Introduction to Scientific Programming using GPGPU and CUDA

Day 2

Sergio Orlandini
s.orlandini@cineca.it

Mario Tacconi
m.tacconi@cineca.it

CINECA SCAI SuperComputing Applications and Innovation

- Tools from CUDA-Toolkit
  - Profiler
  - CUDA-GDB
  - CUDA-memcheck
  - Parallel Nsight
- CUDA-Enabled Libraries
  - CUBLAS
  - CUFFT
  - CUSPARSE
  - CURAND
  - MAGMA, THRUST, CUDDP, …
- Hands on

Profiling tools: built-in

- CUDA toolkit prodives useful tools for profiling your code

```
export CUDA_PROFILE=1
export CUDA_PROFILE_CONFIG=$HOME/.config
```

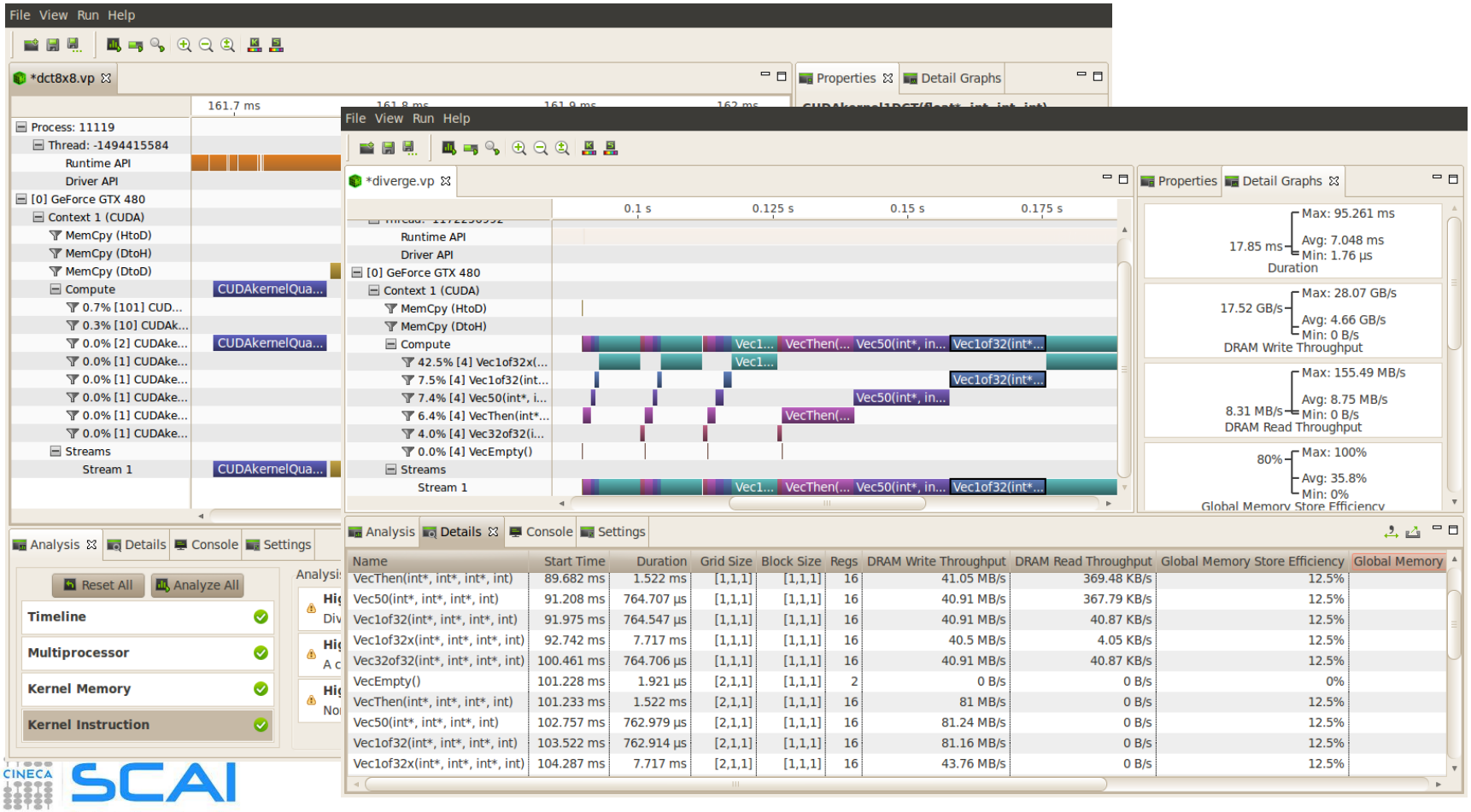
```
// Contents of config
gld_coherent
gld_incoherent
gst_coherent
gst_incoherent
```

```
gld_incoherent: Number of non-coalesced global memory loads
gld_coherent: Number of coalesced global memory loads
gst_incoherent: Number of non-coalesced global memory stores
gst_coherent: Number of coalesced global memory stores
local_load: Number of local memory loads
local_store: Number of local memory stores
branch: Number of branch events taken by threads
divergent_branch: Number of divergent branches within a warp
instructions: instruction count
warp_serialize: Number of threads in a warp that serialize
based on address conflicts to shared or constant memory
cta_launched: executed thread blocks
```

```
method,gputime,cputime,occupancy,gld_incoherent,gld_coherent,gst_incoherent,gst_coherent
method=[ memcopy ] gputime=[ 438.432 ]
method=[ _Z17reverseArrayBlockPiS_ ] gputime=[ 267.520 ] cputime=[ 297.000 ] occupancy=[ 1.000 ]
gld_incoherent=[ 0 ] gld_coherent=[ 1952 ] gst_incoherent=[ 62464 ] gst_coherent=[ 0 ]
method=[ memcopy ] gputime=[ 349.344 ]
...
...
...
```

Profiling: Visual Profiler

- Traces hosts and device calls, data transfers, kernel launches, shows overlapping streams and measure performances
- supports automated analysis (hardware counters)



Debugging: CUDA-GDB

- CUDA Toolkit also provides a cuda-gdb text debugger
  - the traditional gdb enchanced with CUDA extentions

```
(cuda-gdb) info cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count Virtual PC Filename Line
Kernel 0* (0,0,0) (0,0,0) (0,0,0) (255,0,0) 256 0x0000000000866400
bitreverse.cu 9
(cuda-gdb) thread
[Current thread is 1 (process 16738)]
(cuda-gdb) thread 1
[Switching to thread 1 (process 16738)]
#0 0x000019d5 in main () at bitreverse.cu:34
34 bitreverse<<<1, N, N*sizeof(int)>>>(d);
(cuda-gdb) backtrace
#0 0x000019d5 in main () at bitreverse.cu:34
(cuda-gdb) info cuda kernels
Kernel Dev Grid SMs Mask GridDim BlockDim Name Args
0 0 1 0x00000001 (1,1,1) (256,1,1) bitreverse data=0x110000
```

Debugging: CUDA-MEMCHECK

- Very useful to find out-of-bound, misaligned global memory accesses
- stand alone or integrated in CUDA-GDB

```
$ cuda-memcheck --continue ./memcheck_demo
========= CUDA-MEMCHECK
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
Sync: no error
Running out_of_bounds_kernel
Ran out_of_bounds_kernel: no error
Sync: no error
========= Invalid __global__ write of size 4
========= at 0x00000038 in memcheck_demo.cu:5:unaligned_kernel
========= by thread (0,0,0) in block (0,0,0)
========= Address 0x200200001 is misaligned
=========
========= Invalid __global__ write of size 4
========= at 0x00000030 in memcheck_demo.cu:10:out_of_bounds_kernel
========= by thread (0,0,0) in block (0,0,0)
========= Address 0x87654320 is out of bounds
=========
=========
========= ERROR SUMMARY: 2 errors
```

Parallel NSight

- Plug-in available for Eclipse and Microsoft VisualStudio IDE
- Aggregates all external functionalities:
  - Debugger (fully integrated)
  - Visual Profiler
  - Memory correctness checker
- As a plug-in, it extends all the convenience of IDE development to CUDA
- Very fast growing community of users and developers
- A number of very nice (and useful) features:
  - supports multi-GPU
  - remote debug and profiling
  - PTX assembly view
  - warp inspector

Parallel NSight

voxelpipe_demo_vc10 (Debugging) - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Nsight Data Tools Test Analyze Window Help

Debug x64 SubFrame CUDA (1,0,0), (21,0,0)

Process: [1840] voxelpipe_demo.exe Thread: [2874912] <No Name> Stack Frame: CUmodule 05508fe0 - [2] trace - Line 148

Connections: localhost

CUDA Info 1

Warps Filter: Viewing 128/128

Current	blockIdx	Warp Index	PC	Active Mask	Status	Exception	File Name	Source Lin	Lanes
	(0, 0, 0)	0	0x003e1ad8	0xffffff80	Breakpoint	None	rt_render.cu	163	
	(0, 0, 0)	1	0x003e1ad8	0xffffff00	Breakpoint	None	rt_render.cu	163	
	(0, 0, 0)	2	0x003e1ad8	0xfffffc00	Breakpoint	None	rt_render.cu	163	
	(0, 0, 0)	3	0x003e1ad8	0xfffff800	None	None	rt_render.cu	163	
➡	(1, 0, 0)	0	0x003e1298	0x03e00000	Breakpoint	None	rt_render.cu	148	
	(1, 0, 0)	1	0x003e1298	0x07c00000	Breakpoint	None	rt_render.cu	148	
	(1, 0, 0)	2	0x003ede70	0xffffffff	None	None	ci_include.h	423	

