Computer Architecture

Assignment 3

Mohammed Ameen, B210515CS

Introduction

The assignment focuses on analyzing performance of matrix multiplication using

- Sequential implementation which runs exclusively on a CPU. (SEQ)
- Parallel implementation using CUDA or HIP for a GPU but without using shared memory. (P1)
- Parallel implementation using CUDA or HIP for a GPU while using shared memory effectively. (P2)

Methodology

1. The experiment is done on **Intel Xeon E5-2630** v3 (16) @ 3.200GHz with **NVIDIA Tesla K20C GPU** using **CUDA** programming model.
2. The wall time of execution of each function or kernel is taken as the execution time.
3. Following dimensions have been chosen for the following block sizes:
    a. 128 threads per block: 16 x 8
    b. 256 threads per block: 16 x 16
    c. 512 threads per block: 32 x 16
4. For parallel computing without using shared memory, a naive method has been implemented where each thread will calculate one element of the resultant matrix by accessing a complete row and column from the first and second operands respectively.
5. Parallel computing using shared memory is implemented using 2D block tiling where the resultant matrix is divided into multiple sub matrices. Each thread computes a single element of the smaller matrices while the threads belonging to a

block collectively computes a single small matrix. Performance is further improved by applying loop unrolling to independent loops.

Performance Comparisons

Theoretical CGMA values for P1 and P2

Parallel Computing Without Shared Memory (P1)

No. of global memory accesses in a loop = 2 (one each for an element from A and B)

Total no. of global memory accesses = 2 * 8192 + 1 = 16385

No. of floating point operations in a single loop = 2 (one addition and one multiplication)

Total no. of floating point operations = 2 * 8192 = 16384

CGMA = (no. of FP operations) / (no. of global memory accesses) = 16384 / 16385 = 0.99993

Parallel Computing With Shared Memory (P2)

As the tiling block sizes vary for each of the three cases of block size (threads per block), the no. of access are also different in each.

1. **128 threads per block**

   Total no. of global memory accesses = (8 * 16 + 128 - 1) / 128 + (16 * 16 + 128 - 1) / 128 + 1 = 1 + 2 + 1 = 4

   Total no. of floating point operations = 2 * 16 = 32

   CGMA = 32 / 4 = 8

2. **256 threads per block**

Total no. of global memory accesses = (16 * 16 + 256 - 1) / 256 + (16 * 16 + 256 - 1) / 256 + 1 = 1 + 1 + 1 = 3

Total no. of floating point operations = 2 * 16 = 32

CGMA = 32 / 3 = 10.667

3. **512 threads per block**

Total no. of global memory accesses = (16 * 32 + 512 - 1) / 512 + (16 * 32 + 512 - 1) / 512+ 1 = 1 + 2 + 1 = 4

Total no. of floating point operations = 2 * 16 = 32

CGMA = 32 / 4 = 8

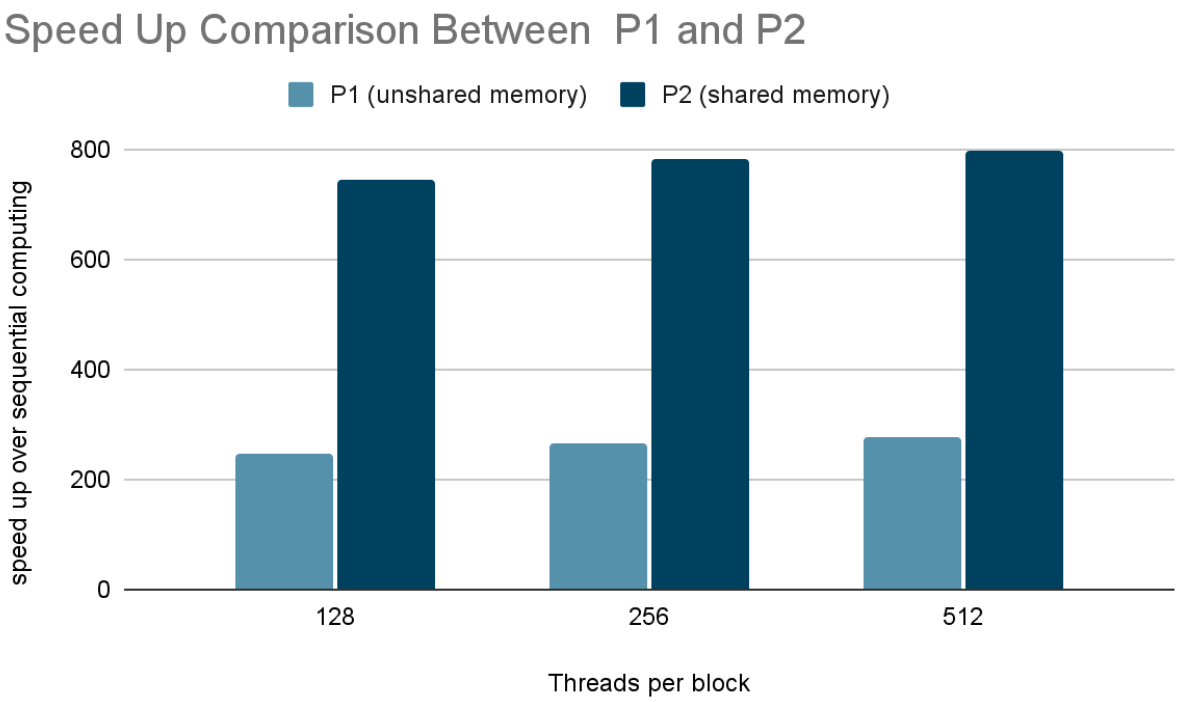
It is apparent from CGMA values that the computation with shared memory does floating computation per memory access (computation-bound) and hence, will perform better on GPUs where computational resources are underutilized. But the CGMA value comparison cannot give a broader view of the analysis.

Experimental Observations

The following observations have been made on executing the matrix multiplication in all three ways:

- Execution time for SEQ: 1196764.500000 ms.
- Execution time for P1:
    - 128 threads / block: 4826.181641 ms.
    - 256 threads / block: 4489.145020 ms.
    - 512 threads / block: 4333.833496 ms.

- Execution time for P2:
    - 128 threads / block: 1610.475220 ms.
    - 256 threads / block: 1529.561523 ms.
    - 512 threads / block: 1500.983643 ms.



Conclusion

The analysis shows that utilizing GPU's resources for computing problems that are parallelizable can speed up the performance by order or hundreds, and even thousands. We can further improve the performance by using shared memory within blocks, which can reduce global memory access of GPU considerably.