Parallel Computing

Timing and Debugging

Wrap-Up of CUDA

Matrix Multiplication

Histogram

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

Note: Jason Sanders uses HANDLE_ERROR instead of our ERR_CHECK

(.../00_error/cuda_error_check.h)

Before we start...

Some nice ideas:

/home/Tit4/lecture60/gpu-course/00_error/

(ERR_CHECK instead of HANDLE_ERROR)

/home/Tit4/lecture60/gpu-course/4_dot/dot-special-new.cu

(dynamic vector size allocation in kernel through <<<n,m,size>>>)

Recap of 6: dot_perfect.cu :

Fat Threads! New variable gridDim.x !

Use of gridDim.x * blockDim.x to get size of grid,

Relation to <<<n,m>> in kernel launch

Block Reduction on Host instead of AtomicAdd!

Also used for histogram later.

Note nice profiling nvprof used in 7_matmul/gpu_script.sh

https://docs.nvidia.com/cuda/profiler-users-guide/index.html

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

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

NVIDIA DEVELOPER ZONE   CUDA TOOLKIT DOCUMENTATION   Search

CUDA Toolkit v7.5

CUDA-GDB

▷ 1. Introduction
2. Release Notes
▷ 3. Getting Started
▷ 4. CUDA-GDB Extensions
▷ 5. Kernel Focus
▷ 6. Program Execution
▷ 7. Breakpoints & Watchpoints
▷ 8. Inspecting Program State
▷ 9. Event Notifications
▷ 10. Automatic Error Checking
▷ 11. Walk-Through Examples
▷ 12. Advanced Settings
A. Supported Platforms
B. Known Issues

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.

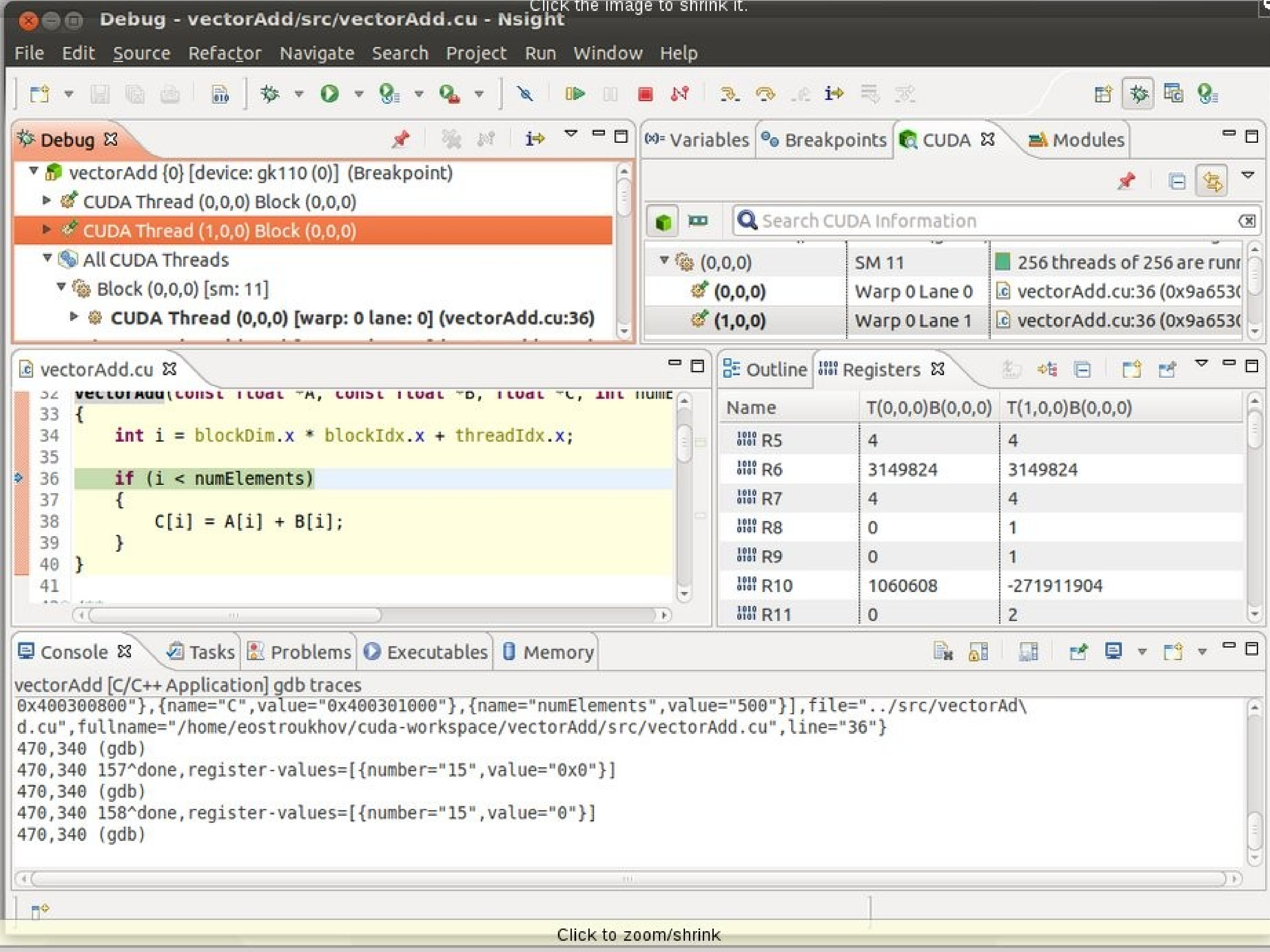
CUDA-GDB runs on Linux and Mac OS X, 32-bit and 64-bit. CUDA-GDB is based on GDB 7.6 on both Linux and Mac OS X.

