Parallel Thinking & CUDA Programming

Task1

(Look at .ipynb file for more details)

Vector addition

```
__global__ void vectorAdd(const float* A, const float* B, float* C, int
N) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < N) {
       C[i] = A[i] + B[i];
   }
}
```

Elementwise multiply-and-scale

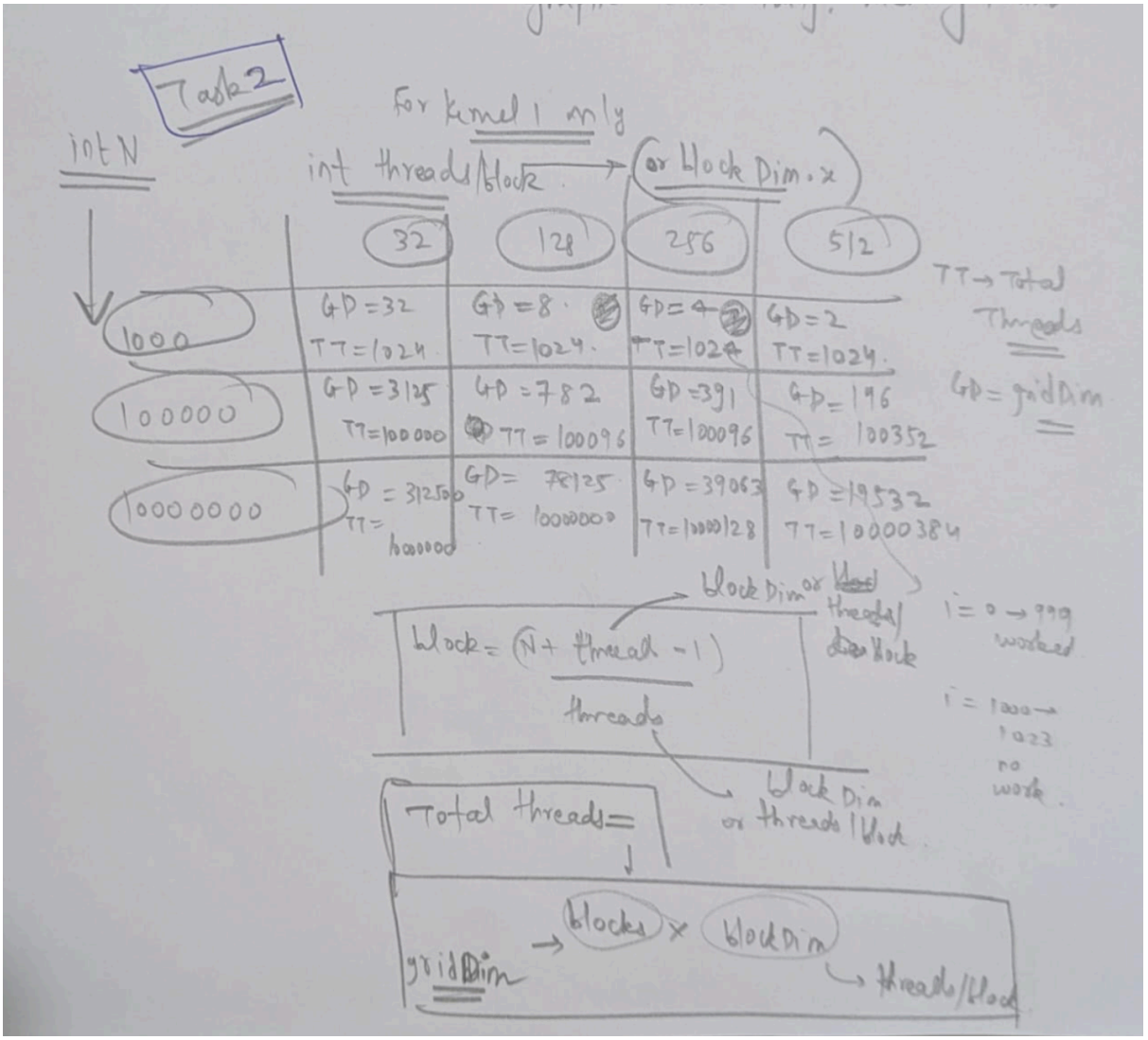
```
__global__ void multiplyScale(const float* A, const float* B, float* C,
float alpha, int N) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < N) {
       C[i] = alpha * A[i] * B[i];
   }
}
```

ReLU activation

```
__global__ void reluKernel(const float* A, float* C, int N) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < N) {
       C[i] = (A[i] > 0.0f) ? A[i] : 0.0f;
   }
}
```

Task2

OBSERVATION TABLE:



During this task, the vector addition kernel was executed using different combinations of input sizes and block dimensions in order to understand how CUDA schedules threads and how correctness is maintained across configurations.

Effect of Launching More Threads Than Required

In several configurations, the total number of threads launched was greater than the number of elements in the input array. In such cases, only the threads whose computed global index corresponded to valid array indices performed computation. Threads whose indices exceeded the input size simply exited without executing any memory operation. This behavior is expected in CUDA and does not affect correctness when handled properly.

Role of Bounds Checking in CUDA Kernels

Since CUDA kernels are launched using fixed-size blocks, it is common for some threads to map to indices outside the valid data range. To ensure safe execution, a conditional check was included in the kernel to restrict computation to valid indices only. This prevents unintended memory access, which could otherwise lead to incorrect results or silent memory corruption.

Correctness Across Configurations

All tested grid and block configurations produced correct results. Although different block sizes resulted in varying numbers of idle threads, correctness was preserved in every case due to proper index checking inside the kernel. No configuration produced unexpected behavior.

Block Size Selection: Qualitative Discussion

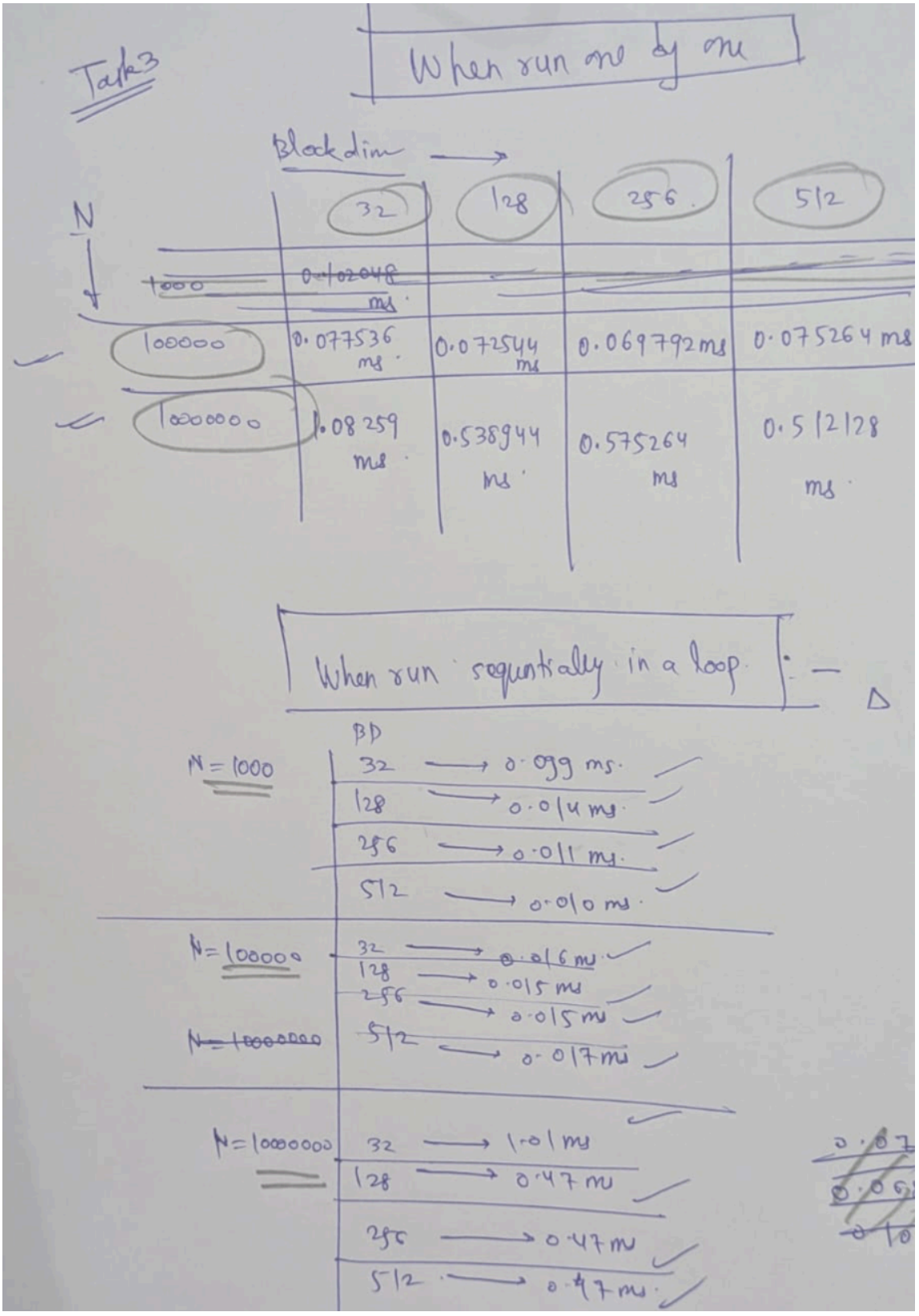
Choosing different block sizes primarily affected how threads were grouped and how many blocks were launched. Smaller block sizes tended to reduce the number of excess threads, while larger block sizes reduced the number of blocks required. In this task, block size did not influence correctness because the kernel performed independent elementwise operations and included appropriate bounds checking. Performance implications of block size selection were not analyzed as they were beyond the scope of this task.

Conclusion

This task demonstrates that for elementwise CUDA kernels, correctness depends on launching sufficient threads and enforcing proper bounds checking rather than on a specific choice of block size. Different grid and block configurations may alter thread utilization but do not affect correctness when implemented correctly.

Task3

OBSERVATION TABLE:



The difference between the upper table (manual runs) and the lower table (looped runs) is expected GPU behavior.

Upper table:
Manual- timings are larger & noisier
Lower table:
Ran the kernel inside a loop -timings are smaller & more consistent

From your looped runs: (for N=10000000)

block 32 ~1.01 ms
block 128 ~0.47 ms
block 256 ~0.47 ms
block 512 ~0.47 ms

This shows:

- 32 threads → underutilization
- ≥128 threads → GPU saturated
- plateau after occupancy is sufficient

For small N (1000, 100k)

- Kernel is too small
- Launch overhead dominates
- Larger blocks reduce scheduling cost

So :

512 < 256 < 128 < 32 can be observed

Conclusion:
Kernel execution times differed when measured via single runs versus sequential execution in a loop. The initial kernel launch incurs additional overhead due to CUDA context initialization and GPU warm-up. When kernels were executed sequentially, this overhead was amortized, resulting in more stable and lower execution times. All reported performance observations are based on the steady-state measurements obtained from looped execution.

Imp Note:
For larger input sizes, execution time stabilized for block sizes of 128 and above. **Minor variations** between block sizes were observed, which can be attributed to runtime scheduling and measurement noise rather than fundamental differences in kernel behavior.

References:

https://docs.nvidia.com/cuda/cuda-programming-guide/
https://github.com/NVIDIA/cuda-samples

My notes:

https://docs.google.com/document/d/1C23_Z_uXYJhSXqJfS9Wmlq7vpTruUN85SmBw9dwxkGE/edit?usp=sharing