<u>comparative execution time as well as analysis of the performance of different kernels (how many times faster, why is it faster, overheads in calling the kernel, etc.)</u>

The programming language and number of threads that we use affects the run time of the kernel in our stencil vector program.

	2^20	2^24	2^28
C	125000 microseconds	2002000 microseconds	31359000 microseconds
CUDA	85.800 microseconds	765.78 microseconds	13448 microseconds
x86-64	83000 microseconds	1327000 microseconds	20797000 microseconds
SIMD	14000 microseconds	268000 microseconds	3912000 microseconds

As we can see on the table above, CUDA performed the fastest but this is due to having more workers or threads to run the kernel at the same time. In our CUDA program we runned with 1024 threads and blocks for all input sizes which certainly gave it an advantage over C, x86-62 and SIMD. For the input size 2^20 CUDA kernel ran 1457 times faster than C, 967 times faster than x86-64, and 163 times faster than SIMD. For the input size 2^24 CUDA kernel ran 2614 times faster than C, 1732 times faster than x86-64, and 350 times faster than SIMD. For the input size 2^28 CUDA kernel ran 2331 times faster than C, 1546 times faster than x86-64, and 290 times faster than SIMD. However, CUDA still has some overheads due to the transfer of host to device and vice-versa.

The second fastest would be SIMD. This is due to the fact that the language is a low level programming language which is close to machine language as well as it has a certain degree of parallelism. In SIMD we can use shr <array size>,<value> which shifts the bits to the right. This allows the array to be divided into 2 raised to <value>. In our case the value we used is 3, thus our array was divided into 8 and processed at the same time. For input size 2^20 SIMD performed almost 6 times faster than x86-64 and almost 9 times faster than C. For input size 2^24 SIMD performed almost 5 times faster than x86-64, and 74 times faster than C.

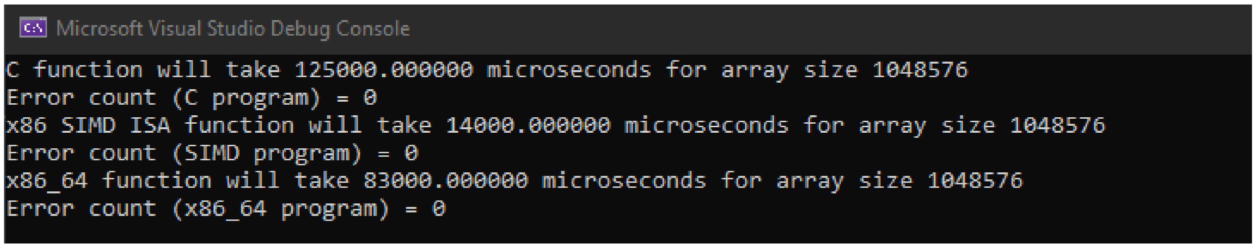
Following SIMD in terms of runtime would be x86-64. This is due to x86-62 being a low level programming language which makes it close to machine language which means there's no need for translation or interpretation. However in x86-64 we processed the program sequentially instead of parallel thus it is slower than SIMD. For input size 2^20, 2^24 and 2^28 x86-62 performed approximately 1.5 times faster than C.

Lastly C had the slowest run time of the kernel compared to all the programming languages discussed above. This is due to C being a high level programming language which means it still needs translation or interpretation for it to be understood by the machine which makes it slower. Additionally, we also performed the kernel sequentially wherein we did not have any threads to work on the kernel in parallel.

In conclusion, the programming language and parallelism affects the run time of programs. Parallelism improves the efficiency of a program by allowing threads or workers to work on the kernel. The more threads we have the faster the kernel would be processed. However we should not mindlessly increase thread counts as it eats more resources, possible race conditions, and we should be mindful of the block size to optimize the number of threads. Moreover low level programming languages run faster than high level programming languages due to the run time consumed for the translation of high level programming languages to commands our machine can understand.

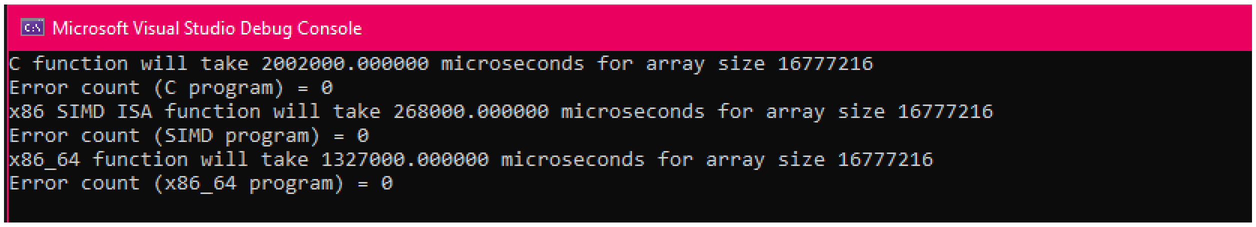
<u>Screenshot of the program output with correctness check C, x86-64, SIMD YMM register</u>

For input size 2^20:



```
C function will take 125000.000000 microseconds for array size 1048576
Error count (C program) = 0
x86 SIMD ISA function will take 14000.000000 microseconds for array size 1048576
Error count (SIMD program) = 0
x86_64 function will take 83000.000000 microseconds for array size 1048576
Error count (x86_64 program) = 0
```

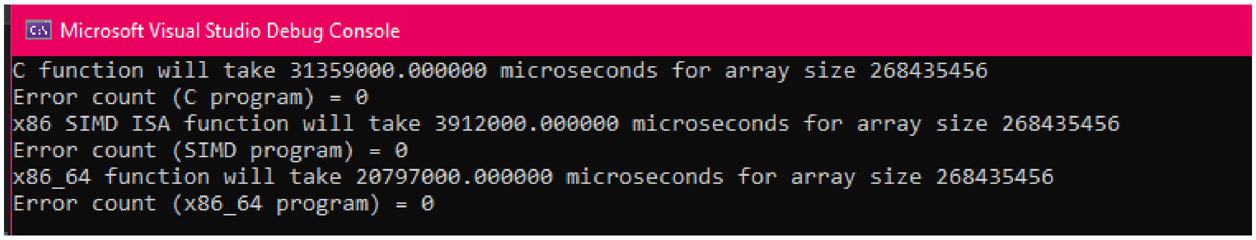
For input size 2^24:



Microsoft Visual Studio Debug Console

```
C function will take 2002000.000000 microseconds for array size 16777216
Error count (C program) = 0
x86 SIMD ISA function will take 268000.000000 microseconds for array size 16777216
Error count (SIMD program) = 0
x86_64 function will take 1327000.000000 microseconds for array size 16777216
Error count (x86_64 program) = 0
```

For input size 2^28:

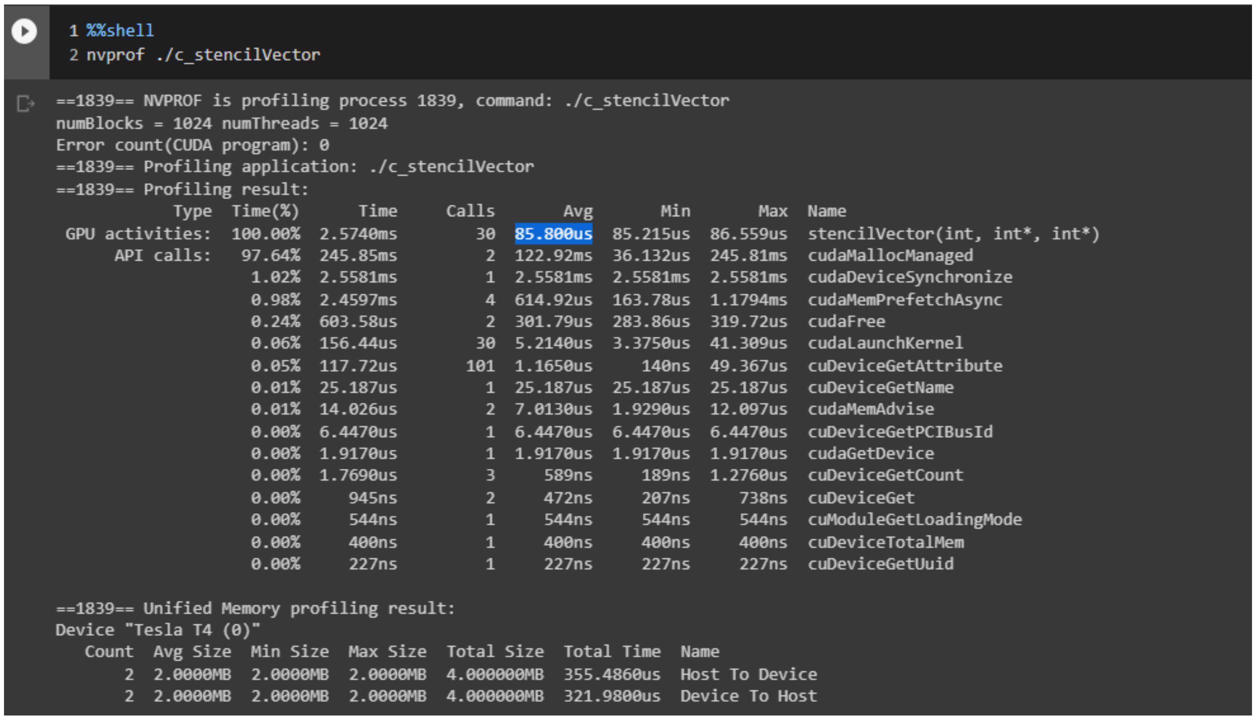


Microsoft Visual Studio Debug Console

```
C function will take 31359000.000000 microseconds for array size 268435456
Error count (C program) = 0
x86 SIMD ISA function will take 3912000.000000 microseconds for array size 268435456
Error count (SIMD program) = 0
x86_64 function will take 20797000.000000 microseconds for array size 268435456
Error count (x86_64 program) = 0
```

Screenshot of the program output including correctness check (CUDA, optional)

For input size 2^20:

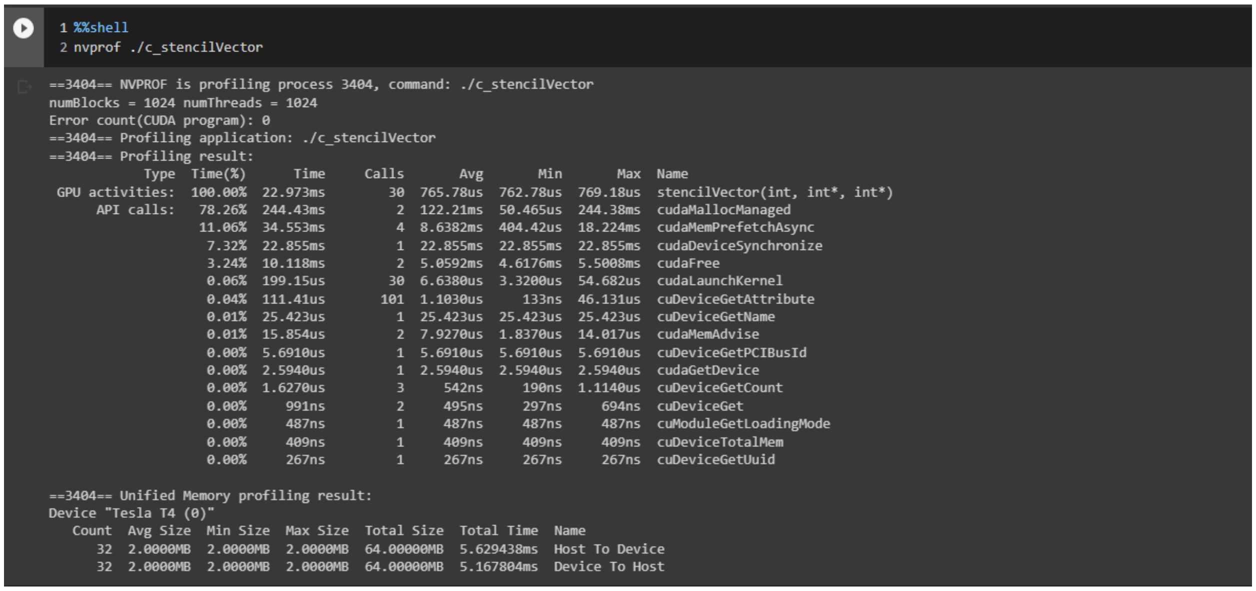


```
%%shell
nvprof ./c_stencilVector
```

```
==1839== NVPROF is profiling process 1839, command: ./c_stencilVector
numBlocks = 1024 numThreads = 1024
Error count(CUDA program): 0
==1839== Profiling application: ./c_stencilVector
==1839== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  2.5740ms        30  85.800us  85.215us  86.559us  stencilVector(int, int*, int*)
      API calls:   97.64%  245.85ms         2  122.92ms  36.132us  245.81ms  cudaMallocManaged
                    1.02%  2.5581ms         1  2.5581ms  2.5581ms  2.5581ms  cudaDeviceSynchronize
                    0.98%  2.4597ms         4  614.92us  163.78us  1.1794ms  cudaMemPrefetchAsync
                    0.24%  603.58us         2  301.79us  283.86us  319.72us  cudaFree
                    0.06%  156.44us        30  5.2140us  3.3750us  41.309us  cudaLaunchKernel
                    0.05%  117.72us       101  1.1650us     140ns  49.367us  cuDeviceGetAttribute
                    0.01%  25.187us         1  25.187us  25.187us  25.187us  cuDeviceGetName
                    0.01%  14.026us         2  7.0130us  1.9290us  12.097us  cudaMemAdvise
                    0.00%  6.4470us         1  6.4470us  6.4470us  6.4470us  cuDeviceGetPCIBusId
                    0.00%  1.9170us         1  1.9170us  1.9170us  1.9170us  cudaGetDevice
                    0.00%  1.7690us         3     589ns     189ns  1.2760us  cuDeviceGetCount
                    0.00%     945ns         2     472ns     207ns     738ns  cuDeviceGet
                    0.00%     544ns         1     544ns     544ns     544ns  cuModuleGetLoadingMode
                    0.00%     400ns         1     400ns     400ns     400ns  cuDeviceTotalMem
                    0.00%     227ns         1     227ns     227ns     227ns  cuDeviceGetUuid

==1839== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       2  2.0000MB  2.0000MB  2.0000MB  4.000000MB  355.4860us  Host To Device
       2  2.0000MB  2.0000MB  2.0000MB  4.000000MB  321.9800us  Device To Host
```

For input size 2^24:

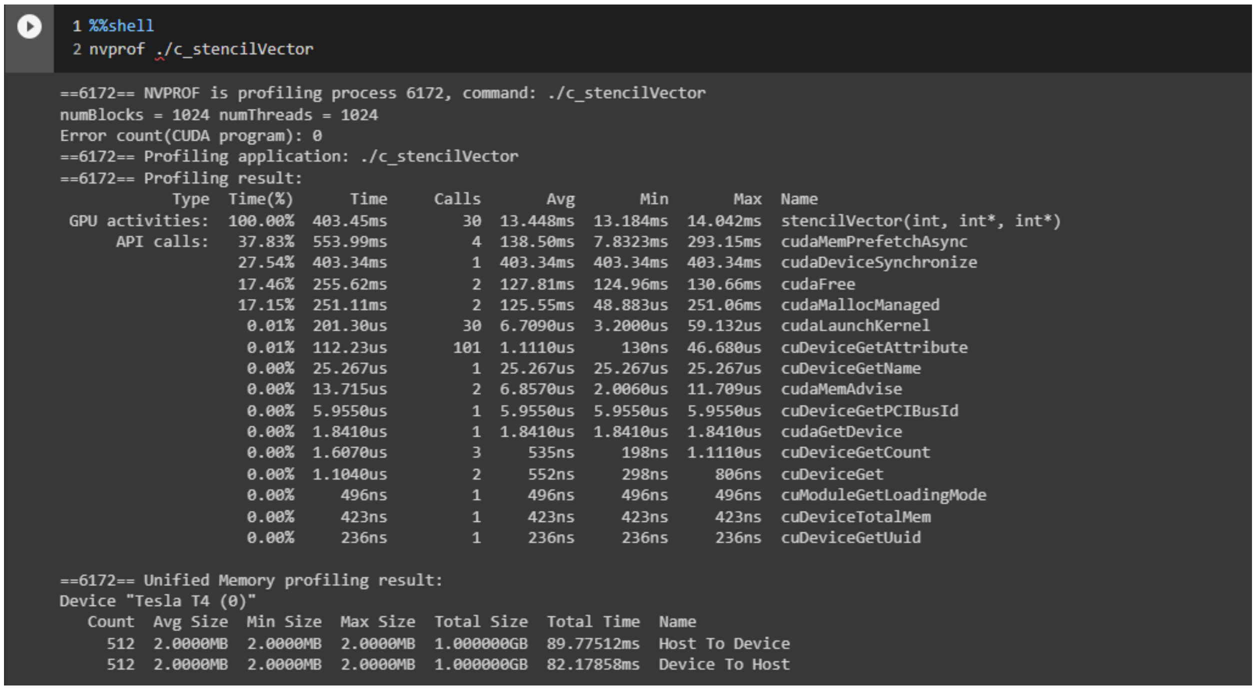


```
%%shell
nvprof ./c_stencilVector
```

```
==3404== NVPROF is profiling process 3404, command: ./c_stencilVector
numBlocks = 1024 numThreads = 1024
Error count(CUDA program): 0
==3404== Profiling application: ./c_stencilVector
==3404== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  22.973ms        30  765.78us  762.78us  769.18us  stencilVector(int, int*, int*)
      API calls:   78.26%  244.43ms         2  122.21ms  50.465us  244.38ms  cudaMallocManaged
                   11.06%  34.553ms         4  8.6382ms  404.42us  18.224ms  cudaMemPrefetchAsync
                    7.32%  22.855ms         1  22.855ms  22.855ms  22.855ms  cudaDeviceSynchronize
                    3.24%  10.118ms         2  5.0592ms  4.6176ms  5.5008ms  cudaFree
                    0.06%  199.15us        30  6.6380us  3.3200us  54.682us  cudaLaunchKernel
                    0.04%  111.41us       101  1.1030us     133ns  46.131us  cuDeviceGetAttribute
                    0.01%  25.423us         1  25.423us  25.423us  25.423us  cuDeviceGetName
                    0.01%  15.854us         2  7.9270us  1.8370us  14.017us  cudaMemAdvise
                    0.00%  5.6910us         1  5.6910us  5.6910us  5.6910us  cuDeviceGetPCIBusId
                    0.00%  2.5940us         1  2.5940us  2.5940us  2.5940us  cudaGetDevice
                    0.00%  1.6270us         3     542ns     190ns  1.1140us  cuDeviceGetCount
                    0.00%     991ns         2     495ns     297ns     694ns  cuDeviceGet
                    0.00%     487ns         1     487ns     487ns     487ns  cuModuleGetLoadingMode
                    0.00%     409ns         1     409ns     409ns     409ns  cuDeviceTotalMem
                    0.00%     267ns         1     267ns     267ns     267ns  cuDeviceGetUuid

==3404== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      32  2.0000MB  2.0000MB  2.0000MB  64.00000MB  5.629438ms  Host To Device
      32  2.0000MB  2.0000MB  2.0000MB  64.00000MB  5.167804ms  Device To Host
```

For input size 2^28:



```
%%shell
nvprof ./c_stencilVector
```

```
==6172== NVPROF is profiling process 6172, command: ./c_stencilVector
numBlocks = 1024 numThreads = 1024
Error count(CUDA program): 0
==6172== Profiling application: ./c_stencilVector
==6172== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  403.45ms        30  13.448ms  13.184ms  14.042ms  stencilVector(int, int*, int*)
      API calls:   37.83%  553.99ms         4  138.50ms  7.8323ms  293.15ms  cudaMemPrefetchAsync
                   27.54%  403.34ms         1  403.34ms  403.34ms  403.34ms  cudaDeviceSynchronize
                   17.46%  255.62ms         2  127.81ms  124.96ms  130.66ms  cudaFree
                   17.15%  251.11ms         2  125.55ms  48.883us  251.06ms  cudaMallocManaged
                    0.01%  201.30us        30  6.7090us  3.2000us  59.132us  cudaLaunchKernel
                    0.01%  112.23us       101  1.1110us     130ns  46.680us  cuDeviceGetAttribute
                    0.00%  25.267us         1  25.267us  25.267us  25.267us  cuDeviceGetName
                    0.00%  13.715us         2  6.8570us  2.0060us  11.709us  cudaMemAdvise
                    0.00%  5.9550us         1  5.9550us  5.9550us  5.9550us  cuDeviceGetPCIBusId
                    0.00%  1.8410us         1  1.8410us  1.8410us  1.8410us  cudaGetDevice
                    0.00%  1.6070us         3     535ns     198ns  1.1110us  cuDeviceGetCount
                    0.00%  1.1040us         2     552ns     298ns     806ns  cuDeviceGet
                    0.00%     496ns         1     496ns     496ns     496ns  cuModuleGetLoadingMode
                    0.00%     423ns         1     423ns     423ns     423ns  cuDeviceTotalMem
                    0.00%     236ns         1     236ns     236ns     236ns  cuDeviceGetUuid

==6172== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     512  2.0000MB  2.0000MB  2.0000MB  1.000000GB  89.77512ms  Host To Device
     512  2.0000MB  2.0000MB  2.0000MB  1.000000GB  82.17858ms  Device To Host
```