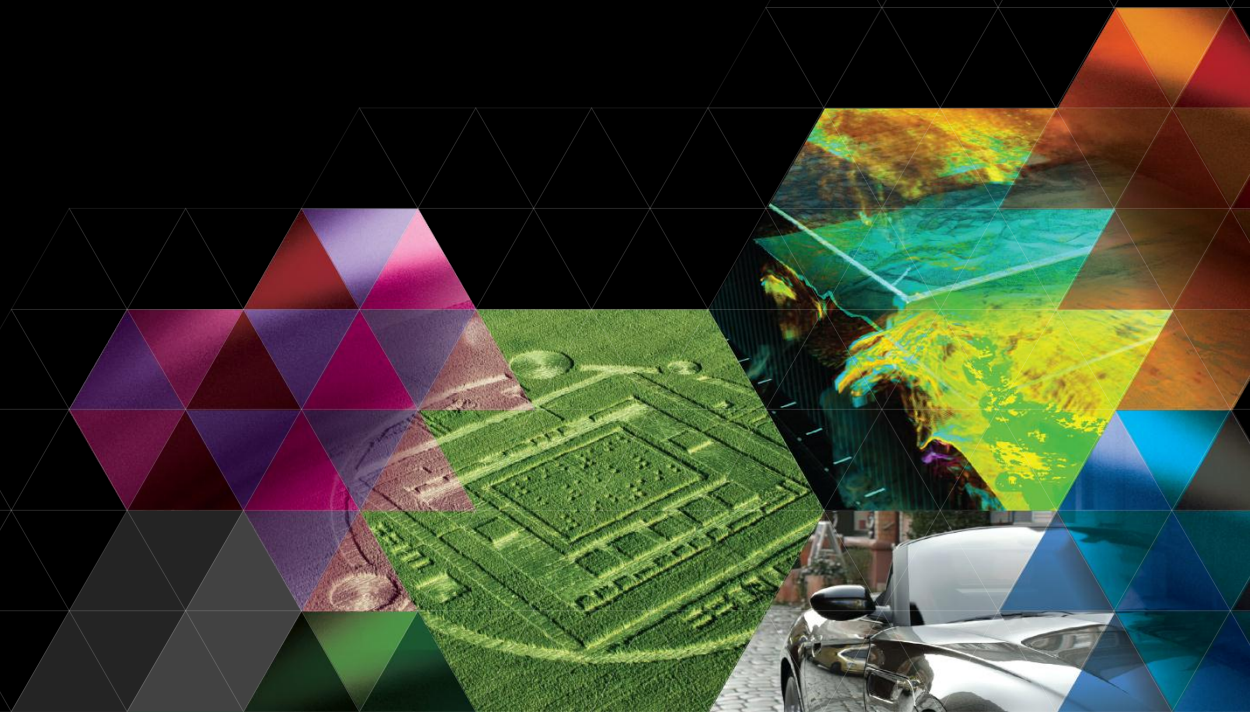
GPU TECHNOLOGY CONFERENCE

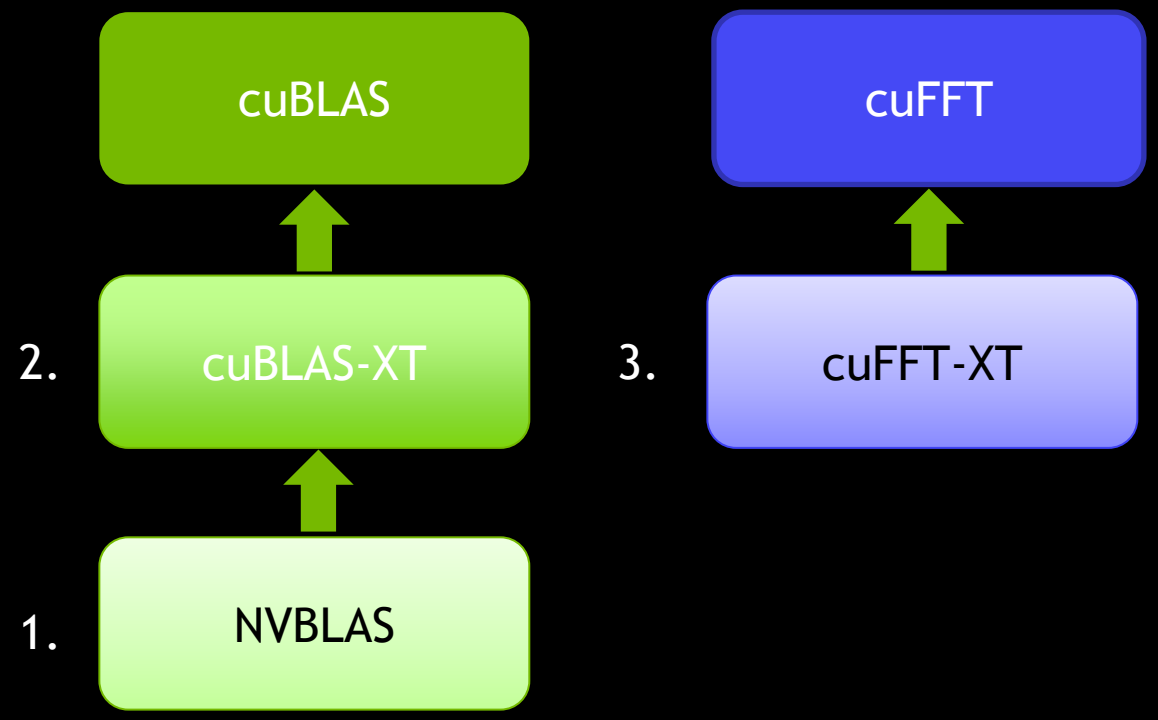
NVIDIA

RAPID MULTI-GPU PROGRAMMING WITH CUDA LIBRARIES

Nikolay Markovskiy



CUDA 6



WHAT IS NVBLAS?

