**Team-mwy Project report**

**Team member:** Weigeng Li, Haoyang Ma, Sijing Yu

**Problem statement.**

In this project we use C++ to write cuda code. Codes are tested on our local server which has a Linux environment and a RTX 3090 GPU. In this Project we involve 3 cuda kernel to implement Vector adding, Matrix Multiplication and Tiled Matrix Multiplication with Shared Memory problem. In order to compare how size of vector or matrix effect running time in GPU and CPU, a CPU version code is also needed.

**Section 2: Implementation.**

**Vector Add:**

In vectoradd.cu we implement vector add cuda kernel and CPU codes. The size of vector various from 2^01 to 2^20 in each test. First, we allocated memory of 3 vector x, y and z in host. The element in vector x is assigned a float number 10.0 while element in vector y is 20.0. Then we allocated memory in device and copy x and y to device memory. After than is a for loop which use different length of vector and count the time spent in GPU and CPU. The dimension of block is 1. Threads in each block various. If the length of vector is smaller than 256, than it will only have threads same as the length and only 1 block. Otherwise, blocks will have 256 threads and there will be more block in grid. The dimension of grid is also 1, number of blocks in each grid various based on the length of vector. We various the number of thread because unused thread will cost more time in reading and cost time in if statement. Because it is a simple adding question we also made an error detect function to detect wrong result (z[i] !=30 is wrong)

In GPU kernel we use a for loop instead of an if statement. In the hint each thread will not only takes the adding of one element. When the length of vector is smaller than blockDim.x \* gridDim.x, the for loop only one time so it the same as if, else each thread will also take the adding of element in the blockDim.x \* gridDim.x stride after.

**Matrix Multiplication:**

The size of Matrix varies from 2\*2 to 2^9 \*2\*9. Each thread needs to calculate a element in matrix so the dimension of block and grid is 2 now. Each block has 32\*32 thread if the Matrix is bigger than 32\*32. Otherwise, same size as the matrix to save time from reading data and if statement. Number of blocks in each grid various based on the width of matrix. Because the result of each element various based on the size of matrix, we do not have error detect in this problem

The kernel code is basis from the hint and change to our version. In this problem we are facing N\*N matrix multiply N\* N matrix, so we don’t need width variable. Each thread calculate the product of column in matrix x and row in matrix y and sum them together. Then write them to matrix z.

**Tiled Matrix Multiplication:**

The CPU code is the same as Matrix Multiplication. We use fixed block size here. We change the size of shard memory to 32\*32 to fit the max size of 32\*32 thread in each block. The number of grids is also modified to fit numbers of block. We synchronize the threads both after block initialization and multiplication.

**Section 3: Experimental results.**

**Evaluation methodology:**

Vector add: Change the size of vector form 21 to 220

Matrix Multiplication: Change wide and length of Matrix from 21 to 29

We mark the start time and the end time and calculate difference of time in . CPU running time only contains the calculation, GPU running time contains copying memory between host and device and running the GPU kernel function.

文本

描述已自动生成

Figure 3.1 Original output (Unit of time: milliseconds)

From The original Output,

Including these data will increase the scale of pot so I removed them to made the plot better.

Figure 3.2 Vector Add result

Figure 3.2 shows the time spend of CPU and GPU while adding two 2^N length vector. We can see when the length of Vector reach 2^12, GPU is faster than CPU. Also, time spent of GPU does not increase as rapidly as CPU.

­­

Figure 3.3 Matrix Multiplication Result

From Figure 3.3, After 25\* 25 matrix, CPU runs slower than GPU. Because matix multiplication need more caculation so the point of CPU slower than GPU comes earlier than expected in Vector Add section(25\* 25 =210<212)

Figure 3.3 Tiled Matrix Multiplication Result

In Tiled Matrix Multiplication we get similar result to Matrix Multiplication, to learn more we can compare the GPU result.

Figure 3.4 Compare of Tiled Matrix Multiplication

From Figure 3.4 in most time Tiled Matrix Multiplication runs faster than Matrix Multiplication. Because the use of shared memory, the times of accessing data is less which increase the overall running time.

**Section 4: Summary and Takeaway.**

In this project, we implement 3 coda kernel which are Vector adding, Matrix Multiplication and Tiled Matrix Multiplication with shared memory problem by C++. Then we compare the running time in GPU and CPU. We find that when matrix size become big enough, the speed of these three Matrix operations in GPU is much faster than CPU. The difference of speed will become larger with the increase of matrix size. Because multiplying matrix is more compute-intensive operations than matrix add, the difference of running time in GPU and CPU appears starting from smaller matrix size than matrix add operations. Finally, we find in most time Tiled Matrix Multiplication runs faster than Matrix Multiplication

In this project, we learned how to use CUDA to accelerate computation. We know the difference between CPU and GPU computing and know how GPU acceleration works. After this project, we can implement some simple CUDA acceleration in our daily programming.

**Section 5: Team Contribution.**

Weigeng Li: Implement Vector adding in GPU and CPU.

Haoyang Ma: Implement Vector multiplication in GPU and CPU. makefile and test.sh

Sijing Yu: Implement Tiled Matrix Multiplication with shared memory problem in GPU and CPU.

We write the report together.

**Section 6: Artifact evaluation.**

Please run test.sh. This script will automatically make and run the 3 cuda files. Make sure to give a execute permission(chmod u+x test.sh) or use sh to run this script(sh test.sh).