Matrix Multiplication

Nur Dean

PhD Program in Computer Science
The Graduate Center, CUNY

05/01/2017

Today, I will talk about matrix multiplication and 2 parallel algorithms to use for my matrix multiplication calculation.

Overview

Definition of A Matrix

- A matrix is a rectangular two-dimensional array of numbers
- We say a matrix is *mxn* if it has m rows and n columns.
- We use a_{ij} to refer to the entry in i^{th} row and j^{th} column of the matrix A.

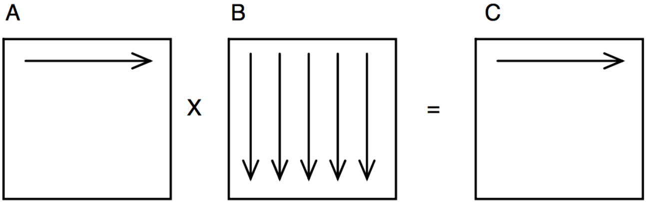
- Matrix multiplication is a fundamental linear algebra operation that is at the core of many important numerical algorithms.
- If A,B, and C are NxN matrices, then $C = AB$ is also an NxN matrix, and the value of each element in C is defined as:

$$C_{ij} = \sum_{k=0}^{N} A_{ik} B_{kj}$$

Algorithm 1 Sequential Algorithm

```
for (i=0; i < n; i++ ) do
    for (j = 0; i < n; j++) do
        c[i][j] = 0;
        for (k=0; k < n; k++) do
            c[i][j]+ = a[i][k] * b[k][j]
        end for
    end for
end for
```

- During the first iteration of loop variable i the first matrix A row and all the columns of matrix B are used to compute the elements of the first result matrix C row
- This algorithm is an iterative procedure and calculates sequentially the rows of the matrix C. In fact, a result matrix row is computed per outer loop (loop variable i) iteration.



As each result matrix element is a scalar product of the initial matrix A row and the initial matrix B column, it is necessary to carry out $n^2(2n-1)$ operations to compute all elements of the matrix C. As a result the time complexity of matrix multiplication is;
$T_1 = n^2(2n-1)\tau$

where τ is the execution time for an elementary computational operation such as multiplication or addition.

Checkerboard

Most parallel matrix multiplication functions use a checkerboard distribution of the matrices. This means that the processes are viewed as a grid, and, rather than assigning entire rows or entire columns to each process, we assign small sub-matrices. For example, if we have four processes, we might assign the element of a 4x4 matrix as shown below, checkerboard mapping of a 4x4 matrix to four processes.

Process 0 a_{00} a_{01} a_{10} a_{11}	**Process 1** a_{02} a_{03} a_{12} a_{13}
Process 2 a_{20} a_{21} a_{30} a_{31}	**Process 3** a_{22} a_{23} a_{32} a_{33}

Fox's Algorithm

Process 0	**Process 1**
a_{00} a_{01}	a_{02} a_{03}
a_{10} a_{11}	a_{12} a_{13}
Process 2	**Process 3**
a_{20} a_{21}	a_{22} a_{23}
a_{30} a_{31}	a_{32} a_{33}

- Fox's algorithm is a one that distributes the matrix using a checkerboard scheme like the above.
- In order to simplify the discussion, lets assume that the matrices have order n, and the number of processes, p, equals n^2. Then a checkerboard mapping assigns a_{ij}, b_{ij}, and c_{ij} to process (i, j).
- In a process grid like the above, the process (i,j) is the same as process $p = i * n + j$, or, loosely, process (i, j) using row major form in the process grid.

