Report on A2-CSC591: SIMD Skyline dominance tests-Harper Yan

**Project Summary:**

The Skyline Computation Project aims to compare the performance of two implementations—scalar and SIMD (Single Instruction, Multiple Data)—of a Skyline computation algorithm. The purpose of the project is to analyze and understand the efficiency gains achieved through SIMD parallelization compared to a traditional scalar approach.

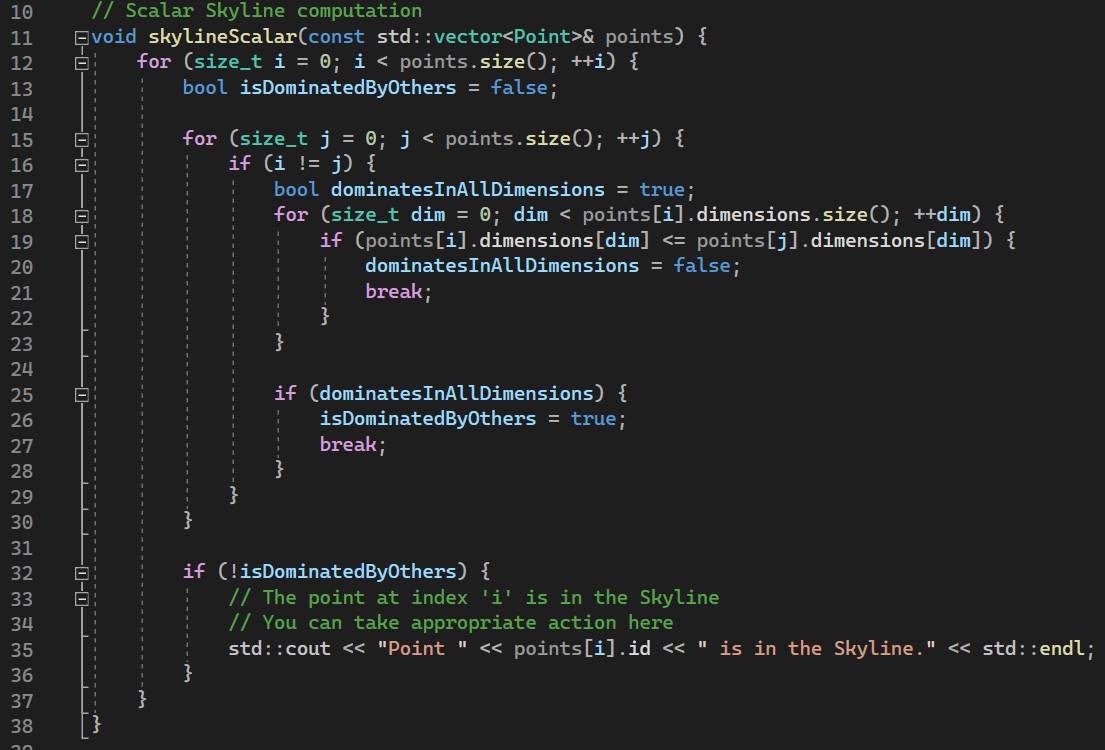
**Methods Used:**

- Scalar Implementation: The scalar implementation utilizes a straightforward algorithm where each data point is compared against all others to determine whether it is dominated by any other point in all dimensions.

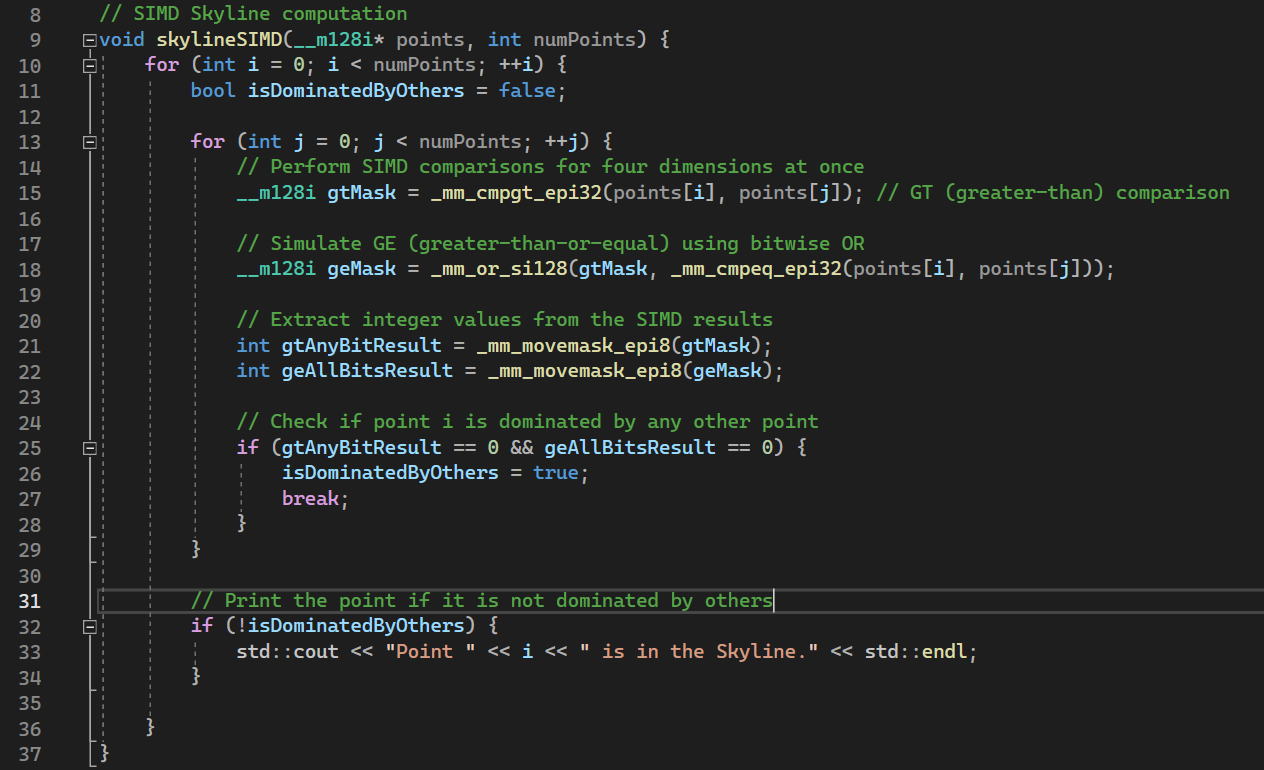
- SIMD Implementation: The SIMD implementation leverages Intel's SSE2 intrinsics to parallelize the dominance checks across multiple dimensions, aiming to exploit parallelism in modern processors.

**Code Explanation:**

The crucial part of this project is the function performing dominance tests between two points. The scalar version of this function is very straightforward and we present it without explanation.



We also provide the SIMD version of this function:



Explanation:

1. The function takes an array of `\_\_m128i` points (assumed to represent 4D integer points) and the number of points (`numPoints`) as input parameters. The input array is 100 four-dimensional random integers generated by C++.

2. It iterates through each point (`i`) in the array. For each point `i`, it enters a nested loop that iterates through all other points (`j`) in the array.

3. The GT (greater-than) comparison creates a mask (`gtMask`) where each bit indicates whether the corresponding dimension of `i` is greater than the corresponding dimension of `j`.

6. The GE (greater-than-or-equal) is simulated using a bitwise OR operation (`geMask`) on the GT mask and an equality comparison between dimensions.

7. Integer values are extracted from the SIMD results using `\_mm\_movemask\_epi8`. This is used to examine if any/all bits are set in the two masks. If any bit is set for GT, then the integer value of GT is not 0; If all bits are set for GE, the integer value should be 65536.

8. It checks if any bit is set in the GT mask (`gtAnyBitResult`) and all bits are set in the GE mask (`geAllBitsResult`). If both conditions are false, it means point `i` is not dominated by any other point in all dimensions.

