

# SDK, libraries, debugger, more on timing



UNIVERSITY OF  
**TORONTO**

# CUDA SDK



## ❖ The CUDA installation comes with many goodies:

- Documentation
  - `/scinet/arc/cuda-5.0/doc/pdf`
  - `/Developer/NVIDIA/CUDA-5.0/doc/pdf` (HP laptop)
- Samples
  - `/Developer/NVIDIA/CUDA-5.0/samples` (HP laptop; absent on arc?)
- Binaries (profiler, debugger, IDE)

# Documentation



## ❖ CUDA/doc/pdf

Name	Date Modified	Size	Kind
CUDA_C_Best_Practices_Guide.pdf	Sep 29, 2012 1:03 AM	2.2 MB	PDF
CUDA_C_Programming_Guide.pdf	Sep 29, 2012 1:03 AM	2.9 MB	PDF
CUDA_Compiler_Driver_NVCC.pdf	Sep 29, 2012 1:03 AM	1.8 MB	PDF
CUDA_CUBLAS_Users_Guide.pdf	Sep 29, 2012 1:03 AM	2.6 MB	PDF
CUDA_CUFFT_Users_Guide.pdf	Sep 29, 2012 1:03 AM	1.7 MB	PDF
CUDA_CUSPARSE_Users_Guide.pdf	Sep 29, 2012 1:03 AM	2.7 MB	PDF
CUDA_Debugger_API.pdf	Sep 29, 2012 1:03 AM	2 MB	PDF
CUDA_Developer_G...imus_Platforms.pdf	Sep 29, 2012 1:03 AM	678 KB	PDF
CUDA_Dynamic_Pa...ramming_Guide.pdf	Sep 29, 2012 1:03 AM	1 MB	PDF
CUDA_GDB.pdf	Sep 29, 2012 1:03 AM	1.8 MB	PDF
CUDA_Getting_Star...uide_For_Linux.pdf	Sep 29, 2012 1:03 AM	1.7 MB	PDF
CUDA_Getting_Star..._For_Mac_OS_X.pdf	Sep 29, 2012 1:03 AM	1.7 MB	PDF
CUDA_Getting_Star...osoft_Windows.pdf	Sep 29, 2012 1:03 AM	1.6 MB	PDF
CUDA_Memcheck.pdf	Sep 29, 2012 1:03 AM	1.7 MB	PDF
CUDA_Profiler_Users_Guide.pdf	Sep 29, 2012 1:03 AM	2.1 MB	PDF
CUDA_Samples_Gu..._New_Features.pdf	Sep 29, 2012 1:03 AM	2.1 MB	PDF
CUDA_Samples_Release_Notes.pdf	Sep 29, 2012 1:03 AM	1.7 MB	PDF



# CUDA SDK code samples



❖ Many code samples.

```
retina CUDA-5.0/samples $ pwd
/Developer/NVIDIA/CUDA-5.0/samples
retina CUDA-5.0/samples $ ls
0_Simple          6_Advanced      bin
1_Utilities      7_CUDALibraries common
2_Graphics       Documentation.html doc
3_Imaging        Makefile        release
4_Finance        Makefile-e      tools
5_Simulations    Samples.html
retina CUDA-5.0/samples $
```

**Bandwidth Test**

This is a simple test program to measure the memcopy bandwidth of the GPU and memcopy bandwidth across PCI-e. This test application is capable of measuring device to device copy bandwidth, host to device copy bandwidth for pageable and page-locked memory, and device to host copy bandwidth for pageable and page-locked memory.

**asyncAPI**

This sample uses CUDA streams and events to overlap execution on CPU and GPU.

**Clock**

This example shows how to use the clock function to measure the performance of kernel accurately.

**Simple Atomic Intrinsic**

A simple demonstration of global memory atomic instructions. Requires Compute Capability 1.1 or higher.