- Drop-in replacement of BLAS
  - Built on top of cuBLAS-XT
  - BLAS Level 3
- Zero coding effort
  - R, Octave, Scilab , etc
- Limited only by amount of host memory
  - Large data sets on single GPU
  - PCI transfers can become the bottleneck in complicated scenarios

NVBLAS SUPPORTED API

Routine	Types	Operation
gemm	S,D,C,Z	multiplication of 2 matrices
syrk	S,D,C,Z	symmetric rank-k update
herk	C,Z	hermitian rank-k update
syr2k	S,D,C,Z	symmetric rank-2k update
her2k	C,Z	hermitian rank-2k update
trsm	S,D,C,Z	triangular solve with multiple right-hand sides
symm	S,D,C,Z	symmetric matrix-matrix multiplication
hemm	C,Z	hermitian matrix-matrix multiplication

OCTAVE

- Assignment 1
  - ‘cd $HOME/nvblas’
  - ‘vim ./sgemm.m’
  - Generate random data for matrices A and B:

```
N = 4096;
A = single(rand(N*N)) ;
B = single(rand(N*N)) ;
```

  - Matrix product and time the code:

```
beginTime  =  clock();
C = A * B;
elapsed = etime(clock(), start);
disp(elapsed);
```

FLOPS

- ‘cp sgemm.m sgemm-flops.m’
- ‘vim sgemm-flops.m’
- Find FLOPS:
  - Number of operations(ADD, MULTIPLY) / elapsedTime
- ‘srun octave ./sgemm-flops.m’

FLOPS

- ‘srun octave ./sgemm-flops.m’

- Calculate FLOPS:
  - Number of operations(ADD, MULTIPLY) / elapsedTime

```
gFlops = (2*N*N*N) / ( elapsedTime * 1e+9);
disp(gFlops);
```

~ 2.4 GFLOPS with bundled BLAS

cp progress/sgemm-flops.m .

OPTIMIZED BLAS

- Assignment 2
- ‘vim ./run_cpu.sh’
- Library should be in LD_LIBRARY_PATH

```
export OMP_NUM_THREADS=2
LD_PRELOAD=libopenblas.so octave ./sgemm-flops.m
```

- ‘srun octave ./run_cpu.sh’

OPTIMIZED BLAS

- Assignment 2

```
export OMP_NUM_THREADS=2
LD_PRELOAD=libopenblas.so octave ./sgemm-flops.m
```

~ 83.0 GFLOPS with OpenBLAS (2 threads)

~380 GFLOPS with OpenBLAS on Ivy Bridge socket

OPTIMIZED BLAS

- Assignment 2.5

```
export OMP_NUM_THREADS=2
LD_PRELOAD=libopenblas.so octave ./sgemm-flops.m
```

N = 8192;

~84 GFLOPS with OpenBLAS (2 threads)

~390 GFLOPS with OpenBLAS on Ivy Bridge socket

NVBLAS

- Configuration file is mandatory
  - **export** NVBLAS_CONFIG_FILE=$FILE_NAME
  - Default: ‘nvblas.conf’ in current dir
- Mandatory for nvblas.conf:
  - Path to CPU BLAS library: NVBLAS_CPU_BLAS_LIB
    - Use LD_LIBRARY_PATH if needed
- NVBLAS_GPU_LIST 0 1

Keyword	Meaning
ALL	All compute-capable GPUs detected on the system will be used by NVBLAS
ALL0	GPU device 0, AND all others GPUs detected that have the same compute-capabilities as device 0 will be used by NVBLAS

NVBLAS

- Assignment 3: Accelerate on 1 and 2 GPUs
  - Find and open ‘nvblas.conf’ in current directory

```
NVBLAS_CPU_BLAS_LIB libopenblas.so
NVBLAS_LOGFILE nvblas.log
```

  - ‘srun octave ./run_gpu.sh’

```
LD_PRELOAD=$CUDA_PATH/lib64/libnvblas.so octave ./sgemm-flops.m
```

or

```
LD_PRELOAD=libnvblas.so octave ./sgemm-flops.m
```

NVBLAS

- Assignment 3: Accelerate on 1 and 2 GPUs
  - Find and open ‘nvblas.conf’ in current directory

```
NVBLAS_CPU_BLAS_LIB libopenblas.so
NVBLAS_LOGFILE nvblas.log
NVBLAS_GPU_LIST ALL
```

  - Execute your script:

```
LD_PRELOAD=libnvblas.so octave ./sgemm-flops.m
```

~ **380/660 GFLOPS with NVBLAS** on 8192 problem

NVBLAS

- Assignment 4: Optimize
  - Find and open ‘nvblas.conf’ in current directory

```
NVBLAS_CPU_BLAS_LIB OpenBLAS/libopenblas.so
NVBLAS_LOGFILE nvblas.log
NVBLAS_GPU_LIST ALL
NVBLAS_AUTOPIN_MEM_ENABLED
```

  - Execute your script:

```
LD_PRELOAD=libnvblas.so octave ./sgemm-flops.m
```

NVBLAS_TILE_DIM 4096

NVBLAS

- Assignment 4: Optimize
  - Find and open ‘nvblas.conf’ in current directory

```
NVBLAS_CPU_BLAS_LIB OpenBLAS/libopenblas.so
NVBLAS_LOGFILE nvblas.log
NVBLAS_GPU_LIST ALL
NVBLAS_AUTOPIN_MEM_ENABLED
```

  - Execute your script:

```
LD_PRELOAD=libnvblas.so octave ./sgemm-flops.m
```

~ 1590 GFLOPS with NVBLAS on 8192 problem

One can also increase NVBLAS_TILE_DIM

NVBLAS: MORE OPTIONS

- NVBLAS_GPU_DISABLED_<BLAS_FUNC_NAME>
  - E.g. NVBLAS_GPU_DISABLED_SGEMM
- NVBLAS_CPU_RATIO_<BLAS_FUNC_NAME>
  - E.g. NVBLAS_CPU_RATIO_SGEMM 0.07

- See CUDA documentation for more information
  - cuda-6.0/doc/pdf/NVBLAS_Library.pdf

