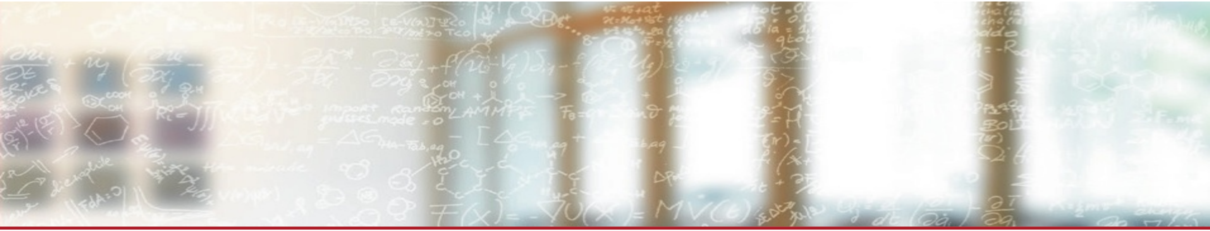
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Introduction to the GPU architecture

High-Performance Computing with Python

*Vasileios Karakasis, **Theofilos Manitaras**, CSCS*

June 21-23, 2022

Overview

- Introduction to the GPU architecture
  - Differences from the CPUs
  - Execution model
  - Memory model
- Programming GPUs with Numba

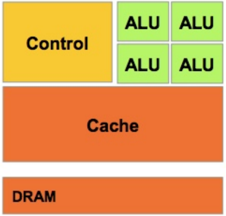
Low latency or high throughput

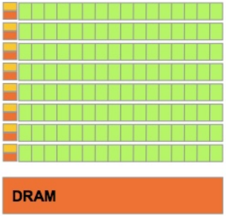
CPU

- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution

GPU

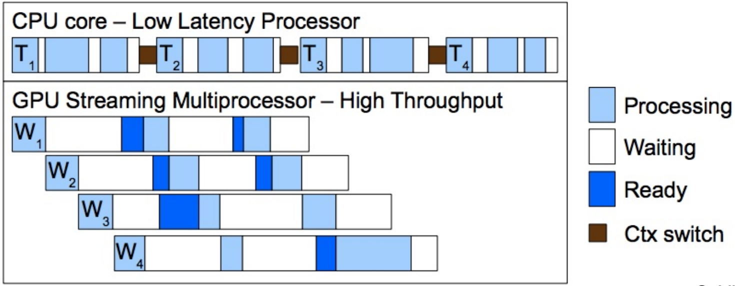
- Optimized for data-parallel, throughput computation
- Architecture tolerant of memory latency
- More transistors dedicated to computation





GPUs are throughput devices

- CPU cores are optimized to minimize latency between operations
- GPUs aim to minimize latency between operations by scheduling multiple thread bundles (warps)



Current applications not designed for many-core

- Exposing sufficient fine-grained parallelism for multi- and many-core processors is hard
- New programming models are required
- New algorithms are required
- Existing code has to be rewritten or refactored

Current applications not designed for many-core

- Exposing sufficient fine-grained parallelism for multi- and many-core processors is hard
- New programming models are required
- New algorithms are required
- Existing code has to be rewritten or refactored

…and compute nodes are under-utilized

- Users are not getting the most out of allocations
- The amount of parallelism on-node is only going to increase!

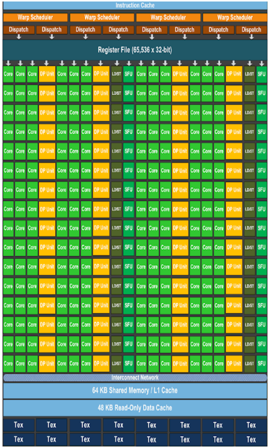
Architecture overview

The P100 GPU (Pascal architecture)

© NVIDIA Corp. 2016

The SM architecture

- Multiple lightweight single-threaded cores (64 on P100)
- Synchronous execution on groups of 32 threads/cores
  - All 32 threads execute the same instruction
- Very large register file partitioned per core (256 KB)
- Warp scheduler
  - Picks up the next ready warp
  - Very fast warp switching
- User-managed shared fast memory (64 KB on P100)



Execution model

Host-directed execution

- CPU sets up and launches *kernels* on the GPU
- CPU manages the memory on the GPU
  - Allocations, transfers in and out of the GPU

Execution model

Host-directed execution

- CPU sets up and launches *kernels* on the GPU
- CPU manages the memory on the GPU
  - Allocations, transfers in and out of the GPU

Unified memory between CPU and GPU

- Virtual address space shared between CPU and GPU
- The CUDA driver and the hardware take care of the page migration
- Introduced with Kepler, significantly improved with Pascal

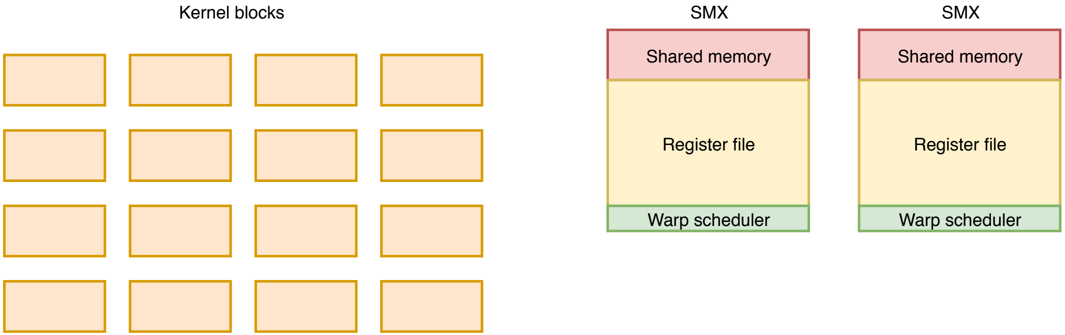
Execution model

How this huge parallelism is managed on the GPU?

- An application launches *kernels* to be executed on the GPU
- Each kernel comprises several *blocks* or *gangs* of threads
- A thread block may only run on a single SM
- Multiple thread blocks might be accomodated in a single SM, if…
  - there are enough registers,
  - there is enough shared memory or
  - hardware limits are not reached (active warps)
- Warps of any *active* block may be scheduled to run on the SM cores

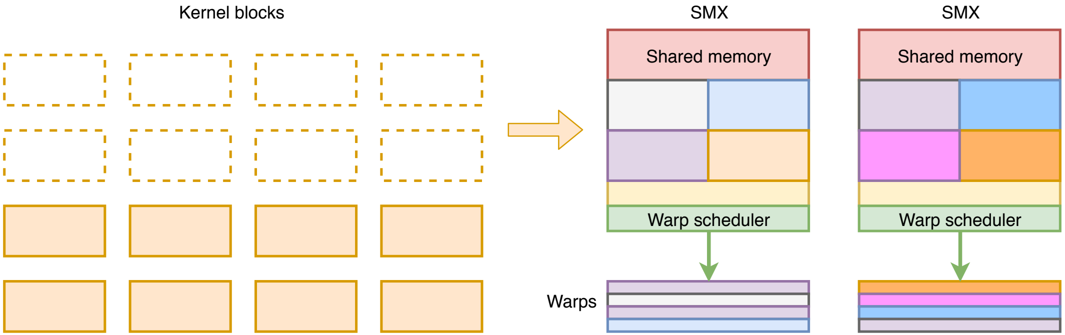
Execution model

How GPU threads are executed on the SMs



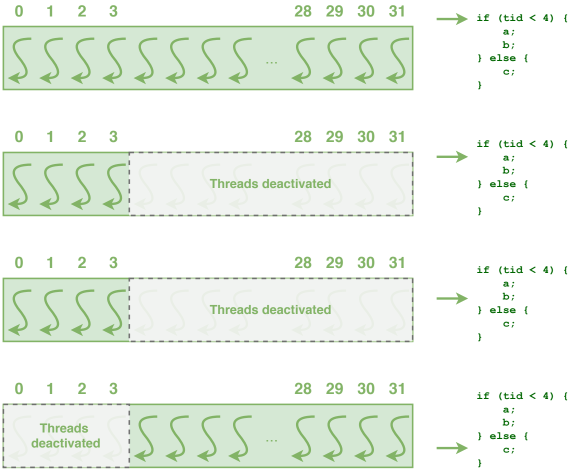
Execution model

How GPU threads are executed on the SMs



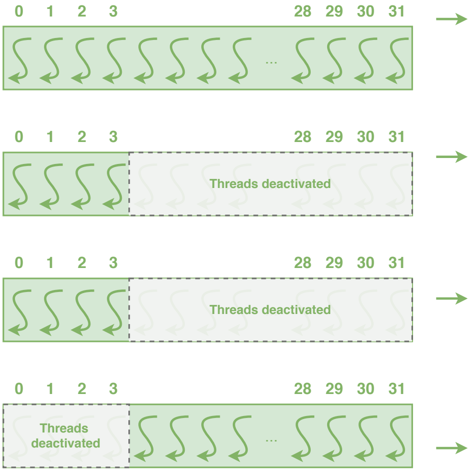
Execution model

Branching – Can individual threads execute different code?



Execution model

Branching – Can individual threads execute different code?




```
if (tid < 4) {
    a;
    b;
} else {
    c;
}
```

```
if (tid < 4) {
    a;
    b;
} else {
    c;
}
```

```
if (tid < 4) {
    a;
    b;
} else {
    c;
}
```

```
if (tid < 4) {
    a;
    b;
} else {
    c;
}
```

- Each thread in a warp can take a different path, but…
- warp is executing all branches, deactivating the non participating threads.

Execution model
Implications

- Lots of parallelism is needed to cover execution latencies
  - Enough warps must be available for scheduling
- Global synchronization is not possible
  - Not all the blocks of a kernel run simultaneously
  - *Synchronization is only possible within the threads of a block*
- If program's control flow diverges within a warp → redundant execution
  - Both branches are executed by the warp redundantly

Memory model
Memory hierarchy

- Global high bandwidth memory (732 GB/s on P100)
  - Accessible from all thread gangs
  - Data persistent across kernel invocations
  - Memory accesses of warp threads are *coalesced* into one or two memory transactions if they are properly aligned and regular
- L1 cache/Shared memory
  - Shared among the threads of a single thread gang
  - One-cycle access latency, if warp threads access different locations
  - Software or hardware managed
  - No cache coherency across SMs
  - No sequential consistency → enforced by synchronization primitives

Memory model

Memory hierarchy (cont'd)

- Local memory
  - Not visible to the programmer
  - The compiler may place there automatic variables
- Constant memory
  - Cached part of the global memory used for storing constants
  - Visible to the programmer
- Texture and surface memory
  - Cached part of the global memory optimized for 2D accesses

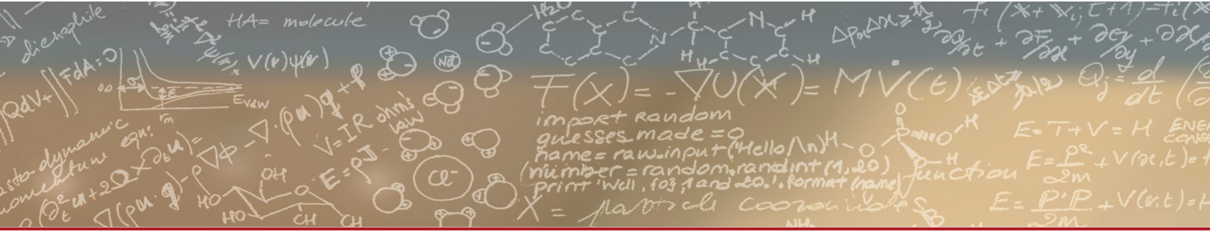
How to take advantage of the GPUs?

- Low-level native APIs
  - CUDA C/C++ for NVIDIA GPUs
  - ROCm for AMD GPUs
- Using directives, such as OpenMP 4.5 / OpenACC
- Using higher-level libraries, such as Kokkos etc.

- Higher-level languages, such as Python
  - **Numba**, **CuPy**, PyCUDA, PyOpenCL etc.

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention