Matrix product and Power Algorithms

CSCI 694 Presentation

Satakarni Bommuluri

Theory

GENERAL MATRIX-MATRIX PRODUCT

Theory

- The product of two matrices **A € R** $^{m \times n}$ and **B € R** $^{p \times q}$ is possible, if and only if, n = p.

- The number of columns of first matrix must be equal to number of rows of second matrix.

- This is called conformability.

Algorithm

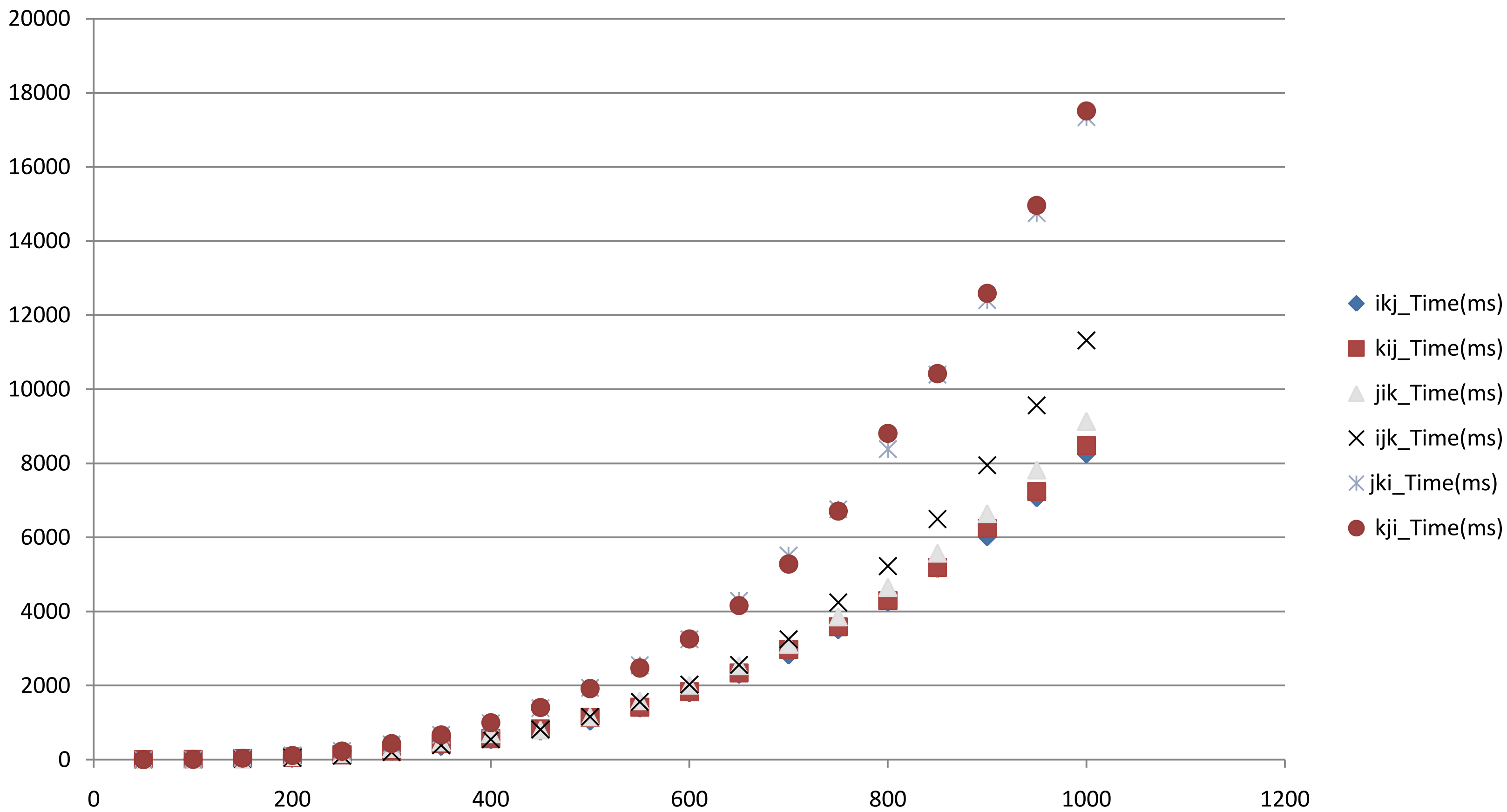
GENERAL MATRIX-MATRIX PRODUCT

(T – method) Algorithm

```
function: C = matmat.ijk(A,B)
m = rows(A); r = cols(A); n = cols(B)
C(1:m, 1:n) = 0
for i = 1:m
for j = 1:n
    for k = 1:r
            C(i,j) = C(i,j) + A(i,k)B(k,j)
    end
end
end
end matmat.ijk
```

