

# Оптимизированные библиотеки CUDA

Романенко А.А.  
[arom@ccfit.nsu.ru](mailto:arom@ccfit.nsu.ru)

# Библиотеки

- cuBLAS — Базовые функции линейной алгебры
- cuSPARSE - BLAS для разреженных векторов и матриц
- cuFFT — преобразование Фурье
- cuRAND — генерация псевдо\квази-случайных чисел
- NPP – NVidia Performance primitives
- CUDA Video Decode — работа с потоковым видео
- Thrust – Шаблоны параллельных алгоритмов
- math.h - C99 floating-point Library

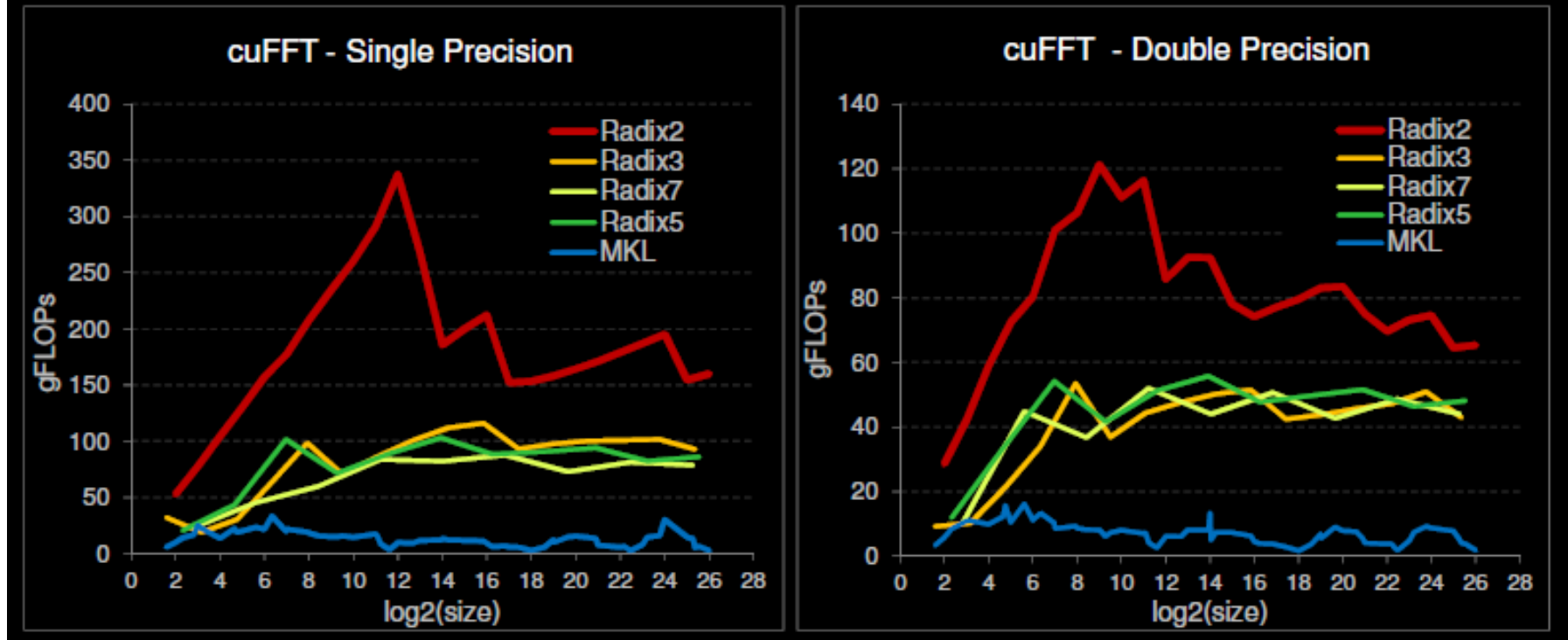
# Производительность



<http://developer.nvidia.com/content/cuda-40-math-libraries-performance-boost>

# FFTs up to 10x Faster than MKL

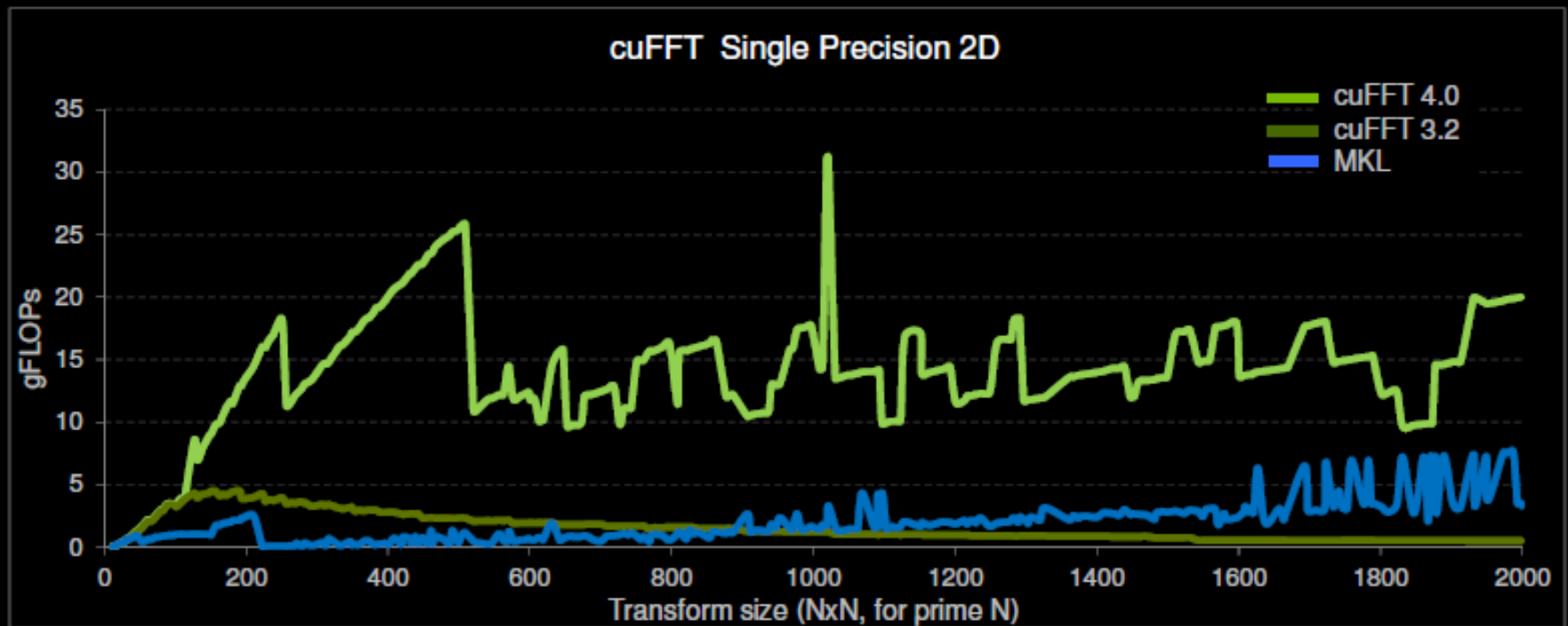
1D used in audio processing and as a foundation for 2D and 3D FFTs



- MKL 10.1r1 on Intel Quad Core i7-940 1333, 2.93Ghz
- cuFFT 4.0 on Tesla C2070, ECC on Performance measured for ~16M total elements, split into batches of transforms of the size on the x-axis

## 2D/3D primes now use Bluestein Algorithm

Significant performance improvement for 2D and 3D transform sizes



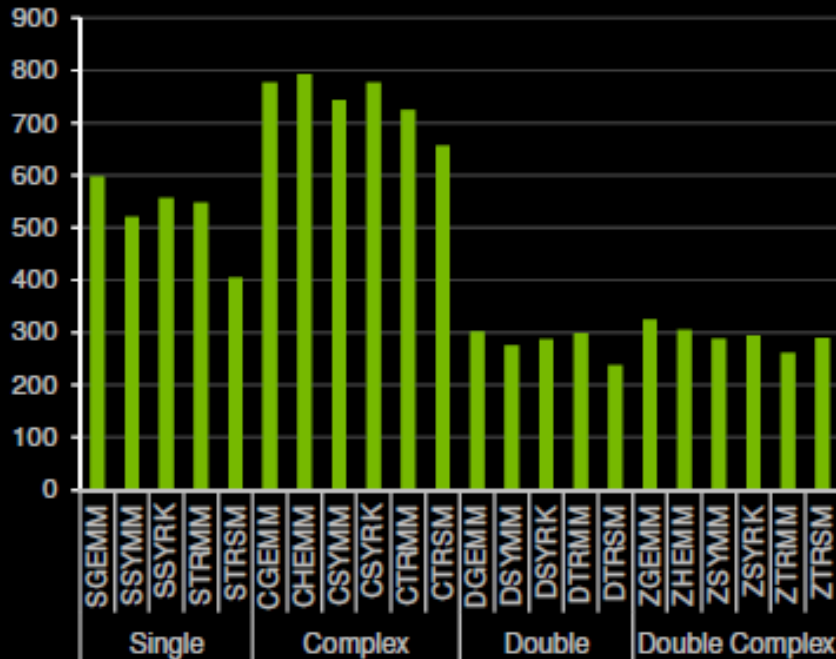
- MKL 10.1r1 on Intel Quad Core i7-940 1333, 2.93Ghz
- cuFFT4.0 on C2070, ECC on



