Lab 2: Matrix Multiplication

Monday, September 10

CompSci 531, Fall 2018

Outline

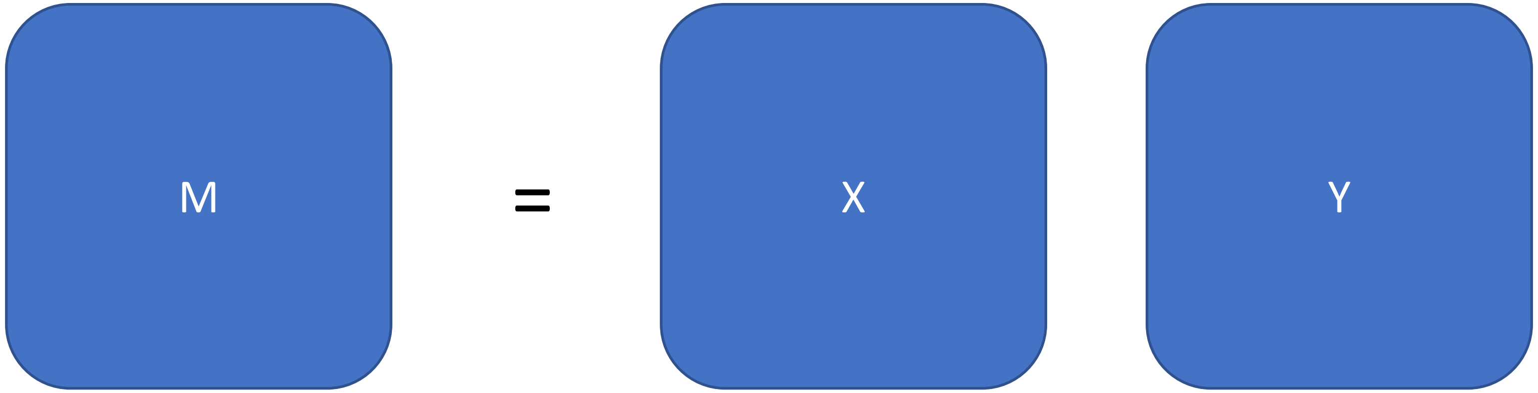
- Review Strassen's Algorithm
- Detour – Matrix Squaring Divide and Conquer
- Implementing Strassen's Algorithm

Review Strassen’s Algorithm

- Divide and Conquer at a High Level:
  - **Check** for your base case.
  - **Divide** your problem into multiple identical subproblems.
  - **Recursively** solve each subproblem.
  - **Merge** the solutions to your subproblems.

Review Strassen's Algorithm

- Recall the matrix multiplication problem: we have two n by n matrices X and Y, and we want to compute M = XY



By definition: $M_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$. So that gives us an O($n^3$) iterative algorithm for free. What about recursion?

Review Strassen's Algorithm

- Break each matrix up into four (n/2) by (n/2) sub-matrices as follows:

$$\begin{bmatrix} M_A & M_B \\ M_C & M_D \end{bmatrix} = \begin{bmatrix} X_A & X_B \\ X_C & X_D \end{bmatrix} \begin{bmatrix} Y_A & Y_B \\ Y_C & Y_D \end{bmatrix}$$

- Note
  - $M_A = X_A Y_A + X_B Y_C$
  - $M_B = X_A Y_B + X_B Y_D$
  - $M_C = X_C Y_A + X_D Y_C$
  - $M_D = X_C Y_B + X_D Y_D$

There are 8 recursive subproblems to solve!

