Software and Programming of High Performance Systems

2nd Exercise

Papageorgiou Spyridon Maximos Frountzos

The experiments were conducted on a system running 11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz, 16.0 GB DDR4 3600MT/S, and NVIDIA GeForce RTX 4060 8GB.

Question 1 (SIMD & WENO5)

a)

To enable automatic vectorization of the function ***weno_minus_reference*** the compile options (for gcc) are ***Q1***.

For vectorization of the function with OpenMP, before for loop the OpenMP SIMD option (#pragma omp simd) was enabled and the compile options (for gcc) are ***Q2***.

In the case of vectorization with SSE/AVX, AVX-512 SIMD matrices and a new ***weno_minus_core_avx512*** function called by ***the weno_minus_reference*** were used***.*** More specifically, the new function calculates the results per 16 floats using the AVX-512 matrices and when there are more elements (maximum 15) they are calculated with the scalar function. The same is done if the size of the inputs is less than 16 elements. The compile options (for gcc) are ***Q3.***

For the correct operation of the last implementation, a change was made to the ***myalloc*** function so that the alignment is done per 64 bytes instead of 32 since 16 floats x 4 bytes/float = 64 bytes/vector is processed. Also, in order to compile without errors in any case, the weno_minus_core_avx512 function is defined only when the __AVX512F__ flag has been activated by the compiler.

- *Q1 = -march=native -ftree-vectorize -O3 -fopt-info-vec-all -Wall*
- *Q2 = -fopenmp -O3 -Wall*
- *Q3 = -mavx512f -march=native -mavx -O3 -fno-tree-vectorize -Wall*

b)

To manage the different implementations, the following variables are used in the ***main*** function: *debug, debugEntries, verbose, optScenario*. When *debug* == 1 then the ***benchmark*** function is called once with input size *debugEntries.* If *verbose* == 1 then the values with which the inputs to the ***myalloc*** function were initialized are displayed***.*** All ***benchmark*** calls take the variable optScenario as an argument and depending on its value the corresponding implementation is applied.

- For the value 0, no optimization is applied and the scalar implementation of weno is executed.
- For value 1, auto vectorization is enabled and the scalar implementation is executed.
- For value 2, vectorization with OpenMP is enabled using scalar implementation.
- For value 3, and as long as the size of the inputs is not less than 16, the implementation with the AVX-512 matrices is called.

For each of these cases it is necessary to compile with the correct options. There is a correspondence between the Qi flags and the variable *optScenario,* i.e. if *optScenario* = i, then the correct options are Qi. In the case of *optScenario* = 0, Q0 = -Wall. Another change is that in the function ***weno_minus_reference*** the inputs were set with restrict, making the assumption that there is no aliasing***,*** so that the compiler does not bother with it***.***

Software and Programming of High Performance Systems

2nd Exercise

Papageorgiou Spyridon Maximos Frountzos

Below are performances for input size 1024.

```
Hello, weno benchmark!
nentries set to 1.024000e+03
using scalar core without optimizations
        time for call 1: 3.194809e-05
using scalar core without optimizations
        time for call 2: 2.002716e-05
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

```
Hello, weno benchmark!
nentries set to 1.024000e+03
using scalar core with compiler options
        time for call 1: 1.907349e-05
using scalar core with compiler options
        time for call 2: 9.059906e-06
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

```
Hello, weno benchmark!
nentries set to 1.024000e+03
using scalar core with OpenMP SIMD
        time for call 1: 2.193451e-05
using scalar core with OpenMP SIMD
        time for call 2: 1.096725e-05
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

```
Hello, weno benchmark!
nentries set to 1.024000e+03
using AVX512 core
        time for call 1: 1.406670e-05
using AVX512 core
        time for call 2: 1.907349e-06
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

Software and Programming of High Performance Systems

2nd Exercise

Papageorgiou Spyridon Maximos Frountzos

Below are executions for the maximum input size that we managed to implement without the process stopping from the operating system due to high memory consumption (~= 10^8).

```
Hello, weno benchmark!
nentries set to 1.677722e+08
using scalar core without optimizations
        time for call 1: 3.316660e+00
using scalar core without optimizations
        time for call 2: 3.192710e+00
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

```
Hello, weno benchmark!
nentries set to 1.677722e+08
using scalar core with compiler options
        time for call 1: 1.910782e+00
using scalar core with compiler options
        time for call 2: 1.229235e+00
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

```
Hello, weno benchmark!
nentries set to 1.677722e+08
using scalar core with OpenMP SIMD
        time for call 1: 1.848270e+00
using scalar core with OpenMP SIMD
        time for call 2: 1.646671e+00
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

