Lab (2)

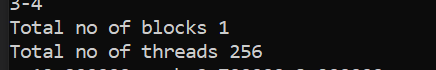
|  |  |  |
| --- | --- | --- |
| Name | Sec | BN |
| Basma Hatem Elhoseny | 1 | 16 |
| Sara Hossam El Zayat | 1 |  |

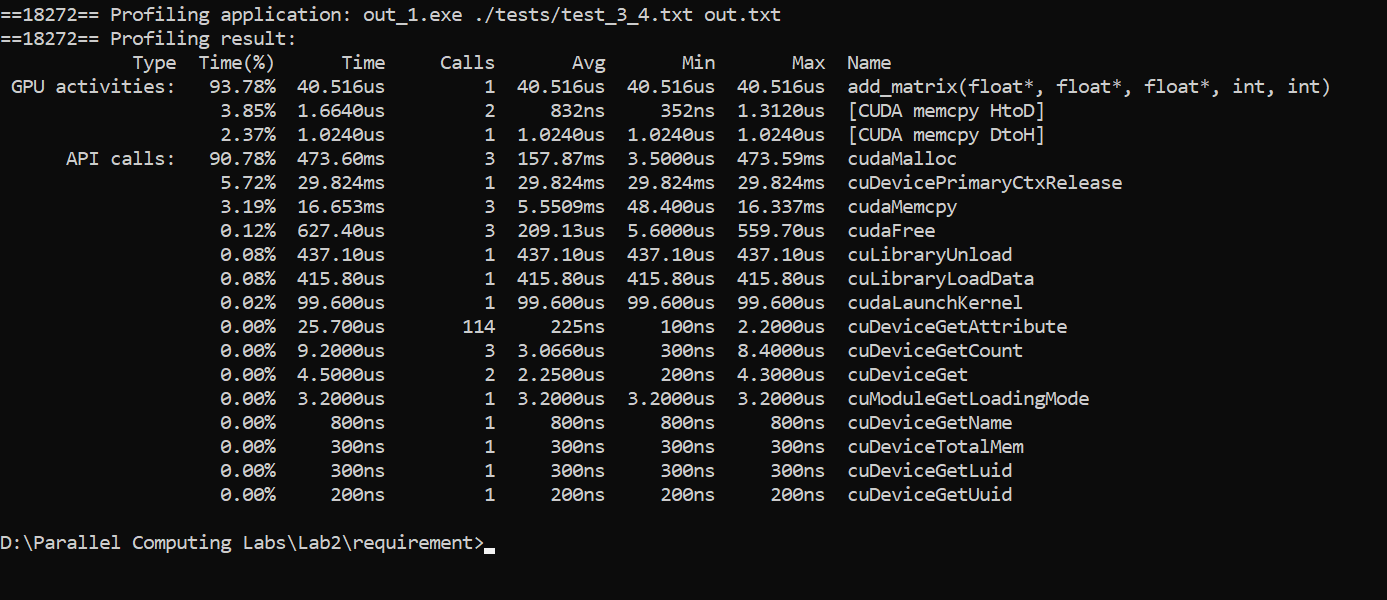
## Requirement(1) [Matrix Addition]:

### Kernel (1) Each Thread Produces one output matrix element:

#### Case(1) 3\*4 Matrix

With configuration that kernel size 16\*16 and one thread per element so:

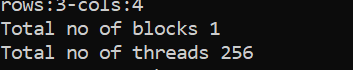


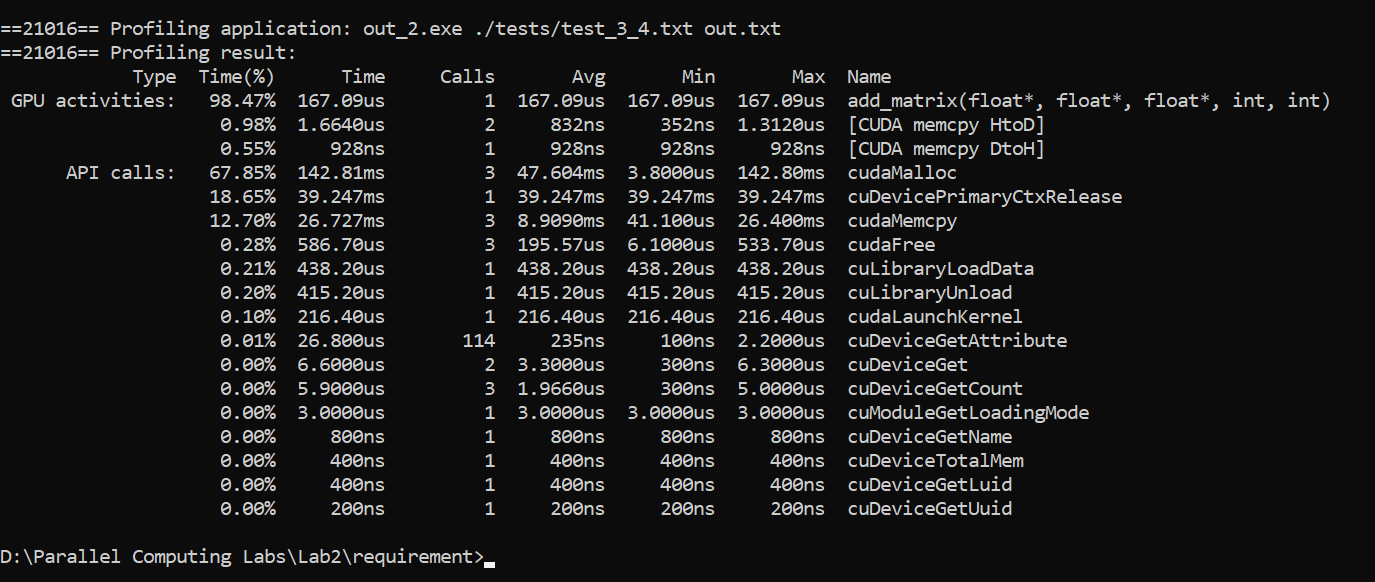


### Kernel(2) Each Thread Produces one output matrix row:

#### Case(1) 3\*4 Matrix

With configuration that kernel size 256 and one thread per rows so:



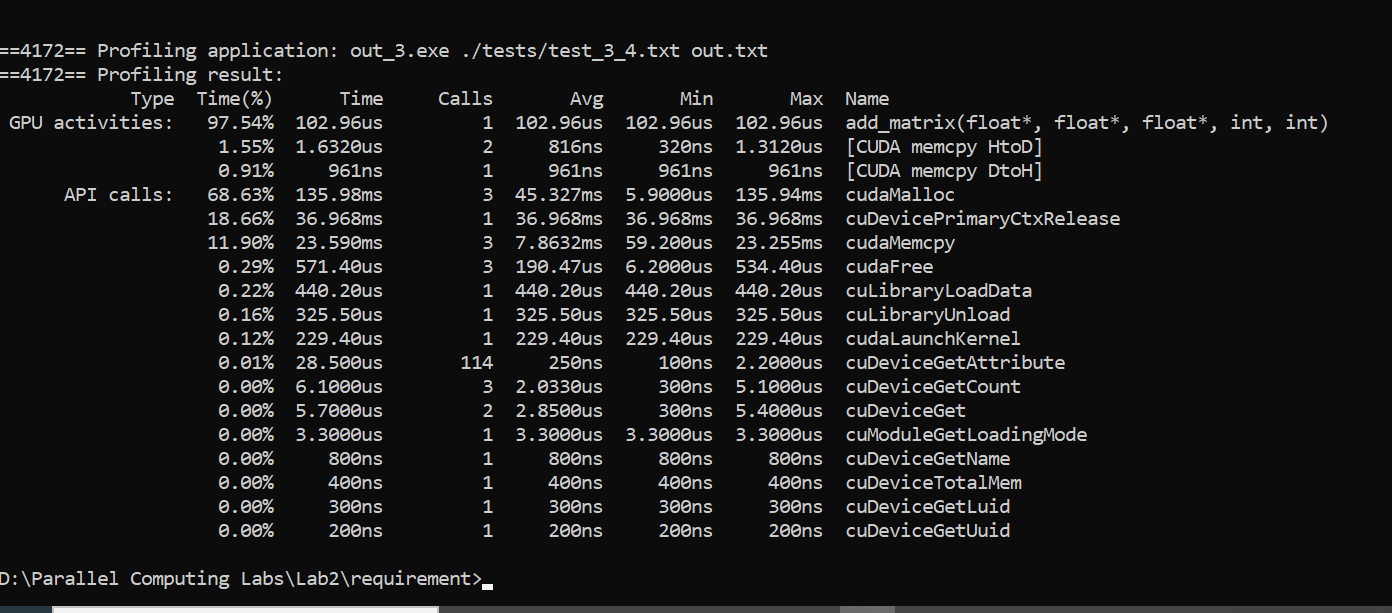


### Kernel(3) Each Thread Produces one output matrix column:

#### Case(1) 3\*4 Matrix

With configuration that kernel size 256 and one thread per rows so:





Comments:

* Case 3\*4 it is clear that the kernel 1 is the fastest regarding computing the addition function and nearly the 3 kernels have near copying time form host to device and vice versa 😊
* We think whatever the matrix size, kernel (1) will be the fastest Are we right ?! 🤔 Let’s see <3

|  |  |  |  |
| --- | --- | --- | --- |
| Matrix Shape | Kernel(1) [element] | Kernel(2) [row] | Kernel(3) [col] |
| 3\*4 | 40.516us | 167.09us | 102.96us |
| 2\*2 | 39.140us | 70.120us | 69.959us |
| 2\*4 | 39.460us | 133.65us | 509.88us |
| 100\*100 | 133.77ms | 16.774ms | 50.205ms |
| 50\*200 | 121.61ms | 24.791ms | 39.865ms |
| 200\*50 | 105.29ms | 19.865ms | 63.888ms |
| 10000\*1 | 32.520ms | 141.53ms | 2.26523s |
| 1\*10000 | 40.811ms | 310.43ms | 158.18ms |
|  |  |  |  |

**Trying Different Configuration for the Block Size:**

Case (1) 16\*16 🡺 Show above

Case (2) 4\*4

Case (3) 32\*48

## Requirement(2) [Matrix Multiplication]: