The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program

of the Lobachevsky State University of Nizhni Novgorod

among the world's research and education centers

Strategic initiative

“Achieving leading positions in the field of supercomputer technology

and high-performance computing”

Introduction to GPU programming

Lecture 8. CUDA libraries

Nizhni Novgorod

2014

Author: S.I. Bastrakov

OBJECTIVES

The objective of this lecture is to introduce three CUDA libraries: CUBLAS, CUFFT, CURAND, that can be used to develop and optimize a wide set of applications.

ABSTRACT

This lecture considers three libraries that are part of CUDA Toolkit: CUBLAS, CUFFT, and CURAND. We present basic functionality of these libraries and different ways to use them.

BRIEF OVERVIEW

Acronym BLAS stands for Basic Linear Algebra Subprograms, a standard created in 1979. It is base for LAPACK benchmarks. There are many optimized implementations on Fortran and C, including Goto BLAS, BLAS in Intel MKL, BLAS in ACML, and CUBLAS in CUDA Toolkit. All BLAS procedure names have structure <character code><name><mode>.

General concept of using CUBLAS is calling routines (not kernels) from cublas.h from host side, link with cublas.lib. Data is transferred between CPU and GPU using special functions: cublasInit, cublasShutdown, cublasAlloc, cublasFree, cublasSetVector, cublasGetVector, cublasSetMatrix, cublasGetMatrix. Here are some examples of BLAS computational routines: cublasScopy, cublasSdot, cublasSaxpy, cublasSgemv.

CUFFT is a library implementing various versions of fast Fourier transform. As with BLAS, there are many optimized implementations, most notable FFTW and FFT in Intel MKL. CUFFT supports 1D, 2D, and 3D transforms, in-place and out-of-place.

To use CUFFT one needs to include cufft.h, link with cufft.lib. Data is stored and computing is performed on GPU. Interface of CUFFT is very similar to FFTW. Supported data types: cufftReal = float (R), cufftDoubleReal = double (D), cufftComplex = float2 (C), cufftDoubleComplex = double2 (Z). Supported transform types: CUFFT_R2C, CUFFT_C2R, CUFFT_C2C, CUFFT_D2Z, CUFFT_Z2D, CUFFT_Z2Z, and transform directions: CUFFT_FORWARD и CUFFT_INVERSE.

Each transform is described by plan. Plan has cufftHandle type and is created by one of the following routines: cufftPlan1d, cufftPlan2d, cufftPlan3d, cufftPlanMany, and destroyed with cufftDestroy. Transform is executed by one of the following routines: cufftExecC2C, cufftExecR2C, cufftExecC2R, cufftExecZ2Z, cufftExecD2Z, cufftExecZ2D.

Here is an example of in-place forward and inverse transform for 10 single-precision complex signals of size 256:

```
#define NX 256
#define BATCH 10
cufftHandle plan;
cufftComplex *data;
cudaMalloc((void**)&data, sizeof(cufftComplex)*NX*BATCH);
cufftPlan1d(&plan, NX, CUFFT_C2C, BATCH);
cufftExecC2C(plan, data, data, CUFFT_FORWARD);
cufftExecC2C(plan, data, data, CUFFT_INVERSE);
cufftDestroy(plan);
cudaFree(data);
```

CURAND is optimized library for preudo- and quasi-random number generation on host and GPU introduced in 2010. It consists of 2 parts: Host API and Device API.

To use Host API include curand.h, link with curand.lib. There are routines to generate random numbers: curandCreateGenerator/curandCreateGeneratorHost, curandSetPseudoRandomGeneratorSeed, curandDestroyGenerator, curandGenerate, curandGenerateNormal, curandGenerateUniform, curandGenerateUniformDouble. Example of using Host API to generate 100 random floats with commentary:

```
size_t n = 100;
size_t i;
curandGenerator_t gen;
float *devData, *hostData;
/* Allocate n floats on host */
hostData = (float *)calloc(n, sizeof(float));
/* Allocate n floats on device */
cudaMalloc((void **)&devData, n * sizeof(float));
/* Create pseudo-random number generator */
curandCreateGenerator(&gen,
      CURAND_RNG_PSEUDO_DEFAULT);
/* Set seed */
curandSetPseudoRandomGeneratorSeed(gen, 1234ULL);
/* Generate n floats on device */
curandGenerateUniform(gen, devData, n);
/* Copy device memory to host */
cudaMemcpy(hostData, devData, n * sizeof(float),
      cudaMemcpyDeviceToHost);
/* Show result */
for(i = 0; i < n; i++) {
    printf("%1.4f ", hostData[i]);
}
printf("\n");
/* Cleanup */
curandDestroyGenerator(gen);
cudaFree(devData);
free(hostData);
```

CURAND Device API consists of functions with __device__ qualifier declared in curand_kernel.h. Generator states have to be initialized properly and are stored in curandState* and curandStateSobol32* data types. Example of using Device API to calculate what proportion of pseudo-random ints have low bit set:

```
__global__ void setup_kernel(curandState *state)
{
      int id = threadIdx.x + blockIdx.x * 64;
      /* Each thread gets same seed, a different sequence  number, no offset
*/
```

```
      curand_init(1234, id, 0, &state[id]);
}
__global__ void generate_kernel(curandState *state, int *result)
{
      int id = threadIdx.x + blockIdx.x * 64;
      int count = 0;
      unsigned int x;
      /* Copy state to local memory for efficiency */
      curandState localState = state[id];
/* Generate pseudo-random unsigned ints */
      for(int n = 0; n < 100000; n++) {
            x = curand(&localState);
            /* Check if low bit set */
            If(x & 1) { count++; }
      }
      /* Copy state back to global memory */
      state[id] = localState;
      /* Store results */
      result[id] += count;
}
```

FOR STUDENTS

Detailed information about CUBLAS is presented in [1], about CUFFT in [2], about CURAND in [3].

REFERENCES

1. NVIDIA CUBLAS Documentation [http://docs.nvidia.com/cublas/index.html#axzz3JRcPurfI]
2. NVIDIA CUFFT Documentation [http://docs.nvidia.com/cufft/index.html#axzz3JRcPurfI]
3. NVIDIA CURAND Documentation [http://docs.nvidia.com/curand/index.html#axzz3JRcPurfI]

INDIVIDUAL WORK

1. Find out which functionality does CUBLAS provide. Compare it with other BLAS implementations.
2. Find out which functionality does CUFFT provide. Compare it with other FFT implementations.
3. Find out which functionality does CURAND provide. Compare it with other PRNG implementations.