CS 583
Evgenios Kornaropoulos

Lecture 1: Introduction

What is an Algorithm? A set of well-defined rules for solving a computational problem. It is typically accompanied by a proof that it is correct as well as a bound on the amount of time it takes to terminate with an answer

Why study Algorithms? Essential for doing rigorous work in any branch of Computer Science. Algorithms are everywhere! For example:

Computer Networks: Routing protocols in communication networks are based on classical shortest path algorithms.

Cryptography: Public key cryptography relies on number-theoretic algorithms

Computer Graphics: Computational primitives supplied by geometric algorithms

Databases: Database indices rely on balanced search data structures

Computational Biology: Use dynammic programming algorithms for genome similarity

Machine Learning: Clustering Algorithms is at the core of unsupervised learning

Home > News

New Algorithm Makes CPUs 15 Times Faster Than GPUs in Some AI Work

By Anton Shilov last updated April 10, 2021

CPUs can beat GPUs in some AI workloads

Comments (1)

(Image credit: Intel)

GPUs are known for being significantly better than most CPUs when it comes to AI deep neural networks (DNNs) training simply because they have more execution units (or cores). But a new algorithm proposed by computer scientists from Rice University is claimed to actually flip the tables and make CPUs a whopping 15 times faster than some leading-edge GPUs.

The most complex compute challenges are usually solved using brute force methods, like either throwing more hardware at them or inventing special-purpose hardware that can solve the task. DNN training is without any doubt among the most compute-intensive workloads nowadays, so if programmers want maximum training performance, they use GPUs for their workloads. This happens to a large degree because it is easier to achieve high performance using compute GPUs as most algorithms are based on matrix multiplications.

CPU with Randomized Hashing Algorithms, Data Structures, and Asynchronous Parallelism

vs.

NVIDIA Tesla V100 Volta 32GB GPU with Tensorflow (cost ≈ $4.700)

=

CPU with "smart" algorithms is 10x faster

Paper: "SLIDE: In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems" in MLSys 2020

Warmup: Integer Multiplication

! Important: Distinguish between the description of the problem being solved and the method of solution.

In this class, we will introduce the computational problem (inputs and desired output), and then come up with one or more algorithms that solve the problem.

Problem

Input: Two n-digit non-negative integers x and y

Output: The product $x \cdot y$

Assess the performance by counting the number of primitive operations it performs

1. Adding two single digit numbers
2. Multiplying two single digit numbers
3. Adding a zero to the beginning/end of a number

Grade-School Algorithm

$$\begin{array}{r} 5678 \\ \times\ 1234 \\ \hline 22712 \\ 17034 \\ 11356 \\ 5678 \\ \hline 7006652 \end{array}$$

partial product

n rows

Analysis of the Number of Primitive Operations:

- For the first partial product, we multiplied 4 with each of the digits 5, 6, 7, and 8 of the first number. These are four primitive operations, or more generally n. Each carry digit might force us to add a single digit (the carry) to a double digit number (the product of two digits). Therefore, we need two additions per carry.
- Each partial product involves n multiplications (one per digit) and at most $2n$ additions (at most two per carry)
- We have n partial products, one per digit of the second number.
- To compute all partial products, we need $n \cdot 3n = 3n^2$ primitive operations

Overall, the amount of work grows quadratically with the number of digits

Can we do better?

Let's first apply a sequence of steps and later we will explain what the algorithm is and why it is correct

$$\begin{array}{r} \overbrace{56}^{a}\overbrace{78}^{b} \\ \times\ \underbrace{12}_{c}\underbrace{34}_{d} \\ \hline \end{array}$$

Step 1: Compute $a \cdot c = 56 \cdot 12 = 672$

Step 2: Compute $b \cdot d = 78 \cdot 34 = 2652$

Step 3: Compute $(a+b) \cdot (c+d) = 134 \cdot 46 = 6164$

Step 4: Subtract the results from Step 1, 2 from step 3, $6164 - 672 - 2652 = 2840$

Step 5: Add two trailing zeros to Step 4, four trailing zeros to step 1 and add both to Step 2

$10^4 \cdot 672 + 10^2 \cdot 2840 + 2652 = 7006652$ which is the same!

Before explaining the above algorithm called Karatsuba Algorithm let's first discuss a simpler version.