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.



Click the image to shrink it.
Debug - vectorAdd/src/vectorAdd.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
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 runn
(0,0,0) Warp 0 Lane 0 vectorAdd.cu:36 (0x9a653
(1,0,0) Warp 0 Lane 1 vectorAdd.cu:36 (0x9a653
vectorAdd.cu
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
Console
Tasks
Problems
Executables
Memory
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)
Click to zoom/shrink

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.

8. histo - histogram using fat threads and atomic add on shared and global memory, timing
7. matmul - matrix multiplication with tiled access shared memory (expect Friday)

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	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/WS21/cuda/files/cuda-histograms.pdf

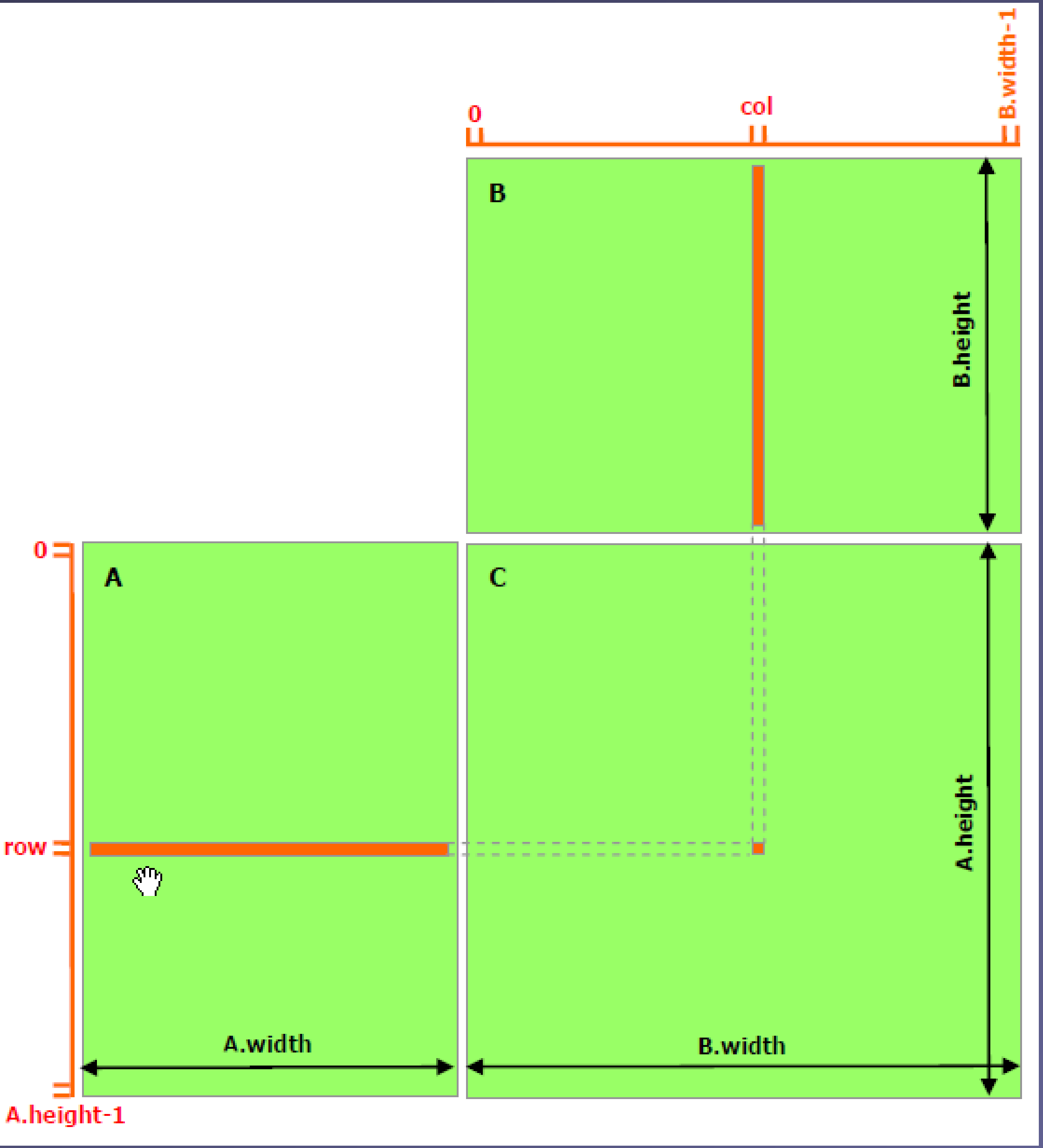
(Link on our webpage)

