GPU TECHNOLOGY CONFERENCE

# CUDA Libraries and Ecosystem Overview

Peter Messmer, NVIDIA

# 3 Ways to Accelerate on GPU

Application

Libraries | Directives | Programming Languages

Easiest Approach for 2x to 10x Acceleration — Maximum Performance

Effort Level

# Constant progress on library development

2007 | 2008 | 2009 | 2010 | 2011

**CUDA Toolkit 1.x**
- Single precision
- cuBLAS
- cuFFT
- math.h

**CUDA Toolkit 2.x**
- Double Precision support in all libraries

**CUDA Toolkit 3.x**
- cuSPARSE
- cuRAND
- printf()
- malloc()

**CUDA Toolkit 4.x**
- Thrust
- NPP
- assert()

# CUDA Math Libraries

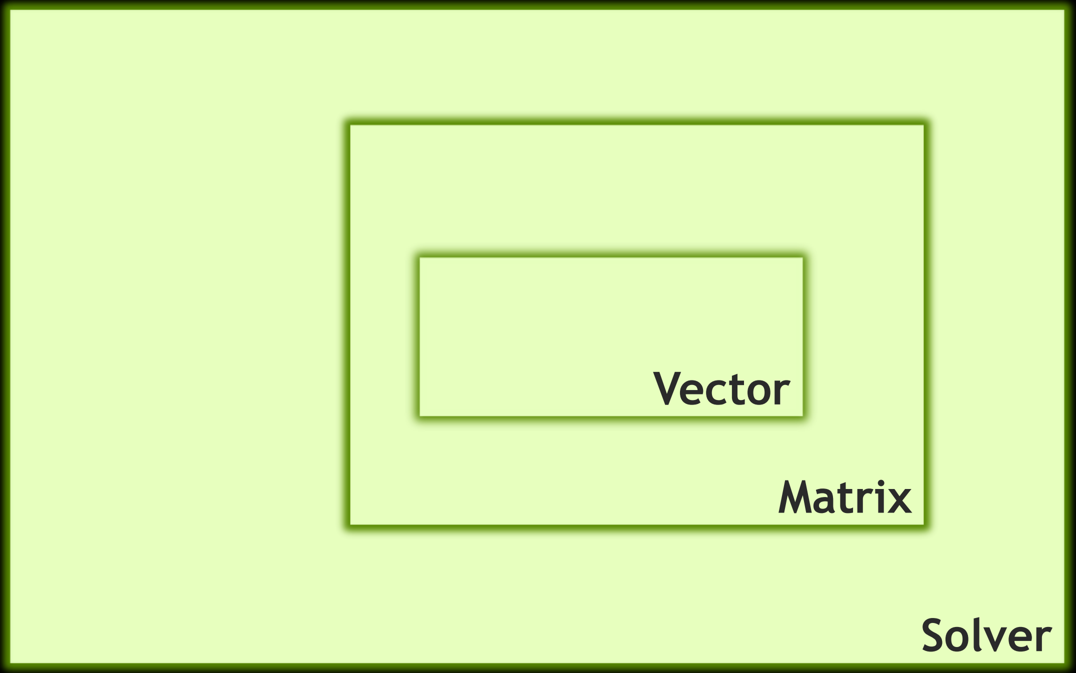
High performance math routines for your applications:

- 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

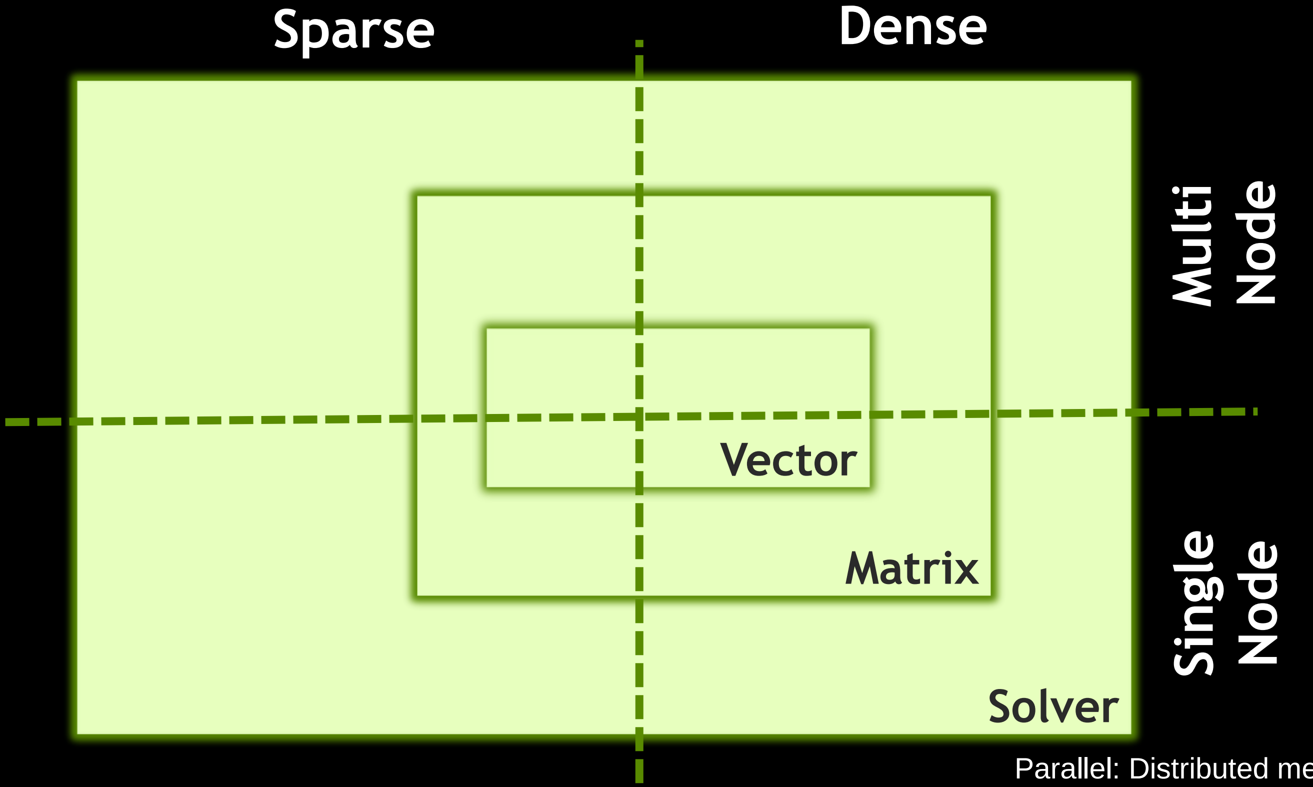
Included in the CUDA Toolkit    Free download @ www.nvidia.com/getcuda

# Linear Algebra

# A Birds Eye View on Linear Algebra

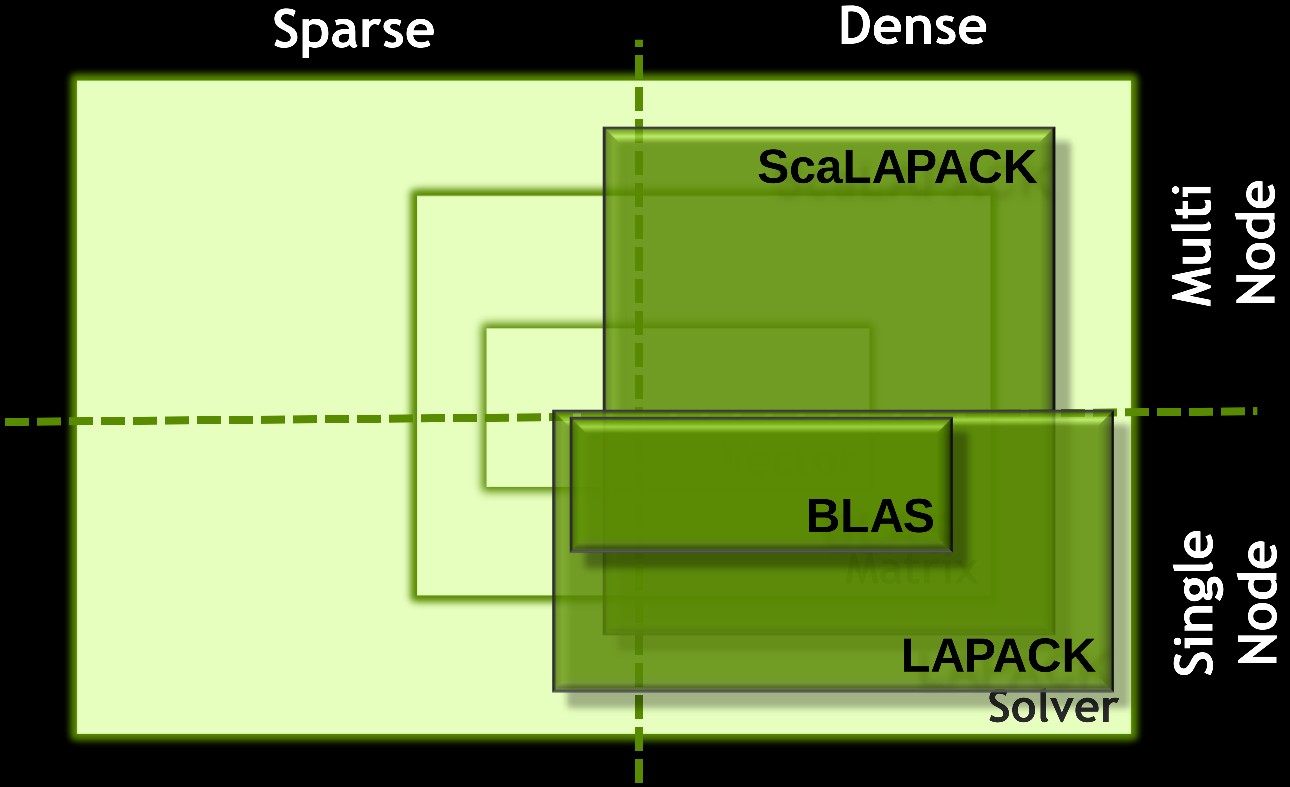


# A Birds Eye View on Linear Algebra

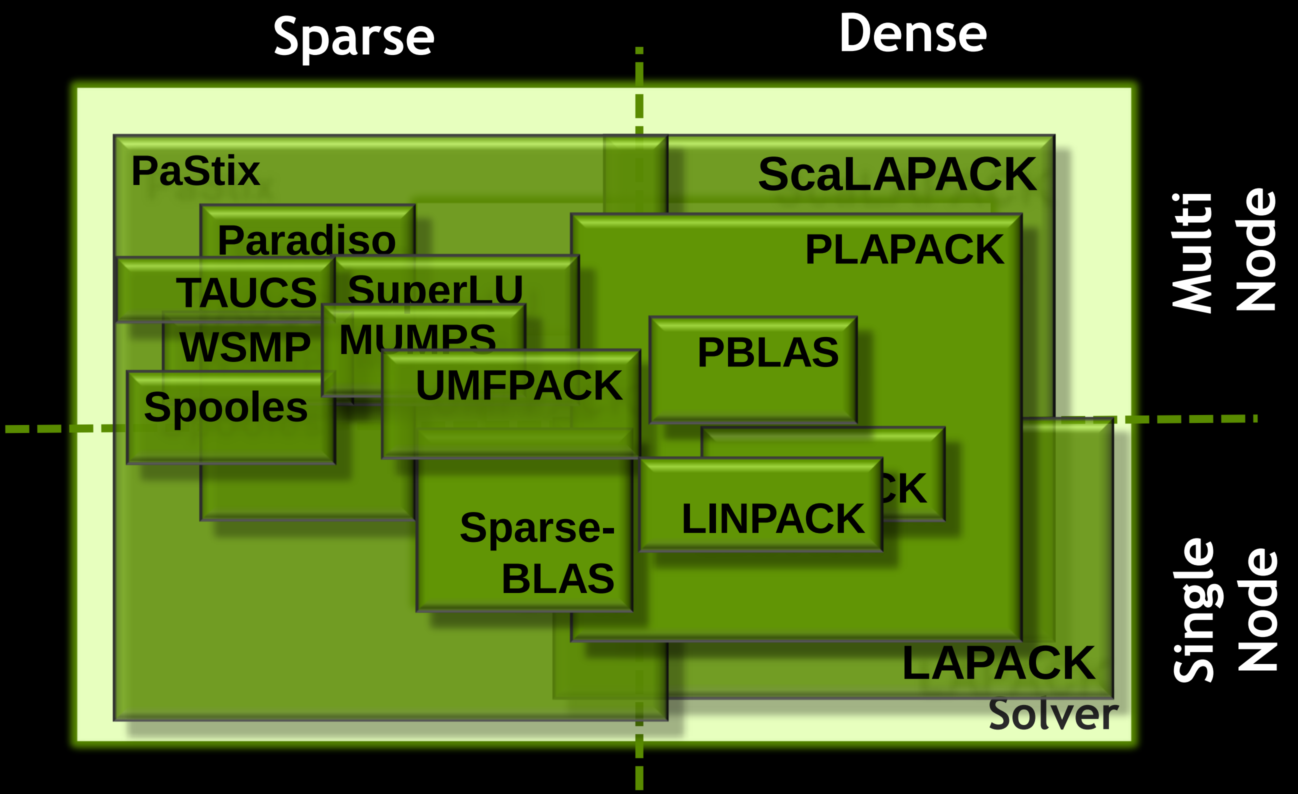



Parallel: Distributed memory parallel
Serial: Single node

# Sometimes it seems as if there's only three …

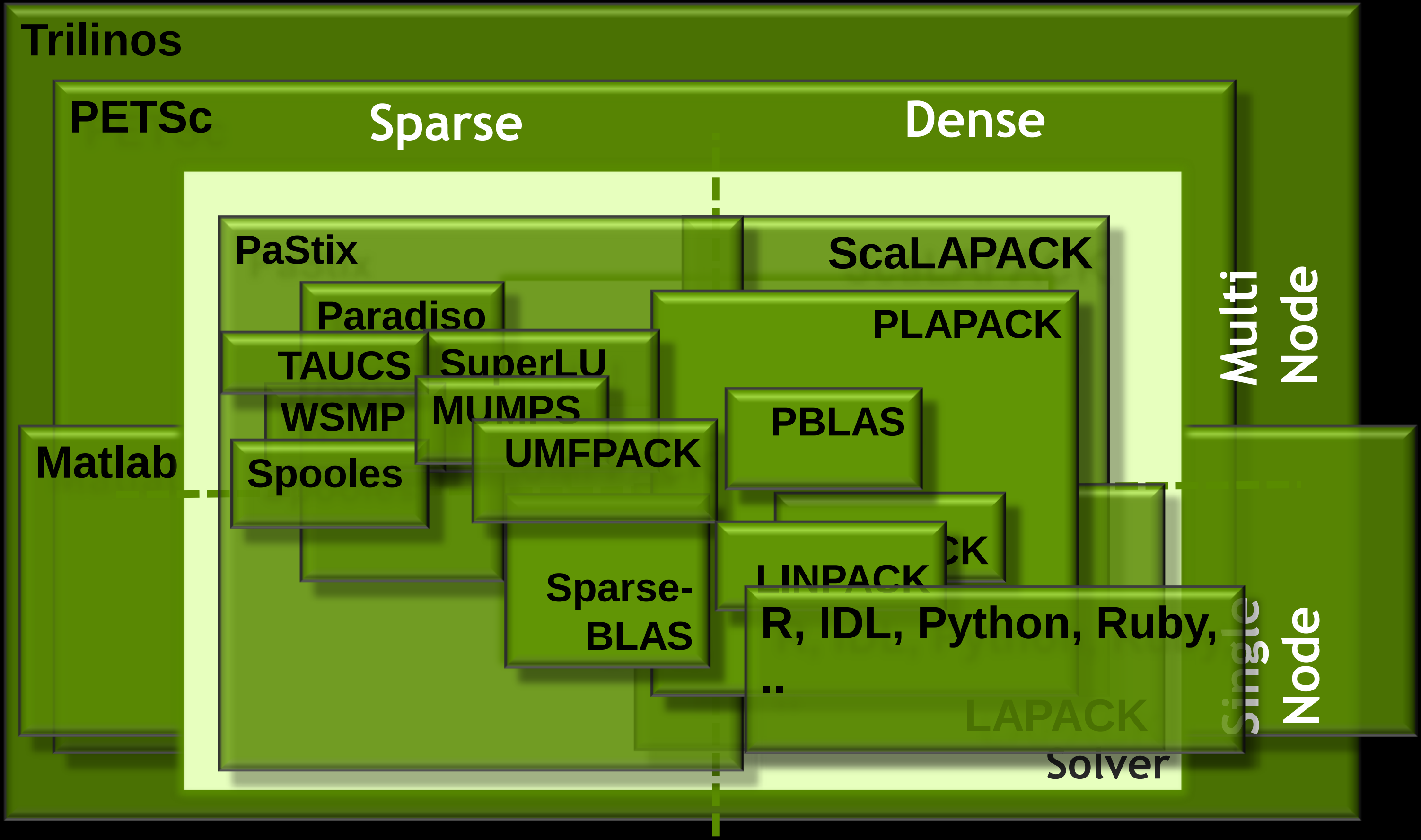


# .. but there is more ...

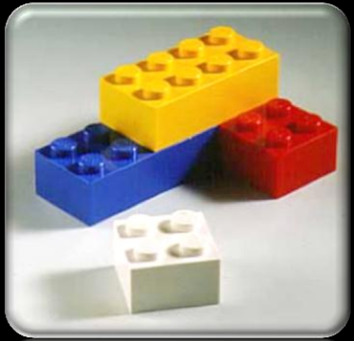



GPU TECHNOLOGY CONFERENCE

# … and even more ..



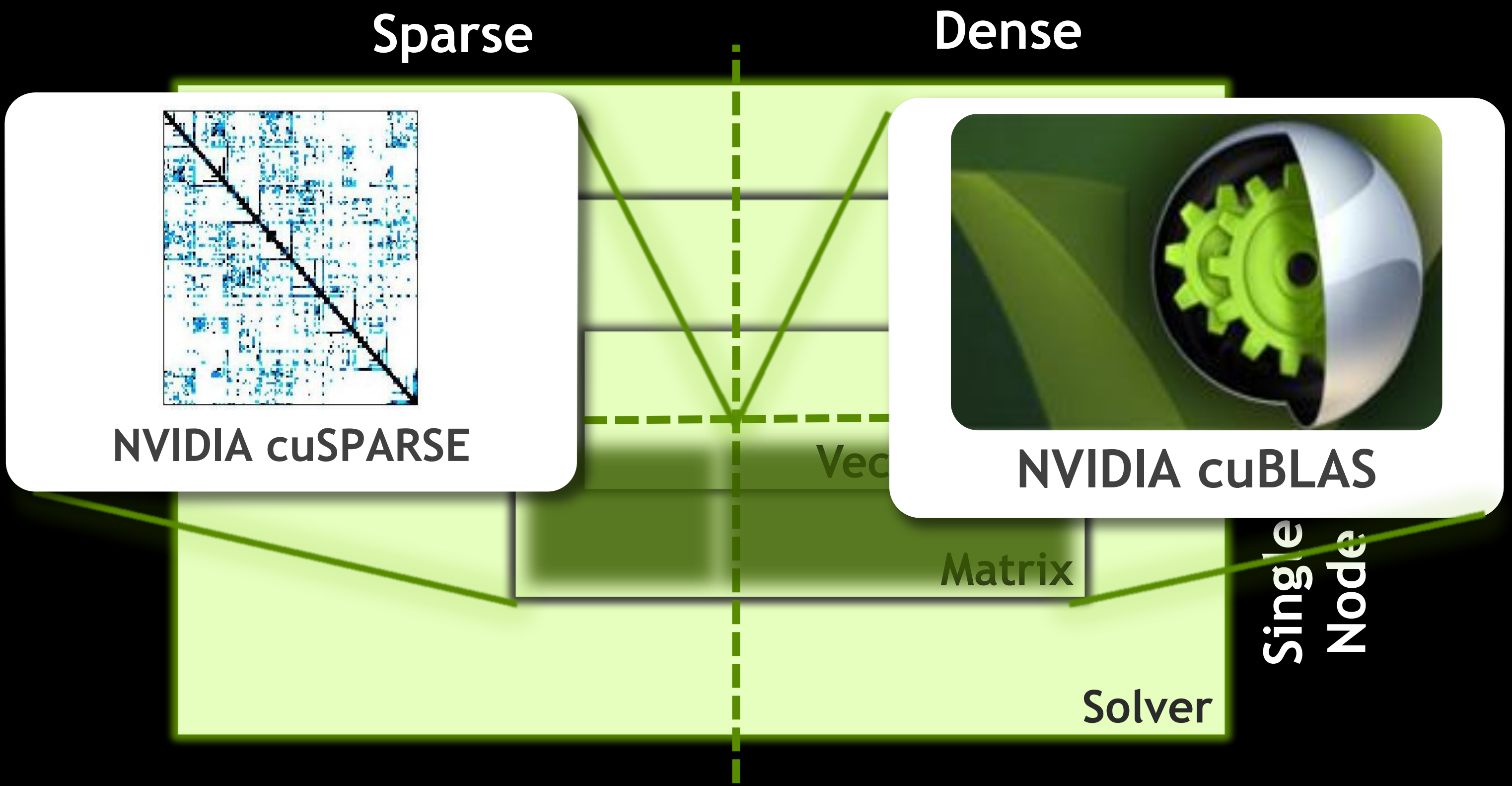
# NVIDIA CUDA Library Approach



- Provide basic building blocks
- Make them easy to use
- Make them fast

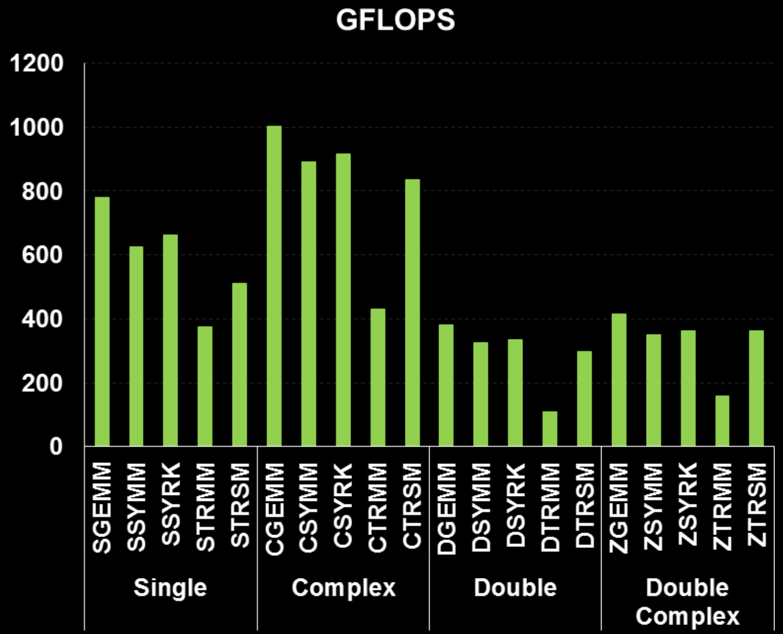
- Provides a quick path to GPU acceleration
- Enables ISVs to focus on their "secret sauce"
- Ideal for applications that use CPU libraries

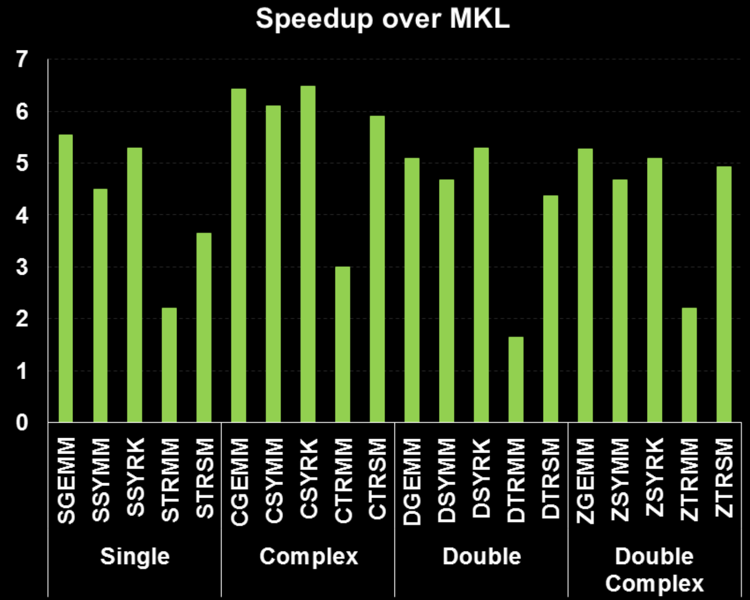
