

Friday, Feb. 16:

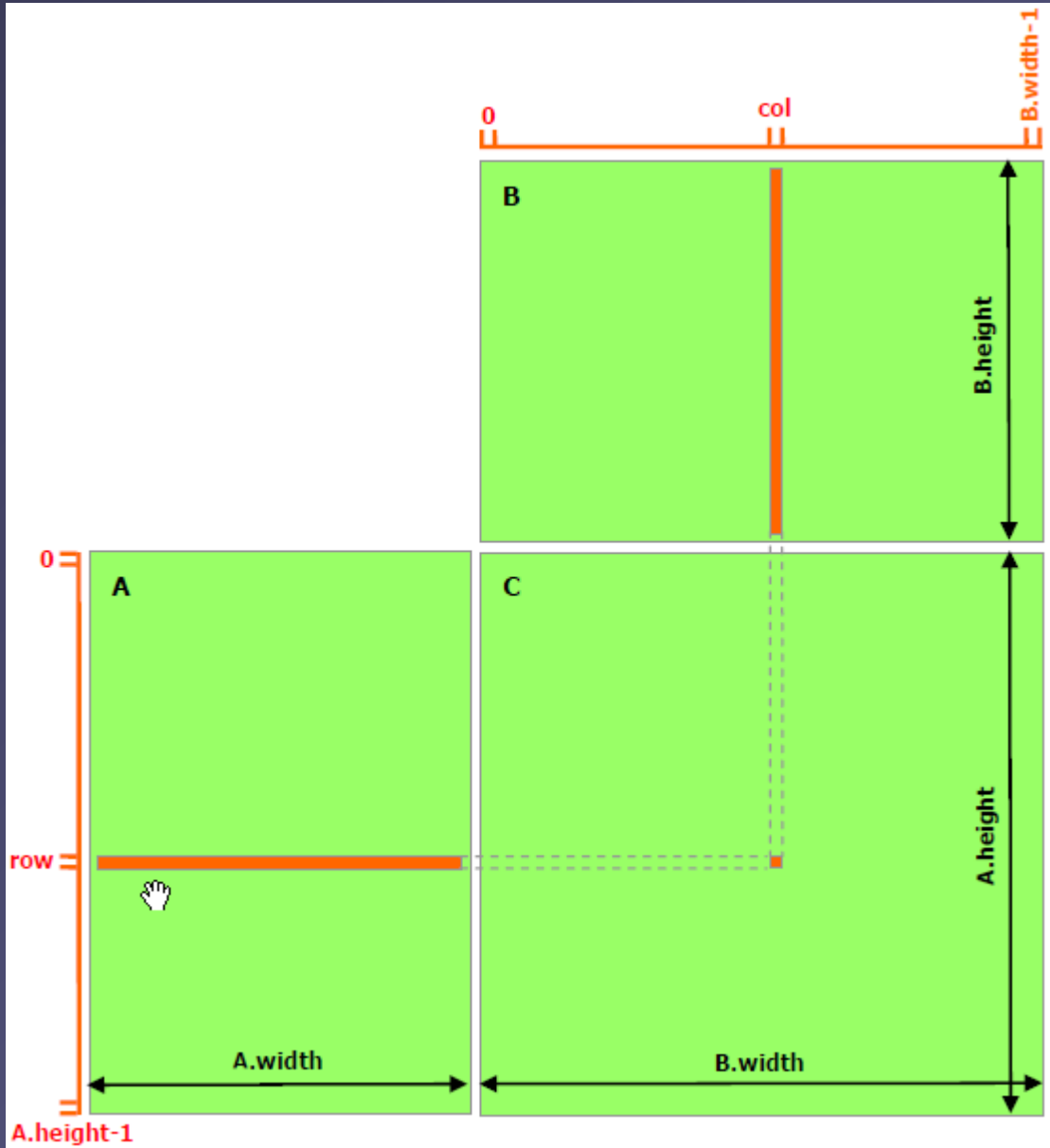
Matrix Multiplication

Histograms (from Jason Sanders' book; see our webpage link)

