

# Introduction to Scientific Programming using GPGPU and CUDA



Day 2

***Luca Ferraro***

[l.ferraro@Cineca.it](mailto:l.ferraro@ Cineca.it)

***Sergio Orlandini***

[s.orlandini@Cineca.it](mailto:s.orlandini@ Cineca.it)

## ■ Tools del CUDA-Toolkit

- Profiler
- CUDA-GDB
- CUDA-memcheck
- Parallel NSight

## ■ Librerie CUDA-Enabled

- CUBLAS
- CUFFT
- CUSPARSE
- CURAND
- MAGMA, THRUST, CUDDP, ...

## ■ Esercizi



# Profiling tools: built-in

- Il CUDA toolkit mette a disposizione degli utili strumenti di profiling

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

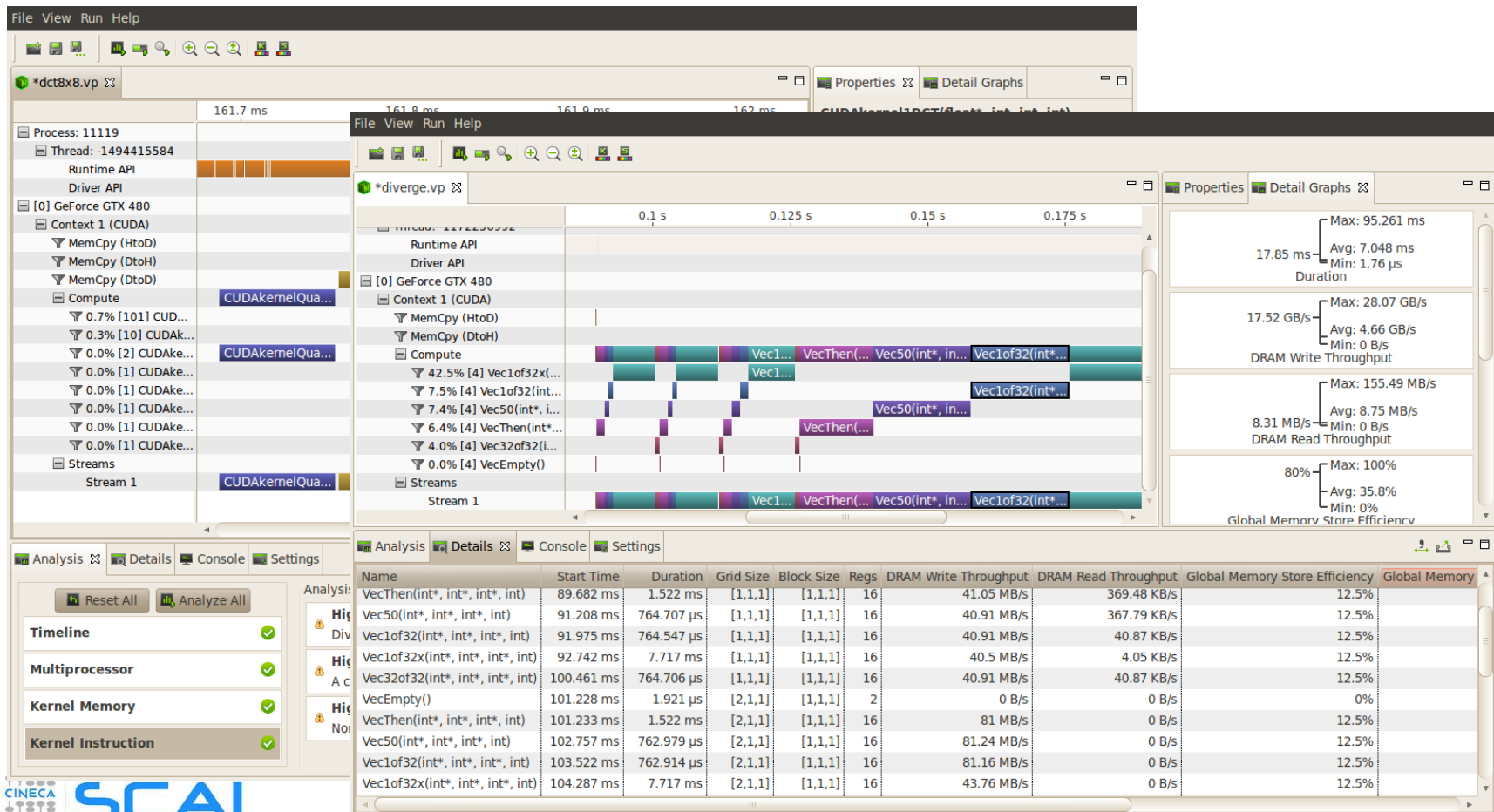
```
// Contents of config
gld_coherent
gld_incoherent
gst_coherent
gst_incoherent
```

```
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=[ memcpy ] 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=[ memcpy ] gputime=[ 349.344 ]
```

# Profiling: Visual Profiler

- Traccia su HOST le chiamate al driver, trasferimenti, kernel e mostra gli overlap tra stream di esecuzione differenti
- Analisi automatica delle prestazioni (hardware counters)



# Debugging: CUDA-GDB

## ■ Debugger tradizionale gdb con estensioni CUDA

```
(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 0x00000000000866400
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

- Tool capace di controllare buffer overflows, misaligned global memory accesses e memory leaks
- Richiamabile anche da 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 disponibile per diversi IDEs (Eclipse, VisualStudio)
- Aggrega diverse funzionalità:
  - Debugger (fully integrated)
  - Visual Profiler
  - Memory correctness checker
- Come plug-in, estende le funzionalità degli IDE per CUDA
- Tool in veloce evoluzione che consente:
  - Debug remoto e profiling
  - PTX assembly view,
  - warp inspector
  - expression lamination

# Parallel NSight

voxelpipe\_demo\_vc10 (Debugging) - Microsoft Visual Studio (Administrator)

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

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

Connections: localhost

### CUDA Info 1

Current	blockIdx	Warp Index	PC	Active Mask	Status	Exception	File Name	Source Lin	Lanes
	( 0, 0, 0)	0	0x003e1ad8	0xffffffff80	Breakpoint	None	rt_render.cu	163	
	( 0, 0, 0)	1	0x003e1ad8	0xffffffff80	Breakpoint	None	rt_render.cu	163	
	( 0, 0, 0)	2	0x003e1ad8	0xffffffffc0	Breakpoint	None	rt_render.cu	163	
	( 0, 0, 0)	3	0x003e1ad8	0xffffffff80	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 WarpWatch 1

Name	ray_inv.x	ray_inv.y	ray...
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...

### Disassembly

Address: 148: const uint32 leaf\_index = node.get\_index(<

```
0x003e1298 2800400010019de4 MOV R6, c[0x0][0x4];
0x003e12a0 28000000f0c01dde4 MOV R7, RZ;
0x003e12a8 28000000f0c01dde4 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 = 24}	__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



# 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 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.100848322 3 14 15 16 17 18 19 2 21 22 23 24 25 26 27 28 29 3 31 32 33 34 35 36 37 38 39 4

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 CPU 725  
0x3C2BCC0 725

Device Context 0x3C2BCC0 Level 0

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-It4\c\program files (x86)\nvidia corporation\nvidia demos\supersonic sled\bin\supersonicsled.exe"  
Exit Status 259

Output Find Results 1

Ready

Server Explorer  
Toolbox  
App Profiler Session Explorer  
Solution Explorer  
Team Explorer  
Class View  
Properties

# Librerie CUDA

- CUDA include librerie matematiche di uso comune:
  - **CUBLAS**: Basic Linear Algebra Subprograms
  - **CUFFT** : Fast Fourier Transform
  - **CUSPARSE**: algebra lineare per matrici sparse
  - **CURAND**: numeri pseudorandom e quasirandom
- Altre librerie matematiche disponibili per CUDA :
  - **MAGMA** (Matrix Algebra on GPU and Multicore Architectures)  
<http://icl.cs.utk.edu/magma/>
  - **THRUST** (CUDA library for parallel algorithms in C++)  
<http://code.google.com/p/thrust/>
  - **CUDPP** (Data Parallel Primitives): parallel prefix-sum, sort, reduction  
<http://code.google.com/p/cudpp/>
  - **CULA**: Lapack, free solo in singola precisione  
<http://www.culatools.com/contact/cuda-training/>
  - **NPP** (Nvidia Performance Primitives) : image e video processing, funzioni statistiche (media, deviazione standard ...)  
[http://developer.nvidia.com/object/npp\\_home.html](http://developer.nvidia.com/object/npp_home.html)