# NVIDIA's Foundation for LinAlg on GPUs



# cuBLAS Level 3 Performance

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

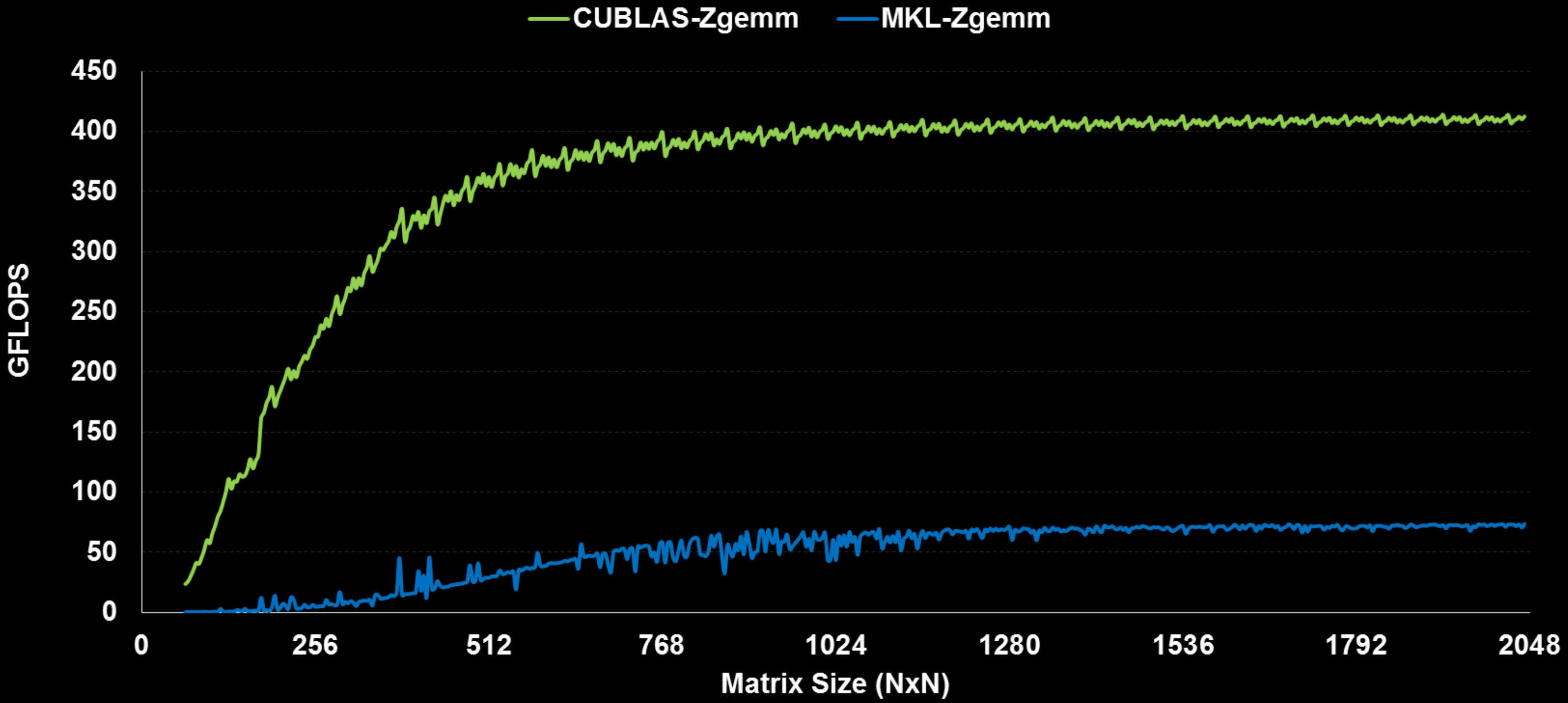



- 4Kx4K matrix size
- cuBLAS 4.1, Tesla M2090 (Fermi), ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

Performance may vary based on OS version and motherboard configuration

# ZGEMM Performance vs Intel MKL




- cuBLAS 4.1 on Tesla M2090, 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

# cuBLAS: Legacy and Version 2 Interface

- Legacy Interface
  - Convenient for quick port of legacy code
- Version 2 Interface
  - Reduces data transfer for complex algorithms
    - Return values on CPU or GPU
    - Scalar arguments passed by reference
  - Support for streams and multithreaded environment
  - Batching of key routines