```
Hello, weno benchmark!
nentries set to 1.677722e+08
using AVX512 core
        time for call 1: 9.265079e-01
using AVX512 core
        time for call 2: 4.018250e-01
minus: verifying accuracy with tolerance 1.00000e-05...passed!
```

For a smaller input size, there is not much difference in execution times, while for a larger one, the implementation with AVX-512 SIMD matrices seems better, but not for long. We also observe that implementation without optimizations is, as we expect, the worst in any case, but still not by a large margin. In conclusion, we see that the implementation with manual vectorization is not necessarily optimal and that the other implementations give similar results in a much simpler way.

Software and Programming of High Performance Systems

2nd Exercise

Papageorgiou Spyridon Maximos Frountzos

Question 2 (CUDA & CMM)

b)

The basic elements of the implementation are as follows. Initially, we decided not to define, initialize and transfer the input matrices from the host to the device. So only the result matrices(E, F) are defined on the host, while the input matrices (A, B, C, D) are defined with ***cudaMalloc*** on the device where they are initialized (in a deterministic way for debugging) with the kernel ***gpu_fill_matrix***. Then the function ***gpu_complex_matrix_multiply*** is called, which defines the intermediate calculation matrices (AC, BD, AD, BC) on the device with ***cudaMalloc*** and calculates them with the kernel ***gpu_matrix_multiply***. When these calculations are done the matrices E and F are calculated with the kernels ***gpu_matrix_subtract*** and ***gpu_matrix_add*** respectively. Then, E and F are copied to the host with ***cudaMemcpy***. For the execution of the kernels, 32x32 threads per block (1024 threads which is the maximum number of threads/block for this GPU) with N/32xN/32 blocks were used. In the case where N<32 we have a single block and NxN threads.

c)

In order to compare performances, a serialized version of the problem was implemented on a CPU that follows the same logical order. A ***get_wtime*** function was also used for timekeeping. The following variables are used in the ***main*** function to execute the code with different options***:*** *singleExec, maxExp, minExp, cpuExec, verbose, checkResults.* If *singleExec* == 1 then the calculation is done only once and in the way defined by *cpuExec*, i.e. if *cpuExec* == 1 then the CPU implementation is executed, while if it is 0 the GPU implementation is executed. In case *singleExec* == 0 then both implementations are executed for problem size N = 2^*minExp* to N = 2^(*maxExp - 1)* and if *checkResults* == 1 then it is checked if the results are the same. If *verbose* == 1 then only in the case of the single execution are the results displayed.

Below are the results for *singleExec* == 0, *minExp* == 1, *maxExp* == 11 and *checkResults* == 1.

```
CPU and GPU outputs are equal for N=2
CPU and GPU outputs are equal for N=4
CPU and GPU outputs are equal for N=8
CPU and GPU outputs are equal for N=16
CPU and GPU outputs are equal for N=32
CPU and GPU outputs are equal for N=64
CPU and GPU outputs are equal for N=128
CPU and GPU outputs are equal for N=256
CPU and GPU outputs are equal for N=512
CPU and GPU outputs are equal for N=1024
```

Software and Programming of High Performance Systems

2nd Exercise

Papageorgiou Spyridon Maximos Frountzos

Below are the results for *singleExec* == 0, *minExp* == 10 and *maxExp* == 13 and *checkResults* == 0.

```
running on CPU
        CPU finished with computation time:5.01439seconds for N=1024
running on GPU
        GPU finished with computation time:0.426594seconds for N=1024
running on CPU
        CPU finished with computation time:111.691seconds for N=2048
running on GPU
        GPU finished with computation time:3.57324seconds for N=2048
running on CPU
        CPU finished with computation time:1707.01seconds for N=4096
running on GPU
        GPU finished with computation time:25.1478seconds for N=4096
```

Below are the results for *singleExec* == 0, *minExp* == 2 and *maxExp* == 11 and *checkResults* == 0.

```
running on CPU
        CPU finished with computation time:2.465e-06seconds for N=4
running on GPU
        GPU finished with computation time:0.000162079seconds for N=4
running on CPU
        CPU finished with computation time:2.144e-06seconds for N=8
running on GPU
        GPU finished with computation time:9.0219e-05seconds for N=8
running on CPU
        CPU finished with computation time:2.3777e-05seconds for N=16
running on GPU
        GPU finished with computation time:0.000165844seconds for N=16
running on CPU
        CPU finished with computation time:6.3893e-05seconds for N=32
running on GPU
        GPU finished with computation time:0.000230603seconds for N=32
running on CPU
        CPU finished with computation time:0.000952901seconds for N=64
running on GPU
        GPU finished with computation time:0.00040225seconds for N=64
running on CPU
        CPU finished with computation time:0.00580577seconds for N=128
running on GPU
        GPU finished with computation time:0.00122717seconds for N=128
running on CPU
        CPU finished with computation time:0.0596seconds for N=256
running on GPU
        GPU finished with computation time:0.00888813seconds for N=256
running on CPU
        CPU finished with computation time:0.63088seconds for N=512
running on GPU
        GPU finished with computation time:0.0548295seconds for N=512
running on CPU
        CPU finished with computation time:5.16296seconds for N=1024
running on GPU
        GPU finished with computation time:0.448865seconds for N=1024
```

We notice that for small values of N the execution on a CPU is much faster than on a GPU, which is due to the transfer of data for processing to the GPU. The larger the N, the more the previous behavior is compensated, since now the cost of data transfer is not as significant compared to that of the operations that need to be done.