18645 Mini Project1 Report

BY TEAM034

Xinran Fang(xinranf) & Xin Wang(xinw3)

1 Matrix Multiplication

- **Optimization Goal**

  The goal of this project is to optimize the matrix multiplication process to make the OpenMP version achieve at least 5X speed up compared to the sequential version. For this purpose, our team use SIMD to compute the same multiple process simultaneously by using single instruction. Considering the hardware environment to use SIMD instructions will accelerate the program.

- **General performance**
  - Sequential

```
******Sequential*****
Test Case 1     0.00219727 milliseconds
Test Case 2     0.00317383 milliseconds
Test Case 3     0.0981445 milliseconds
Test Case 4     0.11499 milliseconds
Test Case 5     3488.72 milliseconds
Test Case 6     4918.18 milliseconds
```

  - After Optimization

```
******OMP*****
Test Case 1     0.00390625 milliseconds
Test Case 2     0.00195312 milliseconds
Test Case 3     0.0239258 milliseconds
Test Case 4     0.0129395 milliseconds
Test Case 5     191.624 milliseconds
Test Case 6     183.613 milliseconds
```

- **Optimization Process**

  → Use OpenMP for Multicore application

  Add `#pragma omp parallel for` before some for loops.

  When a thread encounters a `parallel` construct, a team of threads is created to execute the parallel region. All threads in the new team, including the master threads, execute the region, in which shows the concept of parallelism programming.

```
******OMP*****
Test Case 1     0.00415039 milliseconds
Test Case 2     0.000976562 milliseconds
Test Case 3     0.0358887 milliseconds
Test Case 4     0.0378418 milliseconds
Test Case 5     1098.42 milliseconds
Test Case 6     2673.36 milliseconds
```

  → Cache

  Transpose the second matrix before do multiplication.

  The cache miss rate of the naive matrix multiplication is very high using the oridinary matrix multiplication. In order to reduce the miss rate, we need to utilize the spatial locality. So we do transpose before the multiplication.

  → SIMD

- SSE

We change the original matrix transpose and multiplication code to SIMD version to utilize the "Single Instruction Multiple Data". By using `_mm_loadu_ps`, `_mm_add_ps` etc, we can load or add 4 float data at the same time so that do not need to execute for loop 4 times. The performance has been changed dramatically.

```
******OMP*****
Test Case 1     0.00195312 milliseconds
Test Case 2     0.00219727 milliseconds
Test Case 3     0.0200195 milliseconds
Test Case 4     0.0158691 milliseconds
Test Case 5     242.244 milliseconds
Test Case 6     262.546 milliseconds
```

- AVX

It's the upgraded version of SSE, which can load and do math operations with 8 numbers at the same time.

```
******OMP*****
Test Case 1     0.00390625 milliseconds
Test Case 2     0.00195312 milliseconds
Test Case 3     0.0239258 milliseconds
Test Case 4     0.0129395 milliseconds
Test Case 5     191.624 milliseconds
Test Case 6     183.613 milliseconds
```

- **Optimization Results**

  Speed up: 26.87x

2 K-means

- **Optimization Goal**

  For the part two of the mini project1 is to optimize k-means method to obtain the goal that the speedup should be at least 1.5X compared to current OpenMP version. To achieve this purpose, our team applied SIMD, loop unrolling and multicore methods. By applying these methods, we can compute data parallelly and take advantage of multicore to operate the different processes at the same time.

- **General Performance**

  – Before Optimization (OpenMP Version)

```
Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.0064 sec
Computation timing =     0.0057 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.0120 sec
Computation timing =     0.4728 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.5748 sec
Computation timing =     9.1008 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.7304 sec
Computation timing =   115.4060 sec
Finished: SUCCESS
```

- After Optimization

```
Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.0064 sec
Computation timing =     0.0011 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.0116 sec
Computation timing =     0.0395 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.5615 sec
Computation timing =     1.5781 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.7184 sec
Computation timing =    15.3127 sec
Finished: SUCCESS
```

- **Optimization Process**

  After analyzing the k-means algorithm, we found out that euclid_dist_2 function are most called. Therefore, optimizing euclid_dist_2 function is the most effective way to speedup the entire project. This function is to calculate the distance between two points. We decided to use SIMD to load four values and compute these values simultaneously. Also, find_nearest_cluster function plays an important role in the k-means project. We decided to use loop unrolling to optimize the program's execution speed. Loop unrolling can limit the iteration times. However, effectiveness of loop unrolling is not as good as SIMD.

  → Use OpenMP for Multicore application

  Add `#pragma omp parallel`.

  Similar to the optimization with matrix_mul.

  → SIMD

  We change the `euclid_dist_2()` function to SIMD version. The theory is as before. The performance has been improved a lot.

```
Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.0064 sec
Computation timing =     0.0011 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=7089 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/te

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.0116 sec
Computation timing =     0.0395 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=191681 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.5615 sec
Computation timing =     1.5781 sec
Writing coordinates of K=32 cluster centers to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/
Writing membership of N=488565 data objects to file "/afs/andrew.cmu.edu/usr12/jchong/18645_spring_2017/codes/

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
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
I/O time           =     0.7184 sec
Computation timing =    15.3127 sec
Finished: SUCCESS
```

- **Optimization Results**

  Speed up: 7.54x

3 Task 0 Answers

- Question 1: running time for sequential and OpenMp implementations.
  Sequential: 4918.18 ms
  OpenMP: 183.613 ms
- Question 2:
  a) The configuration of K-means algorithm being tested.

	kmeans01.dat	kmeans02.dat	kmeans03.dat	kmeans04.dat
numObjs	351	7089	191681	488565
numCoords	34	4	22	8
numClusters	32	32	32	32
threshold	0.0010	0.0010	0.0010	0.0010

Table 1. the configuration of K-means algorithm being tested.

  b) The running time for I/O and computation for sequential and OpenMP implementations.

	kmeans01.dat	kmeans02.dat	kmeans03.dat	kmeans04.dat
I/O	0.0063 sec	0.0124 sec	0.5512 sec	1.3222 sec
Computation	0.0165 sec	0.4252 sec	21.8485 sec	164.3284 sec

Table 2. I/O and computation time for sequential version

	kmeans01.dat	kmeans02.dat	kmeans03.dat	kmeans04.dat
I/O	0.0064 sec	0.0116 sec	0.5615 sec	0.7184 sec
Computation	0.0011 sec	0.0395 sec	1.5781 sec	15.3127 sec

Table 3. I/O and computation time for OpenMP version

Reference

1. http://www.openmp.org/
2. https://software.intel.com/sites/default/files/m/d/4/1/d/8/Intro_to_Intel_AVX.pdf
3. http://blog.csdn.net/xb554790401/article/details/38404265
4. https://msdn.microsoft.com/en-us/library/s3h4ay6y(v=vs.90).aspx
5. https://git-scm.com/