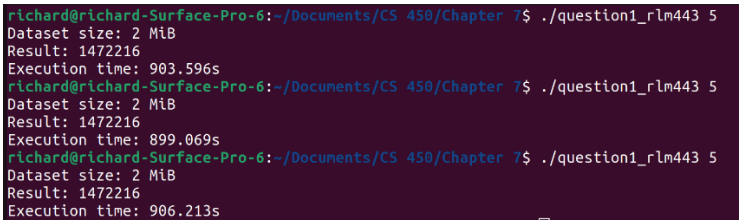
CS 450 – Assignment 7

Richard McCormick (RLM443)

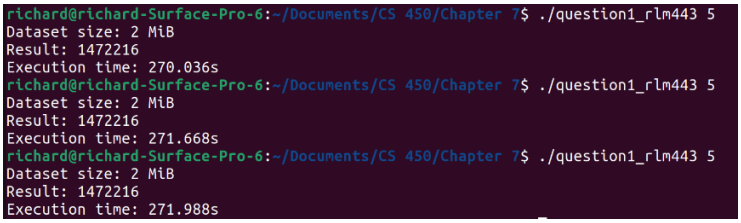
Question 1:

Subquestion (ϵ = 5)	#Cores	Time (s) No Opt.	Time (s) w/ -O3	Speedup	Parallel Efficiency
Sequential	1	902.9593	NA	1	1
(a)	8	271.2306	NA	3.329	41.6%
Sequential	1	NA	26.9554	1	1
(b)	8	NA	7.7292	3.487	43.5%

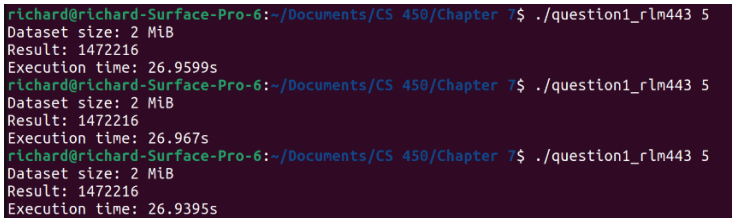
Question 1 – Sequential – No Optimizations



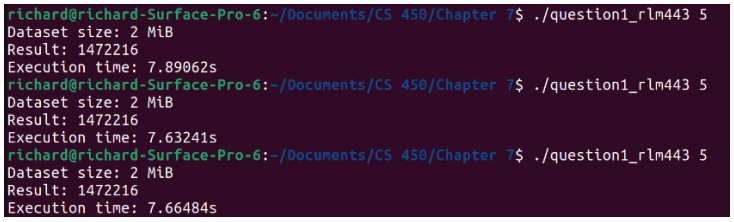
Question 1 – Parallel – No Optimizations



Question 1 – Sequential – O3



Question 1 – Parallel – O3



Question 2:

Subquestion (ϵ = 5)	#Cores	Time (s) No Opt.	Time (s) w/ -O3	Speedup	Parallel Efficiency	T1/T2 No Opt.	T1/T2 -O3
Sequential	1	0.6997	NA	1	1	NA	NA
(b)	8	0.2363	NA	2.96	37.01%	NA	NA
Sequential	1	NA	0.1713	1	1	NA	NA
(c)	8	NA	0.0608	2.82	35.2%	NA	NA
(d)	NA	NA	NA	NA	NA	1,148	127

Question 2 – Sequential – No Optimizations

```
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.70292s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.698438s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.697593s
```

Question 2 – Parallel – No Optimizations

```
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.236069s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.235626s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.237088s
```

Question 2 – Sequential – O3

```
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.167347s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.171483s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.175013s
```

Question 2 – Parallel – O3

```
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.0586388s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.0647407s
richard@richard-Surface-Pro-6:~/Documents/CS 450/Chapter 7$ ./question2_rlm443 5
Dataset size: 2 MiB
Result: 1472216
Execution time: 0.0590745s
```

Question 2 (Cont.):

(e) Is your new algorithm faster? Why or why not? What optimizations worked well? What ideas did you try that did not perform well?