Timing and Debugging

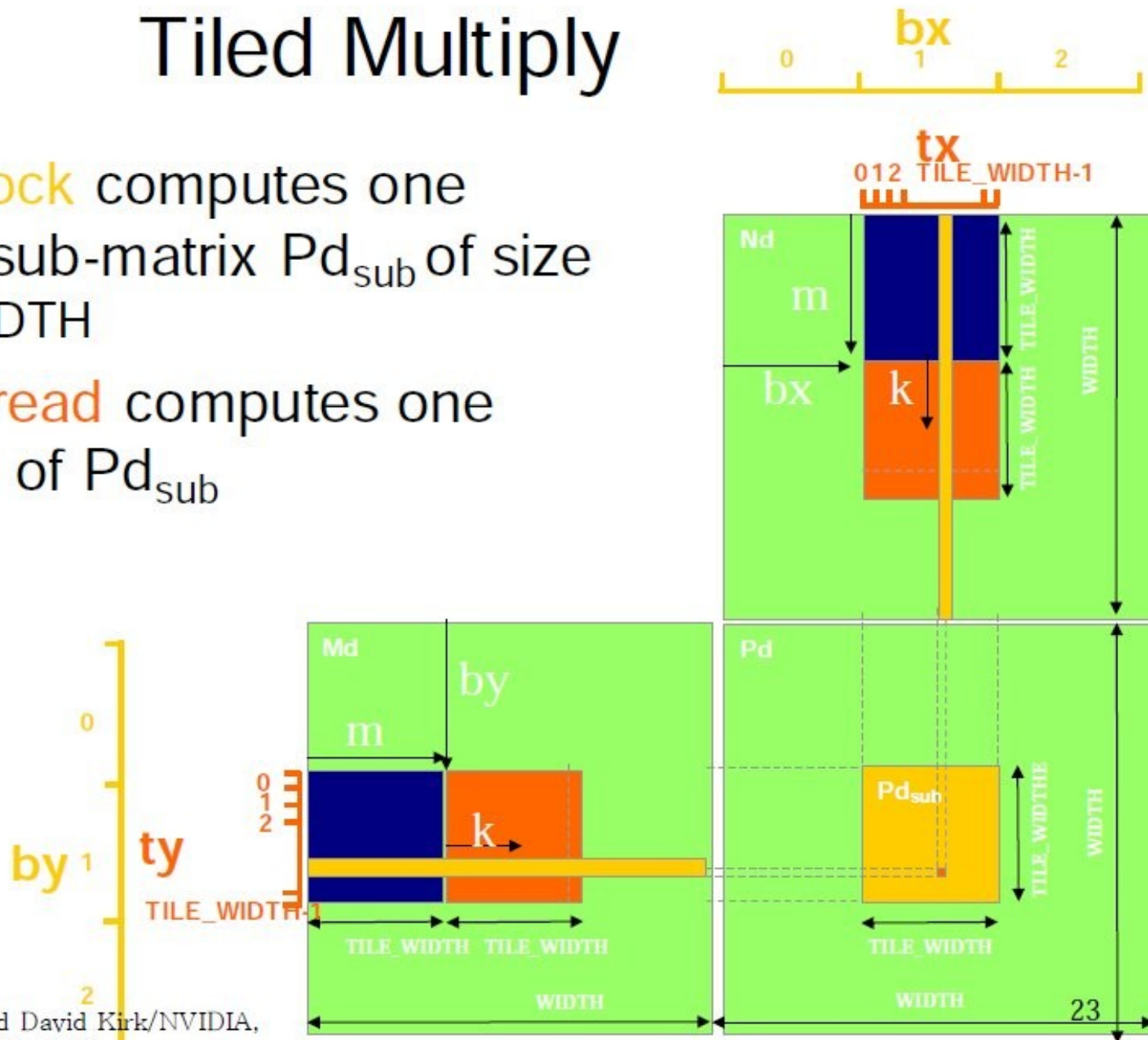
Wrap-Up of CUDA/Outlook

Matrix Intuitive Multiply



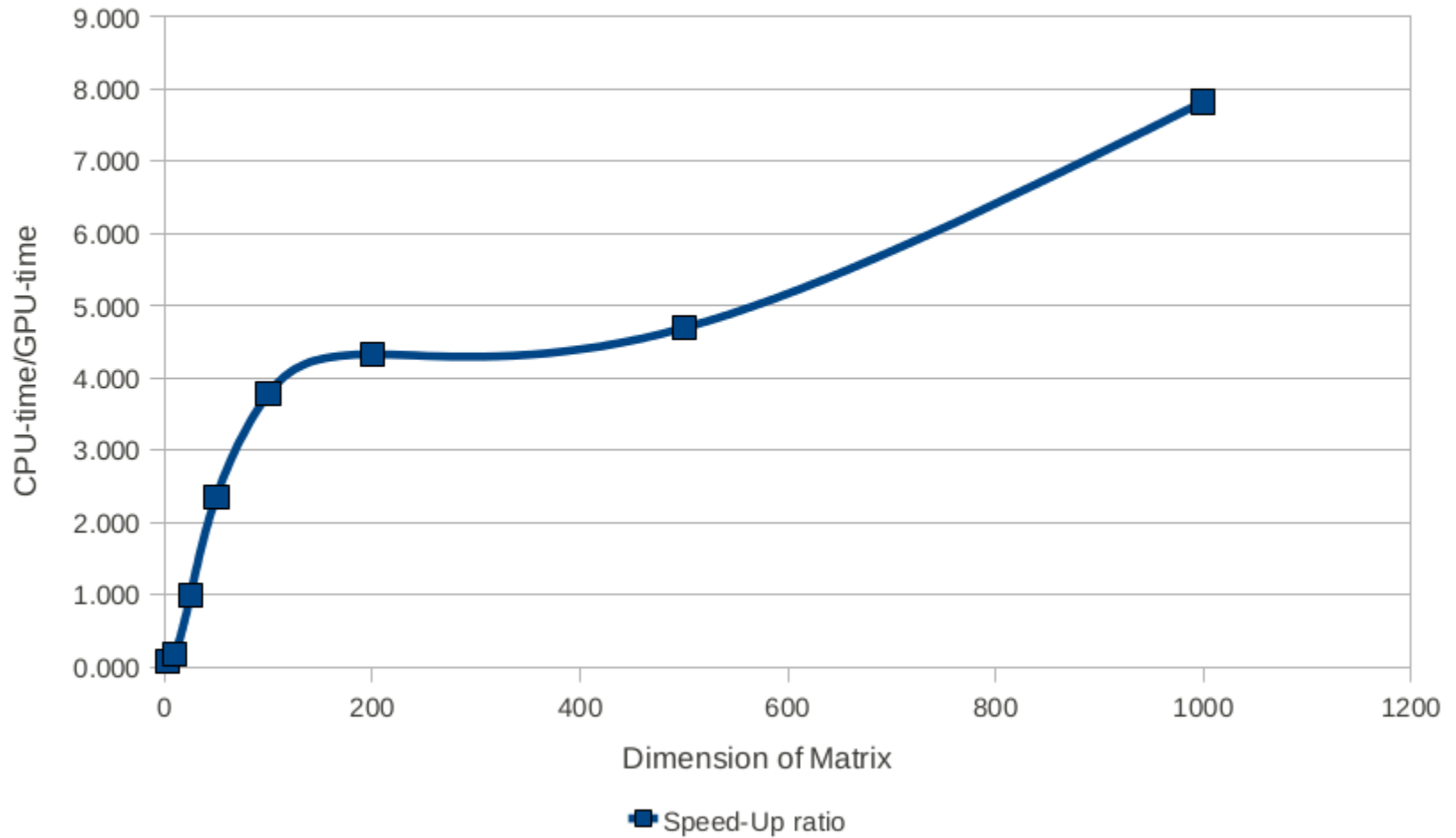
Tiled Multiply

- Each **block** computes one square sub-matrix Pd_{sub} of size $TILE_WIDTH$
- Each **thread** computes one element of Pd_{sub}



Speed-Up Ratio

GPU speed-up over CPU



Histogram Computation

8_histo

Task:

