CH-231-A

Algorithms and Data Structures

ADS

Lecture 12

Dr. Kinga Lipskoch

Spring 2022

Matrix Multiplication: Strassen's Idea (1)

Strassen's idea:
Multiply matrices with 7 multiplications and 18 additions.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) & r &= P_5 + P_4 - P_2 + P_6 \\
P_2 &= (a + b) \cdot h & s &= P_1 + P_2 \\
P_3 &= (c + d) \cdot e & t &= P_3 + P_4 \\
P_4 &= d \cdot (g - e) & u &= P_5 + P_1 - P_3 - P_7 \\
P_5 &= (a + d) \cdot (e + h) & & \\
P_6 &= (b - d) \cdot (g + h) & & \\
P_7 &= (a - c) \cdot (e + f) & &
\end{aligned}
$$

$$
\begin{array}{ccccc}
\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] & = & \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] & \cdot & \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right] \\
C & = & A & \cdot & B
\end{array}
$$

Matrix Multiplication: Strassen's Idea (2)

Strassen's idea:

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
\qquad
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$

$$
\underset{C}{\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right]} = \underset{A}{\left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right]} \cdot \underset{B}{\left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]}
$$

Matrix Multiplication: Strassen's Algorithm

Strassen's algorithm:

1. Divide:
   Partition A and B into $(n/2) \times (n/2)$ submatrices.
   Form terms to be multiplied using $+$ and $-$.
2. Conquer:
   Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
3. Combine:
   Form C using $+$ and $-$ on $(n/2) \times (n/2)$ submatrices.

Matrix Multiplication: Complexity

Complexity of Strassen's algorithm:
Recurrence:

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$n^{log_b a} = n^{log_2 7} = n^{2.81}$
Case 1: $T(n) = \Theta(n^{\lg 7})$

2.81 may not seem much smaller than 3, but the difference is in the exponent, therefore the impact on running time is significant.

Strassen's algorithm beats the standard algorithm for $n \geq 32$ or so.

Matrix Multiplication: More

Best known algorithm:
Latest improvement in 2014 in the following publication:
Francois LeGall, Powers of Tensors and Fast Matrix Multiplication, 30 Jan 2014

$T(n) = O(n^{2.3728639})$

- Only of theoretical interest.
- Most approaches that are faster than Strassen's are not used in practice.
- They are only faster for very large n.
- One cannot get better than $O(n^2)$, cf. Case 3.

Intermediate Conclusion

- Definitions
- First example of an algorithm – Insertion Sort
- Asymptotic analysis
- First powerful concept – Divide & Conquer
- Solve recurrences for analysis

Recall: Sorting Problem

- Input:
  - Sequence $< a_1, a_2, ..., a_n >$ of numbers
- Output:
  - Permutation $< a'_1, a'_2, ..., a'_n >$
  - Such that $a'_1 \leq a'_2 \leq ... \leq a'_n$

Recall: Insertion & Merge Sort

Time complexity:

	Insertion Sort	Merge Sort
Best case	$\Theta(n)$	$\Theta(n \lg n)$
Average case	$\Theta(n^2)$	$\Theta(n \lg n)$
Worst case	$\Theta(n^2)$	$\Theta(n \lg n)$

Visualizations:

http://www.sorting-algorithms.com/insertion-sort

http://www.sorting-algorithms.com/merge-sort

What about storage space complexity?

In-situ Sorting

- Definition:
  - In-situ algorithms refer to algorithms that operate with $\Theta(1)$ memory
- In-situ sorting:
  - Sorting algorithms that need only a constant number of additional storings
- Insertion Sort:
  - In-situ sorting
- Merge Sort:
  - Not in-situ sorting