# CUDA Math Library

- simply by 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`
- Additional Inverse Error Functions
  - `erfinv`, `erfcinv`
- Optimized Reciprocal Functions
  - `rsqrt`, `rcbrt`
- Floating Point Data Attributes
  - `signbit`, `isfinite`, `isinf`, `isnan`
- Bessel Functions
  - `j0`, `j1`, `jn`, `y0`, `y1`, `yn`

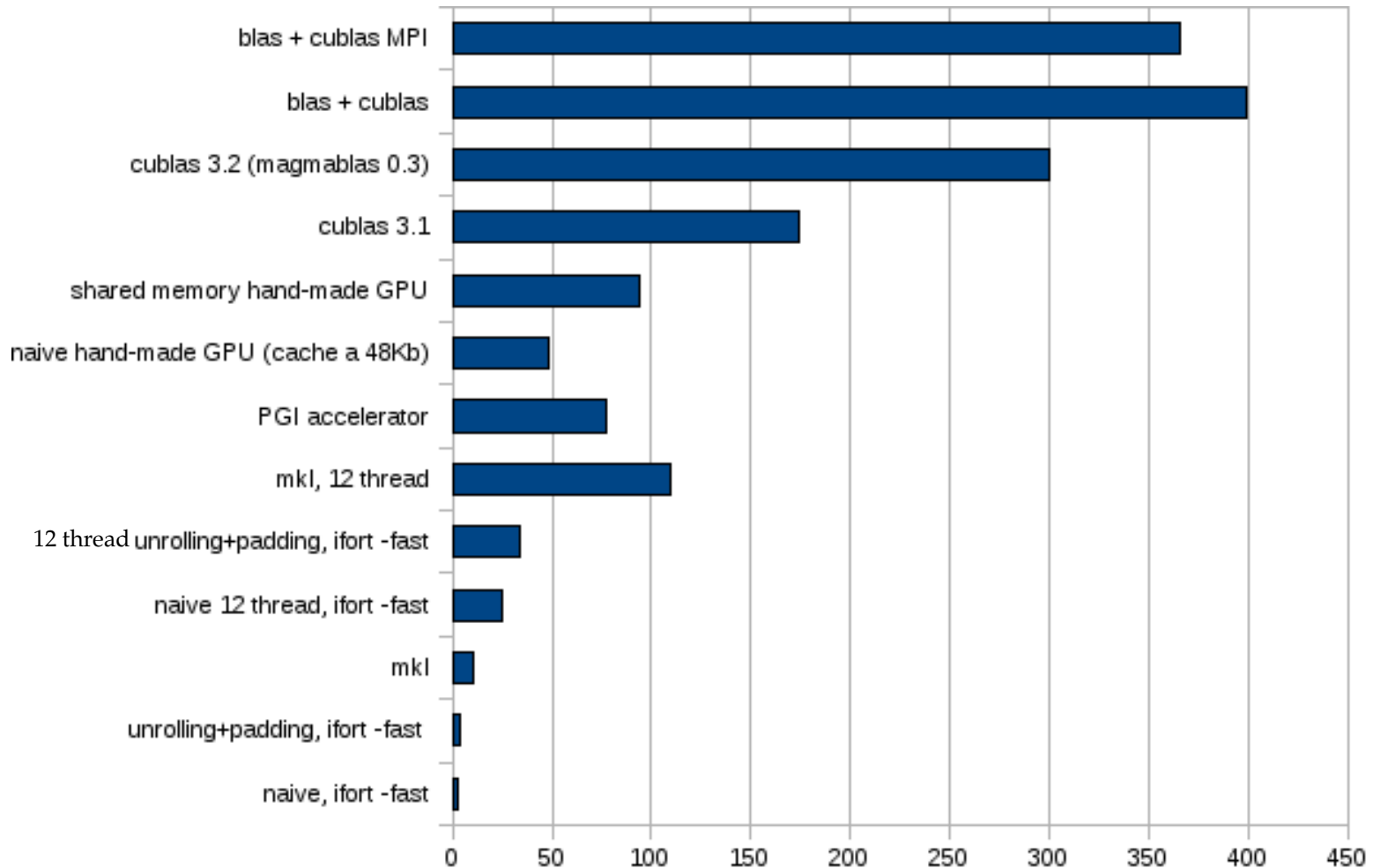
# CUBLAS: CUDA Basic Linear Algebra Subprograms

- La libreria BLAS è lo standard accettato per quanto riguarda accuratezza e interfaccia delle librerie che implementano operazioni base di algebra lineare
  - BLAS Level 1:  $\mathbf{y} \leftarrow \alpha\mathbf{x} + \beta\mathbf{y}$   $\mathcal{O}(N)$
  - BLAS Level 2:  $\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$   $\mathcal{O}(N^2)$
  - BLAS Level 3:  $\mathbf{C} \leftarrow \alpha\mathbf{A}\mathbf{B}\mathbf{x} + \beta\mathbf{C}$   $\mathcal{O}(N^3)$
- Sono disponibili implementazioni vendor delle BLAS ottimizzate per diverse architetture
- La libreria CUBLAS è un'implementazione delle BLAS basata sul CUDA runtime
- È sviluppata e supportata direttamente da NVIDIA ed è inclusa nel CUDA Toolkit
- Le funzioni CUBLAS supportano il calcolo su una singola GPU (le routine CUBLAS non sono in grado di auto-parallelizzare il calcolo su più GPU)
- Il layout dei dati segue la convenzione FORTRAN (column-major)
- È responsabilità dell'utente la gestione della memoria sul *device*
- La libreria CUBLAS mette a disposizione delle helper function per semplificare la gestione del CUBLAS runtime e il trasferimento dei dati host/device

# CUBLAS: DGEMM performance

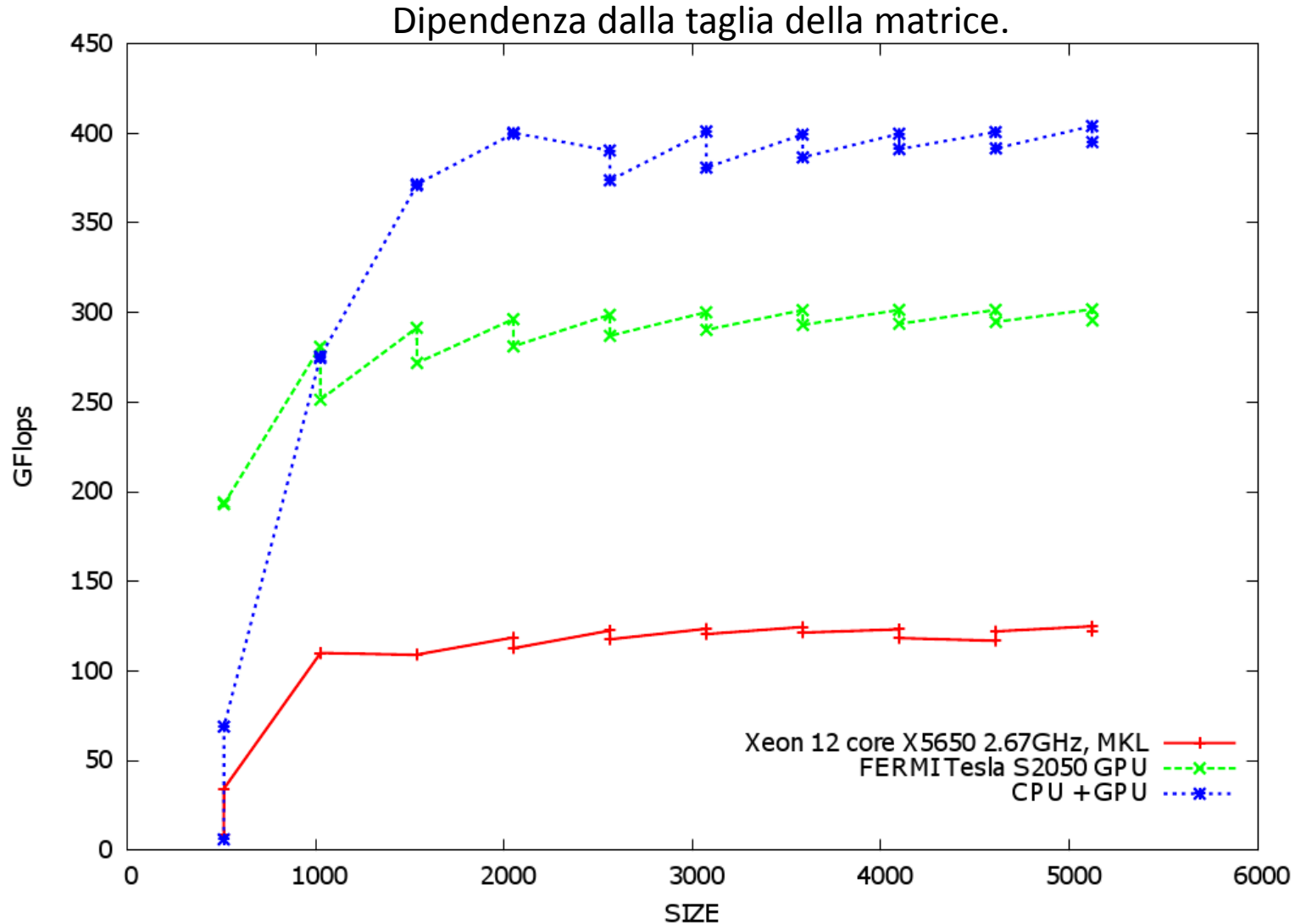
- Performance cluster con FERMI: prodotto matrice-matrice:

