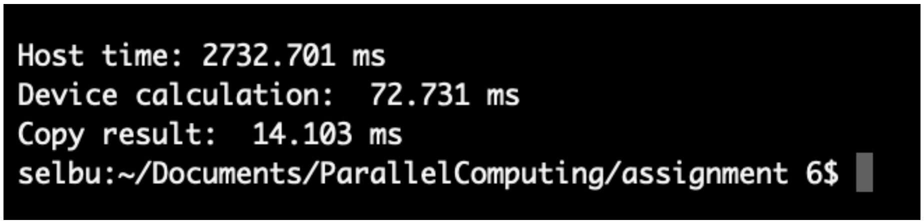
A06 - Assignment 6

Student: **Deidier Simone - 133020**

Answers to the theory questions

1. **Yes, we got a speedup**, these are the measured times:



```
Host time: 2732.701 ms
Device calculation:  72.731 ms
Copy result:  14.103 ms
selbu:~/Documents/ParallelComputing/assignment 6$
```

Figure 1: *Sequential code timing, cuda kernel and memory operations.*

If we consider the single calculation of data as a time frame, the **speedup** we have achieved with the use of a **GPU** is:

$$\text{Speed-up} = \frac{T_{\text{CPU}}}{T_{\text{GPU-calc}}} = \frac{2732.701ms}{72.731ms} \approx 37.573$$

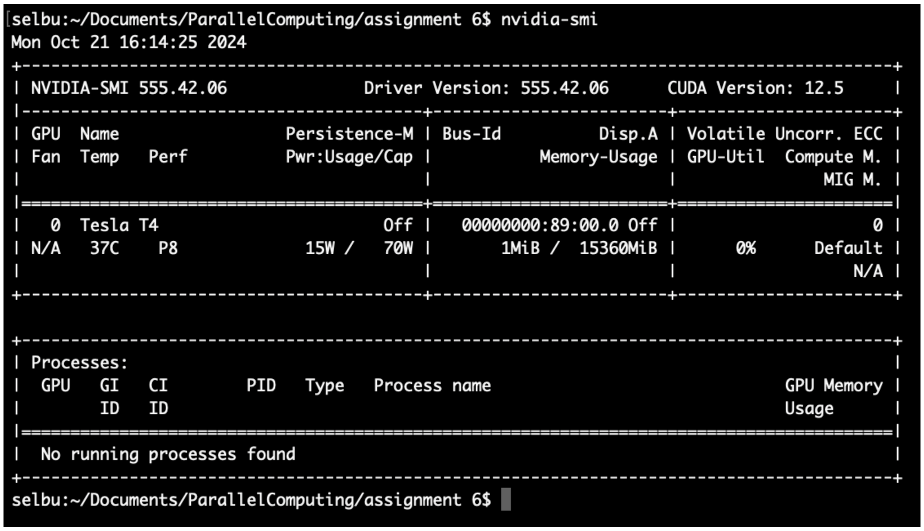
If, on the other hand, we consider as **GPU time** both the calculation time and the time for **copying data into memory**, then we have that:

$$\text{Speed-up} = \frac{T_{\text{CPU}}}{T_{\text{GPU-calc}} + T_{\text{GPU-mem}}} = \frac{2732.701ms}{72.731ms + 14.103ms} \approx 31.47$$

The significant **speed-up** achieved with **CUDA parallelisation** is due to the **GPU's ability to execute a large number of operations in parallel**, computationally intensive efficiency, **CUDA kernel optimisation** and **efficient memory management**. These factors combine to drastically reduce execution time compared to a **sequential version** running on a **CPU**.

2. The **GPU** used to test the code is an **Nvidia Tesla T4**, as shown by the information obtained with the `nvidia-smi` command:

3. **SIMD** (*Single Instruction Multiple Data*) is a **parallelism model** in which a single instruction is executed simultaneously on **multiple data**. This model is commonly used in **vector processing units** and SIMD extensions of modern CPUs. **SPMD** (*Single Program Multiple Data*) is a **parallelism model** in which **multiple processors** execute the same program, but on **different data**. Each processor can execute different instructions depending on the data it is processing. **CUDA** uses the **SPMD model**: a **CUDA kernel** is a function that is executed in parallel



```
[selbu:~/Documents/ParallelComputing/assignment 6$ nvidia-smi
Mon Oct 21 16:14:25 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 555.42.06              Driver Version: 555.42.06      CUDA Version: 12.5     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:89:00.0 Off |                    0 |
| N/A   37C    P8             15W /   70W |       1MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
selbu:~/Documents/ParallelComputing/assignment 6$ 
```

Figure 2: *Output of the nvidia-smi command.*

by many threads, each thread executing the same kernel, but on **different data**. The threads are organised in **blocks**, and the blocks are organised in a **grid**. Each thread has a unique identifier that can be used to access a different data element. Although all threads execute the same **kernel**, they may follow different execution paths depending on the data they process.