Question 1 Not yet answered Not graded

List the names of all the members of your group. Note that only one of the group members needs to submit this report.

Maurice Salameh
Iyad Al Arab
Osman Fayad

Question 2 Not yet answered Marked out of 2.00

Describe the second optimization that you applied to your parallel implementation and what source of inefficiency this optimization alleviates. If there were any changes that you made to this second optimization compared to what you stated that you plan to do in the previous milestone, state what those changes were and why you made them.

For the second optimization, we applied register tiling that aims to solve the following inefficiencies:

- "top", "topleft", and "max" are being computed and stored in memory in one iteration and are being reused in the next iteration.

Question 3 Not yet answered Marked out of 1.00

Implement your second optimization in `kernel2.cu` and push your implementation to github. Include the link to the repository below.

https://github.com/momox369/GPU-Optimized-Needleman-Wunsch

Question 4 Not yet answered Marked out of 4.00

Describe the code for your parallel implementation with your second optimization applied. Make sure to replicate the code you are describing below with proper formatting and line numbers. Refer to these line numbers in your description while you *conceptually* describe what the code on each line or set of lines is doing.

```
1.    __shared__ int previousDiagonal[SEQUENCE_LENGTH];
2.    __shared__ int sm_sequence2[SEQUENCE_LENGTH];
3     int threadIteration = 1;
4     int count   = 2 * SEQUENCE_LENGTH - 1;
5     int col     = threadIdx.x + 1;
6     int top     = (col) * DELETION;
7     int topLeft = threadIdx.x * DELETION;
```

We defined outside the main loop two instead of 3 arrays (lines 1-2) in shared memory and initialized "top" and "topLeft" in registers (lines 6-7). sm_sequence2 reduces access to main memory.

```
1.  sm_sequence2[threadIdx.x] = sequence2[blockIdx.x*SEQUENCE_LENGTH + threadIdx.x];
2.  int seq1Value = sequence1[blockIdx.x*SEQUENCE_LENGTH + threadIdx.x];
```

We initialized starting values to compute (lines 1-2).

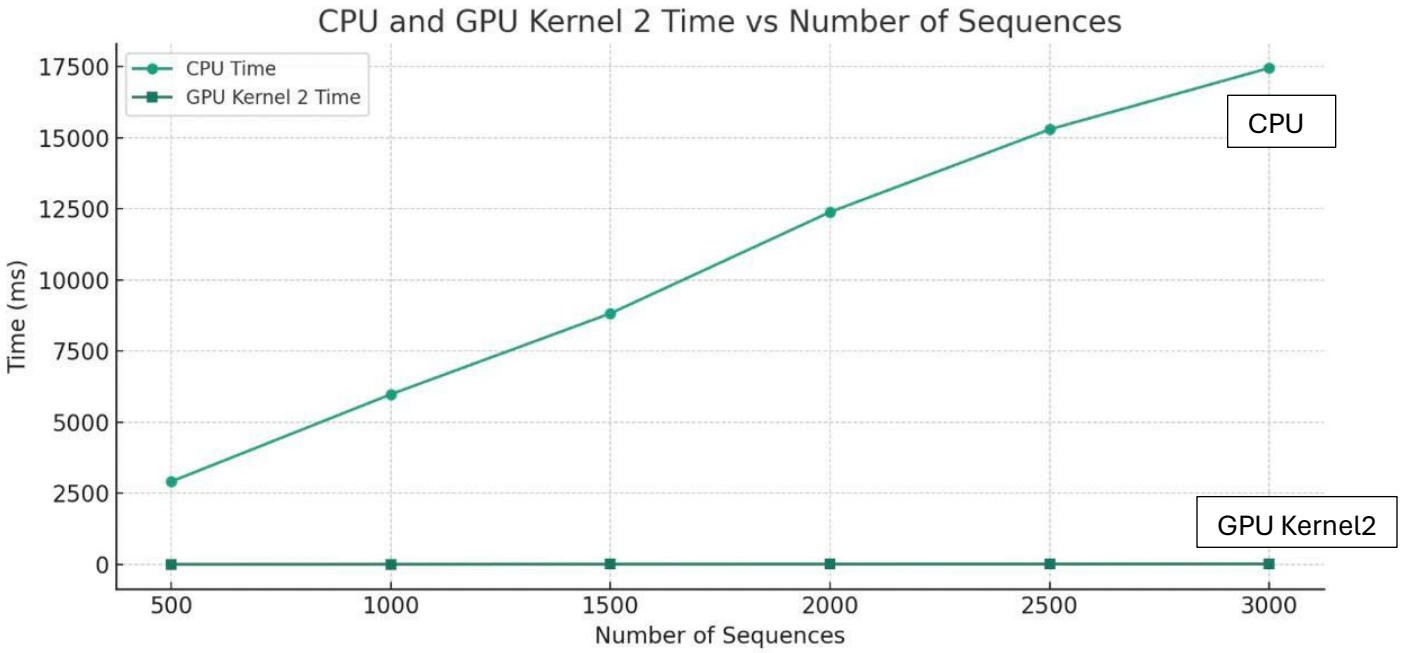
```
1.     int left    = (col == 1) ? (row)  * INSERTION : (previousDiagonal[col-2]);
2.     int insertion = top + INSERTION;
3.    int deletion  = left + DELETION;
4.   int match     = topLeft + (
5.            (sm_sequence2[(row-1)] == seq1Value)
6.            ? MATCH
7.            : MISMATCH
8.            ); //check if there is a match
9.        int max = (insertion > deletion) ? insertion : deletion;
10.       max = (match > max) ? match : max;
11.       top = max;
12.       topLeft = left;
```

"left" is computed by fetching values from memory (line 1). Storing "max" and "left" in registers to be used in next iteration – register tiling (lines 11-12).

Question 5 Not yet answered Marked out of 2.00

Evaluate your second optimization in comparison with the sequential code, the basic parallel implementation, and the first optimization. You only need to report the kernel time for the parallel implementations (do not worry about copy time). You should run both implementations on multiple input sizes and include a plot that shows how the execution time for each implementation varies for different input sizes. Describe the trend that you observe in the plot. Make sure to also specify the details for the system you are running on, including the type of CPU and GPU you are using and the amount of memory available in the system.

Number of sequences	CPU time (ms)	GPU Kernel 2 Time (ms)
500	2913	3.087
1000	5980	5.39
1500	8826	8.023
2000	12390	10.406
2500	15298	12.787
3000	17459	15.17



As the number of sequences increases, the execution time on the CPU increases significantly. In contrast, the execution time on the GPU remains relatively constant or increases at a much slower rate (from 3 ms to 15) compared to the CPU. As the number of sequences increases, the acceleration factor tends to increase, indicating that the GPU becomes increasingly more advantageous over the CPU for larger workloads.

Question 6 Not yet answered Marked out of 1.00

Describe the third optimization that you plan to apply to your code. Explain what bottleneck or other source of inefficiency this optimization targets and why you think it will be effective. You do not need to describe multiple optimizations. You only need to describe one optimization that you will implement for the next milestone. Use illustrations where needed. Make sure to cite any references that you use.

We plan to apply double buffering in the third optimization because there is a false dependency between two buffers prevDiagonal and previousPreviousDiagonal.