

An abstract, three-dimensional representation of the NVIDIA logo, composed of several curved, metallic-looking segments that overlap to form the iconic 'V' shape. The segments have a brushed metal texture and are set against a dark, textured background.

Fast GPU Development with CUDA Libraries

Jeff Larkin



3 Ways to Accelerate Applications



Applications

Libraries

“Drop-in”
Acceleration

OpenACC
Directives

Easily Accelerate
Applications

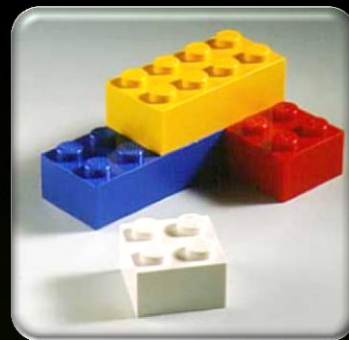
Programming
Languages

Maximum
Flexibility

NVIDIA CUDA Library Approach



- Provide basic building blocks
 - Make them easy to use
 - Make them fast
-
- Provides a quick path to GPU acceleration
 - Enables developers to focus on their “secret sauce”
 - Ideal for applications that use CPU libraries



CUDA Math Libraries



High performance math routines for your applications:

- **cuFFT – Fast Fourier Transforms Library**
- **cuBLAS – Complete BLAS Library**
- **cuSPARSE – Sparse Matrix Library**
- **cuRAND – Random Number Generation (RNG) Library**
- **NPP – Performance Primitives for Image & Video Processing**
- **Thrust – Templated C++ Parallel Algorithms & Data Structures**
- **math.h - C99 floating-point Library**

Included in the CUDA Toolkit

Free download @ www.nvidia.com/getcuda

Linear Algebra

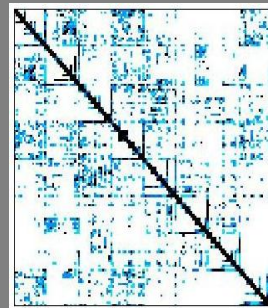


Dense



cuBLAS

Sparse



cuSPARSE

Drop-In Acceleration

```
int N = 1 << 20;
```

```
// Perform SAXPY on 1M elements: y[]=a*x[]+y[]  
saxpy(N, 2.0, d_x, 1, d_y, 1);
```


Drop-In Acceleration (Step 1)

```
int N = 1 << 20;
```

```
// Perform SAXPY on 1M elements: d_y[]=a*d_x[]+d_y[]  
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
```



Add “cublas” prefix and use
device variables

Drop-In Acceleration (Step 2)

```
int N = 1 << 20;  
cublasInit();
```



Initialize CUBLAS

```
// Perform SAXPY on 1M elements: d_y[]=a*d_x[]+d_y[]  
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
```

```
cublasShutdown();
```



Shut down CUBLAS

Drop-In Acceleration (Step 3)

```
int N = 1 << 20;  
cublasInit();  
cublasAlloc(N, sizeof(float), (void**)&d_x);  
cublasAlloc(N, sizeof(float), (void*)&d_y);
```



Allocate device vectors

```
// Perform SAXPY on 1M elements: d_y[]=a*d_x[]+d_y[]  
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
```

```
cublasFree(d_x);  
cublasFree(d_y);  
cublasShutdown();
```



Deallocate device vectors

Drop-In Acceleration (Step 4)

```
int N = 1 << 20;  
cublasInit();  
cublasAlloc(N, sizeof(float), (void**)&d_x);  
cublasAlloc(N, sizeof(float), (void**)&d_y);
```

```
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);  
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);
```



Transfer data to GPU

```
// Perform SAXPY on 1M elements: d_y[]=a*d_x[]+d_y[]  
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
```

```
cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);
```



Read data back GPU

```
cublasFree(d_x);  
cublasFree(d_y);  
cublasShutdown();
```

cuBLAS Interface



```
err = idamax(n, col, 1, size);
```

```
err = dscal(n, val, row, 1);
```

```
dgemm('N','N',m,n,k,A,m,  
      B,k,0.0,C,m)
```

```
err = cublasIdamax(hdl, n, col, 1, size);
```

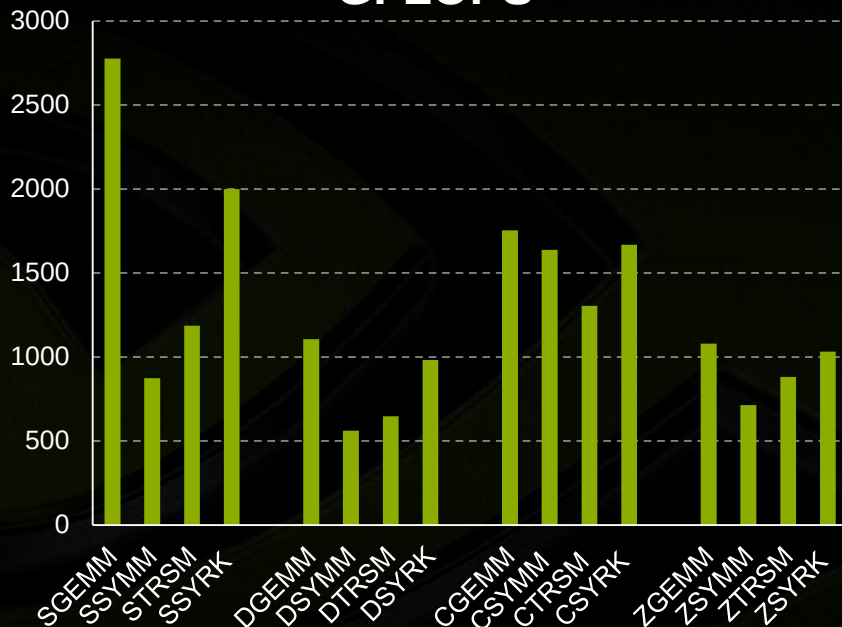
```
err = cublasDscal(hdl, n, val, row, 1);
```

```
cublasDgemm(hdl, 'N', 'N', m, n, k, d_A, m,  
            d_B, k, 0.0, d_C, m)
```

