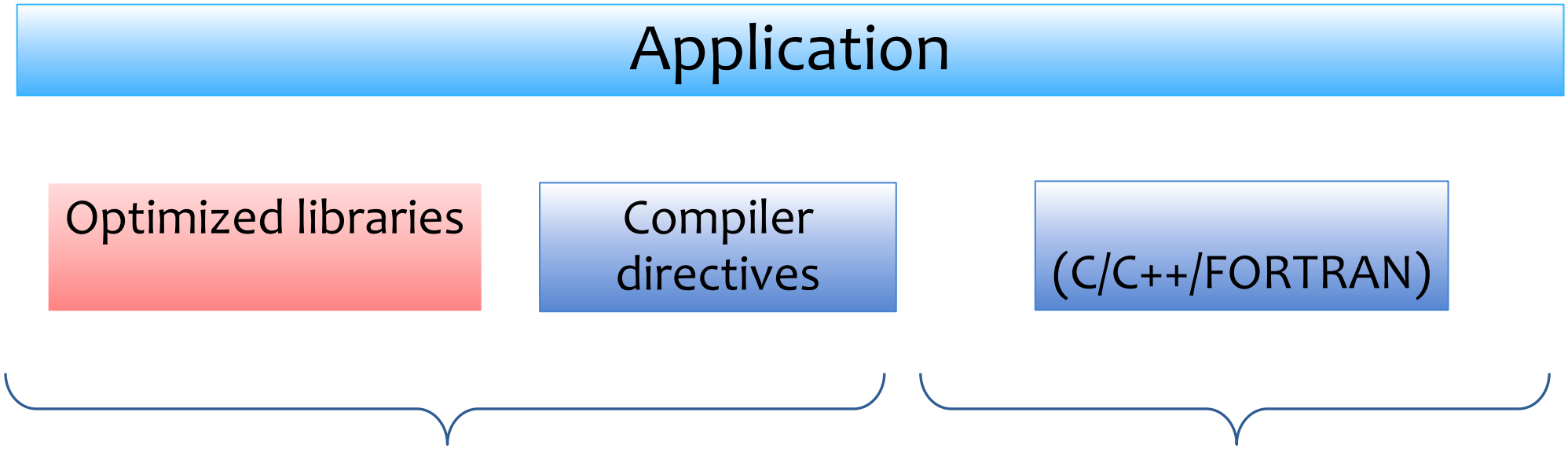
CUDA performance libraries

Alexey A. Romanenko
arom@ccfit.nsu.ru
Novosibirsk State University

3 approaches for launching program on GPU



Libraries

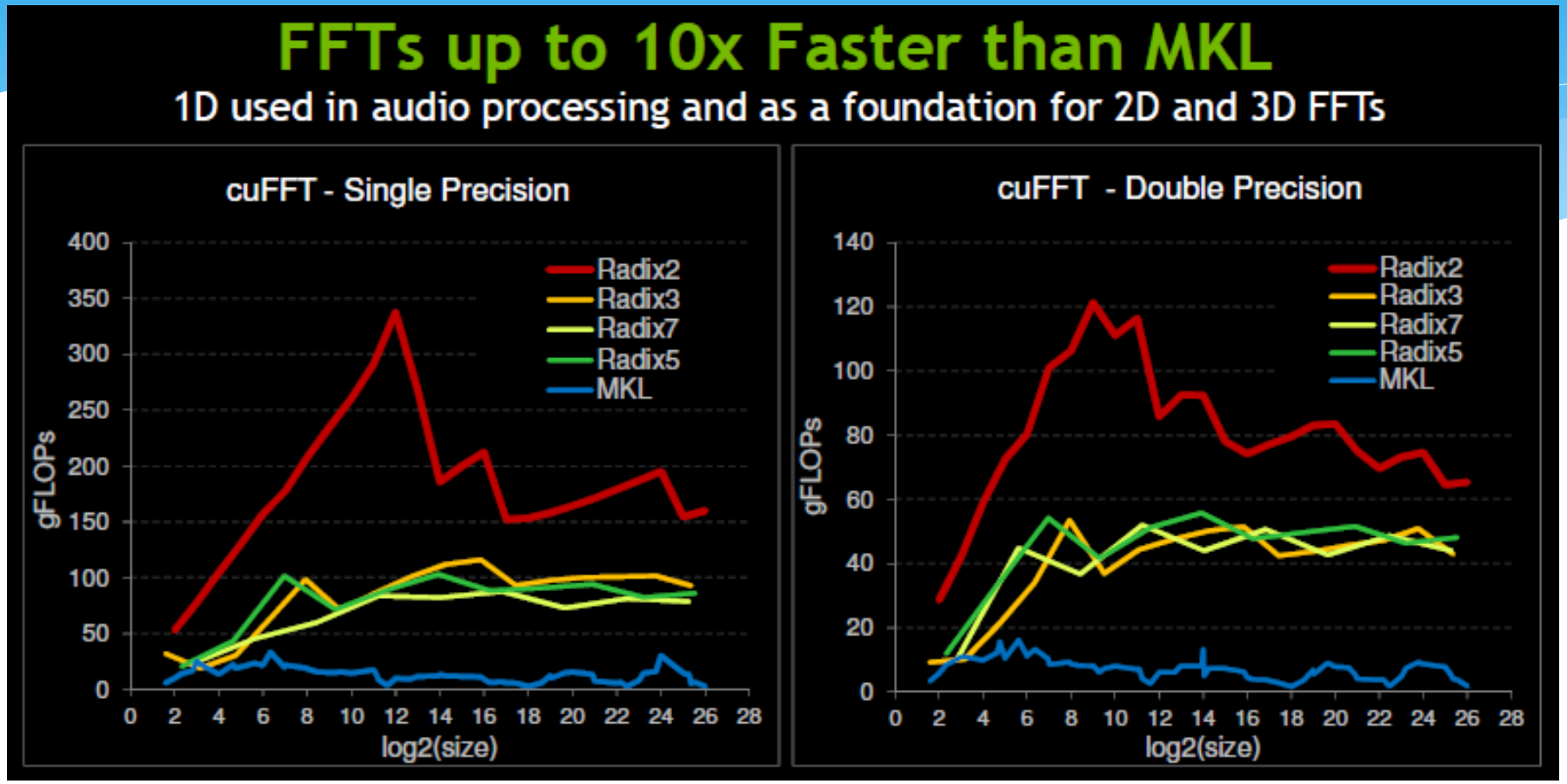
* cuBLAS — Basic Linear Algebra Subroutines
* cuSPARSE - Sparse BLAS
* cuFFT — Fast Fourie Transform
* cuRAND — Random number generators
* NPP – NVidia Performance primitives
* CUDA Video Decode — работа с потоковым видео
* Thrust – Templates for parallel algorithms
* math.h - C99 floating-point Library

Performance

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

FFTs up to 10x Faster than MKL

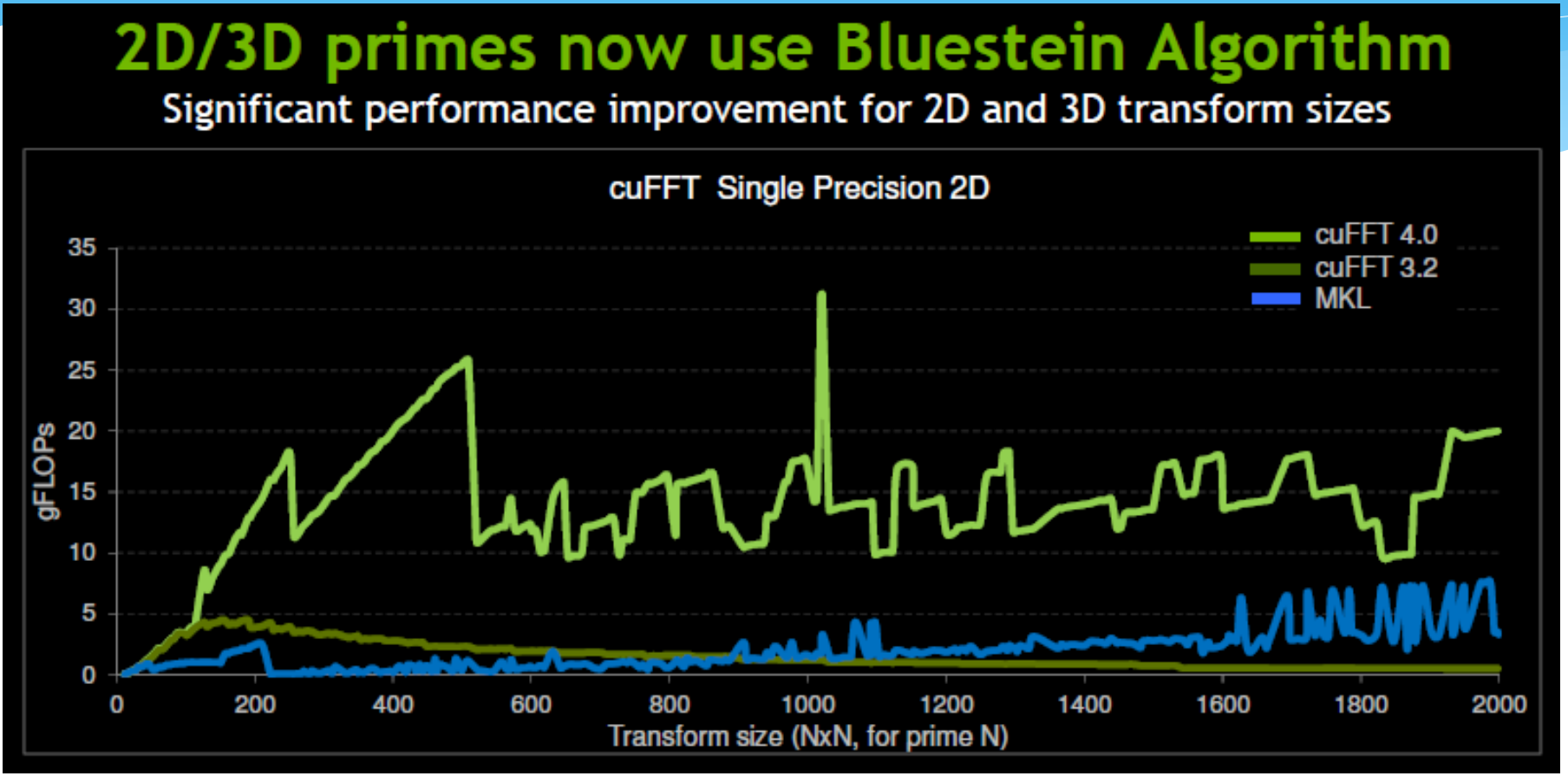
1D used in audio processing and as a foundation for 2D and 3D FFTs



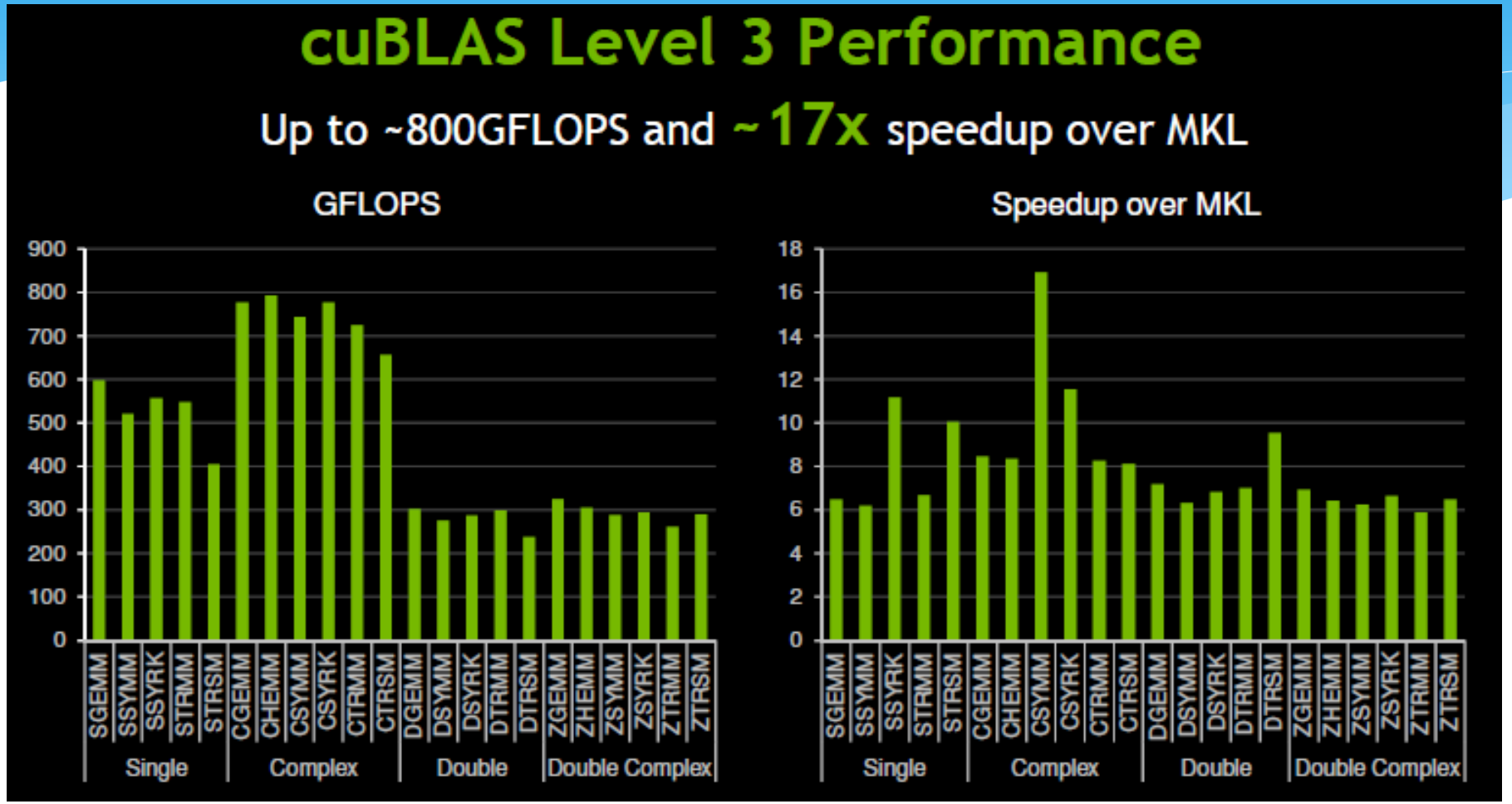
- MKL 10.1r1 on Intel Quad Core i7-940 1333, 2.93Ghz
- cuFFT 4.0 on Tesla C2070, ECC on
- Performance measured for ~16M total elements, split into batches of transforms of the size on the x-axis

2D/3D primes now use Bluestein Algorithm

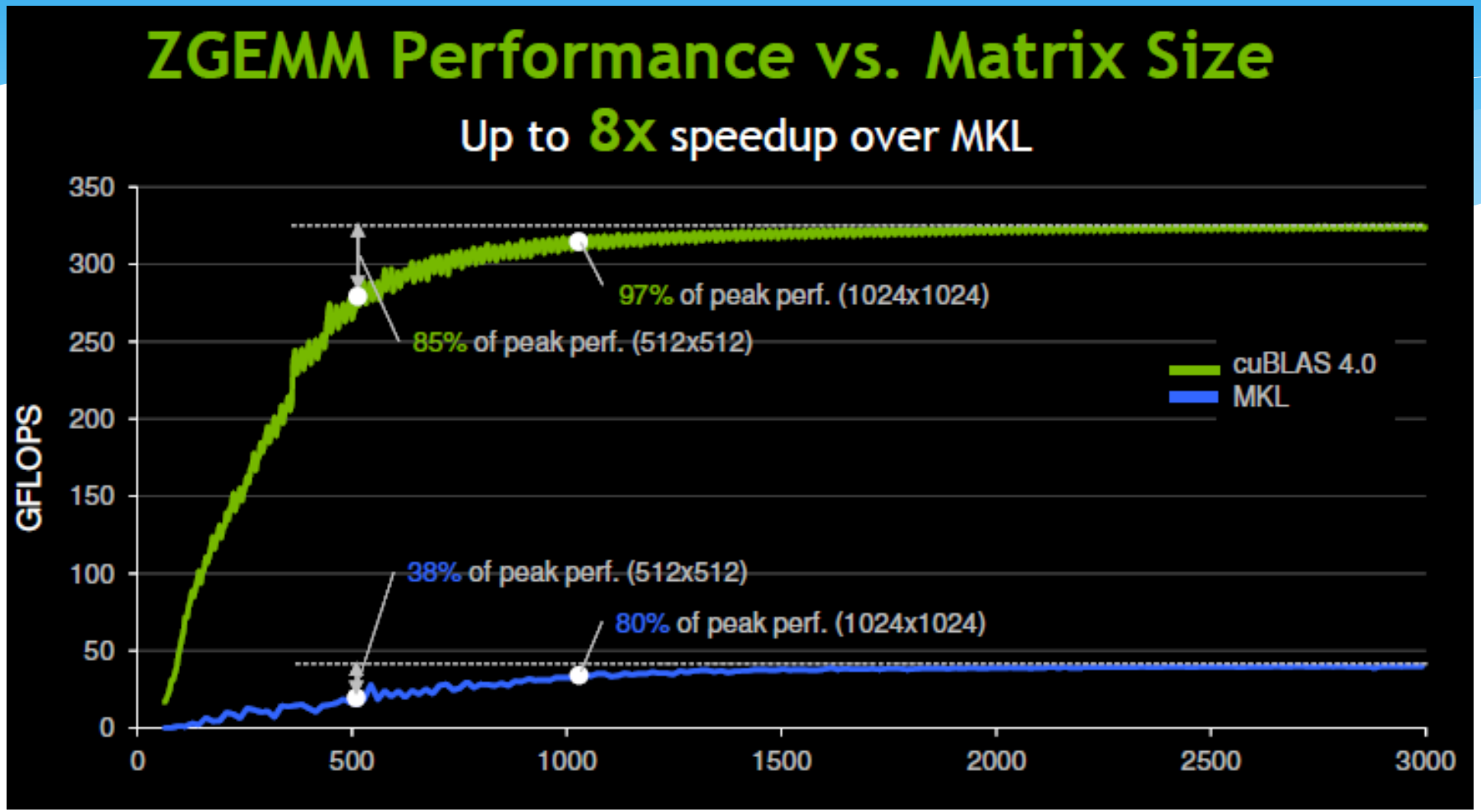
Significant performance improvement for 2D and 3D transform sizes



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



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



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

BLAS

* BLAS - Basic Linear Algebra Subprograms
* Standart. Introduced in the begining of 90-x
* http://www.netlib.org/blas/

BLAS. Functionality

* Level 1: vector operations
  * y = ax + y
  * a = x'y
* Level 2: vector-matrix operations
  * y = aAx + by
  * A = axy' + A
* Level 3: matrix operations
* C = aAB + C

BLAS. Implementations

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

Function naming (1)

* <character> <name> <mod> ( )
* character
  * s - real, single precision
  * c - complex, single precision
  * d - real, double precision
  * z - complex, double precision

Function naming (2)

* BLAS Level 1
  * ?dot - dot product
  * ?rot — rotation of the vector
  * ?swap — swap two vectors
* <mod>
  * c - conjugated vector
  * u - unconjugated vector
  * g - Givens rotation

Matrix storage format (3)

* BLAS level 2,3 <name>
  * ge - general matrix
  * gb - general band matrix
  * sy - symmetric matrix
  * sp - symmetric matrix (packed storage)
  * sb - symmetric band matrix
  * he - Hermitian matrix
  * hp - Hermitian matrix (packed storage)
  * hb - Hermitian band matrix
  * tr - triangular matrix
  * tp - triangular matrix (packed storage)
  * tb - triangular band matrix

Function naming (4)

* BLAS level 2 <mod>
  * mv - matrix-vector product
  * sv - solving a system of linear equations with matrix-vector operations
  * r - rank-1 update of a matrix
  * r2 - rank-2 update of a matrix
* BLAS level 3 <mod>
  * mm - matrix-matrix product
  * sm - solving a system of linear equations with matrix-matrix operations
  * rk - rank-k update of a matrix
  * r2k - rank-2k update of a matrix

cuBLAS. Features

- BLAS is originaly desighnied for Fortran.As the result:
  - Matrix stores its elements in column-major storage format
  - One-base indexing.
  - One should use macros for C\C++:
    - `#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))`
    - `#define IDX2C(i,j,ld) (((j)*(ld))+(i))`
  - Extra functions for handling errors.

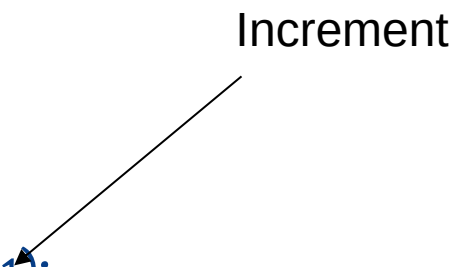
Compiling

* Header file — cublas.h
* Libraries
  * cublas.so (Linux),
  * cublas.dll (Windows),
  * cublas.dylib (Mac OS X)

Example

```
#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();
```



cuSPARSE

* Level 1 — operations over sparse and dense vectors
* Level 2 — operations over sparse matixes and dense vectors
* Level 3 — operations over sparse and dense matrixes

* Type conversion and auxiliary functions

Sparse vector format

- Index base format

  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

Sparse matrix formats

* dense
* COO
* CSR
* CSC
* ….

Functions

- Level 1, 2, 3 — cusparse{S,D,C,Z}<function name>
- Auxiliary functions
  - Init library, set types of vectors and matrixes, etc.
- Type conversions
  - Convert one sparse matrix format to another. For example, cusparse{S,D,C,Z}csc2dense cusparse{S,D,C,Z}csr2csc

cuFFT

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

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

- Discrete Fast Fourie Transform
  - Real\complex data
  - 1D, 2D, 3D

Data types

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

Some constants

- Conversion type
  - CUFFT_R2C
  - CUFFT_C2R
  - CUFFT_C2C
  - CUFFT_Z2D
  - CUFFT_D2Z
  - CUFFT_Z2Z
- Direction of convertion
  - CUFFT_FORWARD
  - CUFFT_INVERSE

Some functions

- Create computational plan
  - cufftPlan1d, cufftPlan2d, cufftPlan3d
- Destroy the plan
  - cufftDestroy
- Perform FFT
  - cufftExecC2C, cufftExecR2C, cufftExecC2R, cufftExecZ2D, cufftExecD2Z, cufftExecZ2Z

Example

```
#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);
```

Building program

* Header file
  * #include <cufft.h>
* Library
  * libcufft.a

cuRAND. Scenario

1. Create generator's stream
   curandCreateGenerator(), curandCreateGeneratorHost()
2. Set parameters
3. Allocate memory for result
   cudaMalloc(), cudaMallocHost()
4. Generate sequence
5. Process the result
6. If necessary, goto item 4
7. Free memory
8. Destroy generator's stream
   curandDestroyGenerator()

Parameters

- Type
  - CURAND_RNG_PSEUDO_DEFAULT
  - CURAND_RNG_PSEUDO_XORWOW
  - CURAND_RNG_QUASI_DEFAULT
  - CURAND_RNG_QUASI_SOBOL32
- Options
  - **Seed**
  - **Offset**
  - **Ordering**
    - CURAND_ORDERING_PSEUDO_DEFAULT
    - CURAND_ORDERING_PSEUDO_BEST
    - CURAND_ORDERING_QUASI_DEFAULT

Example

```
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(...)
* __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(...)

Example

```
__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;
}
```

Building program

* Header files
  * #include <curand.h> // host API
  * #include <curand_kernel.h> // device API
* Library
  * libcurand.a

NPP

- Analog of IPP (Intel Performance primitives)
- Different types of functions (statistics, logical, arithmetical, etc.)
  - ~420 image processing functions (+70 в 4.0)
  - ~500 signal processing funcrions (+400 в 4.0)

CUDA Video Decode

* Decoding of video stream on GPU into video memory
* Post-processing of uncompressed video on CUDA
* formats
  * MPEG-2, VC-1, H.264 (AVCHD)
* Contents of the library
  * cuviddec.h
  * nvcuvid.h
  * nvcuvid.lib
  * nvcuvid.dll (Windows)
  * libnvcuvid.so (Linux)

Thrust

- C++ template library for CUDA
- Analogues
  - C++ STL
  - Intel TBB (Thread building blocks)
- Introduced in CUDA 4.0
- Thrust allows you to prototype your application easily.
- Documentation
  - http://wiki.thrust.googlecode.com/hg/html/index.html

Thrust. Example

```
# 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;
}
```