# Version 2 Interface helps reducing memory transfers

Index transferred to CPU, CPU needs vector elements for scale factor

- Legacy Interface

```
idx = cublasIsamax(n, d_column, 1);
err = cublasSscal(n, 1./column[idx], row, 1);
```

# Version 2 Interface helps reducing memory transfers

- Legacy Interface

```
idx = cublasIsamax(n, d_column, 1);
err = cublasSscal(n, 1./d_column[idx], row, 1);
```

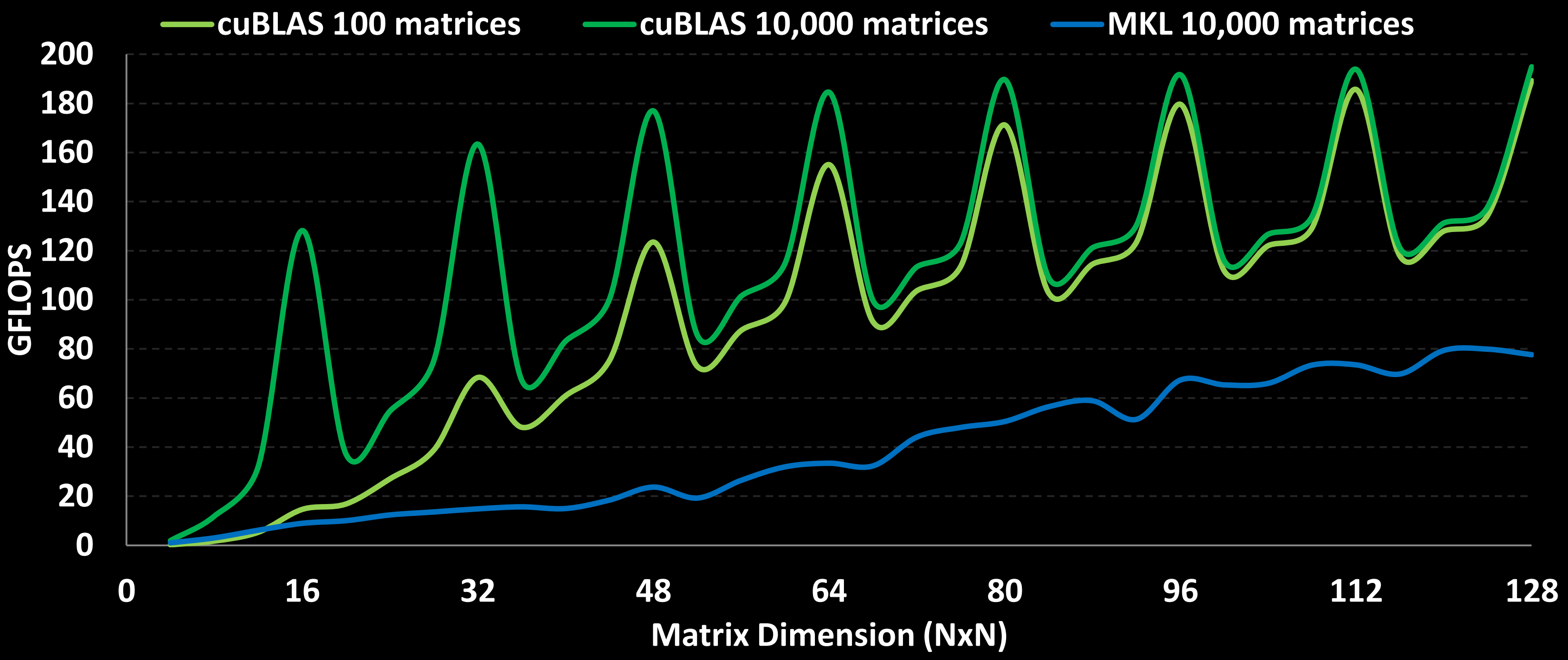
Index transferred to CPU, CPU needs vector elements for scale factor

- Version 2 Interface

```
err = cublasIsamax(handle, n, d_column, 1, d_maxIdx);
kernel<<< >>> (d_column, d_maxIdx, d_val);
err = cublasSscal(handle, n, d_val, d_row, 1);
```

All data remains on the GPU

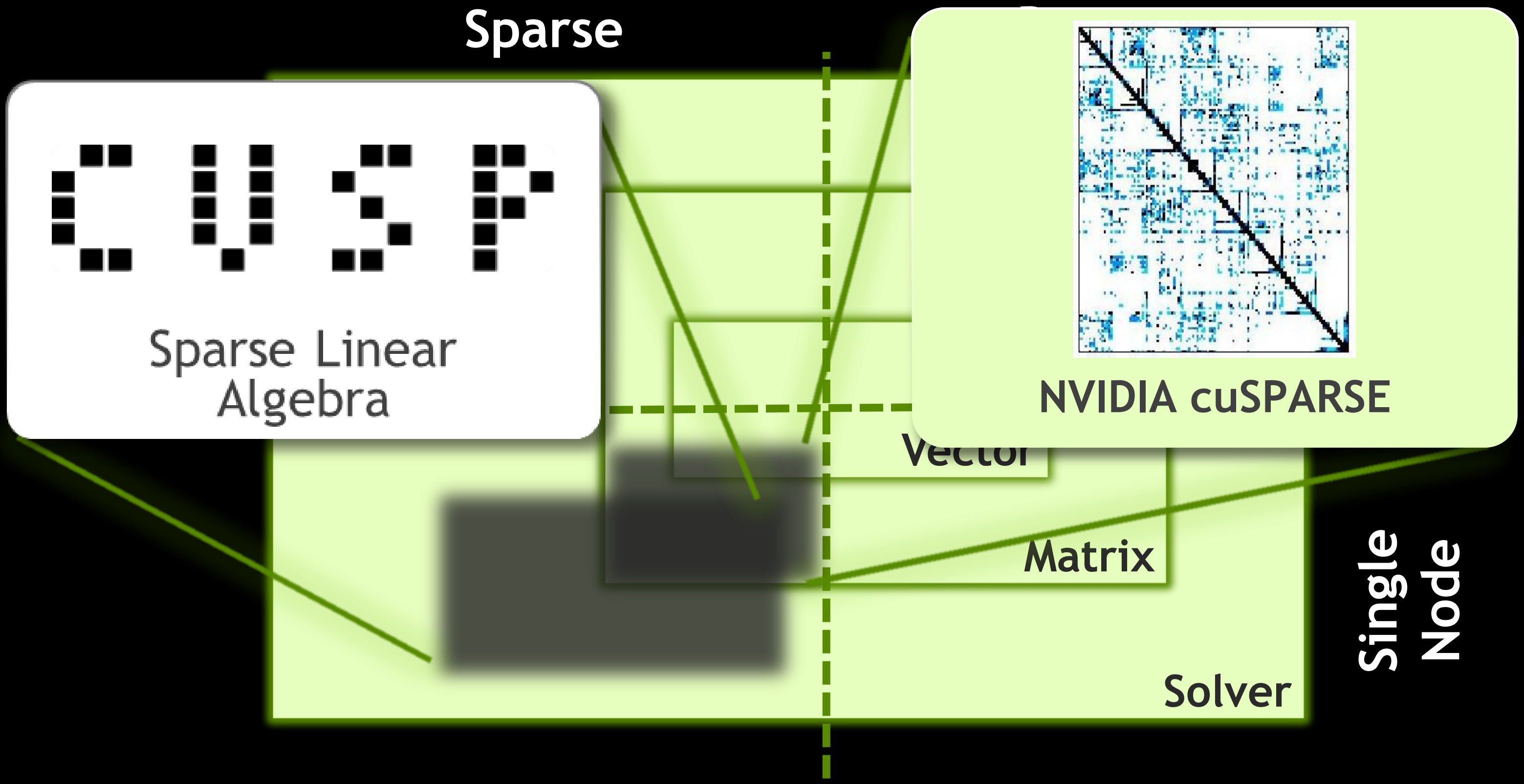
# cuBLAS Batched GEMM API improves performance on batches of small matrices