Review Strassen's Algorithm

- Yields the recurrence $T(n) = 8T(n/2) + O(n^2)$. So $T(n) = O(n^3)$. No better than the iterative algorithm!
- Strassen's insight: The run time is dominated by the branching factor of 8. What if we could reduce that? Let:

$S_1 = (X_B - X_D)(Y_C + Y_D)$
$S_2 = (X_A + X_D)(Y_A + Y_D)$
$S_3 = (X_A - X_C)(Y_A + Y_B)$
$S_4 = (X_A + X_B)(Y_D)$
$S_5 = (X_A)(Y_B - Y_D)$
$S_6 = (X_D)(Y_C - Y_A)$
$S_7 = (X_C + X_D)(Y_A)$

$\Rightarrow$

$M_A = S_1 + S_2 - S_4 + S_6$
$M_B = S_4 + S_5$
$M_C = S_6 + S_7$
$M_D = S_2 + S_3 + S_5 - S_7$

Review Strassen's Algorithm

- We went from 8 matrix multiplications (recursive calls) and 4 matrix additions (merge steps) to 7 matrix multiplications and 18 matrix additions.
- $T(n) = 7\ T(n/2) + O(n^2)$. So $T(n) = O(n^{\lg(7)}) \sim O(n^{2.81})$.
- Does this matter? We'll test that out in a minute.

Outline

- ~~Review Strassen's Algorithm~~

- Detour – Matrix Squaring Divide and Conquer

- Implementing Strassen's Algorithm

Detour – Matrix Squaring Divide and Conquer

- Work on the following problem in groups. Let A be an n x n matrix. We want to compute AA, the square of A.

1. Show that just **five** multiplications are sufficient to compute the square of a 2 x 2 matrix.
2. Suppose we run Strassen's algorithm but use 5 multiplications per recursive step instead of 7 using our observation from part 1. If this worked, what would be the asymptotic runtime?
3. Why does this **not** work?
4. *If you have time, try to give a reduction to prove that an $O(n^c)$ time algorithm (for $2 \leq c < 3$) for matrix squaring implies an $O(n^c)$ time algorithm for matrix multiplication.

Detour – Matrix Squaring Divide and Conquer

1. Note that : $\begin{bmatrix} a & b \\ c & e \end{bmatrix}^2 = \begin{bmatrix} a^2 + bc & b(a+d) \\ c(a+d) & cb + d^2 \end{bmatrix}$
2. The runtime would be $\log_2 5 \approx 2.32$
3. Not all of the subproblems are matrix squaring problems! (Plus, matrix multiplication, unlike scalar, is not commutative)
4. Suppose we have an O(n^c) algorithm for matrix squaring, and we want an O(n^c) algorithm for matrix multiplication (say of n x n matrices A and B). Define the 2n x 2n matrix $M = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}$. Then: $M^2 = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}$, so we can read off the answer to AB.

- ~~Review Strassen's Algorithm~~

- ~~Detour – Matrix Squaring Divide and Conquer~~

- Implementing Strassen's Algorithm
  - Does $n^{2.81}$ really matter much compared to n^3?

Implementing Strassen’s Algorithm

- Break into groups of ~ 3.
- Code up 3 simple matrix multiplication algorithms:
  - Iterative algorithm by definition
  - Naïve recursive algorithm
  - Strassen’s recursive algorithm
- To test, generate random 32x32, 64x64, 128x128, and 256x256 matrices (in whatever way is convenient, use smallish integers).
- Time all of your algorithms, and try to explain your results.
- (ProTip – you may be able to improve your recursive algorithms by using the iterative algorithm once you get to small matrices, maybe 8x8 or 16x16).

Implementing Strassen's Algorithm

Run times in milliseconds				
n	Iterative	Recursive	Strassen	R Library
32	67	78	149	0
64	552	552	342	1
128	4155	4101	2444	2
256	31730	34315	20071	15

Conclusion

- Many recursive divide and conquer algorithms can be sped up if you can reduce the number of recursive calls, maybe at the expense of a larger merge step.
  - (But this improvement might not be large until you work with larger problem sizes)
- There are tricks that matter in practice but not in theory. Examples:
  - In many languages, basic operations like matrix multiplication, summing vectors, etc., are *heavily* optimized, and you shouldn’t reinvent the wheel (outside of this exercise).
  - Combining recursive and iterative methods rather than recursing all the way to the trivial base case often helps.