# High-productivity Software Development for Accelerators

Thomas Bradley, NVIDIA

# 3 Ways to Accelerate Applications

Applications

| Libraries | OpenACC Directives | Programming Languages |
| --- | --- | --- |
| "Drop-in" Acceleration | Easily Accelerate Applications | Maximum Flexibility |

# 3 Ways to Accelerate Applications

Applications

Libraries

OpenACC Directives

Programming Languages

CUDA Libraries are interoperable with OpenACC

"Drop-in" Acceleration

Easily Accelerate Applications

Maximum Flexibility

# 3 Ways to Accelerate Applications

Applications

| Libraries | OpenACC Directives | Programming Languages |
|---|---|---|
| "Drop-in" Acceleration | Easily Accelerate Applications | Maximum Flexibility |

**CUDA Languages are interoperable with OpenACC**

# 3 Ways to Accelerate Applications

Applications

| Libraries | OpenACC Directives | Programming Languages |
| --- | --- | --- |
| "Drop-in" Acceleration | Easily Accelerate Applications | Maximum Flexibility |

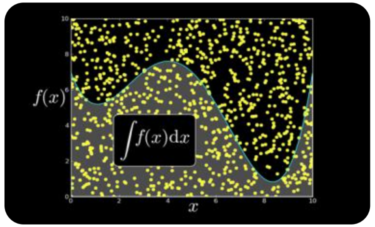
**All of the above!**

# DEVELOPING WITH LIBRARIES

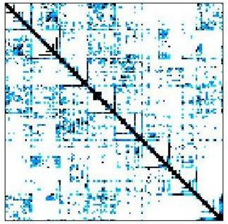
# GPU Accelerated Libraries

## "Drop-in" Acceleration for Your Applications

NVIDIA cuBLAS




NVIDIA cuRAND



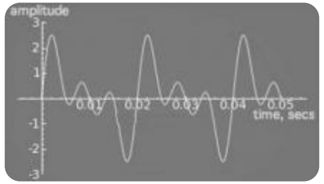
NVIDIA cuSPARSE

NVIDIA NPP

Vector Signal Image Processing

GPU Accelerated Linear Algebra

Matrix Algebra on GPU and Multicore




NVIDIA cuFFT

IMSL Library

Building-block Algorithms for CUDA

Sparse Linear Algebra

C++ STL Features for CUDA

# 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 |

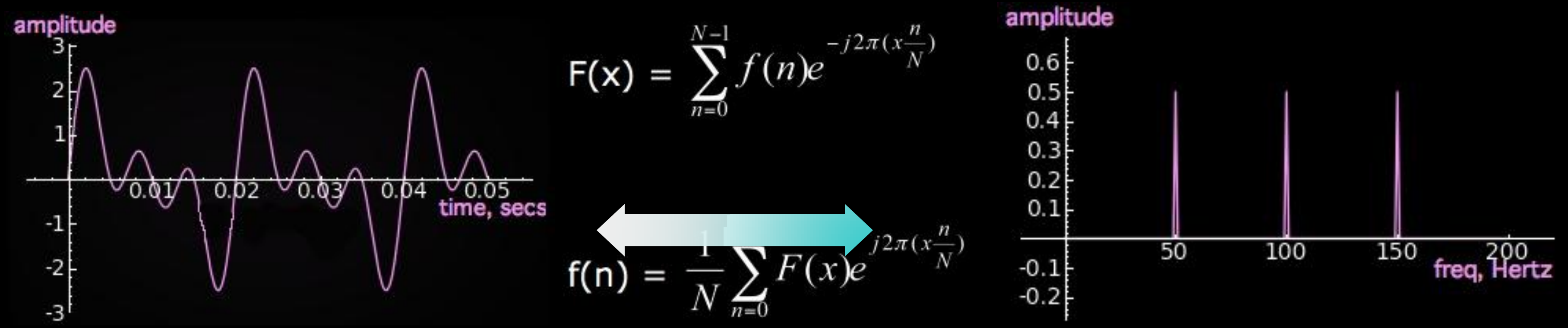
**Included in the CUDA Toolkit** Free download @ www.nvidia.com/getcuda