CUDA Info 1 | Modules

rt_render.cu

(Unknown Scope)

```
143             node_index = node.get_index(); // jump to child
144         }
145         else
146         {
147             // leaf intersection
148             const uint32 leaf_index = node.get_index();
149             const Bvh_leaf leaf     = geometry.m_bvh_leaves[ leaf_index ];
150             const uint32 leaf_end   = leaf.get_index() + leaf.get_size();
151             const uint32 leaf_begin = leaf.get_index();
152
153             for (uint32 tri_index = leaf_begin; tri_index < leaf_end; ++tri_index)
```

100 %

Disassembly

Address:

Viewing Options

```
148:            const uint32 leaf_index = node.get_index(
0x003e1298   2800400010019de4   MOV R6, c[0x0][0x4];
0x003e12a0   28000000fc01dde4   MOV R7, RZ;
0x003e12a8   28000000fc01dde4   MOV R7, RZ;
0x003e12b0   2800000018019de4   MOV R6, R6;
0x003e12b8   4801000018411c03   IADD R4.CC, R4, R6;
0x003e12c0   480000001c515c43   IADD.X R5, R5, R7;
0x003e12c8   2800000010011de4   MOV R4, R4;
0x003e12d0   2800000014015de4   MOV R5, R5;
0x003e12d8   2800000014015de4   MOV R5, R5;
```

Locals

Name	Value	Type
leaf	{m_size = 67106176, m_index = 0}	__local__
leaf_index	'leaf_index' has no value at the target location.	
leaf_end	'leaf_end' has no value at the target location.	
leaf_begin	'leaf_begin' has no value at the target location.	
node	{m_packed_data = 2147484877, m_skip_node = 248	__local__
__T21669	{x = -1.4394605, y = -1.8220775, z = -2.150774}	__local__
ray_inv	{x = -1.4394605, y = -1.8220775, z = -2.150774}	__local__
node_index	'node_index' has no value at the target location.	

Call Stack

Name	Language
CUmodule 05508fe0 - [2] trace - Line 148	CUDA
CUmodule 05508fe0 - [1] render_pixel - Line 409	CUDA
CUmodule 05508fe0 - [0] rt_trace_primary_kernel - Line 493	CUDA

CUDA WarpWatch 1

Name	ray_inv.x	ray_inv.y	ray
Type	__local__ float	__local__ float	__loca
0	-1.4444908	-1.7955524	-2.17
1	-1.44425	-1.7967783	-2.17
2	-1.4440092	-1.7980076	-2.17
3	-1.4437686	-1.7992405	-2.17
4	-1.4435281	-1.800477	-2.17
5	-1.4432876	-1.8017174	-2.17
6	-1.4430474	-1.8029615	-2.17
7	-1.4428074	-1.8042094	-2.16
8	-1.4425675	-1.8054608	-2.16
9	-1.4423276	-1.8067161	-2.16
10	-1.4420878	-1.8079749	-2.16
11	-1.4418485	-1.8092378	-2.16
12	-1.4416089	-1.8105046	-2.16
13	-1.4413697	-1.8117749	-2.16
14	-1.4411306	-1.8130492	-2.16
15	-1.4408917	-1.8143274	-2.15
16	-1.4406527	-1.8156093	-2.15
17	-1.4404141	-1.8168953	-2.15
18	-1.4401754	-1.818185	-2.15
19	-1.439937	-1.8194786	-2.15
20	-1.4396986	-1.820776	-2.15
21	-1.4394605	-1.8220775	-2.15
22	-1.4392225	-1.8233831	-2.14
23	-1.4389844	-1.8246926	-2.14
24	-1.4387469	-1.8260059	-2.14
25	-1.4385092	-1.8273233	-2.14
26	-1.4382718	-1.828645	-2.14
27	-1.4380344	-1.8299706	-2.14
28	-1.4377974	-1.8313001	-2.14
29	-1.4375603	-1.832634	-2.13
30	-1.4373236	-1.8339716	-2.13
31	-1.4370868	-1.8353136	-2.13

CUDA WarpWatch 1 | Output

Ready

Parallel NSight

supersonicsled - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Nsight Data Tools Test Analyze Window Help

Debug | Win32 | RuntimeApiTrace_t

SupersonicSled1111...pture_000.nvreport | Activity1.nvact*

Timeline

Row Filters

Time

0.0 % [71] fluid_advectVelocity_k
0.0 % [70] fluid_diffuseProject_k
0.0 % [70] fluid_updateVelocity_k
0.0 % [70] fluid_updateParticles_k
0.0 % [71] fluid_offsetVelocities_k
0.0 % [284] fluid_setVelocity_k
Streams
Counters
DX
Frames
Device Context 0x3C2BCC0
Device 0x200
Draw Calls
System
CPU Usage
Core 0
Core 1
Core 2

Row Information
Draw Calls [DirectX Draw Calls Row]
Cursor Information
161491 [DirectX Draw Call Workload]
8850 [DirectX Command Buffer Workload]
DrawIndexed [DirectX Draw Function Call]
0x3C2BCC0 [DirectX Device Context]

Time Range

Start	Before Capture
End	After Capture
Duration	

System

Process	supersonicsled.exe [5812]
# Threads	32
Command Line	"c:\temp\sdomine-lt4\c\program files (x86)\nvidia corporation\nvidia demos\supersonic sled\bin\supersonicsled.exe"
Exit Status	259

Output | Find Results 1

Ready

CUDA Enabled Libraries

- CUDA Toolkit includes many usefull libraries:
  - CUBLAS: Basic Linear Algerba Subprograms
  - NVBLAS: multi-GPUs accelerated drop-in BLAS built on top of cuBLAS
  - CUFFT : Fast Fourier Transform
  - CUSPARSE: space matrix linear algebra
  - CURAND: pseudorandom and quasirandom number generator
  - NPP: image,signal processing, statistic (nVIDIA Performance Primitives)
  - THRUST: vector-based library for parallel algorythms in C++
- other CUDA enabled libraries outside the CUDA Toolkit :
  - MAGMA (Matrix Algebra on GPU and Multicore Architectures)
    http://icl.cs.utk.edu/magma/
  - CUDPP (Data Parallel Primitives): parallel prefix-sum, sort, reduction
    http://code.google.com/p/cudpp/
  - CULA: CUDA LAPACK API (single precision version is freely available)
    http://www.culatools.com/contact/cuda-training/
  - CUSP: CUDA Sparse Solver and graph:
    http://developer.nvidia.com/object/npp_home.html

for a complete list, visit CUDA-Zone and look for GPU-Accelerated Libraries

CUDA Math Library

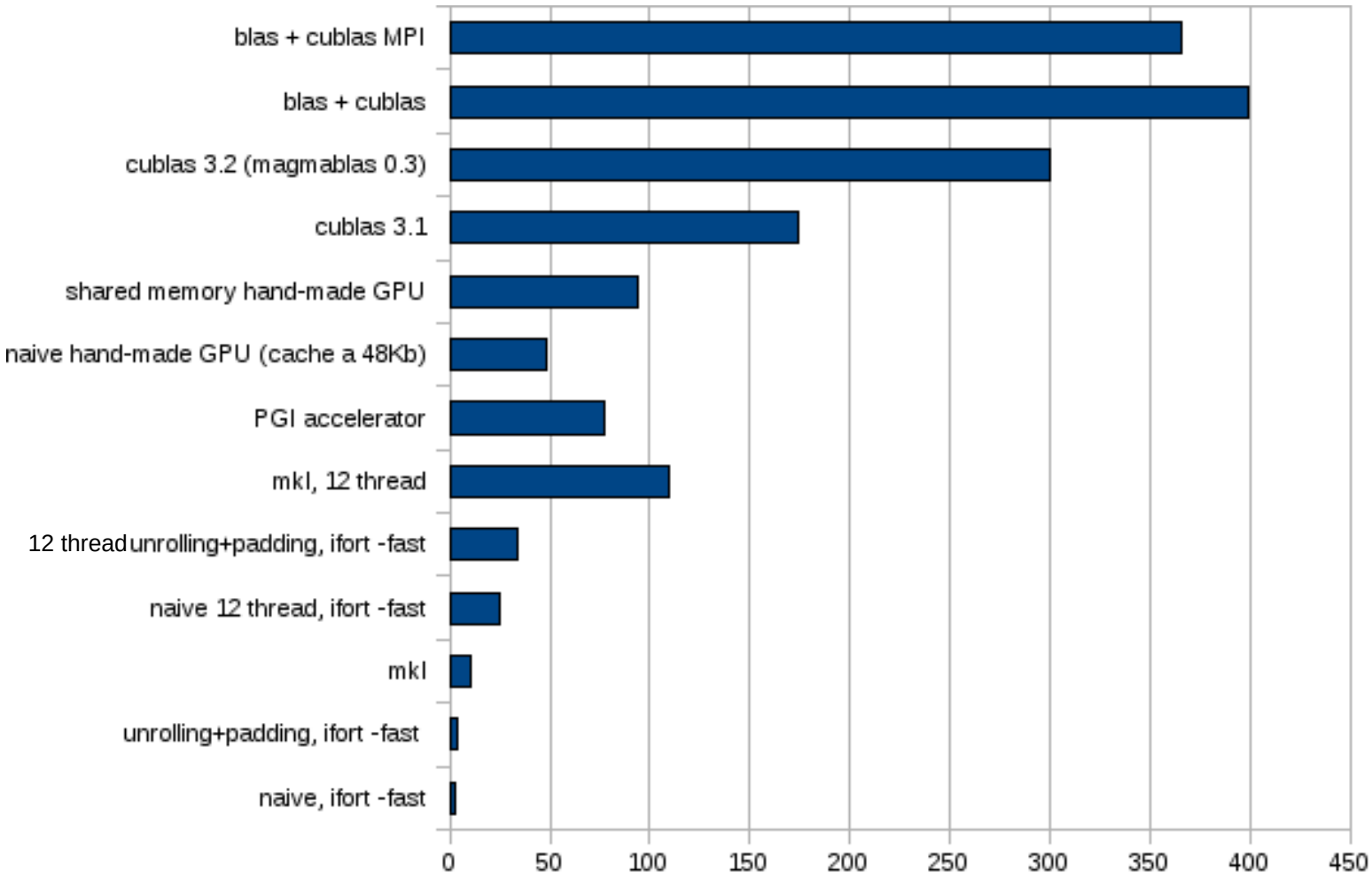
- Of course, CUDA Toolkit provides mathematical functions as other higher level language
  - by simply adding "`#include math.h`" in your source code
  - Complete support for all C99 standard float and double math functions
- IEEE-754 accurate for float, double, and all rounding modes
  - Extended Trigonometry and Exponential Functions
    - cospi, sincos, sinpi, exp10
  - Inverse Error Functions (erfinv, erfcinv)
  - Optimized Reciprocal Functions (rsqrt, rcbrt)
  - Bessel Functions (j0,j1,jn,y0,y1,yn)
- Floating Point Data Attributes (signbit, isfinite, isinf, isnan)
- intrinsic versions are also provided

cuBLAS Library

- BLAS is a standard in terms of interface and accuracy for most other libraries which implements linear algebra operations
  - BLAS Level 1:
  - BLAS Level 2:
  - BLAS Level 3:
- There are vendor optimized implementation for many hardware architecture (Intel, Power, ARM, etc)
- CUDA Toolkit provide a CUDA-enabled implementation of all BLAS
- WARNING: data layout in memory follows column-major ordering and 1-based indexing
- Function naming convention: cublas + BLAS name

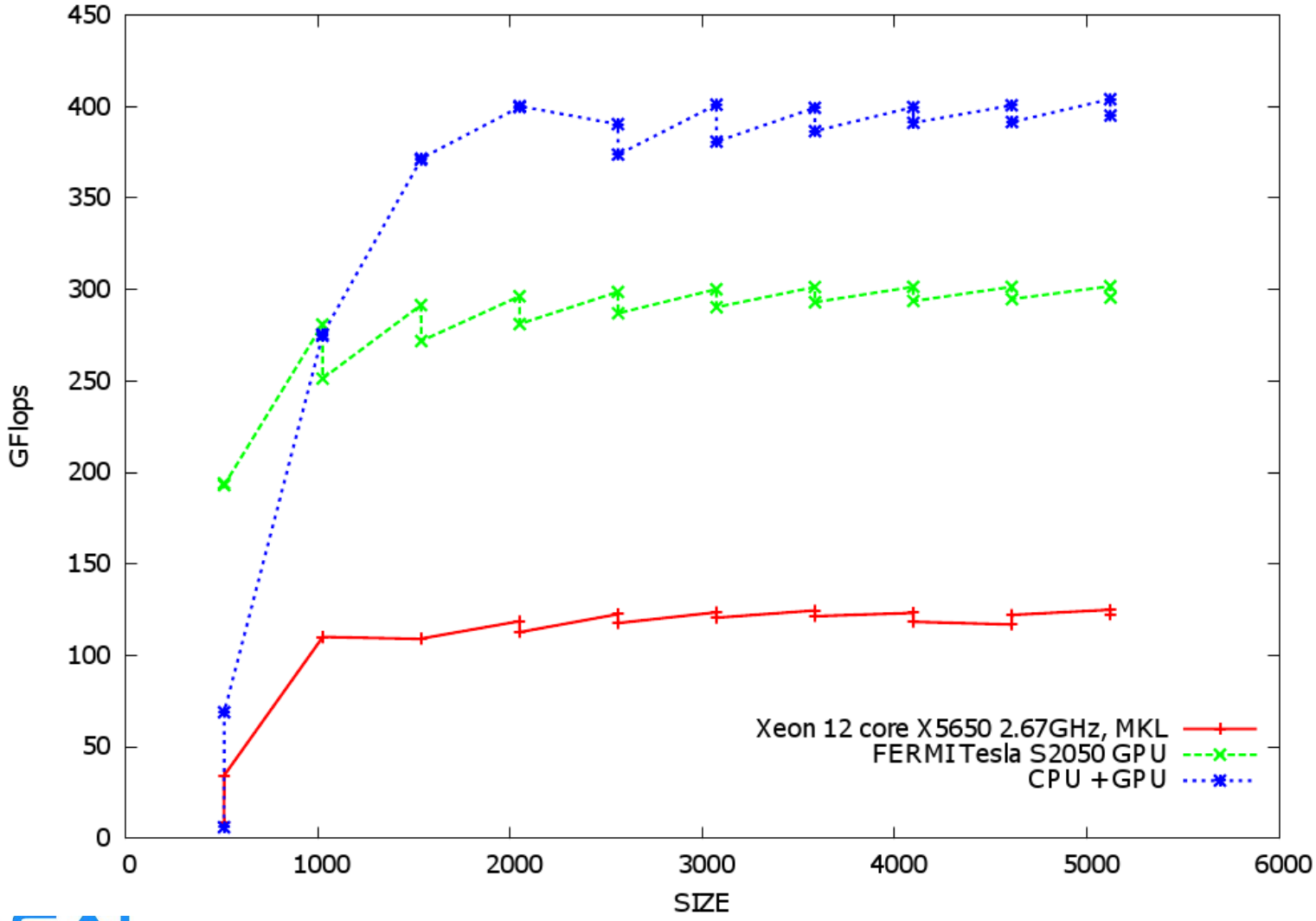
cuBLAS: DGEMM performance

- Here we show a simple performance study case for DGEMM (double-precision dense matrix matrix product - old system)



CUBLAS: DGEMM performance

- Performance versus matrix size dependency



cuBLASXT

- Starting with CUDA 6.0, the cuBLAS Library exposes two API
  - the regular cuBLAS API
  - the new cuBLASXT API
- With cuBLAS API
  - the application must allocate the required matrices and vectors in the GPU memory space
  - fill them with data, call the sequence of desired cuBLAS functions,
  - then upload the results from the GPU memory space back to the host
- With cuBLASXT API
  - the application must allocate data using managed memory
  - the library will take care of dispatching the operations to one or multiple GPUs present in the system

cuFFT

- cuFFT is the CUDA version of the Fast Fourier Transform
  - based on Cooley-Turkey and Bluestein algorithm
- cuFFT API is very similar to the FFTW one
  - as FFTW does, cuFFT use the *workplan* concept to optimize its work
  - once a *workplan* is computed, the librery itself mantains necessary information to execute FFT operation on data many times efficiently
  - WARNING: cuFFT follow row-major convention for data in memory
- Other key features:
  - provides 1D, 2D, 3D transform
    - for many real and complex types (single, double, quad precision)
    - in-place and out-of-place transforms
  - non-normalized output:
    - IFFT(FFT(A)) = len(A) * A
  - support for asynchronous operation on CUDA streams
  - thread-safe (**CUDA 4.1**)

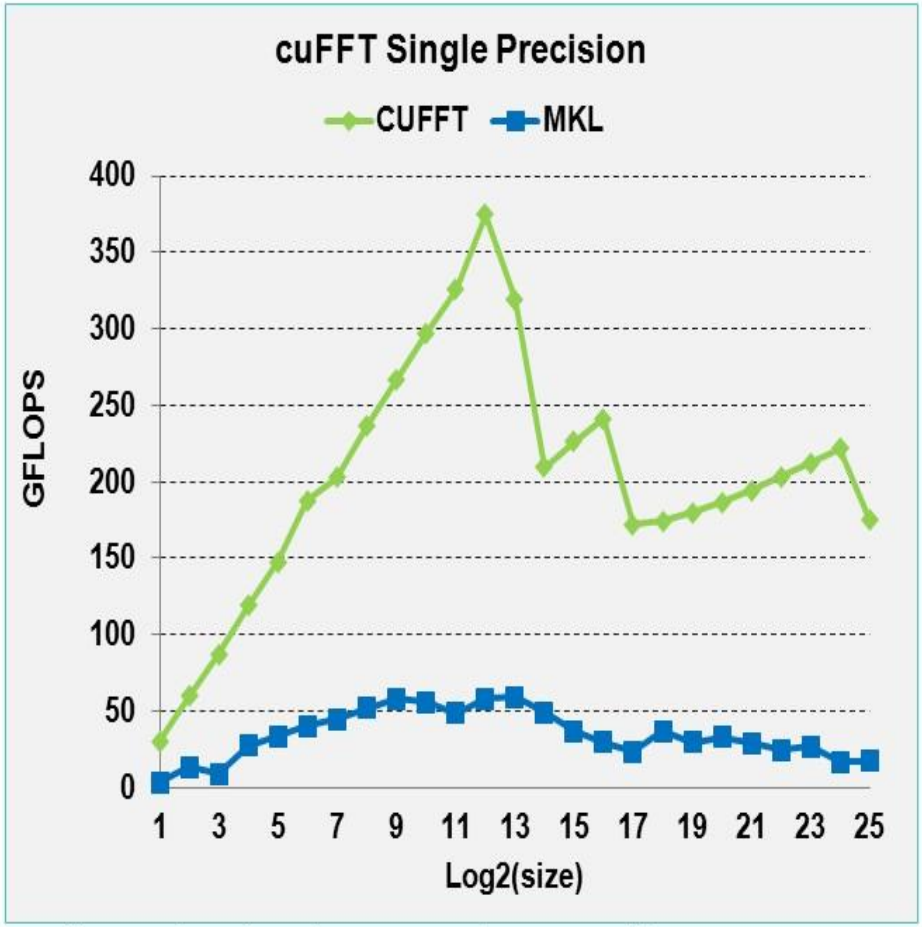
cuFFT sample: 2D complex-complex

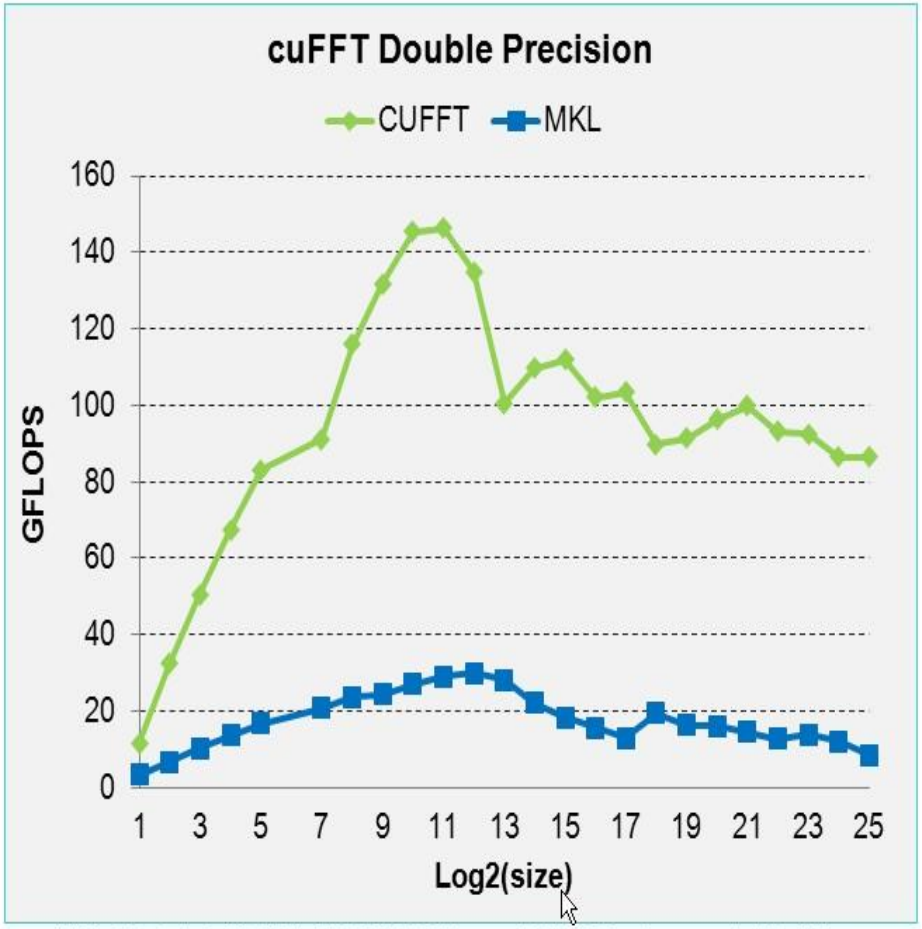
```
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata, *odata;
cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
cudaMalloc((void**)&odata, sizeof(cufftComplex)*NX*NY);
…
/* create a plan for FFT 2D */
 cufftPlan2d(&plan, NX,NY, CUFFT_C2C);
/* use plan for "out of place" transform */
 cufftExecC2C(plan, idata, odata, CUFFT_FORWARD);
/* back transform "in place" */
 cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);
/* if input output pointers differ, "out of place" is implied */

/* destroy plan and free resources */
 cufftDestroy(plan);
 cudaFree(idata), cudaFree(odata);
```

cuFFT: performances of FFT1D

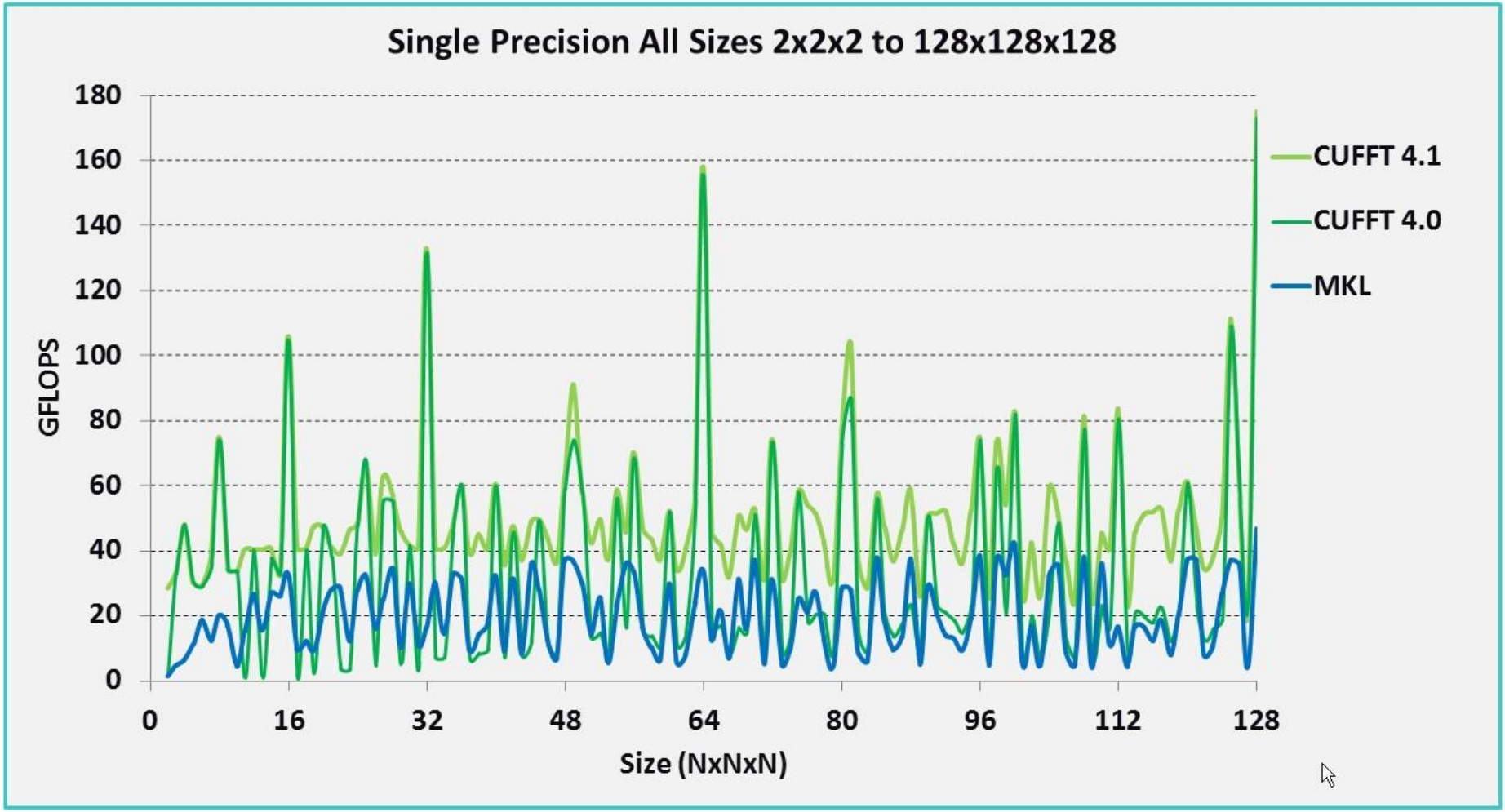




- Measured on sizes that are exactly powers-of-2
- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz
- Performance may vary based on OS version and motherboard configuration

cuFFT: performances of FFT3D

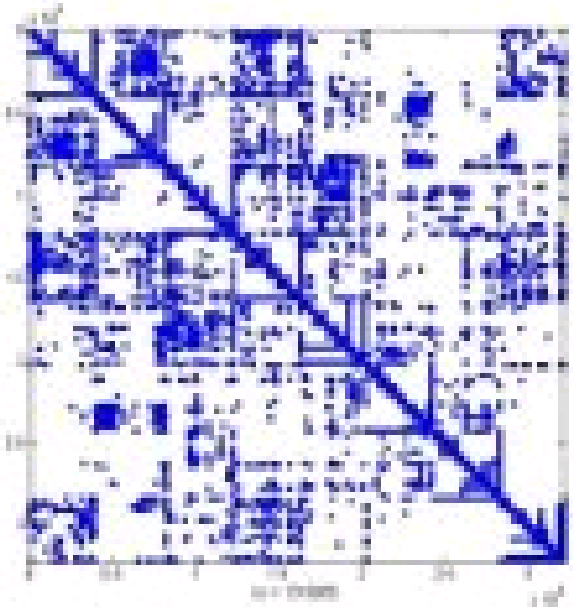


- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

- Performance may vary based on OS ver. and motherboard config.

