Friday, Feb. 24:

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

Timing and Debugging

Wrap-Up of CUDA

Using Tensor Cores in CUDA (only preview)

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, Mar 5, 2023
You may run later, but contact me before:
spurzem@ari.uni-heidelberg.de

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 (cudaEventSynchronize)

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

NVIDIA DEVELOPER ZONE CUDA TOOLKIT DOCUMENTATION Search

CUDA-GDB (PDF) - v7.5 (older) - Last updated September 1, 2015 - Send Feedback -

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.

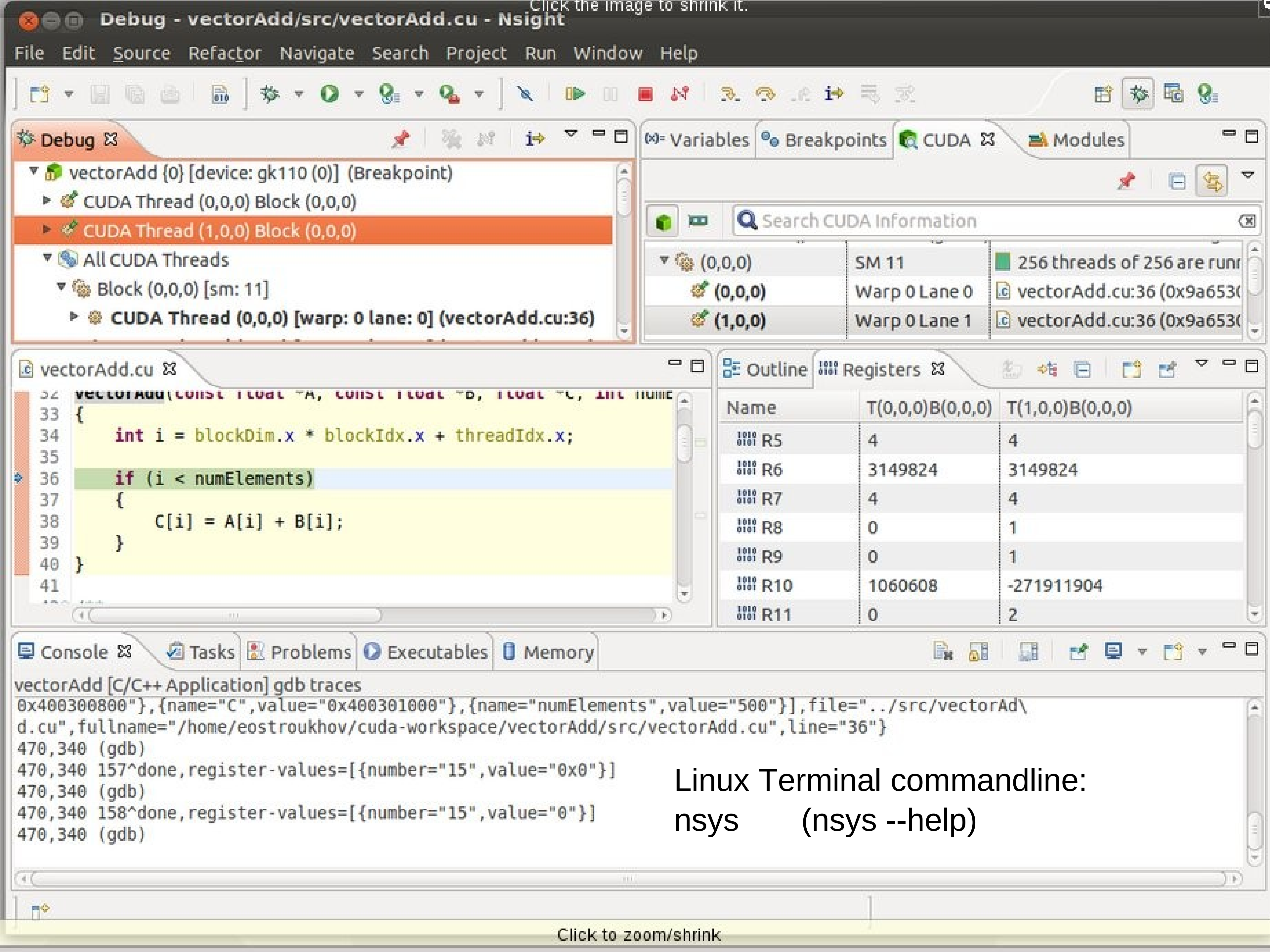
CUDA-GDB runs on Linux and Mac OS X, 32-bit and 64-bit. CUDA-GDB is based on GDB 7.6 on both Linux and Mac OS X.

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.



Linux Terminal commandline:
nsys (nsys --help)