**More information on CUDA libraries:**

developer.nvidia.com/technologies/libraries

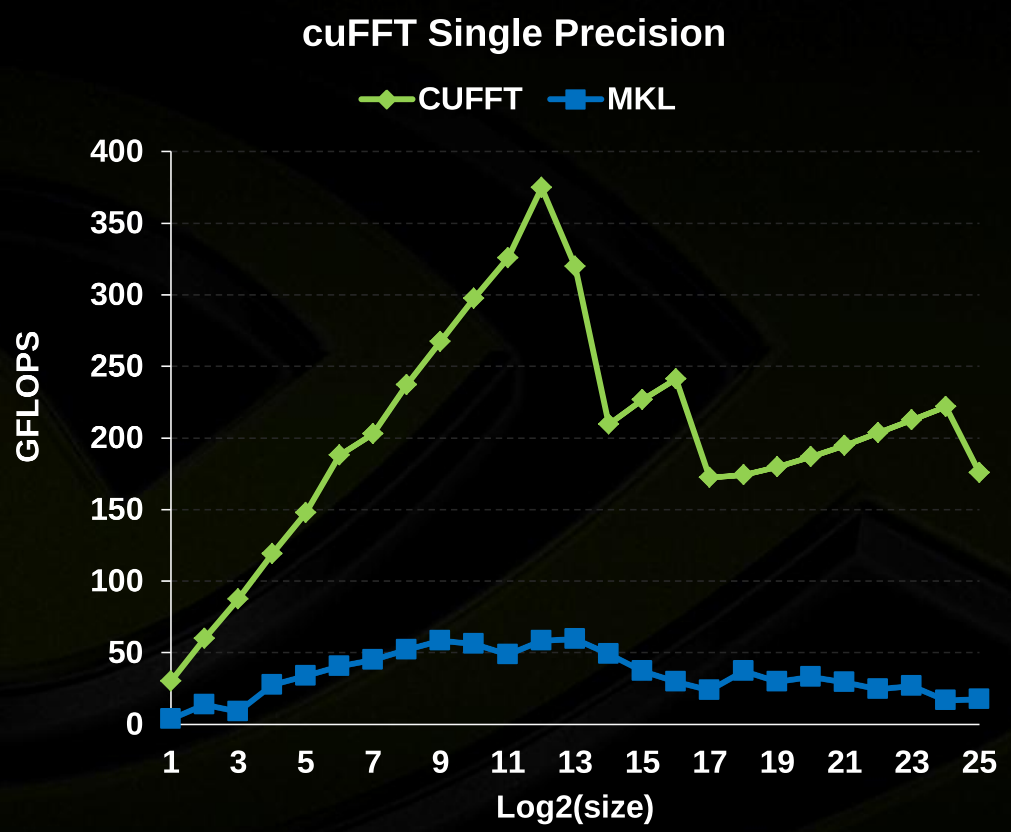
# cuFFT: Multi-dimensional FFTs

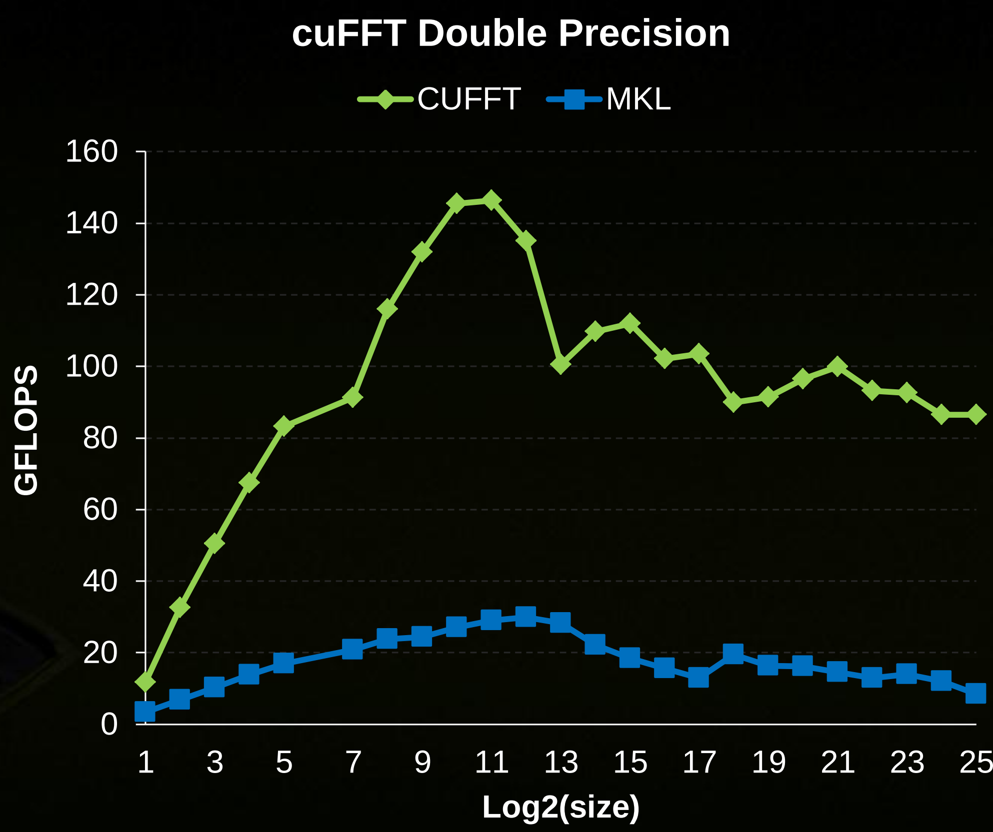
- New in CUDA 4.1
  - Flexible input & output data layouts for all transform types
    - Similar to the FFTW “Advanced Interface”
    - Eliminates extra data transposes and copies
  - API is now thread-safe & callable from multiple host threads
  - Restructured documentation to clarify data layouts



# FFTs up to 10x Faster than MKL

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




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

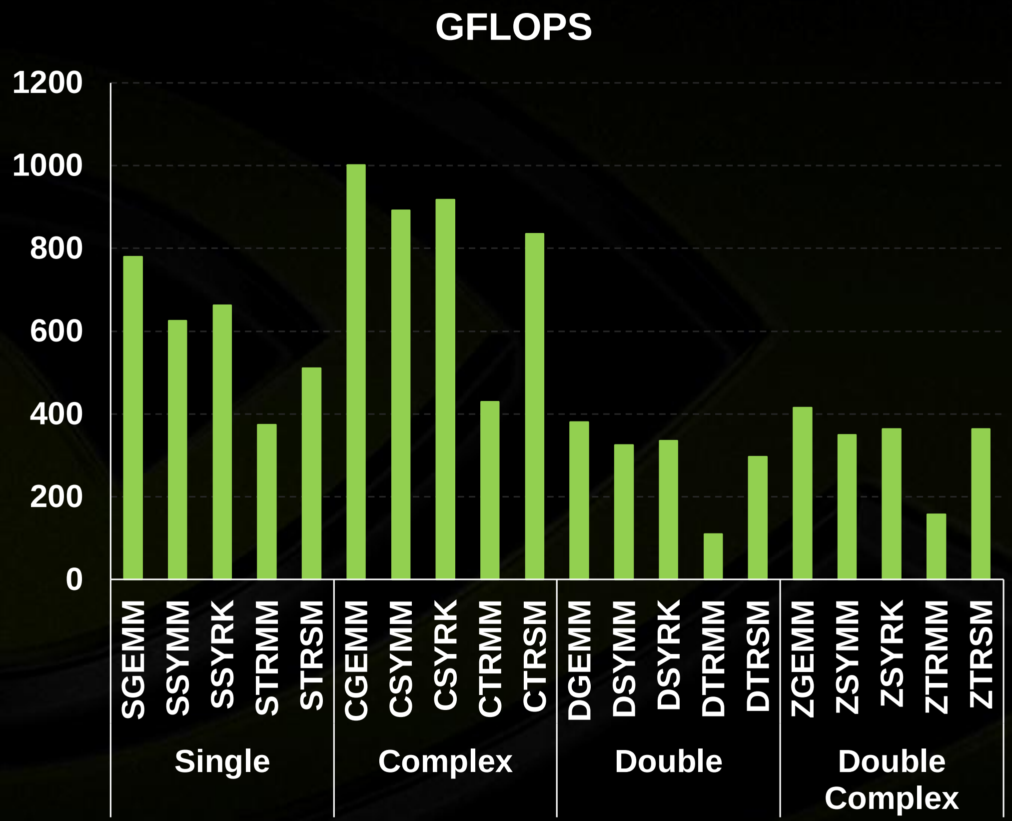
# cuBLAS: Dense Linear Algebra on GPUs

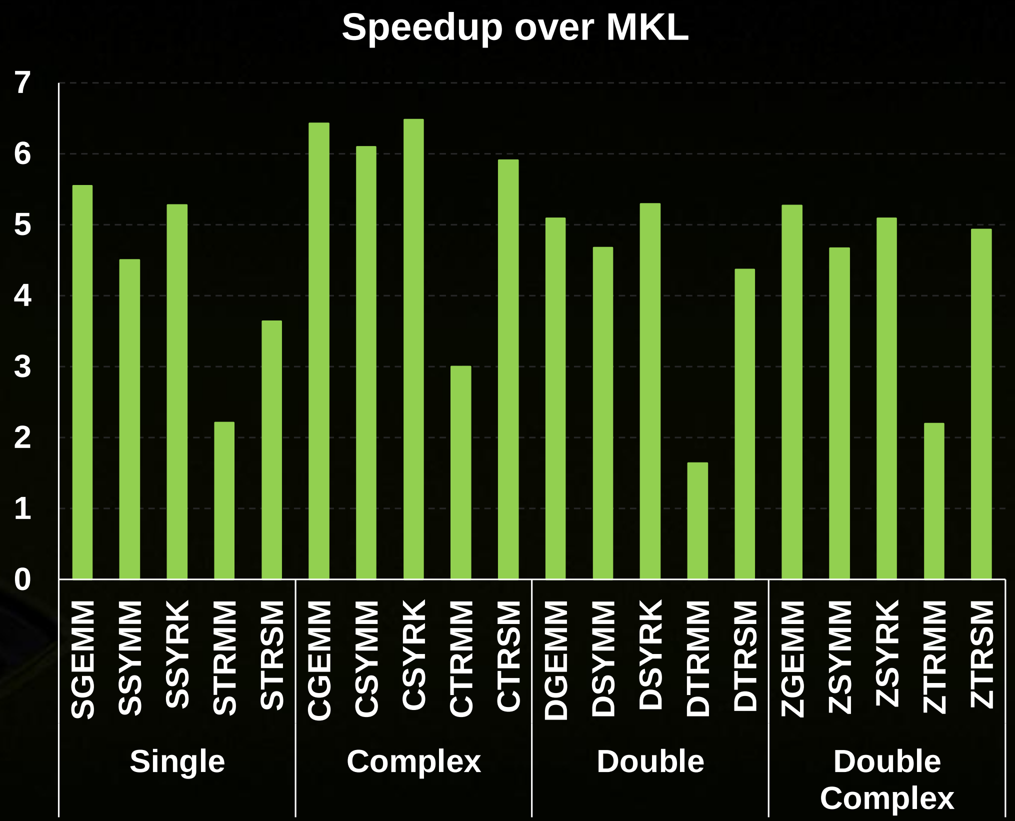
- Complete BLAS implementation plus useful extensions
  - Supports all 152 standard routines for single, double, complex, and double complex

- New in CUDA 4.1
  - New batched GEMM API provides >4x speedup over MKL
    - Useful for batches of 100+ small matrices from 4x4 to 128x128
  - 5%-10% performance improvement to large GEMMs

# cuBLAS Level 3 Performance

## Up to 1 TFLOPS sustained performance and >6x speedup over Intel MKL

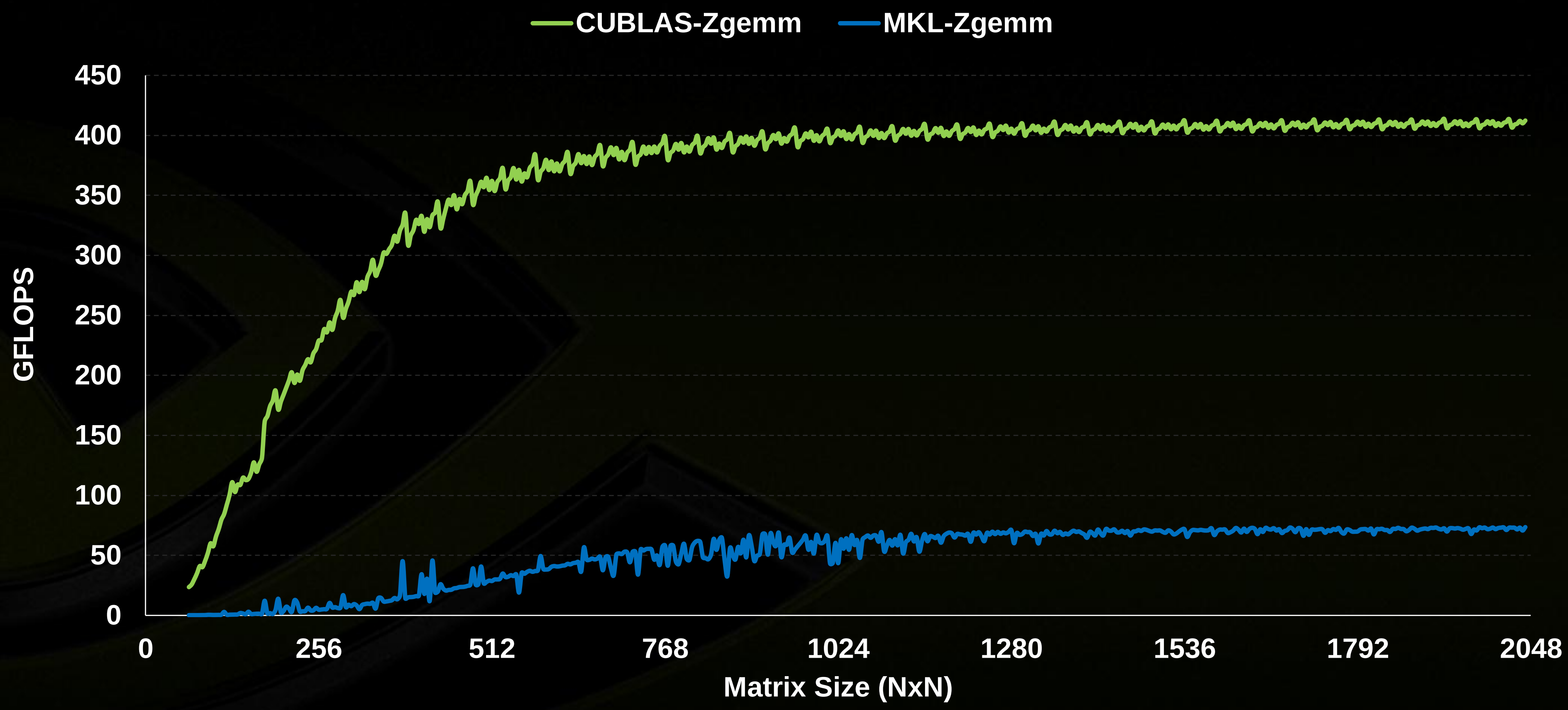



- 4Kx4K matrix size
- cuBLAS 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

Performance may vary based on OS version and motherboard configuration

# ZGEMM Performance vs Intel MKL




Performance may vary based on OS version and motherboard configuration

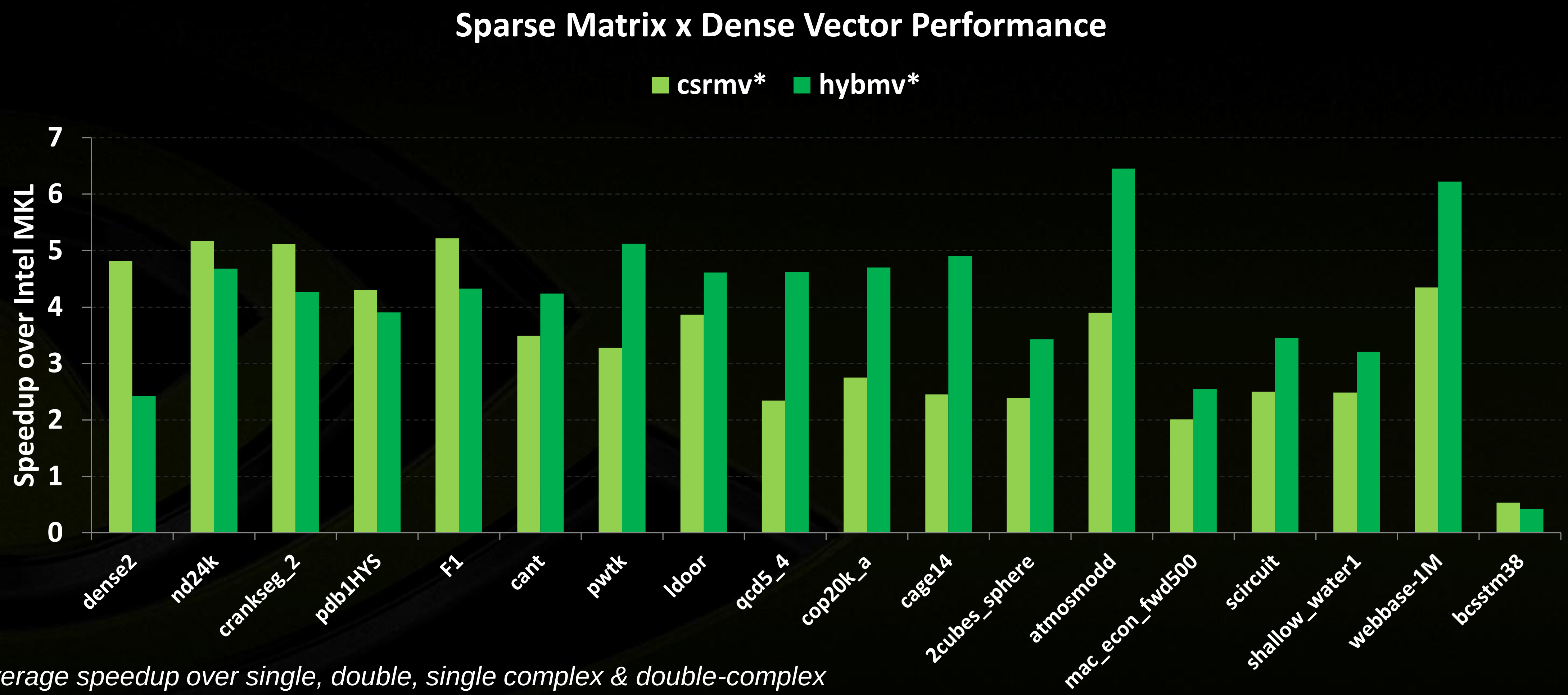
- cuBLAS 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

# cuSPARSE: Sparse linear algebra routines

- Sparse matrix-vector multiplication & triangular solve
  - APIs optimized for iterative methods
- New in 4.1
  - Tri-diagonal solver with speedups up to 10x over Intel MKL
  - ELL-HYB format offers 2x faster matrix-vector multiplication

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & \cdots & \cdots & \cdots \\ 2.0 & 3.0 & \cdots & \cdots \\ \cdots & \cdots & 4.0 & \cdots \\ 5.0 & \cdots & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

# cuSPARSE is >6x Faster than Intel MKL



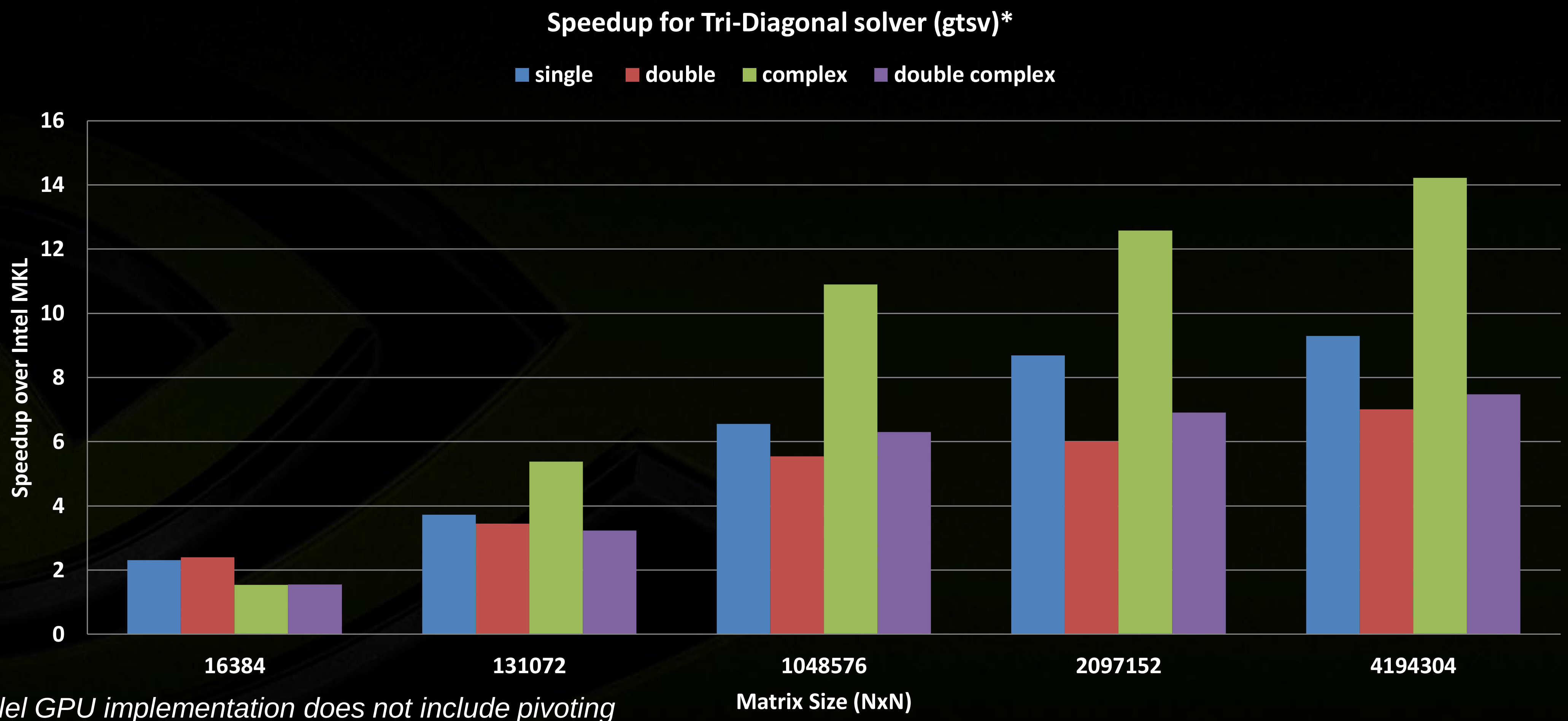

**Average speedup over single, double, single complex & double-complex*

Performance may vary based on OS version and motherboard configuration



- cuSPARSE 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

# Tri-diagonal solver performance vs. MKL




*Parallel GPU implementation does not include pivoting*

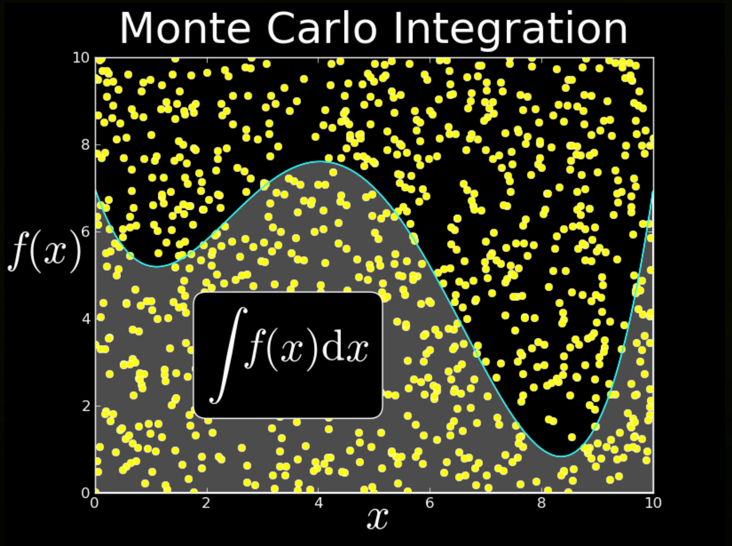
Performance may vary based on OS version and motherboard configuration

- cuSPARSE 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

# cuRAND: Random Number Generation

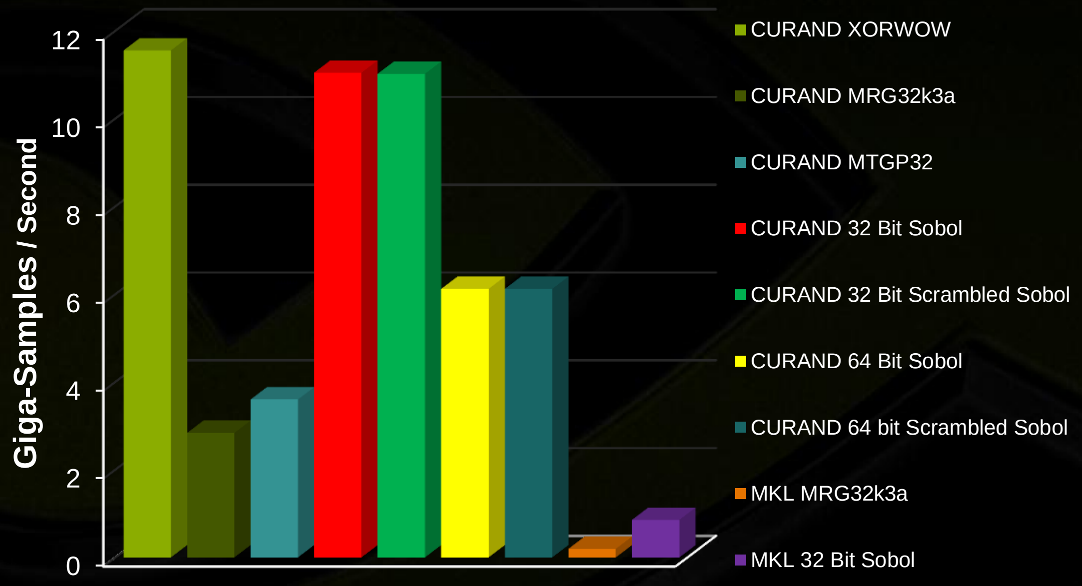
- Pseudo- and Quasi-RNGs
- Supports several output distributions
- Statistical test results reported in documentation

- New commonly used RNGs in CUDA 4.1
  - MRG32k3a RNG
  - MTGP11213 Mersenne Twister RNG

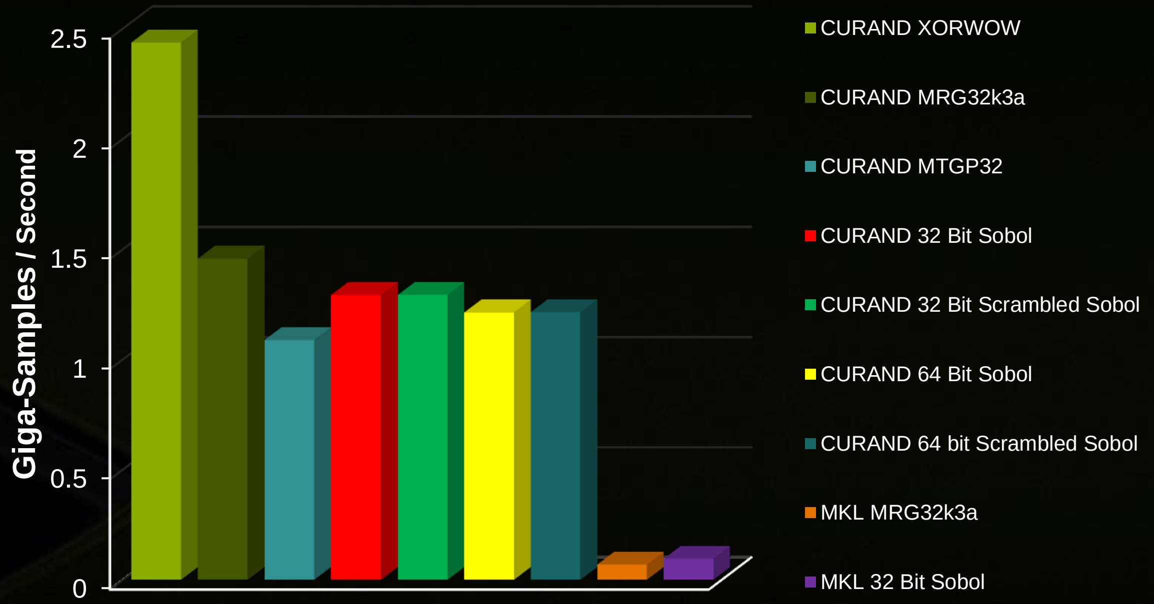


# cuRAND Performance compared to Intel MKL

## Double Precision Uniform Distribution




## Double Precision Normal Distribution




Performance may vary based on OS version and motherboard configuration

- cuRAND 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 @ 3.33 GHz

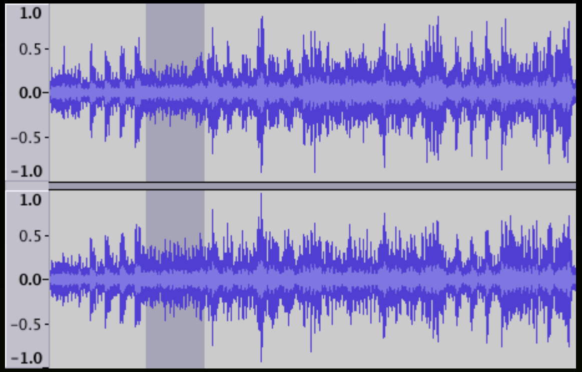
# 1000+ New Functions in NPP 4.1

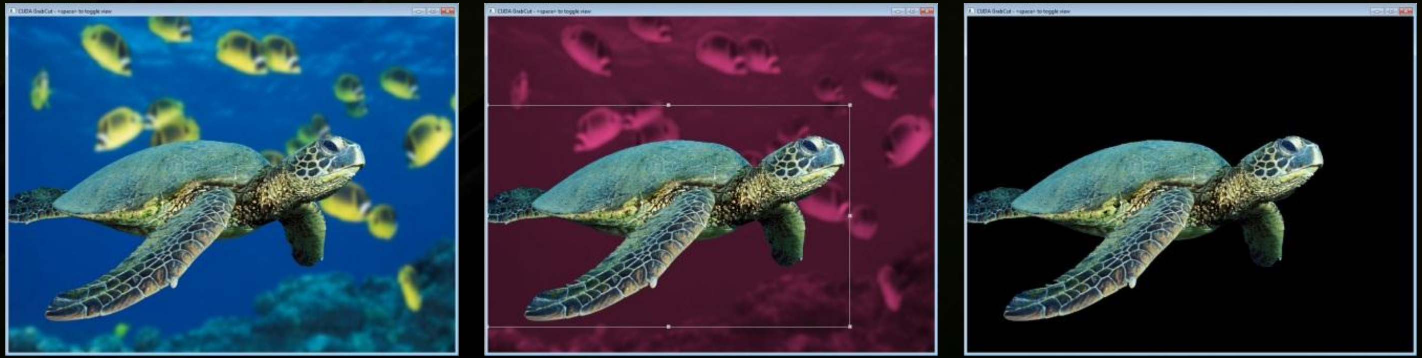
Up to **40x** speedups

- NVIDIA Performance Primitives (NPP) library includes over 2200 GPU-accelerated functions for image & signal processing

  Arithmetic, Logic, Conversions, Filters, Statistics, etc.

- Most are 5x-10x faster than analogous routines in Intel IPP



http://developer.nvidia.com/content/graphcuts-using-npp

* NPP 4.1, NVIDIA C2050 (Fermi)
* IPP 6.1, Dual Socket Core™ i7 920 @ 2.67GHz

# USING CUDA LIBRARIES WITH OPENACC

# Sharing Data with Libraries

- CUDA libraries and OpenACC both operate on device arrays

- OpenACC provides mechanisms for interop with library calls
  - deviceptr data clause
  - host_data construct

- Note: same mechanisms provide interop with custom CUDA C/C++/Fortran code

# deviceptr Data Clause

`deviceptr( list )` **Declares that the pointers in *list* refer to device pointers that need not be allocated or moved between the host and device for this pointer.**

**Example:**

**C**

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

**Fortran**

```
$!acc data deviceptr(d_input)
```

# host_data Construct

**Makes the address of device data available on the host.**

`host_data( list )` **Tells the compiler to use the device address for any variable in *list*. Variables in the list must be present in device memory due to data regions that contain this construct**

**Example**

**C**

```
#pragma acc host_data use_device(d_input)
```

**Fortran**

```
$!acc host_data use_device(d_input)
```

# Example: 1D convolution using cuFFT

- Perform convolution in frequency space
  1. Use cuFFT to transform input signal and filter kernel into the frequency domain
  2. Perform point-wise complex multiply and scale on transformed signal
  3. Use cuFFT to transform result back into the time domain

- Perform step 2 using OpenACC

- Code walk-through follows, code available

# Source Excerpt

```
// Transform signal and kernel
error = cufftExecC2C(plan, (cufftComplex *)d_signal,
                     (cufftComplex *)d_signal, CUFFT_FORWARD);
error = cufftExecC2C(plan, (cufftComplex *)d_filter_kernel,
                     (cufftComplex *)d_filter_kernel, CUFFT_FORWARD);

// Multiply the coefficients together and normalize the result
printf("Performing point-wise complex multiply and scale.\n");
complexPointwiseMulAndScale(new_size,
                            (float *restrict)d_signal,
                            (float *restrict)d_filter_kernel);

// Transform signal back
error = cufftExecC2C(plan, (cufftComplex *)d_signal,
                     (cufftComplex *)d_signal, CUFFT_INVERSE);
```

This function must execute on device data

# OpenACC convolution code

```
void complexPointwiseMulAndScale(int n, float *restrict signal,
                                 float *restrict filter_kernel)
{
// Multiply the coefficients together and normalize the result
#pragma acc data deviceptr(signal, filter_kernel)
    {
#pragma acc kernels loop independent
        for (int i = 0; i < n; i++) {
            float ax = signal[2*i];
            float ay = signal[2*i+1];
            float bx = filter_kernel[2*i];
            float by = filter_kernel[2*i+1];
            float s = 1.0f / n;
            float cx = s * (ax * bx - ay * by);
            float cy = s * (ax * by + ay * bx);
            signal[2*i] = cx;
            signal[2*i+1] = cy;
        }
    }
}
```

Note: The PGI C compiler does not currently support structs in OpenACC loops, so we cast the Complex* pointers to float* pointers and use interleaved indexing

# Summary

- Use deviceptr data clause to pass pre-allocated device data to OpenACC regions and loops

- Use host_data to get device address for pointers inside acc data regions

- The same techniques shown here can be used to share device data between OpenACC loops and
  - Your custom CUDA C/C++/Fortran/etc. device code
  - Any other CUDA Library that uses CUDA device pointers

# DEVELOPING WITH THRUST

# Rapid Parallel C++ Development

- Resembles C++ STL
  - High-level interface
  - Enhances developer productivity
  - Enables performance portability between GPUs and multicore CPUs
- Flexible
  - CUDA, OpenMP, and TBB backends
- Extensible and customizable
  - Integrates with existing software
- Open source

```
// generate 32M random numbers on host
thrust::host_vector<int> h_vec(32 << 20);
thrust::generate(h_vec.begin(),
                 h_vec.end(),
                 rand);

// transfer data to device (GPU)
thrust::device_vector<int> d_vec = h_vec;

// sort data on device
thrust::sort(d_vec.begin(), d_vec.end());

// transfer data back to host
thrust::copy(d_vec.begin(),
             d_vec.end(),
             h_vec.begin());
```

http://developer.nvidia.com/thrust or http://thrust.googlecode.com

# Processing Rainfall Data


```
day         [0  0  1  2  5  5  6  6  7  8  ... ]
site        [2  3  0  1  1  2  0  1  2  1  ... ]
measurement [9  5  6  3  3  8  2  6  5  10 ... ]
```

Notes

1) Time series sorted by day
2) Measurements of zero are excluded from the time series

# Storage Options

- Array of structures

```
struct Sample
{
  int day;
  int site;
  int measurement;
};
thrust::device_vector<Sample> data;
```

- Structure of arrays (Best Practice)

```
struct Data
{
  thrust::device_vector<int> day;
  thrust::device_vector<int> site;
  thrust::device_vector<int> measurement;
};
Data data;
```

# Processing Rainfall Data

- Total rainfall at a given site
- Total rainfall at each site
- Total rainfall between given days
- Number of days with any rainfall
- Number of days where rainfall exceeded 5
- Day where total rainfall reached 32

- Additional problem: Sort unsorted input

# Total Rainfall at a Given Site

```
struct one_site_measurement
{
  int site;
  one_site_measurement(int site) : site(site) {}

  __host__ __device__ int operator()(thrust::tuple<int,int> t)
  {
    if( thrust::get<0>(t) == site )
      return thrust::get<1>(t);
    else
      return 0;
  }
};

int compute_total_rainfall_at_one_site(int i, const Data &data)
{
  // Fused transform-reduce (best practice).
  return thrust::transform_reduce(
    thrust::make_zip_iterator(thrust::make_tuple(data.site.begin(), data.measurement.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(data.site.end(),   data.measurement.end())),
    one_site_measurement(i),
    0,
    thrust::plus<int>());
}
```

# Total Rainfall at Each Site

```
template <typename Vector>
void compute_total_rainfall_per_site(const Data &data, Vector &site, Vector &measurement)
{
  // Copy data to keep the original data as it is.
  Vector tmp_site(data.site);
  Vector tmp_measurement(data.measurement);

  // Sort the “pairs” (site, measurement) by increasing value of site.
  thrust::sort_by_key(tmp_site.begin(), tmp_site.end(), tmp_measurement.begin());

  // Reduce measurements by site (Assumption: site/measurement are big enough).
  thrust::reduce_by_key(tmp_site.begin(), tmp_site.end(), tmp_measurement.begin(),
                        site.begin(),
                        measurement.begin());
}
```

```
tmp_site         [0 = 0   1 = 1 = 1 = 1    2 = 2 = 2   3  ... ]
tmp_measurement  [6 + 2   3 + 3 + 6 + 10   9 + 8 + 5   5  ... ]
```

```
site         [0   1   2   3  ... ]
measurement  [8  22  22   5  ... ]
```

# Total Rainfall Between Given Days

```
int compute_total_rainfall_between_days(int first_day, int last_day, const Data &data)
{
  // Search first_day/last_day using binary searches.
  int first = thrust::lower_bound(data.day.begin(), data.day.end(), first_day) - data.day.begin();
  int last  = thrust::upper_bound(data.day.begin(), data.day.end(), last_day)  - data.day.begin();

  // Reduce the measurements between the two bounds.
  return thrust::reduce(data.measurement.begin() + first, data.measurement.begin() + last);
}
```

```
                 lower_bound( ... , 2)      upper_bound( ... , 6)

day          [0   0   1   2   5   5   6   6   7   8   ... ]
measurement  [9   5   6   3   3   8   2   6   5  10   ... ]
```

# Number of Days with Any Rainfall

```
int compute_number_of_days_with_rainfall(const Data &data)
{
  return thrust::inner_product(data.day.begin(), data.day.end() - 1,
                               data.day.begin() + 1,
                               1,
                               thrust::plus<int>(),
                               thrust::not_equal_to<int>());
}
```

day [0 0 1 2 5 5 6 6 7 8 ... ]

= ≠ ≠ ≠ = ≠ = ≠ ≠

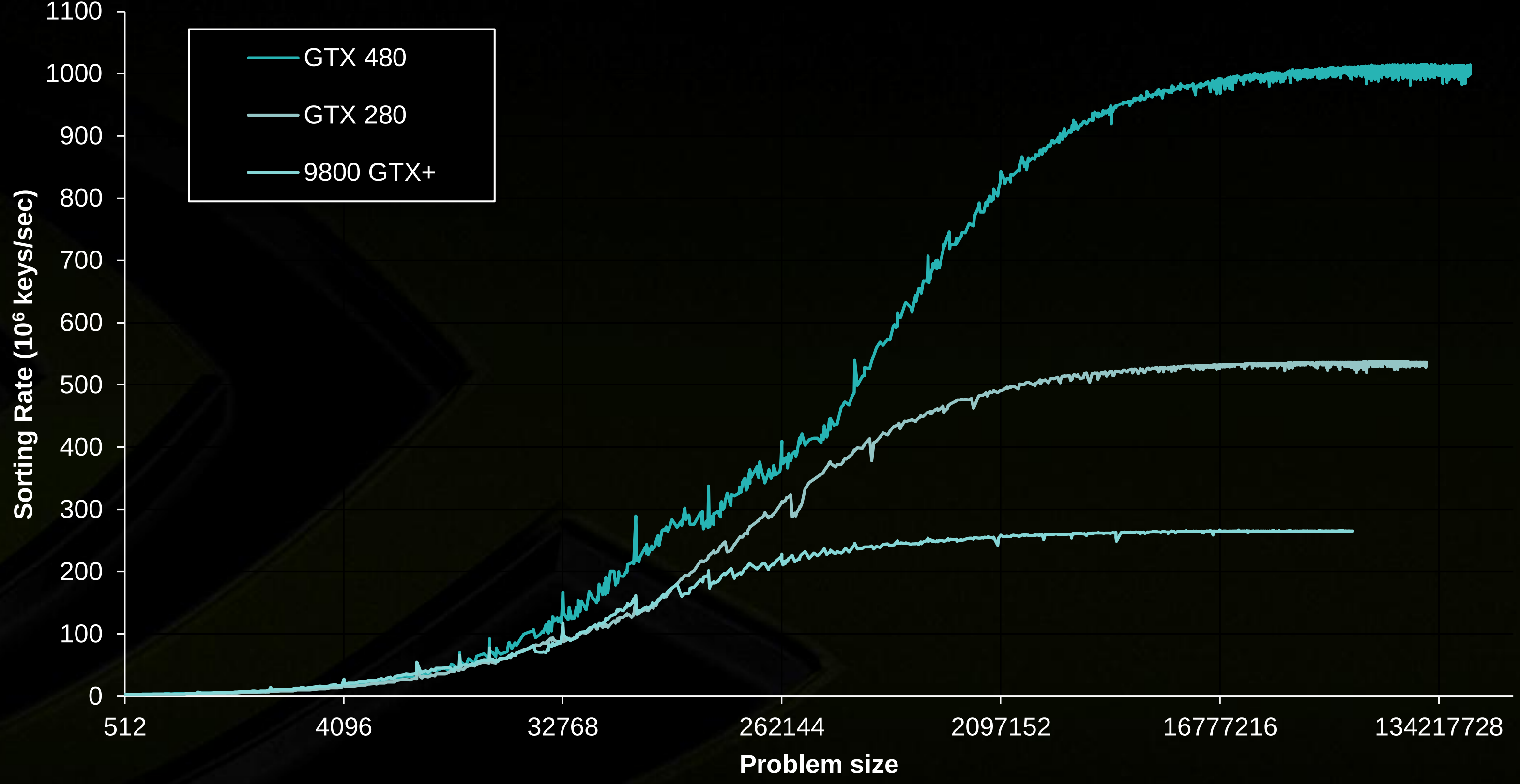
day shifted [0 0 1 2 5 5 6 6 7 8 ... ]

