ProgSet 2

CS 124: Data Structures and Algorithms

Due: Wednesday, March 27, 2024 **Denny Cao and Ossimi Ziv**

---

§1 Quantitative Results

§1.1 Analytical Crossover Calculation

The crossover point is the point at which `strassen` becomes faster than the naive algorithm. We can calculate the crossover point by setting the two algorithms equal to each other and solving for n.

The runtime of `standard` is

$$T(n) = n^2(2n-1)$$

assuming that all arithmetic operations have a cost of 1 by Task 1. This is because, for each of the resulting n^2 numbers in the resulting matrix, there are a total of n multiplications and $n-1$ additions.

For `strassen`, we only run "one layer" of the algorithm, with the subproblems using `standard` in order to calculate the resulting matrix. There are two cases:

1. n is a power of 2. This means that the runtime of `strassen` is

   $$T'(n) = 7T(n/2) + 18(n/2)^2$$

   as there are 7 subproblems and 18 matrix additions ($(n/2)^2$ elements in each) in the algorithm, since with powers of 2, we can continuously halve the matrix and the subproblems will remain powers of 2 (even).

   We can now set the two algorithms equal to each other and solve for n:

   $$\begin{aligned} 2n^3 - n^2 &= 7(2(n/2)^3 - (n/2)^2) + 18(n/2)^2 \\ &= \frac{7}{4}n^3 - \frac{7}{4}n^2 + \frac{18}{4}n^2 \\ 0 &= -\frac{1}{4}n^3 + \frac{15}{4}n^2 \\ &= -\frac{1}{4}n^2(n-15) \\ n &= 15 \end{aligned}$$

   We can see that the crossover point is $n_0 = 15$.

2. n is not a power of 2. This means that the runtime of `strassen` is

   $$T'(n) = 7T((n+1)/2) + 18((n+1)/2)^2$$

   This is because the algorithm will pad the matrix by adding a column and row of zeros to the matrix, making the input size for subproblems $(n+1)/2$.

We can now set the two algorithms equal to each other and solve for n:

$$\begin{aligned} 2n^3 - n^2 &= 7(2((n+1)/2)^3 - ((n+1)/2)^2) + 18((n+1)/2)^2 \\ 0 &= \frac{7}{4}(n+1)^3 + \frac{11}{4}(n+1)^2 - 2n^3 + n^2 \\ &= \frac{7}{4}\left(n^3 + 3n^2 + 3n + 1\right) + \frac{11}{4}\left(n^2 + 2n + 1\right) - 2n^3 + n^2 \\ &= -\frac{1}{4}n^3 + 9n^2 + \frac{43}{4}n + \frac{18}{4} \\ n &= 37.17 \end{aligned}$$

Thus, the crossover point is around $n_0 = 37$.

We combine the two cases to get the crossover point for all n:

- For $n < 15$, `standard` is faster.
- For $15 \leq n \leq 37$, it is unclear which algorithm is faster.
- For $n > 37$, `strassen` is faster.

Thus, the theoretical crossover point is $n_0 = 37$.

§1.2 Empirical Crossover

We obtain the empirical crossover point by running the two algorithms on random matrices of size n with entires 0 and 1 and timing them. We then plot the results and find the point at which, for all n greater than the crossover point, `strassen` is faster. A subset of the results are shown below, taking the average of 5 runs for each matrix size between 1 and 50.

Matrix Size (n)	Average Time Strassen (ms)	Average Time Standard (ms)	Matrix Size (n)	Average Time Strassen (ms)	Average Time Standard (ms)
1	0.04005432	0.01764297	26	3.75418663	4.20546532
2	0.25367737	0.04434586	27	4.70714569	4.67915535
3	2.44951248	0.24557114	28	4.69846725	5.22465706
4	1.03759766	0.45967102	**29**	**5.78403473**	**5.77650070**
5	2.76184082	0.08940697	30	5.74755669	6.42280579
6	0.08349419	0.05903244	31	6.94327354	7.02953339
7	0.19207001	0.09231567	32	6.93907738	7.74512291
8	0.14777184	0.13208389	33	8.31937789	8.45537186
9	0.29582977	0.18420219	34	8.70699883	9.58261489
10	0.26364326	0.24600029	35	9.94524956	10.25118828
11	0.44384003	0.32444000	36	9.80730057	10.96653938
12	**0.41499138**	**0.42495727**	37	11.51275635	12.00428009
13	0.70180892	0.57215691	38	11.48490906	12.88061142
14	0.67152977	0.70323944	39	13.39626312	13.96603584
15	0.98776817	0.87027550	40	13.42787743	15.08932114
16	0.96721649	1.02620125	41	15.63568115	16.28928185
17	1.36899948	1.19962692	42	15.51976204	17.43607521
18	1.30710602	1.41773224	43	17.73638725	18.69397163
19	1.76682472	1.64866447	44	17.71345139	19.96340752
20	1.73182487	1.92041397	45	20.18704414	21.47617340
21	2.31013298	2.22172737	46	20.11113167	22.76115417
22	2.32915878	2.55317688	47	22.77207374	24.14793968
23	3.03268432	2.95033455	48	22.81498909	25.85563660
24	3.01985741	3.30181122	49	25.80103874	27.45056152
25	3.78847122	3.72204781	50	27.79173851	29.60662842

Figure 1: Average runtimes of `strassen` and `standard` for matrix sizes n

We observe that, for $n > 29$, `strassen` is faster than `standard`. Thus, the empirical crossover point is $n_0 = 30$.

We also observe that, for $n < 12$, `standard` is faster than `strassen`, and for $12 \leq n \leq 29$, it is unclear which algorithm is faster.

§2 Discussion

§2.1 Implementation

§2.1.1 Padding: Handling Odd Sized Matrices

An interesting difficulty we encountered was how to efficiently implement padding into the algorithm. The initially intuitive solution we had was to pre-process the matrix with padding up to the nearest power of 2, and then post-process the result by only returning up to the original size after `strassen` algorithm had finished running to remove the pads. However, we wanted (yet struggled) to find an implementation that incorporated all the padding into the single `Strassen` function. The various calls of recursion made it difficult to keep track of when un-padding should be done, as defining it within the function would cause issues with recursive calls.

Our next idea was to incorporate the padding as part of the `split` function call. We add a row and column of 0s for padding if the input matrix x has an odd size, which will result in an even sized matrix, and then call `split` again to split the matrix evenly. We now prove that this is correct.

Claim 2.1 — The result of multiplying the padded matrices is the same as the result of multiplying the original matrices.

Proof. Let A and B be the original matrices, and A' and B' be the padded matrices. We have that

$$A' = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}, \quad B' = \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix}$$

We can now multiply the matrices:

$$\begin{aligned} A'B' &= \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} AB & 0 \\ 0 & 0 \end{bmatrix} \end{aligned}$$

We can see that, after removing the padding, the result is the same as the result of multiplying the original matrices, and the proof is complete. □

It was not necessarily unexpected, but timing the algorithms took a significant number of hours when iterating on the larger matrix sizes. This made it important to manage when edits would be made as to not interfere with the code while the program was running as then we would have to re-run to re-time it with the modifications.

§2.2 Optimizations

A small optimization we decided to add after preliminary testing was implementing the `winograd` form of the algorithm, maintaining the asymptotic runtime but reducing the number of additions/subtractions from 18 to 15. Mathematically, this changes the constant number of operations performed, and hypothetically reduces the cross-over point from 37 to 34

$$2n^3 - n^2 = 7T\left(\frac{n}{2}\right) + \mathbf{15}\left(\frac{n}{2}\right)^2$$

$$2n^3 - n^2 = 7(2(\frac{n}{2})^3 - (\frac{n}{2})^2) + 15(\frac{n}{2})^2$$

Reducing the calculation gives $n_0 = 12$ for powers of two. Similarly, accounting for the padding on numbers not power of two with the updated number of operations, we have:

$$2n^3 - n^2 = 7(2(\frac{n+1}{2})^3 - (\frac{n+1}{2})^2) + \mathbf{15}(\frac{n+1}{2})^2$$

Which reduces to $n_0 \approx 34$, a minor reduction estimate.

NEED DATA TO CHECK IF WINOGRAD IS ACTUALLY FASTER

§2.3 Matrix Choice

We chose to multiply all matrices between 1-50: powers of two and non-powers of two. The matrix choice considerably varies the number of operations that need to be conducted as for non-powers of two, padding and un-padding has to applied, adding extra cost and changing the size of the matrix that is operated on. Thus, to ensure that our n_0 value was sound across all variations of the operations and we'd get general results, we decided to not limit which types of matrices we tested.