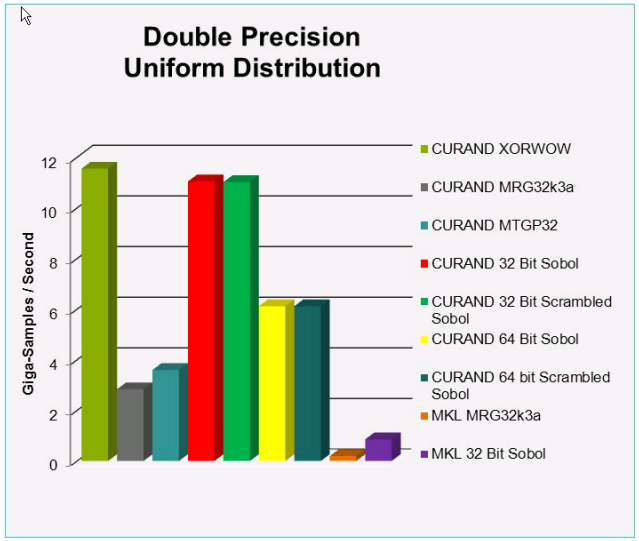
cuSPARSE

- support for dense, COO, CSR, CSC, ELL/HYB and Blocked CSR sparse matrix formats
- Level 1 routines for sparse vector x dense vector operations
- Level 2 routines for sparse matrix x dense vector operations
- Level 3 routines for sparse matrix x multiple dense vectors (tall matrix)
- Routines for sparse matrix by sparse matrix addition and multiplication
- Conversion routines that allow conversion between different matrix formats
- Sparse Triangular Solve
- Tri-diagonal solver
- Incomplete factorization preconditioners ilu0 and ic0

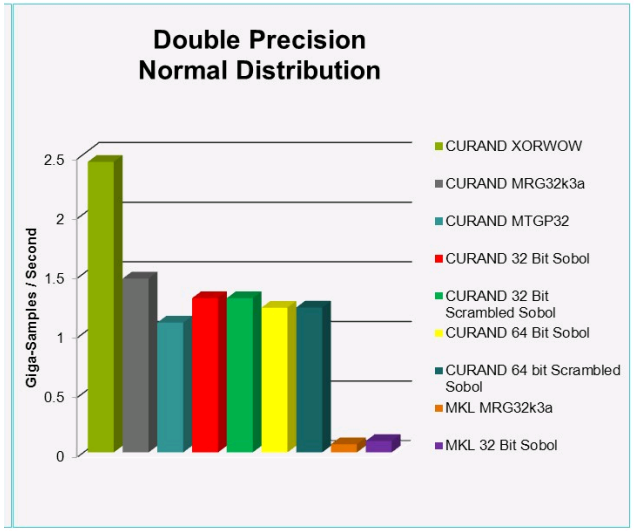


cuRAND

- **Flexible usage model**
  - Host API for generating random numbers in bulk on the GPU
  - Inline implementation allows use inside GPU functions/kernels, or in your host code
- **Four high-quality RNG algorithms**
  - MRG32k3a
  - MTGP Merseinne Twister
  - XORWOW pseudo-random generation
  - Sobol' quasi-random number generators, including support for scrambled and 64-bit RNG
- **Multiple RNG distribution options**
  - Uniform distribution
  - Normal distribution
  - Log-normal distribution
  - Single-precision or double-precision



- cuRAND 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 ver. and motherboard config.

cuRAND

1. **Create a generator:**

   curandCreateGenerator()

2. **Set a seed:**

   curandSetPseudoRandomGeneratorSeed()

3. **Generate the data from a distribution:**

   curandGenerateUniform()/curandGenerateUniformDouble() : Uniform

   curandGenerateNormal()/cuRandGenerateNormalDouble() : Gaussian

   curandGenerateLogNormal/curandGenerateLogNormalDouble() : Log-Normal

4. **Destroy the generator:**

   curandDestroyGenerator()

cuRAND

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand.h>

int main() {

int i, n = 100;
curandGenerator_t gen;
float *devData, *hostData;

// Allocate n floats on host
hostData = (float *) calloc (n,
 sizeof(float));

// Allocate nfloats on device
cudaMalloc((void **) &devData, n *
 sizeof(float));

// Create psudo-random number generator
curandCreateGenerator(&gen,
 CURAND_RNG_PSEUDO_DEFAULT);

// set seed
curandSetPseudoRandomGeneratorSeed(gen,
 1234ULL);
```

```
// generate n float on device
curandGenerateUniform(gen, devData, n);

// copy device memory to host
cudaMemcpy(hostData, devData, n *
 sizeof(float),
   cudaMemcpyDeviceToHost);


// show result
for (i = 0; i < n; i++) {
  printf("%1.4f ", hostData[i]);
}
printf("\n");

// Cleanup
curandDestroyGenerator(gen);

cudaFree(devData)
free(hostData)

return 0;

}
```

cuRAND

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand_kernel.h>

___global__ void
 setup_kernel(curandState *state)
{
  int id = threadIdx.x - blockIdx.x *
 64;
  // each thread gets same seed
  curand_init(1234, id, 0,
 &state[id]);
}

__global__ void generate_kernel(
  curandState *state, int *result)
{
  int id = threadIdx.x + blockIdx.x *
 64;
  int count = 0;
  unsigned int x;

  curandState localState = state[id];
```

```
  // generate pseudo-random unsigned
  for (int n = 0; n < 1000000; n++) {
    x = curand(&localState);
  }

  // copy state back to global memory
  state[id] = localState;

  // store results
  result[id] += count;
}
```

CUDPP

- CUDPP: CUDA Data Parallel Primitives library
- collection of many *data-parallel* algorithms:
  - prefix-sum (“scan”)
  - parallel sort
  - reduction
- Important building blocks for a wide variety of data-parallel algorithms, including sorting, stream compaction, and building data structures such as trees and summed-area tables
- provides primitives and other complex operation functions such as:
  - hash table
  - array compaction
  - tridiagonal linear system solver
  - sparse matrix-vector product
- Specifications
  - open source project in C/C++
  - Support for Windows, Linux and OSX
  - open source: http://cudpp.github.io/

MAGMA: Matrix Algebra on GPU and Multicore Architectures

- LAPACK (Linear Algebra PACKage) is the de facto standar linear algebra operations
  - **built on BLAS**
- MAGMA is essentially a re-implementation of standard legacy LAPACK on heterogeneous architectures such as GPU + multicore CPU
  - **built on topo of cuBLAS**
- MAGMA 1.x support multi-GPU CUDA enabled environment, and its able to overlap computation on CPU cores (essentially through optimized multithreaded version of BLAS and LAPACK for the CPU side)
- Developed by the ICL group (Innovative Computing Laboratory) + many external collaborations + users community
- open source: http://icl.cs.utk.edu/projectsfiles/magma/
- WARNING: memory data layout follow the FORTRAN (column-major) convention

MAGMA: C/C++ usage

- MAGMA is entirelly developed in C. So it's usage is very easy in a C/c++ code
- The library interface is just in one file:
  - `magma.h`
- user must explicitly manage memory on host and device using traditional CUDA runtime APIs

```
// Reduction of a symmetrix matrix into tridiagonal form
#include <cuda.h>  //
#include <magma.h> //

// magma_int_t magma_dsytrd( char uplo, magma_int_t n, double *A,
//                  magma_int_t lda, double *d, double *e,
//                  double *tau, double *work, magma_int_t *lwork,
//                  double *da, double *dc, magma_int_t *info);

cudaError_t stat;
double *da, *dwork;
stat = cudaMalloc((void**)&da, n*n*sizeof(double));
stat = cudaMalloc((void**)&dwork, workSize* sizeof(double));
magma_dsytrd('U', n, A, lda, diagonal, offdiagonal, tau, work, lwork, da, dwork,
     &info)
```

MAGMA: F90/2003 usage

- In order to use MAGMA with F90/2003 requires the programmer to provide `interface` and the `ISO_C_BINDING` module

```
!! Native C interface:
!!  magma_int_t magma_dsytrd( char uplo, magma_int_t n, double *A,
!!                  magma_int_t lda, double *d, double *e,
!!                  double *tau, double *work, magma_int_t *lwork,
!!                  double *da, double *dc, magma_int_t *info);
!!
!! Interface for F90/2003:
subroutine magma_dsytrd(uplo, n, a, lda, d, e, tau, work, lwork, da, dc, info)

  bind(C, name="magma_dsytrd")
  use iso_c_binding
  implicit none

  character, value:: uplo
  integer(C_INT), value :: n, lda
  integer(C_INT) :: info, lwork
  type(C_PTR), value :: a, d, e, tau, work, da, dc

  ! NB: type(C_PTR), value == void*
end subroutine magma_dsytrd
```

CUDA Thrust

A C++ template library for CUDA

- Mimics the C++ STL

- Two containers
  - Manage memory on host and device:

    thrust::host_vector<T>

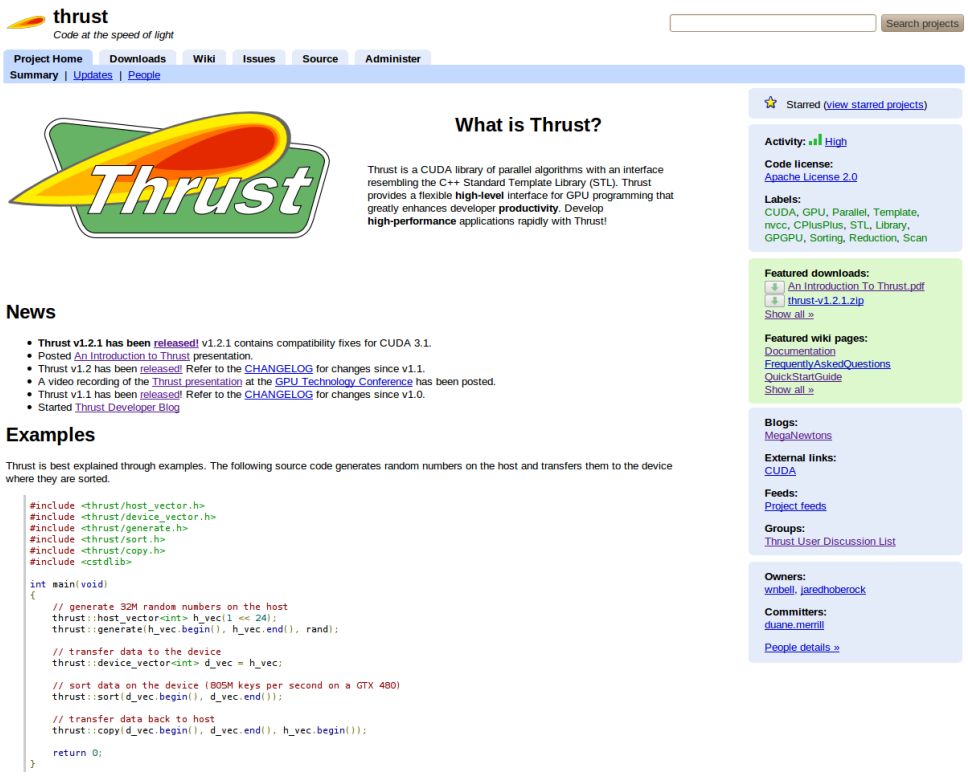
    thrust::device_vector<T>

- Algorithms
  - Sorting, reduction, scan, etc:

    thrust::sort()

    thrust::reduce()

    thrust::inclusive_scan()
  - act on ranges of the container data by pair of iterators (a sort of pointers)



CUDA Thrust

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>

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

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device (846M keys per second on GeForce GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

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

    return 0;
}
```

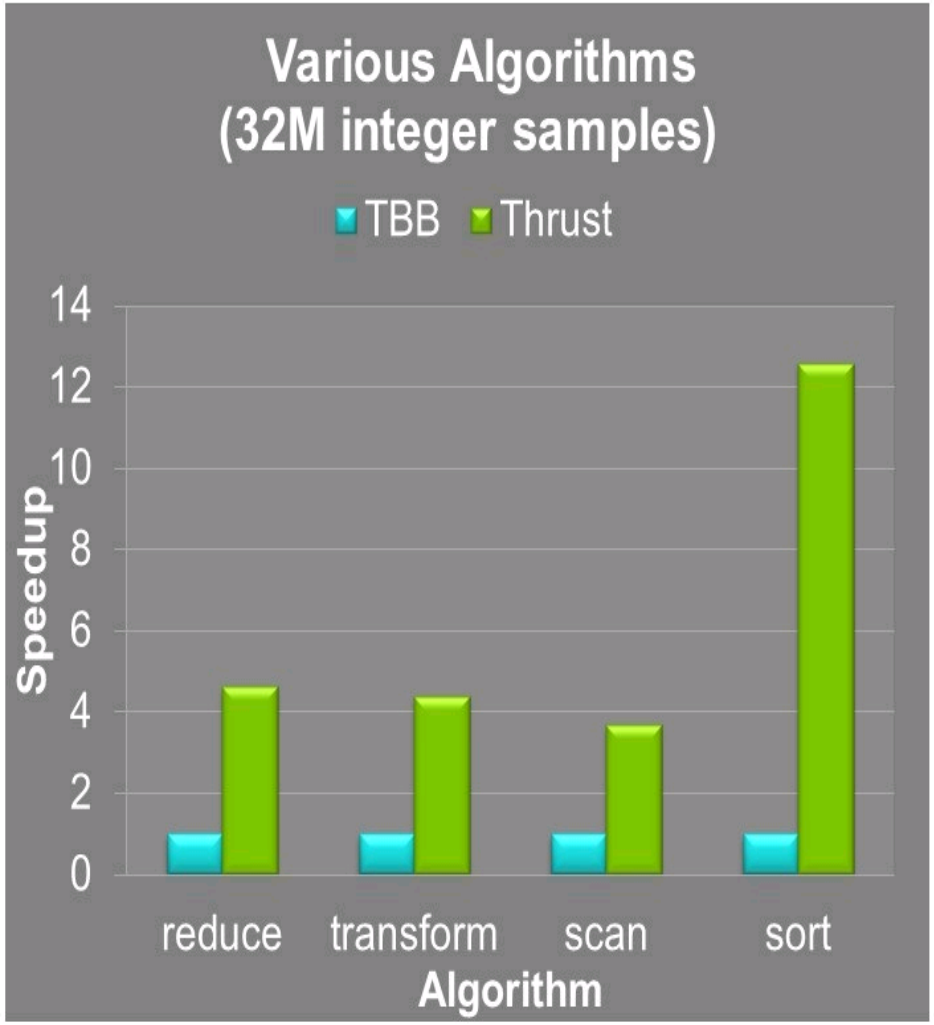
CUDA Thrust

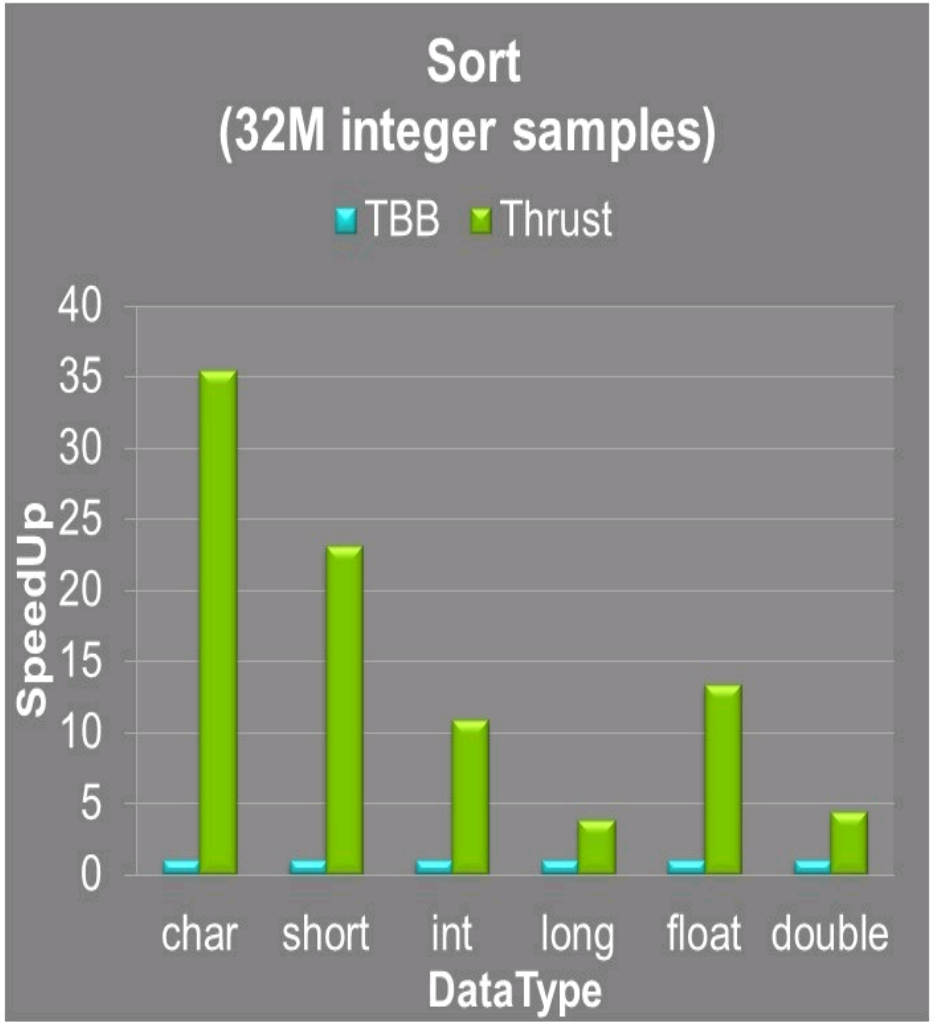
```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