Technical Blog

Subscribe

https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

Technical Walkthrough

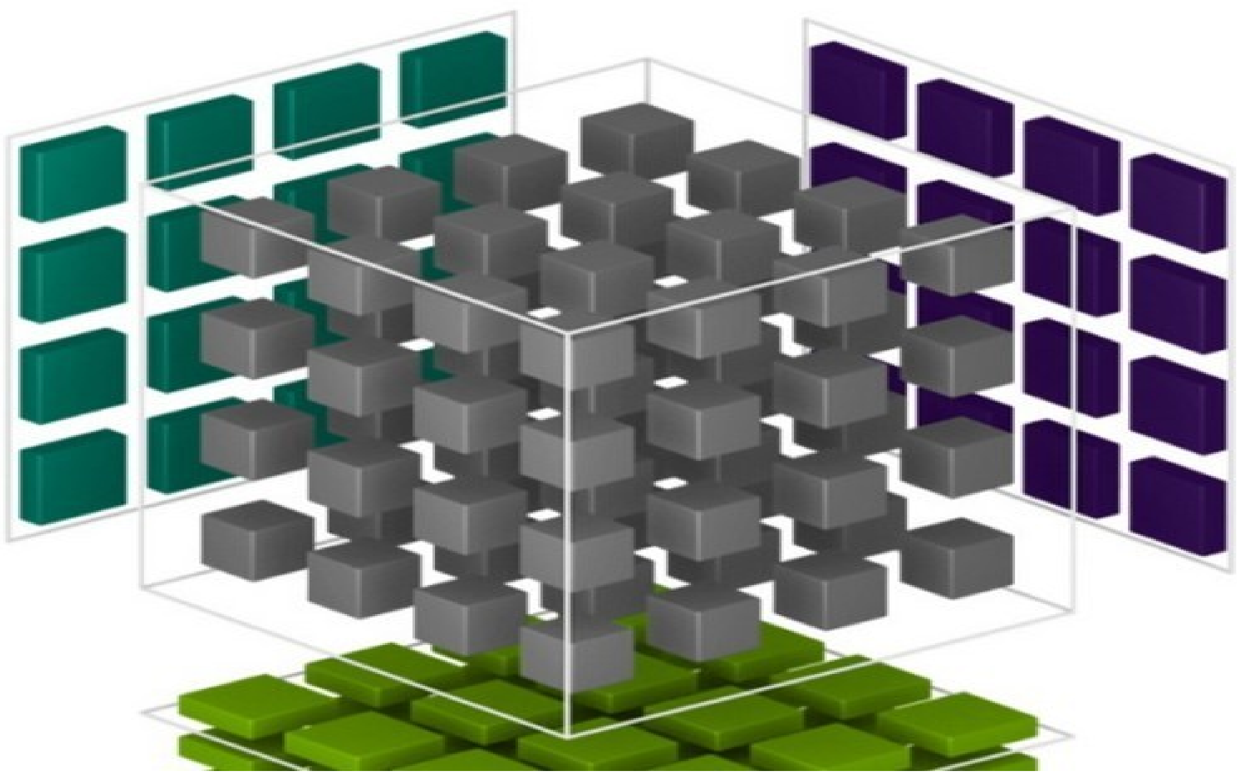
Oct 17, 2017 English

Programming Tensor Cores in CUDA 9

By Jeremy Appleyard and Scott Yokim

Discuss (14) +1 Like

Tags: CUDA, cuDNN, FP16, Linear Algebra, Mixed Precision, Tensor Cores, Volta



How to use Tensor Cores: Libraries, e.g. cuBLAS, cuDNN

Directly in CUDA C++ (since CUDA 9.0):

#include <mma.h>

using namespace nvcuda;

Tensor Cores are exposed in CUDA 9.0 via a set of functions and types in the nvcuda::wmma namespace (**W**arp **M**atrix **M**ultiply **A**ccumulate)

```
// Loop over the K-dimension
for (int i = 0; i < K; i += WMMA_K) {
    int aRow = warpM * WMMA_M;
    int aCol = i;
    int bRow = i;
    int bCol = warpN * WMMA_N;

    // Bounds checking
    if (aRow < M && aCol < K && bRow < K && bCol < N) {
        // Load the inputs
        wmma::load_matrix_sync(a_frag, a + aRow + aCol * lda, lda);
        wmma::load_matrix_sync(b_frag, b + bRow + bCol * ldb, ldb);

        // Perform the matrix multiplication
        wmma::mma_sync(acc_frag, a_frag, b_frag, acc_frag);
    }
}
```

Warp size is the number of threads in a warp, which is a sub-division used in the hardware imple-mentation to coalesce memory access and instruction dispatch.

This is only the inner part of the CUDA kernel, for more see URL above!

Wrapping Up 1

Exercises (CUDA Lectures in afternoon)

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 , blockId.x, blockDim.x, gridDim.x	Threads, Blocks
(threadId.y, blockId.y, blockdim.y, gridDim.y	(matmul coming with 2D grids)
kernel<<<n,m>>> (...)	kernel calls
Kernel<<n,m,size>>(…)	kernel call with dyn. alloc. size
kernel<<<dimBlock,dimGrid>>>(…)	dim3 variable type (matmul)
__global__	device code
__shared__	shared memory on GPU
cudaMalloc / cudaFree	manage global memory of GPU
cudaMemcpy / cudaMemset	copy/set to or from memory
cudaGetDeviceProperties	get device properties in program
cudaEventCreate, cudaEventRecord, cudaEventSynchronize, cudaEventElapsedTime, cudaEventDestroy	CUDA profiling
AtomicAdd (on global or shared mem.)	atomic functions

Wrapping Up 3

What we have not yet learnt...

__constant__	constant memory on GPU
__device__	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)

Additional deeper material:

Lectures by Prof. Wen-Mei Hwu Chicago in Berkeley 2012 and Beijing 2013, see http://iccs.lbl.gov/workshops/tutorials.html
(down on page links to all lecture files, also available on request from spurzem@nao.cas.cn)

Lecture1: Computational thinking
Lecture2: Parallelism Scalability
Lecture3: Blocking Tiling
Lecture4: Coarsening Tiling
Lecture5: Data Optimization
Lecture6: Input Binning
Lecture7: Input Compaction
Lecture8: Privatization
See also:
http://freevideolectures.com/Course/2880/Advanced-algorithmic-techniques-for-GPUs/1