18645 Mini Project2 Report

team034: Xinran Fang(xinranf) & Xin Wang(xinw3)

1 Matrix-Multiplication

1.1 Optimization goal

The goal for the matrix multiplication project is to modify the code so that it could not only work for power of 2 input matrix sizes, but also work for any input sizes matrix. And the code should achieve at least 150 GFLOPS for CUDA version. To improve the speed of the matrix multiplication, our group focused on implementing parallel computing platform CUDA, as well as other methods such as loop unrolling and __syncthreads().

1.2 Optimization process

- **Data consideration**

For the given code, it can only implement matrix multiplication for power of 2 input sizes. For test case like matrix size is equal to 33, the program will hang and get stuck.

```
******CUDA*****

Test Case 1     0.00308881 Gflop/s

Test Case 2     0.00625612 Gflop/s

Test Case 3     0.213029 Gflop/s

Test Case 4+ cp cuda_results.xml /afs/andrew.cmu.edu/usr12/jchong/workspace/team034_matmul/
```

To ensure that the program can apply to any input size, we modify the dimBlock and dimGrid values which pass to kernel function, as well as the conditions in the matrix_mul_kernel function.

```
****** Fetching data file ******

******CUDA*****

Test Case 1     0.0028249 Gflop/s

Test Case 2     0.00597619 Gflop/s

Test Case 3     0.186756 Gflop/s

Test Case 4     0.221641 Gflop/s

Test Case 5     104.121 Gflop/s

Test Case 6     104.396 Gflop/s
+ cp cuda_results.xml /afs/andrew.cmu.edu/usr12/jchong/workspace/team034_matmul/
```

- **Parallelization consideration**

CUDA is suited for the highly parallel program, by taking advantages of how the matrix multiplication works, CUDA can break the program down into many threads and execute them in parallel. The built-in device variable such like, blockDim, blockIdx and threadIdx are used to identify and differentiate the GPU threads which will execute the kernel in parallel. Including threadIdx and blockIdx, we can label the thread index and block index, we create 2D grid which provides much faster accesses.

Each threads in CUDA has their own registers and also shares memory address space. ThreadIdx variable contains the thread index within the block, which shares the execute instructions in parallel.

The other method our group used is loop unroll. Adding `#pragma unroll` in the global function and including -O3 in Makefile file, the program can execute instructions parallel and increase the speed.

1.3 Optimization results

- **Before optimization**

```
****** Fetching data file ******

******CUDA*****

Test Case 1     0.0031054 Gflop/s

Test Case 2     0.00621862 Gflop/s

Test Case 3     0.20882 Gflop/s

Test Case 4+ cp cuda_results.xml /afs/andrew.cmu.edu/usr12/jchong/workspace/team034_matmul/
[xUnit] [INFO] - Starting to record.
[xUnit] [INFO] - Processing CppUnit-1.12.1 (default)
[xUnit] [INFO] - [CppUnit-1.12.1 (default)] - 1 test report file(s) were found with the pattern '*.xml' relative to
'/afs/andrew.cmu.edu/usr12/jchong/workspace/team034_matmul' for the testing framework 'CppUnit-1.12.1 (default)'.
[xUnit] [ERROR] - Clock on this slave is out of sync with the master, and therefore
I can't figure out what test results are new and what are old.
Please keep the slave clock in sync with the master.
[xUnit] [INFO] - Setting the build status to FAILURE
[xUnit] [INFO] - Stopping recording.
Finished: FAILURE
```

- **After optimization**

```
******CUDA*****
Test Case 1     0.0031182 Gflop/s
Test Case 2     0.00620243 Gflop/s
Test Case 3     0.213356 Gflop/s
Test Case 4     0.229992 Gflop/s
Test Case 5     163.804 Gflop/s
Test Case 6     173.19 Gflop/s
```

Achieve at least 163 Gflop/s.

For square matrices A, B, C, which have dimension of M, N, K, where M==N==K. Using CUDA, the amount of computation is 2 * M * N * K (flops). Since we are calculating floating points, the computation-to-memory ratio is (⅛ flop/bytes). In each thread, the program loads one row from the first matrix and one column from the second matrix from the global memory, does the calculations and stores the result back to result_matrix in global memory. However, there are some other calculations which causes the speedup cannot reach its maximum.

2 K-Means

2.1 Optimization goal

Achieve at least 1.5x speedup compared to initial cuda version.

- **Using CUDA to Introduce Manycore Programming**

CUDA is a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU.

- **Warp**

A group of threads with consecutive thread indexes are bundled into a warp, in which one full warp is executed on a single CUDA core. Blocks are divided into warps of 32 threads for execution. But unlike Grid and Block, its implementation detail is not directly accessible by programmers.

- **Single Instruction Multiple Thread (SIMT)**

In SIMT, multiple threads are processed by a single instruction in lock step. It's similar to SIMD, but with multiple threads.

- **Shared Memory**

Because it's on-chip, shared memory is more efficient than global memory.

2.2 Optimization process

- **Make it runnable**

At first, The program always failed in the last two test cases. The information provided is "invalid configuration argument." Then we change the parameter `numThreadsPerClusterBlock` to 1024 and it worked. It's very misleading because in the comment, it says that this parameter can not be longer than an unsigned char. But actually it should be the maximum threads per block.

```
Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans01.dat
numObjs       = 351
numCoords     = 34
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 10
I/O time           =     0.0090 sec
Computation timing =     0.3245 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/tea

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans02.dat
numObjs       = 7089
numCoords     = 4
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 66
I/O time           =     0.0206 sec
Computation timing =     0.2870 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans03.dat
numObjs       = 191681
numCoords     = 22
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 38
I/O time           =     0.5505 sec
Computation timing =     0.5382 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans04.dat
numObjs       = 488565
numCoords     = 8
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 1
I/O time           =     0.6225 sec
Computation timing =     0.2944 sec
Number of mismatches exceeds 5% of data set
Finished: FAILURE
```

- **Make it correct**

After it can run on all the test cases, it still failed. Because the correctness cannot meet the expectation. We analyzed the result and found that the wrong output came from the last case, in which the loop iteration is only one and it quit the program. The stop criteria of this kmeans is when the condition `delta > threshold && loop++ < 500` cannot be meet. Maybe in the last case, the delta has already less than the threshold in the first loop. So we add another condition, which is `loop < 3`, this way, the last case can continue even if its delta has less than the threshold.

```
Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans01.dat
numObjs       = 351
numCoords     = 34
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 10
I/O time           =     0.0100 sec
Computation timing =     0.3376 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans02.dat
numObjs       = 7089
numCoords     = 4
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 66
I/O time           =     0.0204 sec
Computation timing =     0.2981 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans03.dat
numObjs       = 191681
numCoords     = 22
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 38
I/O time           =     0.5448 sec
Computation timing =     1.0474 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans04.dat
numObjs       = 488565
numCoords     = 8
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 241
I/O time           =     0.6817 sec
Computation timing =     4.2327 sec
Finished: SUCCESS
```

- **Kernelization**

We change the for loop to kernel function which defined by __global__ keywords. Then when invoking kernel, we pass the block size and the number of threads to the kernel function so that multiple threads can execute the program at the same time. Our computation time has been shortened a lot.

```
Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans01.dat
numObjs       = 351
numCoords     = 34
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 10
I/O time           =     0.0088 sec
Computation timing =     0.3021 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/tea

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans02.dat
numObjs       = 7089
numCoords     = 4
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 66
I/O time           =     0.0168 sec
Computation timing =     0.3016 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans03.dat
numObjs       = 191681
numCoords     = 22
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 38
I/O time           =     0.5732 sec
Computation timing =     0.4681 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans04.dat
numObjs       = 488565
numCoords     = 8
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 241
I/O time           =     0.6730 sec
Computation timing =     0.7539 sec
Finished: SUCCESS
```

- **Loop unrolling**

By implementing `#pragma unroll`, our performance has been improved a little.

```
Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans01.dat
numObjs       = 351
numCoords     = 34
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 10
I/O time           =     0.0095 sec
Computation timing =     0.2999 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/tea

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans02.dat
numObjs       = 7089
numCoords     = 4
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 66
I/O time           =     0.0133 sec
Computation timing =     0.2726 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans03.dat
numObjs       = 191681
numCoords     = 22
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 38
I/O time           =     0.5751 sec
Computation timing =     0.4337 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/t

Performing **** Regular Kmeans (CUDA version) ****
Input file:     /afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/team034/fastcode/kmeans/kmeans04.dat
numObjs       = 488565
numCoords     = 8
numClusters   = 32
threshold     = 0.0010
Loop iterations    = 241
I/O time           =     0.6530 sec
Computation timing =     0.6667 sec
Finished: SUCCESS
```

- **Change Makefile**

We add `-O3` to Makefile and made some optimization to the final results.

2.3 Optimization results

Finally, our computation time is about 0.6667 sec compared to the original 4.2327 sec, which has a **6.3x** speed up.

The expected speed up for implementing CUDA is 10.5x. Because there already exists small portion of CUDA code in original program, the final speed up may be less than expected.

3 Reference

[1]. https://devblogs.nvidia.com/parallelforall/easy-introduction-cuda-c-and-c/
[2]. http://stackoverflow.com/questions/16619274/cuda-griddim-and-blockdim
[3]. http://www.ciemat.es/EUFORIA/recursos/doc/pdf/717802527_151020101001.pdf
[4]. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf
[5]. https://en.wikipedia.org/wiki/K-means_clustering#Cluster_analysis
[6]. http://cs.nyu.edu/courses/spring12/CSCI-GA.3033-012/lecture5.pdf
[7]. https://cvw.cac.cornell.edu/gpu/shared_mem_exec