A Recursive Approach

A number x with an even number n of digits can be expressed with two $n/2$-digit numbers, i.e., the first half and the second half

$$x = 10^{n/2} \cdot a + b \quad , \text{ similarly } \quad y = 10^{n/2} c + d$$

Therefore

$$\begin{aligned} x \cdot y &= (10^{n/2} \cdot a + b) \cdot (10^{n/2} c + d) \\ &= 10^n \cdot (a \cdot c) + 10^{n/2} \cdot (a \cdot d + b \cdot c) + b \cdot d \end{aligned}$$

multiplication of $n/2$-digit numbers

We can recurse to compute the $n/2$-digit multiplications

$$5678 \cdot 1234 = 10^4 \cdot (\underbrace{56 \cdot 12}) + 10^2 (\underbrace{56 \cdot 34 + 78 \cdot 12}) + 78 \cdot 34$$

recursion

$$56 \cdot 12 = 10^2 \cdot (5 \cdot 1) + 10 \cdot (5 \cdot 2 + 6 \cdot 1) + 6 \cdot 2 = 500 + 160 + 12 = 672 \qquad \ldots$$

RecIntMult

Input: Two n-digit positive integers x and y
Output: The product $x \cdot y$
Assumption: n is a power of two

---

1. If $n = 1$ then
2. compute $x \cdot y$ in one step and return the result
3. else
4. $a, b \leftarrow$ first and second halves of x
5. $c, d \leftarrow$ first and second halves of y
6. Recursively compute $temp1 \leftarrow a \cdot c$, $temp2 \leftarrow a \cdot d$, $temp3 \leftarrow b \cdot c$
   $temp4 \leftarrow b \cdot d$
7. Compute $10^n \cdot temp1 + 10^{n/2} \cdot (temp2 + temp3) + temp4$ and return the result

Karatsuba Multiplication Algorithm

Optimized recursive algorithm that makes 3 recursive calls instead of 4.

$$x \cdot y = 10^n \cdot (a \cdot c) + 10^{n/2} \cdot (\underbrace{a \cdot d + b \cdot c}_{\text{Term } Ⓐ}) + b \cdot d$$

We don't care about term $a \cdot d$ or $b \cdot c$
we just need their sum!

Can we express $(a\cdot d + b\cdot c)$ using the terms $a\cdot c$, $b\cdot d$, and one more multiplication?

- First observe that a term of the form $(z_1+z_2)\cdot(z_3+z_4)$ can give us two products (i.e., $a\cdot c$ and $b\cdot d$) from four terms (i.e., $z_1, z_2, z_3,$ and z_4). Next, we assign values to z_1, z_2, z_3, z_4
- Both $(a+d)\cdot(c+b)$ and $(a+b)\cdot(c+d)$ can give us the desired terms $a\cdot c$ and $b\cdot d$.

① $(a+d)\cdot(c+b) = a\cdot c + a\cdot b + d\cdot c + d\cdot b \Rightarrow \underbrace{(a\cdot b + d\cdot c)}_{\text{✗ NOT Term A}} = (a+d)\cdot(c+b) - a\cdot c - d\cdot b$

② $(a+b)\cdot(c+d) = a\cdot c + a\cdot d + b\cdot c + b\cdot d \Rightarrow \underbrace{(a\cdot d + b\cdot c)}_{\text{✓ Term A}} = \overset{\#1}{(a+b)\cdot(c+d)} - \overset{\#2}{a\cdot c} - \overset{\#3}{b\cdot d}$

$\Downarrow$

compute $a\cdot d + b\cdot c$ with three multiplications

Karatsuba

Input: Two n-digit positive integers x and y
Output: The product $x\cdot y$
Assumption: n is a power of two

---

1. If $n=1$ then
2. compute $x\cdot y$ in one step and return the result
3. else
4. $a, b \leftarrow$ first and second halves of x
5. $c, d \leftarrow$ first and second halves of y
6. $p \leftarrow (a+b)$ and $q \leftarrow c+d$
7. Recursively compute $temp1 \leftarrow a\cdot c$, $temp2 \leftarrow p\cdot q$, $temp3 \leftarrow b\cdot d$
8. Compute $10^n\cdot temp1 + 10^{n/2}\cdot(temp2 - temp1 - temp3) + temp3$ and return the result

Karatsuba Algorithm makes only three recursive calls. So it must be faster than the simple Recursive Algorithm. In a later class we will see the tools to calculate the gains, i.e., Divide-and-Conquer Section.

Mergesort Algorithm

Problem: Sorting

Input: An array of n numbers in arbitrary order
Output: An array of the same numbers, sorted from smallest to largest

Input

5	4	1	8	7	2	6	3

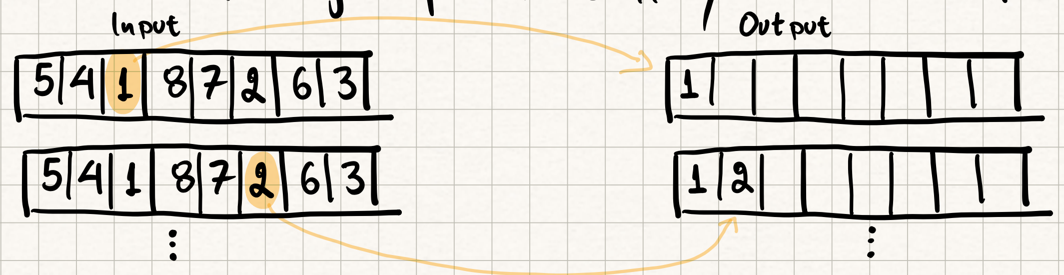
Output

1	2	3	4	5	6	7	8

A simple Sorting first.