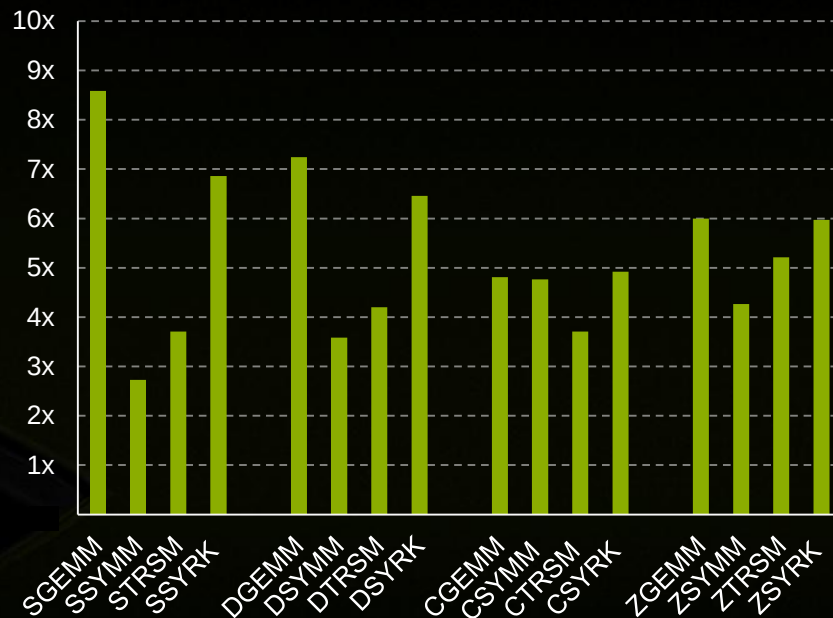
cuBLAS Level 3: >1 TFLOPS double-precision



GFLOPS



Speedup over MKL



- MKL 10.3.6 on Intel SandyBridge E5-2687W @3.10GHz
- CUBLAS 5.0.30 on K20X, input and output data on device

GPU-Callable Libraries



New in CUDA 5.0

Call cuBLAS library function from GPU code

Supported on K20 and K20x only

Encourages third party libraries

cuSPARSE Interface

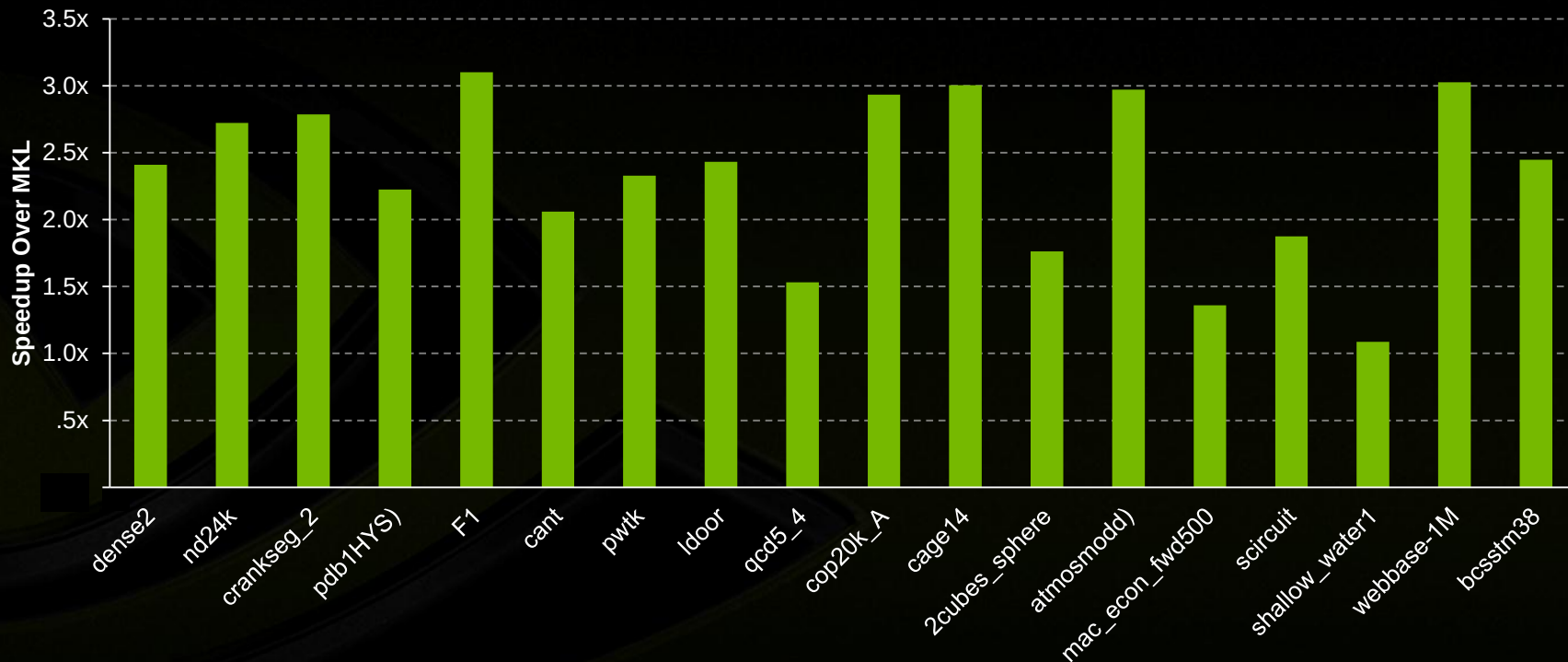


```
mkldcsrmmv(transa, m, k,  
            alpha, descr,  
            val, indx, pntbr, pntre,  
            x, beta, y);
```

```
err = cusparseDcsrmmv(hdl, transa, m, k,  
                      nnz, alpha, descr,  
                      val, indx, col,  
                      x, beta, y);
```

