


# 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”

**Lobachevsky State University of Nizhni Novgorod**

***Faculty of Computational mathematics and cybernetics***

# *Introduction to GPU programming*

## 08 Lecture
## CUDA Libraries

Bastrakov S.I.
Software department

# Contents

- CUBLAS
- CUFFT
- CURAND

# CUBLAS

# BLAS

- BLAS: Basic Linear Algebra Subprograms.
- Base for LAPACK.
- Created in 1979.
- Many optimized implementations on Fortran and C:
  - Goto BLAS
  - BLAS in Intel MKL
  - BLAS in ACML
  - CUBLAS in CUDA Toolkit
  - …

# Naming convention

- All BLAS procedure names have structure:
  <character code><name><mode>(…)

# Using CUBLAS

- General concept: calling routines (not kernels) from **cublas.h** from host side, link with **cublas.lib**.
- Data is transferred using special functions.
- **cublasInit()**
- **cublasShutdown()**
- **cublasAlloc**(int n, int elemSize, void **devicePtr)
- **cublasFree**(const void *devicePtr)

# Using CUBLAS

- **cublasSetVector** (int n, int elemSize, const void *x, int incx, void *y, int incy)
- **cublasGetVector** (int n, int elemSize, const void *x, int incx, void *y, int incy)
- **cublasSetMatrix** (int rows, int cols, int elemSize, const void *A, int lda, void *B, int ldb)
- **cublasGetMatrix** (int rows, int cols, int elemSize, const void *A, int lda, void *B, int ldb)

# Some CUBLAS routines

- void **cublasScopy** (int n, const float *x, int incx, float *y, int incy)
- float **cublasSdot** (int n, const float *x, int incx, const float *y, int incy)
- void **cublasSaxpy** (int n, float alpha, const float *x, int incx, float *y, int incy)
- void **cublasSgemv** (char trans, int m, int n, float alpha, const float *A, int lda, const float *x, int incx, float beta, float *y, int incy)

# CUFFT

# FFT

- FFT – fast Fourier transform.
- Many optimized implementations:
  - FFTW ;
  - FFT in Intel MKL;
  - CUFFT in CUDA Toolkit
  - …

# Functionality of CUFFT

- **1D**, **2D**, **3D**.
- **In-place**, **out-of-place**.

# Using CUFFT

- Include **cufft.h**.
- Link with **cufft.lib**.
- Data and computing on GPU.
- Interface very similar to FFTW.

# Data and transform types

- Data types:
  - **cufftReal** = float (**R**);
  - **cufftDoubleReal** = double (**D**);
  - **cufftComplex** = float2 (**C**);
  - **cufftDoubleComplex** = double2 (**Z**).
- Transform types: **CUFFT_R2C**, **CUFFT_C2R**, **CUFFT_C2C**, **CUFFT_D2Z**, **CUFFT_Z2D**, **CUFFT_Z2Z**.
- Transform directions: **CUFFT_FORWARD** и **CUFFT_INVERSE**.

# Plans

- Each transform is described by **plan**.
- Plan has cufftHandle type.
- **cufftPlan1d**(…), **cufftPlan2d**(…), **cufftPlan3d**(…), **cufftPlanMany**(…).
- cufftResult **cufftPlan1d**(cufftHandle* plan, int nx, cufftType type, int batch).
- **cufftDestroy**(cufftHandle plan).

# Executing transform

- **cufftExecC2C**(…), **cufftExecR2C**(…), **cufftExecC2R**(…), **cufftExecZ2Z**(…), **cufftExecD2Z**(…), **cufftExecZ2D**(…).
- cufftResult **cufftExecC2C**(cufftHandle plan, cufftComplex* idata, cufftComplex* odata, int direction)

# Example…

- Forward and inverse transform for single-precision complex data
- BATCH=10, size NX=256
- In-place