CUBLAS-XT

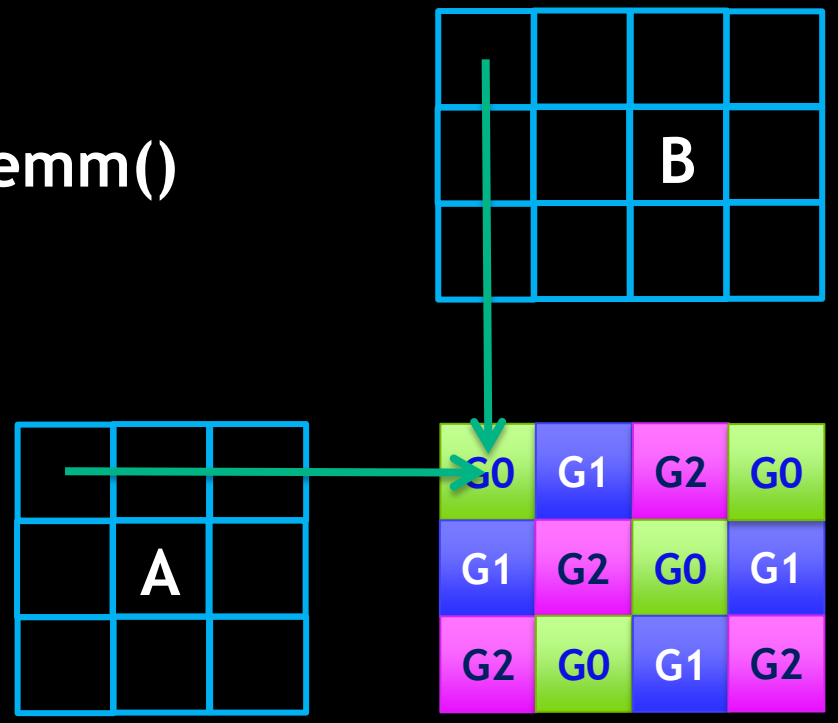
- Level 3 BLAS

Routine	Types	Operation
gemm	S,D,C,Z	multiplication of 2 matrices
syrk	S,D,C,Z	symmetric rank-k update
herk	C,Z	hermitian rank-k update
syr2k	S,D,C,Z	symmetric rank-2k update
her2k	C,Z	hermitian rank-2k update
trsm	S,D,C,Z	triangular solve with multiple right-hand sides
symm	S,D,C,Z	symmetric matrix-matrix multiplication
hemm	C,Z	hermitian matrix-matrix multiplication

CUBLAS-XT

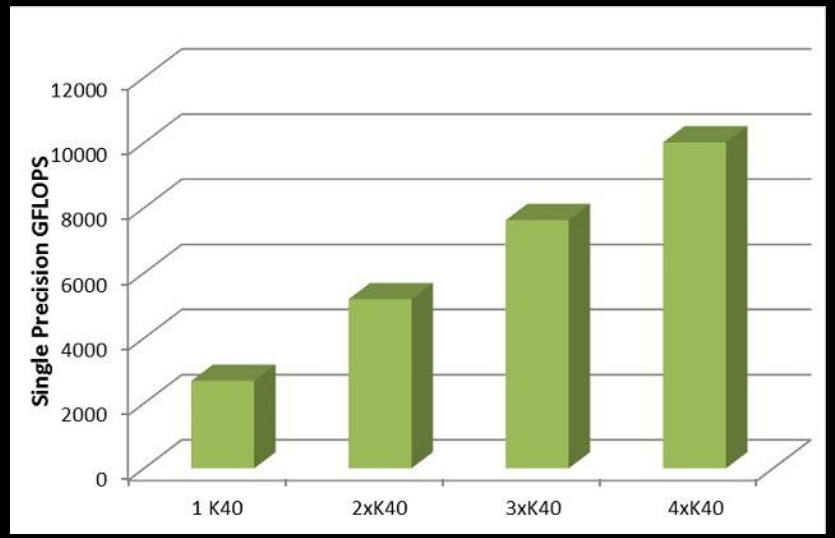
- Host interface
  - Tiling strategy
- Example **cublasXt<t>gemm()**

- Tiling for 3 GPUs:



CUBLAS-XT

- Two versions of library
- CUDA 6.0 version: limited to Gemini boards (Tesla K10, GeForce GTX 690)
- Premium version: https://developer.nvidia.com/cublasxt
- 64 bit (UVA support)
- Hybrid CPU-GPU computation
  - Currently **cublasXt<t>gemm()**



- Problem size can be larger than available GPU memory

CUBLAS WORKFLOW

```
#include <cublas_v2.h>
// Allocate and fill h_A and h_B with data:
…
cublasCreate(&handle);
cudaMalloc(&d_A, mem_size_A);
… // Allocate d_B, d_C
cudaMemcpy(d_A, h_A, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, cudaMemcpyHostToDevice);

cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, M, N, K,
&alpha, d_A, M, d_B, K, &beta, d_C, M);

cudaMemcpy(h_C, d_C, cudaMemcpyDeviceToHost);
cudaFree(…);
cublasDestroy(handle);
```

CUBLAS-XT EXERCISE 1

- Start from cuBLAS example
  - ‘cd $HOME/cublas-xt’
  - ‘vim ./cublas.cpp’
- Build cublas.cpp and run
  - See instructions.txt on how to build
  - g++ -O2 -I$CUDA_HOME/include -L$CUDA_HOME/lib64 -lcublas -lcudart cublas.cpp
  - ‘srun a.out’

CUBLAS-XT EXERCISE 1

- Start from cuBLAS example
  - ‘cd $HOME/cublas-xt’
  - ‘vim ./cublas.cpp’
- Build cublas.cpp and run
  - See instructions.txt on how to build
  - g++ -O2 -I$CUDA_HOME/include -L$CUDA_HOME/lib64 -lcublas -lcudart cublas.cpp
  - ‘srun a.out’

~ 1050 GFLOPS

CUBLAS-XT WORKFLOW

```
// Allocate and fill h_A and h_B with data:
#include <cublasXt.h>
…
cublasXtHandle_t handle;
cublasXtCreate(&handle);
const int nDevices = 2;
int deviceId[nDevices] = {0, 1};

cublasXtDeviceSelect(handle, nDevices, deviceId);

cublasXtSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, M, N,
K, &alpha, h_A, M, h_B, K, &beta, h_C, M);

cublasXtDestroy(handle);
```

CUBLAS-XT EXERCISE 2

- cp cublas.cpp cublas-xt.cpp
- Implement the same functionality using XT features
- g++ -I$CUDA_HOME/include -L$CUDA_HOME/lib64 -lcublas -lcudart cublas-xt.cpp
- Execute on 1 and 2 GPUs

CUBLAS-XT EXERCISE 2

- cp cublas.cpp cublas-xt.cpp
- Implement the same functionality using XT features
- g++ -I$CUDA_HOME/include -L/$CUDA_HOME/lib64 -lcublas -lcudart cublas-xt.cpp
- Execute on 2 GPUs

~ 772 GFLOPS on 2 GPUs

cp progress/cublas-xt.cpp .