size 4096x4096; doppia precisione



# CUBLAS: DGEMM performance

- Performance DGEMM cluster FERMI: prodotto matrice-matrice



# CUFFT

- CUFFT è la libreria per la Fast Fourier Transform fornita da NVIDIA
- FFT basata sugli algoritmi Cooley-Turkey e Bluestein
- L' API fornita è simile all' interfaccia avanzata di FFTW
  - analogamente a FFTW, CUFFT usa il concetto di *workplan*
  - Una volta che è stato creato un *workplan*, la libreria mantiene le informazioni necessarie ad eseguire più volte il piano senza ricalcolare la configurazione
  - Modello molto efficace perché i diversi tipi di FFT hanno bisogno di diverse configurazioni di thread e risorse GPU
  - attenzione, CUFFT segue la convenzione C per il layout in memoria delle matrici (row-major)
- Altre caratteristiche chiave:
  - trasformate 1D, 2D, 3D per tipi di dato reale e complesso
  - supporto a dati in singola e doppia precisione
  - supporto all' esecuzione asincrona e in stream
  - thread-safe (**CUDA 4.1**)

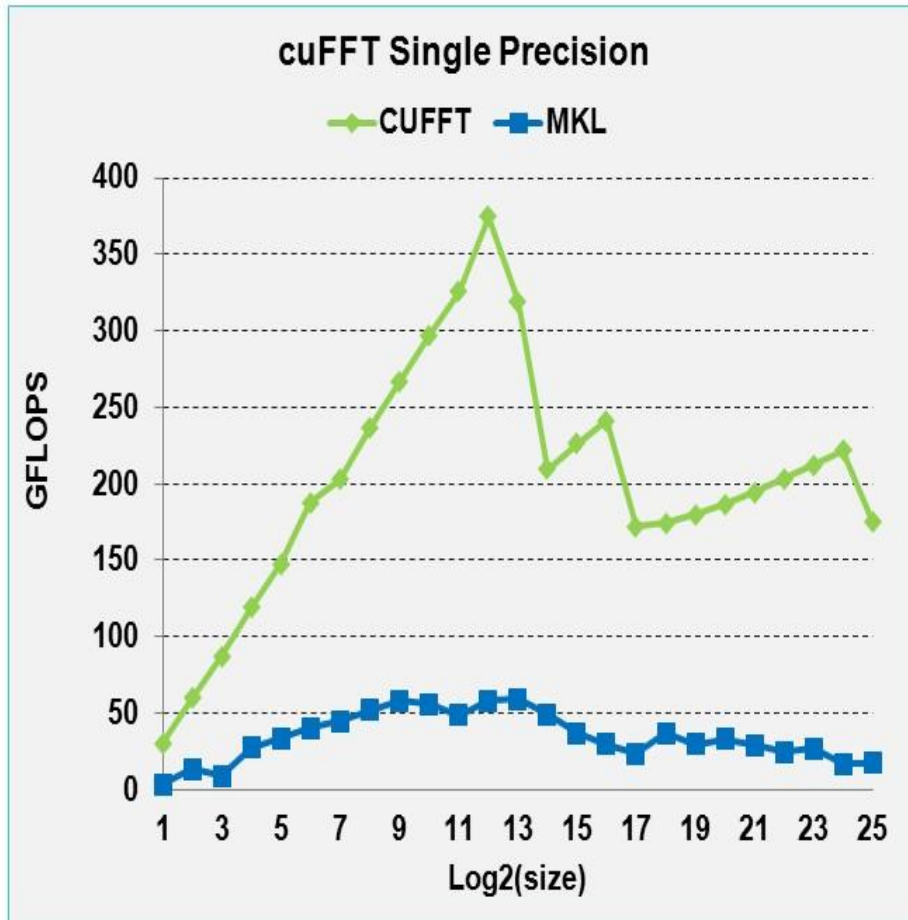
# CUFFT esempio: trasformata 2D complex-complex

```
#define NX 256
#define NY 128

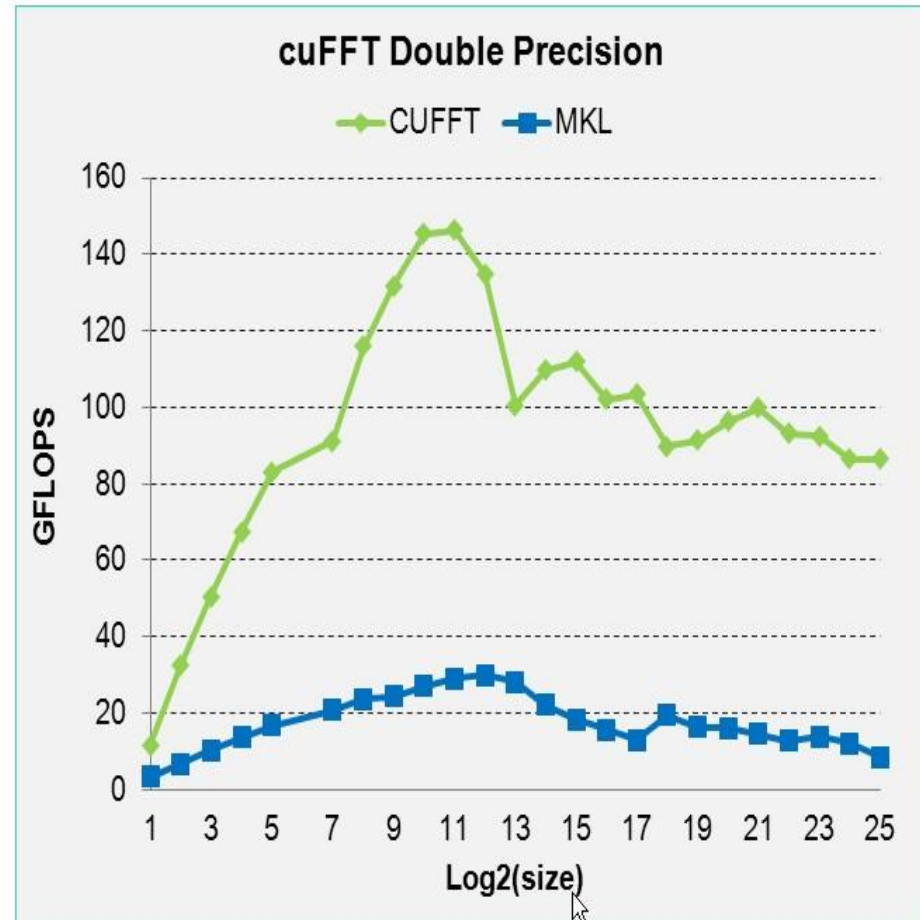
cufftHandle plan;
cufftComplex *idata, *odata;
cudaMalloc((void**) &idata, sizeof(cufftComplex) * NX * NY);
cudaMalloc((void**) &odata, sizeof(cufftComplex) * NX * NY);
...
/* Crea un piano FFT 2D */
cufftPlan2d(&plan, NX, NY, CUFFT_C2C);
/* Usa il piano CUFFT per trasformare il segnale "out of place"
*/
cufftExecC2C(plan, idata, odata, CUFFT_FORWARD);
/* Trasformata inversa del segnale "in place" */
cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);
/* Puntatori ad input ed output diversi implicano trasformate
"out of place" */
/* Elimina il piano CUFFT */
cufftDestroy(plan);
cudaFree(idata), cudaFree(odata);
```



# CUFFT: prestazioni FFT1D



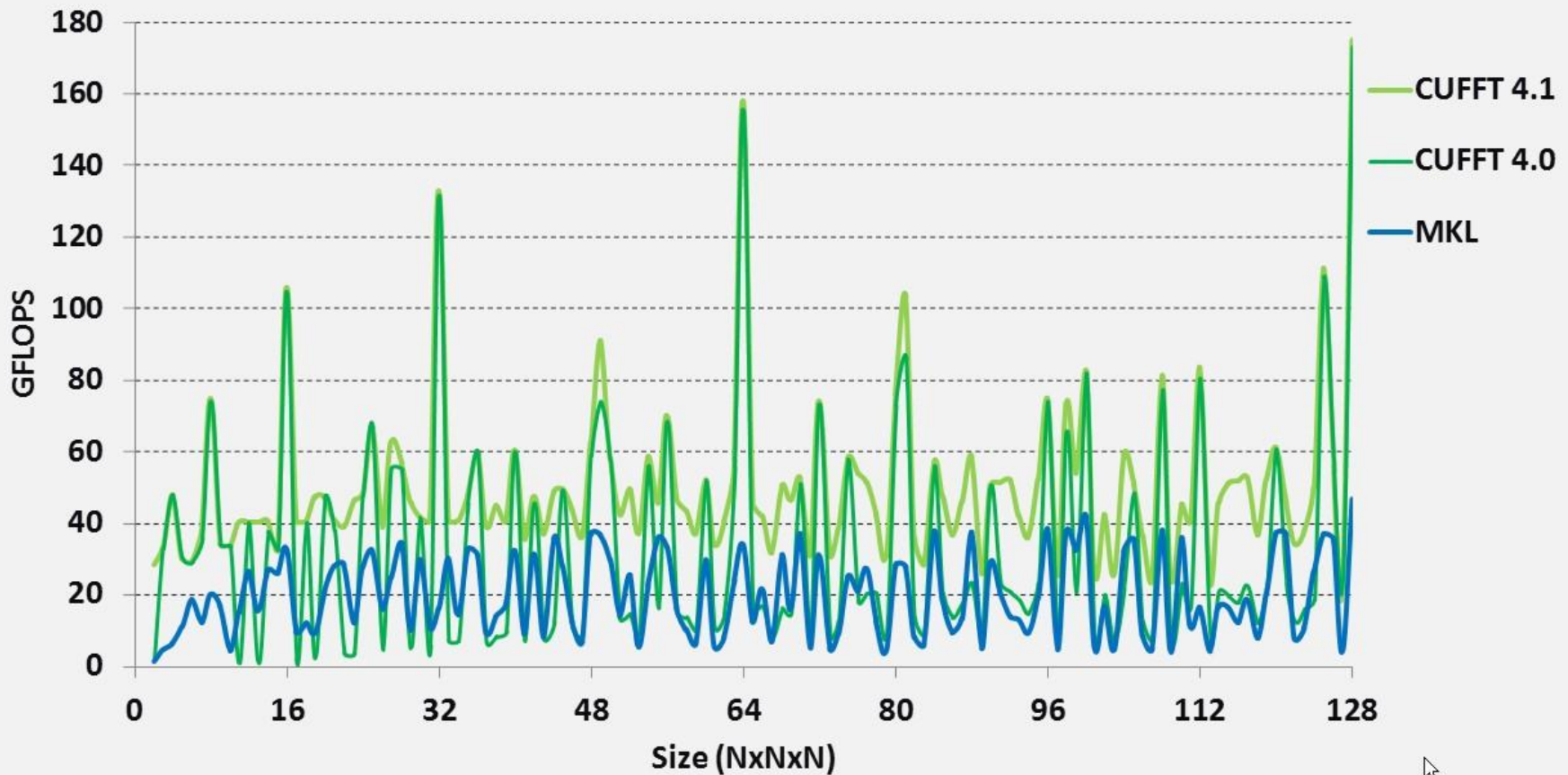
- Measured on sizes that are exactly powers-of-2
- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz



- 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: prestazioni FFT3D

Single Precision All Sizes 2x2x2 to 128x128x128

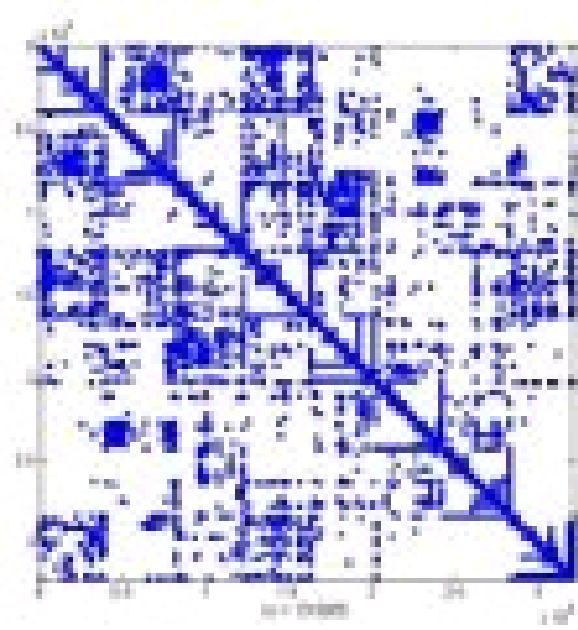


- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

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

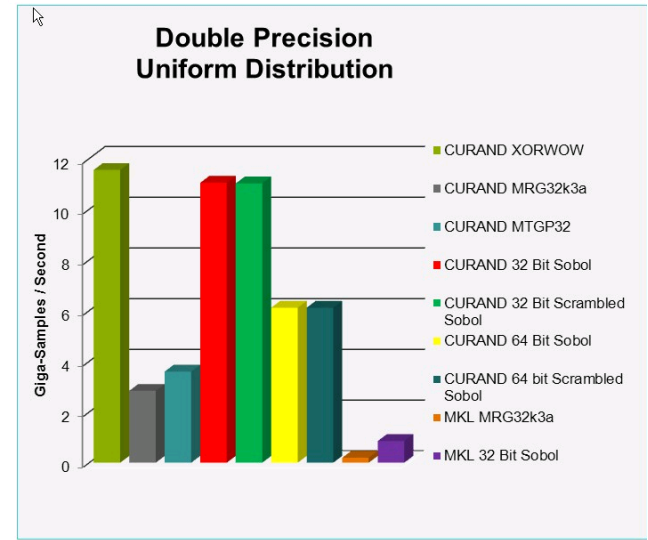
# CuSPARSE

- Supports 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

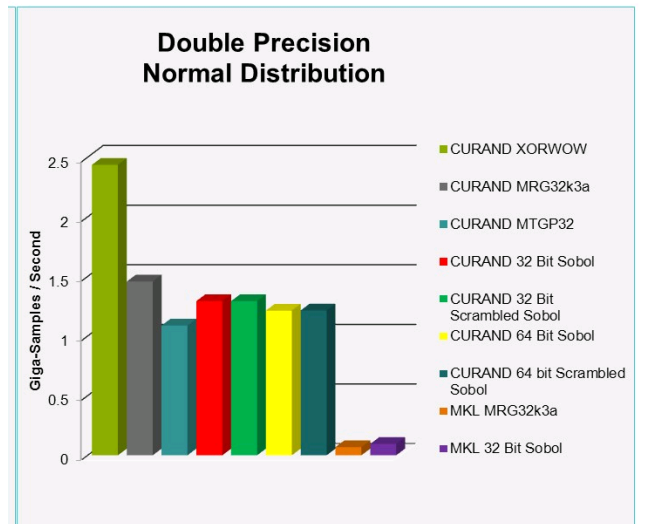


# CuRAND

- **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 n floats on device
cudaMalloc((void **) &devData, n *
    sizeof(float));

// Create pseudo-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;

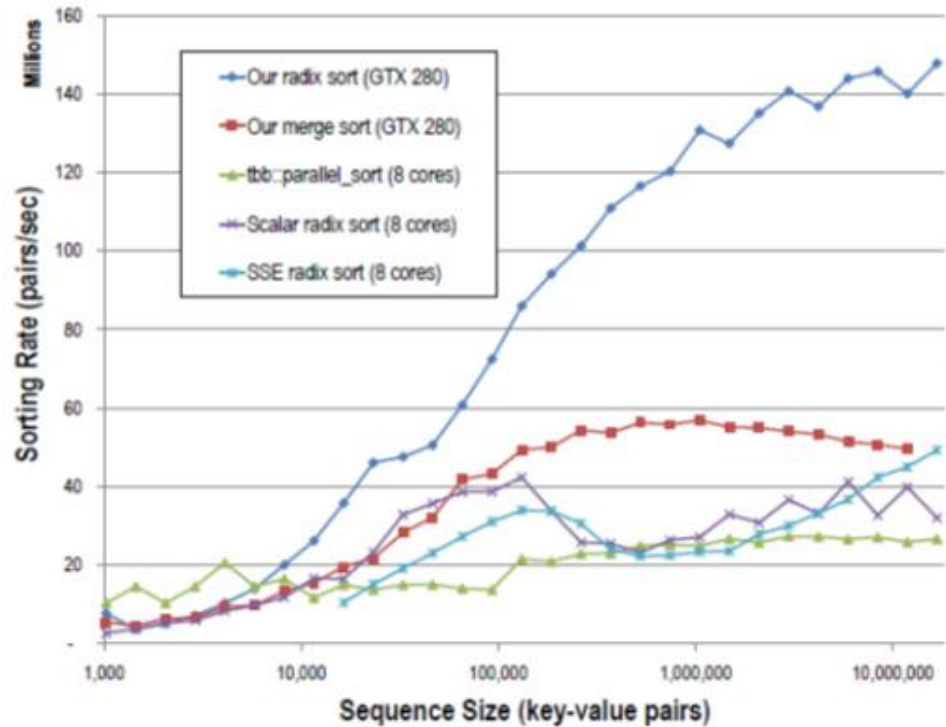
// store results
result[id] += count;
}
```

# CUDPP 2.0

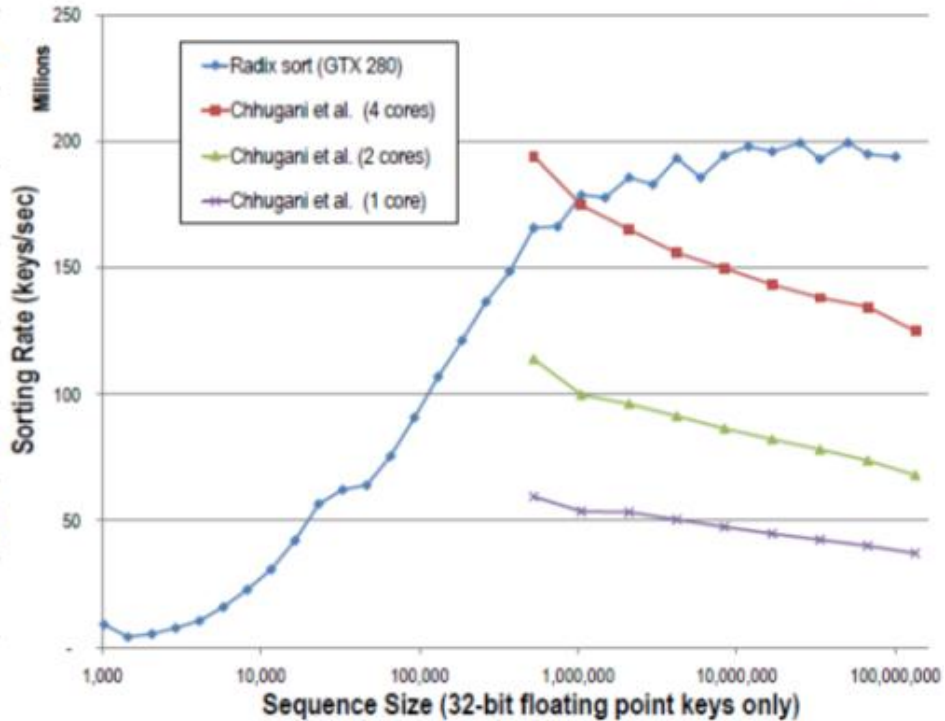
- CUDPP: CUDA Data Parallel Primitives library
- Libreria di algoritmi *data-parallel* per:
  - prefix-sum (“scan”)
  - parallel sort
  - reduction
- Operazioni complesse costruite sulle primitive:
  - hash table
  - array compaction
  - solutore di sistemi lineari tridiagonali
  - moltiplicazione matrice-sparsa/vettore
- Caratteristiche
  - Scritta in C/C++, disponibile API in C
  - Supporto per Windows, Linux e OSX
  - open source: <http://code.google.com/p/cudpp/>



# CUDPP 2.0: prestazioni



(a) 8-core Clovertown (key-value pairs)



(b) 4-core Yorkfield (float keys only)

# MAGMA: Matrix Algebra on GPU and Multicore Architectures

- La libreria LAPACK (Linear Algebra PACKage) è lo standard accettato per quanto riguarda accuratezza e interfaccia delle librerie che implementano operazioni complesse di algebra lineare
  - **Sono basate su BLAS**
- MAGMA è essenzialmente una re- implementazione di LAPACK su architetture eterogenee GPU + CPU multicore
  - **Sono basate su CUBLAS** e un sotto insieme di routine BLAS ottimizzate per GPU Fermi e Tesla sviluppate internamente al progetto MAGMA (`magma_blas`)
- MAGMA 1.x supporta il calcolo su più GPU CUDA enabled e CPU core (essenzialmente mediante versioni vendor multithreaded di BLAS e LAPACK)
- É sviluppata da ICL group (Innovative Computing Laboratory) + collaboratori esterni + user community
- sono open source: <http://icl.cs.utk.edu/projectsfiles/magma/>
- Il layout dei dati segue la convenzione FORTRAN (column-major)
- É responsabilità dell'utente la gestione della memoria sul *device* e dei necessari trasferimenti dati

# MAGMA: uso in C/C++

- MAGMA è sviluppata nativamente in C. Pertanto in ambiente C/C++ il suo utilizzo è immediato.
- l'interfaccia alla libreria è contenuta nel file:
  - `magma.h`
- è necessario allocare la memoria sul device usando il CUDA runtime

```
// Riduzione di una matrice simmetrica alla forma tridiagonale
// Interfaccia 'experimental'
#include <cuda.h> // occorrono le funzioni per il memory management
#include <magma.h> // uso l'interfaccia GPU 'experimental'
// 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: uso in F90/2003

