Lab 3

Dr. Shirley Moore

Oscar Galindo

Accelerator Programming – Heat Equation

1. Description of your strategy for the shared memory parallelization and optimization for the CPU. Please describe what programming model you chose and how you applied it. Please also describe the steps you went through in optimizing the performance of your program.

Sequential Time = 34.081634 seconds

For this parallelization I decided to first pursue an OpenMP driven parallelization. Nevertheless, through experimentation I realized that OpenACC offered better results. My strategy was as follows:

1. I searched for parallelizable section of code, in particular loops are of interest, with this in mind I observed the program basically had two areas where loops were used.
2. The first area which initializes the values of W and U was parallelized using OpenMP
3. Then I observed in the second section we had a sort of “work” section that basically made the biggest number of computations for the entire program.
4. With this in mind I executed the work and obtained the following results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Platform | Time | Initialization Parallelized | Working Section Parallelized |
| Full Parallel – 8 | OpenMP | 6.289s | Y | Y |
| Full Parallel - 16 | OpenMP | 4.258s | Y | Y |
| Full Parallel - 32 | OpenMP | 2.874s | Y | Y |

1. With the previous results I decided to try to not parallelize the initialization of variables and found an improvement by not initializing the data structures in parallel but only in the case of when the code was executed with 16 threads.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Platform | Time | Initialization Parallelized | Working Section Parallelized |
| Work Parallel – 8 | OpenMP | 6.595s | N | Y |
| Work Parallel - 16 | OpenMP | 3.600s | N | Y |
| Work Parallel - 32 | OpenMP | 2.877s | N | Y |

1. By using the previous results, I also decided to attempt to parallelize with OpenACC and found that by using less threads the code achieved a better performance. For this experimentation I decided not to include the case when the initialization was parallelized since I observed an improvement as seen between the cases with 16 threads in the previous tables. This experimentation basically made use of the pragmas #pragma acc parallel loop and #pragma acc parallel loop collapse (2) when needed, I also included the optimization level 4 as a deviation from what we discuss in class, these modifications obtained the following performance:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Platform | Time | Initialization Parallelized | Working Section Parallelized |
| Work Parallel – 4 | OpenACC | 1.034s | N | Y |
| Work Parallel – 8 | OpenACC | 0.688s | N | Y |
| Work Parallel - 16 | OpenACC | 0.670s | N | Y |
| Work Parallel - 32 | OpenACC | 1.23s | N | Y |

1. Using the previous results, I made the decision of going for an OpenACC implementation. As an improvement I included the directives of #pragma acc kernels reduction(…) gang(500) vector(500) so that the vectorization spawned 500 threads or 1 thread per computation in each of the 500 blocks in the 500x500 entries in arrays W,U, with this I specially tried to find the optimal number of threads to include in the problem and found the following results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Platform | Time | Initialization Parallelized | Working Section Parallelized |
| Work Parallel OPT – 4 | OpenACC | 1.010s | N | Y |
| Work Parallel OPT– 8 | OpenACC | 0.715s | N | Y |
| Work Parallel OPT - 10 | OpenACC | 0.610s | N | Y |
| Work Parallel OPT - 12 | OpenACC | 0.600s | N | Y |
| Work Parallel OPT - 14 | OpenACC | 0.647s | N | Y |
| Work Parallel OPT - 16 | OpenACC | 0.671s | N | Y |
| Work Parallel OPT - 32 | OpenACC | 1.064s | N | Y |

1. By obtaining the previous I decided to stop since the Speedup was already 0.60s/34.081634s = 56.8!
2. Description of your strategy for GPU parallelization and optimization. Please describe what programming model you chose and how you applied it. Please also describe the steps you went through in optimizing the performance of your program.
3. For this code I also started by looking what code could be parallelized. Indeed, I used what I learned in the previous problem for the CPU by using OpenACC.
4. I first ran an experiment and noticed there were many communications in/out from the device to the host. Observe:

Text

Description automatically generated

1. Because of the previous observations I decided to include a couple of pragmas #pragma acc data copyin of (u,w) and a #pragma acc data copyout alongside the #pragmas parallel loop reduction(…). This caused the execution to improve and have the following running time:

Text

Description automatically generated

1. I decided to implement a similar strategy as in the last steps of the CPU parallelization since the approach seemed even more reasonable for the qualities of the GPU, that is to have many threads specially if the problem was embarrassingly parallel as this heat problem, these changes caused the following execution performance:

Text

Description automatically generated

1. Although the gains were minimal between the modifications to vectorize and those to simply run with GPU code the Speedup was

34.081634/1.193 = 28.56!

* Why the CPU version vs. the GPU version did perform twice as fast can probably be attributed at the costs of communication between host and device.

1. Correct operation of the CPU and GPU versions.

* Output with correct execution with CPU: Text

  Description automatically generated
* Output with correct execution with GPU:

Text

Description automatically generated

1. Acceptable speedup on the CPU
   * Speedup was 56.8
2. Acceptable speedup on the GPU
   * Speedup was 28.56