CUDA-enabled Math Libraries Overview

NVIDIA Libraries

- http://developer.nvidia.com/cuda-tools-ecosystem
- CUDA Toolkit
  - CUBLAS – dense Linear Algebra
  - CUSPARSE – sparse Linear Algebra
  - CUFFT – Fast Fourier Transforms
  - CURAND – random numbers generators
- Also:
  - Thrust – C++ template library
  - NPP – image & signal processing

3rd party Libraries

- MAGMA – dense Linear Algebra, an open-sourced equivalent to CUBLAS with extensions
- CUSP – sparse Linear Algebra and iterative solvers
- OpenCurrent – PDE solvers for regular grid

CUBLAS

- Implements BLAS program interface
- Multidimensional arrays allocated in column-major order (fortran)

Level	Computantional complexity	Functions examples
1 (vector operations)	O(N)	AXPY: y := alpha x + y DOT: dot := (x,y)
2 (matrix-vector)	O(N^2)	GEMV – multiplate a general matrix-vector
3 (matrix-matrix)	O(N^3)	GEMM – multiplate two general matrices

- Documentation is shipped together with CUDA toolkit (CUBLAS_Library.pdf)

CUBLAS

- Naming rule: cublas<T><func>
  - <T> - data type {F, D, C, Z} corresponds to real, single and double prescizion; complex, single and double prescision
  - <func> - 3-4 letter of classic name of BLASfunction
  - Example: cublasDgemm

- In API 'v2' (available since CUDA 4.0) handles are used to make library thread-safe (when using several cuda Streams or multiple GPU)

CUBLAS - pipeline

- Initialize CUBLAS handle using *cublasCreate* function
- Allocate required memory on GPU and load the input data
- Call the required CUBLAS function sequence
- Transfer the results to host memory
- Free CUBLAS handle using *cublasDestroy* function

CUSPARSE

- **Purpose**: sparse matrix & vectors operations
- Sparse = significant amount of zero elements → dense representation is ineffective, indexed formats are used instead:
  - Sparse vector: 2 arrays – indexes and values
  - Saprse matrix: Coordinate Format (COO), Compressed Sparse Row (Column) Format (CSR, CSC)

CUSPARSE – features

- 4 groups of functions: cusparse<T><func>
  - On sparse vectors and dense vectors
  - On sparse matrices and dense vectors
  - On sparse vectors and multiple dense vectors
  - Matrix Format conversions
- Supports real and complex data of single and double precision

CUSPARSE – pipeline

1) Allocate memory and initialize data on GPU
2) Initialize the CUSPARSE library
3) Perform required CUSPARSE operation
4) Release memory and operation handle

CURAND

- Library of pseudo- and quasi- random values generators
- Based on XORWOW and Sobol algorithms
- Available distributions: uniform, normal, lognormal
- Has two interfaces:
  - For host (generation on CPU or on GPU)
  - For device (generation on GPU)

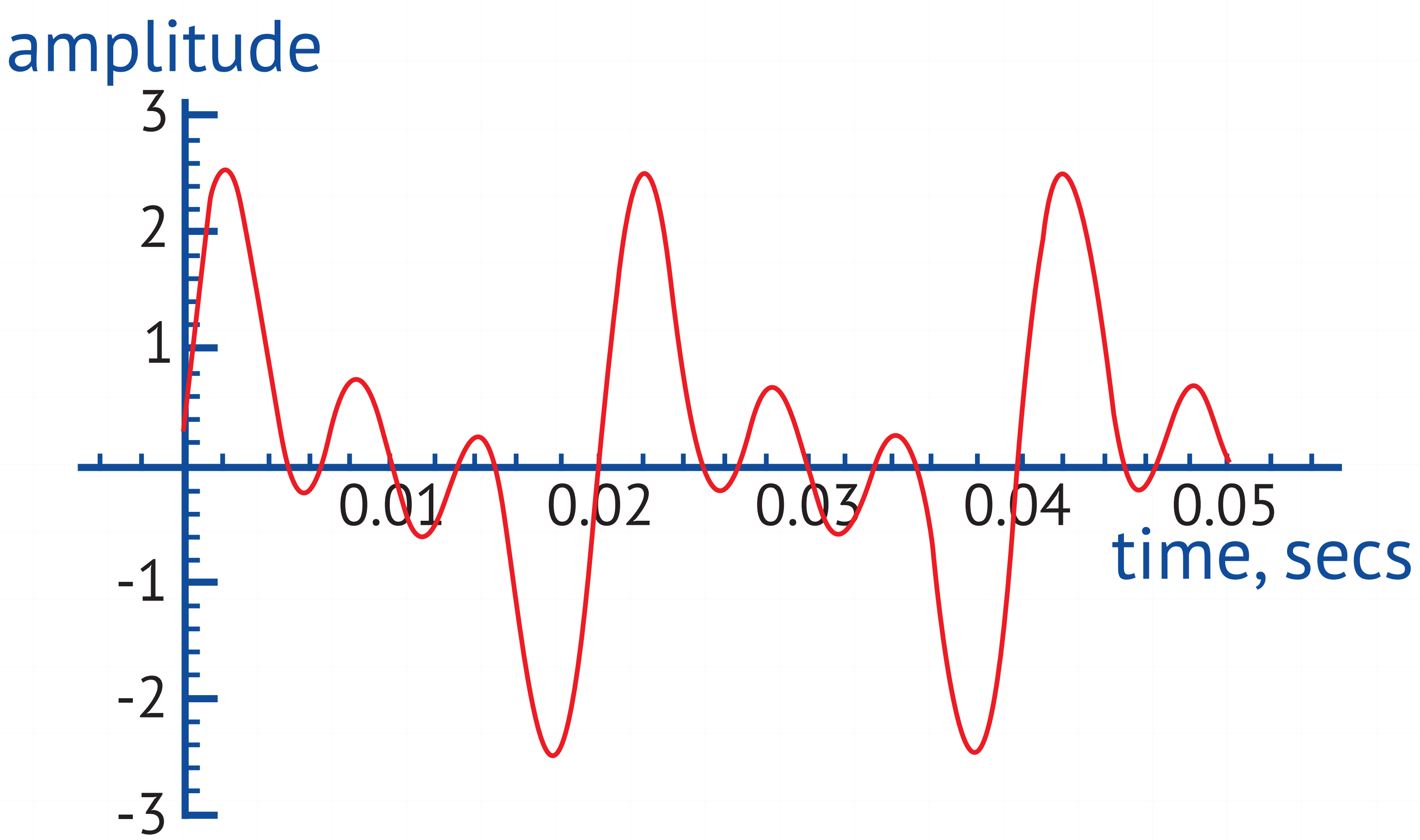
CURAND – pipeline (host)

