

Practical GPU Programming

Dominik Göddeke (TU Do)

Robert Strzodka (MPII), Christian Sigg (NVIDIA)

**38th SPEEDUP Workshop on High-Performance
Computing**

EPF Lausanne, Switzerland, September 7-8, 2009

http://www.speedup.ch/workshops/w38_2009.html

Lesson 0

- **Goal: Get CUDA running on these machines**
 - And figure out which hardware is installed
- **Log on**
 - Username: anonym
 - Passwort: guest
- **Set up CUDA**
 - This is specific to this lab
 - On your machines at home/work, follow the instructions of the CUDA toolkit installer
 - Open a terminal
 - Set up the environment:
 - `source /usr/local/bin/nvidia.bash`

Lesson 0

- **Download course material**
 - From machine 075
- **Figure out details of the CPU**
 - `cat /proc/cpuinfo`
- **Compile and run first sample**
 - `make lesson0`
 - `./lesson0`
- **Interpret the output**

Lesson 1

- **Lesson 2: saxpy**

- for (i=0; i<N; i++) y[i] = alpha*x[i] + y[i]

- **Live code walkthrough**

- Device initialization
 - CUDA error checking
 - Host and device kernel
 - Launch configuration

- **Compile and run**

- make lesson1 && ./lesson1

- **Questions**

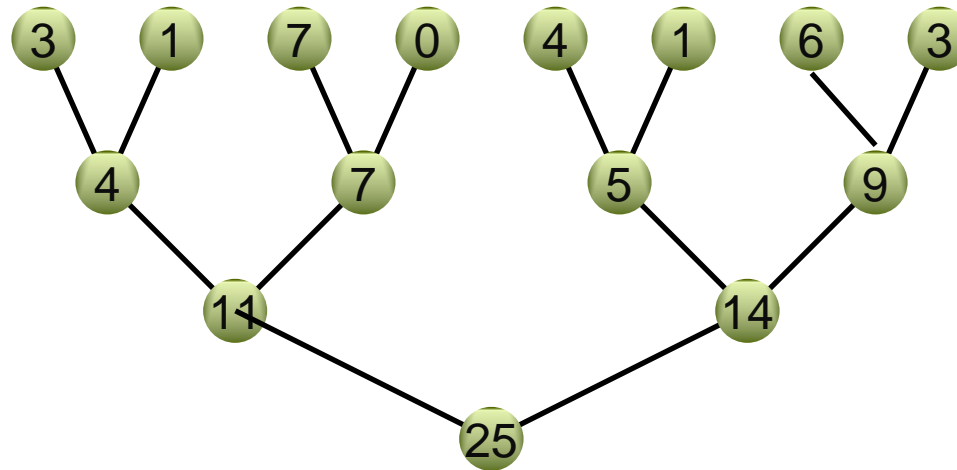
- Perform experiments for increasing problem size
 - Performance of our kernel vs. CUBLAS reference implementation

Lesson 2

- **Reduction: 2-norm**

- $\text{Sum} = x[0] * x[0];$
- For $(i=1; i < N; i++)$ $\text{sum} += x[i] * x[i];$
- $\text{Result} = \text{sqrtf}(\text{sum});$

- **Idea: Binary tree**



Lesson 2

- **Some hints:**

- Do not compute the final result on the device, write one kernel that creates a local intermediate result corresponding to each thread block
 - New device array that stores all the partial sums
 - Partial result array is read back to host and final result is computed there
- You are free to write any parallel reduction you want
 - Most trivial version: One thread per block, sums up e.g. 256 values
- For better performance, make clever use of shared memory
 - Tree summation from leaves to root is performed in shared memory
 - `extern __shared__ float smem[];`
- For even better performance, avoid very fine-grained branches
 - `If (threadIdx.x % 2 == 0) ...`

Lesson 2

- **Framework code is provided**
 - CPU reference
 - Memory allocation and transfer
 - CUBLAS version
 - Timing and debugging infrastructure
- **Quick framework code walkthrough**
- **Correctness and performance**
 - How fast are you compared to CUBLAS?