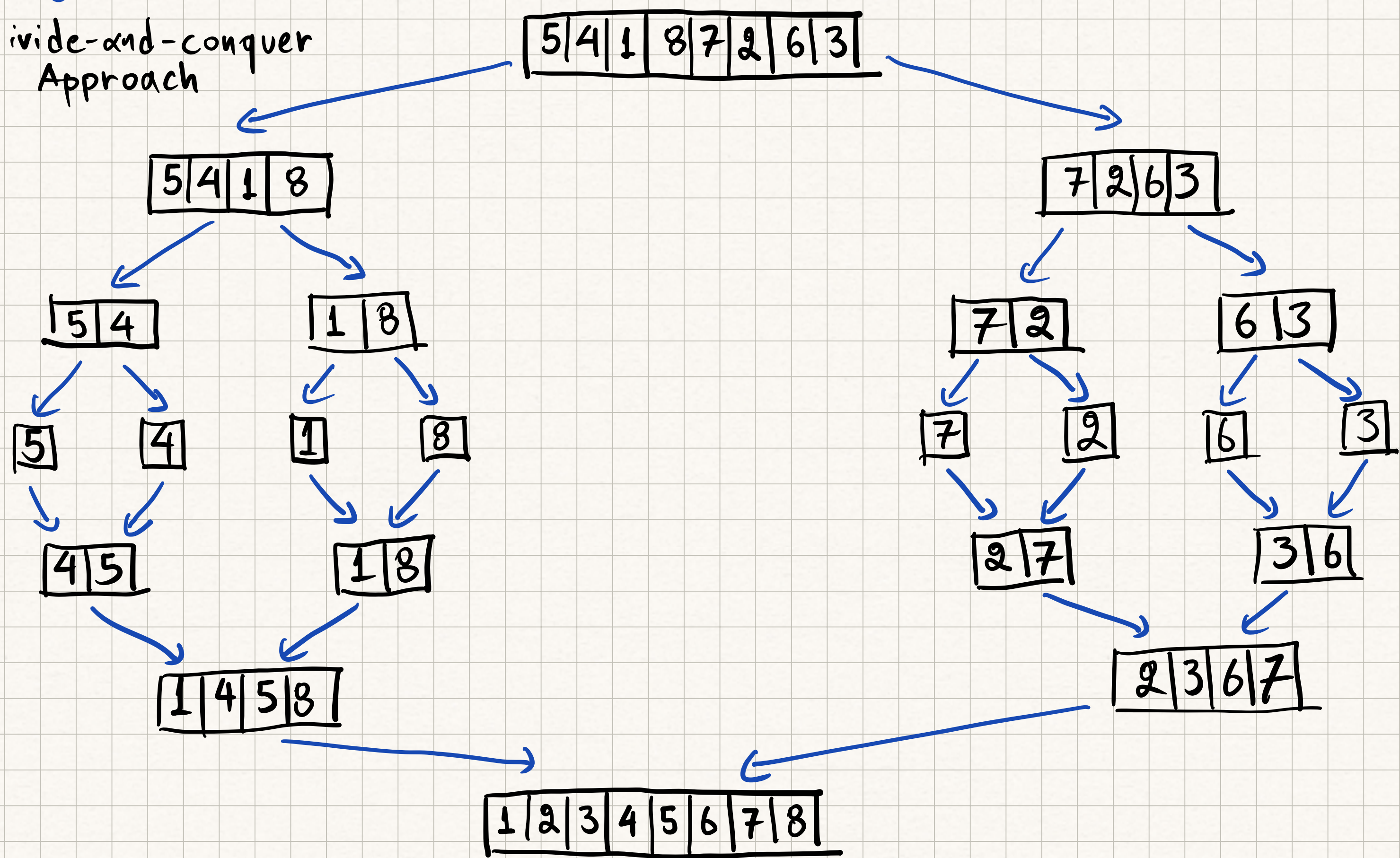
SelectionSort: Scan through input to identify minimum and copy it to output.
Scan through input to identify second minimum...!