On kepler: 8_histo/

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

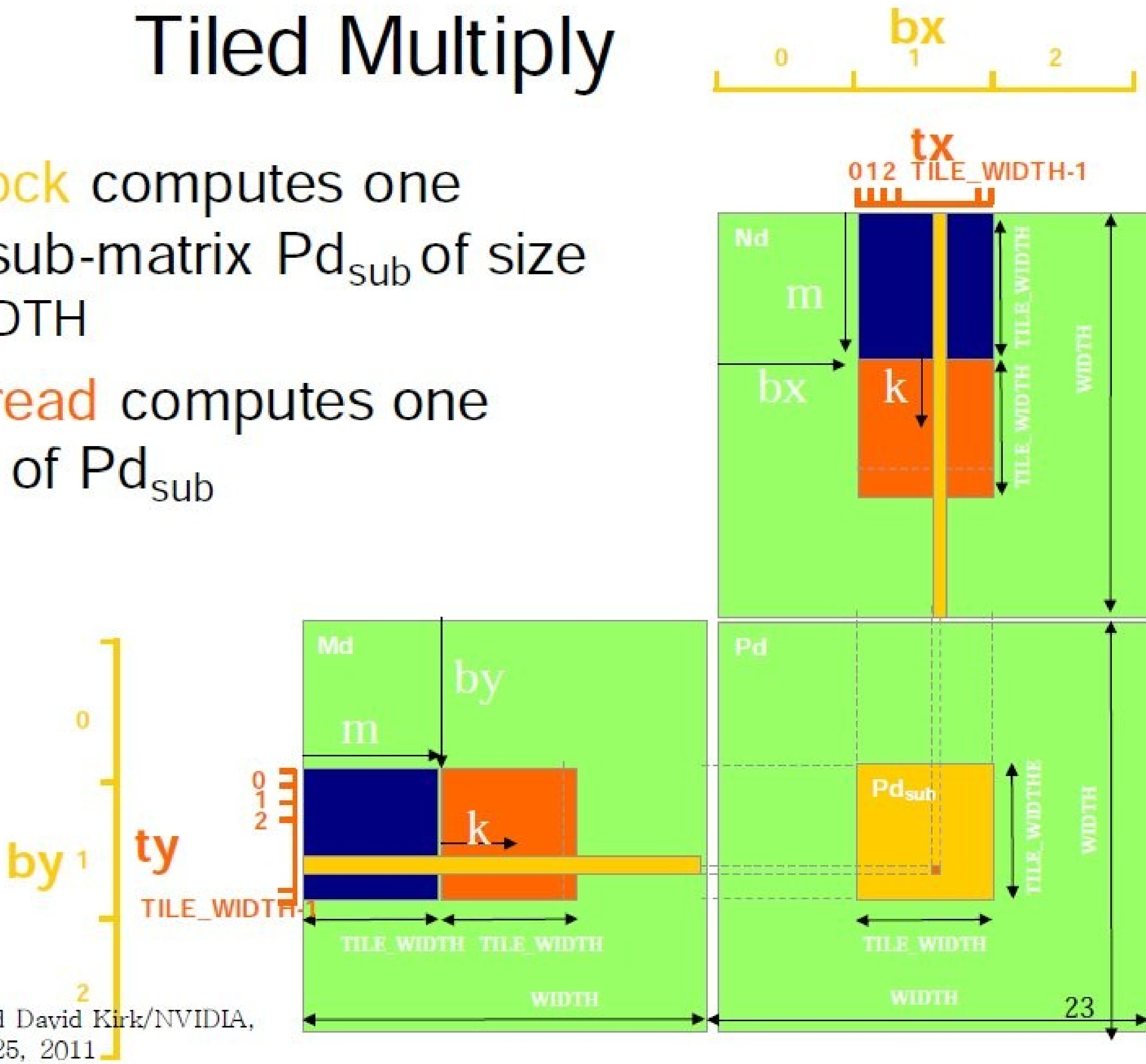
histo-no-atomic.cu (atomic only on global memory)

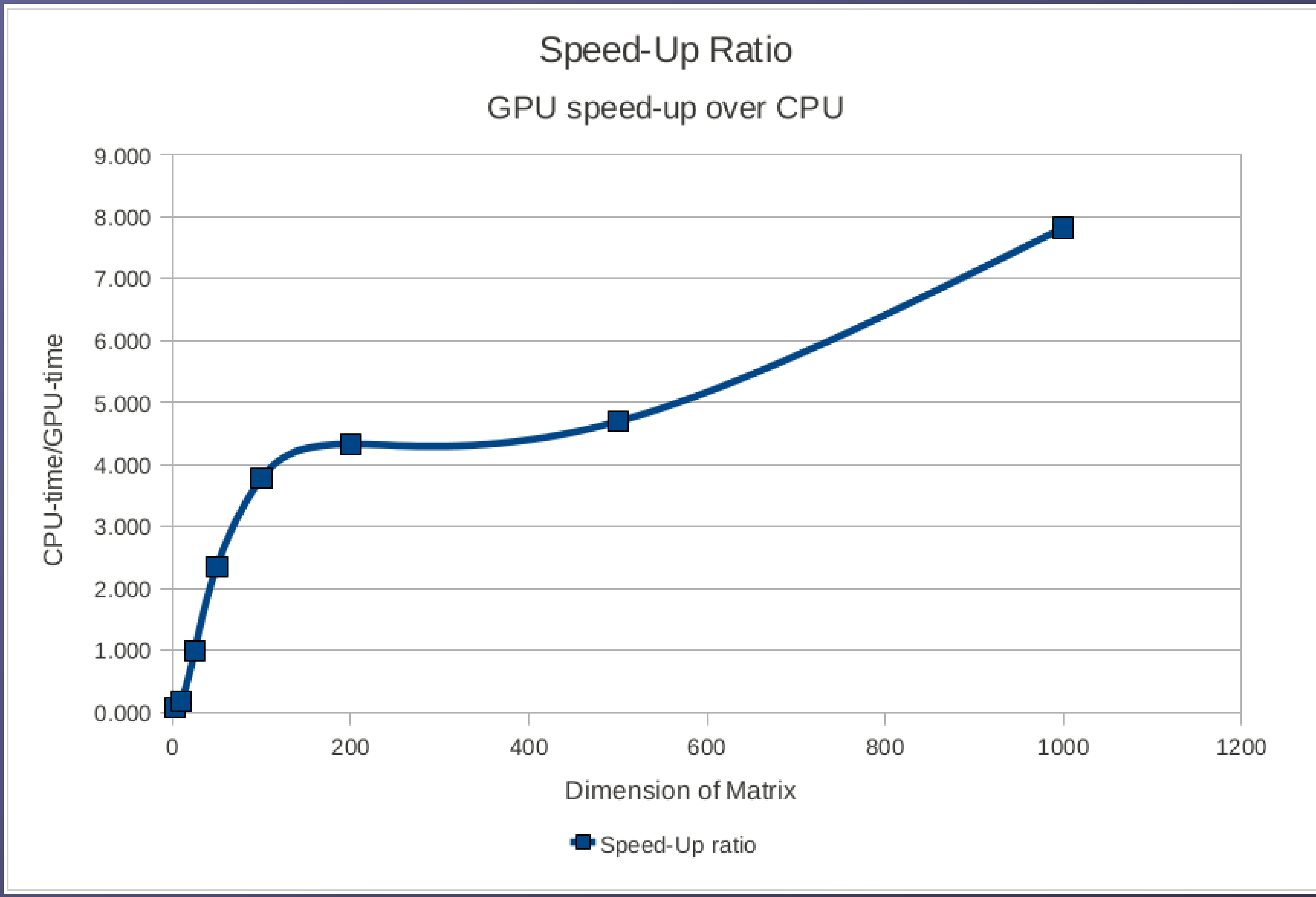
Matrix Intuitive Multiply



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
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0.000
CPU-time/GPU-time
0
200
400
600
800
1000
1200
Dimension of Matrix
Speed-Up ratio

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:

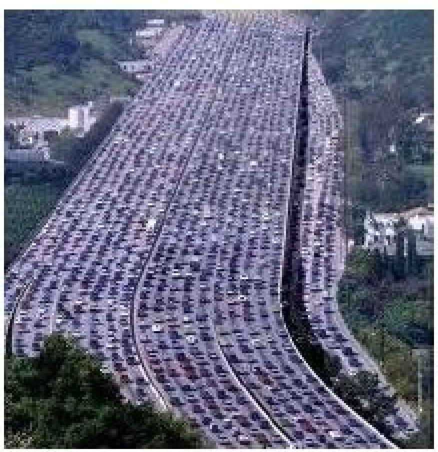
* Output files of small experiments on your lecture account (0_hello, 1_add, … , 7-matmul, 8-histo)
* Return two plots, one data file, and a few comments to your tutors Deadline? Agree with tutors, no strict deadline, but please NOT one day before you need the certificate! Outputs of the 8 Nbody runs on your lecture account.
* Notice: Student Queues will close Sunday, Mar 6, 23:59 (latest). You can run later, but contact me please spurzem@ari.uni-heidelberg.de

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

Massive Parallelism - Regularity



Main Hurdles to Overcome

- Serialization due to conflicting use of critical resources
- Over subscription of Global Memory bandwidth
- Load imbalance among parallel threads

Computational Thinking Skills

- The ability to translate/formulate domain problems into computational models that can be solved efficiently by available computing resources
  - Understanding the relationship between the domain problem and the computational models
  - **Understanding the strength and limitations of the computing devices**
  - **Defining problems and models to enable efficient computational solutions**

DATA ACCESS CONFLICTS

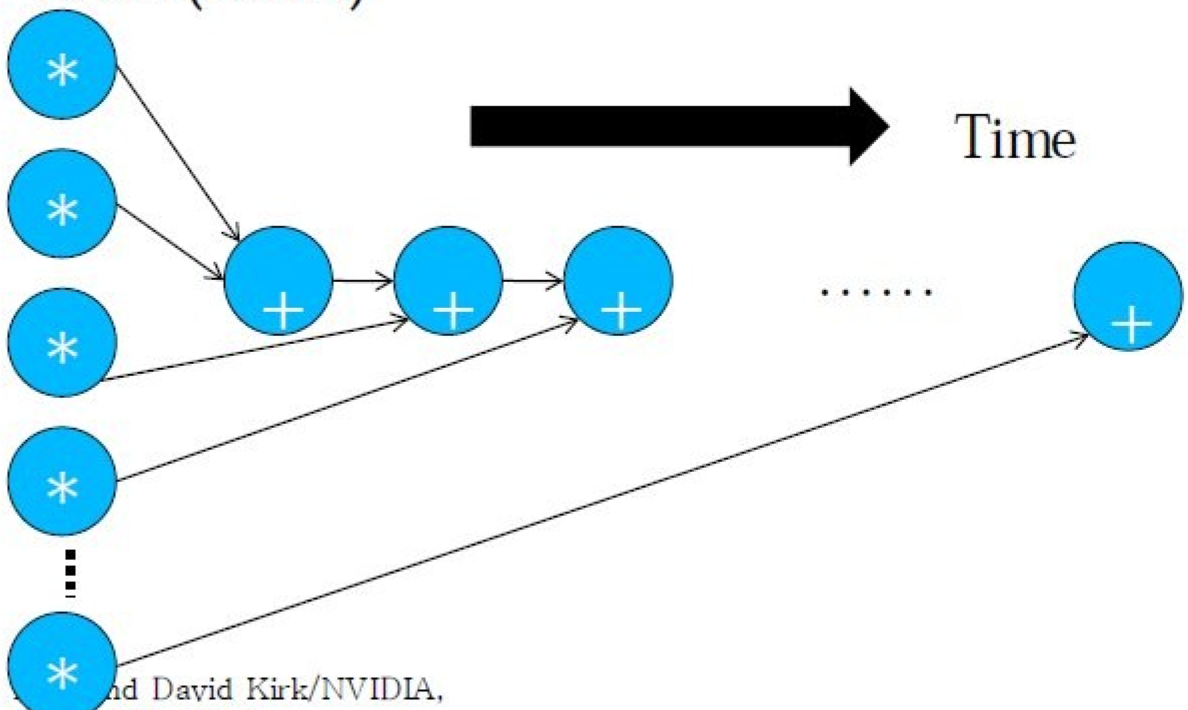
Conflicting Data Accesses Cause Serialization and Delays

- Massively parallel execution cannot afford serialization

- Contentions in accessing critical data causes serialization

A Simple Example

- A naïve inner product algorithm of two vectors of one million elements each
  - All multiplications can be done in time unit (parallel)
  - Additions to a single accumulator in one million time units (serial)



How much can conflicts hurt?

- Amdahl's Law
  - If fraction X of a computation is serialized, the speedup can not be more than 1/(1-X)
- In the previous example, X = 50%
  - Half the calculations are serialized
  - No more than 2X speedup, no matter how many computing cores are used

GLOBAL MEMORY BANDWIDTH

Global Memory Bandwidth

Ideal

Reality

Global Memory Bandwidth

- Many-core processors have limited off-chip memory access bandwidth compared to peak compute throughput
- Fermi
  - 1 TFLOPS SPFP peak throughput
  - 0.5 TFLOPS DPFP peak throughput
  - 144 GB/s peak off-chip memory access bandwidth
    - 36 G SPFP operands per second
    - 18 G DPFP operands per second
  - To achieve peak throughput, a program must perform 1,000/36 = ~28 SPFP (14 DPFP) arithmetic operations for each operand value fetched from off-chip memory

LOAD BALANCE

Load Balance

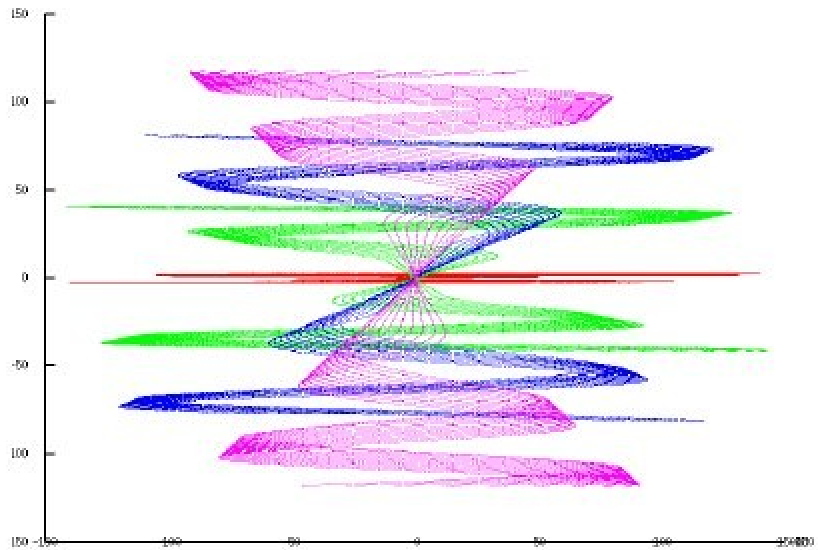
- The total amount of time to complete a parallel job is limited by the thread that takes the longest to finish

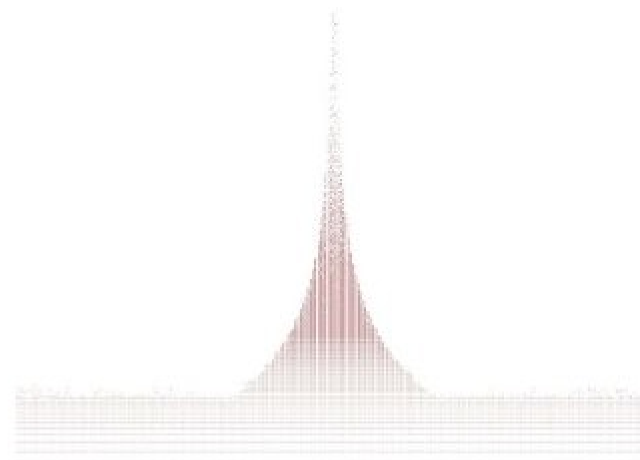
How bad can it be?

- Assume that a job takes 100 units of time for one person to finish
  - If we break up the job into 10 parts of 10 units each and have fo10 people to do it in parallel, we can get a 10X speedup
  - If we break up the job into 50, 10, 5, 5, 5, 5, 5, 5, 5, 5 units, the same 10 people will take 50 units to finish, with 9 of them idling for most of the time. We will get no more than 2X speedup.

How does imbalance come about?

- Non-uniform data distributions
  - Highly concentrated spatial data areas
  - Astronomy, medical imaging, computer vision, rendering, …
- If each thread processes the input data of a given spatial volume unit, some will do a lot more work than others





Eight Algorithmic Techniques (so far)

Technique	Contention	Bandwidth	Locality	Efficiency	Load Imbalance	CPU Leveraging
Tiling		X	X			
Privatization	X		X			
Regularization				X	X	X
Compaction		X				
Binning		X	X	X		X
Data Layout Transformation	X		X			
Thread Coarsening	X	X	X	X		
Scatter to Gather Conversion	X					

http://courses.engr.illinois.edu/ece598/hk/

You can do it.

- Computational thinking is not as hard as you may think it is.
  - Most techniques have been explained, if at all, at the level of computer experts.
  - The purpose of the course is to make them accessible to domain scientists and engineers.

ANY MORE QUESTIONS?