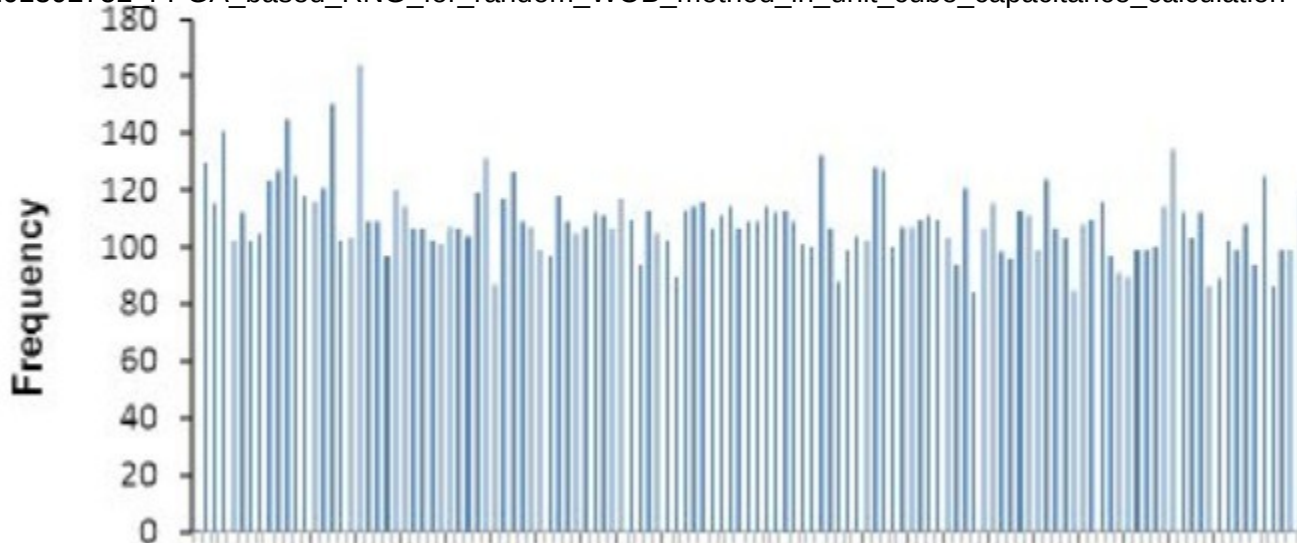
- 100 million integers $0 \leq n_i < 256$;
- Randomly distributed with equal probability;
- What is the frequency (number) with which every integer occurs?

Expect: equal distribution again, with fluctuations.

Histogram of 15000 uniform random number

Source: https://www.researchgate.net/publication/261392752_FPGA_based_RNG_for_random_WOB_method_in_unit_cube_capacitance_calculation

261392752_FPGA_based_RNG_for_random_WOB_method_in_unit_cube_capacitance_calculation



This Timing API is used in 8_histo/histo.cu !

Timing with CUDA Event API

```
int main ()
{
    cudaEvent_t start, stop;
    float time;

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

    cudaEventRecord (start, 0);

    someKernel <<<grids, blocks, 0, 0>>> (...);

    cudaEventRecord (stop, 0);
    cudaEventSynchronize (stop);
    cudaEventElapsedTime (&time, start, stop);

    cudaEventDestroy (start);
    cudaEventDestroy (stop);

    printf ("Elapsed time %f sec\n", time*.001);

    return 1;
}
```

CUDA Event API Timer are,

- OS independent
- High resolution
- Useful for timing asynchronous calls

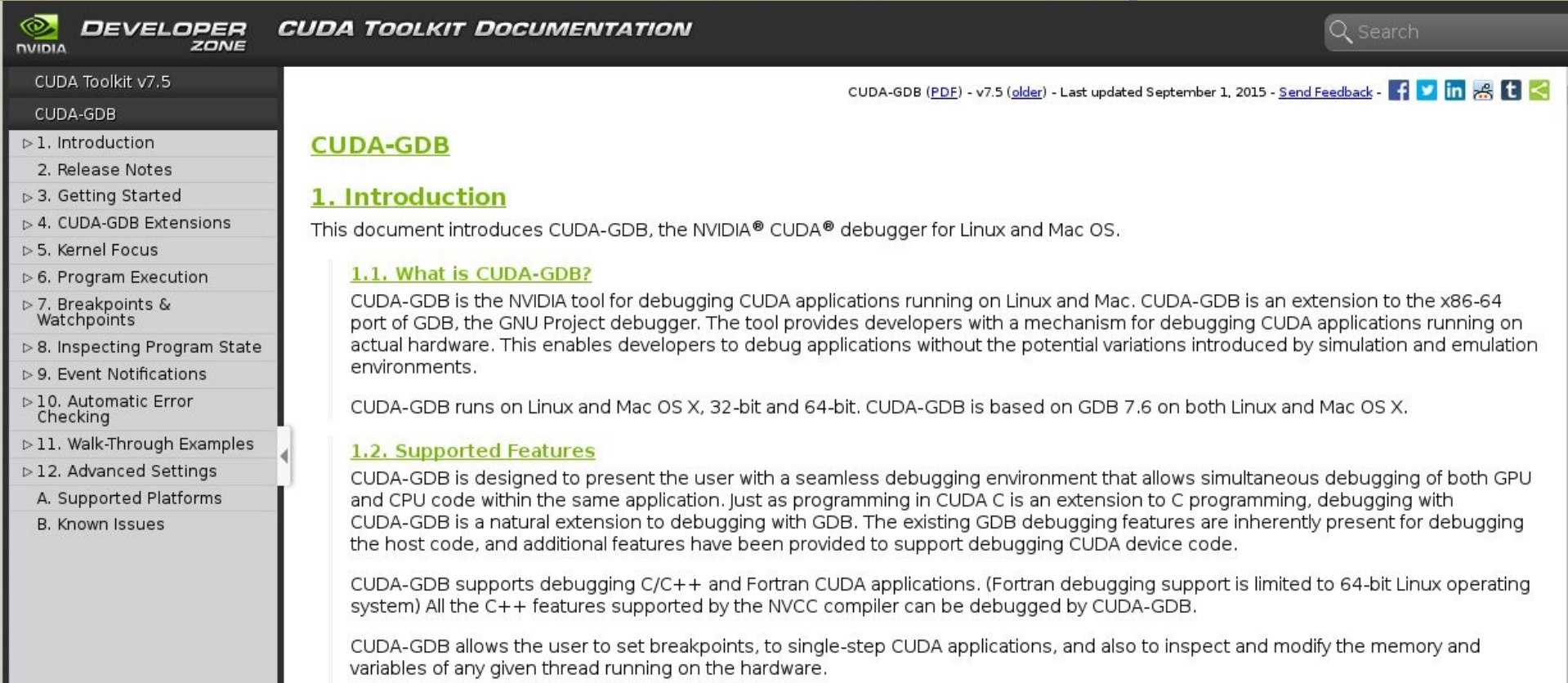
← Ensures kernel execution has completed