\# of operations on arrays of length n scales with n^2



Mergesort Example

Divide-and-conquer Approach



MergeSort

Input: Array A of distinct integers
Output: Array with the same integers, sorted from smallest to largest

```
1. //Ignoring base case, no recursion needed
2. C ← recursively sort first half of A
3. D ← recursively sort second half of A
4. return Merge(C,D)
```

Merge

Input: Sorted arrays C and D (length n/2 each)
Output: Sorted array B (length n)
Simplifying Assumption: n is even

```
1. i ← 1 , j ← 1
2. for k = 1 to n do
3.    if C[i] < D[j] then
4.        B[k] ← C[i]
5.        i ← i+1
6.    else
7.        B[k] ← D[j]
8.        j ← j+1
```

Running Time of Merge

- Suppose the two sorted arrays have length $\ell/2$ each.
- Line 1 performs two initialization steps (two operations).
- The for loop in Line 2 executes ℓ times.
- Each iteration performs a comparison in Line 3, an assignment to B in Lines 4 or 7, an increase of counters i/j in Lines 5 or 8, and an increase of k (total four operat.)
- A sloppy (but convenient) inequality is $4\ell+2$ operations are less than $4\ell+2\ell=6\ell$.
- Thus, 6ℓ is an <u>upper bound</u> on the number of operations performed by Merge.

Running Time of MergeSort

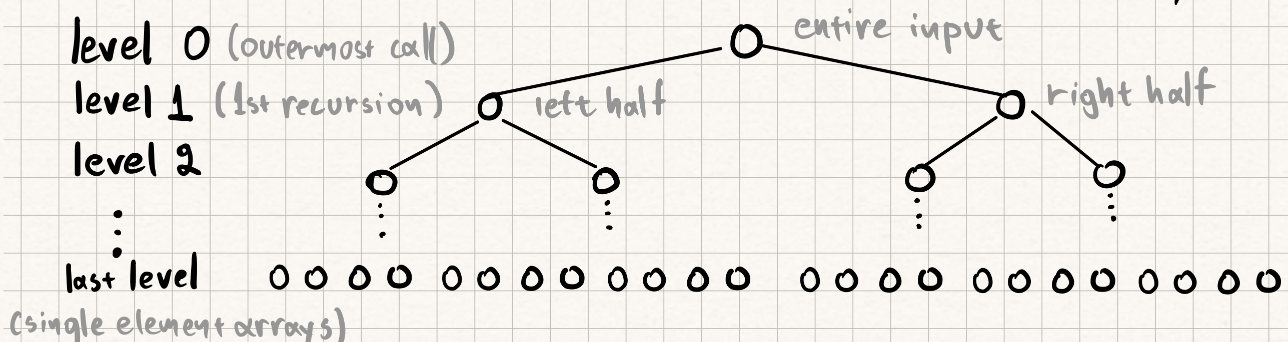
* Tension between competing forces:
Explosion of the number of subproblems through recursion
Shrinking inputs that each subproblem is responsible for

Theorem 1.2. For every input array of length $n \geq 1$, the MergeSort algorithm performs at most:

$$6n\log_2 n + 6n$$

operations, where $\log_2$ is the base-2 logarithm.

Proof: We use a recursion tree where nodes correspond to recursive calls and children of a node correspond to recursive calls made by that node.



Quiz 1 : How many levels does the recursion tree have, as a function of the length n of the input array?

a) A constant number
b) $\log_2 n$
c) $\sqrt{n}$
d) n

Quiz 2 : Fill in the blanks: at each level $j = 0, 1, 2, \ldots$ of the recursion tree, there are [blank] subproblems, each operating on a subarray of length [blank].

a) 2^j and 2^j
b) $n/2^j$ and $n/2^j$
c) 2^j and $n/2^j$
d) $n/2^j$ and 2^j

Proceed level-by-level, suppose we analyze level j of the recursion tree.
MergeSort makes two recursive calls (ignore their cost for now) and invokes the Merge subroutine. From previous analysis, Merge performs at most 6ℓ operations on ℓ-element array.
Let ℓ_m be the number of elements in subproblem m.

The total work in level j (ignoring the cost of recursion) is

$$6\ell_1 + 6\ell_2 + \dots + 6\ell_{2^j} = 6(\ell_1 + \ell_2 + \dots + \ell_{2^j}) = 6n$$

Recall from Quiz 2 that we have 2^j subproblems at level j of the recursion tree.

Thus, the work per level-j subproblem is $6n/2^j$.

Total work by level-j of recursion tree

$$\underbrace{\underbrace{\text{number of level-}j\text{ subproblems}}_{2^j} \times \underbrace{\text{work per level-}j\text{ subproblem}}_{6n/2^j}} = 6n$$

* The upper-bound on the work done at level j is independent of j.
Thus, this is the perfect balance between the tension from doubling the number of subprolems at every level and halving the amount of work per subproblem.

For the final piece of the analysis, we want to compute the number of operations across all levels of the recursion tree. The recursion tree has $\log_2 n + 1$ levels (from 0 to $\log_2 n$, inclusive). Thus, the total number of operations are:

$$\underbrace{\text{number of levels}}_{=\log_2 n + 1} \times \underbrace{\text{work per level}}_{< 6n} < 6n\log_2 n + 6n.$$

∎