CUBLAS-XT EXERCISE 3

- Optimize cublas-xt.cpp

```
// Use pinned memory
cudaMallocHost((void **) &h_A, mem_size_A);
cudaFreeHost((void *) h_A);
// Optional: optimize tile size
int blockDim;
// Get current tile size
cublasXtGetBlockDim(handle, &blockDim);
// Set tile size
cublasXtSetBlockDim(handle, blockDim);
```

CUBLAS-XT EXERCISE 3

- Optimize cublas-xt.cpp

~ **2000** GFLOPS using `cudaMallocHost`

CUBLAS-XT EXERCISE 3

- Optimize cublas-xt.cpp

~ **2000** GFLOPS using `cudaMallocHost`

Alternative route:

```
cublasXtSetPinningMemMode(handle,
CUBLASXT_PINNING_ENABLED);
```

Calls `cudaHostRegister`/`cudaHostUnregister` on the fly

~ 1700 GFLOPS

cp progress/cublas-opt-xt.cpp .

CUBLAS-XT: MORE OPTIONS

- cublasXt(G,S)etPinningMemMode(…)
- cublasXtSetCpuRatio(…)
- Other cublasXt API Math Functions

- See CUDA documentation for more information
  - cuda-6.0/doc/pdf/CUBLAS_Library.pdf

CUFFT-XT

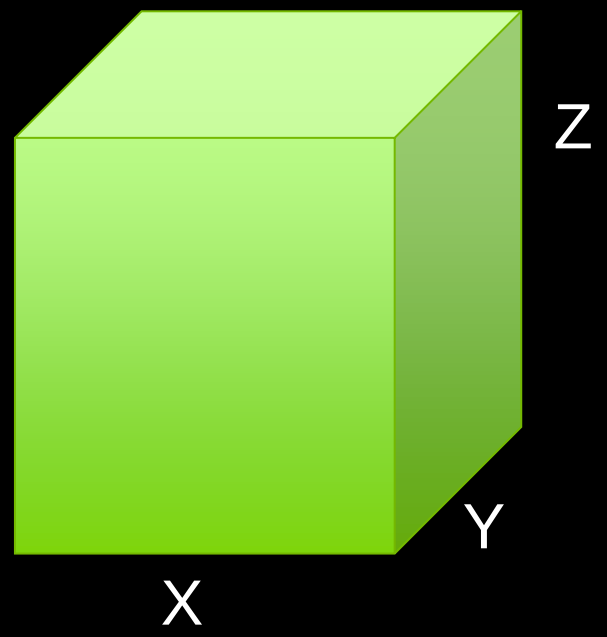
- CUFFT-XT API uses device pointers for data
  - Assumes app is allocating device buffers
  - Assumes app is copying to/from host memory as needed
- Need to manage (alloc/free/copy) buffers on multiple devices
  - Use cufftXT cudaMalloc/cudaFree with library-specific policies
  - Use cufftXT cudaMemcpy to transition to/from library-specific data organizations
- Different parallelization strategies (e.g. batched, 1D, 3D)

CUFFT-XT

- Limited to Gemini boards (e.g. K10)
- The GPUs must support the Unified Virtual Address Space
- On Windows, the GPU board must be operating in Tesla Compute Cluster (TCC)
- When number of batches 1
  - Only **C2C** and **Z2Z** transform types are supported
  - Only in-place transforms are supported
  - The transform dimensions must be powers of 2
  - The size of the transform must be greater than or equal to 32

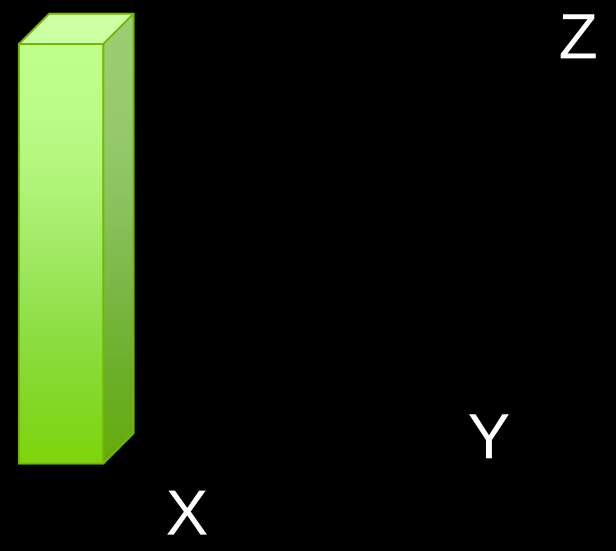
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z
  - X * Z 1D transforms of size Y
  - Y * Z 1D transforms of size X



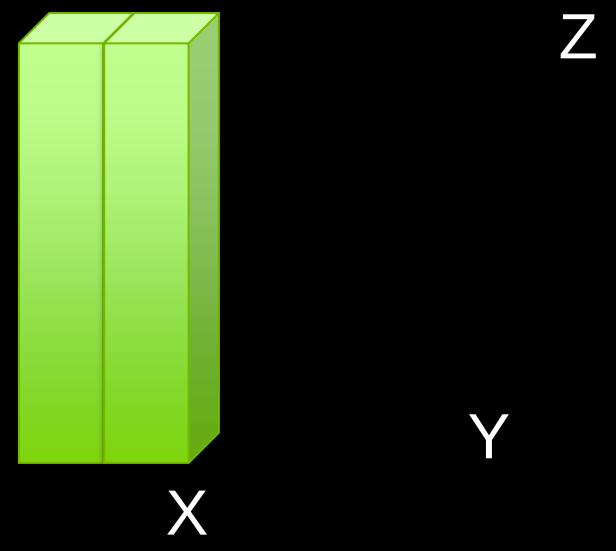
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