Performance may vary based on OS version and motherboard configuration

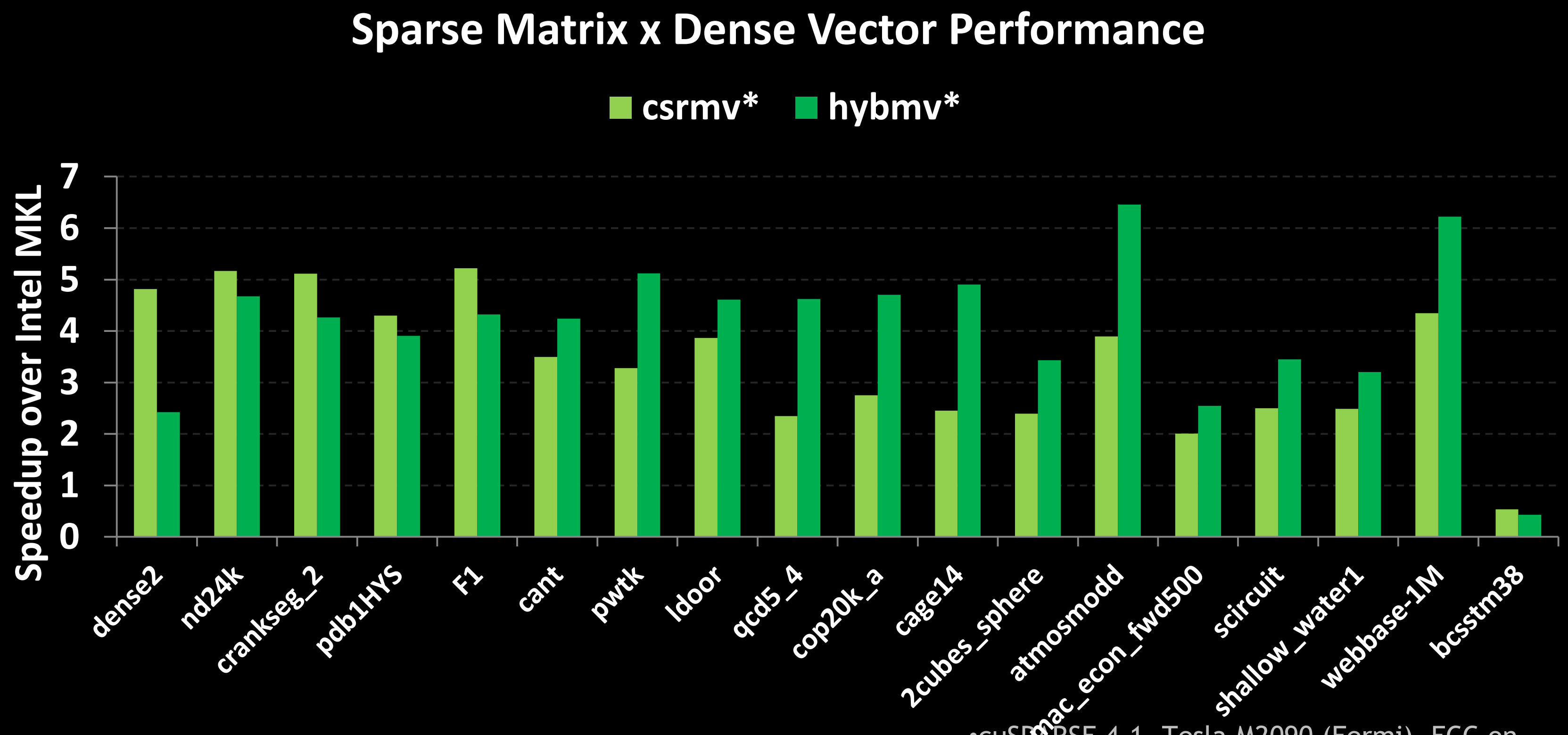
- cuBLAS 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

