Grade School Revisited: How To Multiply Two Numbers
Lecture 22, November 6, 2008

Gauss’ Complex Puzzle

Remember how to multiply two complex numbers $a + bi$ and $c + di$?

$$(a+bi)(c+di) = [ac – bd] + [ad + bc] i$$

Input: $a, b, c, d$
Output: $ac-bd, ad+bc$

If multiplying two real numbers costs $1 and adding them costs a penny, what is the cheapest way to obtain the output from the input?

Can you do better than $4.02$?
Gauss’ $3.05$ Method

Input: a,b,c,d
Output: ac-bd, ad+bc

\[
\begin{align*}
  c \quad X_1 &= a + b \\
  c \quad X_2 &= c + d \\
  $ \quad X_3 &= X_1 X_2 = ac + ad + bc + bd \\
  $ \quad X_4 &= ac \\
  $ \quad X_5 &= bd \\
  c \quad X_6 &= X_4 - X_5 = ac - bd \\
  cc \quad X_7 &= X_3 - X_4 - X_5 = bc + ad \\
\end{align*}
\]

The Gauss optimization saves one multiplication out of four. It requires 25% less work.

Time complexity of grade school addition

\[
T(n) = \text{amount of time grade school addition uses to add two } n\text{-bit numbers}
\]

We saw that \(T(n)\) was linear
\[
T(n) = \Theta(n)
\]

Time complexity of grade school multiplication

\[
T(n) = \text{The amount of time grade school multiplication uses to add two } n\text{-bit numbers}
\]

We saw that \(T(n)\) was quadratic
\[
T(n) = \Theta(n^2)
\]
Any addition algorithm takes $\Omega(n)$ time

Claim: Any algorithm for addition must read all of the input bits

Proof: Suppose there is a mystery algorithm $A$ that does not examine each bit

Give $A$ a pair of numbers. There must be some unexamined bit position $i$ in one of the numbers

Is there a sub-linear time method for addition?

Any addition algorithm takes $\Omega(n)$ time

If $A$ is not correct on the inputs, we found a bug

If $A$ is correct, flip the bit at position $i$ and give $A$ the new pair of numbers. $A$ gives the same answer as before, which is now wrong.
Grade school addition can’t be improved upon by more than a constant factor.

Can we even break the quadratic time barrier?
In other words, can we do something very different than grade school multiplication?

Grade School Addition: Θ(n) time. Furthermore, it is optimal.

Grade School Multiplication: Θ(n^2) time

Is there a clever algorithm to multiply two numbers in linear time?

Despite years of research, no one knows! If you resolve this question, Carnegie Mellon will give you a PhD!

**Divide And Conquer**

An approach to faster algorithms:
- **DIVIDE** a problem into smaller subproblems
- **CONQUER** them recursively
- **GLUE** the answers together so as to obtain the answer to the larger problem
Multiplication of 2 \( n \)-bit numbers

\[
\begin{align*}
X &= a \cdot 2^{n/2} + b \\
Y &= c \cdot 2^{n/2} + d \\
X \times Y &= ac \cdot 2^n + (ad + bc) \cdot 2^{n/2} + bd
\end{align*}
\]

\( \text{MULT}(X, Y): \)
- If \(|X| = |Y| = 1\) then return \(XY\)
- else break \(X\) into \(a; b\) and \(Y\) into \(c; d\)
- return \(\text{MULT}(a, c) \cdot 2^n + (\text{MULT}(a, d) + \text{MULT}(b, c)) \cdot 2^{n/2} + \text{MULT}(b,d)\)

Same thing for numbers in decimal!

\[
\begin{align*}
X &= a \cdot 10^{n/2} + b \\
Y &= c \cdot 10^{n/2} + d \\
X \times Y &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd
\end{align*}
\]

Multiplying (Divide & Conquer style)

\[
12345678 \times 21394276
\]

\[
1234\times 2139 \quad 1234\times 4276 \
5678\times 2139 \quad 5678\times 4276
\]

\[
12*21 \quad 12*39 \quad 34*21 \quad 34*39
\]

\[
1*2 \quad 1*1 \quad 2*2 \quad 2*1
\]

\[
2 \quad 1 \quad 4 \quad 2
\]

Hence: \(12^*21 = 2 \times 10^2 + (1 + 4)10^1 + 2 = 252\)

\[
\begin{align*}
X &= \quad a & b \\
Y &= \quad c & d \\
X \times Y &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd
\end{align*}
\]
Multiplying (Divide & Conquer style)

12345678 * 21394276
1234*2139  1234*4276  5678*2139  5678*4276
2521468971411326
*10^4 + *10^2 + *10^2 + *1 = 2639526

\[ X = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \]
\[ Y = \begin{pmatrix} & \\ \end{pmatrix} \]
\[ X \times Y = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \]

Divide, Conquer, and Glue

\[ \text{MULT}(X, Y) \]

MULT(X,Y):

if \(|X| = |Y| = 1\)
then return \(XY\),
else...

\[ \text{MULT}(X, Y) \]