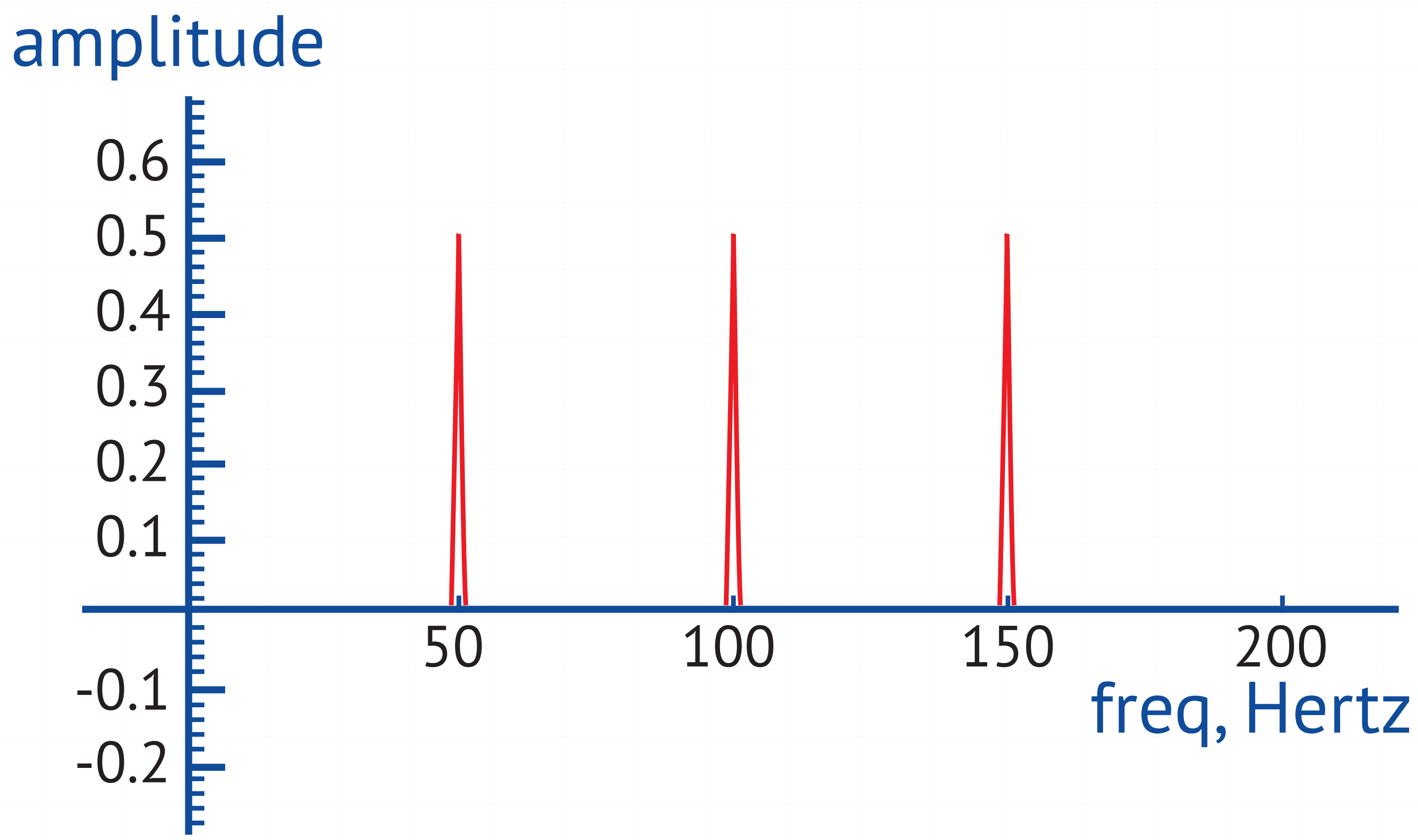
- Create generator using *curandCreateGenerator()*
- Set properties of generator (for example, initial state: *curandSetPseudoRandomGeneratorSeed()*)
- Allocate the GPU memory using *cudaMalloc()*
- Run the generator, using the appropriate distribution
- Use the results of generator
- If necessary, generate additional data
- Destroy generator *curandDestroyGenerator()*

CURAND – pipeline (device)

- Allocate the array of states (*curandState*) of generators in each thread
- Create and run the kernel, initializing the *curandState* for each thread
- Create and run the kernel, using random values, generated by *curand(curandState)*
- Free memory

CUFFT – the Fast Fourier Transforms





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

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

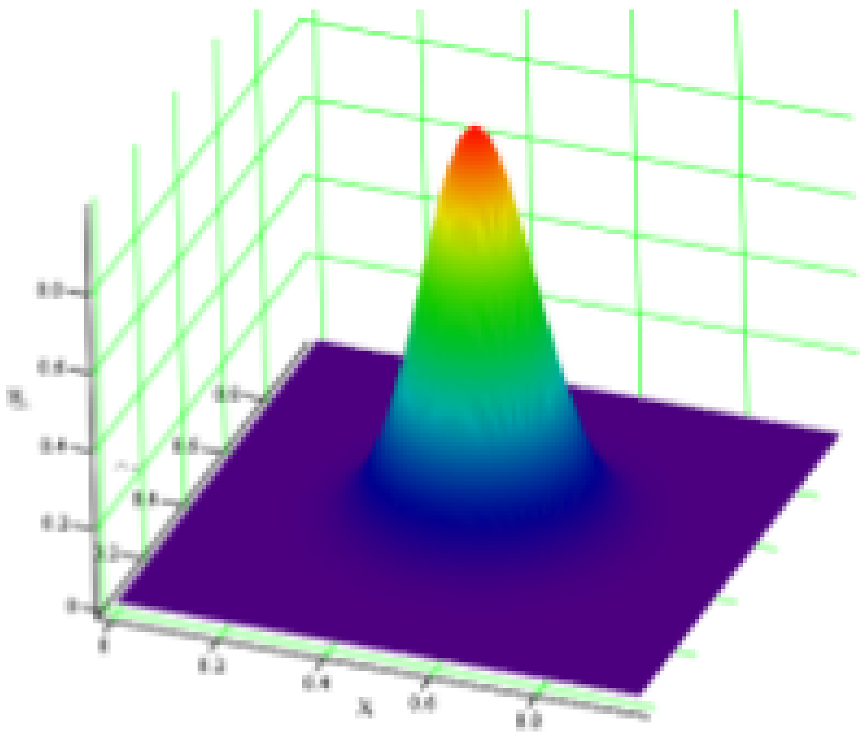
CUFFT – the Fast Fourier Transforms

- The interface is similar to FFTW, supports FFTW compatibility mode
- Implements Cooley–Tukey and Bluestein's FFT algorithms
- 1-d, 2-d and 3d real and complex transforms of single or double precision
- Supports asynchronous transforms with CUDA streams
- Unnormalized output: IFFT(FFT(A))=len(A)*A

CUFFT – the pipeline

1) Allocate memory on GPU
2) Create FFT transform plan with the required size and type
3) Perform FFT transforms on data in GPU memory
4) Delete transform plan and release the GPU memory

CUFFT – sample



- Consider the Poisson equation:

$$\begin{cases} \Delta u(p) = \rho(p), p \in \Omega = (x,y), 0 \leq x, y \leq 1, \\ \rho(x,y) = \dfrac{s(x,y) - 2\sigma^2}{\sigma^4} exp^{-\frac{s(x,y)}{2\sigma^2}}. \end{cases}$$

- The known exact solution is: $u_0(x,y) = exp^{\frac{s(x,y)}{2\sigma^2}}$

- With some assumptions the approximate numerical solution can be found:

$$\overline{u}(n,m) = \overline{\rho}(n,m)h^2(W^{-n} + W^n + W^{-m} + W^m - 4)^{-1}, \qquad \overline{\rho} = \frac{1}{N^2}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1}\rho(x_k, y_j)W^{nk+mj}$$

$$u(x_k, y_j) = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1}\overline{u}(n,m)W^{nk+mj}. \qquad W = exp^{i\frac{2\pi}{N}}$$