# The cuSPARSE - CUSP Relationship



GPU TECHNOLOGY CONFERENCE

# cuSPARSE is >6x Faster than Intel MKL

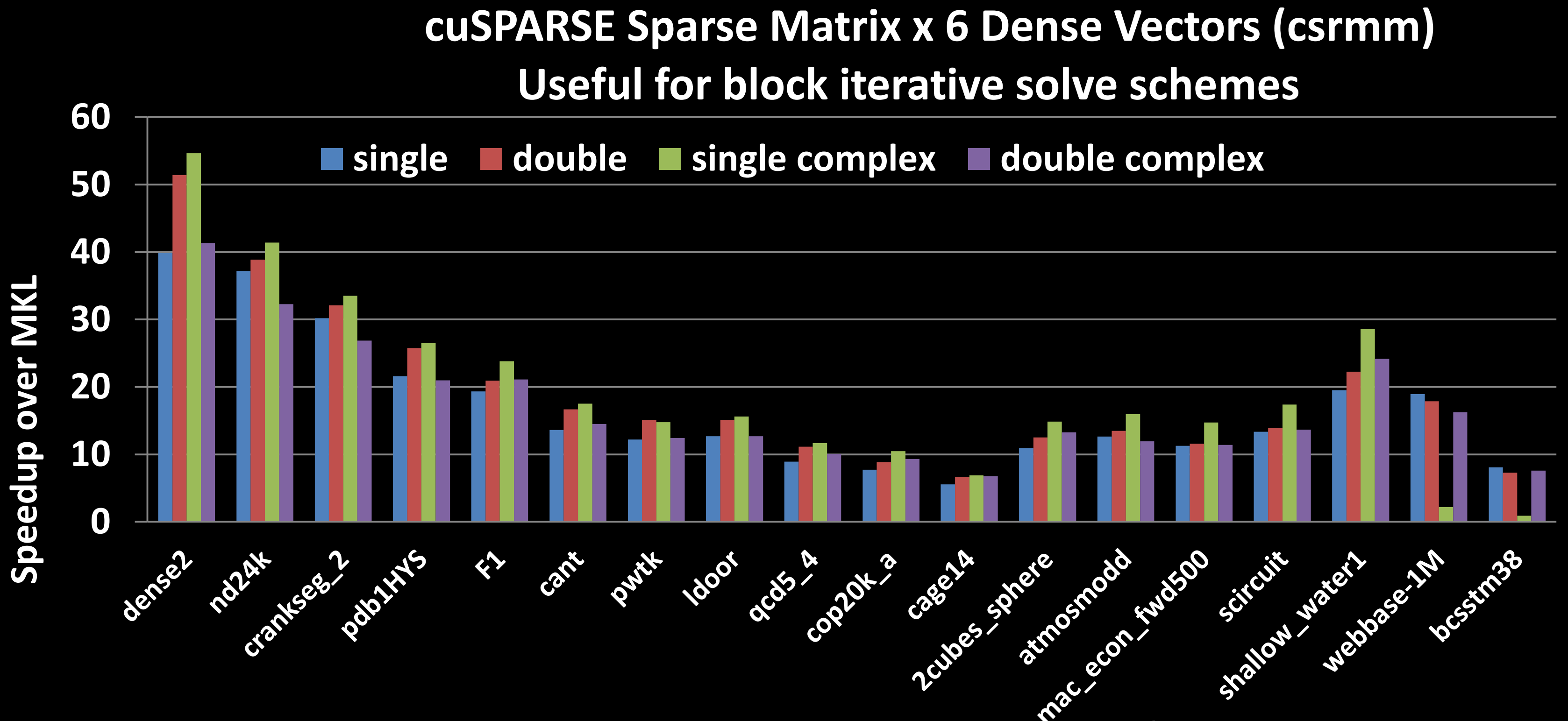



**Average speedup over single, double, single complex & double-complex*

- cuSPARSE 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

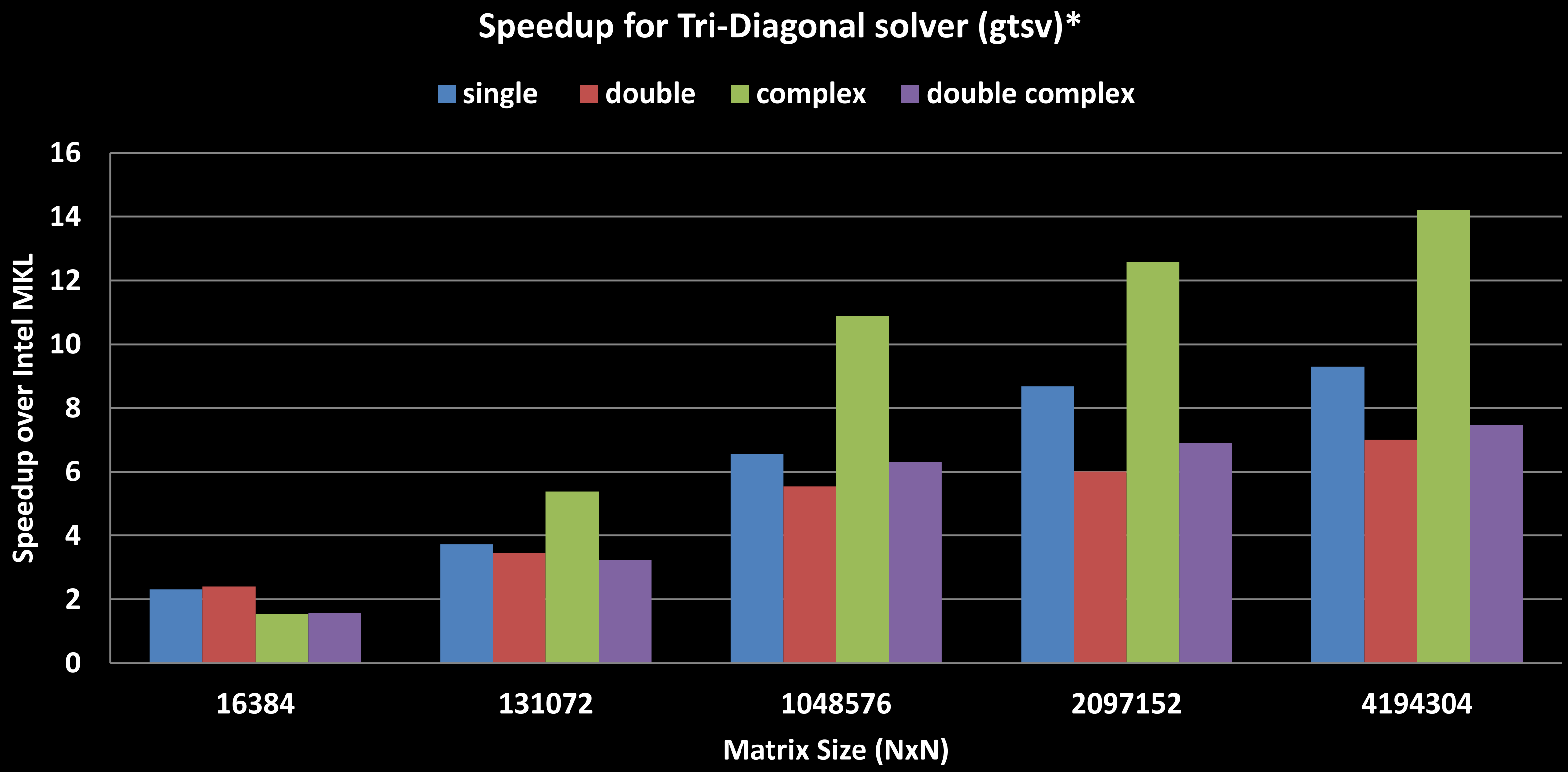
# Up to 40x faster with 6 CSR Vectors




- cuSPARSE 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

# Tri-diagonal solver performance vs. MKL

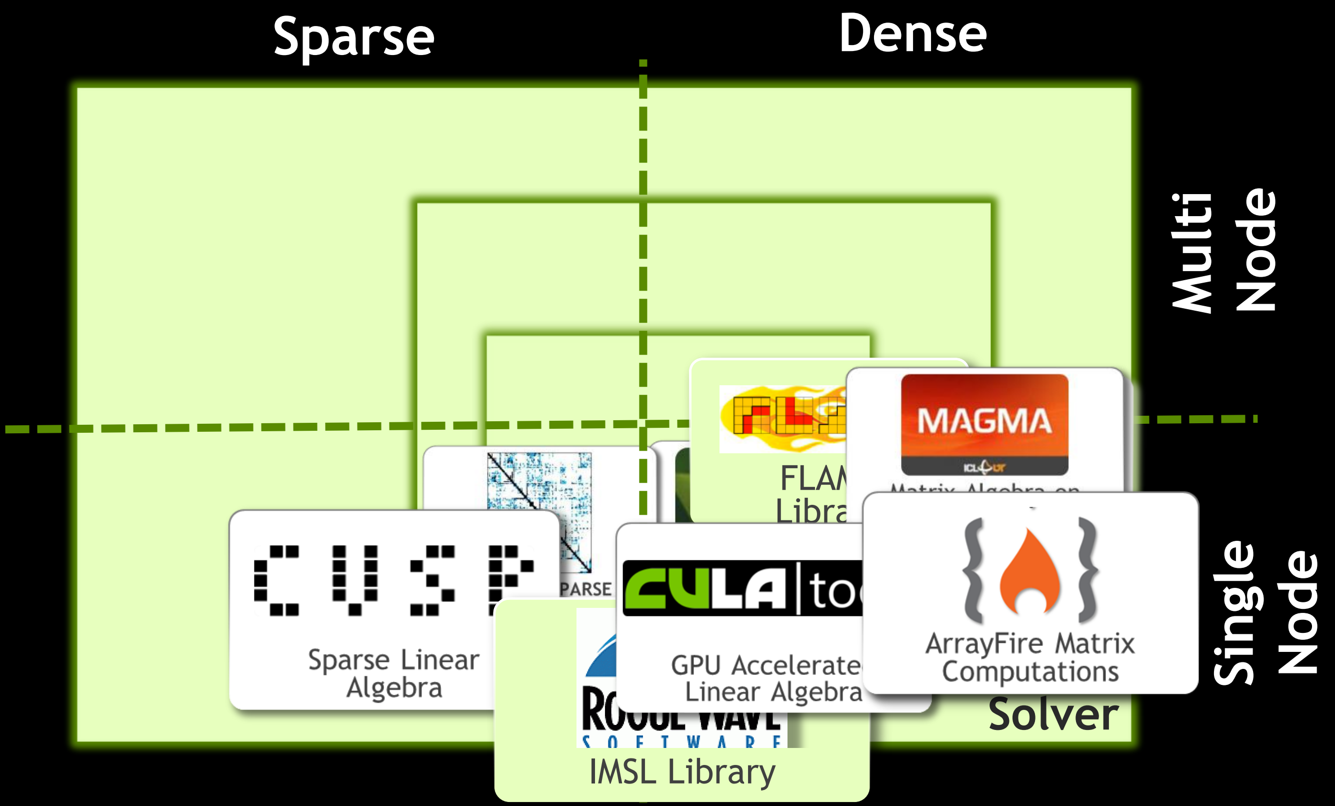



**Parallel GPU implementation does not include pivoting*

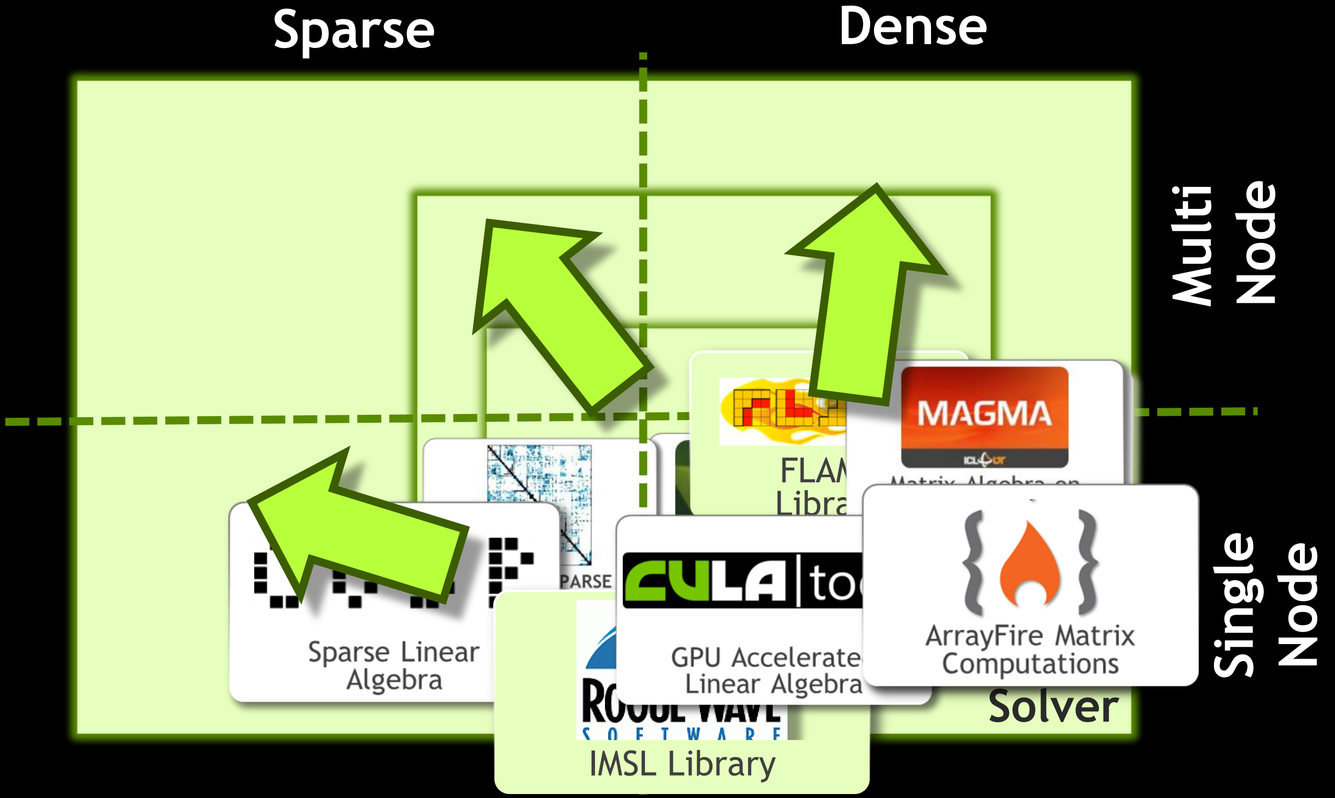
Performance may vary based on OS version and motherboard configuration

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

# Third Parties Extend the Building Blocks



# Third Parties Extend the Building Blocks



# 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, INFO)
```

CULA tools

GPU Accelerated Linear Algebra

S0307 (Wed)

- ArrayFire (LibJacket)
  - "Matlab-esque" interface
  - Array container object
  - Solvers, Factorizations, SVD, Eigensolvers

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

ArrayFire Matrix Computations

S0325 (Wed)

# 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, NRHS, A, LDA, INFO)
```

MAGMA

Matrix Algebra on GPU and Multicore

S0042 (Wed)

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

```
FLASH_LU_piv(A, p)
```

FLAME Library

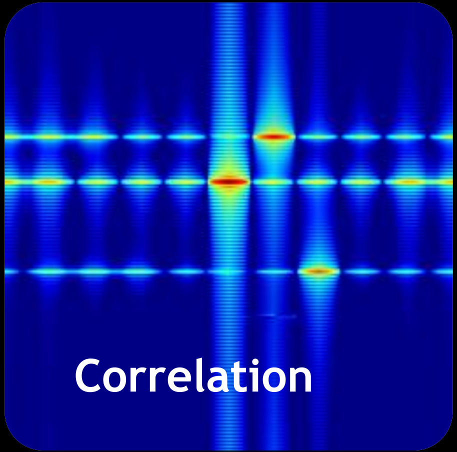
# Toolkits are increasingly supporting GPUs

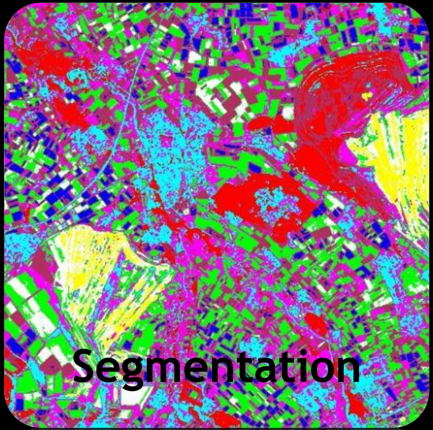
- PETSc
  - GPU support via extension to Vec and Mat classes
  - Partially dependent on CUSP
  - MPI parallel, GPU accelerated solvers

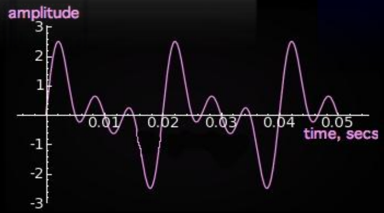
- Trilinos
  - GPU support in KOKKOS package
  - Used through vector class Tpetra
  - MPI parallel, GPU accelerated solvers

