GPU architecture and fundamentals

PRACE Summer of HPC 2022 - Day 4

Leon Bogdanović

University of Ljubljana, FME, LECAD lab

Graphics accelerators: definition

NVIDIA A100

Graphics accelerators or **graphics processing units** (**GPUs**) are devices with:

- many highly parallel streaming multiprocessors
- very high bandwidth memory

Applications:

- for intensive 3D graphical rendering, ray tracing etc. (**graphics** applications)
- for GPGPU (General Purpose GPU) computing (**scientific** and **engineering** applications)

Motivational example: CUDA ray tracing

- **clone the repository** from bitbucket to your `gpu02.hpc.fs.uni-lj.si` account:

```
git clone https://bitbucket.org/lecad-peg/sohpc-accelerators.git
```

- **to build** the executable with the CUDA ray tracer follow these steps:

```
cd sohpc-accelerators/CUDA_ray_tracing/gpu02
source setupenv.sh
make
```

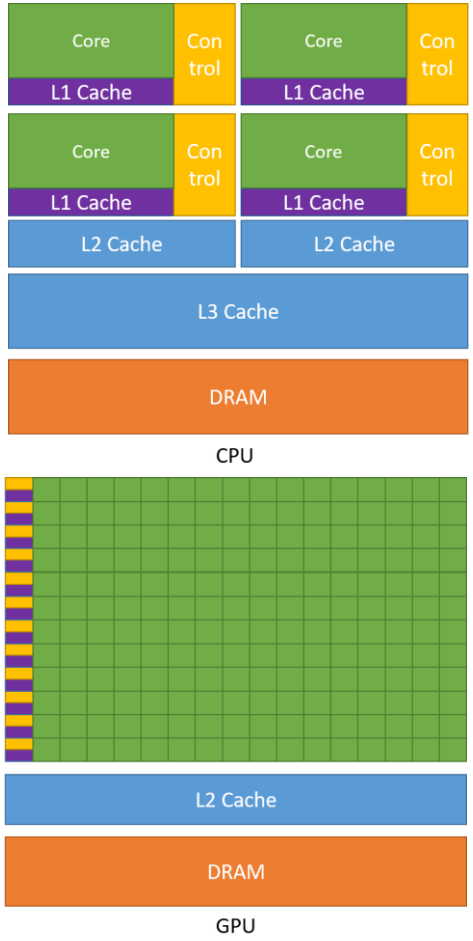
- **to run** the CUDA ray tracer execute:

```
./run.sh
```

or

```
./a.out
```

Graphics accelerators: architecture



CPU and GPU schematics (source: nvidia.com)

GPU architecture (typical):

- **global memory**: 0.5-80 GB, very high bandwidth (up to 2000 GB/s)
- **comparison to CPU** architecture:
  - many more ALUs (green rectangles) than in a CPU
  - can control simple, highly parallel workloads well
  - a “core” (green rectangle) in a GPU is an ALU, a core in a CPU is comprised of ALUs and FPUs
  - less cache memory
- **streaming multiprocessors** (SM):
  - groups of many parallely executing ALU cores
  - many L1 cache registers (32-64 KB)
  - very fast shared memory

GPUs: consumer grade vs. high-end

NVIDIA GeForce 930MX

Output from *deviceQuery*:

Total amount of global memory:

2004 MBytes (2101870592 bytes)

(3) Multiprocessors, (128) CUDA Cores/MP:

384 CUDA Cores

GPU Max Clock rate: 1020 MHz (1.02 GHz)

Memory Bus Width: 64-bit

L2 Cache Size: 1048576 bytes

Output from *bandwidthTest*:

Device to Device Bandwidth, 1 Device(s)

Transfer Size (Bytes) Bandwidth(MB/s)

33554432 13193.8

NVIDIA A100-PCIE-40GB

Output from *deviceQuery*:

Total amount of global memory:

40536 MBytes (42505273344 bytes)

(108) Multiprocessors, (64) CUDA Cores/MP:

6912 CUDA Cores

GPU Max Clock rate: 1410 MHz (1.41 GHz)

Memory Bus Width: 5120-bit

L2 Cache Size: 41943040 bytes

Output from *bandwidthTest*:

Device to Device Bandwidth, 1 Device(s)

Transfer Size (Bytes) Bandwidth(MB/s)

33554432 1053823.2

GPUs for High Performance Computing (HPC)

NVIDIA cards (flagship cards historically)

Model	No. of cores	Memory [GB]	Bandwidth [GB/s]	FP32 [TFlops]
Kepler K40	2280	12 (GDDR5)	240	4.3
Pascal P100	3584	16 (HBM2)	732	10.6
Volta V100	5120	32 (HBM2)	900	15.7
Ampere A100	6912	40 (HBM2)	1638	19.5

AMD Radeon Instinct cards

Model	No. of cores	Memory [GB]	Bandwidth [GB/s]	FP32 [TFlops]
Radeon MI8	4096	4 (HBM)	512	8.2
Radeon MI25	4096	16 (HBM2)	484	12.3
Radeon MI50	3840	16 (HBM2)	1024	13.4
Radeon MI60	4096	32 (HBM2)	1024	14.7

A consumer-grade GPU: just for comparison

Model	No. of cores	Memory [GB]	Bandwidth [GB/s]	FP32 [TFlops]
NVIDIA GeForce 930MX	384	2 (DDR3)	14.4	0.765

HPCFS gpu02 login node: Installing Jupyter notebook

Login to your `gpu02.hpc.fs.uni-lj.si` account and complete the following steps:

- Open a new Konsole shell and load the modules (every time a new shell is opened):

```
module purge
module use /opt/pkg/ITER/modules/all
module load NVHPC/21.2
module load Python/3.8.6-GCCcore-10.2.0
```

- Create a Python virtual environment with needed Python packages (just once):

```
python3 -m venv notebook
notebook/bin/pip install --upgrade pip jupyterlab notebook
```

- Activate the virtual environment `notebook` and start Jupyter Notebook in browser (every time a new shell is opened) :

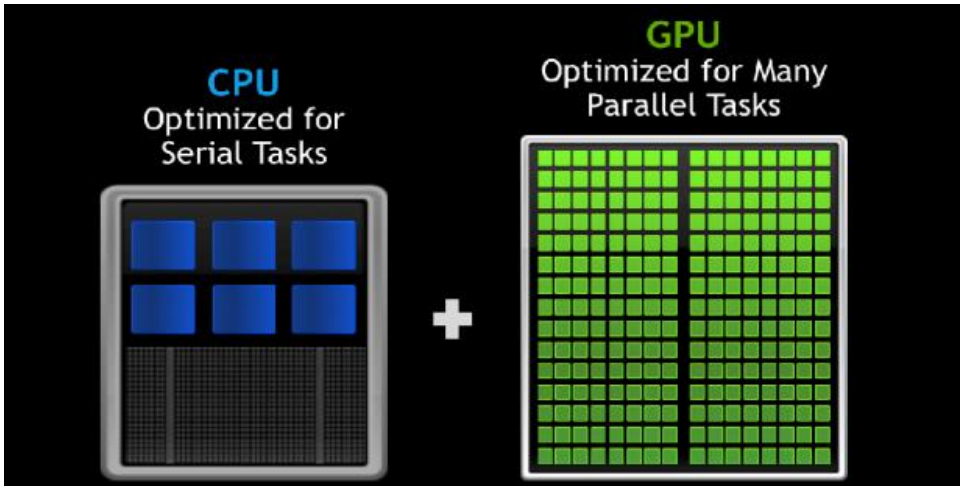
```
source notebook/bin/activate
(notebook) [bogdanl@gpu02 ~]$ cd sohpc-accelerators/notebooks
(notebook) [bogdanl@gpu02 notebooks]$ jupyter notebook
```

Exercise 1: Information and compute capabilities of a GPU

Open the notebook `01_GPU_info.ipynb` and complete the following tasks:

- find general info on the GPU
- find the diagnostic programs `deviceQuery` and `bandwidthTest`
- execute the diagnostic utilities to determine the main characteristics of the GPU (number of SMs, number of CUDA cores, global memory available, memory bandwidth)
- compile and run the OpenCL diagnostic program to determine the OpenCL compute capability of the GPU

GPGPU programming

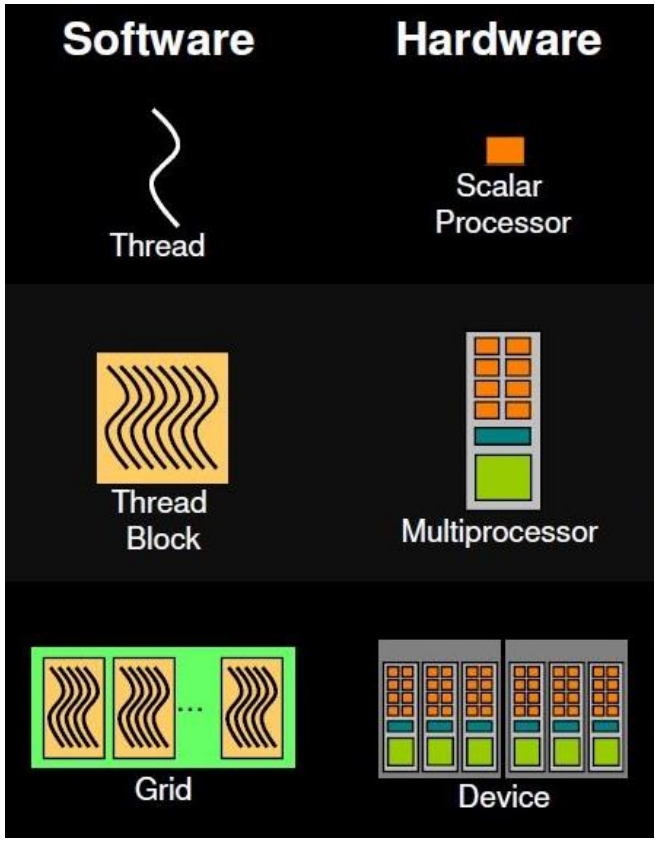


CPU vs. GPU (source: nvidia.com)

General Purpose GPU (GPGPU) programming:

- GPUs are used for **accelerating intensive computational tasks** rather than accelerating graphics tasks
- CPU and GPU are in principle **separate devices** with **separate memory space**
- **GPU is a co-processor to CPU:**
  - **CPU**: Optimized for **serial tasks** and **low-latency** access
  - **GPU**: Optimized for **many parallel tasks** and **throughput**

GPU execution model and terminology



GPU execution model (source: nvidia.com)

GPU execution model:

- uses the concept of a **grid of thread blocks**
- multiple blocks in a grid map onto the many SMs
- each block contains many threads mapping onto the cores in an SM

Terminology:

- **device** is a general reference to the GPU
- **host** is a reference reserved for the CPU

Computing acceleration types

Flexibility

Existing applications	GPU libraries	Directive-based methods	Special programming languages
• GPU accelerated applications • No knowledege needed	• e.g., cuFFT, cuBLAS • “drop-in“ acceleration	• e.g., OpenMP, OpenACC • “automatic“ parallelization	• e.g., CUDA, OpenCL • maximum flexibility

Easy to use