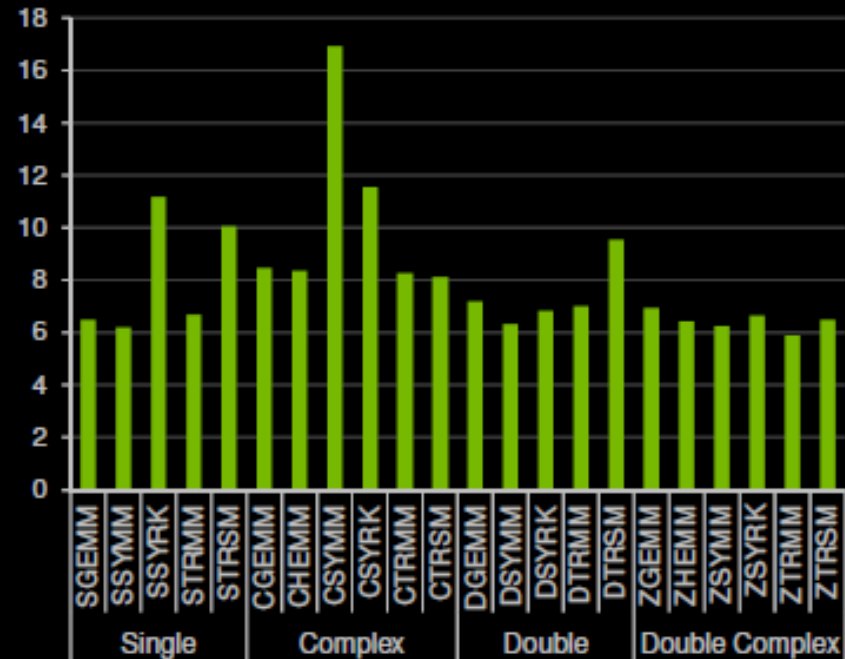
# cuBLAS Level 3 Performance

Up to ~800GFLOPS and ~17x speedup over MKL

GFLOPS



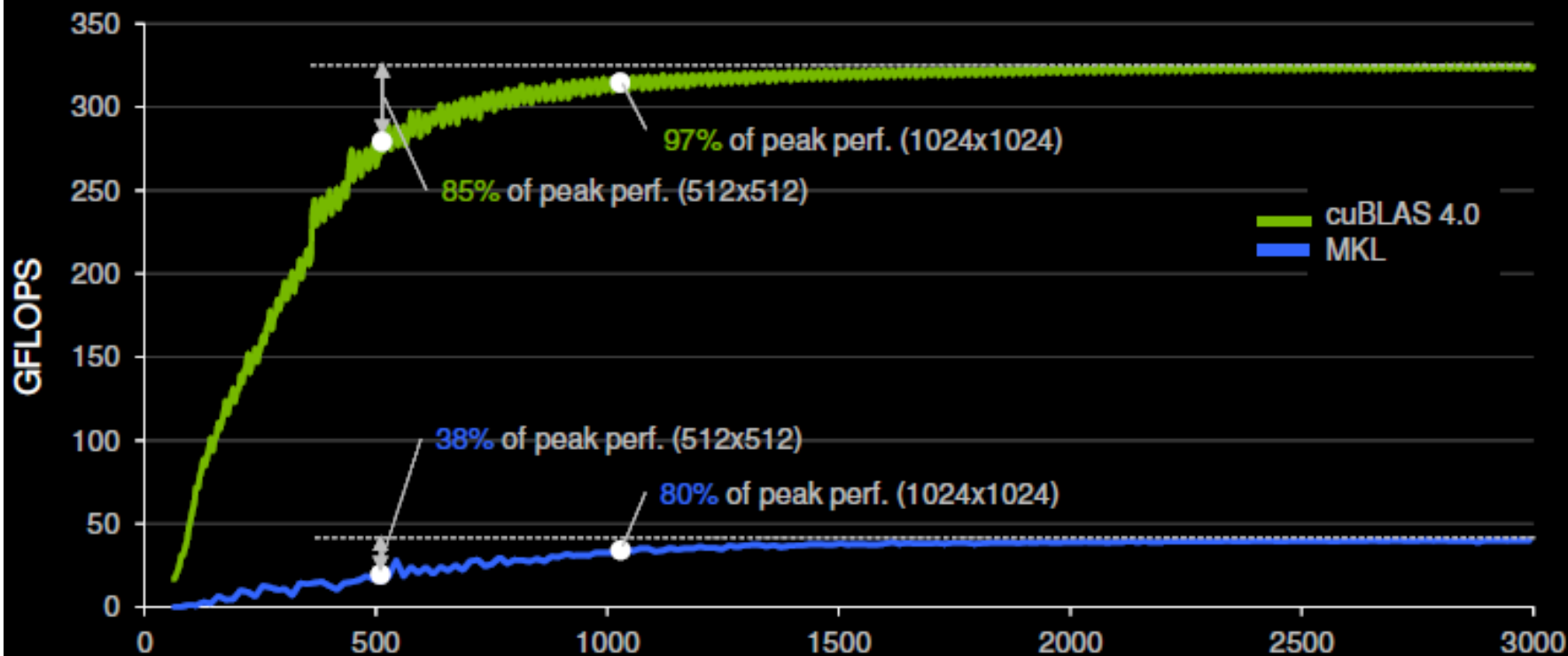
Speedup over MKL



- 4Kx4K matrix size
- cuBLAS 4.0, Tesla C2050 (Fermi), ECC on
- MKL 10.2.3, 4-core Corei7 @ 2.66Ghz

# ZGEMM Performance vs. Matrix Size

Up to **8x** speedup over MKL



- cuBLAS 4.0, Tesla C2050 (Fermi), ECC on
- MKL 10.2.3, 4-core Corei7 @ 2.66Ghz

# BLAS

- BLAS - Basic Linear Algebra Subprograms
- Стандарт API для библиотек, реализующих основные операции линейной алгебры
- <http://www.netlib.org/blas/>



# VLAS. Функциональность

- Уровень 1: Векторные операции
  - $y = ax + y$
  - $a = x'y$
- Уровень 2: Векторно-матричные операции
  - $y = aAx + by$
  - $A = axy' + A$
- Уровень 3: Матричные операции
  - $C = aAB + C$

# BLAS. Реализации

- refblas – C/Fortran77, netlib
- ATLAS – C/Fortran77, netlib
- uBLAS – C++, Boost
- **cuBLAS – C, NVIDIA**
- ACML – C/Fortran77, AMD
- MKL – C/Fortran77, Intel

# Именованние функций (1)

- `<character> <name> <mod> ( )`
- `character`
  - `s` - real, single precision (вещественные данные одинарной точности)
  - `c` - complex, single precision (комплексные данные одинарной точности)
  - `d` - real, double precision (вещественные данные двойной точности)
  - `z` - complex, double precision (комплексные данные двойной точности)

# Именованние функций (2)

- BLAS Level 1
  - ?dot - скалярное произведение векторов
  - ?rot — повернуть вектор
  - ?swap — обменять содержимое векторов
- <mod>
  - c - conjugated vector (сопряжённый вектор)
  - u - unconjugated vector (исходный (несопряжённый) вектор)
  - g - Givens rotation (вращение Гивенса)

# Именованние способов хранения матриц (3)

- BLAS level 2,3 <name>

- ge - general matrix (обычная матрица)
- gb - general band matrix (ленточная матрица)
- sy - symmetric matrix (симметричная матрица)
- sp - symmetric matrix (packed storage) (симметричная упакованная матрица)
- sb - symmetric band matrix (симметричная ленточная упакованная матрица)
- he - Hermitian matrix (эрмитова матрица)
- hp - Hermitian matrix (packed storage) (эрмитова упакованная матрица)
- hb - Hermitian band matrix (эрмитова ленточная матрица)
- tr - triangular matrix (треугольная матрица)
- tp - triangular matrix (packed storage) (треугольная упакованная матрица)
- tb - triangular band matrix (треугольная ленточная матрица)

# Именованные функции (4)

- BLAS level 2 <mod>
  - mv - matrix-vector product (матрично-векторное умножение)
  - sv - solving a system of linear equations with matrix-vector operations (решение системы линейных алгебраических уравнений с одной правой частью)
  - r - rank-1 update of a matrix (добавление матрицы «ранга 1»)
  - r2 - rank-2 update of a matrix (добавление двух матриц «ранга 1»)
- BLAS level 3 <mod>
  - mm - matrix-matrix product (произведение матриц)
  - sm - solving a system of linear equations with matrix-matrix operations (решение системы линейных алгебраических уравнений со многими правыми частями)
  - rk - rank-k update of a matrix (добавление матрицы «полного» ранга)
  - r2k - rank-2k update of a matrix (добавление двух матриц «полного» ранга)



# cuBLAS. Особенности

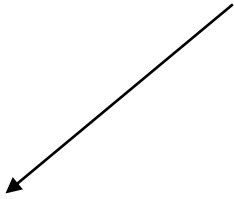
- Библиотека ориентирован на Фортран. Поэтому в целях совместимости с существующими версиями
  - в матрицах последовательно лежат элементы в столбцах (column-major storage format)
  - Нумерация элементов начинается с единицы.
  - Для C\C++ надо использовать макросы:
    - `#define IDX2F(i,j,ld) (((j)-1)*(ld)+(i)-1)`
    - `#define IDX2C(i,j,ld) ((j)*(ld)+(i))`
  - Проверка ошибок отдельными функциями.