- MAGMA è sviluppata nativamente in C. Pertanto in ambiente F90/2003 il suo utilizzo richiede almeno la scrittura di `interface` e uso del modulo `ISO_C_BINDING`
- è necessario allocare la memoria sul device usando il CUDA runtime. Saranno necessarie interfacce a `cudaMalloc/Free`, `cublasSetMatrix/cudaMemcpy`

```
!! Interfaccia nativa C:
!! 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);
!! Interfaccia 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

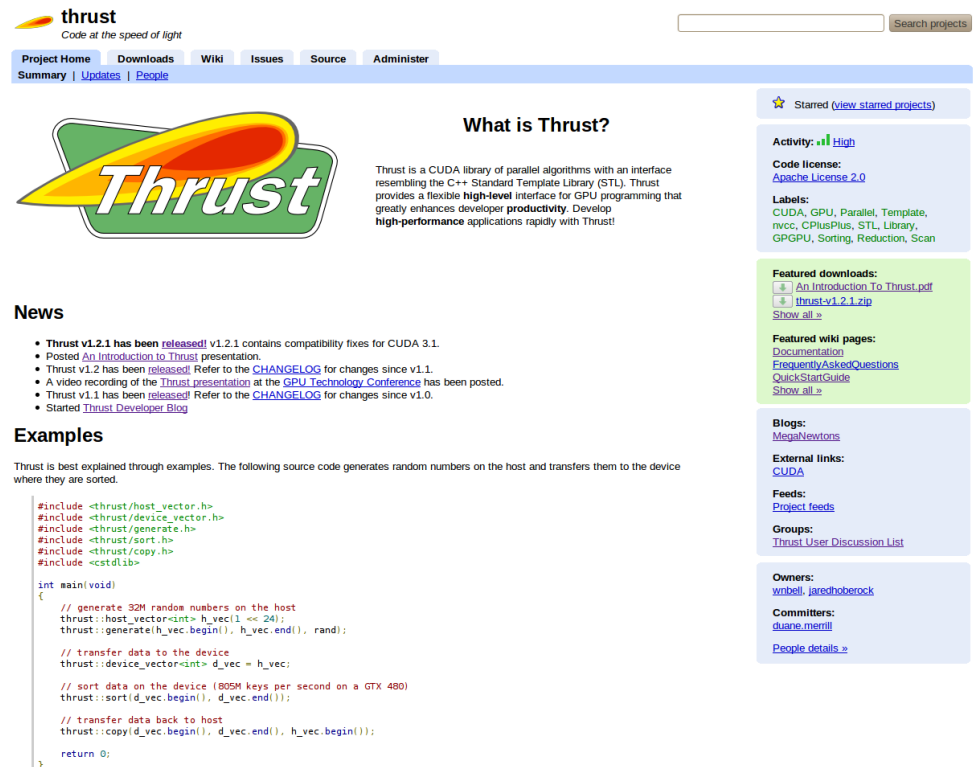
- Sorting, reduction, scan, etc:

`thrust::sort()`

`thrust::reduce()`

`thrust::inclusive_scan()`

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



The screenshot shows the Thrust website homepage. At the top, there's a navigation bar with links for Project Home, Downloads, Wiki, Issues, Source, and Administer. Below the navigation bar is a large logo for Thrust, which is a stylized orange and yellow flame shape with the word "Thrust" in white text. To the right of the logo is a section titled "What is Thrust?" which provides a brief description of the library. Below this is a "News" section with several bullet points about recent releases and updates. Further down is an "Examples" section with a code snippet showing how to use Thrust to generate random numbers on the host and transfer them to the device for sorting. On the right side of the page, there are several sidebar sections: "Stared (view starred projects)", "Activity: High", "Code license: Apache License 2.0", "Labels: CUDA, GPU, Parallel, Template, nvcc, CPlusPlus, STL, Library, GPGPU, Sorting, Reduction, Scan", "Featured downloads: An Introduction To Thrust.pdf, thrust-v1.2.1.zip", "Featured wiki pages: Documentation, Frequently Asked Questions, QuickStartGuide", "Blogs: MegaNewtons", "External links: CUDA", "Feeds: Project feeds", "Groups: Thrust User Discussion List", "Owners: wrbell, jaredhoberock", and "Committers: duane.merill, People details".

# 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>

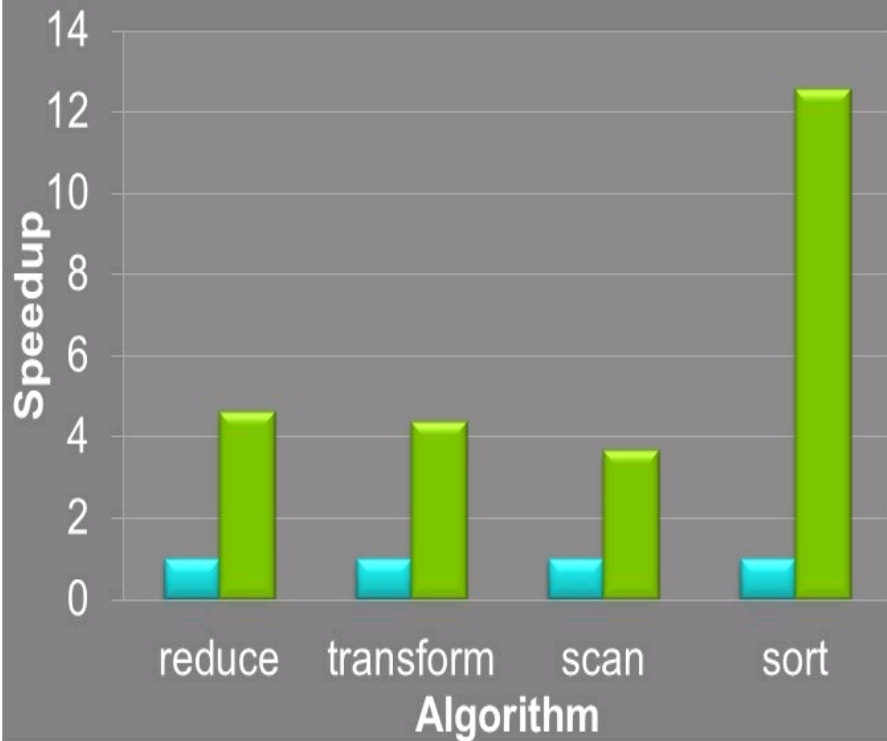
int main(void)
{
    // generate random data on the host