Loop Order	Inner Loop	Middle Loop	Inner Loop Data Acess
matmat.ijk()	Dot	Vector * Matrix	A by row, B by column
matmat.ikj()	Dot	Matrix * vector	A by row, B by column
matmat.jik()	saxpy	Row gaxpy	B by row
matmat.jki()	saxpy	Column gaxpy	A by column
matmat.kij()	saxpy	Row outer product	B by row
matmat.kji()	saxpy	Column outer product	A by column

Performance



Processor :Intel® Core(TM)2 Duo p8600 @ 2.40 GHz 2.40 GHz

Compiler: Microsoft® Visual C++ professional (student edition)

Language: ANSI C

Theory

BLAS - 3

?gemm() BLAS

- *Stands for GEneral Mat-Mat multiplication.*
- *Computes a scalar-matrix-matrix product and adds the result to a scalar-matrix product.*
- c := alpha*op(a)*op(b) + beta*c,
- ? Could be s, d, c and z.

Notation	Precision
s	Single
d	Double
c	Complex Single
Z	Complex Double

Theory

3M METHOD

why complex mat-mat product is complex ?

- Two complex numbers multiplication costs 4 multiplication and 2 additions.
- Product of two square complex matrices of order N results in $2*N^2$ multiplications, on contrast to 2*N operations.
- Solution: We can trade multiplications with additions !

3M Method

- Technique to multiply two complex scalars using three multiplication and five real addition.
- The Number of operation drops to ¾ of the usual operation
- Use it when :

time (3 * SAXPY) < time (1*SGEMM)

time (3 * DAXPY) < time (1*DGEMM)

Algorithm

3M METHOD

Algorithm

Function: Cc = 3M(Ac , Bc)

//precondition: Ac = (Ar , Ai), Bc = (Br , Bi)

//post condition: Cc = (Cr , Ci)

{

S1 = Br – Bi

S2 = Ar + Ai

S3 = Ar – Ai

*R1 = Ar*S1*

*R2 = Br*S2*

*R3 = Bi*S3*

(Cr , Ci) = (Cr + R1 + R3 , Ci + R2 – R1)

}

Implementation

3M METHOD

C:\Windows\system32\cmd.exe

```
      ***** MATRIX A ********
41.000000 + i18467.000000       19169.000000 + i15724.000000    26962.000000 + i24464.000000    23281.000000 + i16827.00
0000
2995.000000 + i11942.000000     32391.000000 + i14604.000000    292.000000 + i12382.000000      19718.000000 + i19895.00
0000
14771.000000 + i11538.000000    25667.000000 + i26299.000000    28703.000000 + i23811.000000    17673.000000 + i4664.000
000
28253.000000 + i6868.000000     32662.000000 + i32757.000000    8723.000000 + i9741.000000      12316.000000 + i3035.000
000

      ***** MATRIX B ********
6334.000000 + i26500.000000     11478.000000 + i29358.000000    5705.000000 + i28145.000000     9961.000000 + i491.00000
0
4827.000000 + i5436.000000      3902.000000 + i153.000000       17421.000000 + i18716.000000    5447.000000 + i21726.000
000
1869.000000 + i19912.000000     17035.000000 + i9894.000000     31322.000000 + i30333.000000    15141.000000 + i7711.000
000
25547.000000 + i27644.000000    20037.000000 + i12859.000000    27529.000000 + i778.000000      22190.000000 + i1842.000
000

      ***** MATRIX C ********
1265405169.000000 + i2397814600.000000  591507298.000000 + i2463619696.000000   1066078203.000000 + i3711232267.000000
1117354530.000000 + i3481785859.000000
1265405169.000000 + i2397814600.000000  1066078203.000000 + i3711232267.000000  1265405169.000000 + i2397814600.000000
1117354530.000000 + i3481785859.000000
1265405169.000000 + i2397814600.000000  1117354530.000000 + i3481785859.000000  1117354530.000000 + i3481785859.000000
1117354530.000000 + i3481785859.000000
1265405169.000000 + i2397814600.000000  1265405169.000000 + i2397814600.000000  1265405169.000000 + i2397814600.000000
1265405169.000000 + i2397814600.000000

Elapsed time is: 0.000000
Press any key to continue . . .
```

CSCI694 - Microsoft... Gmail - Compose M... C:\Windows\system...

