CSE 5311 Notes 12: Matrices

(Last updated 11/12/15 9:12 AM)

STRASSEN'S MATRIX MULTIPLICATION

Matrix addition:

$$m\overset{n}{\begin{bmatrix} \bullet & & \\ & & \\ & & \end{bmatrix}} \quad m\overset{n}{\begin{bmatrix} \bullet & & \\ & & \\ & & \end{bmatrix}} \rightarrow m\overset{n}{\begin{bmatrix} \bullet & & \\ & & \\ & & \end{bmatrix}}$$

takes mn scalar additions.

Everyday matrix multiply:

$$m\overset{n}{\begin{bmatrix} \longleftrightarrow \\ \\ \\ \end{bmatrix}} \quad n\overset{p}{\begin{bmatrix} \updownarrow & & \\ & & \\ & & \end{bmatrix}} \rightarrow m\overset{p}{\begin{bmatrix} \bullet & & \\ & & \\ & & \end{bmatrix}}$$

takes mnp scalar multiplies and $m(n-1)p$ scalar additions to produce result matrix.

Let $m = n = p$.

Best lower bound is $\Omega(n^2)$.

For $n = 2$:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

is done by everyday method using:

8 scalar multiplies

4 scalar additions

but Strassen's method (CLRS, p.79) uses:

7 scalar multiplies

18 scalar additions

When $n = 2^k$: A_{ij} and B_{ij} are $2^{k-1} \times 2^{k-1}$ submatrices to be multiplied *recursively* using Strassen's method.

Suppose $n = 4$.

Everyday method takes $4^3 = 64$ scalar multiplies and 16•3 = 48 scalar additions.

Strassen's method takes:

7 recursive 2×2 matrix multiplies, each using 7 scalar multiplies and 18 scalar additions.

18 2×2 matrix additions, each using 4 scalar additions.

This gives 49 scalar multiplies and 198 scalar additions.

Let $M(k)$ = number of scalar multiplies for $2^k \times 2^k = n \times n$:

$$M(0) = 1$$
$$M(1) = 7$$
$$M(k) = 7M(k-1) = 7^k = 7^{\lg n} = n^{\lg 7} \approx n^{2.81}$$ Note that the constant is 1.

Let $P(k)$ be the number of additions (including subtractions) done for $2^k \times 2^k = n \times n$:

$$P(0) = 0$$
$$P(1) = 18$$
$$P(k) = 18\left(2^{k-1}\right)^2 + 7P(k-1) = 18\left(\frac{2^k}{2}\right)^2 + 7P(k-1)$$

Let $P'\left(2^k\right) = P(k)$

$$P'(n) = 18\left(\frac{n}{2}\right)^2 + 7P'\left(\frac{n}{2}\right) = \frac{9}{2}n^2 + 7P'\left(\frac{n}{2}\right)$$

Master Method:

$a = 7 \quad b = 2 \quad \log_b a = \lg 7 \approx 2.81 \quad f(n) = \frac{9}{2}n^2$

$\frac{9}{2}n^2 = O\left(n^{2.81-\varepsilon}\right)$

Case 1: $P'(n) = \Theta\left(n^{2.81}\right)$

For the state of research on fast matrix multiplication:

http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=2421119.2421134

KRONROD'S ALGORITHM FOR BOOLEAN MATRIX MULTIPLICATION (AKA Four Russians' Method)

Boolean version of matrix multiplication substitutes ∧ for • and ∨ for + in numerical version:

$$c[i][j]=1 \Leftrightarrow (\exists k)(a[i][k]=1 \wedge b[k][j]=1)$$

```
for (i=0;i<n;i++)
  for (j=0;j<n;j++)
  {
    c[i][j]=0;
    for (k=0;k<n;k++)
      if (a[i][k] && b[k][j])
      {
        c[i][j]=1;
        break;
      }
  }
```

Since t bits can be packed into a byte, word, etc., a speed-up may be obtained.

Let $t = 8$ and suppose $n = 256$ for multiplying together $n \times n$ matrices A and B (`kronrodChar5311.c`):

```
unsigned char A[256][32],B[256][32],C[256][32];
```

where an unpacked matrix can be packed with:

```
// Converts column bit subscript to subscript for unsigned char
#define SUB2CHAR(sub) ((sub)/8)
// Determines bit position within unsigned char for column bit subscript
#define SUB2BIT(sub) ((sub)%8)
// Gets 0/1 value for a column bit subscript
#define GETBIT(arr,row,col) ((arr[row][SUB2CHAR(col)]>>SUB2BIT(col))&1)
// Sets bit to 1
#define SETBIT(arr,row,col) \
 (arr[row][SUB2CHAR(col)]=arr[row][SUB2CHAR(col)]|(1<<SUB2BIT(col)))
// Clears bit to 0
#define CLEARBIT(arr,row,col) \
  (arr[row][SUB2CHAR(col)]=arr[row][SUB2CHAR(col)]&(255-(1<<SUB2BIT(col))))

for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    if (Aunpacked[i][j])
      SETBIT(A,i,j);
    else
      CLEARBIT(A,i,j);
```

Row-oriented boolean matrix multiplication is:

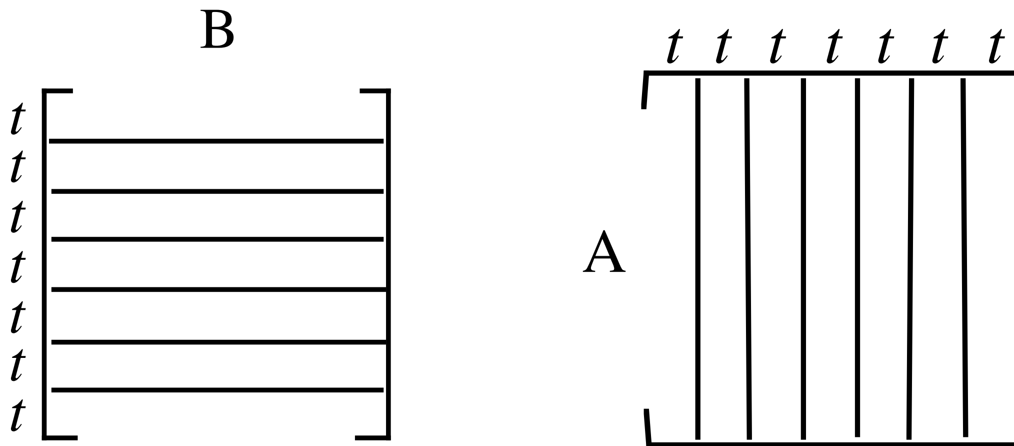
1. Find positions with ones in row `i` of `A` (i.e. column indices).

2. Row `i` of `C` is the result of or'ing together the rows of `B` for the positions from 1.

```
for (i=0; i<n; i++)
{
  for (j=0; j<n/t; j++)       // Clear output row i
    C[i][j]=0;
  for (j=0; j<n/t; j++)
    for (k=0; k<t; k++)
      if ((A[i][j]>>k) & 1)   // Examine row i, bit # t*j+k
        for (p=0; p<n/t; p++) // Set corresponding bits
          C[i][p]=C[i][p] | B[t*j+k][p];
}
```

Time is $O\left(\frac{n^3}{t}\right)$ (giving $\frac{256^3}{8} = 2{,}097{,}152$ byte or's).

Kronrod's technique: Preprocessing + addressing to reduce cost of or'ing together rows



1. $\frac{n}{t}$ groups of t consecutive `B` rows (group `i` is rows `t*i . . . t*i+t-1`).

2. Each group is preprocessed by or'ing together all 2^t possible combinations (`subset`) of the t rows (using $2^t \frac{n}{t}$ extra space for `subset`).

   Each of rows 0 . . . $2^j - 1$ in `subset` are or'ed with row `t*i+j` of `B` to obtain rows 2^j . . . $2^{j+1} - 1$ in `subset`.

3. These results are then referenced by going down the corresponding column of `A` using the t (packed) bits as a subscript to the extra table of or'ed combinations.

Thus, much of the effort for row or'ing is transfered to the preprocessing:

```
unsigned char subset[256][32];

// Initialize output matrix
for (i=0; i<n; i++)
  for (j=0; j<n/t; j++)
    C[i][j]=0;

// Initialize empty subset - same for each group
for (i=0; i<n/t; i++)
  subset[0][i]=0;
// There are n/t groups of t rows in B
for (i=0; i<n/t; i++)
{
  rangeStart=1;
  // Include each row to achieve combinations OR'ed together
  for (j=0; j<t; j++)
  {
    for (k=rangeStart; k<2*rangeStart; k++)
      for (q=0; q<n/t; q++)
        // OR the subset without row j of this group with row j
        // of this group.  First row of group is B[t*i].
        subset[k][q]=subset[k-rangeStart][q] | B[t*i+j][q];
    rangeStart*=2;
  }

  // Update result rows based on this group of t B rows
  for (j=0; j<n; j++)
    for (q=0; q<n/t; q++)
      C[j][q]=C[j][q] | subset[A[j][i]][q];
}
```

Time is $\mathrm{O}\left(\frac{n}{t}\left(2^t \frac{n}{t} + \frac{n^2}{t}\right)\right) = \mathrm{O}\left(2^t \frac{n^2}{t^2} + \frac{n^3}{t^2}\right)$ (giving 524,288 byte or's).

If $t = \lg n$, row-oriented gives $\mathrm{O}\left(\frac{n^3}{\log n}\right)$ time while Kronrod's method is $\mathrm{O}\left(\frac{n^3}{\log^2 n}\right)$.

This technique will be applied in Notes 14 to longest common subsequences.