Note: This lab was conducted using the L4 instance on Colab Pro which contains a 6 physical core Intel Xeon processor with 2-way SMT.

Question 1

CPU:
I achieved a maximum speedup of 1.52x (from 101.68 ms to 66.87 ms). The vectors being processed together reside in the same row (contiguous) so that all the vectors can use the same cache line.

To manage control flow I had a mask vector for each of the vectors being processed at once. The inner loop exits only when all of the mask vectors are false (the while loop condition is no longer met).

I chose to process 4 vectors once as this resulted in the maximum amount of speedup (on another CPU I found that 2 vectors resulted in the maximum speedup).

The limiting factor into how far you can scale ILP is the issue width of the core.

GPU:
I achieved a speedup of 1.17x (from 190.68 ms to 162.72 ms). Like the CPU implementation, the vectors being processed together reside in the same row.

As for control flow, I have a per thread mask vector equal to the number of vectors processed at once (COARSENING_FACTOR). Once the mask is all false, it breaks out of the inner loop.

Processing 4 vectors at once seemed to give the maximum speedup (It may be different on other GPUs).

Beyond 4 independent multiply or add instruction in flight simultaneously, there is no benefit to additional ILP.

Question 2
(My CPU has 6 physical cores)

Parallelizing over 6 physical cores resulted in a 4.5x speedup (from 102.07 ms to 22.56 ms).

I partitioned over rows which seems to be more cache friendly than partitioning over columns

Question 3
I observed a speedup of 124x (from 190.67 ms to 1.54 ms). The absolute run-time of the GPU version is significantly less than the CPU version (66.87 ms vs 1.54 ms).

As for my work partitioning strategy, I split the blocks row-wise and the 4 warps for each block get split column-wise contiguously as they share the same l1 cache.

Question 4
If we double the number of blocks, but half the warps, the run time is almost the same (1.39 ms vs 1.54 ms).

Even if we halve the number of blocks and double the number of warps per block, the run-time is comparable (1.94 ms). Even though we have half the number of SMs active, each warp scheduler will have 2 warps. Since there is so much opportunity to hide latency, the warp schedulers become almost twice as productive, hence why the run-time is comparable.

Question 5
I'm able to speedup my implementation by 1.95x (from 22.56 ms to 11.56 ms) by utilizing 6 threads per core.

Here are some factors that may determine the optimal number of threads:

- Existing level of ILP in code (if the code already has a lot of ILP, threads may be less effective)
- Amount of stalls resulting from the code (threads could then minimize waste using temporal multi-threading)

Question 6

# of Warps	Time (ms)
1	186.42
4	48.21
8	26.38
16	18.87
32	16.31

Even beyond 4 warps per SM, run-time still decreases quite significantly until the limit. Compared to running 4 warps, running 32 warps yields a speedup of almost 3x.

A single warp experiences longer latencies than a CPU (because of less cache and complex control flow), so there is a significant opportunity to hide latencies which explains the significant speedup.

Question 7

# of Warps	Time (ms)
1	5.17
4	1.58
8	1.08
16	0.96
32	0.92

Using multi-threading, I'm able to achieve a maximum speedup of 1.71x. (1.58 ms vs 0.92 ms). The optimal number of warps is the maximum.

Unlike CPU threads, GPU threads have very minimal overhead. As a result, it seems that more threads will almost always be better.

Question 8

CPU:
The best possible speedup I obtain using ILP optimizations is 1.15x (9.89 ms vs 11.46 ms). This speedup is significantly less than the one obtained in the single-threaded single-core setting.

The optimal configuration is:
Thread per core: 6
Vector processed at once: 2

Having 6 threads per core was enough to cover waste from CPU stalls, but was not sufficient to maximize the issue-width of the processor. Hence why processors 2 vectors at once helped.

GPU:
The best possible speedup is 1.19x (0.92 ms vs 0.77 ms). This speedup is comparable to the single core setting.

The optimal configuration is:
32 warps per block
3 vectors processed at once

To be honest, I don’t completely understand why the ILP optimizations on the GPU achieve a decent speedup. It seems like the warp scheduler should context switch with zero overhead to a different warp if it's stalled, so adding ILP optimization shouldn’t help.