**GEFORCE 8 QUADRO FX 4600 or later TESLA**

# deviceQuery



```
[pfeiffer@marten class1]$ /opt/cuda/NVIDIA_GPU_Computing_SDK/C/bin/linux/release/deviceQuery
/opt/cuda/NVIDIA_GPU_Computing_SDK/C/bin/linux/release/deviceQuery Starting...
```

```
CUDA Device Query (Runtime API) version (CUDA static linking)
```

```
There are 2 devices supporting CUDA
```

```
Device 0: "GeForce GTX 470"
```

```
CUDA Driver Version:          3.20
CUDA Runtime Version:         3.20
CUDA Capability Major/Minor version number: 2.0
Total amount of global memory: 1341325312 bytes
Multiprocessors x Cores/MP = Cores: 14 (MP) x 32 (Cores/MP) = 448 (Cores)
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 32768
Warp size: 32
Maximum number of threads per block: 1024
Maximum sizes of each dimension of a block: 1024 x 1024 x 64
Maximum sizes of each dimension of a grid: 65535 x 65535 x 1
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Clock rate: 1.22 GHz
Concurrent copy and execution: Yes
Run time limit on kernels: No
Integrated: No
Support host page-locked memory mapping: Yes
Compute mode: Default (multiple host threads can use this de
vice simultaneously)
Concurrent kernel execution: Yes
Device has ECC support enabled: No
Device is using TCC driver mode: No
```



# simplePrintf



```
[pfeiffer@marten release]$ simplePrintf
> Using CUDA device [0]: GeForce GTX 470
Device 0: "GeForce GTX 470" with Compute 2.0 capability
printf() is called. Output:
```

```
[0, 0]: Value is:10
[0, 1]: Value is:10
[0, 2]: Value is:10
[0, 3]: Value is:10
[0, 4]: Value is:10
[0, 5]: Value is:10
[0, 6]: Value is:10
```

```
[0, 7]:
[3, 0]: //The macro CUPRINTF is defined for architectures
[3, 1]: //with different compute capabilities.
[3, 2]: #if __CUDA_ARCH__ < 200 //Compute capability 1.x architectures
[3, 3]: #define CUPRINTF cuPrintf
[3, 4]: #else //Compute capability 2.x architectures
[3, 5]: #define CUPRINTF(fmt, ...) \
[3, 6]: printf("[%d, %d]:\t" fmt, \
[3, 7]: blockIdx.y*gridDim.x+blockIdx.x, \
[2, 0]: threadIdx.z*blockDim.x*blockDim.y+threadIdx.y*blockDim.x+threadIdx.x, \
[2, 1]: _VA_ARGS__)
[2, 2]: #endif
[2, 3]: __global__ void testKernel(int val)
[2, 4]: {
[2, 5]: CUPRINTF("\tValue is:%d\n", val);
[2, 6]: }
[2, 7]:
```

```
[1, 0]: Value is:10
[1, 1]: Value is:10
[1, 2]: Value is:10
[1, 3]: Value is:10
```

# many more

```
retina CUDA-5.0/samples $ ls
0_Simple          6_Advanced      bin
1_Utillities     7_CUDA Libraries common
2_Graphics       Documentation.html doc
3_Imaging        Makefile        releaseNotesData
4_Finance        Makefile-e      tools
5_Simulations    Samples.html

retina CUDA-5.0/samples $ ls 0_Simple
asyncAPI          simpleMultiCopy
cdpSimplePrint    simpleMultiGPU
cdpSimpleQuicksort simpleP2P
clock             simplePitchLinearTexture
cppIntegration    simplePrintf
cudaOpenMP        simpleSeparateCompilation
inlinePTX         simpleStreams
matrixMul         simpleSurfaceWrite
matrixMulCUBLAS  simpleTemplates
matrixMulDrv      simpleTexture
matrixMulDynlinkJIT simpleTextureDrv
simpleAssert       simpleVoteIntrinsics
simpleAtomicIntrinsics simpleZeroCopy
simpleCallback     template
simpleCubemapTexture template_runtime
simpleIPC          vectorAdd
simpleLayeredTexture vectorAddDrv
simpleMPI
```

# CUDA libraries



- ❖ The best code is that which you don't write
- ❖ **NVIDIA** put lots of effort into optimizing libraries. Use them!
  - CuBLAS (matrix operations)
  - CuFFT
  - CuSparse (sparse matrix)
  - CuRand (random numbers)



```
#include "cublas.h"
// initialize cuBLAS
cublasStatus status;
status=cublasInit();
if(status != CUBLAS_STATUS_SUCCESS) {
    fprintf(stderr, "!!!! CUBLAS initialization error\n");
    exit(1);
}
```