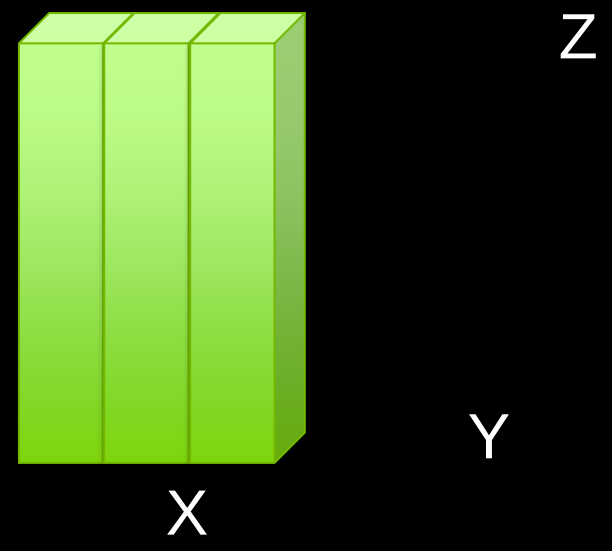
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



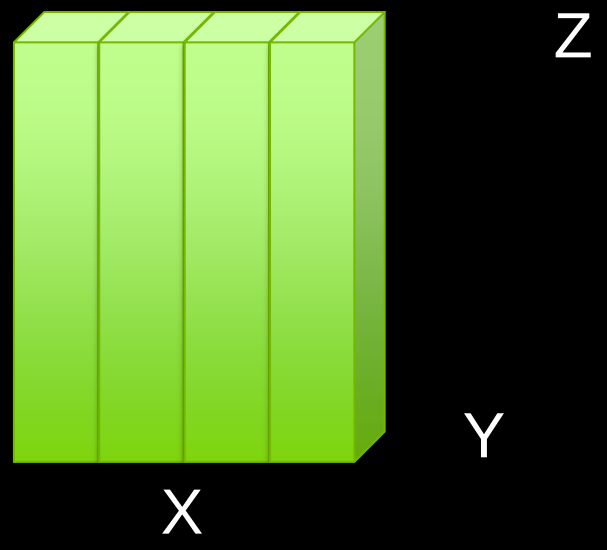
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



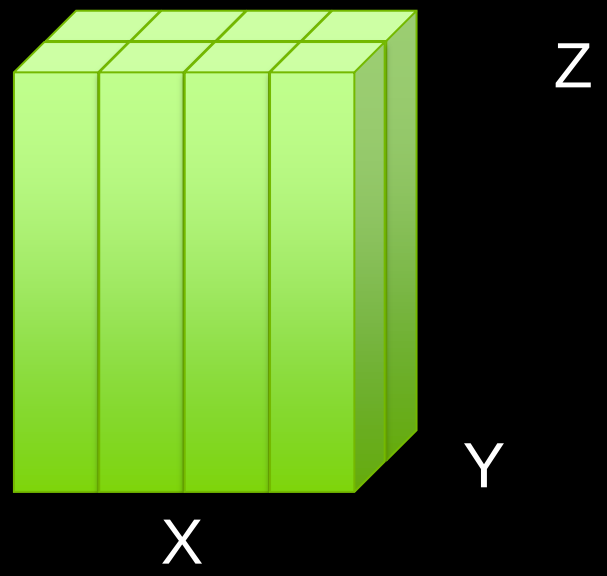
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



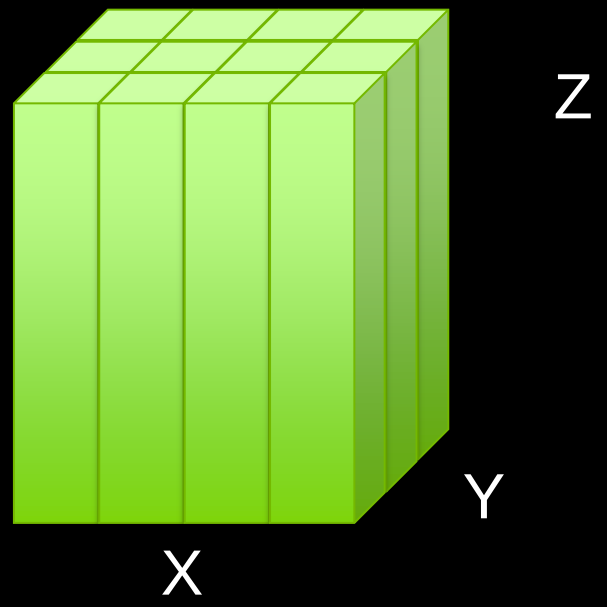
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



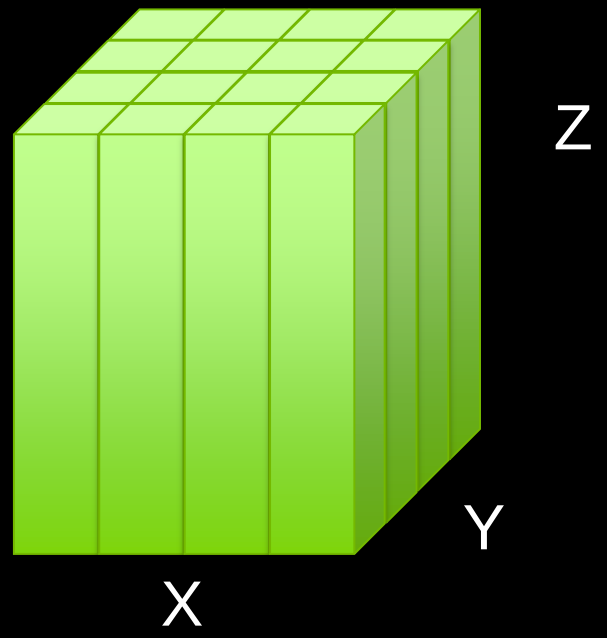
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



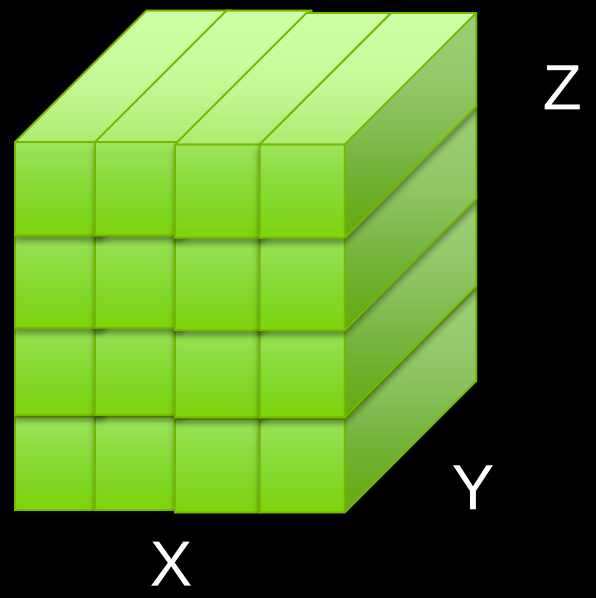
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z



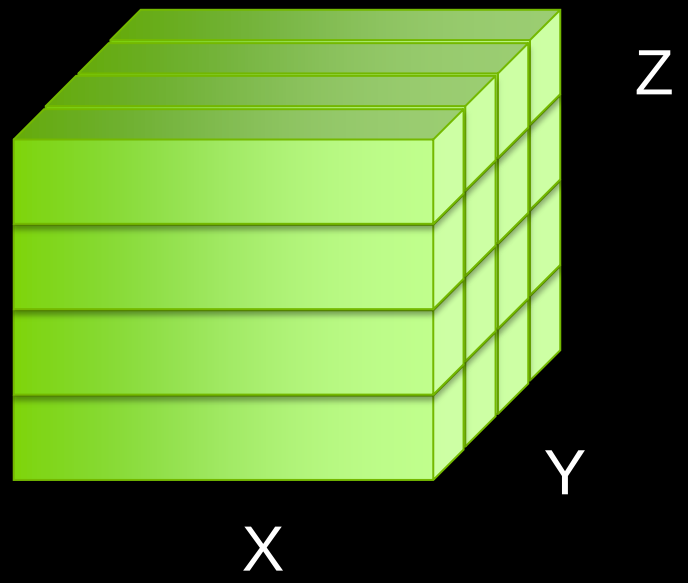
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z
  - X * Z 1D transforms of size Y



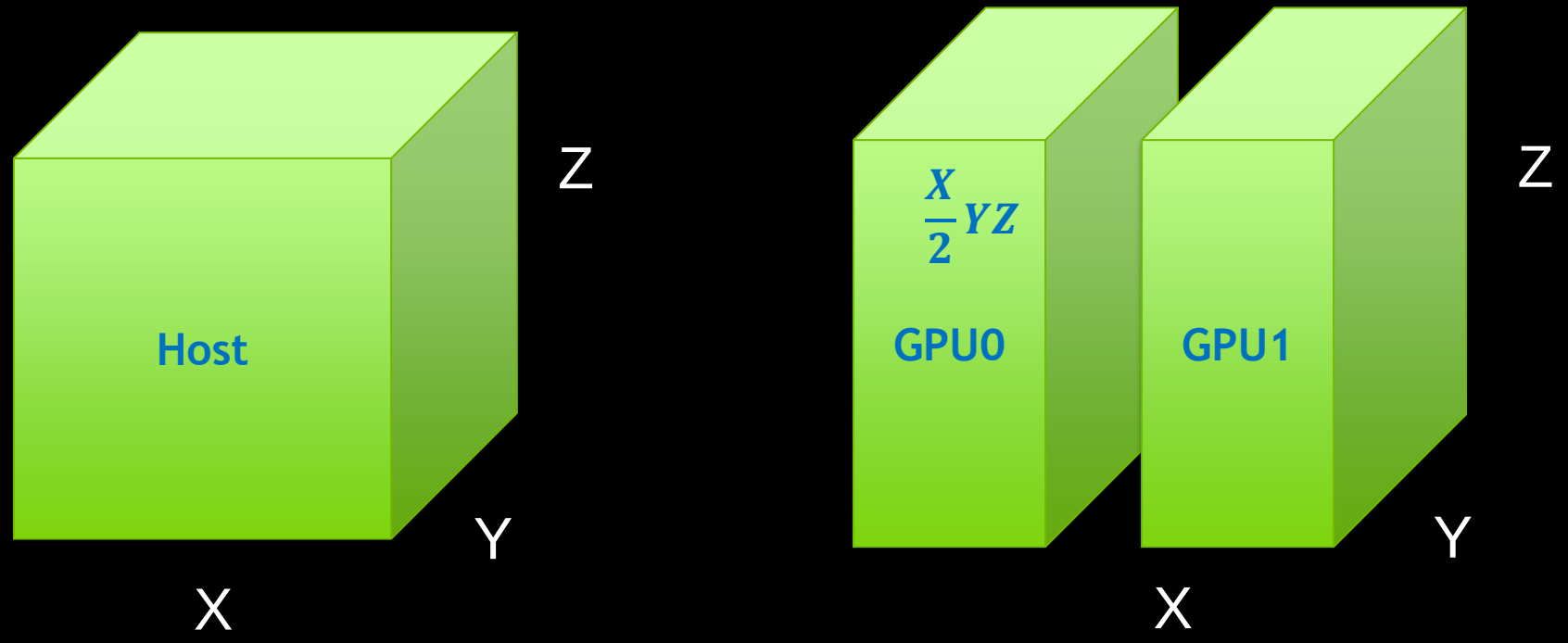
A 3D FFT PRIMER

- In CUFFT, 3D FFTs are done by performing 1D FFTs in each dimension
- Eg: X by Y by Z 3D transform on MyData[X][Y][Z]:
  - X * Y 1D transforms of size Z
  - X * Z 1D transforms of size Y
  - Y * Z 1D transforms of size X



