The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program

of the Lobachevsky State University of Nizhni Novgorod

among the world's research and education centers

Strategic initiative

“Achieving leading positions in the field of supercomputer technology

and high-performance computing”

Introduction to GPU programming

Lecture 5. CUDA thread execution and memory hierarchy

Nizhni Novgorod

2014

Author: S.I. Bastrakov

OBJECTIVES

The objective of this lecture is to discuss GPU architecture, thread execution and memory hierarchy on GPU.

ABSTRACT

This lecture focuses on thread execution and memory hierarchy on GPU. We investigate GPU architecture and discuss scheduling of threads onto CUDA cores, warp mechanism and SIMT. We consider memory classes and show efficient patterns for global and shared memory.

BRIEF OVERVIEW

An NVIDIA GPU consists of several multiprocessors, each multiprocessor contains several CUDA-cores, shared memory and caches, one or several functional units (SFU), instruction prefetch and one or several warp schedulers.

Threads of the same block are executed on the same multiprocessor, they can communicate using shared memory and __syncthreads() barrier synchronization. Threads of different block can communicate only via global memory. There are atomic operations in shared and global memory.

When a kernel is launched, blocks are distributed between multiprocessors, threads of a block are executed by CUDA-cores of a multiprocessor. If there are enough resources, several blocks can be concurrently executed on the same multiprocessor. Large amount of blocks helps automatic scaling on GPUs with more multiprocessors.

All threads running on the same multiprocessor are grouped in warps; a warp consists of threads with sequential identifiers, warp size is currently 32. Warp scheduling is done automatically. CUDA-cores of a multiprocessor execute same instruction for all threads in a warp (SIMT, Single Instruction Multiple Thread). Threads of a warp are always synchronized. Programming in scalar terms: kernel code for one thread, can use conditions.

There are several kinds of memory on GPUs: register, local, shared, global, texture, constant. Shared memory and L1 global memory cache are physically located in the same memory region. Host side controls only global memory using cudaMalloc, cudaFree, cudaMemcpy and cudaMemcpyAsync routines.

Due to SIMT execution, threads in a warp access global memory simultaneously. If memory addresses accessed by threads of the same warp lie in nearby cells, memory access of the whole

warp is performed as a single operation, this is called coalescing. Otherwise memory access is serialized.

Shared memory has latency 4 clocks and small size (16 to 48 KB). A typical use pattern for shared memory is:

- Load intensively used data from global memory.
- Synchronize if necessary.
- Compute using loaded data.
- Synchronize if necessary.
- Write results back to global memory.

Efficient usage of shared memory requires avoiding bank conflicts, although not as important as coalescing for global memory.

FOR STUDENTS

General information about NVIDIA GPU architecture and programming is presented in [1]. CUDA C language is described in [1, 2]. Empirical evaluation of performance of various memory access patterns is presented in [3].

REFERENCES

1. Sanders J., Kandrot E. CUDA by Example: An Introduction to General-Purpose GPU Programming. – Addison-Wesley Professional, 2010. – 312 p.
2. Farber R. CUDA Application Design and Development. – Morgan Kaufmann, 2011. – 336 p.
3. NVIDIA CUDA C Programming Guide. [http://docs.nvidia.com/cuda/cuda-c-programming-guide/].

INDIVIDUAL WORK

1. What are the ways of communication between threads in the same block and in different blocks?
2. What are main differences between global and shared memory?
3. Describe how conditional statements are executed in SIMT.