Housekeeping

Work!!

```
CHK_CUDA( cudaMemcpy(ad, aT, n*n*sizeof(float), cudaMemcpyHostToDevice) );
CHK_CUDA( cudaMemcpy(bd, bT, n*n*sizeof(float), cudaMemcpyHostToDevice) );

cublas_sgemm(ad, bd, n, cd);

CHK_CUDA( cudaMemcpy(ccuda, cd, n*n*sizeof(float), cudaMemcpyDeviceToHost) );
```

```
status=cublasShutdown();
if(status != CUBLAS_STATUS_SUCCESS) {
    fprintf(stderr, "!!!! shutdown error\n");
    return 1;
}
```

Housekeeping

```
void cublas_sgemm(const float *a, const float *b,
                 const int n, float *c) {
    // C = alpha A B + beta C
    cublasSgemm('n', // no transpose on A
               'n', // no transpose on B
               n,   // rows of A = rows of C
               n,   // cols of B = cols of C
               n,   // cols of A = rows of B
               1.f, // alpha
               a,   // A
               n,   // leading dimension A
               b,   // B
               n,   // leading dimension B
               0.f, // beta
               c,   // C
               n    // leading dimension C
    );
    cublasStatus status = cublasGetError();
    if(status != CUBLAS_STATUS_SUCCESS) {
        fprintf(stderr, "!!!! kernel execution error.\n");
        exit(1);
    }
}
```

CUDA\_CUBLAS\_Users\_Guide.pdf

# Timings



```
[pfeiffer@marten class3]$ ./matmult --matsize=2048 --nblocks=256
Matrix size = 2048, Number of blocks = 256.
CPU time      = 90450.7 millisec.
GPU time      = 363.987 millisec.
cuBLAS time   = 36.519 millisec.
CUDA    and CPU results differ by 0.000000
cuBLAS  and CPU results differ by 0.000245
```



# Speed in GFLOPS



❖ matrix multiplication in single-precision

	$256^2$	$512^2$	$1024^2$	$2048^2$
selfwritten CPU	0.78	0.59	0.25	0.2
best selfwritten GPU	30	39	45	47
cuBLAS	80	150	310	477

# cuBLAS summary



## ❖ Housekeeping routines

- initialization
- memory allocation
- copy host -> device and back

## ❖ BLAS-interfaces with data already on device

## ❖ If one does only BLAS, cuBLAS might be all one ever needs

- ... potentially annoying Fortran column-major format

- ❖ Python bindings
- ❖ Very thin layer that makes the whole CUDA structure accessible in python
  - Very flexible, quick development
  - user still needs to know as much about CUDA as for C



# Whetting your appetite

```
1 import pycuda.driver as cuda
2 import pycuda.autoinit
3 import numpy
4
5 a = numpy.random.randn(4,4).astype(numpy.float32)
6 a_gpu = cuda.mem_alloc(a.nbytes)
7 cuda.memcpy_htod(a_gpu, a)
```



[This is examples/demo.py in the PyCUDA distribution.]

[From http://www.nvidia.com/content/GTC-2010/pdfs/2041\\_GTC2010.pdf](http://www.nvidia.com/content/GTC-2010/pdfs/2041_GTC2010.pdf)

# Whetting your appetite

```
1 mod = cuda.SourceModule("""
2     __global__ void twice(float *a)
3     {
4         int idx = threadIdx.x + threadIdx.y*4;
5         a[idx] *= 2;
6     }
7     """)
8
9 func = mod.get_function("twice")
10 func(a_gpu, block=(4,4,1))
11
12 a_doubled = numpy.empty_like(a)
13 cuda.memcpy_dtoh(a_doubled, a_gpu)
14 print a_doubled
15 print a
```



[From http://www.nvidia.com/content/GTC-2010/pdfs/2041\\_GTC2010.pdf](http://www.nvidia.com/content/GTC-2010/pdfs/2041_GTC2010.pdf)

# cuda-dbg



- ❖ Debugger for CUDA code
- ❖ Standard gdb with additional commands
- ❖ Also useful to understand the structure of sm/warp/block/grid/threads
- ❖ [CUDA\\_GDB.pdf](#)

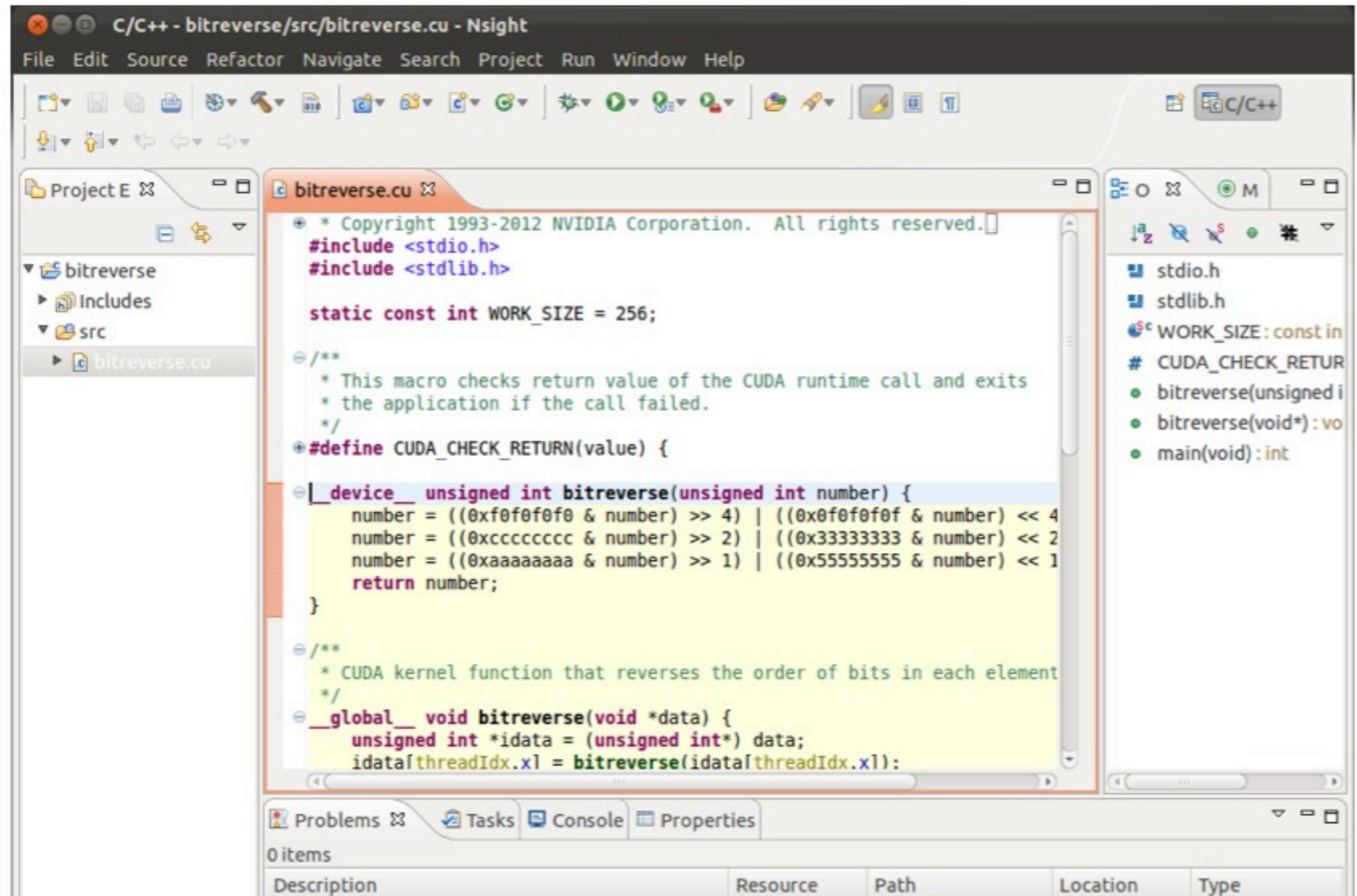
## CUDA-GDB CUDA DEBUGGER

DU-05227-042 \_v5.0 | October 2012

**User Manual**



## ❖ IDE w/ compiler/debugger/profiler



```

C/C++ - bitreverse/src/bitreverse.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help

Project E
  bitreverse
    Includes
    src
      bitreverse.cu

bitreverse.cu
  * Copyright 1993-2012 NVIDIA Corporation. All rights reserved.
  #include <stdio.h>
  #include <stdlib.h>

  static const int WORK_SIZE = 256;

  /**
   * This macro checks return value of the CUDA runtime call and exits
   * the application if the call failed.
   */
  #define CUDA_CHECK_RETURN(value) {

  __device__ unsigned int bitreverse(unsigned int number) {
    number = ((0xf0f0f0f0 & number) >> 4) | ((0xf0f0f0f0 & number) << 4);
    number = ((0xcccccccc & number) >> 2) | ((0x33333333 & number) << 2);
    number = ((0xaaaaaaaa & number) >> 1) | ((0x55555555 & number) << 1);
    return number;
  }

  /**
   * CUDA kernel function that reverses the order of bits in each element
   */
  __global__ void bitreverse(void *data) {
    unsigned int *idata = (unsigned int*) data;
    idata[threadIdx.x] = bitreverse(idata[threadIdx.x]);
  }

Problems Tasks Console Properties
0 items
Description Resource Path Location Type

```

# (a-)synchronous execution

## ❖ Kernels execute *asynchronously*

```
tick(&gputimer);
cuda_sgemm_shared<<<gridsize, blocksize,
                    (2*blocksize.x*blocksize.y*sizeof(float))>>>(ad, bd, n, cd);
gputime = tock(&gputimer);
```

- measures essentially zero time

## ❖ *cudaMemcpy blocking* (NB: asynchronous Memcpy possible)

```
tick(&gputimer);
CHK_CUDA( cudaMemcpy(ad, a, n*n*sizeof(float), cudaMemcpyHostToDevice) );
CHK_CUDA( cudaMemcpy(bd, b, n*n*sizeof(float), cudaMemcpyHostToDevice) );

blocksize = make_uint3( (n+nblocks-1)/nblocks, (n+nblocks-1)/nblocks, 1);
gridsize  = make_uint3( nblocks, nblocks, 1);

cuda_sgemm_shared<<<gridsize, blocksize,
                    (2*blocksize.x*blocksize.y*sizeof(float))>>>(ad, bd, n, cd);

CHK_CUDA( cudaMemcpy(ccuda, cd, n*n*sizeof(float), cudaMemcpyDeviceToHost) );
gputime = tock(&gputimer);
```

- works, but measures kernel-execution and cudaMemcpy



# cudaEvents

- ❖ cudaEvents are inserted into the kernel-queue.
- ❖ They record time when kernel-queue reaches them

## How to time code using CUDA events

```
cudaEvent_t start, stop;
float time;

cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord( start, 0 );
kernel<<<grid,threads>>> ( d_odata, d_idata, size_x, size_y,
                          NUM_REPS);
cudaEventRecord( stop, 0 );
cudaEventSynchronize( stop );

cudaEventElapsedTime( &time, start, stop );
cudaEventDestroy( start );
cudaEventDestroy( stop );
```

