Wissam Antoun

201825674

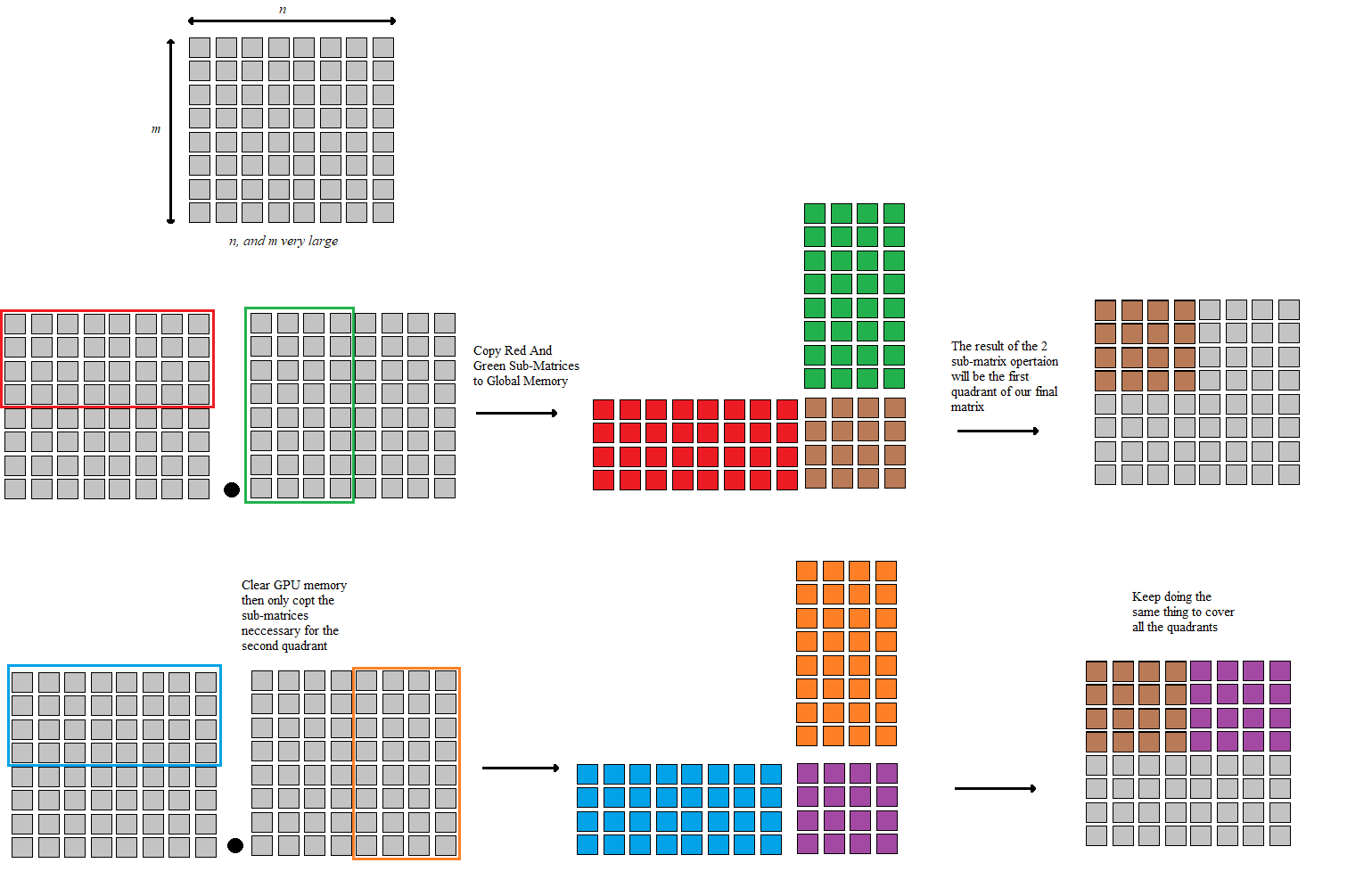
1. **Basic Matrix Mul**
2. Data:

|  |  |  |
| --- | --- | --- |
| Blocksize 16 | Basic | Tiled |
| Time needed to perform Memory allocation on the GPU | 0.001248 | 0.001376 |
| Time needed to perform Memory copy from the host to the device | 1.777888 | 1.19712 |
| Execution time of the kernel on the GPU | 6.11472 | 2.02112 |
| Time needed to copy the resulting Grayscale image to the CPU | 0.807264 | 1.221056 |
| Execution time of the serial implementation of the algorithm on the CPU | 578 | 578 |
| Speedup | 66.42822993 | 130.1604802 |

1. In the table
2. We have 960x800=768000 threads each one executing 2x640=1280 operations, for a total of 983040000 FLOPS
3. We have 960x800=768000 threads performing each 2x640=1280 reads, for a total of 983040000
4. We have 960x800=768000 threads performing each 1 write, for a total of 768000 writes
5. By using tiling, we can reduce the number of global memory accesses thus boost performance
6. **Tiled Matrix Mul – Block size 16**
7. Data: presented in the previous table
8. Presented in the previous table (I ran the 2 multiplication algorithm in the same file, hence 1 CPU execution time value)
9. We have 960x800=768000 threads each one executing 40 phases x16x2=1280 operations, for a total of 983040000 FLOPS
10. We have 960x800=768000 threads performing each 2x40=80 reads since they are storing simultaneously A and B in the Shared Memory. For a total of 61440000 reads
11. We have 960x800=768000 threads performing each 1 write, for a total of 768000 writes
12. **Tiled Matrix Mul – Block size 32**
13. Data:

|  |  |
| --- | --- |
| Blocksize 32 | Tiled |
| Time needed to perform Memory allocation on the GPU | 0.001152 |
| Time needed to perform Memory copy from the host to the device | 1.10912 |
| Execution time of the kernel on the GPU | 2.028096 |
| Time needed to copy the resulting Grayscale image to the CPU | 1.389888 |
| Execution time of the serial implementation of the algorithm on the CPU | 595 |
| Speedup\* | 131.3971648 |

1. In the table
2. We have 960x800=768000 threads each one executing 20 phases x32x2=1280 operations, for a total of 983040000 FLOPS
3. We have 960x800=768000 threads performing each 2x20=40 reads since they are storing simultaneously A and B in the Shared Memory. For a total of 30720000 reads.
4. We have 960x800=768000 threads performing each 1 write, for a total of 768000 writes
5. The execution time between the 2 Block sizes is the same, since im using a GTX 970 with 96KB shared memory per block, which allows up to 48 thread blocks active(using TileWidth of 16) or up to 12 thread blocks active(using a TileWidth of 32). But we are limited by the Max Number of threads per SM (2048), which will allow 8 simultaneous Thread Blocks (using TileWidth of 16) or only 2 Thread Blocks (using a TileWidth of 32). Both of these case will take the time to execute since the number of thread blocks waiting is 6 times the number of blocks being executed.
6. Large matrix Multiplication Algorithm (Divide and Conquer)



To get better results using this algorithm, we can keep large matrices like the red and blue in the global memory to reduce Memory copy times.

1. The Tiled Matrix Kernel hill suffer from issues when the matrix size isn’t a multiple of the Tile size. Fixing these issues will introduce control divergence problems that will negatively hit the performance, but will be less impactful when having large matrices.