# Example

```
#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

# CURAND

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

# Host API

- Include **curand.h**, link with **curand.lib**.
- **curandCreateGenerator/curandCreateGeneratorHost** (curandGenerator_t * generator, curandRngType_t rng_type)
- **curandSetPseudoRandomGeneratorSeed** (curandGenerator_t generator, unsigned long long seed).
- **curandDestroyGenerator** (curandGenerator_t generator)
- **curandGenerate** (curandGenerator_t generator, unsigned int * outputPtr, size_t num).
- **curandGenerateNormal** (curandGenerator_t generator, float * outputPtr, size_t n, float mean, float stddev).

# Host API

- **curandGenerateUniform** (curandGenerator_t generator, float ∗ outputPtr, size_t num).
- **curandGenerateUniformDouble** (curandGenerator_t generator, float ∗ outputPtr, size_t num).

# Example of using Host API…

```
/*
* This program uses the host CURAND API to generate100
* pseudorandom floats.
*/
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand.h>
```

# Example of using Host API…

```
int main(int argc, char *argv[])
{
    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));
```

# Example of using Host API…

```
/* 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);
```

# Example of using Host API

```
    /* Show result */
    for(i = 0; i < n; i++) {
        printf("%1.4f ", hostData[i]);
    }
    printf("\n");
    /* Cleanup */
    curandDestroyGenerator(gen);
    cudaFree(devData);
    free(hostData);
    return EXIT_SUCCESS;
}
```

# Device API

- Include **curand_kernel.h**.
- All functions have __device__ qualifier.
- Work is done using curandState* and curandStateSobol32*.
- Need to initialize properly.

# Device API

- __device__ void **curand_init** (unsigned long long seed, unsigned long long sequence, unsigned long long offset, curandState *state)
- __device__ unsigned int **curand** (curandState *state)
- __device__ float **curand_uniform** (curandState *state)
- __device__ void **curand_init** (unsigned int *direction_vectors, unsigned int offset, curandStateSobol32 *state).
- __device__ unsigned int **curand** (curandStateSobol32 *state) – аналогично с curand (curandState *state).
- __device__ float **curand_uniform** (curandStateSobol32 *state) – аналогично с curand_uniform (curandState *state).

# Example of using Device API…

```
/*
 * This program uses the device CURAND API to calculate
 * what proportion of pseudo-random ints have low bit set.
 */
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand_kernel.h>
```

# Example of using Device API…

```
__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]);
}
```

# Example of using Device API…

```
__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];
```

# Example of using Device API…

```
/* 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;
}
```

# Example of using Device API…

```
int main(int argc, char *argv[])
{
    int i, total;
    curandState *devStates;
    int *devResults, *hostResults;
    /* Allocate space for results on host */
    hostResults = (int *)calloc(64 * 64, sizeof(int));
    /* Allocate space for results on device */
    cudaMalloc((void **)&devResults, 64 * 64 * sizeof(int));
    /* Set results to 0 */
    cudaMemset(devResults, 0, 64 * 64 * sizeof(int));
```

# Example of using Device API…

```
/* Allocate space for prng states on device */
cudaMalloc((void **)&devStates, 64 * 64 *
        sizeof(curandState));
/* Setup prng states */
setup_kernel<<<64, 64>>>(devStates);
/* Generate and use pseudo-random */
for(i = 0; i < 10; i++) {
        generate_kernel<<<64, 64>>>(devStates,
        devResults);
}
```

# Example of using Device API…

```
/* Copy device memory to host */
    cudaMemcpy(hostResults, devResults, 64 * 64 *
            sizeof(int), cudaMemcpyDeviceToHost);
    /* Show result */
    total = 0;
    for(i = 0; i < 64 * 64; i++) {
            total += hostResults[i];
    }
    printf("Fraction with low bit set was %10.13f\n",
    (float)total / (64.0f * 64.0f * 100000.0f * 10.0f));
```

# Example of using Device API…

```
    /* Cleanup */
    cudaFree(devStates);
    cudaFree(devResults);
    free(hostResults);
    return EXIT_SUCCESS;
}
```

# References


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


# Authors


- Bastrakov S.I.,
  Assistant of the Software department of CMC faculty.
  bastrakov@vmk.unn.ru