Popular CUDA Packages

Krishnan Suresh ("Suresh")
suresh@engr.wisc.edu

Associate Professor
Mechanical Engineering

Take-Home Message

- Don’t reinvent the wheel!
- Minimize custom Kernels

Conjugate Gradient

THE UNIVERSITY of WISCONSIN MADISON

- Solve Ax = b via CG (Matlab)

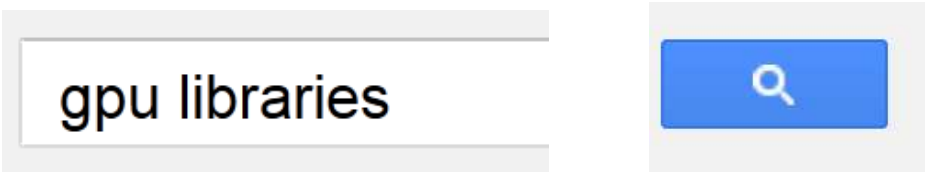
```
function [x] = conjgrad(A,b,x)
    r=b-A*x;
    p=r;
    rsold=r'*r;

    for i=1:size(A)(1)
        Ap=A*p;
        alpha=rsold/(p'*Ap);
        x=x+alpha*p;
        r=r-alpha*Ap;
        rsnew=r'*r;
        if sqrt(rsnew)<1e-10
              break;
        end
        p=r+rsnew/rsold*p;
        rsold=rsnew;
    end
end
```

GPU algorithms:

- Dot-product: Use CUBLAS
- Ax: Use CUSPARSE
- ax+b: Use CUBLAS

CUDA Libraries & Packages



NVIDIA DEVELOPER ZONE

Home

GPU-ACCELERATED LIBRARIES

Adding GPU-acceleration to your application can be as easy as simply calling a library function. Check out the extensive list libraries below. If you would like other libraries added to this list please contact us.

NVIDIA cuFFT
NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster, without having to develop your own custom GPU FFT implementation.

NVIDIA cuBLAS
NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS.

CULA | tools

CULA Tools
GPU-accelerated linear algebra library by EM Photonics, that utilizes CUDA to dramatically improve the computation speed of sophisticated mathematics.

MAGMA

MAGMA
A collection of next gen linear algebra routines. Designed for heterogeneous GPU-based architectures. Supports current LAPACK and BLAS standards.

IMSL Fortran Numerical Library
Developed by RogueWave, a comprehensive set of mathematical and statistical functions that offloads work to GPUs.

NVIDIA cuSPARSE
NVIDIA CUDA Sparse (cuSPARSE) Matric library provides a collection of basic linear algebra subroutines used for sparse matrices that delivers over 8x performance boost.

CUDA Libraries & Packages

1. CUBLAS: Dense Linear Algebra
2. Thrust: Parallel sort, …
3. CuSparse: Sparse Linear Algebra Package
4. Jacket: Matlab Wrapper
5. CULA: Dense and sparse linear algebra
6. MAGMA: Multicore linear algebra
7. CUFFT: Fast Fourier Transform
8. …

CUDA Libraries & Packages

1. CUBLAS: Dense Linear Algebra
2. Thrust: Parallel sort, …
3. CuSparse: Sparse Linear Algebra Package
4. Jacket: Matlab Wrapper
5. CULA: Dense and sparse linear algebra
6. MAGMA: Multicore linear algebra
7. CUFFT: Fast Fourier Transform
8. …

CUBLAS

THE UNIVERSITY of WISCONSIN MADISON

- CUDA implementation of BLAS (Basic Linear Algebra Subprograms)
  - Vector, vector (Level-1)
  - Matrix, vector (Level-2)
  - Matrix, matrix (Level-3)
- Precisions
  - Single: real & complex
  - Double: real & complex (not all functions)
- No kernel calls, shared memory, etc

CUBLAS Library Usage

- **No additional downloads needed**
  - **cublas.lib (in CUDA SDK)**
  - **Add cublas.lib to linker**
  - **#include cublas.h**

CUBLAS Code Structure

1. Initialize CUBLAS: cublasInit()
2. Create CPU memory and data
3. Create GPU memory: cublasAlloc(…)
4. Copy from CPU to GPU : cublasSetVector(…)
5. Operate on GPU : cublasSgemm(…)
6. Check for CUBLAS error : cublasGetError()
7. Copy from GPU to CPU : cublasGetVector(…)
8. Verify results
9. Free GPU memory : cublasFree(…)
10. Shut down CUBLAS : cublasShutDown()

CUBLAS BLAS-1 Functions:

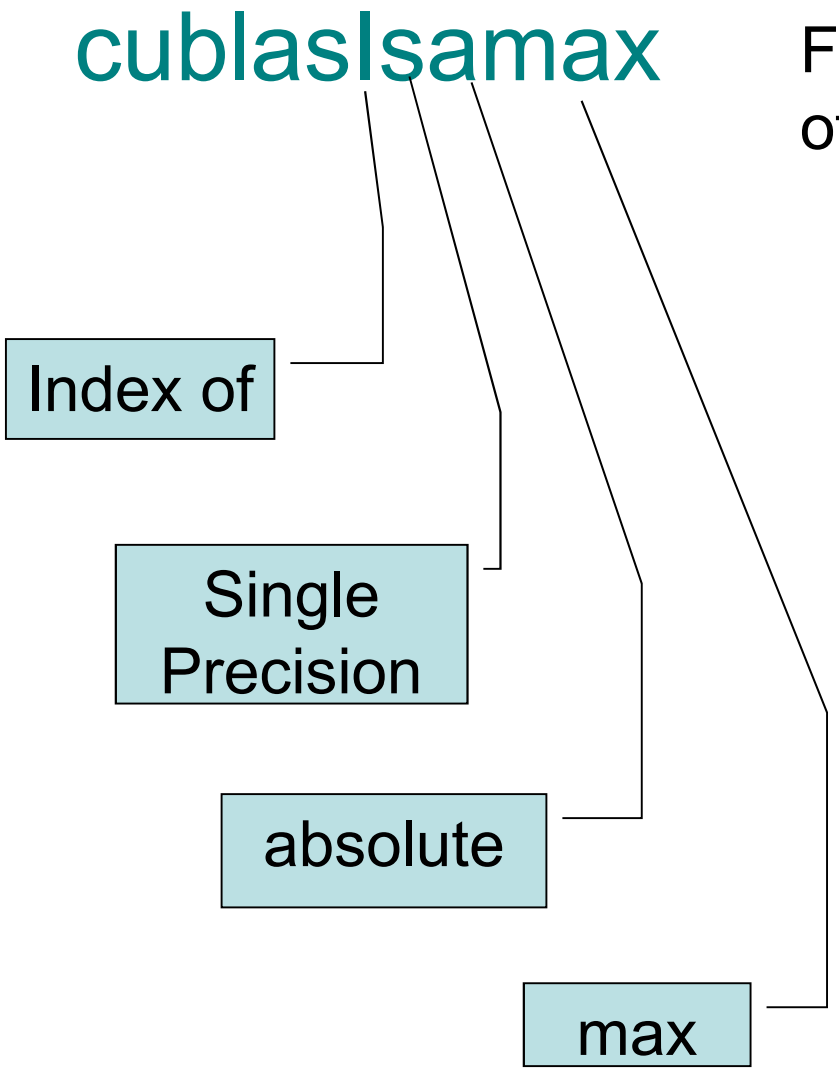
Vector-vector operations

Single-Precision BLAS1 Functions

The single-precision BLAS1 functions are as follows:

- "Function cublasIsamax()" on page 16
- "Function cublasIsamin()" on page 17
- "Function cublasSasum()" on page 18
- "Function cublasSaxpy()" on page 18
- "Function cublasScopy()" on page 19
- "Function cublasSdot()" on page 20
- "Function cublasSnrm2()" on page 21
- "Function cublasSrot()" on page 22
- "Function cublasSrotg()" on page 23
- "Function cublasSrotm()" on page 24
- "Function cublasSrotmg()" on page 25
- "Function cublasSscal()" on page 26
- "Function cublasSswap()" on page 27

CU(BLAS) Naming Convention



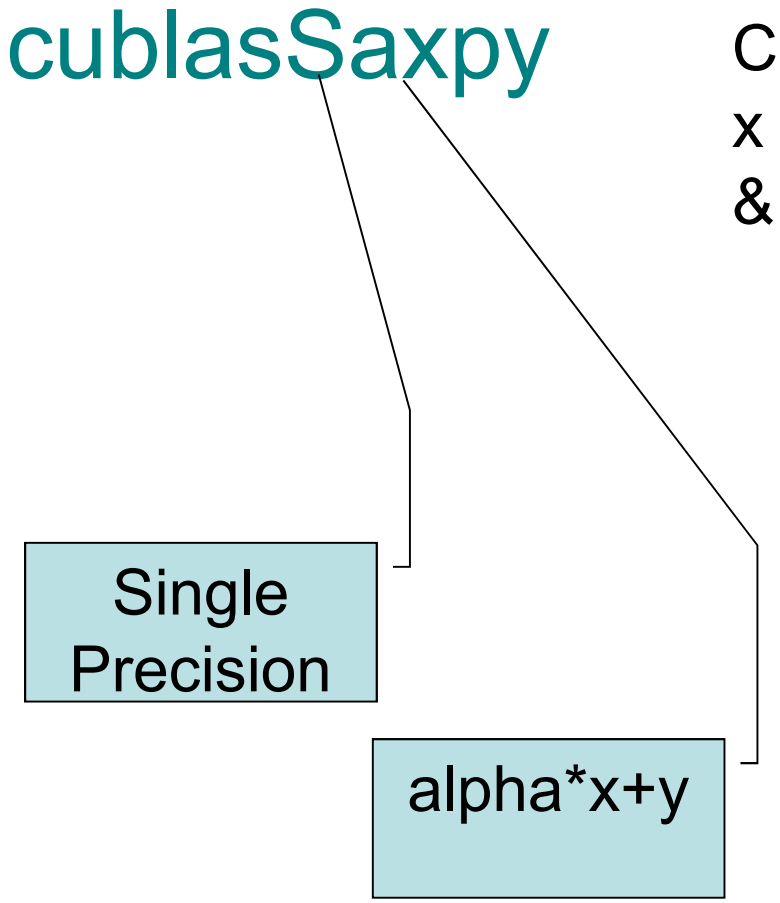
Find the index of the absolute max of a vector of single precision reals

cublasIdamax

cublasIcamax

cublasIzamax

CU(BLAS) Naming Convention



Compute alpha*x+y where
x &y are single precision reals
& alpha is a scalar

cublasDaxpy

CUBLAS Example-1 (CPU)

$$a = x^T y$$

```
/// CPU Allocation and computation
h_x = (float*)malloc(VECTORSIZE * sizeof(float));
h_y = (float*)malloc(VECTORSIZE * sizeof(float));
cpu_dot = 0.0f;
for (i =0; i < VECTORSIZE; i++) {
  h_x[i] = 1.0f*rand()/RAND_MAX;
  h_y[i] = 1.0f*rand()/RAND_MAX;
  cpu_dot += h_x[i]*h_y[i];
}
```

CUBLAS Example-1 (GPU)

$$a = x^T y$$

```
cublasInit();
cublasAlloc(VECTORSIZE, sizeof(float), (void**)&d_x);
cublasAlloc(VECTORSIZE, sizeof(float), (void**)&d_y);
cublasSetVector(VECTORSIZE, sizeof(float), h_x, 1, d_x, 1);
cublasSetVector(VECTORSIZE, sizeof(float), h_y, 1, d_y, 1);

gpu_dot = cublasSdot(VECTORSIZE,d_x,1,d_y,1);

if ((fabs(gpu_dot-cpu_dot) >TOL))
  printf("Dot Product Failure!\
else
  printf("Dot Product Success!\n");

free(h_x); free(h_y);
cublasFree(d_x);  cublasFree(d_y);

cublasShutdown();
```

Increment of 1

- No kernel calls
- No memory mgmt.