    thrust::host_vector<int> h_vec(100);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // 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

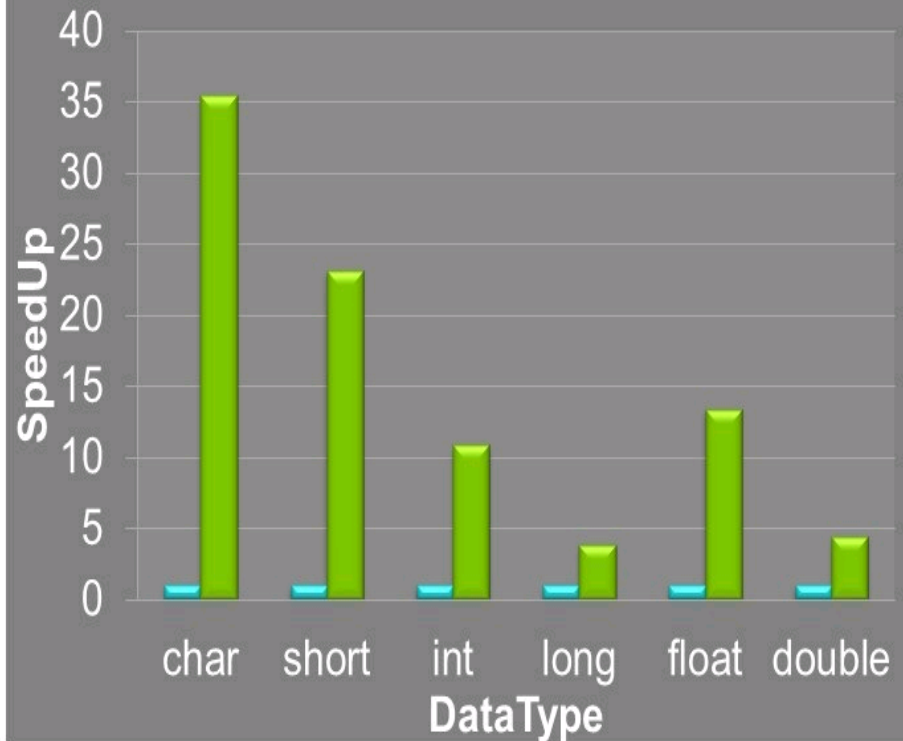
## Various Algorithms (32M integer samples)

■ TBB ■ Thrust



## Sort (32M integer samples)

■ TBB ■ 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>

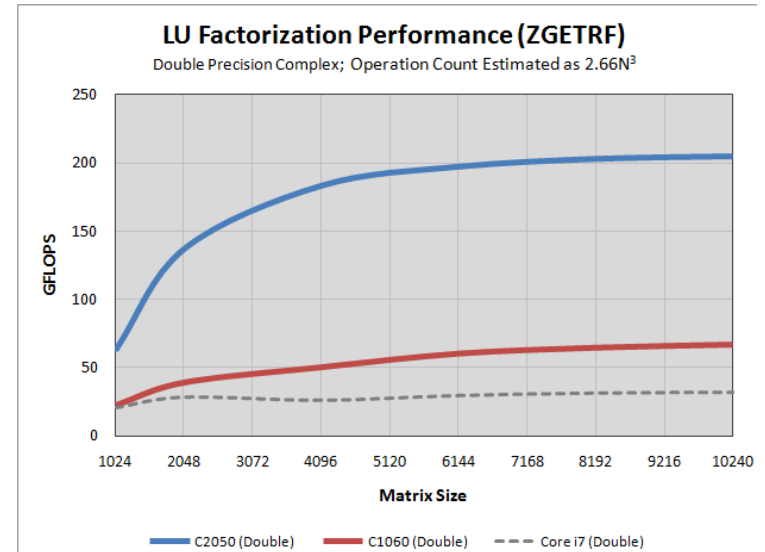
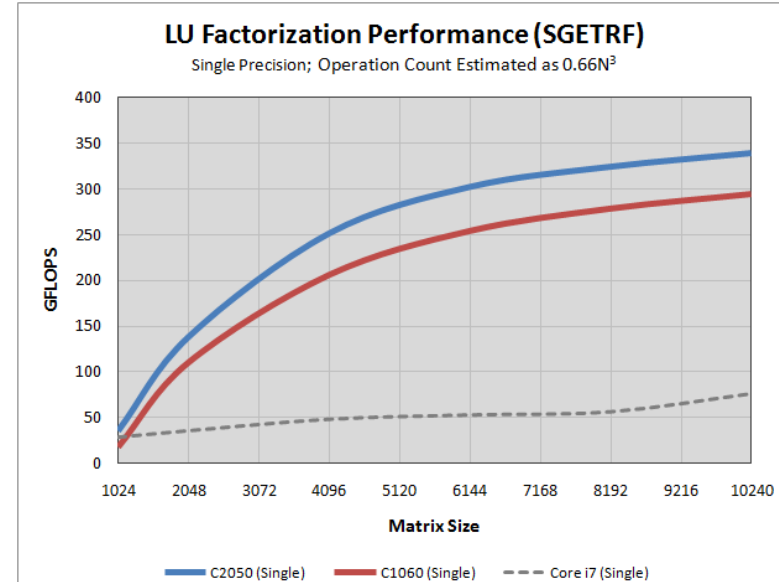
Proprietary library that implements the LAPACK in CUDA, which is available in several versions.

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



# Bibliografia

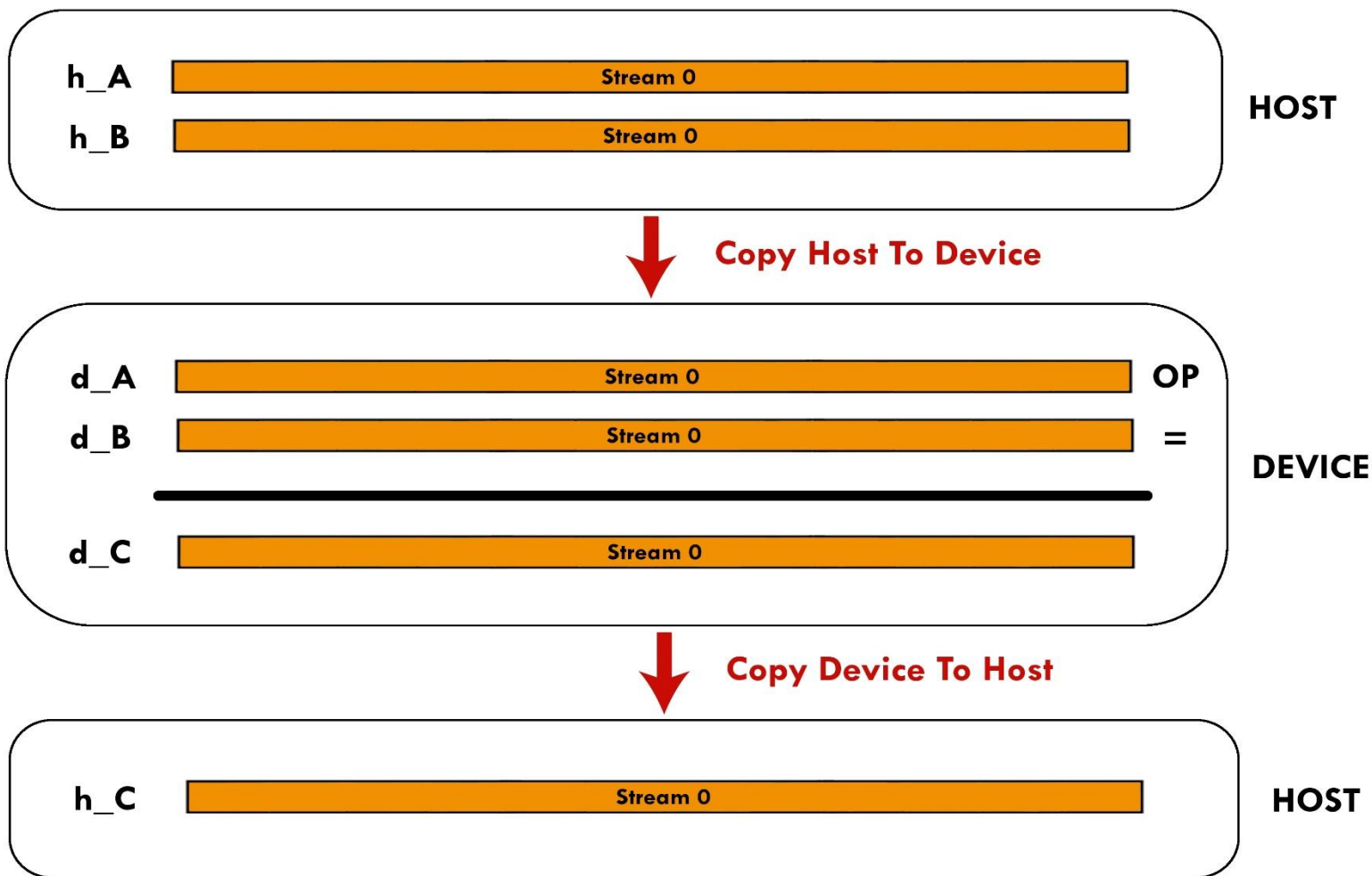
- ✓ CUDA C Programming Guide
- ✓ PGI CUDA fortran, <http://www.pgroup.com/doc/pgicudaforug.pdf>
- ✓ CUDA C Best Practices Guide
- ✓ NVIDIA CUDA Library Documentation (Doxygen –generated Reference Manual)
- ✓ Tuning CUDA Applications for Fermi
- ✓ Kirk and Hwu, [Programming Massively Parallel Processors](#)
- ✓ 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”](#)

# Esercizio 1: versione naive

Scrivere un programma in C o F90 che esegua i seguenti compiti.

- Allocare gli array di dati in singola precisione sull'*host*: `h_A`, `h_B`, `h_C` di dimensione `nSize`
- Inizializzare gli array `h_A` e `h_B` mediante la funzione `initArrayData()` in C o `RANDOM_NUMBER()` in F90
- Allocare gli array di dati in singola precisione sul *device*: `d_A`, `d_B`, `d_C`
- Trasferire gli array `h_A` e `h_B` su `d_A` e `d_B`, rispettivamente
- Lanciare il kernel `arrayFunc()` che elabora i dati contenuti in `d_A` e `d_B` e scrive il risultato su `d_C`
- Copiare `d_C` su `h_C`
- Misurare il tempo di calcolo mediante `cudaEvent` includendo anche i trasferimenti di memoria
- Eseguire la versione `funcArrayCPU()`
- Misurare il tempo di esecuzione tramite `cudaEvent`
- Calcolare lo Speed Up CPU/GPU

# Esercizio 1: versione naive

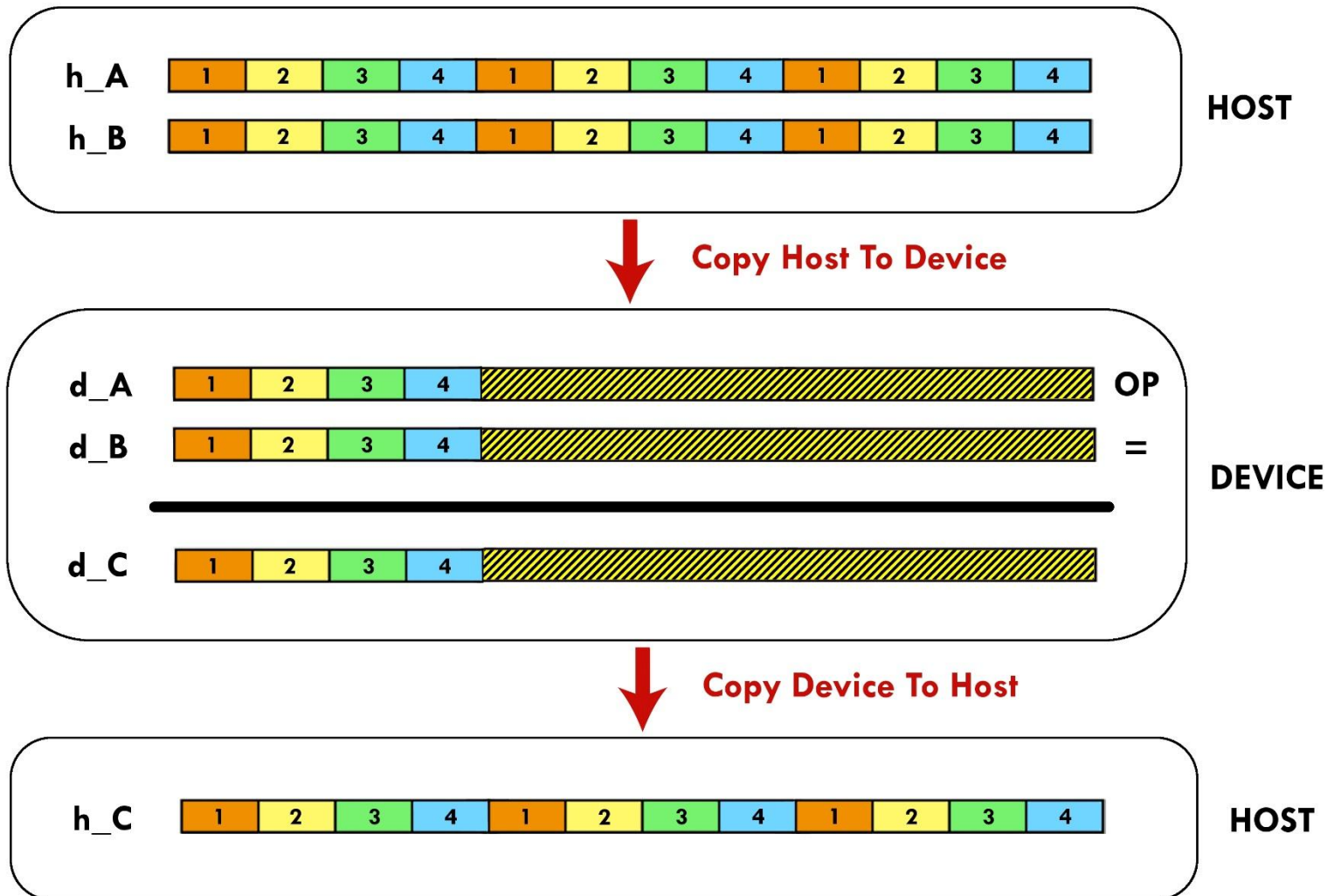


# Esercizio 2: versione cudaStream

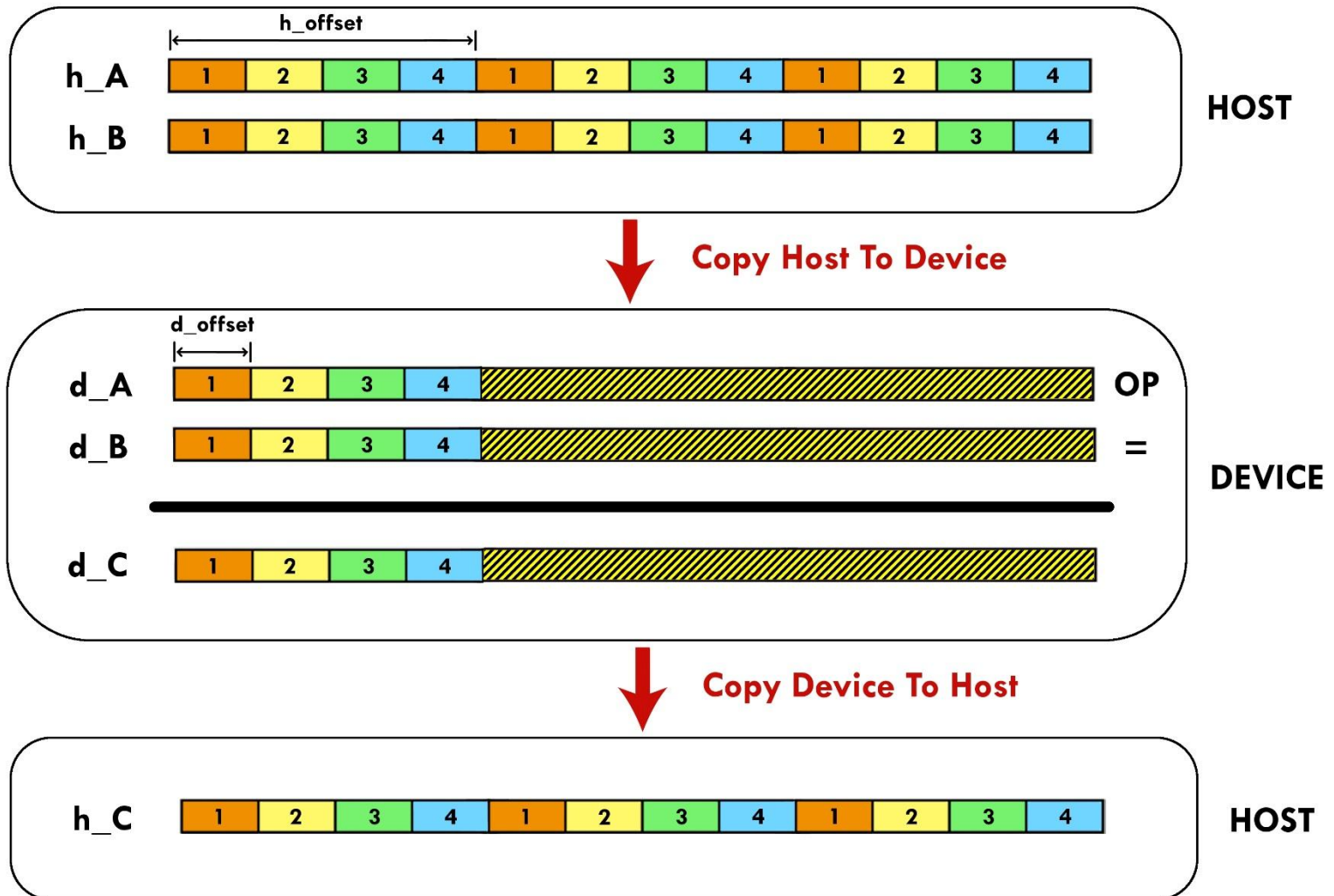
Scrivere un programma in C o F90 che esegua i seguenti compiti.

- Allocare gli array di dati sull'*host*: `h_A`, `h_B`, `h_C` di dimensione `nSize`
- Inizializzare gli array `h_A` e `h_B` utilizzando la funzione `initArrayData` in C o `RANDOM_NUMBER()` in F90