Standard CPU timers will not measure the timing information of the device.

CUDA – GNU Debugger – CUDA-gdb

do not forget: `nvcc -g -G ...` before running ...
(not possible on kepler, login node has no GPU!)

<http://docs.nvidia.com/cuda/cuda-gdb/index.html>



The screenshot shows the NVIDIA Developer Zone documentation page for CUDA-GDB. The page has a dark header with the NVIDIA logo, 'DEVELOPER ZONE', and 'CUDA TOOLKIT DOCUMENTATION'. A search bar is on the right. A left sidebar contains a navigation menu with items like 'CUDA Toolkit v7.5', 'CUDA-GDB', and a list of numbered sections. The main content area is white and contains the following text:

CUDA-GDB (PDF) - v7.5 (older) - Last updated September 1, 2015 - [Send Feedback](#) - [f](#) [t](#) [in](#) [d](#) [t](#) [v](#)

CUDA-GDB

1. Introduction

This document introduces CUDA-GDB, the NVIDIA® CUDA® debugger for Linux and Mac OS.

1.1. What is CUDA-GDB?

CUDA-GDB is the NVIDIA tool for debugging CUDA applications running on Linux and Mac. CUDA-GDB is an extension to the x86-64 port of GDB, the GNU Project debugger. The tool provides developers with a mechanism for debugging CUDA applications running on actual hardware. This enables developers to debug applications without the potential variations introduced by simulation and emulation environments.

CUDA-GDB runs on Linux and Mac OS X, 32-bit and 64-bit. CUDA-GDB is based on GDB 7.6 on both Linux and Mac OS X.

1.2. Supported Features

CUDA-GDB is designed to present the user with a seamless debugging environment that allows simultaneous debugging of both GPU and CPU code within the same application. Just as programming in CUDA C is an extension to C programming, debugging with CUDA-GDB is a natural extension to debugging with GDB. The existing GDB debugging features are inherently present for debugging the host code, and additional features have been provided to support debugging CUDA device code.

CUDA-GDB supports debugging C/C++ and Fortran CUDA applications. (Fortran debugging support is limited to 64-bit Linux operating system) All the C++ features supported by the NVCC compiler can be debugged by CUDA-GDB.

CUDA-GDB allows the user to set breakpoints, to single-step CUDA applications, and also to inspect and modify the memory and variables of any given thread running on the hardware.

Click the image to shrink it.



Debug

- vectorAdd {0} [device: gk110 (0)] (Breakpoint)
 - CUDA Thread (0,0,0) Block (0,0,0)
 - CUDA Thread (1,0,0) Block (0,0,0)**
- All CUDA Threads
 - Block (0,0,0) [sm: 11]
 - CUDA Thread (0,0,0) [warp: 0 lane: 0] (vectorAdd.cu:36)

Variables Breakpoints CUDA Modules

Search CUDA Information

(0,0,0)	SM 11	256 threads of 256 are running
(0,0,0)	Warp 0 Lane 0	vectorAdd.cu:36 (0x9a6530)
(1,0,0)	Warp 0 Lane 1	vectorAdd.cu:36 (0x9a6530)

```

32 vectorAdd(const float *A, const float *B, float *C, int numE
33 {
34     int i = blockDim.x * blockIdx.x + threadIdx.x;
35
36     if (i < numElements)
37     {
38         C[i] = A[i] + B[i];
39     }
40 }
41

```

Outline Registers

Name	T(0,0,0)B(0,0,0)	T(1,0,0)B(0,0,0)
R5	4	4
R6	3149824	3149824
R7	4	4
R8	0	1
R9	0	1
R10	1060608	-271911904
R11	0	2

```

vectorAdd [C/C++ Application] gdb traces
0x400300800"}, {name="C", value="0x400301000"}, {name="numElements", value="500"}], file="../src/vectorAd
d.cu", fullname="/home/eostroukhov/cuda-workspace/vectorAdd/src/vectorAdd.cu", line="36"}
470,340 (gdb)
470,340 157^done, register-values=[{number="15", value="0x0"}]
470,340 (gdb)
470,340 158^done, register-values=[{number="15", value="0"}]
470,340 (gdb)

```

Linux Terminal commandline:
 nsys (nsys --help)

Wrapping Up 1

Exercises afternoons

0. hello, device- first kernel call, hello world, GPU properties
1. add - vector addition using one thread in one block only
2. add-index - vector addition using blocks in parallel, one thread per block only.
3. add-parallel - vector addition using all blocks and threads in parallel
4. dot - scalar product using shared memory of one block only for reduction
5. dot-full - scalar product using shared memory and atomic add across blocks
6. dot-perfect - scalar product; fat threads and final reduction on host.
7. matmul - matrix multiplication with tiled access shared memory.
8. histo - histogram using fat threads and atomic add on shared and global memory, timing

Wrapping Up 2

Elements of CUDA C learnt:

threadId.x , blockDim.x, gridDim.x
(threadId.y, blockDim.y, gridDim.y
kernel<<<n,m>>> (...)
Kernel<<n,m,size>>(…)
kernel<<<dimBlock,dimGrid>>>(…)

__global__

__shared__

cudaMalloc / cudaFree

cudaMemcpy / cudaMemcpy

cudaGetDeviceProperties

cudaEventCreate, cudaEventRecord,

cudaEventSynchronize, cudaEventElapsedTime,

cudaEventDestroy

AtomicAdd (on global or shared mem.)

Threads, Blocks

(matmul coming with 2D grids)

kernel calls

kernel call with dyn. alloc. size

dim3 variable type (matmul)

device code

shared memory on GPU

manage global memory of GPU

copy/set to or from memory

get device properties in program

CUDA profiling

atomic functions

Wrapping Up 3

What we have not yet learnt...

`__constant__`
`__device__`

constant memory on GPU
functions device to device

Intrinsic Functions (`__device__` type)

https://docs.nvidia.com/cuda/cuda-math-api/group__CUDA__MATH__SINGLE.html#group__CUDA__MATH__SINGLE

`__host__`

functions host to host

More atomic functions

`cudaBindTexture`

using texture memory

fat threads for 2D and 3D stencils

thread coalescence opt.

`cudaStreamCreate`, `cudaStreamDestroy`

working with CUDA streams

`<<<n,m,size,s>>>`

kernel call with streams `s`

using Tensor Cores

...

Matrix Multiply and Histogram

Matrix Multiply: Inspired by Lecture of Wen-mei Hwu

<http://whtresearch.sourceforge.net/example.html>

On kepler: 7_matmul/

Histo: Chapter in Book of Jason Sanders

<https://wwwstaff.ari.uni-heidelberg.de/spurzem/lehre/WS22/cuda/files/cuda-histograms.pdf>

(Link on our webpage)

On kepler: 8_histo/

histo.cu (atomic on both shared and global memory)

histo-no-atomic.cu (atomic only on global memory)

Final Remarks

Important Note:

If you do some NBODY research in the future, please contact us (tutors or lecturer); do not use the course code for research it is not fully performant in some respects (openMP).

Remember for course certificate:

- * Output files of small experiments on your lecture account (0_hello, 1_add, ... , 7-matmul, 8-histo)
- * Return two plots, one data file, and a few comments to your tutors
Deadline? About one week, check with your tutors. Outputs of the 8 Nbody runs on your lecture account (one per team of two enough).
- * Notice: Student Queues will close Sunday, Feb. 25, 2024
If you need to run later, we can find a solution, contact me or tutor.
spurzem@ari.uni-heidelberg.de