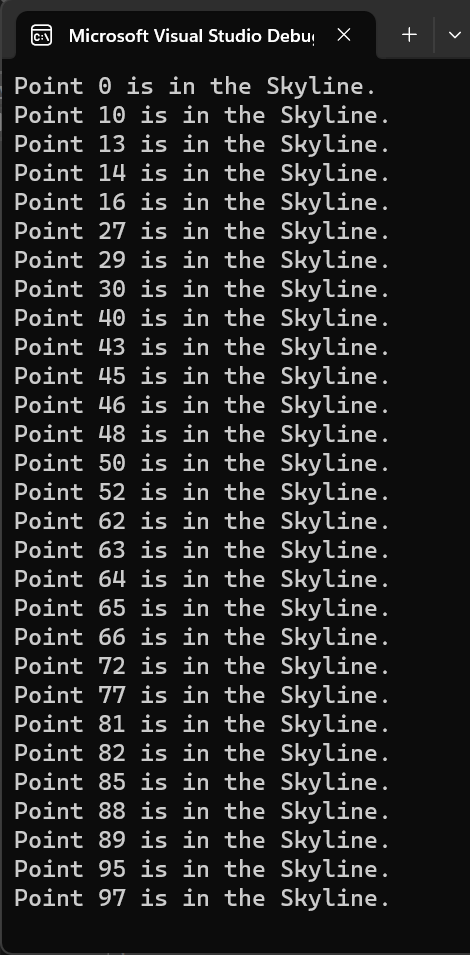
This corresponds to the Right-dominance case in the V-skyline paper. In this project, we don’t seek to utilize the transitivity among points, we simply compare every possible pair of points. Therefore, we only need to rule out the case of Right dominance. So, the check condition is set to be like line 25.

9. If point `i` is not dominated by others, it is considered part of the Skyline, and a message is printed.

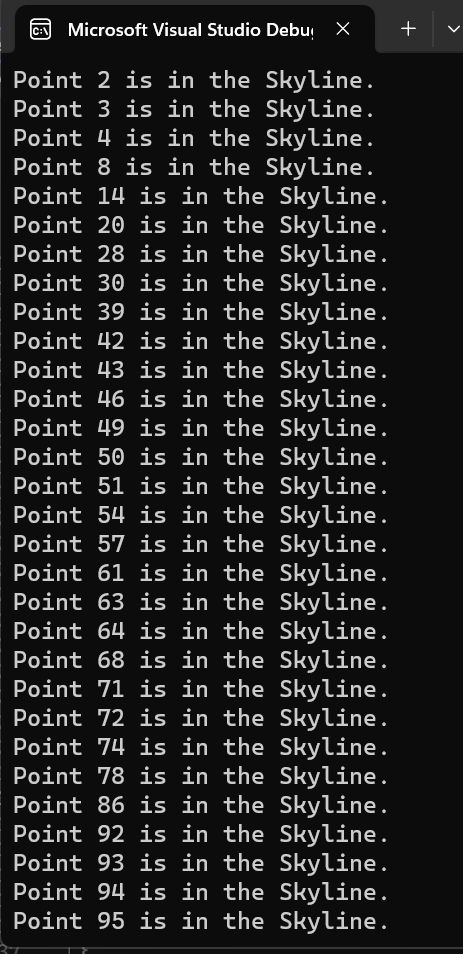
Overall, the function performs SIMD-based dominance checks for each point in the array, identifying points that are not dominated by any other point and printing them as part of the Skyline.

**Experiment:**

We run both the code to demonstrate that they function as expected. This is the program output for skyline\_scalar.

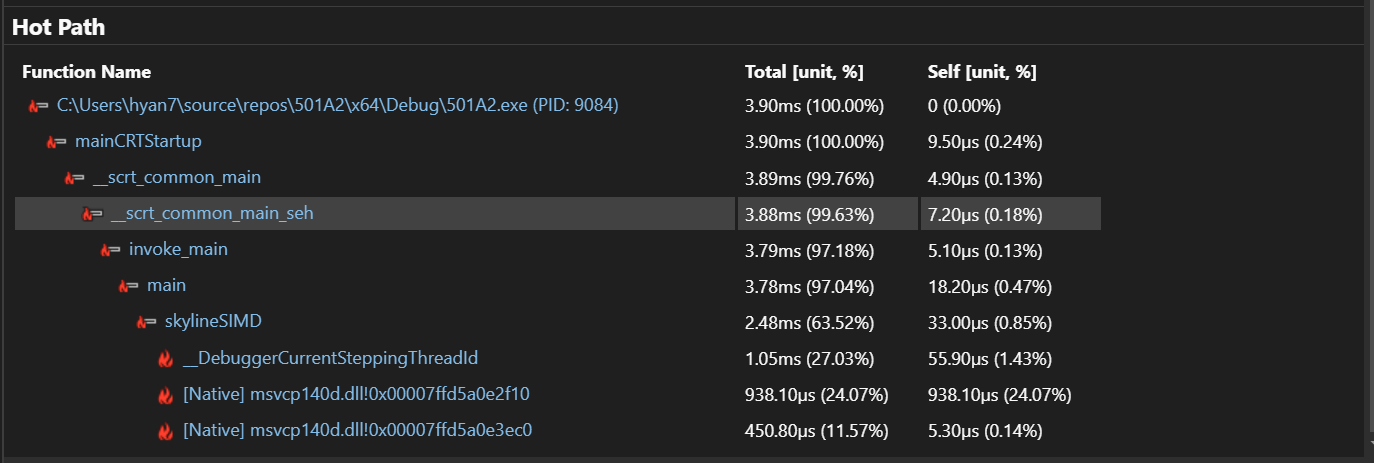


This is the program output for Skyline\_SIMD.



**Performance analysis:**

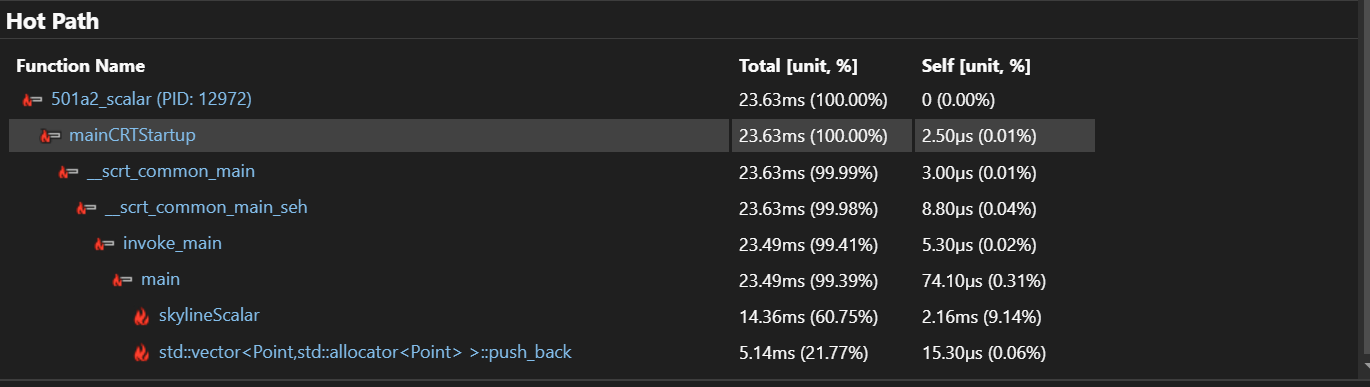
Firstly, we show that the SIMD instructions are retired, by demonstrating the report given by profiler. We can see that, the function skylineSIMD functions properly.



Next, we show that using skylineSIMD provides 2x better performance than the scalar solution.

Since both skyline\_scalar and skylineSIMD selects the non-dominated points for a data set of size 100, containing all 4-dimensional random integers, these two programs performs equal amounts of dominance tests.

The performance report of the scalar solution is given:



We can see that, the skylineSIMD is more than two times faster than the scalar version of the solution.