- Suddividere l'elaborazione di `h_A` e `h_B` in "chunk" di dimensione `chunk_size`
- Creare i `cudaStream` in numero pari a `streams_number`
- Allocare gli array di dati sul *device*: `d_A`, `d_B`, `d_C` dimensionati a `chunk_size * streams_number`
- Assegnare a ciascun `cudaStream` l'elaborazione di un certo numero di chunk. Ogni stream dovrà:
  - Copiare chunk di `h_A` e `h_B` da *host* a *device* in regioni opportune dei buffer `d_A` e `d_B`
  - Elaborare i chunk tramite il kernel `arrayFunc`
  - Copiare il risultato su `h_C`
- Misurare il tempo di esecuzione e valutare lo Speed Up rispetto all'esercizio precedente

# Esercizio 2: versione cudaStream



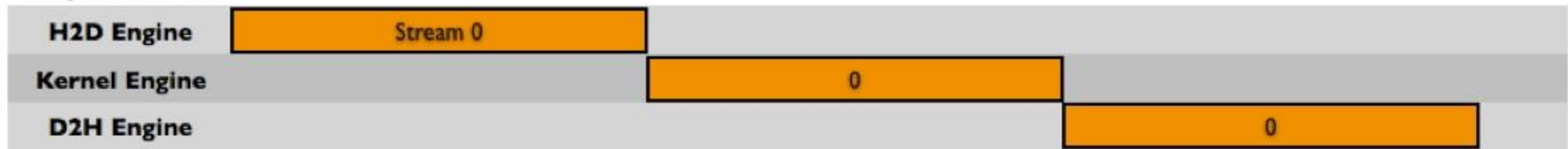
# Esercizio 2: versione cudaStream



# Esercizio 2: versione cudaStream

## Execution Time Lines

### Sequential Version



### Asynchronous Versions



Time →



# Esercizio 2: funzioni del CUDA runtime

## Nuove funzioni del CUDA Runtime necessarie (C for CUDA):

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

## Nuove funzioni del CUDA Runtime necessarie (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)`

# Esercizio 3: Versione cudaStream Multi GPU

Scrivere un programma in C o F90 che esegua i seguenti compiti.

- Allocare gli array di dati sull'*host*: `h_A`, `h_B`, `h_C` di dimensione `nSize`
- Inizializzare gli array `h_A` e `h_B` utilizzando la funzione `initArrayData()` in C o `RANDOM_NUMBER()` in F90
- Suddividere l'elaborazione di `h_A` e `h_B` in "chunk" di dimensione `chunk_size`
- Assegnare a ciascuna GPU un uguale numero di chunk da elaborare
- Allocare gli array di dati sul *device*: `d_A`, `d_B`, `d_C` dimensionati a `chunk_size*streams_number`
- Creare i `cudaStream` in numero pari a `streamsNumber`
- Assegnare a ciascun `cudaStream` l'elaborazione di un certo numero di chunk. Ogni stream dovrà:
  - Copiare chunk di `h_A` e `h_B` da *host* a *device* in regioni opportune dei buffer `d_A` e `d_B`
  - Elaborare i chunk tramite il kernel `arrayFunc`
  - Copiare il risultato su `h_C`
- Misurare il tempo di esecuzione e valutare lo Speed Up rispetto all'esercizio precedente