cuSPARSE performance

Sparse Matrix x Dense Vector

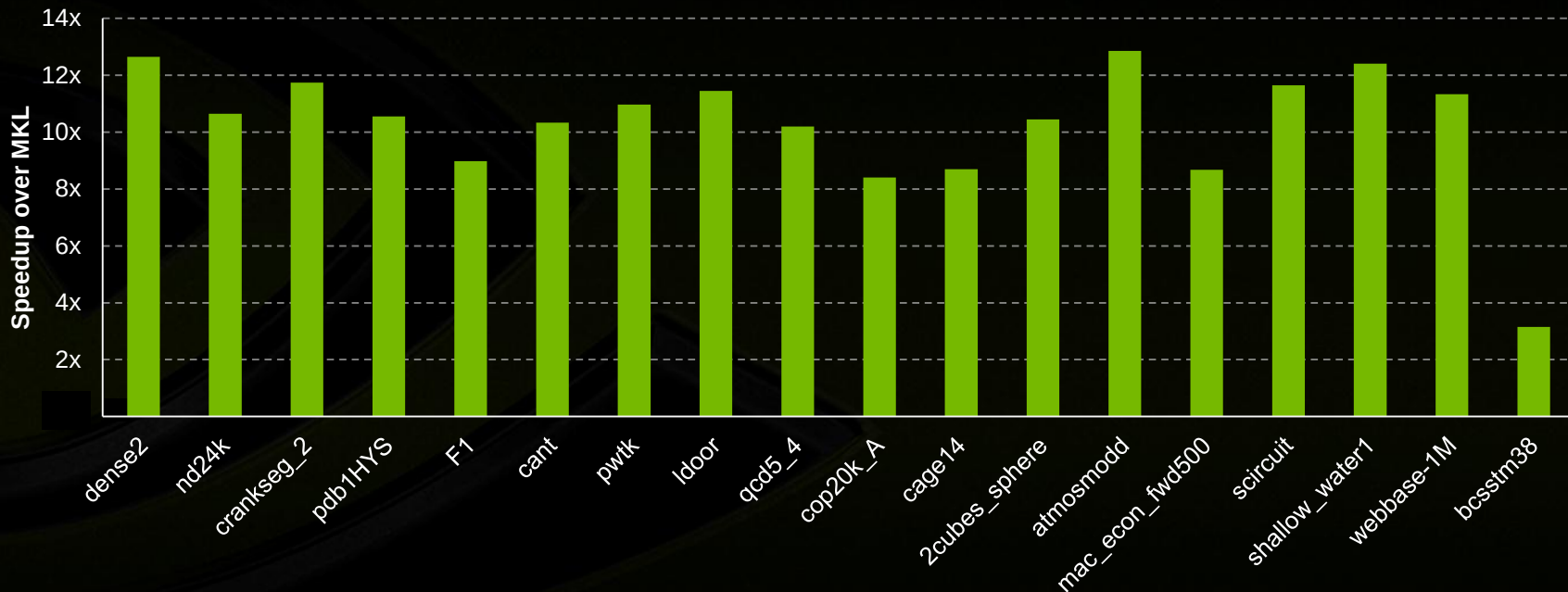


- Average of s/d/c/z routines
- cuSPARSE 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz

cuSPARSE: up to 12x Faster than MKL



Sparse Matrix x 6 Dense Vectors
(useful for block iterative solvers)