# Сборка программы

- Файл заголовка — `cublas.h`
- Библиотеки
  - `cublas.so` (Linux),
  - `cublas.dll` (Windows),
  - `cublas.dylib` (Mac OS X)
- Makefile
  - `USECUBLAS :=1`

# Пример

- `#include "cublas.h"`  
`float* h_A; float* h_B; float* h_C;`  
`float* d_A = 0; float* d_B = 0; float* d_C = 0;`  
`int n2 = N * N;`  
`cublasStatus status;`  
  
`status = cublasInit(); // check status`  
`status = cublasAlloc(n2, sizeof(d_A[0]), (void**)&d_A);`  
`status = cublasAlloc(n2, sizeof(d_B[0]), (void**)&d_B);`  
`status = cublasAlloc(n2, sizeof(d_C[0]), (void**)&d_C);`  
`status = cublasSetVector(n2, sizeof(h_A[0]), h_A, 1, d_A, 1);`  
`status = cublasSetVector(n2, sizeof(h_B[0]), h_B, 1, d_B, 1);`  
`status = cublasSetVector(n2, sizeof(h_C[0]), h_C, 1, d_C, 1);`  
`cublasSgemm('n', 'n', N, N, N, alpha, d_A, N, d_B, N, beta, d_C, N);`  
`status = cublasGetError();`  
`status = cublasGetVector(n2, sizeof(h_C[0]), d_C, 1, h_C, 1);`  
`status = cublasFree(d_A);`  
`status = cublasFree(d_B);`  
`status = cublasFree(d_C);`  
`status = cublasShutdown();`



Increment

# cuSPARSE

- Категории функций
  - Level 1 — операции между разреженными и «плотными» векторами
  - Level 2 — операции между разреженными матрицами и «плотными» векторами
  - Level 3 — операции над разреженными матрицами и множеством «плотных» векторов
  - Вспомогательные функции и функции преобразования типов

# Формат разреженных векторов

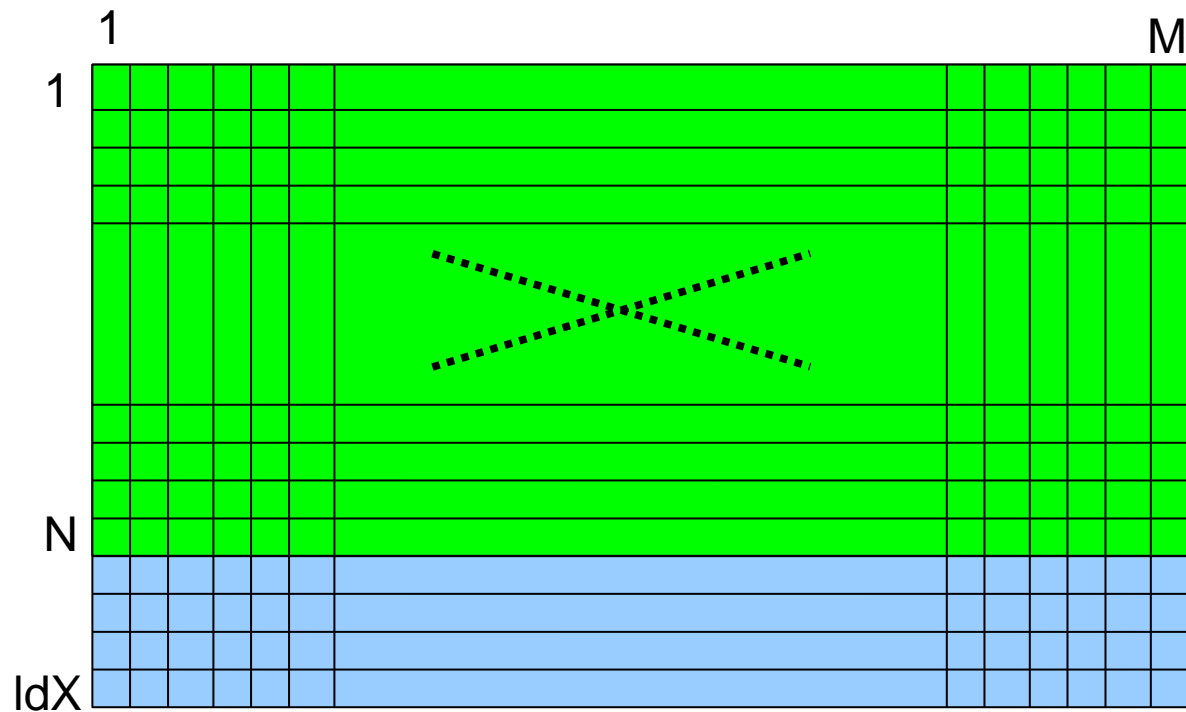
- Индексный формат
  - $V = [1.0, 0.0, 0.0, 2.0, 3.0, 0.0, 4.0]$   
 $Val = [1.0, 2.0, 3.0, 4.0]$   
 $Idx = [1, 4, 5, 7]$  // one-base index  
 $Idx = [0, 3, 4, 6]$  // zero-base index
  - Значение индекса может только увеличиваться
  - Индекс встречается только 1 раз

# Форматы разреженных матриц

- Плотный (полный) формат
- Координатный формат
- Сжатый по строкам
- Сжатый по колонкам



# Полный (плотный) формат



- M — количество столбцов
- N — количество рядов в матрице
- IdX — количество рядов в полной матрице.  $IdX \geq N$
- V — элементы матрицы

# Координатный формат

- Матрица  $M \times N$
- $nnz$  — количество ненулевых элементов
- $cooValA$  — массив значений ненулевых элементов матрицы в порядке приоритета по рядам
- $cooRowIndA$  — массив индексов рядов соответствующих ненулевым элементам матрицы
- $cooColIndA$  — массив индексов столбцов соответствующих ненулевым элементам матрицы

# Координатный формат (пример)

$$A = \begin{pmatrix} 1.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 3.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 7.0 & 8.0 \\ 0.0 & 0.0 & 9.0 & 0.0 & 6.0 \end{pmatrix}$$

`cooValA` = [1.0, 4.0, 2.0, 3.0, 5.0, 7.0, 8.0, 9.0, 6.0]

`cooRowIndxA` = [ 0, 0, 1, 1, 2, 2, 2, 3, 3]

`CooColIndxA` = [ 0, 1, 1, 2, 0, 3, 4, 2, 4]

`cooValA` = [1.0, 4.0, 2.0, 3.0, 5.0, 7.0, 8.0, 9.0, 6.0]

`cooRowIndxA` = [ 1, 1, 2, 2, 3, 3, 3, 4, 4]

`CooColIndxA` = [ 1, 2, 2, 3, 1, 4, 5, 3, 5]

# Сжатый по строкам

- Аналогичен координатному формату, но массив индексов строк сжимается.
- $nnz$  — количество ненулевых элементов
- $csrValA$  — массив значений ненулевых элементов матрицы в порядке приоритета по рядам
- $csrRowPtrA$  — массив длины  $N+1$ . Первые  $N$  — индекс в  $csrValA$  первого элемента в  $i$ -той строке, Последний равен  $nnz+(0$  или  $1)$ .
- $csrColIndA$  — массив индексов столбцов соответствующих ненулевым элементам матрицы

# Формат сжатия по строкам (пример)

$$A = \begin{pmatrix} 1.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 3.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 7.0 & 8.0 \\ 0.0 & 0.0 & 9.0 & 0.0 & 6.0 \end{pmatrix}$$

`csrValA = [1.0 4.0 2.0 3.0 5.0 7.0 8.0 9.0 6.0]`

`csrRowPtrA = [0 2 4 7 9]`

`csrColIndA = [0 1 1 2 0 3 4 2 4]`

`csrValA = [1.0 4.0 2.0 3.0 5.0 7.0 8.0 9.0 6.0]`

`csrRowPtrA = [1 3 5 8 10]`

`csrColIndA = [1 2 2 3 1 4 5 3 5]`

# Сжатый по столбцам

- Аналогичен координатному формату, но массив индексов столбцов сжимается.
- `nnz` — количество ненулевых элементов
- `cscValA` — массив значений ненулевых элементов матрицы в порядке приоритета по столбцам
- `cscColPtrA` — массив длины  $M+1$ . Первые  $M$  — индекс в `csrValA` первого элемента в  $i$ -том столбце, Последний равен `nnz+(0 или 1)`.
- `cscRowIndA` — массив индексов строк соответствующих ненулевых элементов матрицы



# Формат сжатия по столбцам (пример)

$$A = \begin{pmatrix} 1.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 3.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 7.0 & 8.0 \\ 0.0 & 0.0 & 9.0 & 0.0 & 6.0 \end{pmatrix}$$

`cscValA = [1.0 4.0 2.0 3.0 5.0 7.0 8.0 9.0 6.0]`

`cscColPtrA = [0 2 4 6 7 9]`

`cscRowIndA = [0 2 0 1 1 3 2 2 3]`

`cscValA = [1.0 4.0 2.0 3.0 5.0 7.0 8.0 9.0 6.0]`

`cscColPtrA = [1 3 5 7 8 10]`

`cscRowIndA = [1 3 1 2 2 4 3 3 4]`

# Функции

- Level 1, 2, 3 — `cusparse{S,D,C,Z}<имя функции>`
- Вспомогательные функции
  - Инициализация библиотеки, установка типов и параметр матриц и пр.
- Конвертирование типов
  - Преобразование матриц из одного сжатого формата в другой. Например,  
`cusparse{S,D,C,Z}csc2dense`  
`cusparse{S,D,C,Z}csr2csc`

# cuFFT

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn} \quad k = 0, \dots, N - 1$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \quad n = 0, \dots, N - 1.$$

- Дискретное преобразование Фурье
  - Комплексные и вещественные числа
  - Одномерное, двумерное, трехмерное

# Типы данных

- “cufftHandle” - дескриптор\план
  - `typedef unsigned int cufftHandle;`
- cufftResult”
  - `typedef enum cufftResult_t  
cufftResult;`
- cufftReal”
- cufftDoubleReal”
- cufftComplex
- cufftDoubleComplex

# Константы

- Типы преобразования
  - CUFFT\_R2C
  - CUFFT\_C2R
  - CUFFT\_C2C
  - CUFFT\_Z2D
  - CUFFT\_D2Z
  - CUFFT\_Z2Z
- Направление преобразования
  - CUFFT\_FORWARD
  - CUFFT\_INVERSE

# Функции

- Создание плана вычислений
  - `cufftPlan1d`, `cufftPlan2d`, `cufftPlan3d`
- Освобождение плана
  - `cufftDestroy`
- Выполнение
  - `cufftExecC2C`, `cufftExecR2C`, `cufftExecC2R`,  
`cufftExecZ2D`, `cufftExecD2Z`, `cufftExecZ2Z`

# Пример

```
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata, *odata;

cudaMalloc((void**) &idata, sizeof(cufftComplex) * NX * NY);
cudaMalloc((void**) &odata, sizeof(cufftComplex) * NX * NY);

/* Create a 2D FFT plan */
cufftPlan2d(&plan, NX, NY, CUFFT_C2C);

/* Transform the signal out of place */
cufftExecC2C(plan, idata, odata, CUFFT_FORWARD);

/* Inverse transform the signal in place */
cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);

/* Destroy the CUFFT plan */
cufftDestroy(plan);

cudaFree(idata); cudaFree(odata);
```



# Сборка программ

- Исходный код
  - `#include <cuFFT.h>`
- Makefile
  - `USECUFFT := 1`
- Библиотеки
  - `libcufft.a`

# cuRAND

- Библиотека для генерации псевдослучайных и квазислучайных чисел.
- Работает как на стороне GPU так и на стороне центрального процессора
- При одних и тех же начальных условиях (тип генератора, seed и пр.) последовательности чисел будут одинаковые. Как на GPU так и на CPU.

# Последовательность работы

1. Создать генератор  
`curandCreateGenerator()`, `curandCreateGeneratorHost()`
2. Установить параметры
3. Выделить память под результат  
`cudaMalloc()`, `cudaMallocHost()`
4. Сгенерировать последовательность
5. Использовать результат
6. При необходимости вернуться к пункту 4
7. Освободить память
8. Удалить генератор  
`curandDestroyGenerator()`

# Параметры

- Тип

- CURAND\_RNG\_PSEUDO\_DEFAULT
- CURAND\_RNG\_PSEUDO\_XORWOW
- CURAND\_RNG\_QUASI\_DEFAULT
- CURAND\_RNG\_QUASI\_SOBOL32

- Опции

- **Seed** (инициализация, начальное значение)
- **Offset** (смещение от начала исходной последовательности)
- **Ordering** (расположение чисел в памяти)
  - CURAND\_ORDERING\_PSEUDO\_DEFAULT
  - CURAND\_ORDERING\_PSEUDO\_BEST
  - CURAND\_ORDERING\_QUASI\_DEFAULT

# Функции генерации

- **curandGenerate** — 32-bit unsigned int
- **curandGenerateUniform** — равномерное распределение float (0.0, 1.0]
- **curandGenerateNormal** — нормальное распределение с заданными средним значением и стандартным отклонением float
- **curandGenerateUniformDouble** - равномерное распределение double (0.0, 1.0]
- **curandGenerateNormalDouble**— нормальное распределение с заданными средним значением и стандартным отклонением double

