Assignment 1: GPU Architecture and CUDA Basics

- Assignment Group 3
- Giovanni Prete and Max Harrison

Question 1 - Reflection on GPU-Accelerated Computing

1) Architectural differences between GPUs and CPUs

Three main architectural differences between GPUs and CPUs:

1) Specialised vs. general: CPU must be highly general, so they have much more hardware and infrastructure for control flow. They need to be able to handle a wide array of instructions and essentially any kind of computational work. GPUs are more specialised for arithmetic work, so they have many more ALUs (arithmetic logic units) than CPUs. They don't need to handle the same variety as CPUs, so they have much less infrastructure for control flow.
2) Many cores vs. few: CPU cores must be highly flexible in order to provide this generality, making them more expensive. As GPU cores can be more specialised, they are cheaper in both cost and power requirements. This allows GPUs to have many more cores than CPUs.
3) Versions of efficiency: CPUs and GPUs have different models of efficiency for which their architectures are optimised for. CPUs have a latency-oriented architecture: they are designed in order to minimise the latency of single tasks. Data is not necessarily local, so CPUs require large caches to reduce latency. Contrastingly, GPUs have a throughput-oriented architecture: it is designed to maximise the total amount of computation in a given amount of time. Access to data is regular, so there is no need for large caches.

2) Supercomputers that use GPUs

Almost all of the top 10 supercomputers use some form of GPU accelerator, whether in a discrete GPU or in an accelerator containing both CPU and GPU cores (like the AMD Instinct series).

The name of the supercomputer and their GPU vendor is provided in the table below:

Supercomputer Name	Accelerator Vendor	Accelerator Model
El Capitan	AMD	Instinct MI300A
Frontier	AMD	Instinct MI250X

Supercomputer Name	Accelerator Vendor	Accelerator Model
Aurora	Intel	Data Center GPU Max
JUPITER Booster	NVIDIA	GH200 Superchip
Eagle	NVIDIA	H100
HPC6	AMD	Instinct MI250X
Supercomputer Fugaku	-	-
Alps	NVIDIA	GH200 Superchip
LUMI	AMD	Instinct MI250X
Leonardo	NVIDIA	A100 SXM4 64 GB

The "Supercomputer Fugaku" is the only system on the list to not use accelerators.

3) Power efficiency

Power efficiency is quantified by Performance / Power, e.g. throughput as in FLOPs per watt power consumption. The power efficiency for the top 10 supercomputers is provided below:

Supercomputer Name	Power Efficiency (GFlops/Watt)
El Capitan	58.89
Frontier	54.98
Aurora	26.15
JUPITER Booster	60.62
Eagle	-
HPC6	56.48
Supercomputer Fugaku	14.78
Alps	61.05
LUMI	53.43
Leonardo	32.19

We note that the only system to not use accelerators in the top 10, the "Supercomputer Fugaku", has a much lower power efficiency than the other systems. The "Aurora" system also has a relatively low power efficiency, and is the only one to use Intel accelerators.

Question 2 - Your First CUDA Program and GPU Performance Metrics

1) Code comments

The code is contained in `Q2/vecAdd.cu` with the corresponding comments.

2)

For each cell in the vector, two reads and one operation is performed. Thus, for a vector length of N we have that N floating operations and $2N$ memory reads are being performed by the kernel.

3)

The threads per block is set to 32 for all inputs, so for $N = 512$ there are $N/32 = 16$ thread blocks and 512 CUDA threads.

4)

4. For N=512, profile your program with Nvidia Nsight. What Achieved Occupancy did you get? You might find https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison

5)

The program still works as originally written.

6)

For $N = 263,149$, there are $N/32 = 8,224$ thread blocks (as we round up to the nearest whole number) and $8,224 \cdot 32 = 263,168$ CUDA threads.

7)

7. For N=263149, profile your program again with Nvidia Nsight. What Achieved Occupancy do you get now?

8)

8. Further increase the vector length (try 10-16 different vector length), plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel (3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions (see tutorial).

3 - 2D Dense Matrix Multiplication

4 - Rodinia CUDA Benchmarks and Comparison With CPU