[0 + 1 + 1 + 1 + 0 + 1 + 0 + 1 + 1 ... ] + 1

# Number of Days where Rainfall Exceeded 5

```
using namespace thrust::placeholders;

int count_days_where_rainfall_exceeded_5(const Data &data)
{
  size_t N = compute_number_of_days_with_rainfall(data);

  thrust::device_vector<int> day(N);
  thrust::device_vector<int> measurement(N);

  thrust::reduce_by_key(
    data.day.begin(), data.day.end(),
    data.measurement.begin(),
    day.begin(),
    measurement.begin());

  return thrust::count_if(measurement.begin(), measurement.end(), _1 > 5);
}
```

# Scalability




Architectural Advancement → Performance Improvement

# Calling Thrust from CUDA Fortran

C wrapper for Thrust: csort.cu

```
#include <thrust/device_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>

extern "C" {
//Sort for integer arrays
void sort_int_wrapper( int *data, int N)
{
  // Wrap raw pointer with a device_ptr
    thrust::device_ptr<int> dev_ptr(data);
 // Use device_ptr in Thrust sort algorithm
    thrust::sort(dev_ptr, dev_ptr+N);
}

//Sort for single precision arrays
void sort_float_wrapper( float *data, int N)
{
   thrust::device_ptr<float> dev_ptr(data);
   thrust::sort(dev_ptr, dev_ptr+N);
}

//Sort for double precision arrays
void sort_double_wrapper( double *data, int N)
{
   thrust::device_ptr<double> dev_ptr(data);
   thrust::sort(dev_ptr, dev_ptr+N);
}
}
```

# Calling Thrust from CUDA Fortran

Fortran interface to C wrapper using ISO C Bindings

```fortran
module thrust

 interface thrustsort

 subroutine sort_int( input,N) bind(C,name="sort_int_wrapper")
  use iso_c_binding
  integer(c_int),device:: input(*)
  integer(c_int),value:: N
 end subroutine

subroutine sort_double( input,N) bind(C,name="sort_double_wrapper")
  use iso_c_binding
  real(c_double),device:: input(*)
  integer(c_int),value:: N
 end subroutine

subroutine sort_float( input,N) bind(C,name="sort_float_wrapper")
  use iso_c_binding
  real(c_float),device:: input(*)
  integer(c_int),value:: N
 end subroutine

end interface
end module
```

# CUDA Fortran sorting with Thrust

```
program testsort
use thrust
real, allocatable :: cpuData(:)
real, allocatable, device :: gpuData(:)
integer:: N=10
!Allocate CPU and GPU arrays
  allocate(cpuData(N),gpuData(N))
!Fill the host array with random data
 do i=1,N
   cpuData(i)=random(i)
 end do
! Print unsorted data
  print *, cpuData
! Send data to GPU
 gpuData = cpuData
!Sort the data
 call thrustsort(gpuData,N)
!Copy the result back
 cpuData = gpuData
! Print sorted data
  print *, cpuData
!Deallocate arrays
 deallocate(cpuData,gpuData)
end program testsort
```

```
nvcc -c -arch sm_20 csort.cu
pgf90 -rc=rc4.0 -Mcuda=cc20 -O3 -o testsort thrust_module.cuf testsort.cuf csort.o
```

```
 $ ./testsort
Before sorting 4.1630346E-02 0.9124327 0.7832350 0.6540373 100.0000 0.3956419 0.2664442 0.13724658.0488138E-03 0.8788511
After sorting 8.0488138E-03 4.1630346E-02 0.1372465 0.26644420.3956419 0.6540373 0.7832350 0.87885110.9124327 100.0000
```

# SIX WAYS TO SAXPY

# Single precision Alpha X Plus Y (SAXPY)

Part of Basic Linear Algebra Subroutines (BLAS) Library

$$z = \alpha x + y$$

$x, y, z$ : vector

$\alpha$ : scalar

GPU SAXPY in multiple languages and libraries

A menagerie* of possibilities, not a tutorial



*technically, a *program chrestomathy*: http://en.wikipedia.org/wiki/Chrestomathy

# cuBLAS Library

## *Serial BLAS Code*

```
int N = 1<<20;


...

// Use your choice of blas library

// Perform SAXPY on 1M elements
blas_saxpy(N, 2.0, x, 1, y, 1);
```

## *Parallel cuBLAS Code*

```
int N = 1<<20;


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

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

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

cublasShutdown();
```

You can also call cuBLAS from Fortran, C++, Python, and other languages

http://developer.nvidia.com/cublas

# OpenACC Compiler Directives

## *Parallel C Code*

```
void saxpy(int n,
           float a,
           float *x,
           float *y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

## *Parallel Fortran Code*

```
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
$!acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
$!acc end kernels
end subroutine saxpy

...
$ Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x, y)
...
```

http://developer.nvidia.com/openacc

# Thrust C++ Template Library

## *Serial C++ Code*
### *with STL and Boost*

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...




// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
               2.0f * _1 + _2);
```

www.boost.org/libs/lambda

## *Parallel C++ Code*

```
int N = 1<<20;
thrust::host_vector<float> h_x(N), h_y(N);

...

thrust::device_vector<float> x = h_x;
thrust::device_vector<float> y = h_y;


// Perform SAXPY on 1M elements
thrust::transform(x.begin(), x.end(),
                  y.begin(), y.begin(),
                  2.0f * _1 + _2);
```

http://developer.nvidia.com/thrust

# CUDA C

## *Standard C Code*

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;



// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

## *Parallel C Code*

```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, h_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, h_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(h_y, y, N, cudaMemcpyDeviceToHost);
```

http://developer.nvidia.com/cuda-toolkit

# CUDA Fortran

## *Standard Fortran*

```
module mymodule contains
  subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i)+y(i)
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  $ Perform SAXPY on 1M elements
  call saxpy(2**20, 2.0, x, y)

end program main
```

## *Parallel Fortran*

```
module mymodule contains
  attributes(global) subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    attributes(value) :: a, n
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i)+y(i)
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  $ Perform SAXPY on 1M elements
  call saxpy<<<4096,256>>>(2**20, 2.0, x, y)

end program main
```



http://developer.nvidia.com/cuda-fortran

# Thank you

GPU TECHNOLOGY CONFERENCE

# GPU Technology Conference 2012
# May 14-17 | San Jose, California

*Developer Education:* Parallel Programming Languages and Compilers, Tools, Optimization, Libraries, Directives Based Programming

*Research Areas:* Bioinformatics, Computational Physics, Supercomputing, Energy Exploration, Climate & Weather, Finance, CFD, Machine Learning & AI, Quantum Chemistry, and much more!

Register today and learn from the experts!

Academics are entitled to a 40% discount, so please use code GA40CF when you register.

**Learn more at www.gputechconf.com**