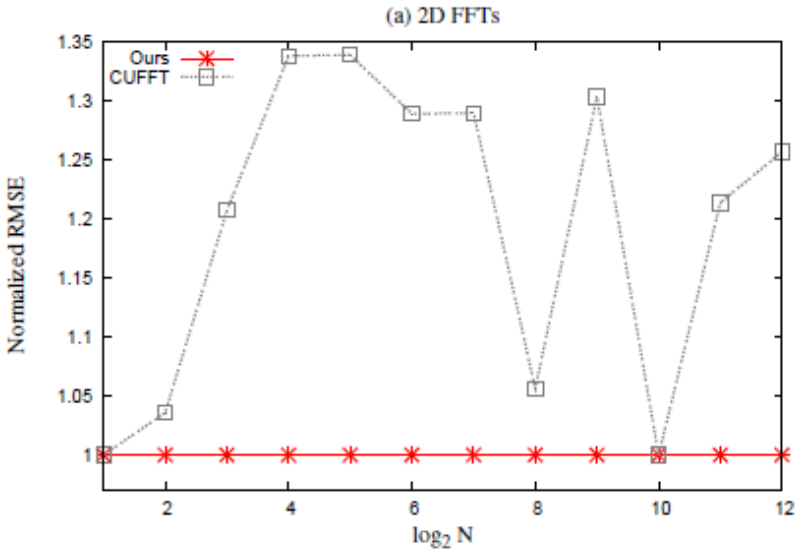
Average speedup:
11.5x over CUFFT

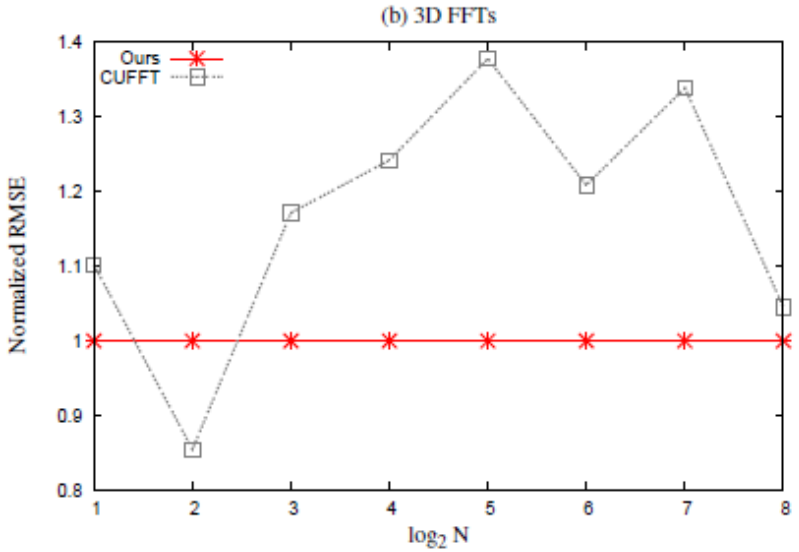
- *Our 3D FFT has greater speedup over CUFFT than 2D FFT*

Evaluation(precision)

- *FFT output is compared with FFTW double precision*

Normalized Root Mean Square Error(RMSE) is compared with CUFFT on GTX280





- In most cases, we achieve better precision on GTX280
- Pre-compute twiddle factor on CPU improves precison

Conclusion and Future Works

➢ *Conclusion*

•An extended I/O tensor representation are proposed for multi-dimensional FFT

•CUDA GPU constraints are expressed in I/O tensor format

•Codelet decomposition, grouping and commutability techniques are explored

•A 2D and 3D FFT library is built on this search space and it achieves great speedup over previous works

•This framework is not limited to CUDA

➢*Future works*

•Extend to non-power-of-two sizes

•Enable auto tuning to achieve better results in the search space

•Include PCI bus transfer time and handle larger FFTs

•Open source the library

Conclusion and Future Works

➢ *Conclusion*

•An extended I/O tensor representation are proposed for multi-dimensional FFT

•CUDA GPU constraints are expressed in I/O tensor format

•Codelet decomposition, grouping and commutability techniques are explored

•A 2D and 3D FFT library is built on this search space and it achieves great speedup over previous works