int main(void)
{
  // generate random data on the host
  thrust::host_vector<int> h_vec(100);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device and compute sum
  thrust::device_vector<int> d_vec = h_vec;
  int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
  return 0;
}
```

CUDA Thrust





- CUDA 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 ver. and motherboard config.

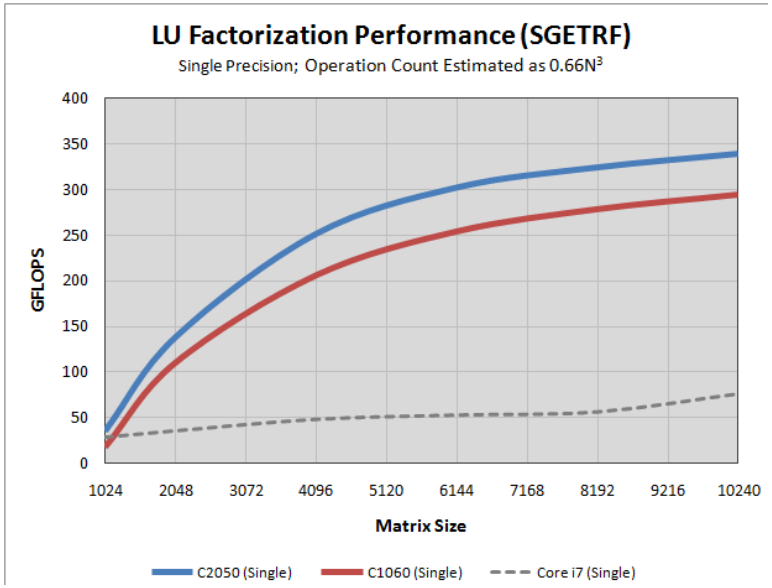
Lapack for CUDA: CULA Library

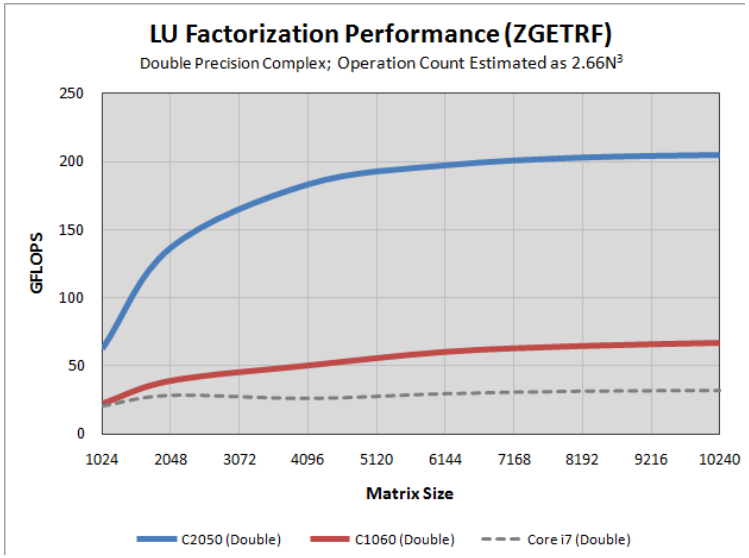
http://www.culatools.com

Proprietary library that implements the LAPACK in CUDA, which is available in several versions.

The speed-up of the picture on the right refers to:

CPU: Quad-core Intel Core i7 930 @ 2.8 GHZ CPU
GPU: NVIDIA Tesla C1060
GPU: NVIDIA Tesla C2050

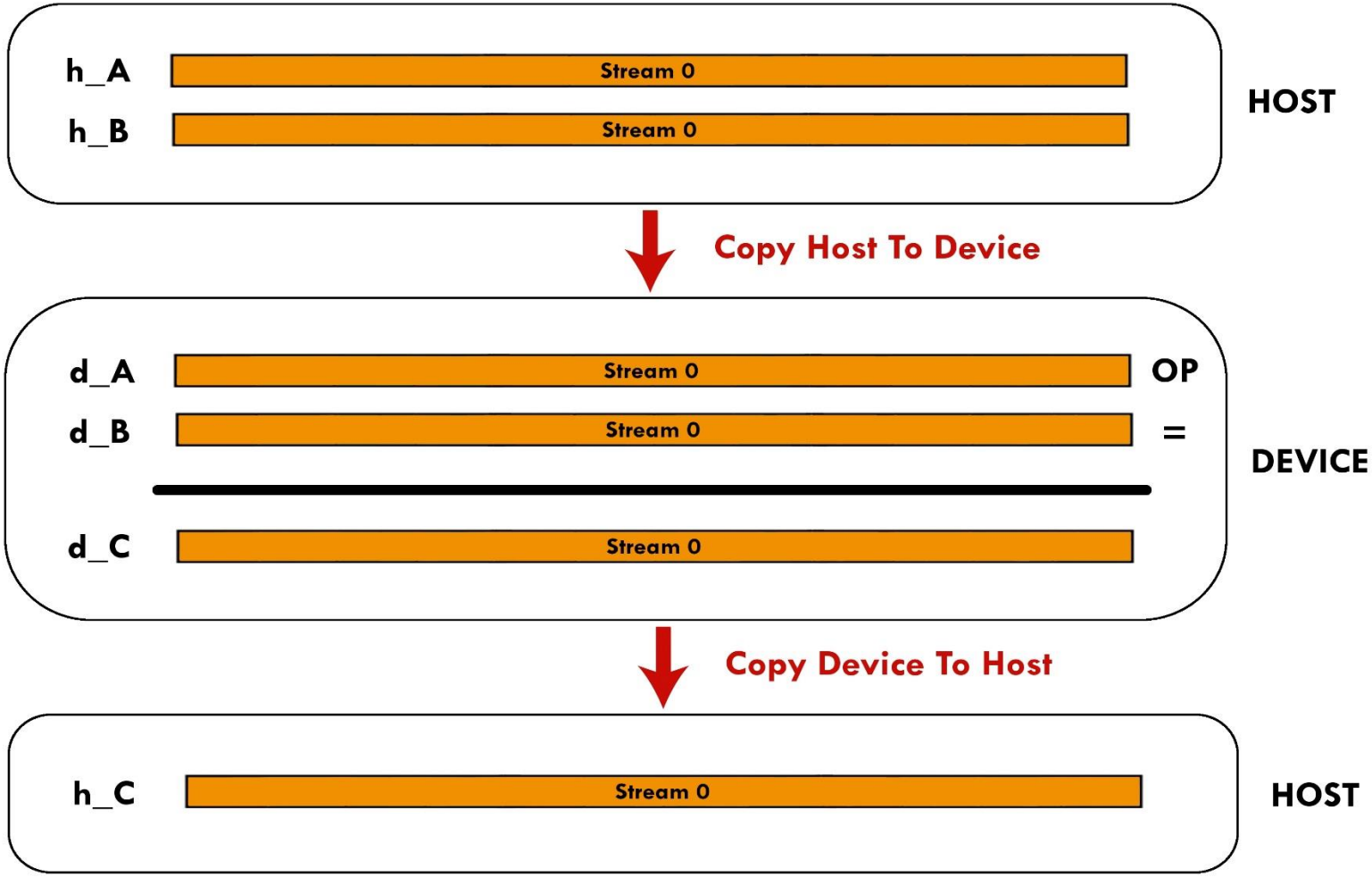




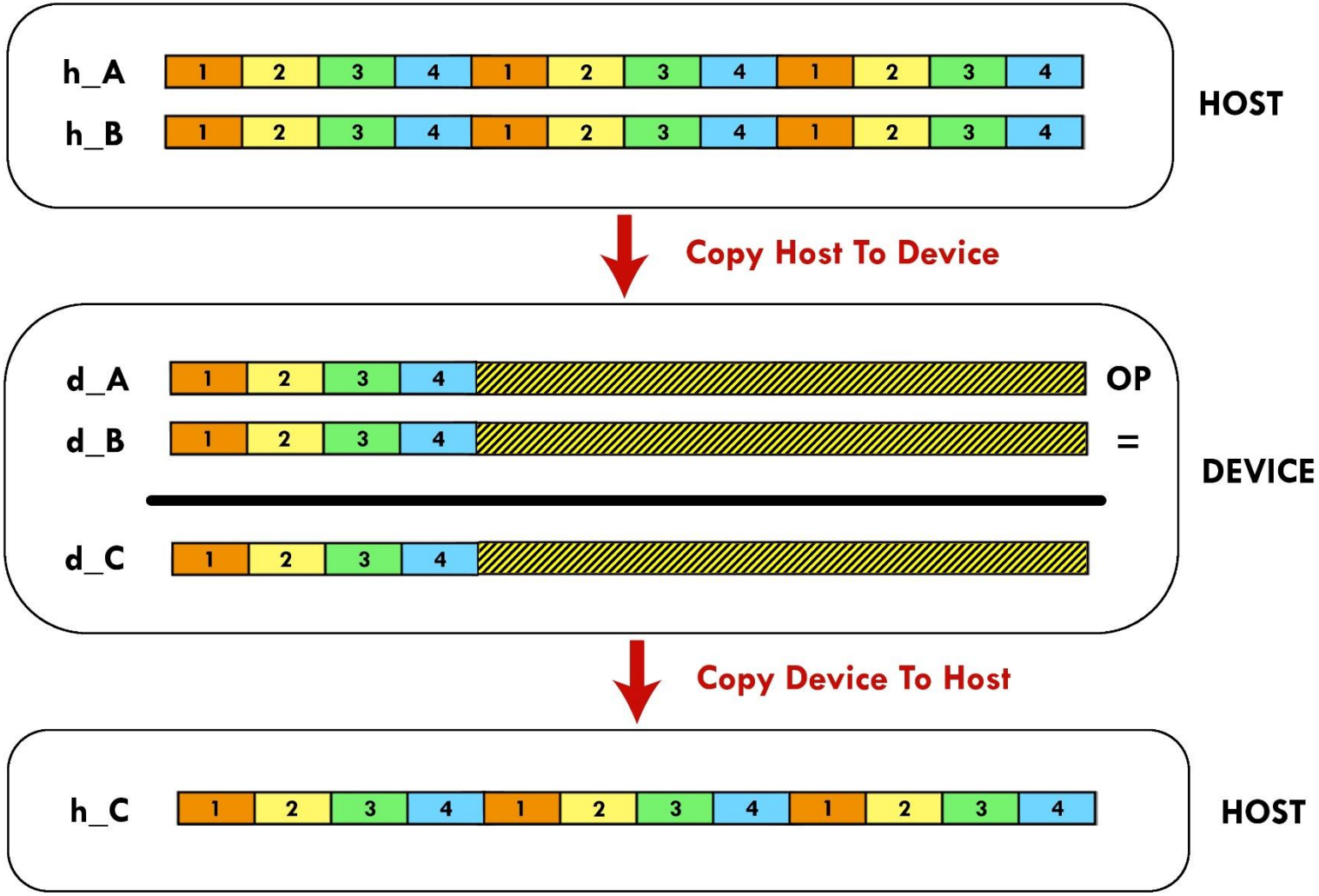
Bibliography

- ✓ CUDA C Programming Guide
- ✓ PGI CUDA fortran, http://www.pgroup.com/doc/pgicudaforug.pdf
- ✓ CUDA C Best Practices Guide, Optimization Guide
- ✓ NVIDIA CUDA Library Documentation (Doxygen –generated Reference Manual)
- ✓ Tuning CUDA Applications for Fermi, Kepler, Maxwell, Pascal, etc
- ✓ Kirk and Hwu, Programming Massively Parallel Processors
- ✓ CUDA by example, http://developer.nvidia.com/object/cuda-by-example.html
- ✓ P. Micikevicius, Fundamental and Analysis-Driven Optimization, GPU Technology Conference 2010 (GTC 2010)
- ✓ V. Volkov, Benchmarking GPUs to tune dense linear algebra
- ✓ V. Volkov, Better performance at lower occupancy, GPU Technology Conference 2010 (GTC 2010)
- ✓ J. Dongarra et al. "An Improved MAGMA GEMM for Fermi GPUs"

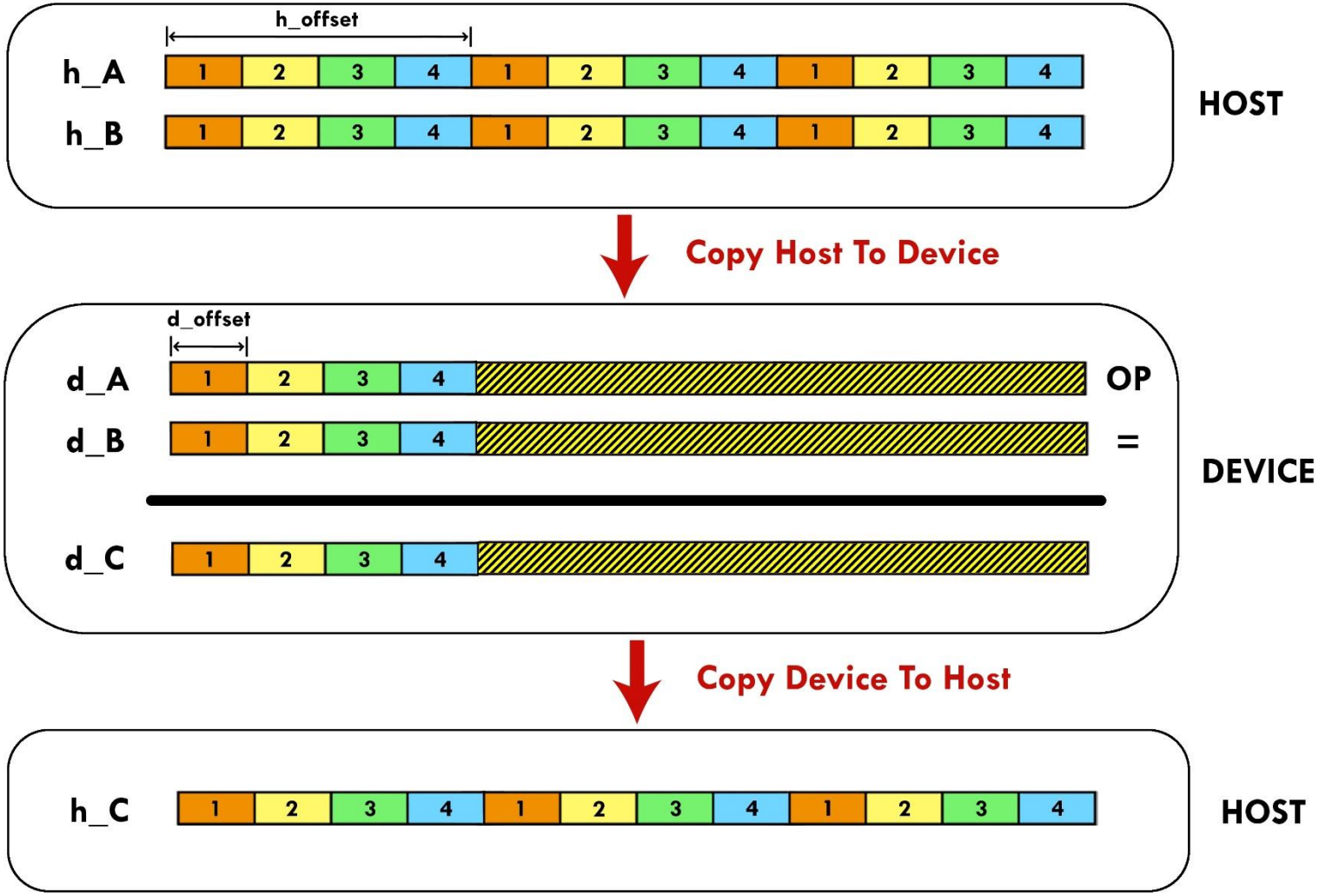
Excercise 1: naive version



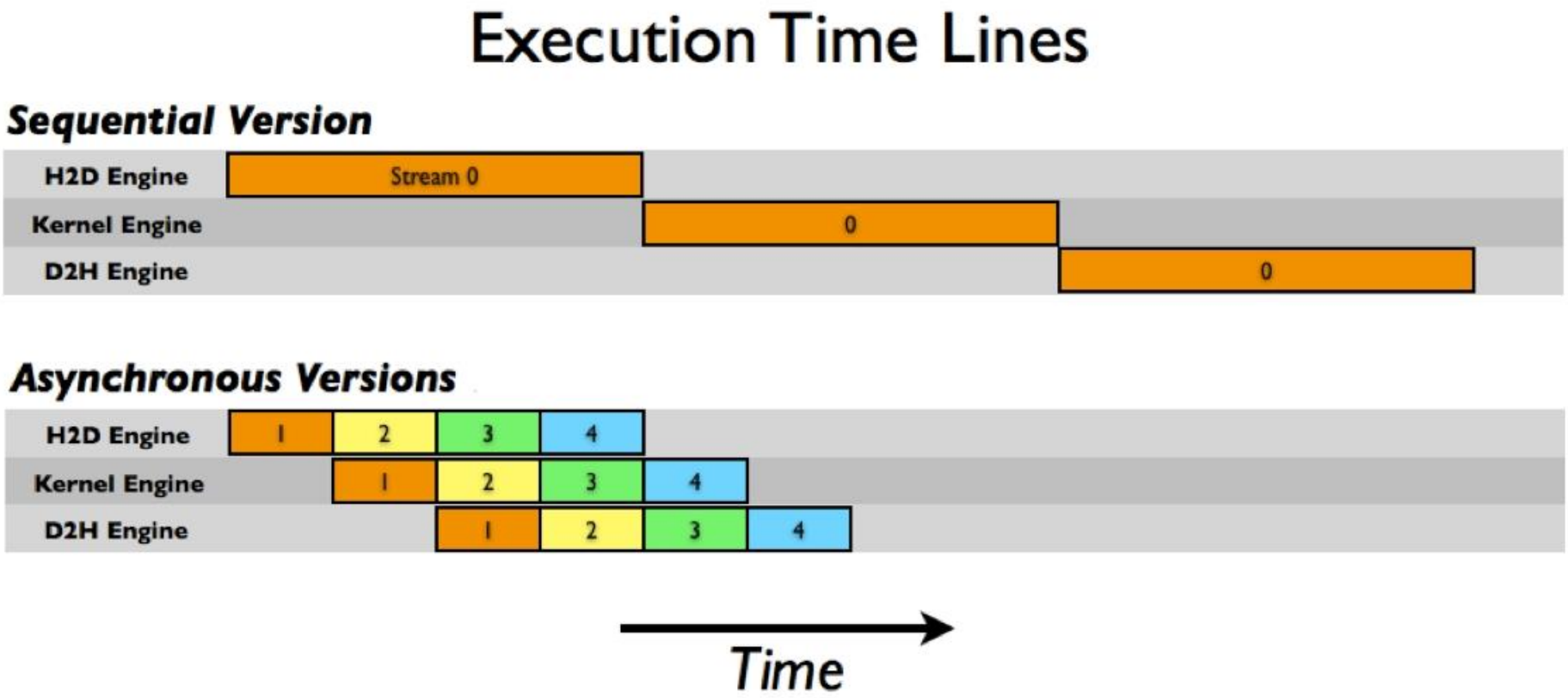
Exercixe 2: cudaStream version



Excercise 2: cudaStream version



Excercise 2: cudaStream version



Excercise 2: CUDA runtime functions

You need to use the following functions (C for CUDA):

- `cudaError_t cudaStreamCreate(cudaStream_t *stream)`
- `cudaError_t cudaStreamDestroy(cudaStream_t stream)`
- `cudaError_t cudaDeviceSynchronize(void)`
- `cudaErrot_t cudaMemcpyAsync(void* dst, void* src, size_t nbyte, enum cudaMemcpyKind kind,cudaStream_t stream)`

You need too use the following functions (CUDA FORTRAN):

- `integer function cudaStreamCreate(stream)`
  `integer :: stream`
- `integer function cudaStreamDestroy(stream)`
  `integer :: stream`
- `integer function cudaDeviceSynchronize()`
- `integer function cudaMemcpyAsync(dst, src, nelements, kind, stream)`