CUBLAS Example-2 (CPU)

THE UNIVERSITY of WISCONSIN MADISON

$$z = \alpha x + y$$

```
/// CPU Allocation and computation
alpha = 1.0f*rand()/RAND_MAX;
h_x = (float*)malloc(VECTORSIZE * sizeof(float));
h_y = (float*)malloc(VECTORSIZE * sizeof(float));
h_ref = (float*)malloc(VECTORSIZE * sizeof(float));
for (i =0; i < VECTORSIZE; i++) {
  h_x[i] = 1.0f*rand()/RAND_MAX;
  h_y[i] = 1.0f*rand()/RAND_MAX;
  h_ref[i] = alpha*h_x[i]+h_y[i];
}
```

CUBLAS Example-2 (GPU)

$$z = \alpha x + y$$

```
cublasAlloc(VECTORSIZE, sizeof(float), (void**)&d_x);
cublasAlloc(VECTORSIZE, sizeof(float), (void**)&d_y);
cublasSetVector(VECTORSIZE, sizeof(float), h_x, 1, d_x, 1);
cublasSetVector(VECTORSIZE, sizeof(float), h_y, 1, d_y, 1);


cublasSaxpy(VECTORSIZE,alpha,d_x,1,d_y,1);


cublasGetVector(VECTORSIZE, sizeof(float), d_y, 1, h_y, 1);
for (i =0; i < VECTORSIZE; i++) {
  if ((fabs(h_ref[i]-h_y[i]) >TOL)) {
    printf("SAXPY Failure!\n");
    break;
  }
}
```

Output stored in d_y

CUBLAS BLAS-2 Functions:

Matrix-Vector Operations

Single-Precision BLAS2 Functions

The single-precision BLAS2 functions are as follows:

- "Function cublasSgbmv()" on page 57
- "Function cublasSgemv()" on page 59
- "Function cublasSger()" on page 60
- "Function cublasSsbmv()" on page 61
- "Function cublasSspmv()" on page 63
- "Function cublasSspr()" on page 64
- "Function cublasSspr2()" on page 65
- "Function cublasSsymv()" on page 66
- "Function cublasSsyr()" on page 67
- "Function cublasSsyr2()" on page 68
- "Function cublasStbmv()" on page 70
- "Function cublasStbsv()" on page 71
- "Function cublasStpmv()" on page 73
- "Function cublasStpsv()" on page 74
- "Function cublasStrmv()" on page 75
- "Function cublasStrsv()" on page 77

$$z = \alpha Ax + \beta y$$

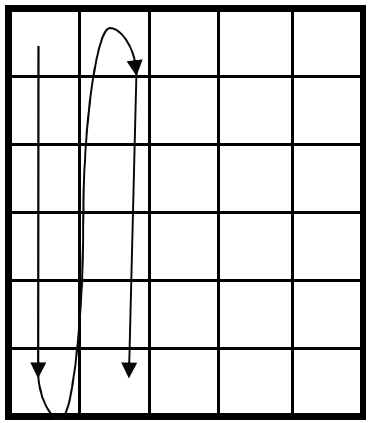
$$A : symmetric\ banded$$

$$x = \alpha A^{-1} y$$

$$A = Upper(or\ Lower)$$

CUBLAS: Caveats

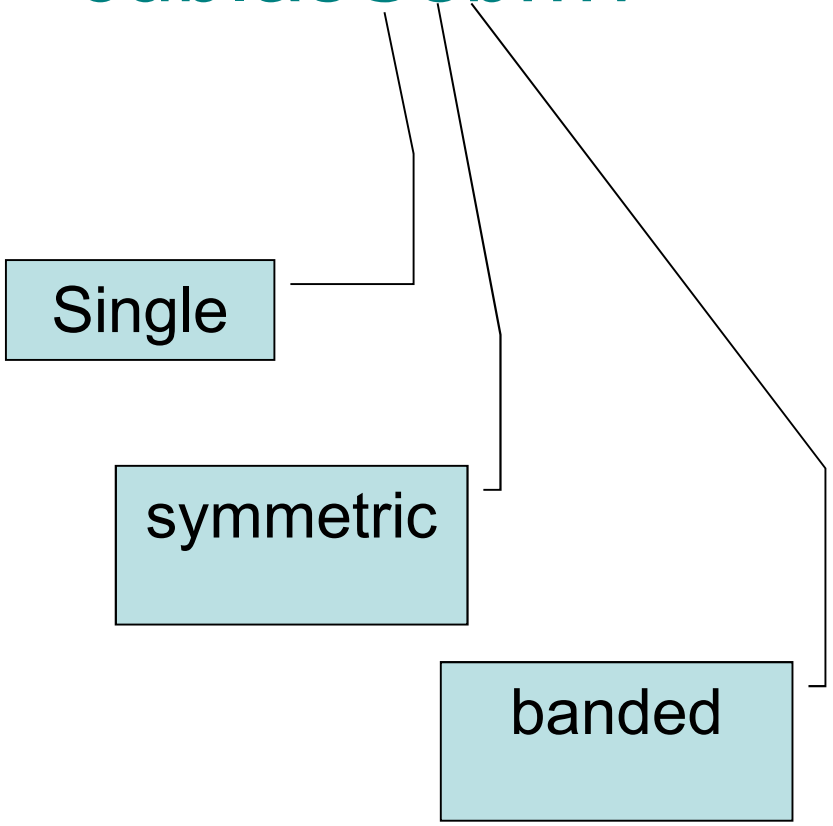
- Solves Ax = b only for Upper/Lower A
- Limited class of sparse matrices
- Column format & 1-indexing (Fortran style)



- C: row format & 0-indexing; use macros

CU(BLAS) Naming Convention

cublasSsbmv



$$z = \alpha Ax + \beta y$$

X	x	x		
x	x	x	x	
x	x	x	x	x
	x	x	x	x
		x	x	x

Example

$$z = \alpha Ax + \beta y$$

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \dots & \\ & & \dots & \dots & -1 \\ & & & -1 & 2 \end{bmatrix}_{(N,N)}$$

Symmetric-Banded

#Super-Diagonals = 1

It is sufficient to store

$$\begin{bmatrix} 2 & -1 & & & \\ & 2 & -1 & & \\ & & 2 & \dots & \\ & & & \dots & -1 \\ & & & & 2 \end{bmatrix}_{(N,N)}$$

Stored as

$$h\_A = \begin{bmatrix} X & -1 & -1 & \dots & -1 \\ 2 & 2 & 2 & \dots & 2 \end{bmatrix}_{(2,N)}$$

CUBLAS Example-3 (CPU)

$$z = \alpha Ax + \beta y$$

$$h\_A = \begin{bmatrix} X & -1 & -1 & \dots & -1 \\ 2 & 2 & 2 & \dots & 2 \end{bmatrix}_{(2,N)}$$

Macro for 0-indexing in C

```
#define IDX2C(i,j,ld) (((j)*(ld))+(i))

h_A = (float*)malloc(2*VECTORSIZE * sizeof(float));

for (j =0; j < VECTORSIZE;j++) {
  h_A[IDX2C(0,j,2)]= -1; //Super-diagonal,(0,0) is never used
  h_A[IDX2C(1,j,2)]= 2; // Diagonal
}
```

$$h\_A: \begin{bmatrix} X \\ 2 \\ -1 \\ 2 \\ -1 \\ \dots \end{bmatrix}$$

CUBLAS Example-3 (CPU)

$$\begin{Bmatrix} z_1 \\ z_2 \\ z_3 \\ \dots \\ z_N \end{Bmatrix} = \alpha \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \dots & \\ & & \dots & \dots & -1 \\ & & & -1 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_N \end{Bmatrix} + \beta \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_N \end{Bmatrix}$$

$$h\_A = \begin{bmatrix} X & -1 & -1 & \dots & -1 \\ 2 & 2 & 2 & \dots & 2 \end{bmatrix}_{(2,N)}$$

```
for (i =0; i < VECTORSIZE; i++) {
  if (i == 0)
    h_z[i] = alpha*h_A[IDX2C(1,0,2)]*h_x[i] +
             alpha*h_A[IDX2C(0,1,2)]*h_x[i+1] + beta*h_y[i];
  else if (i < VECTORSIZE-1)
    h_z[i] = alpha*h_A[IDX2C(0,i,2)]*h_x[i-1] +
             alpha*h_A[IDX2C(1,i,2)]*h_x[i] +
             alpha*h_A[IDX2C(0,i+1,2)]*h_x[i+1] + beta*h_y[i];
  else
    h_z[i] = alpha*h_A[IDX2C(0,i,2)]*h_x[i-1] +
             alpha*h_A[IDX2C(1,i,2)]*h_x[i] + beta*h_y[i];
}
```

CUBLAS Example-3 (GPU)

$$z = \alpha Ax + \beta y$$

$$h\_A = \begin{bmatrix} X & -1 & -1 & \dots & -1 \\ 2 & 2 & 2 & \dots & 2 \end{bmatrix}_{(2,N)}$$

```
cublasAlloc(2*VECTORSIZE, sizeof(float), (void**)&d_A);
cublasSetMatrix(2,VECTORSIZE,sizeof(float),h_A,2,d_A,2);
```

#Rows

```
cublasSsbmv('U',VECTORSIZE,1,alpha,d_A,2,d_x,1,beta,d_y,1);
```

Upper diagonal

#Rows

```
cublasGetVector(VECTORSIZE, sizeof(float), d_y, 1, h_zFromGPU, 1);
```

CUBLAS Optimal Usage

1. Copy from CPU to GPU : cublasSet …(…)
2. Operate on GPU
   - Operation 1
   - Operation 2
   - …
   - Operation n
3. Copy from GPU to CPU : cublasGet…(…)

CUBLAS BLAS-3 Functions:

Matrix-Matrix Operations

The single-precision BLAS3 functions are listed below:

- "Function cublasSgemm()" on page 88
- "Function cublasSsymm()" on page 90
- "Function cublasSsyrk()" on page 92
- "Function cublasSsyr2k()" on page 93
- "Function cublasStrmm()" on page 95
- "Function cublasStrsm()" on page 97

$$C = \alpha AB + \beta C$$

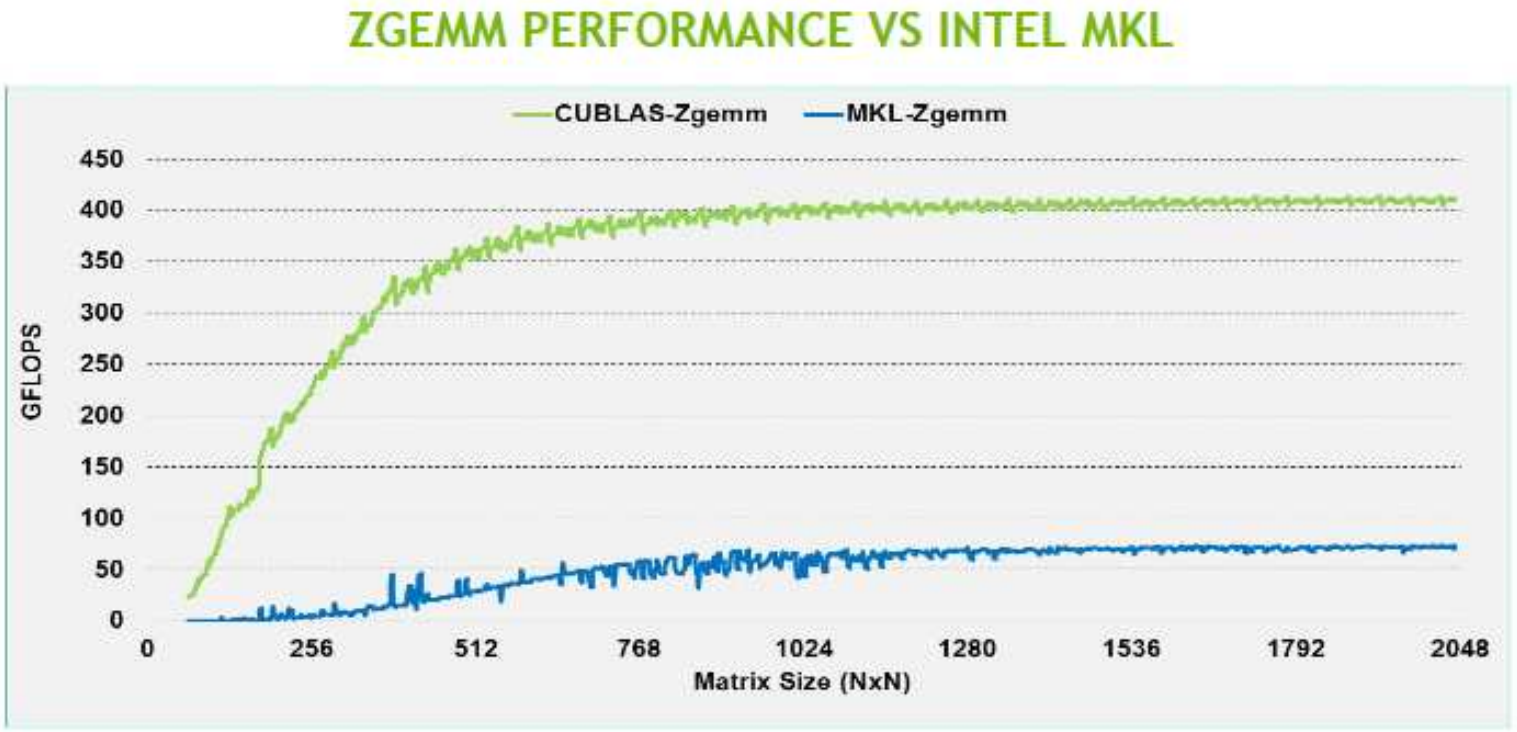
$$X = \alpha A^{-1} B$$

$$A = Upper(or\ Lower)$$

CUBLAS Performance

THE UNIVERSITY of WISCONSIN MADISON

UP TO 1 TFLOPS SUSTAINED PERFORMANCE AND >6X SPEEDUP OVER INTEL MKL



CUDA Libraries & Packages

1. CUBLAS: Dense Linear Algebra
2. Thrust: Parallel sort, …
3. CuSparse: Sparse Linear Algebra Package
4. Jacket: Matlab Wrapper
5. CULA: Dense and sparse linear algebra
6. MAGMA: Multicore linear algebra
7. CUFFT: Fast Fourier Transform
8. …

Thrust

- C++ Template Library using CUDA
- Vector containers:
  - host_vector & device_vector
  - Generalizes std:vector
  - Store any type & dynamically resize
- Numerous algorithms
  - Sort
  - Sum
  - Max

Thrust: Getting started

- **Download to (CUDA include directory)**
  - **http://code.google.com/p/thrust/**
  - **Requires CUDA 2.3**
- **Tutorial:**
  - **http://code.google.com/p/thrust/wiki/Tutorial**

Thrust: Concept


```
// H has storage for 4 integers
thrust::host_vector<int> H(4);

// initialize individual elements
H[0] = 14; H[1] = 20; H[2] = 38; H[3] = 46;


// Copy host_vector H to device_vector D
thrust::device_vector<int> D = H;

// elements of D can be modified
D[0] = 99; D[1] = 88;
```

Thrust Algorithms: Prefix Sum

THE UNIVERSITY of WISCONSIN MADISON

- **Given a sequence:**

$$\{x_1, x_2, x_3, \ldots, x_N\}$$

- **And an operation**

$$\oplus$$

- **Output:**

$$\{x_1, x_1 \oplus x_2, x_1 \oplus x_2 \oplus x_3, \ldots, x_1 \oplus x_2 \oplus x_3 \ldots \oplus x_N\}$$

Prefix Sum

- **Key to numerous algorithms**
- **Also referred to as “Scan” algorithm**
- **Different operations result in different results**

Prefix Sum: Example

THE UNIVERSITY of WISCONSIN MADISON

- **Given a sequence:**

$$\{1, 2, 9, 6, \ldots,\}$$

- **And an operation**

$$+$$

- **Output**

$$\{x_1, x_1 + x_2, x_1 + x_2 + x_3, \ldots, x_1 + x_2 + x_3 \ldots + x_N\}$$

$$\{1, 3, 11, 17, \ldots\}$$

Prefix Sum: Example

THE UNIVERSITY of WISCONSIN MADISON

- **Given a sequence:**

$$\{1, 2, 9, 6, \ldots,\}$$

- **And an operation**

$$*$$

- **Output**

$$\{x_1, x_1 * x_2, x_1 * x_2 * x_3, \ldots, x_1 * x_2 * x_3 \ldots * x_N\}$$

$$\{1, 2, 18, 108, \ldots\}$$

Prefix Sum: Example

THE UNIVERSITY of WISCONSIN MADISON

- **Given a sequence:**

$$\{1, 2, 9, 6, \ldots,\}$$

- **And an operation**

$$\max$$

- **Output**

$$\{x_1, \max(x_1, x_2), \max(\max(x_1, x_2), x_3), \ldots\}$$

$$\{1, 2, 9, 9, \ldots\}$$

Thrust: Examples Set-up


```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <thrust/sort.h>
#include <thrust/transform_reduce.h>
#include <thrust/functional.h>
#include <cmath>
#include <cstdlib>

// generate random data on the host
thrust::host_vector<int> h_vec(N);
for( i = 0; i < h_vec.size(); i++)
  h_vec[i] = rand();
printHostVect(h_vec);


thrust::device_vector<int> d_vec = h_vec;
```

Thrust: Examples


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

//Copy back to host
h_vec = d_vec;
printHostVect(h_vec);



// Carry out a sum
int sum = thrust::reduce(d_vec.begin(), d_vec.end());
printf("\n\nSum: %d \n",sum);
```

Thrust: Examples cont.

$$a = \|x\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_N^2}$$

```
struct squareInt {
    __device__ int operator ()(int x) const {
      return x*x;
    }
};
 // setup arguments
 squareInt                unary_op;
 thrust::plus<float>      binary_op;
```

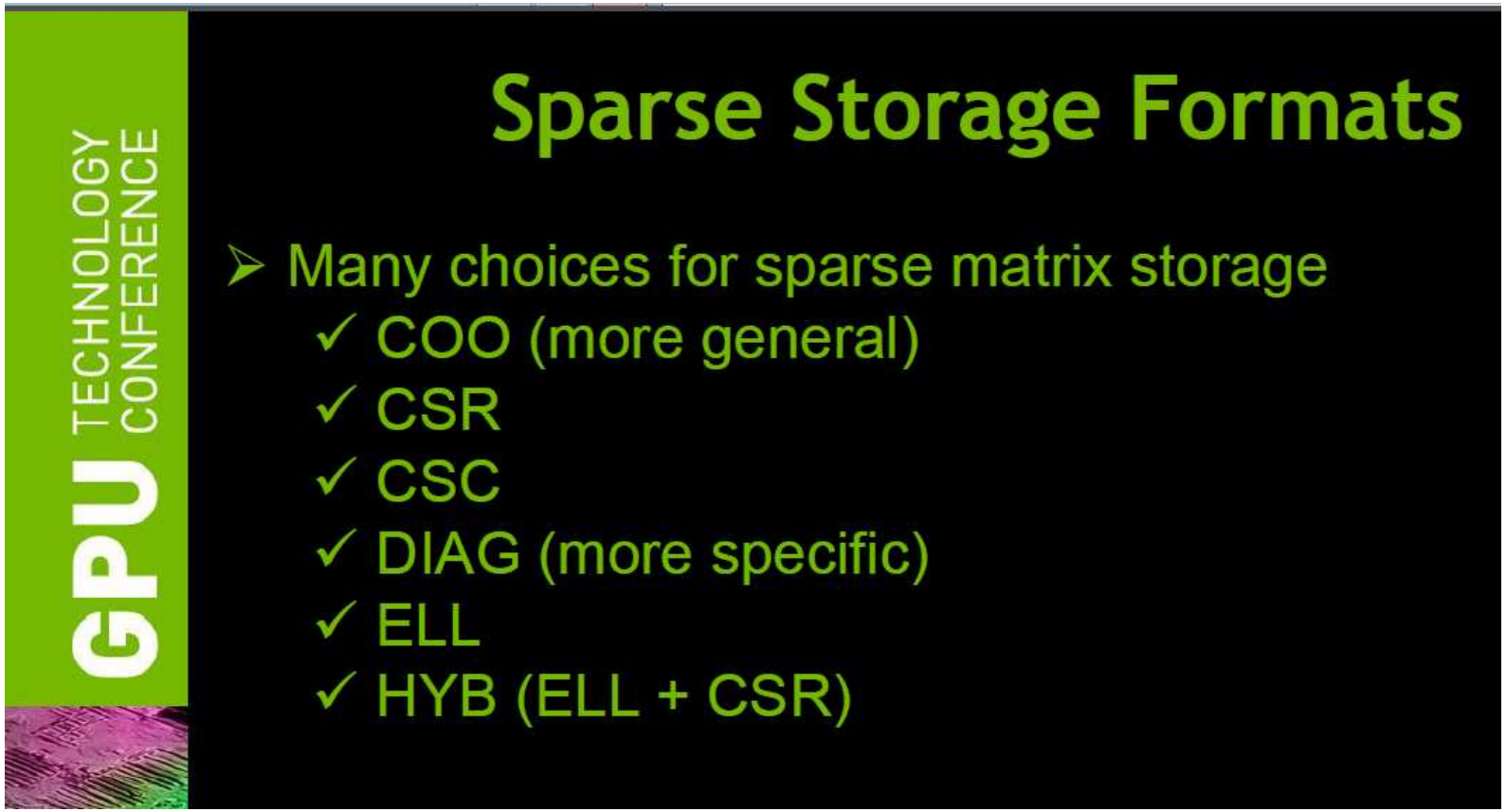
```
// compute norm
float normsqr =thrust::transform_reduce(d_vec.begin(), d_vec.end(), unary_op, 0, binary_op);

printf("\nNorm: %f \n\n",sqrt(normsqr));
```

CUDA Libraries & Packages

1. CUBLAS: Dense Linear Algebra
2. Thrust: Parallel sort, …
3. CuSparse: Sparse Linear Algebra Package
4. Jacket: Matlab Wrapper
5. CULA: Dense and sparse linear algebra
6. MAGMA: Multicore linear algebra
7. CUFFT: Fast Fourier Transform
8. …

Linear Algebra for sparse matrices using CUDA



CuSparse

THE UNIVERSITY of WISCONSIN MADISON

✓ csrmv (matrix-vector multiplication)

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

✓ csrtrsv (triangular solve with a single right-hand-side)

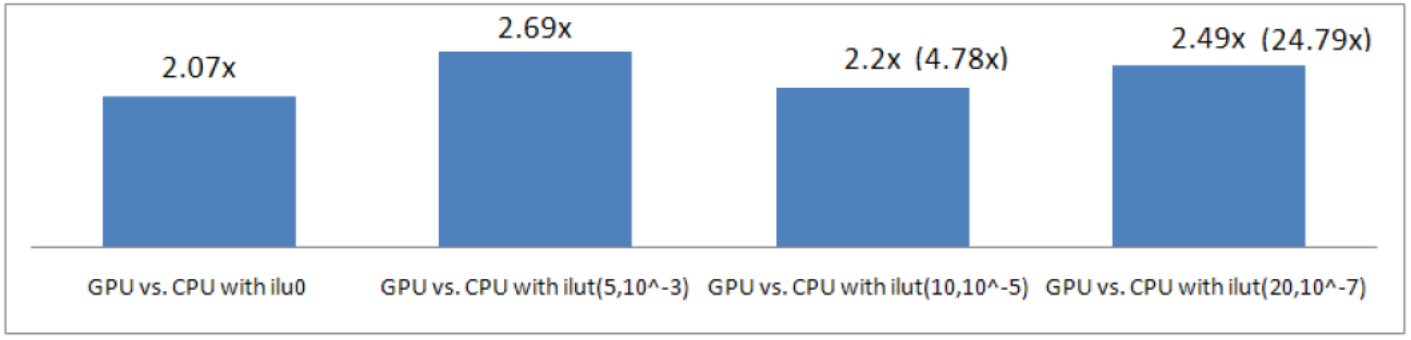
$$\begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 10.0 \\ 20.0 \\ 30.0 \\ 40.0 \end{bmatrix}$$

THE UNIVERSITY of WISCONSIN MADISON

Incomplete-LU and Cholesky Preconditioned Iterative Methods Using CUSPARSE and CUBLAS

Maxim Naumov
NVIDIA, 2701 San Tomas Expressway, Santa Clara, CA 95050

June 21, 2011



CUDA Libraries & Packages

1. CUBLAS: Dense Linear Algebra
2. Thrust: Parallel sort, …
3. CuSparse: Sparse Linear Algebra Package
4. CULA: Dense and sparse linear algebra
5. Jacket: Matlab Wrapper
6. MAGMA: Multicore linear algebra
7. CUFFT: Fast Fourier Transform
8. …

CULA Sparse

THE UNIVERSITY of WISCONSIN MADISON

CULA | sparse is a **GPU-accelerated library** for linear algebra that provides iterative solvers for sparse systems

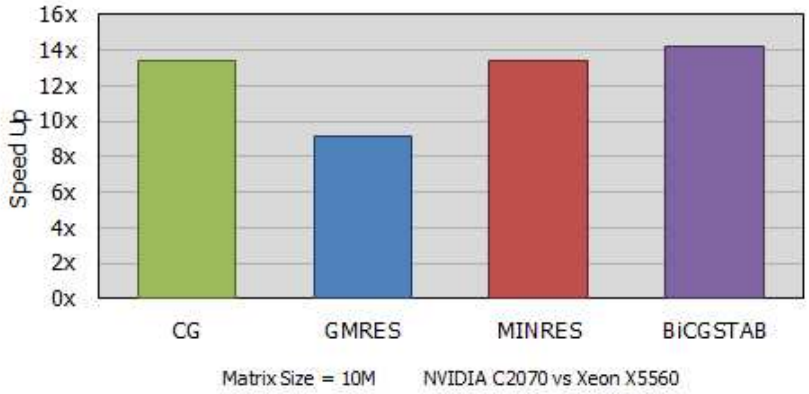
Offering a capability set that is unmatched in a ready-to-integrate library

With many solvers and preconditioners to choose from, you have the power to choose the solution that is best for you. Best of all, it is easy to use, with no complicated callback interfaces and a straightforward configuration.

SOLVERS	PRECONDITIONERS	DATA FORMATS
Biconjugate Gradient (BICG)	Jacobi	Double-precision real and complex
Biconjugate Gradient Stabilized (BICGSTAB)	Block Jacobi	Compressed Sparse Row (CSR)
Conjugate Gradient (CG)	Incomplete LU (Ilu0)	Compressed Sparse Column (CSC)
Generalized Minimum Residual (GMRES)	Reordered (Ilu0)	Coordinate (COO)
Minimum Residual (MINRES)		

With performance that is 10x faster than competing solutions

We are experts at getting the very best performance available. We combine powerful GPU-acceleration with capable numerical algorithms to yield solutions that reach levels of performance not previously possible.



CUFFT

CUDA Implementation of Fast Fourier Transform

Fourier Transform

- Extract frequencies from signal
- Given a function

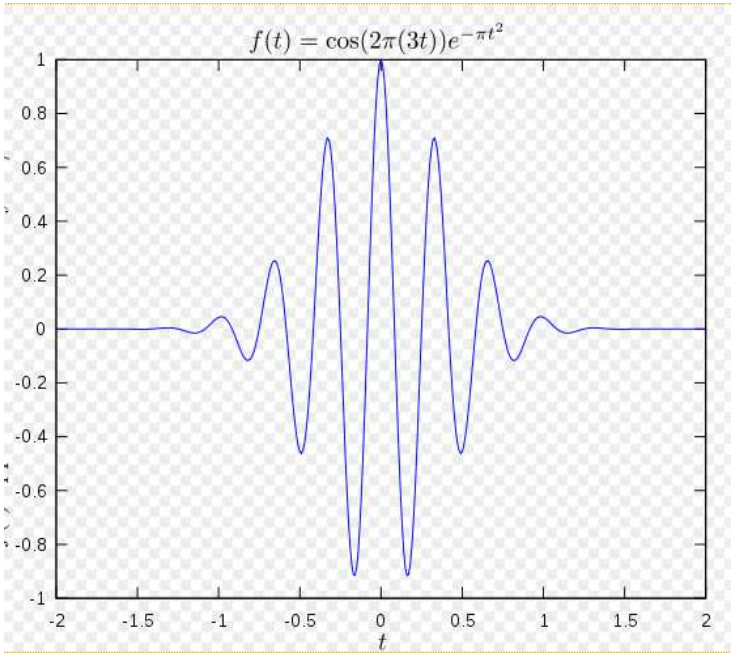
$$f(t); -\infty < t < \infty$$

- 1-D Fourier transform:

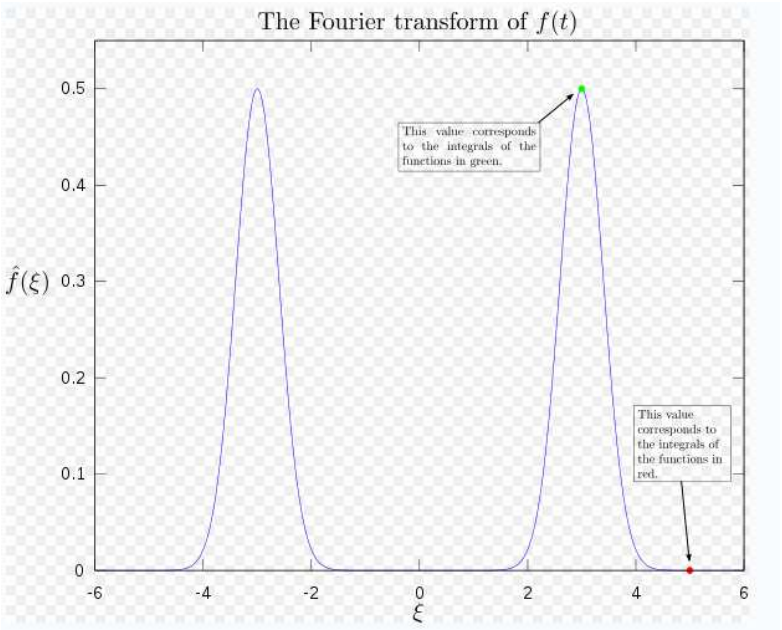
$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \xi t} dt$$

- 2-D, 3-D

Fourier Transform



Continuous Signal



Fourier Transform

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi t} d\xi$$

(Wikipedia)

Discrete Fourier Transform

- Given a sequence

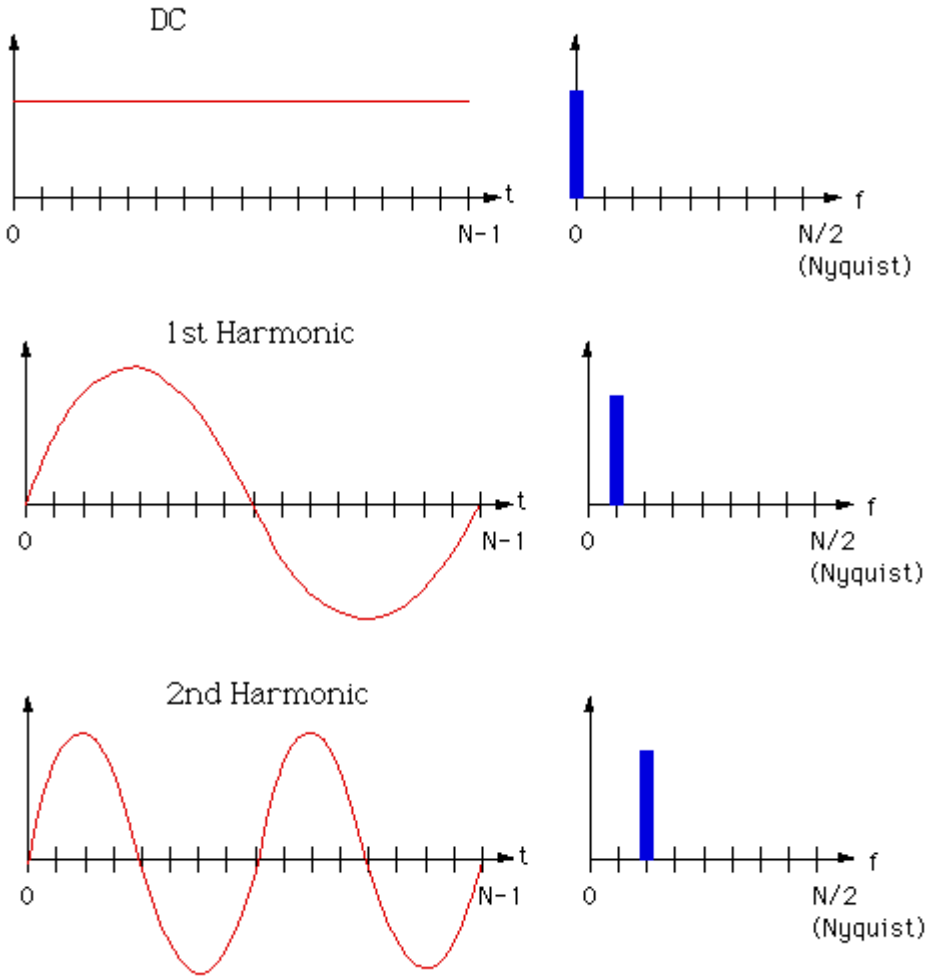
$$x_0, x_1, \ldots, x_{N-1}$$

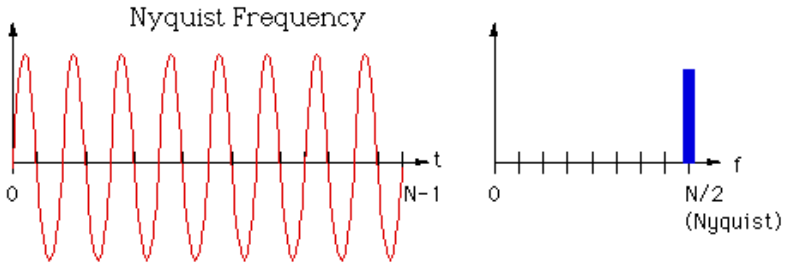
- Discrete Fourier transform (DFT):

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i k n}{N}}$$

… another sequence

DFT Examples





Highest frequency that can be captured correctly

Fast Fourier Transform

- DFT: Naïve O(N^2) operation $\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i k n}{N}}$

$x_0, x_1, \ldots, x_{N-1}$ $\quad\quad$ $\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N-1}$

- FFT: Fast DFT, O(NlogN)

- Key to signal processing, PDE, …

CUFFT

- **Fast CUDA library for FFT**
- **No additional downloads needed**
  - **cufft.lib (in CUDA SDK)**
  - **Add cufft.lib to linker**
  - **#include cufft.h**

CUFFT: Features

- 1-D, 2-D, 3-D
- Precisions
  - Single: real & complex
  - Double: real & complex (not all functions)
- Uses CUDA memory calls & fft data
- Requires a ‘plan’
- Based on FFTW model

CUFFT Example


```
#include <stdio.h>
#include <math.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <cufft.h>

#define VECTORSIZE   16
#define HARMONIC     2
#define pi           3.141592654

 cufftComplex data[VECTORSIZE];
 for(i=0 ; i < VECTORSIZE ; i++){
    data[i].x = cos(HARMONIC*2*pi*i/(VECTORSIZE));
    data[i].y = sin(HARMONIC*2*pi*i/(VECTORSIZE));
 }

 cufftComplex *devPtr;
```

```
cudaMalloc((void**)&devPtr, sizeof(cufftComplex)*VECTORSIZE);
cudaMemcpy(devPtr, data, sizeof(cufftComplex)*VECTORSIZE, cudaMemcpyHostToDevice);
```

CUFFT Example (cont.)


```
cufftHandle plan;

cufftPlan1d(&plan, VECTORSIZE, CUFFT_C2C, 1);

cufftExecC2C(plan, devPtr, devPtr, CUFFT_FORWARD);
```

complex

(batch)

```
cudaMemcpy(data, devPtr, sizeof(cufftComplex)*VECTORSIZE, cudaMemcpyDeviceToHost);
for(i = 0 ; i < VECTORSIZE ; i++)
    printf("data[%d] %f %f\n", i, data[i].x/(VECTORSIZE), data[i].y/(VECTORSIZE));
```

```
cufftDestroy(plan);
cudaFree(devPtr);
```

Acknowledgements

- Graduate Students
- NSF
- UW-Madison
- Kulicke and Soffa
- Luvata
- Trek Bicycles

Publications available at
www.ersl.wisc.edu

Email
suresh@engr.wisc.edu