Cont. Fox's Algorithm

- Fox's algorithm takes n stages for matrices of order n one stage for each term $a_{ik}b_{kj}$ in the dot product
  $C_{ij} = a_{i0}b_{0j} + a_{i1}b_{1i} + . \; . \; . + a_{i,n-1}b_{n-1,j}$
- Initial stage, each process multiplies the diagonal entry of A in its process row by its element of B:
  Stage 0 on process(i,j): $c_{ij} = a_{ii}b_{ij}$
- Next stage, each process multiplies the element immediately to the right of the diagonal of A by the element of B directly beneath its own element of B:
  Stage 1 on process(i,j): $c_{ij} = c_{ij} + a_{i,i+1}b_{i+1,j}$
- In general, during the k^{th} stage, each process multiplies the element k columns to the right of the diagonal of A by the element k rows below its own element of B:
  Stage k on process(i,j): $c_{ij} = c_{ij} + a_{i,i+k}b_{i+k,j}$

Example of the Algorithm Applied to 2x2 Matrices

$$A=\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} \quad B=\begin{vmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{vmatrix}$$

$$C=\begin{vmatrix} a_{00}b_{00}+a_{01}b_{10} & a_{00}b_{01}+a_{01}b_{11} \\ a_{10}b_{00}+a_{11}b_{10} & a_{10}b_{01}+a_{11}b_{11} \end{vmatrix}$$

Assume that we have n^2 processes, one for each of the elements in A, B, and C. Call the processes P_{00}, P_{01}, P_{10}, and P_{11}, and think of them as being arranged in a grid as follows:

P_{00}	P_{01}
P_{10}	P_{11}

- Stage 0

  (a) We want $a_{i,i}$ on process $P_{i,j}$, so broadcast the diagonal elements of A across the rows, $(a_{ii} \rightarrow P_{ij})$ This will place $a_{0,0}$ on each $P_{0,j}$ and $a_{1,1}$ on each $P_{1,j}$. The A elements on the P matrix will be

a_{00}	a_{00}
a_{11}	a_{11}

  (b) We want $b_{i,j}$ on process $P_{i,j}$, so broadcast B across the rows $(b_{ij} \rightarrow P_{ij})$ The A and B values on the P matrix will be

a_{00} b_{00}	a_{00} b_{01}
a_{11} b_{10}	a_{11} b_{11}

(c) Compute $c_{ij} = AB$ for each process

a_{00} b_{00} $c_{00} = a_{00}b_{00}$	a_{00} b_{01} $c_{01} = a_{00}b_{01}$
a_{11} b_{10} $c_{10} = a_{11}b_{10}$	a_{11} b_{11} $c_{11} = a_{11}b_{11}$

We are now ready for the second stage. In this stage, we broadcast the next column (mod n) of A across the processes and shift-up (mod n) the B values.

- Stage 1
  (a) The next column of A is $a_{0,1}$ for the first row and $a_{1,0}$ for the second row (it wrapped around, mod n). Broadcast next A across the rows

a_{01} b_{00} $c_{00} = a_{00}b_{00}$	a_{01} b_{01} $c_{01} = a_{00}b_{01}$
a_{10} b_{10} $c_{10} = a_{11}b_{10}$	a_{10} b_{11} $c_{11} = a_{11}b_{11}$

(b) Shift the B values up. $B_{1,0}$ moves up from process $P_{1,0}$ to process $P_{0,0}$ and $B_{0,0}$ moves up (mod n) from $P_{0,0}$ to $P_{1,0}$. Similarly for $B_{1,1}$ and $B_{0,1}$.

a_{01} b_{10} $c_{00} = a_{00} b_{00}$	a_{01} b_{11} $c_{01} = a_{00} b_{01}$
a_{10} b_{00} $c_{10} = a_{11} b_{10}$	a_{10} b_{01} $c_{11} = a_{11} b_{11}$

(c) Compute $C_{ij} = AB$ for each process

a_{01} b_{10} $c_{00} = c_{00} + a_{01}b_{10}$	a_{01} b_{11} $c_{01} = c_{01} + a_{01}b_{11}$
a_{10} b_{00} $c_{10} = c_{10} + a_{10}b_{00}$	a_{10} b_{01} $c_{11} = c_{11} + a_{10}b_{01}$

The algorithm is complete after n stages and process $P_{i,j}$ contains the final result for $c_{i,j}$.