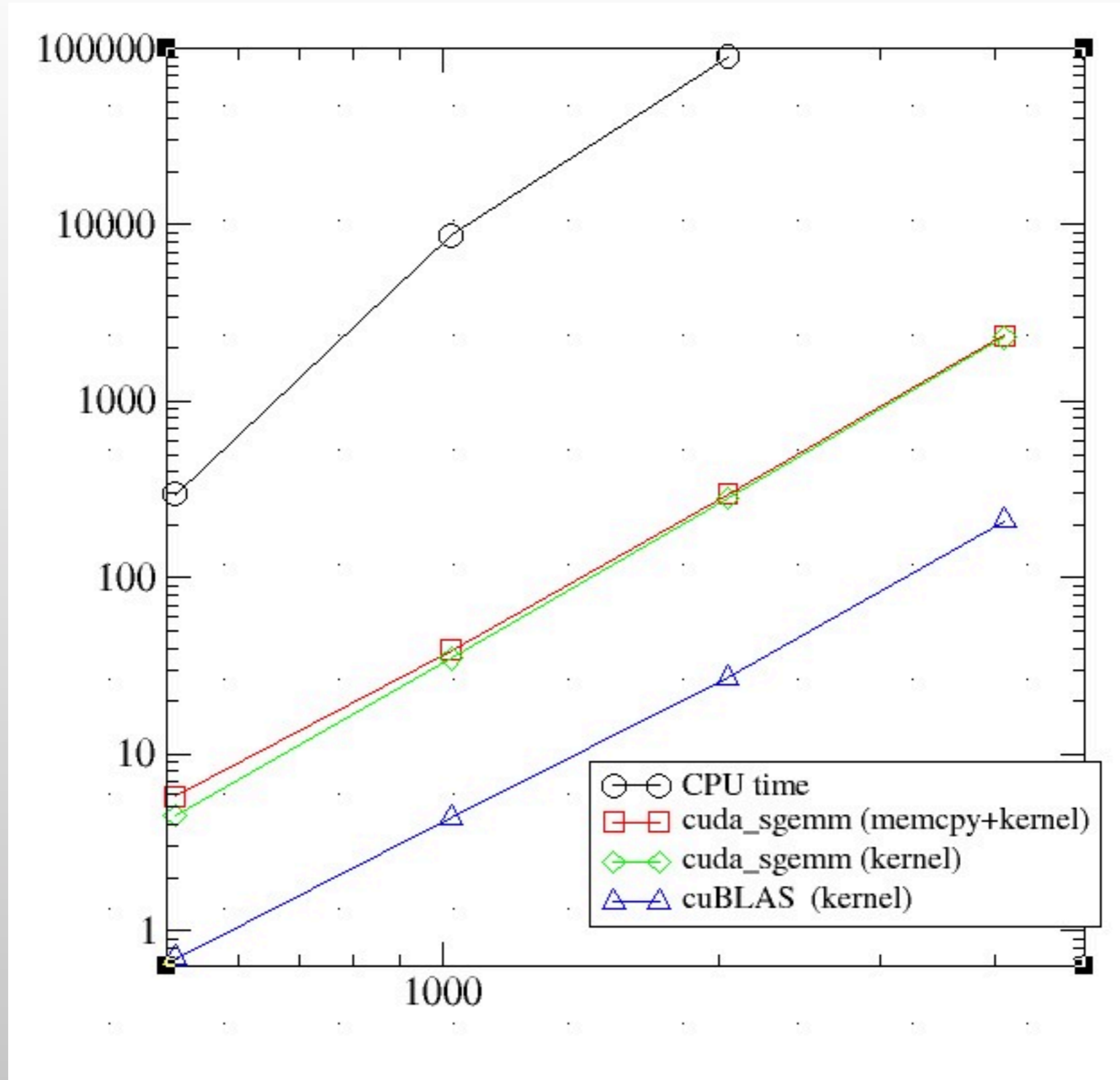
[CUDA\\_C\\_Best\\_Practices\\_Guide.pdf](#), p.15



# matmult.cu w/ cuBLAS timings revisited



```
[pfeiffer@marten class3]$ ./matmult --matsize=512 --nblocks=64
Matrix size = 512, Number of blocks = 64.
CPU time      = 298.689 millisc.
GPU time      = 5.854 millisc      in kernel = 4.49696 millisc
cuBLAS time   = 1.789 millisc      in kernel = 0.706464 millisc
CUDA and CPU results differ by 0.000004
cuBLAS and CPU results differ by 0.000004
[pfeiffer@marten class3]$ ./matmult --matsize=1024 --nblocks=128
Matrix size = 1024, Number of blocks = 128.
CPU time      = 8732.69 millisc.
GPU time      = 39.634 millisc     in kernel = 35.5955 millisc
cuBLAS time   = 7.181 millisc     in kernel = 4.4471 millisc
CUDA and CPU results differ by 0.000007
cuBLAS and CPU results differ by 0.000007
[pfeiffer@marten class3]$ ./matmult --matsize=2048 --nblocks=256
Matrix size = 2048, Number of blocks = 256.
CPU time      = 81077.3 millisc.
GPU time      = 297.248 millisc   in kernel = 283.063 millisc
cuBLAS time   = 36.579 millisc   in kernel = 27.3442 millisc
CUDA and CPU results differ by 0.000015
cuBLAS and CPU results differ by 0.000015
```



# Homework



- ❖ Experience with BLAS/CuBLAS
  
- ❖ Many smallish matrix-operations
  - Check how well CuBLAS does
  - Optional: Try to do better.