Theory

STRASSEN'S ALGORITHM

Theory

- Strassen (1969) devised a algorithm which computes the coefficients of the product of the matrix A € R $^{n \times n}$ and B € R $^{n \times n}$ in less than 4.7 $(n^{\log_2 7}) = n^{2.807}$
- Consider a 2-by-2 matrix multiplication
- T-method requires 8 multiplications and 4 addition
- S – method requires 7 multiplications and 18 additions

Algorithm

STRASSEN'S ALGORITHM

```
function: C = Strassen.ijk(A,B)
//precondition: Matrices A and B of order m*2 k & k = blksize
n = row(A)
for i = 1:n
 for j = 1:n
 S11[i][j] = A11[i][j] + A22[i][j]
S12[i][j] = B11[i][j] + B22[i][j]
S21[i][j] = A21[i][j] + B22[i][j]
S32[i][j] = B12[i][j] -  B22[i][j]
S42[i][j] = B21[i][j] -  B11[i][j]
S51[i][j] = A11[i][j] + A12[i][j]
S61[i][j] = A21[i][j] -  A11[i][j]
S62[i][j] = B11[i][j] + B12[i][j]
S71[i][j] = A12[i][j] -  A22[i][j]
S72[i][j] = B21[i][j] +  B22[i][j]
```

-

for i = 1:n

for j = 1:n

for k = 1:n

$$S_1[i][j] = S_1[i][j] + S_{11}[i][k] * S_{12}[k][j]$$
$$S_2[i][j] = S_2[i][j] + S_{21}[i][k] * B_{11}[k][j]$$
$$S_3[i][j] = S_3[i][j] + A_{11}[i][k] * S_{32}[k][j]$$
$$S_4[i][j] = S_4[i][j] + A_{22}[i][k] * S_{42}[k][j]$$
$$S_5[i][j] = S_5[i][j] + S_{51}[i][k] * B_{22}[k][j]$$
$$S_6[i][j] = S_6[i][j] + S_{61}[i][k] * S_{62}[k][j]$$
$$S_7[i][j] = S_7[i][j] + S_{71}[i][k] * S_{72}[k][j]$$

for i = 1:n

for j = 1:n

$C_{11}[i][j] = S_1[i][j] + S_4[i][j] - S_5[i][j] + S_7[i][j]$

$C_{12}[i][j] = S_3[i][j] + S_5[i][j]$

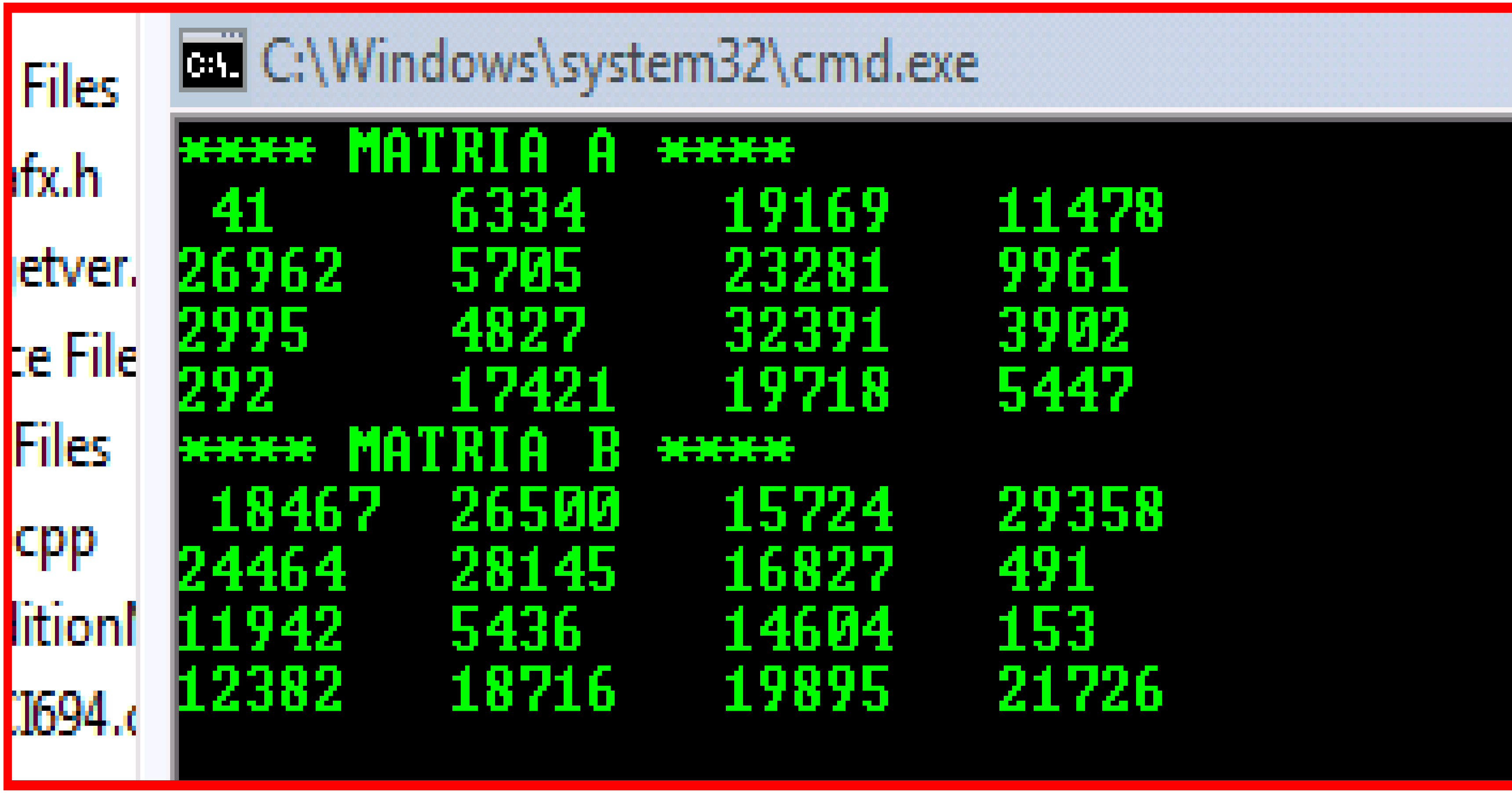
$C_{21}[i][j] = S_2[i][j] + S_4[i][j]$

$C_{22}[i][j] = S_1[i][j] + S_3[i][j] - S_2[i][j] + S_6[i][j]$

end Strassen.ijk(A,B)

Implementation & Results

STRASSEN'S ALGORITHM



Matrix – A and Matrix - B

4 x 4 matrices

```
 **** A11 ******
41      6334
26962   5705

 **** B11 ******
18467   26500
24464   28145

 **** A12 ******
19169   11478
23281   9961

 **** B12 ******
15724   29358
16827   491
```

```
 ***** A21 ********
2995      4827
292       17421

 ***** B21 ********
11942     5436
12382     18716

 ***** A22 ********
32391     3902
19718     5447

 ***** B22 ********
14604     153
19895     21726
```

```
 **** S11 ******
32432   10236
46680   11152

 **** S12 ******
33071   26653
44359   49871
Likewise calculate S21, S32...S71, S72

 **** S1 ******
1526617396      1374889652
2038445848      1800323432
Likewise calculate S2, S3...S6, S7

 **** C11 ******
526748917       498381862
1038833178      1188045817

 **** C12 ******
615525788       256617557
1058118342      1014326230

 **** C21 ******
608524279       464330723
734496818       707185145

 **** C22 ******
678985763       180027942
694064312       138484623
Total time taken for 4 * 4 ordered matriA multiSlicatuion [StrassenÆs method] is 0.053000 seconds
```

Theory

WINOGRAD'S ALGORITHM

Theory

- Modification and improvement of S – Method
- Again, consider a 2-by-2 matrix multiplication
- S – method requires 7 multiplications and 18 additions
- W – method required 7 multiplications but only 15 additions.
- The complexity is improved to $O(n^{2.795})$

Algorithm

WINOGRAD'S ALGORITHM

Algorithm

- S1 = (A21 + A22),
- S2 = S1 – A11,
- S3 = A11 – A21,
- S4 = A12 – S2,
- S5 = B12 – B11,
- S6 = B22 – S5,
- S7 = B22 – B12,
- S8 = S6 – B21,

Algorithm cont.,

- M1 = S2S6,
- M2 = X11Y11,
- M3 = X12Y21,
- M4 = S3S7,
- M5 = S1S5,
- M6 = S4Y22,
- M7 = X22S8,

Algorithm cont.,

- $T1 = M1 + M2$,
- $T2 = T1 + M4$,
- $Z11 = M2 + M3$,
- $Z12 = T1 + M5 + M6$,
- $Z21 = T2 – M7$,
- $Z22 = T2 + M5$

Theory

COPPERSMITH–WINOGRAD ALGORITHM

Theory

- Asymptotically fastest known algorithm for square matrix multiplication as of today !
- Asymptotic complexity is $(O^{2.376})$
- "Not practical, required so large matrices that the modern hardware can't handle" Robinson 2005

Evaluation of Powers

Theory

BINARY METHOD

Theory

- How to calculate x^n ?

1. write n in binary number system
2. Replace 1 by SX and 0 by S
3. Cross off left most SX
4. S – squaring and X – multiply by x
5. n = 23 => 10111 => SXSSXSXSX => SSXSXSX

Theory

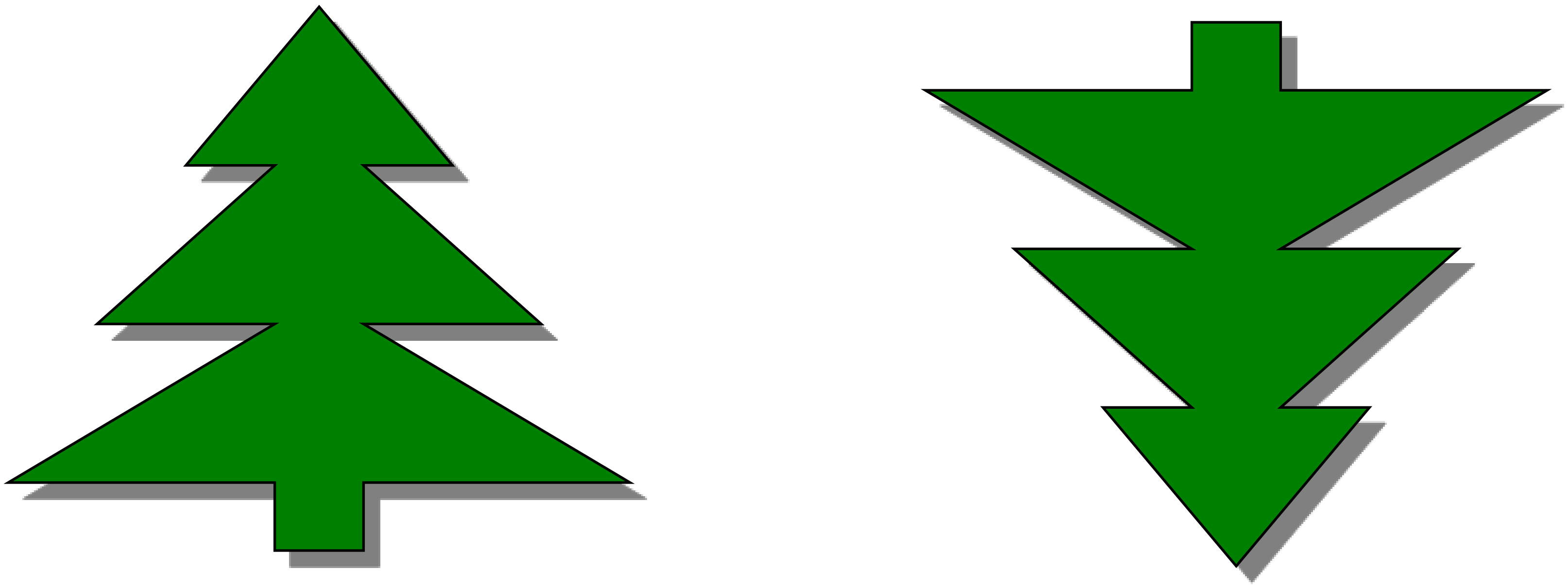
- Known before 200 BC
- Appeared in Pingala’s Hindu Classic *chandha-sutra*
- Required no temporary storage
- Problem : scanning from left to right
- Algorithm A is devised to scan from right – left
- Al-kashi stated algorithm A in A.D. 1427
- Binary method is not optimal

Theory

FACTOR METHOD

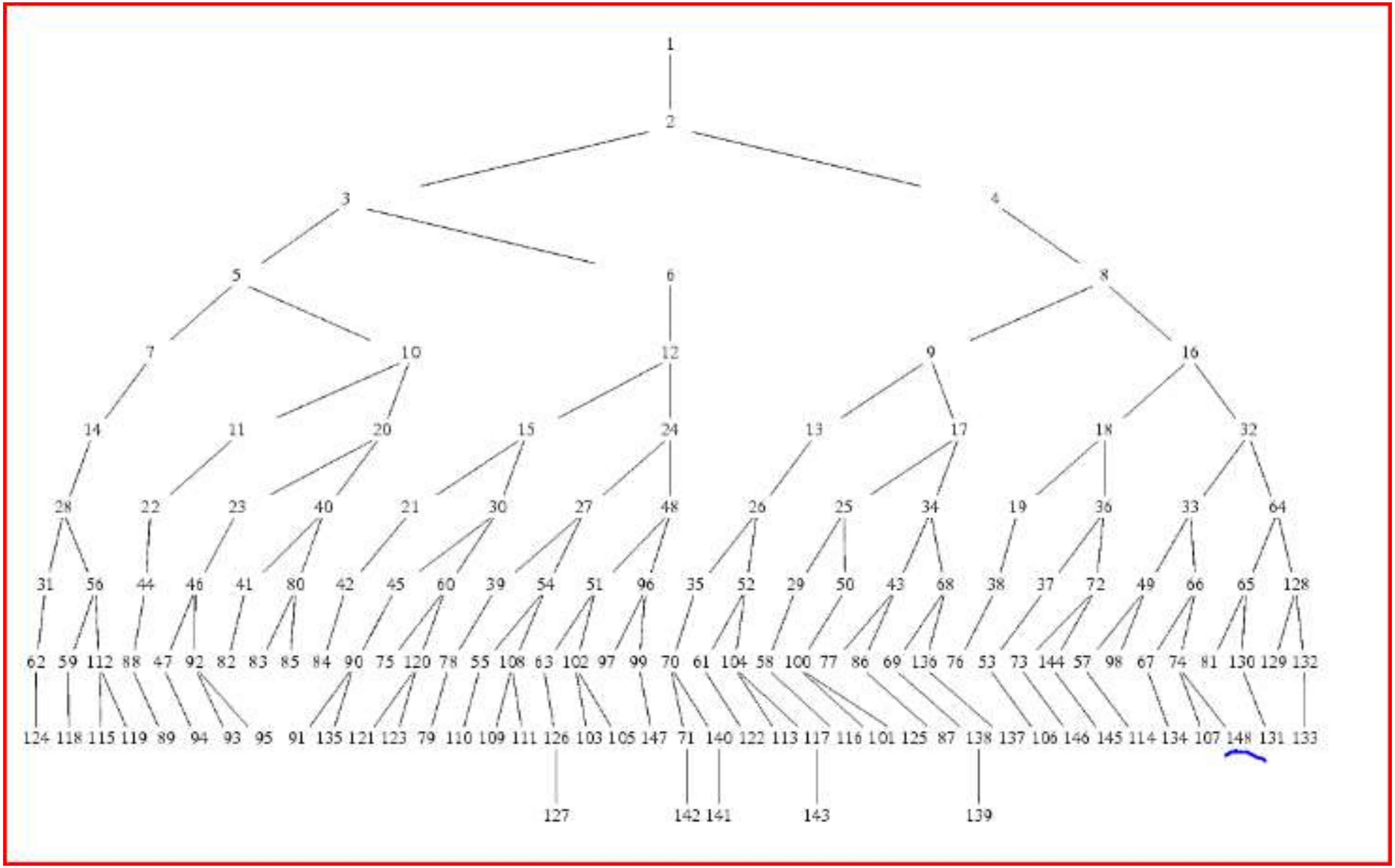
Theory

- Method based on factorization of n.
- If n = p*q, where p is the smallest prime factor of n and q > 1.
- If n is prime number, calculate x^{n-1} and multiply by x
- $y^{11} = y^{10} * y = (y^2)^5 * y$
- Required 8 multiplication on contrast 9 by binary method



Theory

POWER TREE



The "power tree"

Minimizes the number of multiplication, n<= 149

Is power tree optimal ?

- Is power tree provides optimal solution for all values of n?
- No.
- The smallest example illustrated by knuth was

n = 77,154,233 !

- Then what is the economical way to compute x^n ? Of course, given that the only operation permitted is multiplying two already-computed powers?

Theory

ADDITION CHAINS

Theory

- Addition chain is a finite sequence of positive integers

$$1 = a0, a1, ..., ar = n$$

With the property that

$$a_i = a_j + a_k \quad \text{for some } k \le j < i$$

- Addition chains should be strictly monotonic increasing

Minimal length

- Let $l(n)$ be the minimal length of an addition chain for a given number n.
- The $l(n)$ is known only for relatively small values
- For n large,

$$l(n) = \log(n) + (1 + O(1)) (\log(n)/\log(\log(n)))$$

The lower bound was shown by P. Erdos [1960]

Example: A Shortest Addition Chain for n= 148

0	1
1(0)	2
2(1)	4
3(2)	8
4(3, 0)	9
5(4)	18
6(5)	36
7(6, 0)	37
8(7)	74
9(8)	148

Star Chain

- If j or k equals i-1 for all positive indices i --, then each ai in the chain requires the previous ai-1. Such a chain a star-chain.
- Its minimal length is denoted by $l^*(n)$.
- is $l(n) = l^*(n)$?
- Walter Hansen [1958] proved in his theorem that for large values of n,

$$l(n) < l^*(n)$$

Theory & Application

MATRIX POWERS USING EIGEN DECOMPOSITION

Fibonacci series

- The recurrence relation of the Fibonacci series as matrix-vector product
- $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ and vector $U_n = \begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix}$
- Recurrence relation is immediate

$$U_n = A\, U_{n-1} \text{ for all } n >= 1$$

$$\text{for } n = 0 \text{ that is } U_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Therefore, $U_n = A^n\, U_0$

Eigen pairs to Calculate A^n

- Eigen values Characteristic equation:

$$AX = \lambda X$$

$$\Rightarrow (A - \lambda I)X = 0; \text{ if there exits inverse}$$

$$\Rightarrow \det((A - \lambda I)) = 0;$$

$$\Rightarrow \lambda_1 = (1 - \mathrm{sqrt}(5))/2$$

$$\text{And } \lambda_2 = (1 + \mathrm{sqrt}(5))/2$$

Eigen pairs to Calculate A^n

$$AX = \lambda X$$

$$\Rightarrow A(AX) = A(\lambda X)$$

$$\Rightarrow A^2 X = \lambda (A X)$$

$$\Rightarrow A^2 X = \lambda^2 X$$

$$\Rightarrow A^n X = \lambda^n X$$

Eigen pairs to Calculate A^n

- In our case, we are interested in $X = U_0$.
- Therefore, express the vector U_0 as a linear combination of x and y
- $U_0 = ax + by$
- Solving the above linear equation results in
- a = -1/sqrt(5) and b = + 1/sqrt(5)
- U_0 = (-1/sqrt(5)) x + (1/sqrt(5)) y
-

Eigen pairs to Calculate A^n

$$U_n = A^n U_0$$

$$\Rightarrow U_n = (-1/\text{sqrt}(5)\ )\ \lambda_1^n\ x + (1/\text{sqrt}(5))\ \lambda_2^n\ y$$

$$\Rightarrow F_n = (-1/\text{sqrt}(5)\ )\ ((1 - \text{sqrt}(5))/2)^n + (1/\text{sqrt}(5))\ ((1 + \text{sqrt}(5))/2)^n$$

Putting it together…

CONCLUSION

Conclusion

- Let us consider a matrix $A_{n \times n}$, the asymptotic complexity of calculating the A^n is as follow:

1. The naïve method requires $O(n^4)$.
2. The Binary method requires $O(n^3 log(n))$.
3. Eigen decomposition requires $O(n*log(n)) + O(n^3)$.

Hint: $(A^{n \times n})^n = (X^{-1} \lambda X)^n = X^{-1} \lambda^n X$

λ^n required $O(n*log(n)$ and matrix multiplication requires $O(n^3)$

QUESTIONS ?

THANK YOU.