CSC612M G01
Clemenia, Dessa Louise C.
Villanueva, Zayin Benjamin A.
Wong, Jane Charmaine C.

Data Analysis for 1D and 2D Element-wise Matrix Multiplication (Hadamard Product) in C++ and CUDA

Introduction

For this project, the following variations were ran:

1. Using C++
    - 1D vector multiplication
    - 2D element-wise matrix multiplication
2. Using CUDA
    - 1D vector multiplication without dim3
    - 1D vector multiplication with dim3
    - 2D element-wise matrix multiplication with dim3

The following sections discuss the execution time of each run setting on both vectors and matrices of varying sizes. For CUDA programming output, the effect of block sizes and elements on execution time, and the performance of standard 1D versus 2D programming are also studied.

C++ versus CUDA

At the top level, it is very apparent that C++ programs which are not using GPU are a lot slower in executing the Hadamard Product multiplication as compared to using CUDA.

Comparing across the same data structure (e.g., C++ 1D versus CUDA 1D), CUDA is at most 66.48 times faster than C++ (for 2^24 vector size) when doing element-wise multiplication utilizing a 1D array.

1D vector	Ratio
2^20	63.18
2^22	66.34
2^24	66.48

The difference between C++ and CUDA is even greater when comparing 2D matrices. It shows that CUDA is faster by a whopping 174.07 times (for 4096 x 4096 matrix size).

2D matrix	Ratio
1024x1024	75.81
2048x2048	85.37
4096x4096	174.07

This fully supports the idea that having a GPU greatly improves or decreases execution time.

C++ using 1D versus 2D

Utilizing a 1D vector is a common way of transforming some large number of elements. One would think that implementing a 2D matrix will improve the run time, but it is actually the opposite. Comparing 1D vector and 2D matrix across the same number of total elements, 1D vectors are faster by 1.29 times to 2.65 times.

1D versus 2D	Ratio
2^20 or 1024x1024	1.29
2^22 or 2048x2048	1.29
2^24 or 4096x4096	2.65

CUDA using 1D versus 2D

Based on the results obtained for CUDA, standard 2D matrix computations run faster than 1D vector computations, and this is more evident with larger block sizes (1024 or 32x32). 1D vector results, on the other hand, showed that utilizing dim3 type does not affect execution time.

While the difference is not significant, 2D matrix computation with dim3 is still technically the fastest approach among all the test cases for CUDA by a small factor.

1D vectors without dim3 performs almost the same versus 1D vectors using dim3:

1D vector no dim3 versus 1D vector with dim3	256 or 16x16	1024 or 32x32
1024x1024	1.01	1.02
2048x2048	1.00	1.02
4096x4096	1.00	1.02

2D matrix using dim3 performs slightly faster than 1D vector with no dim3 when designating larger thread blocks (1024 or 32 x 32) by 1.07 to 1.10 times:

2D matrix with dim3 versus 1D vector no dim3	256 or 16x16	1024 or 32x32
1024x1024	0.93	1.07
2048x2048	1.00	1.09
4096x4096	0.99	1.10

2D matrix using dim3 performs slightly faster than 1D vector using dim3 when designating larger thread blocks (1024 or 32 x 32) by 1.09 to 1.12 times:

2D matrix with dim3 versus 1D vector with dim3	256 or 16x16	1024 or 32x32
1024x1024	0.94	1.09
2048x2048	1.00	1.11
4096x4096	0.99	1.12

On elements and block sizes

Among all the CUDA iterations done, they all fall within less than 2000 microseconds. More elements to transform requires more execution time as seen across all different settings. But increasing the block size does not necessarily mean it will decrease the execution time.

Take the case of the performance of CUDA using 2D matrix with dim3 in processing the largest matrix (4096 x 4096). When the block size is increased to 32 x 32, the execution time further increases to more than 1000 microseconds, as opposed to using smaller block sizes, where the execution time is less than 900 microseconds.

2D matrix with dim3	8x8	16x16	32x32
1024x1024	59.519	61.47	70.781
2048x2048	221.44	220.03	265.98
4096x4096	892.68	882.25	1039.7

A similar trend can also be observed in the case of CUDA using standard 1D vector without dim3.

1D vector without dim3	256	512	1024
2^20	57.055	62.239	75.677
2^22	219.07	240.15	289.37
2^24	870.54	949.74	1144.5

Conclusion

Processing 1D vectors versus 2D matrices has a large impact on the execution time when implementing element-wise multiplication using C++. This is not the case for CUDA 1D and 2D programming, as it was observed that there was no significant difference in their execution times.

When doing operations on either 1D vectors or 2D matrices, we see a significant improvement in execution time when we utilize the GPU as compared to leveraging solely on the CPU.

Overall, the results of the test cases above reinforce the advantage of leveraging on parallel programming. Whether the problem involves processing a large 2D matrix or a 1D array, programs utilizing GPU consistently outperform programs running on standard CPU power.

To conclude, when dealing with programming problems that can be broken down into sub-tasks that are of similar structure, and GPU resources are available, utilizing CUDA would substantially improve the program's execution time and performance.