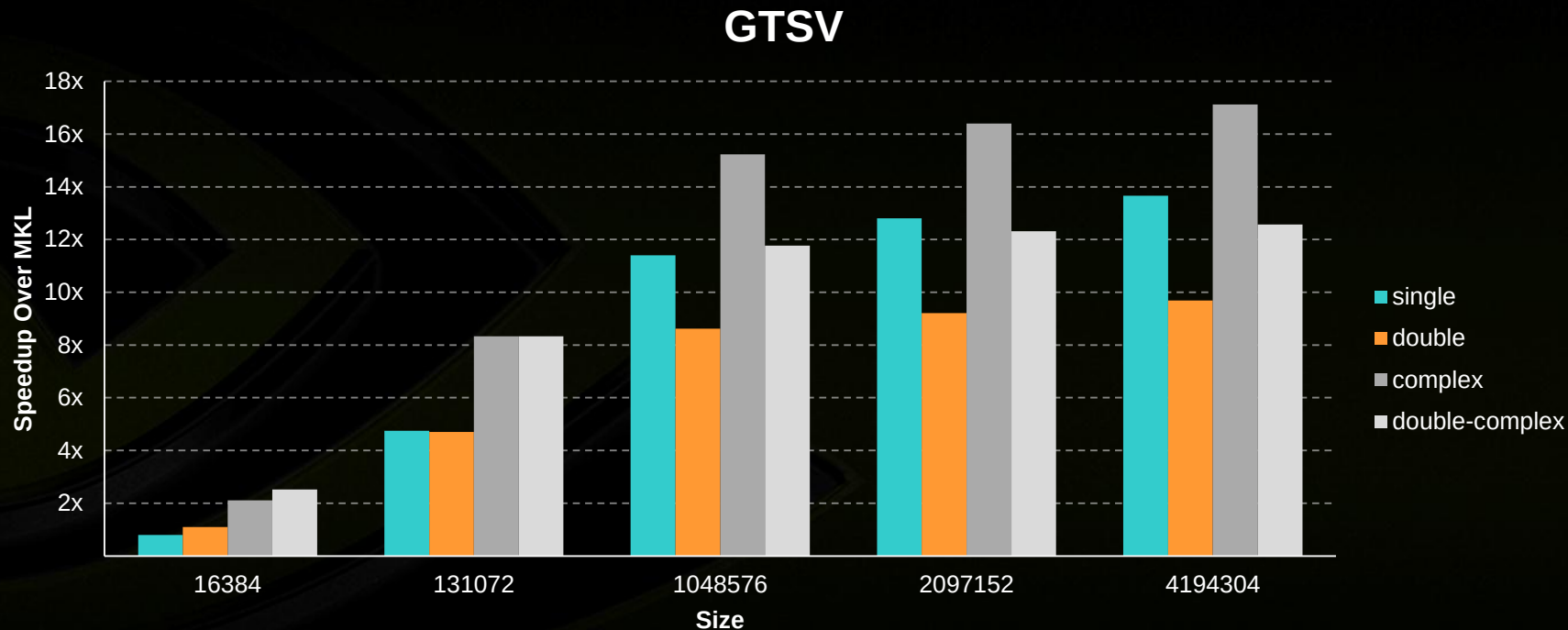


- Average of s/d/c/z routines
- cuSPARSE 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz

Tri-diagonal Solver Performance vs. MKL



Speedup for Tri-Diagonal solver (gtsv)



- cuSPARSE 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz

Different Approaches to Linear Algebra



- CULA tools (dense, sparse)
 - LAPACK based API
 - Solvers, Factorizations, Least Squares, SVD, Eigensolvers
 - Sparse: Krylov solvers, Preconditioners, support for various formats

```
culaSgetrf(M, N, A, LDA, IPIV, INFO)
```

- ArrayFire
 - Array container object
 - Solvers, Factorizations, SVD, Eigensolvers

```
array out = lu(A)
```



EM Photonics



AccelerEyes

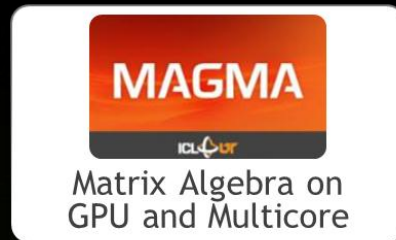
Different Approaches to Linear Algebra (cont.)



- **MAGMA**

- LAPACK conforming API
- Magma BLAS and LAPACK
- High performance by utilizing both GPU and CPU

`magma_sgetrf(M, N, A, LDA, IPIV, INFO)`



ICL

- **LibFlame**

- LAPACK compatibility interface
- Infrastructure for rapid linear algebra algorithm development

`FLASH_LU_piv(A, p)`



UT-Austin

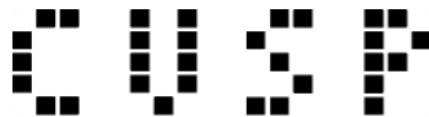
Different Approaches to Linear Algebra (cont.)



- CUSP

- Sparse matrix operations
- Open source
- Supports COO, CSR, ELL, DIA, hybrid, etc.
- Solvers, monitors, preconditioners, etc.

```
cusp::krylov::cg(A, x, b) ;
```



Sparse Linear
Algebra

Toolkits are increasingly supporting GPUs



- **PETSc**

- GPU support via extension to Vec and Mat classes
- Partially dependent on CUSP
- MPI parallel, GPU accelerated solvers

The PETSc logo, consisting of the text "PETSc" in a bold, red, serif font, centered within a light blue rounded rectangular box. The box has a subtle reflection effect below it.

PETSc

- **Trilinos**

- GPU support in KOKKOS package
- Used through vector class Tpetra
- MPI parallel, GPU accelerated solvers

The Trilinos logo, featuring the word "Trilinos" in a stylized, white, cursive font, centered within a blue rounded rectangular box. The box has a subtle reflection effect below it.

Trilinos

GPU Fast Fourier Transforms

cuFFT



- Interface modeled after FFTW

```
fftw_plan PlanA;  
  
fftw_plan_dft_2d(N, M, &PlanA,  
                data, data, FFT_FORWARD)  
  
fftw_execute_dft(PlanA, data,  
                data);
```

```
cufftPlan2d PlanA;  
  
cufftCreatePlan(N, M, &PlanA, CUFFT_C2C);  
  
cufftExecC2C(PlanA, d_data, d_data,  
             CUFFT_FORWARD);
```

- Supports streams and batching (2 and 3-D, too!) for better performance

CUFFT: up to 600 GFLOPS

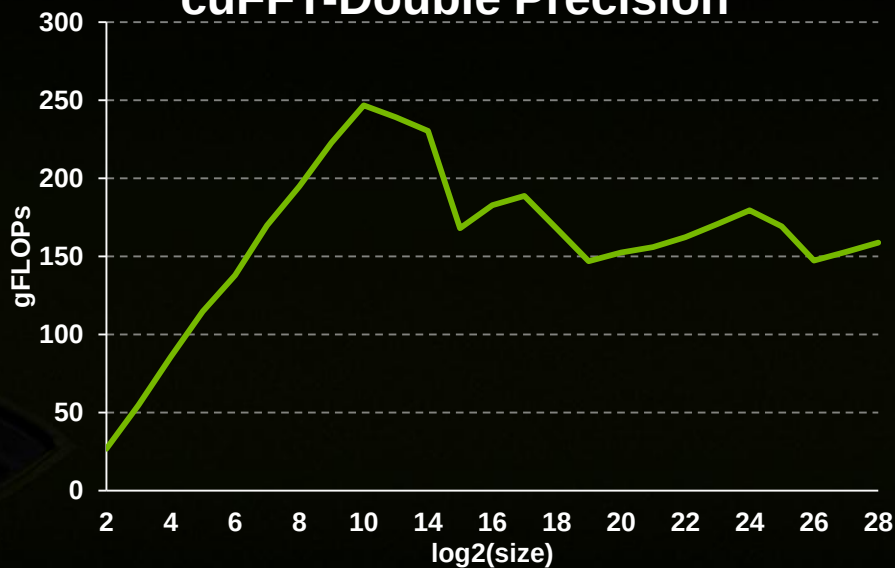


1D used in audio processing and as a foundation for 2D and 3D FFTs

cuFFT-Single Precision



cuFFT-Double Precision



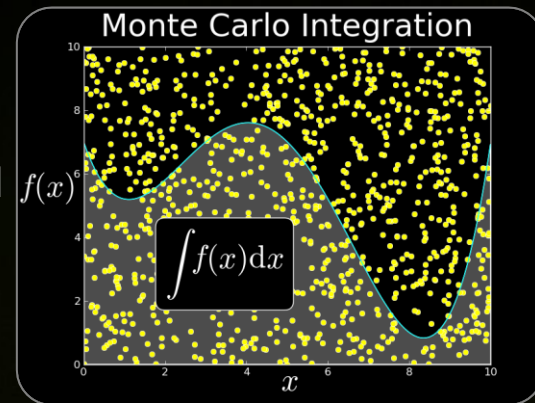
• CUFFT 5.0.30 on K20X, input and output data on device

cuRAND

Random Number Generation on GPU



- **Generating high quality random numbers in parallel is hard**
 - Don't do it yourself, use a library!
- **Large suite of generators and distributions**
 - XORWOW, MRG323ka, MTGP32, (scrambled) Sobol
 - uniform, normal, log-normal
 - Single and double precision
- **Two APIs for cuRAND**
 - Called from CPU: Ideal when generating large batches of RNGs on GPU
 - Called from GPU: Ideal when RNGs need to be generated inside a kernel





cuRAND: Host vs Device API

■ CPU API

```
#include "curand.h"

curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_DEFAULT);
curandGenerateUniform(gen, d_data, n);
```

Generate set of
random
numbers at once

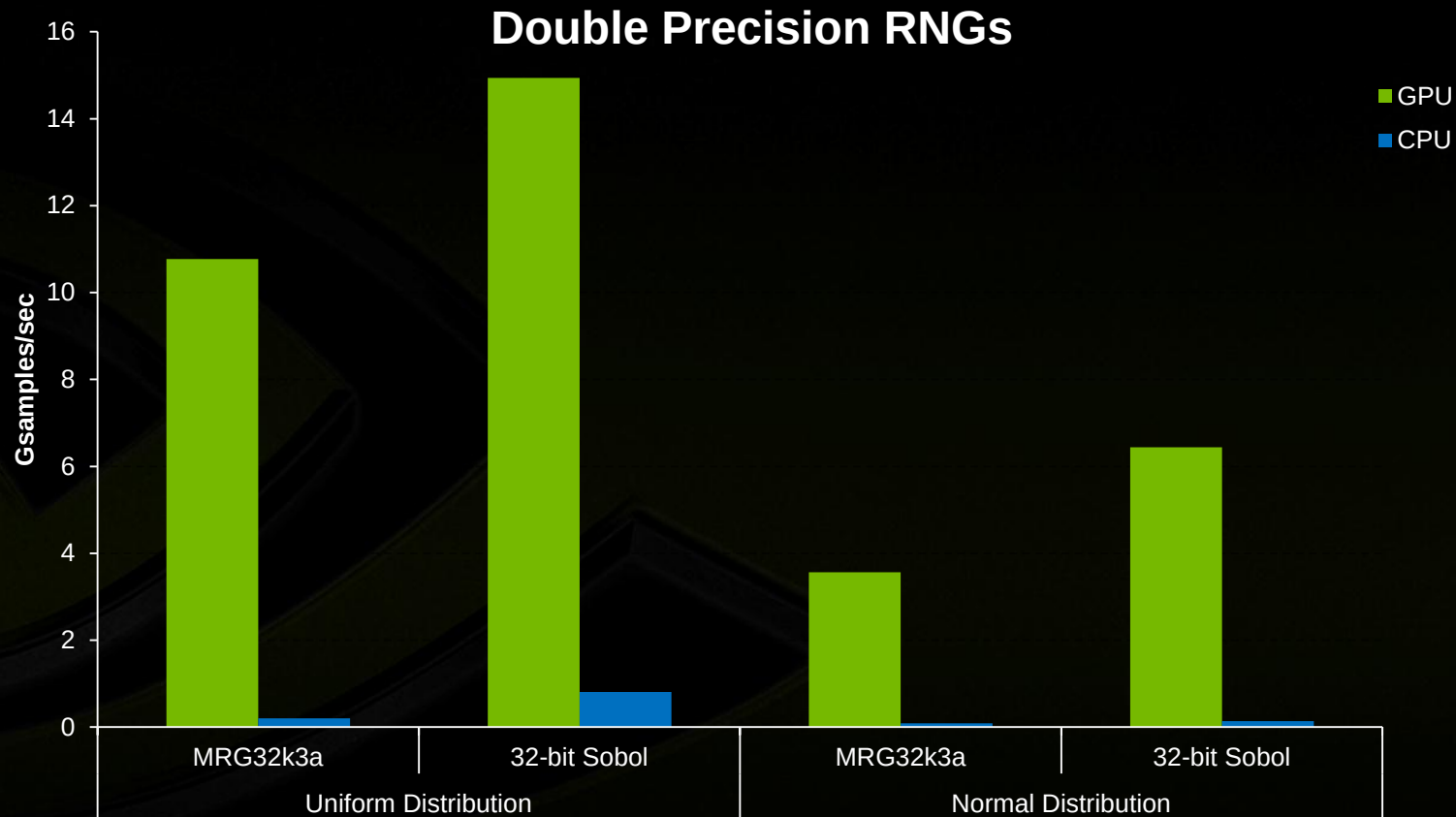
■ GPU API

```
#include "curand_kernel.h"

__global__ void generate_kernel(curandState *state) {
    int id = threadIdx.x + blockIdx.x * 64;
    x = curand(&state[id]);
}
```

Generate random
numbers per thread

cuRAND Performance



Thrust: STL-like CUDA Template Library



- **GPU(device) and CPU(host) vector class**

```
thrust::host_vector<float> H(10, 1.f);  
thrust::device_vector<float> D = H;
```

- **Iterators**

```
thrust::fill(D.begin(), D.begin()+5, 42.f);  
float* raw_ptr = thrust::raw_pointer_cast(D);
```

- **Algorithms**

- **Sort, reduce, transformation, scan, ..**

```
thrust::transform(D1.begin(), D1.end(), D2.begin(), D2.end(),  
thrust::plus<float>()); // D2 = D1 + D2
```

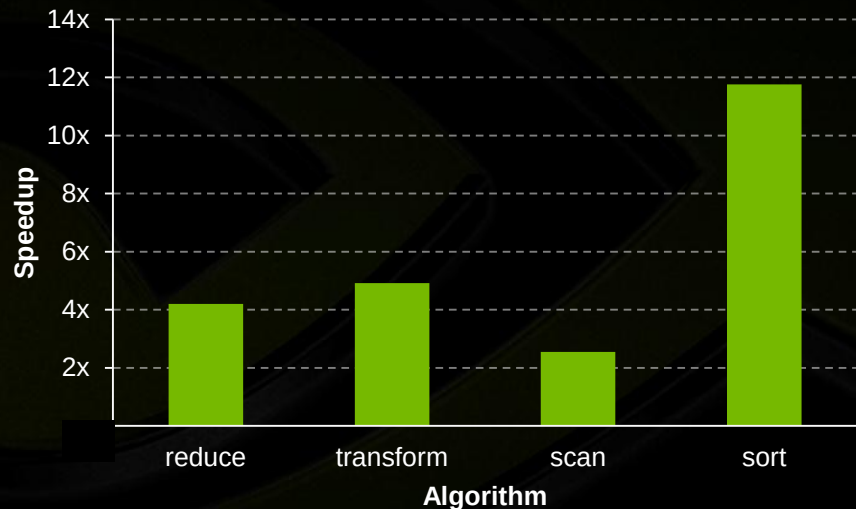


C++ STL Features
for CUDA

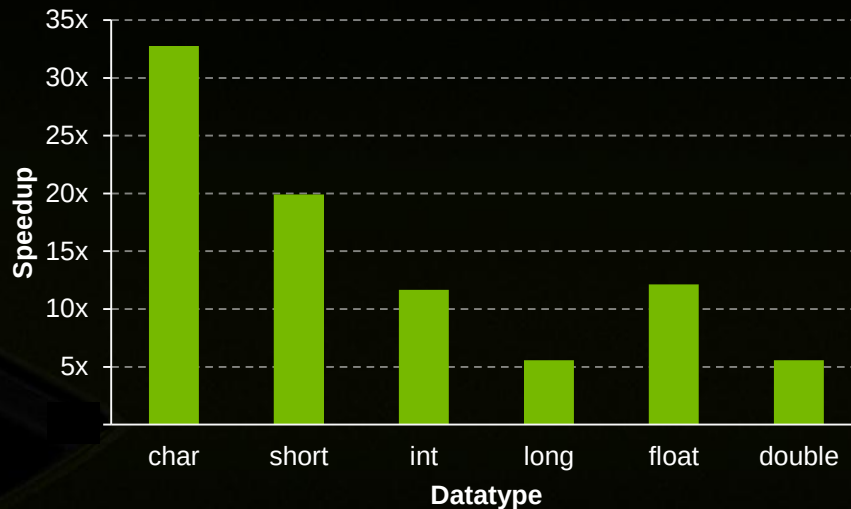
Thrust Performance



Various Algorithms (32M int.) Speedup over TBB



Sort (32M samples) Speedup over TBB



- Thrust 5.0 on K20X, input and output data on device
- TBB 4.1 on Intel SandyBridge E5-2687W @3.10GHz

math.h: C99 floating-point library + extras



CUDA math.h is **industry proven, high performance, accurate**

- **Basic**: +, *, /, 1/, sqrt, FMA (all IEEE-754 accurate for float, double, all rounding modes)
- **Exponentials**: exp, exp2, log, log2, log10, ...
- **Trigonometry**: sin, cos, tan, asin, acos, atan2, sinh, cosh, asinh, acosh, ...
- **Special functions**: lgamma, tgamma, erf, erfc
- **Utility**: fmod, remquo, modf, trunc, round, ceil, floor, fabs, ...
- **Extras**: rsqrt, rcbrt, exp10, sinpi, sincos, cospi, erfinv, erfcinv, ...

