1.) Write up of code changes to nbodu_sp25.cu

1. Atomic Min Helper (atomicMinFloat)

    - What changed is I introduced the a __device__ function that wraps atomicCAS on the integer bit-pattern of a float.

    - Why because CUDA has no built-in atomic minimum for floats, so this helper ensures threads can safely update a single global minimum value.

2. Optimized Kernel (minimum_distance)

    - In the original each thread walked *all* other points in global memory, calling an atomic update for *every* distance computed.

    - Some new things I added are:

        - Shared-Memory Tiling: Blocks cooperatively load chunks of X/Y into __shared__ arrays, cutting global-memory traffic.

        - Per-Thread Local Minima: Each thread computes a local minimum over its tile.

        - Block-Level Reduction: Threads reduce their local minima into one block-minimum in shared memory.

        - Single Atomic per Block: Only thread 0 issues atomicMinFloat once per block, dramatically reducing atomic contention.

3. Host-Side Initialization & Launch

    - Device Min Init: After cudaMalloc(&dmin_dist…), copy in +INFINITY so the first comparison is valid.

    - Dynamic Launch Params: Query deviceProp.maxThreadsPerBlock, choose threads_per_block (capped at 256), compute blocks = ceil(n/threads_per_block), and set shared_bytes = 2 * threads_per_block * sizeof(float).

    - Kernel Invocation: Replace the placeholder launch with

        - minimum_distance<<<blocks, threads_per_block, shared_bytes>>>(
        - dVx, dVy, dmin_dist, num_points
        -);

4. **Robust Error Checking & Timing**

    - Wrapped all CUDA calls (cudaMalloc, cudaMemcpy, kernel launch) in check_error().

    - Preserved event-based timing for H_2D transfer, kernel execution, and D_2H transfer, along with a CPU reference timing for verification.

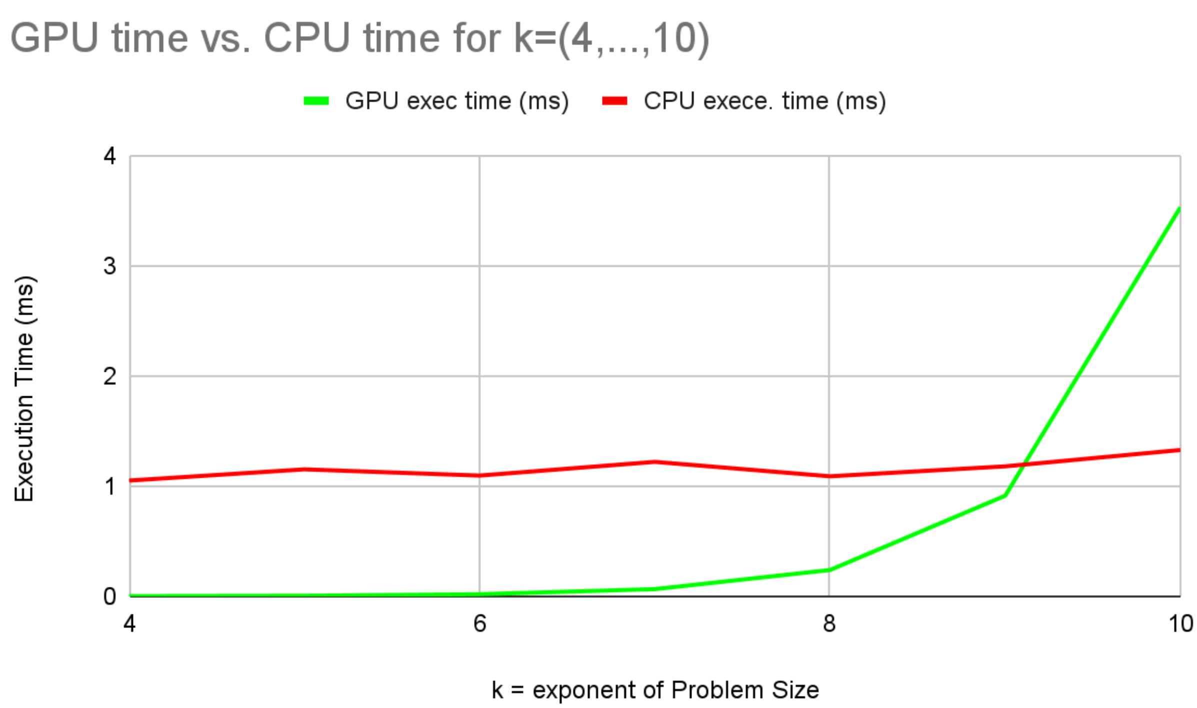
5. **Function Ordering & Visibility**

    - Moved atomicMinFloat above the kernel so the compiler sees it when compiling minimum_distance.

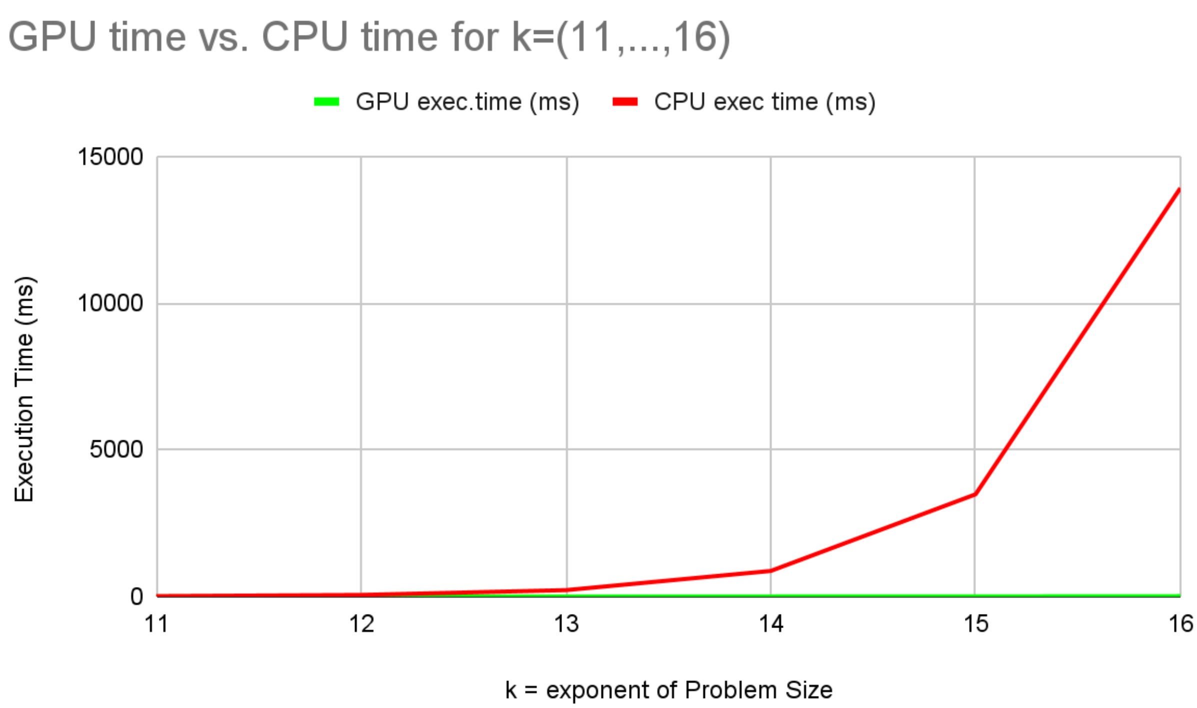
    - Left the original CPU reference function (minimum_distance_host) unchanged for correctness checks.

2.)

n	k	GPU time (ms)	CPU time (ms)
16	4	1.050816	0.002388
32	5	1.152448	0.006277
64	6	1.096	0.019552
128	7	1.22032	0.065952
256	8	1.08944	0.239295
512	9	1.179616	0.913358
1024	10	1.328096	3.529802



n	k	GPU time (ms)	CPU time (ms)
2048	11	1.2944	13.861878
4096	12	1.603136	55.113068
8192	13	2.251712	218.813065
16384	14	3.039264	872.138733
32768	15	5.30224	3482.346436
65536	16	12.033312	13915.45117



Question**:** For what value of n does the GPU code become faster that the CPU code?

Answer: From **n = 1024 (k=10) and onwards**, GPU code has faster execution time than CPU code.

3.)

k	n	Host to Device (ms)	Device to Host (ms)
4	16	0.024576	1.030144
5	32	0.023648	1.351552
6	64	0.02832	1.127328
7	128	0.027168	1.06496
8	256	0.025248	1.201312
9	512	0.026752	1.394688
10	1024	0.027808	1.20016
11	2048	0.03024	1.2288
12	4096	0.03888	1.459552
13	8192	0.049152	2.132192
14	16384	0.078336	3.02752
15	32768	0.098784	5.098912
16	65536	0.174912	11.559104