# Signal Processing

# Common Tasks in Signal Processing

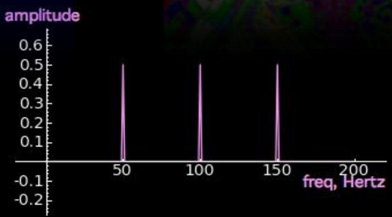




$$F(x) = \sum_{n=0}^{N-1} f(n) e^{-j2\pi(x\frac{n}{N})}$$

$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x) e^{j2\pi(x\frac{n}{N})}$$



GPU TECHNOLOGY CONFERENCE

GPU VSIPL

Vector Signal Image Processing

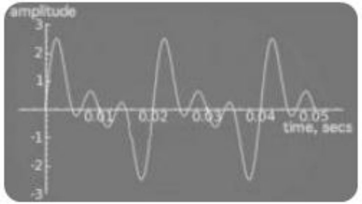
MATLAB

Parallel Computing Toolbox

ArrayFire Matrix Computations

GPULib

GPU Accelerated Data Analysis




NVIDIA cuFFT

NVIDIA NPP

# Libraries for GPU Accelerated Signal Processing

# Basic concepts of cuFFT

- Interface modeled after FFTW
  - Simple migration from CPU to GPU

```
fftw_plan_dft2_2d => cufftPlan2d
```

- "Plan" describes data layout, transformation strategy
  - Depends on dimensionality, layout, type of transform
  - Independent of actual data, direction of transform
  - Reusable for multiple transforms

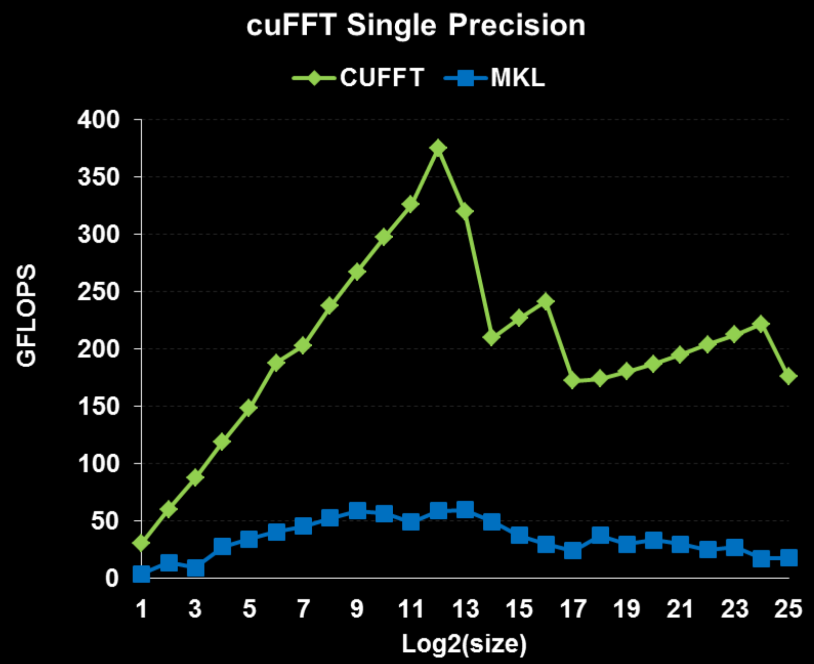
- Execution of plan
  - Depends on transform direction, data

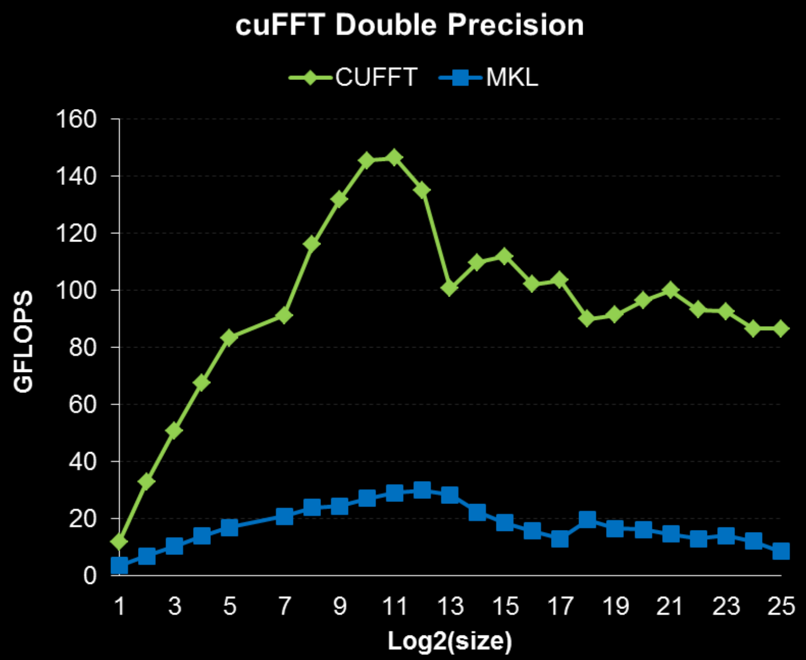
```
cufftExecC2C(plan, d_data, d_data, CUFFT_FORWARD)
```

# Efficient use of cuFFT

- Perform multiple transforms with the same plan
  - Use e.g. in forward/inverse transform for convolution, transform at each simulation timestep, etc.
- Transform in streams
  - cufft functions do not take a stream argument
  - Associate a plan with a stream via `cufftSetStream(plan, stream)`
- Batch transforms
  - Concurrent execution of multiple identical transforms
  - Support for 1D, 2D and 3D transforms

# High 1D transform performance is key to efficient 2D and 3D transforms

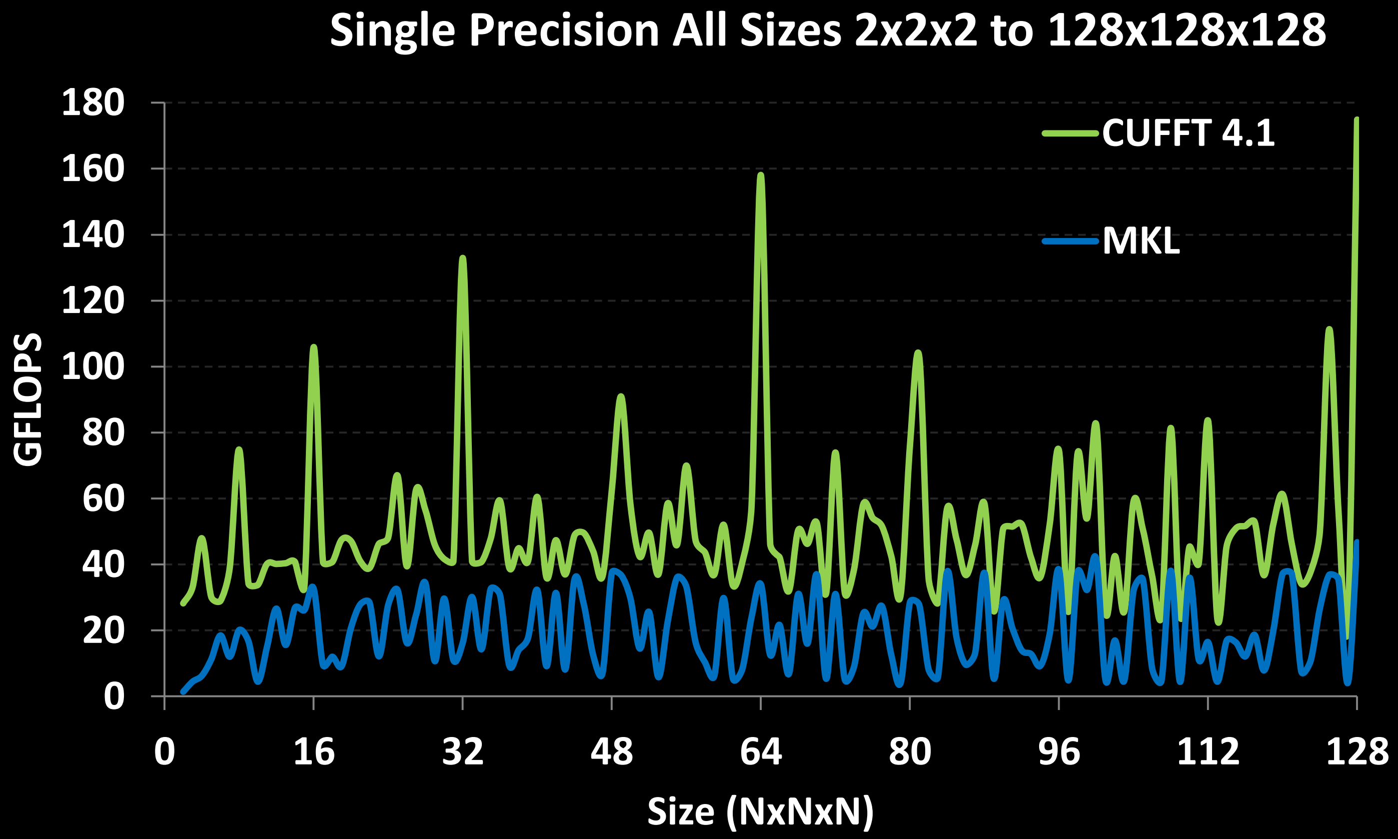



Performance may vary based on OS version and motherboard configuration

- 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

# Optimized 3D transforms

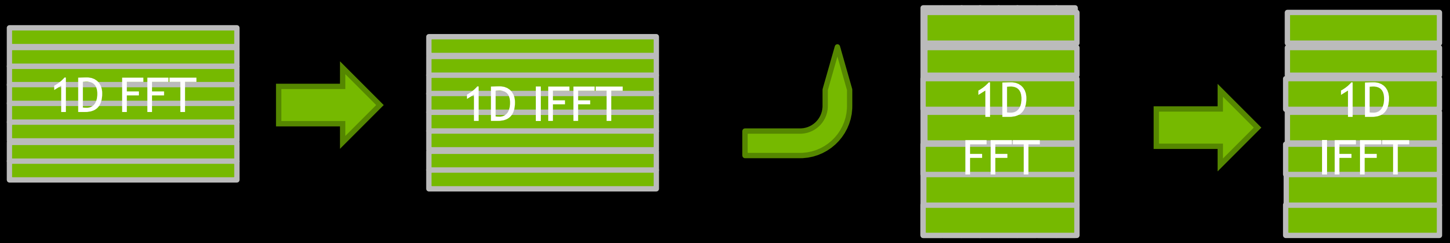



Performance may vary based on OS version and motherboard configuration

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

# cufftPlanMany: Transformation on complex data layouts

- Example: Range-Doppler compression




- No need for explicit transpose with cufftPlanMany
  - Independent input and output strides/internal dimension

```
cufftPlanMany( cufftHandle *plan, int rank, int *n,
    int *inembed, int istride, int idist, // input layout
    int *onembed, int ostride, int odist, // output layout
    cufftType type, int batch)
```