• New in CUDA 5.0

- sincospi[f]() and normcdf[inv][f]()
- sin(), cos() and erfcinvf() more accurate and faster
- Full list of new features and optimizations:

<http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#math>

<http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#math-performance-improvements>

CUDA libs with OpenACC



```
cufftExecPlan(plan, d_signal, d_signal)
```

```
...
```

```
#pragma acc data deviceptr(d_signal)
```

```
#pragma acc loop independent
```

```
for(i=0; i<n; i++) d_signal[i] = 2 * d_signal[i];
```

Explore the CUDA (Libraries) Ecosystem



● CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

developer.nvidia.com/cuda-tools-ecosystem

The screenshot displays the NVIDIA Developer Zone website. At the top, the NVIDIA logo and 'DEVELOPER ZONE' header are visible, along with navigation links for Developer Centers, Technologies, Tools, Resources, and Community. A search bar is located on the right. The main content area is titled 'GPU-Accelerated Libraries' and includes an introductory paragraph about adding GPU-acceleration to applications. Below this, a grid of library cards is shown, each with an icon, title, and brief description:

- NVIDIA cuFFT**: NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster.
- NVIDIA cuBLAS**: NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library.
- CUDA Tools**: GPU-accelerated linear algebra library by EA Photonics.
- MAGMA**: A collection of next-gen linear algebra routines.
- Intel Fortran Numerical Library**: Developed by RogueWave, a comprehensive set of mathematical and statistical functions.
- NVIDIA cuSPARSE**: NVIDIA CUDA Sparse (cuSPARSE) Matrix library provides a collection of basic linear algebra subroutines.
- CUSP**: A GPU accelerated Open Source C++ library of generic parallel algorithms for sparse linear algebra and graph computations.
- ArrayFire**: AccelerEyes ArrayFire Comprehensive GPU function library, including functions for math, signal and image processing.
- NVIDIA cuRAND**: The CUDA Random Number Generation library performs high quality GPU-accelerated random number generation.
- NVIDIA NPP**: NVIDIA Performance Primitives is a GPU accelerated library with a very large collection of 1000s of image processing functions.
- NVIDIA CUDA Math Library**: An industry proven, highly accurate collection of standard mathematical functions.
- Thrust**: A powerful, open source library of parallel algorithms and data structures.

On the right side of the page, there are sections for 'QUICKLINKS' (including The NVIDIA Registered Developer Program, Registered Developers Website, and various download links), 'FEATURED ARTICLES' (highlighting 'Introducing NVIDIA NSIGHT VISUAL STUDIO EDITION 2.2, WITH LOCAL SINGLE GPU CUDA DEBUGGING!'), and 'LATEST NEWS' (listing 'OpenACC Compiler For \$199' and 'Introducing NVIDIA NSIGHT Visual Studio Edition 2.2, With Local Single GPU CUDA Debugging!').

GPU Technology Conference 2013

The Premier Event in Accelerated Computing

- Four days - **March 18-21, San Jose, CA**
- 300+ sessions
- 100+ research posters
- Networking opportunities with experts & colleagues from 40+ countries

It's everything you want to know about accelerated computing -- all in one place!

Visit www.nvidia.com/gtc-compute for more info.
Use Coupon **GMNVE518267NVLG** for a discount on
registration.

Summary



- **CUDA libraries offer high performance for minimal effort**
- **Robust community of 3rd party libraries**
- **Familiar interfaces make porting legacy code easy (“drop-in”)**
- **Enables focus on core IP**

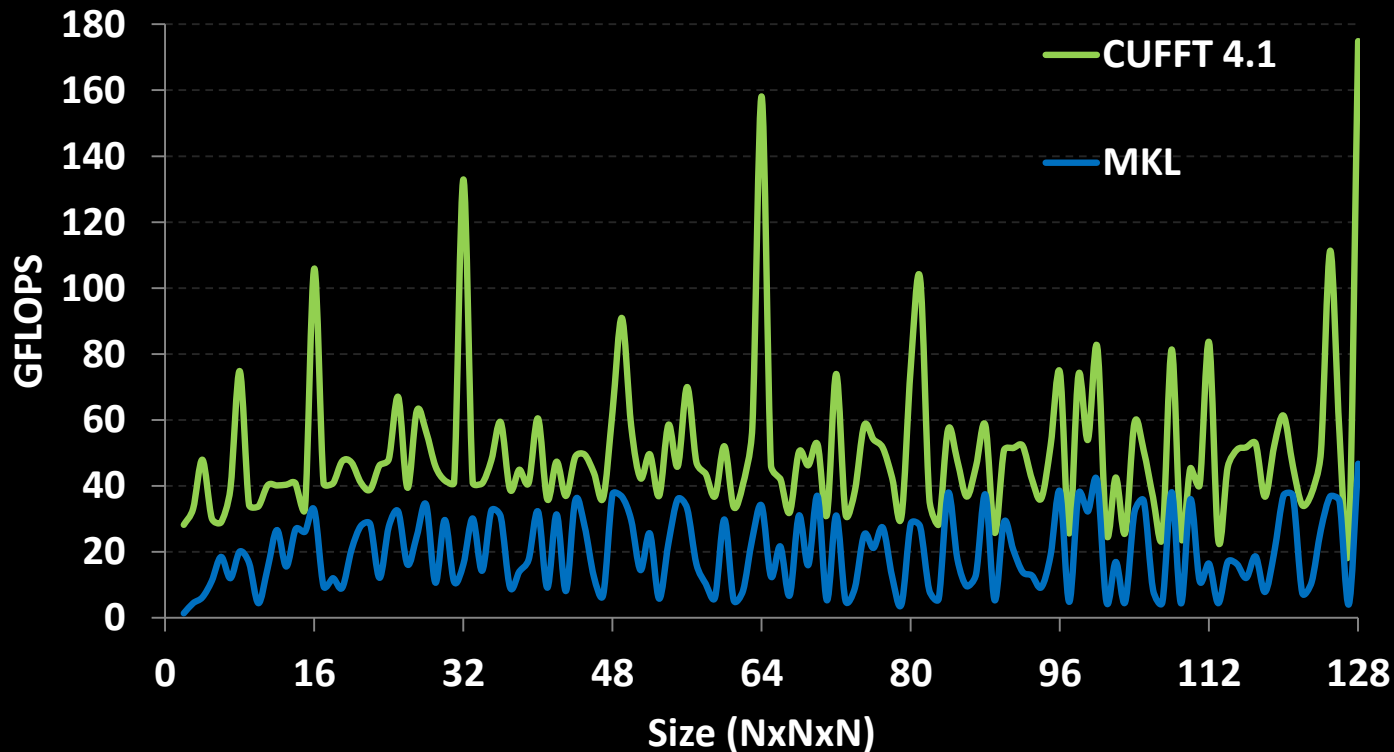
Thank you



Optimized 3D transforms



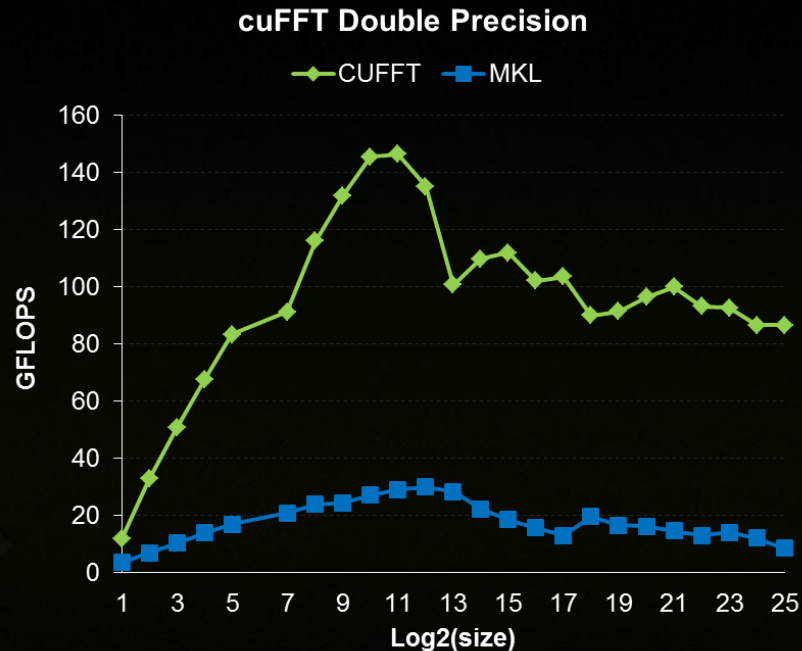
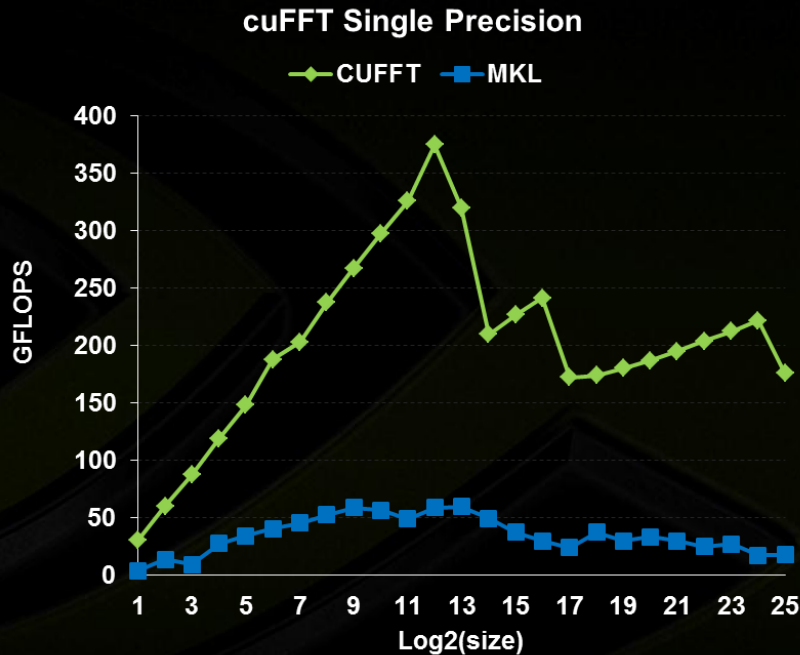
Single Precision All Sizes 2x2x2 to 128x128x128



Performance may vary based on OS version and motherboard configuration

- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

cuFFT Performance -1D



Performance may vary based on OS version and motherboard configuration

- Measured on sizes that are exactly powers-of-2
- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz