PS 5 – Parallel computing

4) Relative speedups with the different types, the serial implementation, parallel and shared. Note that the shared memory version is not complete. I was not able to fix the corners of each threadblock. Some explanation of how to do it properly would be much appreciated in the feedback. I have included my failed attempt in the code.

To run the code in the parallel version, simply change the applykernelfilter\_shared function in main to applykernelfilter\_parallel.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Running 100 iterations with laplacian 1 |  |  |  |  |  |  |  |
| Serial: | Time: | Parallell: | Time: | Shared: |  | Time |  |
| 1 | 15,199 | 1 | 0,288 | 1 |  | 0,123 |  |
| 2 | 15,19 | 2 | 0,292 | 2 |  | 0,153 |  |
| 3 | 15,233 | 3 | 0,245 | 3 |  | 0,139 |  |
| 4 | 15,194 | 4 | 0,231 | 4 |  | 0,128 |  |
| 5 | 15,201 | 5 | 0,233 | 5 |  | 0,123 |  |
| Avg: | 15,2034 |  | 0,2578 |  |  | 0,1332 |  |
| Speedups: | Serial/Par | 58,97362 | par/shared | 1,935435 |  | serial/shared | 114,1396 |

5) The shared memory version I explained in the code. It can be seen in applykernelfilter\_shared.

6)

By using cudaOccupancyGetMaxPotentialBlockSize and using that value and dividing it on 32 in both the directions in our dim3 variable, we get a theoretical occupancy of 1. The reasoning for doing this is that the cudaOccupancyGetMaxPotentialBlockSize returns the value of a maximum sized in 1 direction, 1024. Now we need some way of arranging those 1024 threads and the easiest way of doing that is dividing by 32. Since 32^2 = 1024. That means we now have a square.

By running with for example, total threads in one threadblock = 1000, we get around a 50 % slowing and an occupancy of about 0.87. Bear in mind that this is for the shared version.