ANLY550 Programming Assignment 2

Group: Chang Sun, Yi Li

Analytical Method

The product of two $n * n$ matrices A and B is a $n * n$ matrix C. We know that $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ for $i, j = 1, 2, ..., n$, and there are n multiplications and $n - 1$ additions. As we need to compute n^2 entries of C, the conventional matrix application of multiplying two $n * n$ matrices requires n^3 multiplications and $n^3 - n^2$ additions. So the total arithmetic operation count is $2n^3 - n^2$.

The Strassen's algorithm uses 7 multiplications and 18 additions/substractions per recurrence for multipliying 2 x 2 matrices whose elements are $\frac{n}{2} * \frac{n}{2}$ blocks. So the total operation count per recurrence is $T(n) = 7T(\frac{n}{2}) + 18(\frac{n}{2})^2 = 7(2(\frac{n}{2})^3 - (\frac{n}{2})^2) + 18(\frac{n}{2})^2 = (\frac{7}{4})n^3 + (\frac{11}{4})n^2$.

The cross-over point n_0 is the point that we can stop the Strassen's algorithm and switch to conventional matrix multiplication below that point. To find the cross-over point, set $2n^3 - n^2$ equals to $(\frac{7}{4})n^3 + (\frac{11}{4})n^2$. Solve this equation and we have $n = 15$. Therefore, the cross-over point n_0 is 15. When $n = 16$, the Strassen's algorithm starts to perform better than conventional matrix multiplication.

Experimental Method

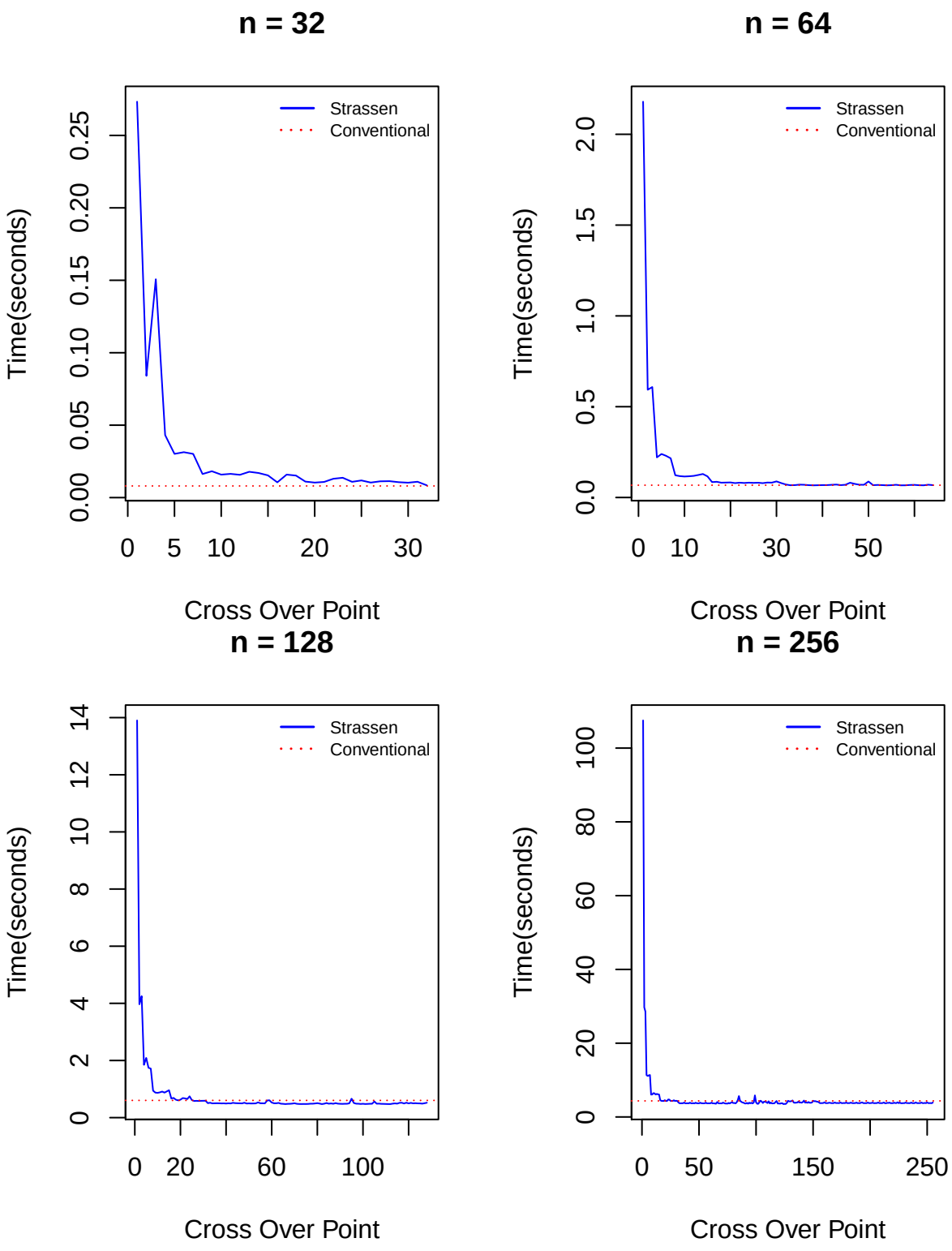
The code is shown in strassen.py.

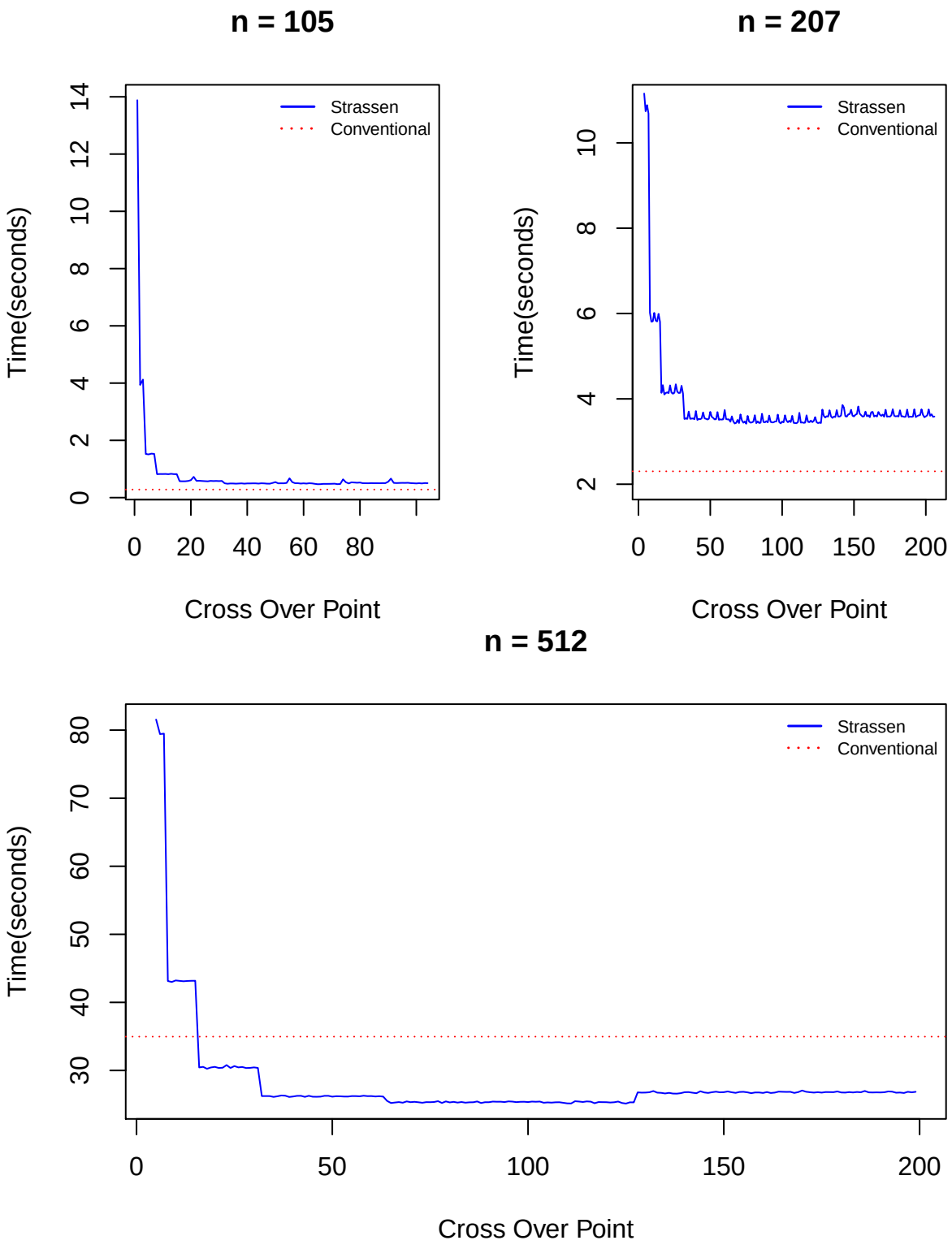
We first define a function called *ikj _ matrix _ product* to do the conventional matrix multiplication. The *add* and *subtract* functions are used to calculate addition and substraction. The Strassen's algorithm is realized in the *StrassenR* function. We initialize each new sub-matrices with the new size $\frac{n}{2}$ and divide the matrices into 4 sub-matrices, named $a_{11}, a_{12}, a_{21}, a_{22}$ and $b_{11}, b_{12}, b_{21}, b_{22}$. So p_1 to p_7 can be calculated, and we can get the resulting matrix, which is matrix C representing A times B.

When the dimension n is a power of 2, we can apply Strassen's algorithm directly. To handle more general dimensions like odd numbers, we fill the matrix with 0s until the dimension of the matrix becomes the closest power of 2 that is greater than n. For example, if A is a 300 x 300 matrix whose dimension is between $2^8 = 256$ and $2^9 = 512$. We add zeros to A and get a new 512 x 512 matrix A whose first 300 x 300 elements are the same as the original A, while other elements equal to 0.

In order to find the cross-over point, we set a range of numbers as the cross-over points and apply these numbers one at a time into the algorithm. We combine the conventional matrix multiplication and Strassen's agorithm together in the *StrassenR* function to get a modified version of Strassen's algorithm: Once the dimension belows the set cross-over point, it runs a conventional matrix multiplication. Otherwise, it recursively calls Strassen's algorithm to do the multiplication.

We test dimensions (n) of 32, 64, 105, 128, 207, 256, and 512 on matrices A and B whose elements are integers randomly selected in the range $[-2, 2]$. (We also tried to use randomly selected float numbers and numbers around 100, and their runtimes are similar, so the types of matrices do not matters a lot in this case.) With each dimension, we set the cross-over point to numbers from 1 to n. Because there is no need to check the cross-over points that are larger than the dimension of the matrix. The results of the cross-over points with runtimes are shown in the graphs below. The blue lines are the runtime for Strassen's algorithm. The red dash lines represent the conventional matrix multiplication, which is the runtime for using only the conventional approach.


n = 32
Strassen
Conventional
Time(seconds)
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
20
30
Cross Over Point
n = 64
Strassen
Conventional
Time(seconds)
0.0
0.5
1.0
1.5
2.0
0
10
30
50
Cross Over Point
n = 128
Strassen
Conventional
Time(seconds)
0
2
4
6
8
10
12
14
0
20
60
100
Cross Over Point
n = 256
Strassen
Conventional
Time(seconds)
0
20
40
60
80
100
0
50
150
250
Cross Over Point



When the cross-over point is set too low, the algorithm does more reccursions of Strassen's algorithm than the ideal situation, thus it causes longer runtime. On the other hand, when the cross-over point is set too high, the algorithm does not maximally utilize the Strassen's algorithm, so it also results longer runtime. Therefore, the minimal runtime corresponds to the cross-over point we are looking for.

We can know from the graphs that as the set cross-over point becomes larger, the runtime initially

decreases sharply. For large dimensions like the graphs $n = 207$ and $n = 512$, we can see there is a stepped pattern for the cross-over points in the range like $[8, 16), [16, 32), [32, 64), [64, 128), [128, 256)$ and so on. The cut off points for the steps are the power of 2. In each step, the runtimes roughly stay the same.

Such a stepped pattern is resulted from the method we treat the matrices. As we mentioned above, when the dimension is not a power of two, we fill 0s to the matrix until the dimension of the matrix becomes the closest power of 2. So the starting matrix's dimension is 2^k for some constant k. According to the recursion, the new matrix's dimension becomes 2^{k-1}, and so on and so forth. So actually, the valid cross-over point is always some power of 2. When the set cross-over point is not a power of 2, it actually works like a set cross-over point which is a power of 2. Consequently, for all cross-over points n_0 in the range $[2^m, 2^{m+1})$, it has similar runtimes with runtimes of 2^m. So we can take 2^m as the cross-over point n_0. This also explains why the cut off points occur at the power of two.

Below is the table showing the experimental results.

$Dimension(n)$	$CrossOverPoint$	$Interval$	n_0
32	32	$[32, 64)$	32
64	54	$[32, 64)$	32
105	65	$[64, 128)$	64
128	82	$[64, 128)$	64
207	72	$[64, 128)$	64
256	124	$[64, 128)$	64
512	121	$[64, 128)$	64

The *CrossOverPoint* column is the cross-over points with the minimal runtime from testing. The *Interval* column is the range that contains the minimal cross-over points. n_0 is the cross-over point n_0. For example, when $n = 128$, the minimal runtime in the experiment is associated with the cross-over point of 82, and 82 is in the range $[64, 128)$. According to our above analysis, the cross-over point n_0 for $n = 128$ is 64. For $n <= 64$, setting the cross-over point greater than the dimension is meaningless and there is no further room for the runtime to decrease. Therefore, we conclude that the experimental cross-over point $n_0 = 64$.

Comparisons

The experimental cross-over point is larger than the analytical one. The experimental cross-over point depends on quite a lot of factors, such as the implementation methods, the hardware, memory access speed, cache considerations, hyperthreading considerations. Additionally, we assume the cost of any single arithmetic operation (adding, subtracting, multiplying, or dividing two real numbers) to be 1. However, in reality the cost may be differnt. Different arithmetic operations can take on various weights. There may exist other cost that is assumed to be 0 in analytical approach, but may actually be not 0 when implementing the program.

Other findings

When we compare the runtimes of using the Strassen's algorithm with only using conventional matrix multiplication, we find the plots with dimensions 105 and 207 are very different, the runtimes for conventional matrix multiplication are always smaller than the Strassen's algorithm. Here is the explanation: Take $n = 207$ as an example, when using the Strassen's algorithm, we enrich the matrix to $n = 256$ first, so we are actually multiplying two 256 x 256 matrices. However, for conventional method, we only times two 207 x 207 matrices. Therefore, when $n = 207$, the Strassen's algorithm always take longer time to run.

We also find that when n becomes larger, the Strassen's algorithm tends to have shorter runtimes than the conventional matrix multiplication. This can be proven from the graphs. When n is small like $n = 32$, the red line representing the conventional method is below the blue line (Strassen's). When n becomes larger, there are intersections between two lines. Eventually, when n is large like $n = 512$, the blue line is below the red line for most points, which means the Strassen's algorithm runs faster than the conventional

method. In general, for large n, the Strassen's algorithm tends to perform better than the conventional matrix multiplication.