12345678 * 21394276

2639526 52765846 12145242 24279128
*10^8 + *10^4 + *10^4 + *1

= 264126842539128

\[ X = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \]
\[ Y = \begin{pmatrix} & \\ \end{pmatrix} \]
\[ X \times Y = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \]
Divide, Conquer, and Glue

\[
\text{MULT}(X,Y): \quad X=a;b \quad Y=c;d
\]

- \[
\text{Mult}(a,c) \]
- \[
\text{Mult}(a,d) \]
- \[
\text{Mult}(b,c) \]
- \[
\text{Mult}(b,d) \]
Divide, Conquer, and Glue

\[
\text{MULT}(X, Y):
\]

\[
X = a; b \quad Y = c; d
\]

\[
\begin{align*}
ac & \\
ad & \\
\text{Mult}(b, c) & \\
\text{Mult}(b, d) & \\
\end{align*}
\]
Divide, Conquer, and Glue

MULT(X,Y):

X = a; b  Y = c; d

XY = ac2^n + (ad+bc)2^{n/2} + bd

Divide, Conquer, and Glue

MULT(X,Y):

X = a; b  Y = c; d

XY = ac2^n + (ad+bc)2^{n/2} + bd

Time required by MUL{T}T

T(n) = time taken by MULT on two n-bit numbers

What is T(n)? What is its growth rate?

Big Question: Is it Θ(n^2)?

T(n) = 4 T(n/2) + (k'n + k'')

conquering time

divide and glue

Recurrence Relation

T(1) = k for some constant k
T(n) = 4 T(n/2) + k' + k'' for constants k' and k''

MULT(X,Y):

If |X| = |Y| = 1 then return XY
else break X into a;b and Y into c;d

return MULT(a,c) 2^n + (MULT(a,d)
+ MULT(b,c)) 2^{n/2} + MULT(b,d)

Recurrence Relation

T(1) = 1
T(n) = 4 T(n/2) + n

MULT(X,Y):

If |X| = |Y| = 1 then return XY
else break X into a;b and Y into c;d

return MULT(a,c) 2^n + (MULT(a,d)
+ MULT(b,c)) 2^{n/2} + MULT(b,d)
Technique: Labeled Tree Representation

\[
T(n) = n + 4 T(n/2)
\]

\[
T(n) = T(n/2) + T(n/2) + T(n/2) + T(n/2)
\]

\[
T(1) = 1
\]

\[
T(n) = 4 T(n/2) + (k'n + k'')
\]
Divide and Conquer MUL T: $\Theta(n^2)$ time
Grade School Multiplication: $\Theta(n^2)$ time
MULT revisited

MULT(X,Y):
If |X| = |Y| = 1 then return XY
else break X into a;b and Y into c;d
return MULT(a,c) 2^n + (MULT(a,d)
 + MULT(b,c)) 2^{n/2} + MULT(b,d)

MULT calls itself 4 times. Can you see a way
to reduce the number of calls?

Gauss’ optimization

Input: a,b,c,d
Output: ac-bd, ad+bc

c X_1 = a + b
c X_2 = c + d
$ X_3 = X_1 X_2 = ac + ad + bc + bd$
$ X_4 = ac$
$ X_5 = bd$
c X_6 = X_4 - X_5 = ac - bd$
c X_7 = X_3 - X_4 - X_5 = bc + ad

Karatsuba, Anatolii Alexeevich (1937-)

Sometime in the late 1950’s
Karatsuba had formulated
the first algorithm to break
the n^2 barrier!

Gaussified MULT
(Karatsuba 1962)

MULT(X,Y):
If |X| = |Y| = 1 then return XY
else break X into a;b and Y into c;d

\[ e = \text{MULT}(a,c) \]
\[ f = \text{MULT}(b,d) \]
return
\[ e 2^n + (\text{MULT}(a+b,c+d) - e - f) 2^{n/2} + f \]

T(n) = 3 T(n/2) + n
Actually: T(n) = 2 T(n/2) + T(n/2 + 1) + kn
\[ T(n) = \]

\[ n \]

\[ T(n/2) \quad T(n/2) \quad T(n/2) \]

\[ n = T(n) \]

\[ T(n/2) \quad T(n/2) \quad T(n/2) \]

\[ n/2 \quad T(n/4) \quad T(n/4) \quad T(n/4) \]

\[ \log_2(n) \]

\[ 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1 \]

\[ n(1+3/2+(3/2)^2+\ldots+(3/2)^{\log_2 n}) = 3n^{1.58…} - 2n \]
Dramatic Improvement for Large $n$

$$T(n) = 3n^{\log_2 3} - 2n$$

$$= \Theta(n^{\log_2 3})$$

$$= \Theta(n^{1.58\ldots})$$

A huge savings over $\Theta(n^2)$ when $n$ gets large.

## Multiplication Algorithms

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kindergarten</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Grade School</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>$n^{1.58\ldots}$</td>
</tr>
<tr>
<td>Fastest Known</td>
<td>$n \log(n) \log\log(n)$</td>
</tr>
</tbody>
</table>
Here’s What You Need to Know…

- Gauss’s Multiplication Trick
- Proof of Lower bound for addition
- Divide and Conquer
- Solving Recurrences
- Karatsuba Multiplication