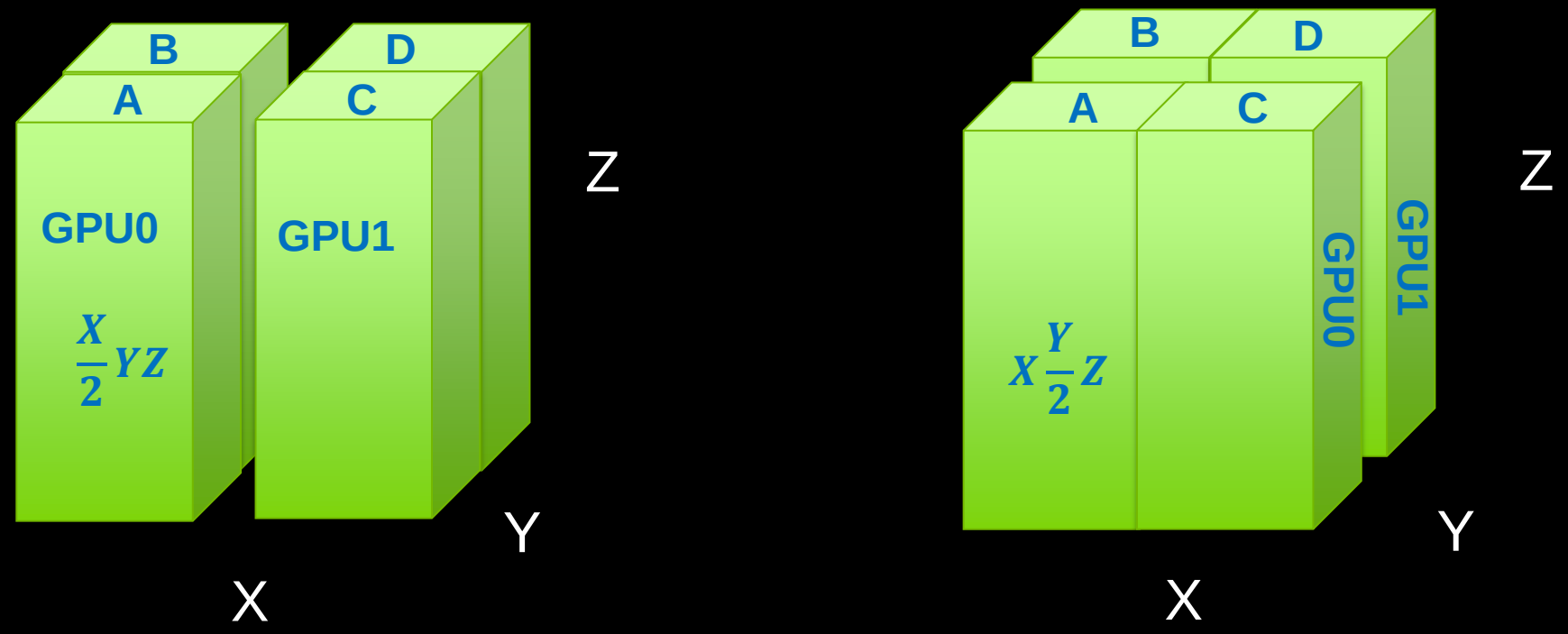
MULTI-GPU VERSION, STEP 1

- Copy Slabs from Host to GPUs
- X/2 * Y 1D transforms of size Z per GPU
- X/2 * Z 1D transforms of size Y per GPU



MULTI-GPU VERSION, STEP 2

- Copy Columns Between GPUs
- Y/2 * Z 1D transforms of size X per GPU



CUFFT WORKFLOW

```
// Populate h_signal
…
cufftHandle plan; cufftResult ret; cudaError_t error;
ret = cufftPlan3d(&plan, nx, ny, nz, CUFFT_C2C);

cufftComplex *d_signal_in;
cudaMalloc(&d_signal_in, mem_size);
cudaMemcpy(d_signal_in, h_signal, mem_size, cudaMemcpyHostToDevice);

cufftExecC2C(plan, d_signal_in , d_signal_in, CUFFT_FORWARD);

cudaFree(d_signal_in);
cufftDestroy(plan);
```

CUFFT EXERCISE

- Open cufft3d.cpp example
  - ‘cd $HOME/cufft-xt’
  - ‘vim cufft3d.cpp‘
- Build and run
  - g++ -I$CUDA_HOME/include -L$CUDA_HOME/lib64 -lcufft -lcudart cufft3d.cpp
  - ‘srun a.out’

CUFFT EXERCISE

- Open cufft3d.cpp example
  - ‘cd $HOME/cufft-xt’
  - ‘vim cufft3d.cpp‘
- Build and run
  - g++ -I$CUDA_HOME/include -L$CUDA_HOME/lib64 -lcufft -lcudart cufft3d.cpp
  - ‘srun a.out’

~ 163 GFLOPS or 12.3 msec

CUFFT-XT WORKFLOW

```
// Populate h_signal
…
cufftHandle plan; cufftResult ret;
ret = cufftCreate (&plan);
const int nDevices = 2; int deviceId[nDevices] = {0,1};
size_t workSize[nDevices];
ret = cufftXtSetGPUs(plan, nDevices, deviceIds);
cufftMakePlan3d(plan, nx, ny, nz, CUFFT_C2C, workSize);

cudaLibXtDesc *d_signal_in;
cufftXtMalloc(plan, &d_signal_in, CUFFT_XT_FORMAT_INPLACE);
cufftXtMemcpy(plan, d_signal_in, h_signal, CUFFT_COPY_HOST_TO_DEVICE);

cufftXtExecDescriptorC2C(plan,d_signal_in,d_signal_in,CUFFT_FORWARD);

cufftXtFree(d_signal_in);
cufftDestroy(plan);
```