•This framework is not limited to CUDA

➢*Future works*

•Extend to non-power-of-two sizes

•Enable auto tuning to achieve better results in the search space

•Include PCI bus transfer time and handle larger FFTs

•Open source the library

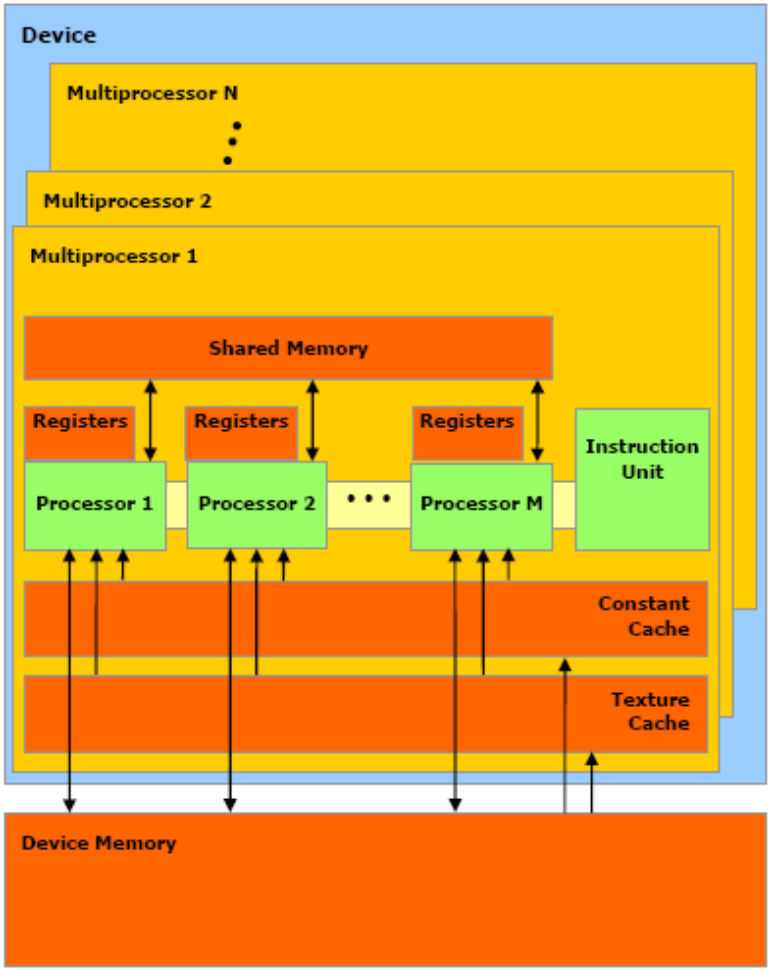
Questions?

Appendix: CUDA Review

➢*CUDA: general purpose programming architecture*

Each GPU has a number of Streaming Multiprocessors(SM) , each containing 8 Scalar Processor (SP)

E.g. GTX280 has 30 SMs, 240SPs, 16K registers and 16K Bytes shared memory and 1G Byte Global memory for the whole device.



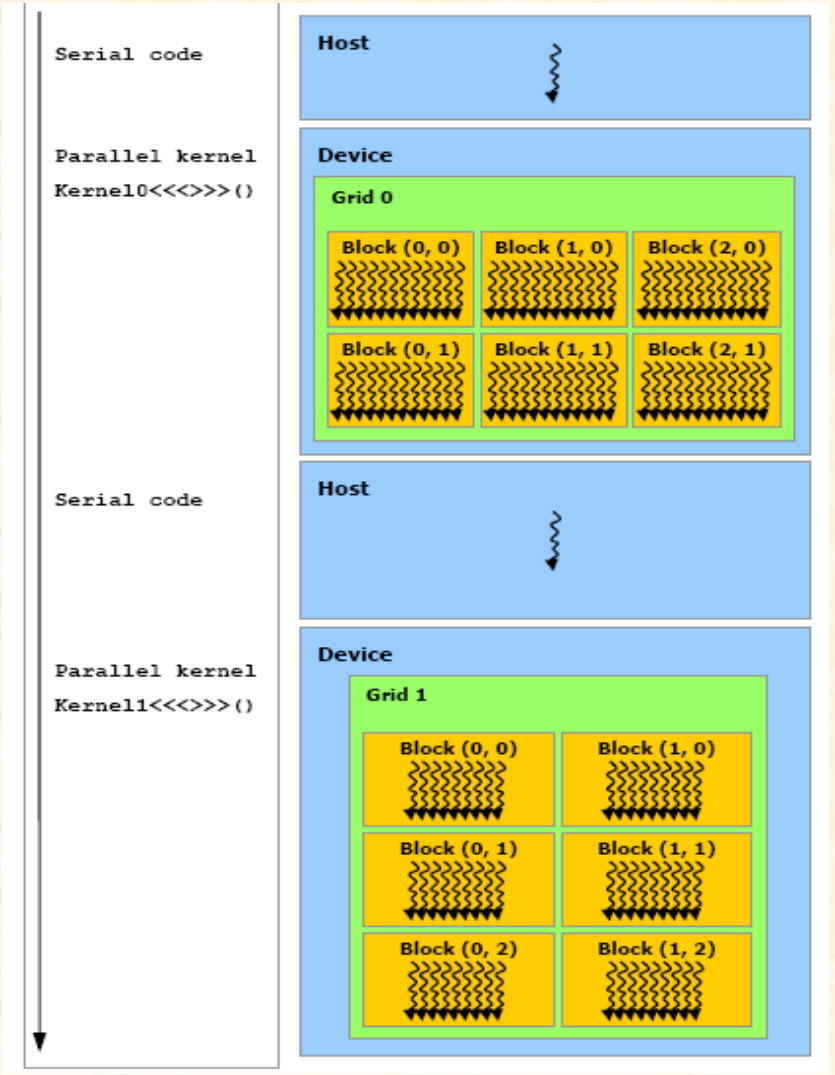
Courtesy: NVIDIA

Appendix: CUDA Review

- *Programming model*

A kernel contains some Grids, Blocks and threads. Threads in a block are executed in single instruction multiple threads(SIMT) fashion.

Threads are allowed to diverge while at a cost of performance decrease.



Courtesy: NVIDIA

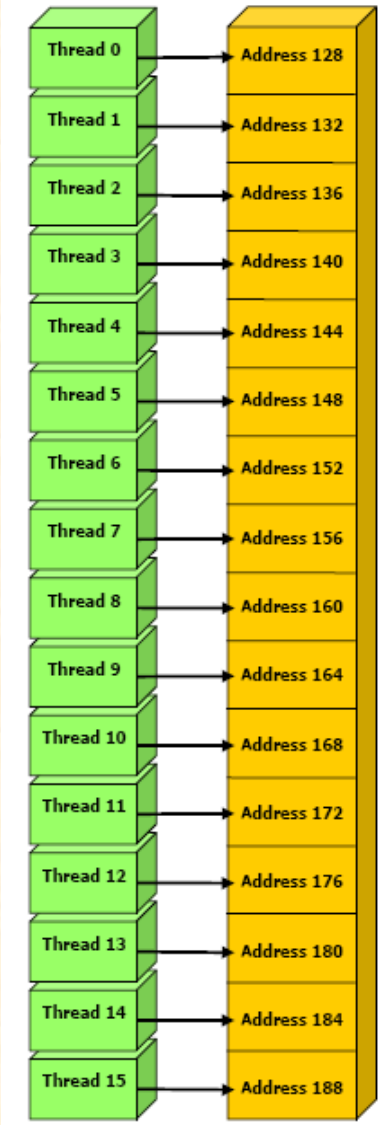
Appendix: CUDA Review

Courtesy: NVIDIA

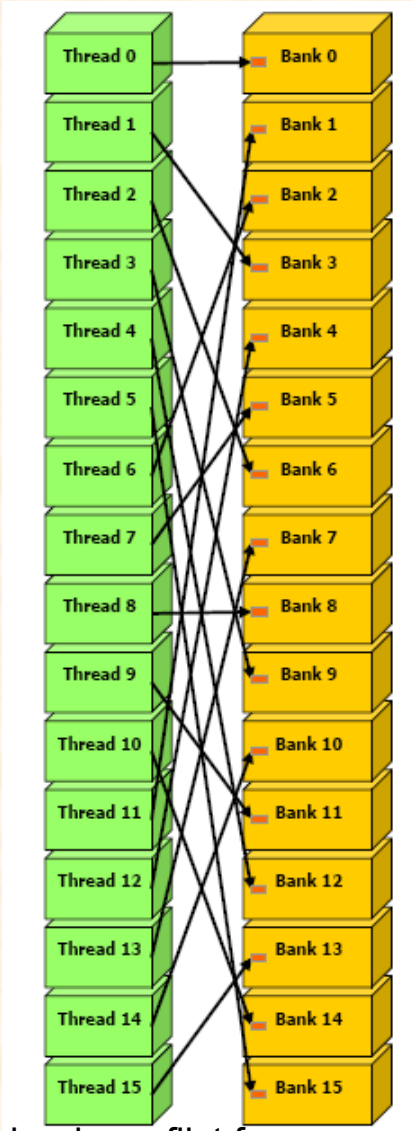
➢ *Memory performance*

Global memory is of high priority in optimization. The left figure shows a simple coalesced access for a half warp.

Shared memory is almost as fast as register when there is no bank conflict in its 16 banks.



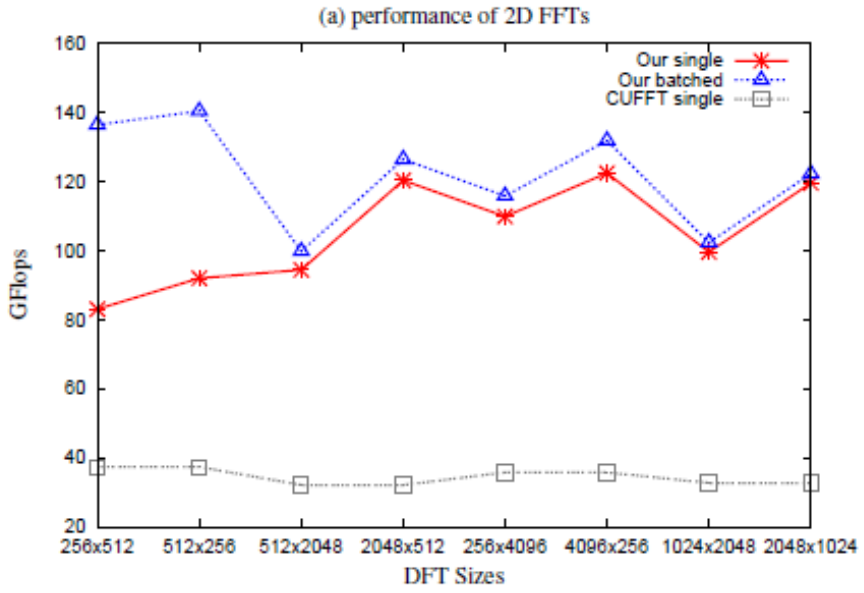
One Coalesced access pattern on global memory

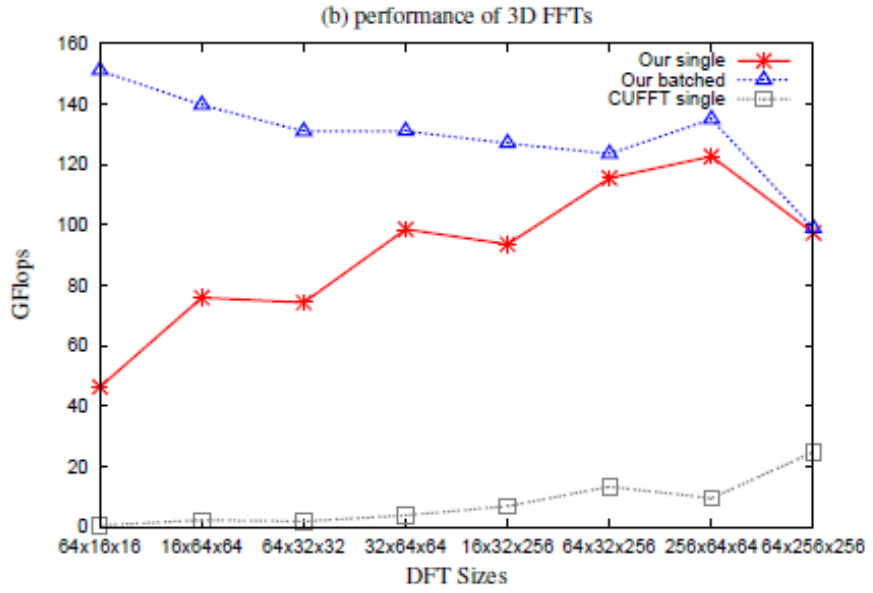


A bank conflict free access pattern on shared memory

Evaluation(GTX280)

- *2D FFT of size $N_1 x N_2$ and 3D FFT of size $N_1 x N_2 x N_3$ on GTX280*





Speedup: 3x over CUFFT

28x over CUFFT