# Пример

```
int main(int argc, char *argv[]){
    size_t n = 100;
    curandGenerator_t gen;
    float *devData, *hostData;
    hostData = (float *)calloc(n, sizeof(float));
    cudaMalloc((void **)&devData, n * sizeof(float));
    /* Create pseudo-random number generator */
    curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_DEFAULT);
    curandSetPseudoRandomGeneratorSeed(gen, 1234ULL); /* Set seed */

    curandGenerateUniform(gen, devData, n);
    cudaMemcpy(hostData, devData, n * sizeof(float),
                cudaMemcpyDeviceToHost);

    for(int i = 0; i < n; i++) { printf("%1.4f ", hostData[i]); }
    printf("\n");

    curandDestroyGenerator(gen); /* Cleanup */
    cudaFree(devData);
    free(hostData);
    return EXIT_SUCCESS;
}
```

# Device API

- `__device__ void curand_init(...)`
  - Требуется много ресурсов. Рекомендуется проводить инициализацию отдельным ядром
- `__device__ unsigned int curand(curandState *state)`
- `__device__ float curand_uniform(curandState *state)`
- `__device__ float curand_normal (curandState *state)`
- `__device__ double curand_uniform_double (...)`
- `__device__ double curand_normal_double(...)`
- `__device__ float2 curand_normal2(...)`
- `__device__ double2 curand_normal2_double(...)`



# Пример

```
__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(curandState *state, int *result) {
    int id = threadIdx.x + blockDim.x * blockIdx.x;
    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);
        ...
    }
    /* Copy state back to global memory */
    state[id] = localState;
}
```

# Сборка программ

- Исходный код
  - `#include <curand.h> // host API`
  - `#include <curand_kernel.h> // device API`
- Makefile
  - `USECURAND := 1`
- Библиотеки
  - `libcurand.a`

# NPP

- Аналог IPP (Intel Performance primitives)
- Арифметические, логические, преобразования, фильтрации, статистические и пр.
  - ~420 функций обработки изображений (+70 в 4.0)
  - ~500 функций обработки сигналов (+400 в 4.0)

# CUDA Video Decode

- Декодирование видеопотоков на GPU в видеопамять
- Пост-обработка несжатого видео на CUDA
- Форматы
  - MPEG-2, VC-1, H.264 (AVCHD)
- Состав библиотеки
  - cuviddec.h
  - nvcuvid.h
  - nvcuvid.lib
  - nvcuvid.dll (Windows)
  - libnvcuvid.so (Linux)

# ФУНКЦИИ

- **cuvidCreateDecoder**(CUvideodecoder \*phDecoder, CUVIDDECODECREATEINFO \*pdci);
- **cuvidDestroyDecoder**(CUvideodecoder hDecoder);
- **cuvidDecodePicture**(CUvideodecoder hDecoder, CUVIDPICPARAMS \*pPicParams);
- **cuvidMapVideoFrame**(CUvideodecoder hDecoder, int nPicIdx, CUdeviceptr \* pDevPtr, unsigned int \* pPitch, CUVIDPROCPARAMS \*pVPP);
- **cuvidUnmapVideoFrame**(CUvideodecoder hDecoder, CUdeviceptr DevPtr);

# Thrust

- Библиотека шаблонов C++ для CUDA
- Аналоги
  - C++ STL
  - Intel TBB (Thread building blocks)
- Появилась в CUDA 4.0
- Позволяет быстро создавать приложения и прототипы.
- Документация
  - <http://wiki.thrust.googlecode.com/hg/html/index.html>

# Thrust. Пример

```
# include <thrust/device_vector.h>
# include <thrust/transform.h>
# include <thrust/sequence.h>
# include <thrust/copy.h>
# include <thrust/fill.h>
# include <thrust/replace.h>
# include <thrust/functional.h>
# include <iostream>

int main ( void ){

    thrust::device_vector <int>X(10); // allocate three device_vectors with 10 elements
    thrust::device_vector <int>Y(10); thrust::device_vector <int>Z(10);

    thrust::sequence(X.begin(), X.end()); // initialize X to 0,1,2,3, ....

    // compute Y = -X
    thrust::transform(X.begin(), X.end(), Y.begin(), thrust::negate<int>());

    thrust::fill(Z.begin(), Z.end(), 2); // fill Z with twos

    // compute Y = X mod 2
    thrust::transform(X.begin(), X.end(), Z.begin(), Y.begin(), thrust::modulus<int>());

    thrust::replace(Y.begin(), Y.end(), 1, 10); // replace all the ones in Y with tens

    // print Y
    thrust::copy(Y.begin(), Y.end(), std::ostream_iterator<int>(std::cout, "\n"));

    return 0;
}
```





**Math.h**