CUFFT-XT WORKFLOW

```
cufftResult cufftCreate (cufftHandle *plan);
```

Input	
plan	Pointer to a **cufftHandle** object

Output	
plan	Contains a cuFFT plan handle value

CUFFT-XT WORKFLOW

```
cufftResult cufftXtSetGPUs (cufftHandle plan, int nGPUs,
int *whichGPUs);
```

Input	
plan	**cufftHandle** returned by **cufftCreate**
nGPUs	Number of GPUs to use
whichGPUs	The GPUs to use

CUFFT-XT WORKFLOW

```
cufftResult cufftXtMalloc (cufftHandle plan,
cudaLibXtDesc **descriptor, cufftXtSubFormat format);
```

Input	
plan	**cufftHandle** returned by **cufftCreate**
descriptor	Pointer to a pointer to a **cudaLibXtDesc** object
format	**cufftXtSubFormat** value

Output	
descriptor	Pointer to a pointer to a **cudaLibXtDesc** object

CUFFT-XT WORKFLOW

```
cufftResult cufftXtMemcpy (cufftHandle plan, void
*dstPointer, void *srcPointer, cufftXtCopyType type);
```

Input	
plan	**cufftHandle** returned by **cufftCreate**
dstPointer	Pointer to the destination address
srcPointer	Pointer to the source address
type	**cufftXtCopyType** value

Output	
idata	Contains the complex Fourier coefficients

CUFFT-XT WORKFLOW

```
cufftResult cufftXtExecDescriptorC2C(cufftHandle plan,
cudaLibXtDesc *idata, cudaLibXtDesc *idata, int direction);
```

Input	
plan	**cufftHandle** returned by **cufftCreate**
idata	Pointer to the complex input data (in GPU memory)
idata	Pointer to the complex output data (in GPU memory)
direction	The transform direction: **CUFFT_FORWARD** or **CUFFT_INVERSE**

Output	
idata	Contains the complex Fourier coefficients

CUFFT-XT EXERCISE 1

- Implement forward FFT on 2 GPUs
  - Start from 3d fft example ‘cp cufft3d.cpp cufft3d-xt.cpp’
  - Modify accordingly
  - Build and run on 2 GPUs

CUFFT-XT WORKFLOW

```
// Populate h_signal
…
cufftHandle plan; cufftResult ret;
ret = cufftCreate (&plan);
const int nDevices = 2; int deviceId[nDevices] = {0,1};
size_t workSize[nDevices];
ret = cufftXtSetGPUs(plan, nDevices, deviceIds);
cufftMakePlan3d(plan, nx, ny, nz, CUFFT_C2C, workSize);

cudaLibXtDesc *d_signal_in;
cufftXtMalloc(plan, &d_signal_in, CUFFT_XT_FORMAT_INPLACE);
cufftXtMemcpy(plan, d_signal_in, h_signal, CUFFT_COPY_HOST_TO_DEVICE);

cufftXtExecDescriptorC2C(plan,d_signal_in,d_signal_in,CUFFT_FORWARD);

cufftXtFree(d_signal_in);
cufftDestroy(plan);
```

CUFFT EXERCISE

- Implement forward FFT on 2 GPUs
  - Start from 3d fft example “cp cufft3d.cpp cufft3d-xt.cpp“
  - Modify accordingly
  - Build and run on 2 GPUs

~ 264 GFLOPS or 7.6 msec

cp progress/cufft3d-xt.cpp .

CUFFT EXERCISE 2

- 1D FFT

```
cufftMakePlan1d(plan, signal_size, CUFFT_C2C, 1, workSize);
```

CUFFT EXERCISE 2

- 1D FFT

```
cufftMakePlan1d(plan, signal_size, CUFFT_C2C, 1, workSize);
```

~ 145 GFLOPS or 13.9 msec

cp progress/cufft1d-xt.cpp .

CUFFT EXERCISE 3

- Copy the data to natural order on GPUs

```
cufftXtMemcpy (plan, d_signal_out, d_signal_in,
CUFFT_COPY_DEVICE_TO_DEVICE);

cufftXtMemcpy (plan, h_signal, d_signal_out,
CUFFT_COPY_DEVICE_TO_HOST);
```

vim progress/cufft1d-nat-xt.cpp

CUFFT EXERCISE 4

- Batched FFT

```
int n[nDevices], rank = 1, batch = nDevices;
int istride, idist, ostride, odist;
int *inembed = NULL, *onembed = NULL;
n[0] = signal_size / nDevices;
cufftMakePlanMany (plan, rank, n, inembed, istride,
idist, onembed, ostride, odist, CUFFT_C2C, batch,
workSize);
```

cp progress/cufft1d-xt.cpp cufft1d-batch-xt.cpp`

CUFFT EXERCISE 4

- Batched FFT

```
int n[nDevices], rank = 1, batch = nDevices;
int istride, idist, ostride, odist;
int *inembed = NULL, *onembed = NULL;
n[0] = signal_size / nDevices;
cufftMakePlanMany (plan, rank, n, inembed, istride,
idist, onembed, ostride, odist, CUFFT_C2C, batch,
workSize);
```

~ 384 GFLOPS or 5.2 msec

cp progress/cufft1d-batch-xt.cpp .

CUFFT-XT: MORE FEATURES

- cufftXtQueryPlan(…)
  - Get factorization for conversion between permuted and linear indexes
  - Allows to work with shuffled data
- More advanced use patterns in documentation

- See CUDA documentation for more information
  - cuda-6.0/doc/pdf/CUFFT_Library.pdf

GPU TECHNOLOGY CONFERENCE

NEXT STEPS

- Download CUDA
  - https://developer.nvidia.com/cuda-toolkit
- Join the community
  - https://developer.nvidia.com/cuda-zone
- Some relevant future lab sessions
  - S4870 - Getting More Parallelism Out of Multiple GPUs
  - S4792 - Leveraging Accelerated Core Algorithms Using NVIDIA AmgX
  - S4791 - Building a Sparse Linear Solver using CUDA Libraries