The new algorithm is certainly faster. It makes use of multiple optimizations that significantly reduce the runtime of the program. Specifically, modifications to both the algorithm and the structure of the code improved performance. Some ideas that I tried that did not work included trying to store parts of the computation as points rather than doubles and trying to use a vector for the points instead of the array.

(f) Give a more detailed comparison of the performance of your algorithm in comparison to the brute force algorithm. Use other metrics to convey why your program is now faster. Examples include: reduction in floating point operations, reduction in the number of points compared, counting cache misses, and others.

There are two main changes that I credit to the performance increase in this code. The first was modifying the for loops so that they were not going back over points which had already been visited. This significantly reduced the number of total calculations and gave a significant boost to performance.

The second change I made was in the actual calculation of the distance between each point. I removed the math functions which were called in the original implementation and simplified it to use normal operators. Additionally, I began comparing the distance without the square root function to epsilon squared, instead of epsilon. This allowed the square root function to be removed entirely, and significantly sped up the runtime.

I also added a break in the inner for loop. This triggers when the difference in the distance from origin of both points is greater than epsilon. The assumption is that no points after this will be closer than epsilon, so we can save computations. This change allowed my code to go from ~8 seconds to ~0.25 seconds without the O3 optimization.

Other metrics which can show the increase in efficiency include a reduction in the number of total calculations made, a significant reduction in the number of points compared, and a complete reduction in number of repeated calculations.

(g) Do these metrics translate linearly to the observed reduction in response time as compared to the brute force algorithm? Why or why not?

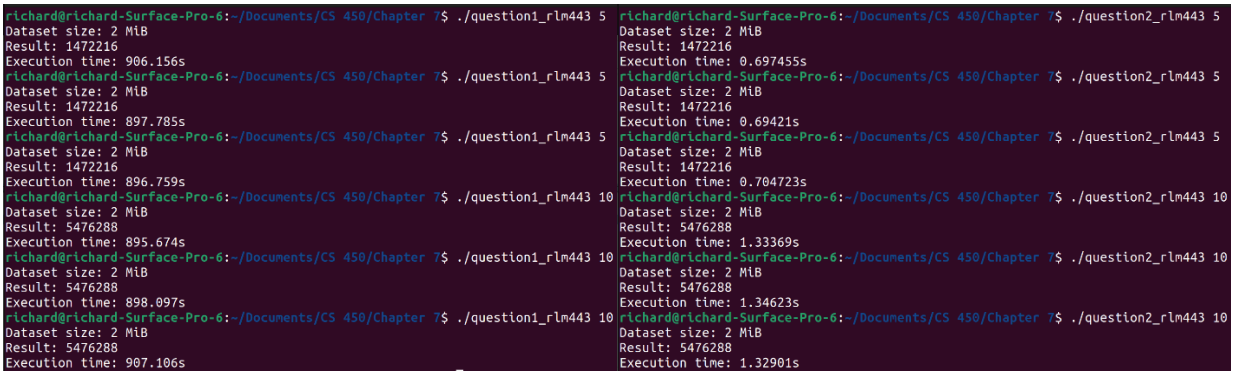
These metrics do translate linearly to a reduction in response time compared to the brute force algorithm. This is because as the size of the input grows, the improved algorithm will reduce the number of computations required proportionally. There will be less computations per input, and the computations that are performed will be simpler and require less resources.

Bonus Question:

The following are results for the *sequential* versions of my algorithms.

Epsilon	Brute-Force	Improved	Ratio
5	900.233	0.6986	1,289
10	900.292	1.3363	674

Left: Brute Force, no opt. Right: Improved, no opt.



Epsilon	Brute-Force -O3	Improved -O3	Ratio
5	26.957	0.170991	158
10	26.903	0.32107	84

Left: Brute Force, -O3. Right: Improved, -O3.