# Basic concepts of NPP

- Collection of high-performance GPU processing
  - Initial focus on Image, Video and Signal processing
    - Growth into other domains expected
  - Support for multi-channel integer and float data

- C API => name disambiguates between data types, flavor

**`nppiAdd_32f_C1R (…)`**

  - "Add" two single channel ("C1") 32-bit float ("32f") images, possibly masked by a region of interest ("R")

# NPP features a large set of functions

- Arithmetic and Logical Operations
  - Add, mul, clamp, ..
- Threshold and Compare
- Geometric transformations
  - Rotate, Warp, Perspective transformations
  - Various interpolations
- Compression
  - jpeg de/compression
- Image processing
  - Filter, histogram, statistics

NVIDIA NPP

S0404 (Tue)

# GPU-VSIPL - Vector Image Signal Processing Library

- Open Industry Standard for Signal Processing
- Focus on embedded space, but support for GPU, CPU, ..
- Separate memory spaces integral part of API
- Support for single precision float, fft, matrix factorization
- GPU enabled version from Georgia Tech
  - Lite and Core API

```
vsip_ccfftmip_f(d->fft_plan_fast,d->z_cmview);
```

- VSIPL++ by Mentor Graphics

  S0620 (Tue)

# Multi-GPU FFT

- Many problems too large for single GPU
- Careful about data layout
  - perform 1D transforms on a single GPU if possible

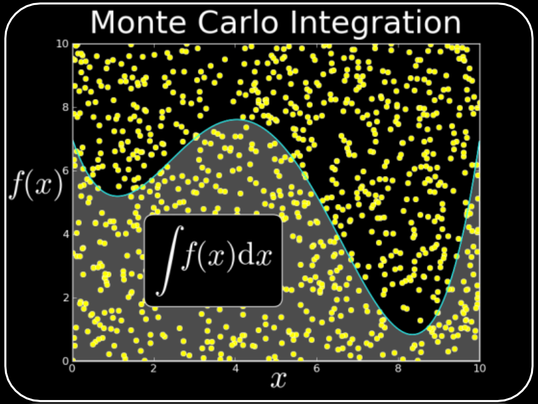
- Minimize data transfer cost (GPU direct)
  - Multi-Dimensional distributed memory FFT
    requires all-to-all

- Various presentations here at GTC:
  - Akira Nukada, Tokyo Institute of Technology, S0290 (Wed)
  - Filippo Spiga, ICHEC, S0220 (Thu)

# 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
  - Host: Ideal when generating large batches of RNGs on GPU
  - Device: Ideal when RNGs need to be generated inside a kernel



# cuRAND: Host vs Device API

- Host API

```
#include “curand.h”
curandCreateGenarator(&gen, CURAND_RNG_PSEUDO_DEFAULT);
curandGenerateUniform(gen, d_data, n);
```

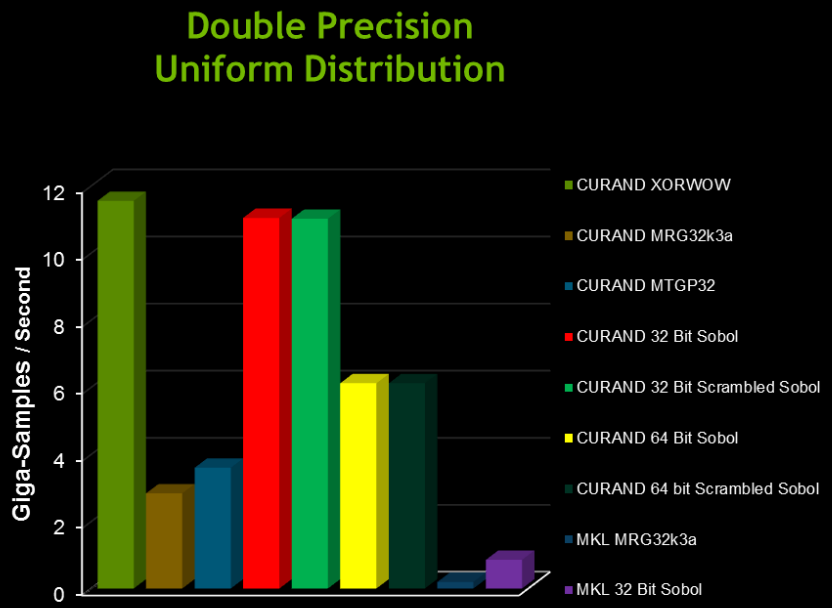
Generate set of random numbers at once

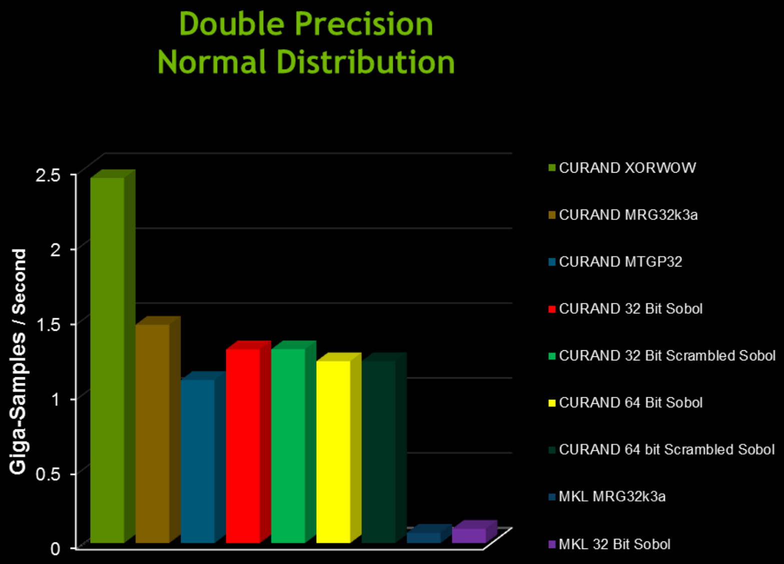
- Device API

```
#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 compared to Intel MKL




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

# Next steps..

GPU TECHNOLOGY CONFERENCE

# Thurst: STL-like CUDA Template Library

- Device and host vector class

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

- Iterators

```
thrust::fill(D.begin(), D.begin()+0, 42.f);
float* raw_ptr = thrust::raw_pointer_cast(D);
```

- Algorithms
  - Sort, reduce, transformation, scan, ..

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

Thrust

C++ STL Features for CUDA

S0602 (Tue),
S0653 (Thu)

# Using Libraries with OpenACC

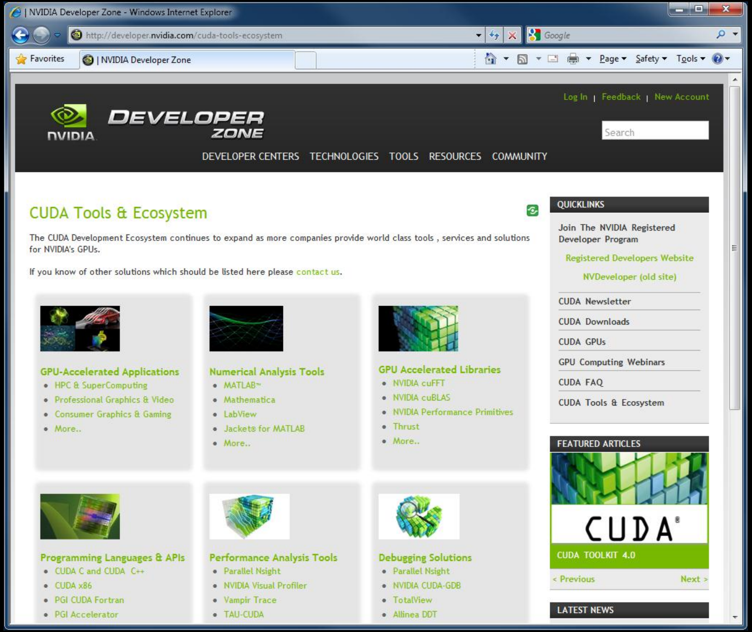
- Libraries often require explicit device data
- Device data transparent in OpenACC
- Inform OpenAcc about device variables with **`data deviceptr`** clause

```
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




- Attend GTC library talks

GPU TECHNOLOGY CONFERENCE

# Summary

- CUDA libraries offer a broad range of high-performance functions
- 3rd party libraries provide extended functionality
- By sticking to commonly used interfaces, legacy code can be moved quickly to GPUs ("drop-in")
- Libraries enable developers to focus on their core IP
- Libraries interact well with other parts of the CUDA ecosystem

# Thank you