Example 3x3 Fox's Algorithm

Consider multiplying 3x3 block matrices:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 2 \\ 2 & 0 & 3 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 2 & 9 \\ 4 & 4 & 5 \\ 4 & 2 & 6 \end{bmatrix}$$

Stage 0:

Process $(i, i \quad mod 3)$	Broadcast along row i
(0,0)	a_{00}
(1,1)	a_{11}
(2,2)	a_{22}

$$\begin{array}{ccc} a_{00}, b_{00} & a_{00}, b_{01} & a_{00}, b_{02} \\ a_{11}, b_{10} & a_{11}, b_{11} & a_{11}, b_{12} \\ a_{22}, b_{20} & a_{22}, b_{21} & a_{22}, b_{22} \end{array}$$

Process (i, j) computes:

c_{00}=1x1=1	c_{01}=1x0=0	c_{02}=1x2=2
c_{10}=1x2=2	c_{11}=1x0=0	c_{12}=1x3=3
c_{20}=1x1=1	c_{21}=1x2=2	c_{22}=1x1=1

Shift-rotate on the columns of B

Stage 1:

Process $(i,(i+1) \quad mod 3)$	Broadcast along row i
(0,1)	a_{01}
(1,2)	a_{12}
(2,0)	a_{20}

$$\begin{matrix} a_{01}, b_{10} & a_{01}, b_{11} & a_{01}, b_{12} \\ a_{12}, b_{20} & a_{12}, b_{21} & a_{12}, b_{22} \\ a_{20}, b_{00} & a_{20}, b_{01} & a_{20}, b_{02} \end{matrix}$$

Process (i,j) computes:

c_{00}=1+(2x2)=5	c_{01}=0+(2x0)=0	c_{02}=2+(2x3)=8
c_{10}=2+(2x1)=4	c_{11}=0+(2x2)=4	c_{12}=3+(2x1)=5
c_{20}=1+(1x1)=2	c_{21}=2+(1x0)=2	c_{22}=1+(1x2)=3

Shift-rotate on the columns of B

Stage 2:

Process $(i,(i+2) \quad mod 3)$	Broadcast along row i
(0,2)	a_{02}
(1,0)	a_{10}
(2,1)	a_{21}

$$\begin{matrix} a_{02}, b_{20} & a_{02}, b_{21} & a_{02}, b_{22} \\ a_{10}, b_{00} & a_{10}, b_{01} & a_{10}, b_{02} \\ a_{21}, b_{10} & a_{21}, b_{11} & a_{21}, b_{12} \end{matrix}$$

Process (i,j) computes:

c_{00}=5+(1x1)=6	c_{01}=0+(1x2)=2	c_{02}=8+(1x1)=9
c_{10}=4+(0x1)=4	c_{11}=4+(0x0)=4	c_{12}=5+(0x2)=5
c_{20}=2+(1x2)=4	c_{21}=2+(1x0)=2	c_{22}=3+(1x3)=6

Algorithm 2 Fox's Algorithm Psuedocode

```
/* my process row = i , my process column = j */
q = sqrt(p);
dest = ((i-1) mod q , j);
for (stage=0; stage<q; stage++ )
{
        k_bar=(i+stage) mod q;
        (a) Broadcast A[i,k_bar] across process row i;
        (b) C[i,j] = C[i,j] + A[i,k_bar]*B[k_bar,j];
        (c) Send B[(k_bar+1) mod q, j] to dest;
        Receive B[(k_bar+1) mod q, j] from source;
}
```

Analysis of Fox's Algorithm

- Let A, B be nxn matrices, and $C = A * B$, $C_{ij} = \sum_{k=0}^{q-1} A_{ik} B_{kj}$
- Let $p = q^2$ number of processors organized in a q x q grid
- Store $(i,j)^{th}$ n/q x n/q block of A, B, and C on process (i,j)
- Execution of the Fox algorithm requires q iterations, during which each processor multiplies its current blocks of the matrices A and B, and adds the multiplication results to the current block of the matrix C.With regard to the above mentioned assumptions,
  Computation time:
  $$q\left(\frac{n}{q}x\frac{n}{q}x\frac{n}{q}\right) = \frac{n^3}{q^2} = \frac{n^3}{p}$$
- As a result, the speedup and efficiency of the algorithm look as follows:
  $$S_p = \frac{n^3}{n^3/p} = p$$
  $$E_p = \frac{n^3}{p.(n^3/p)} = 1$$

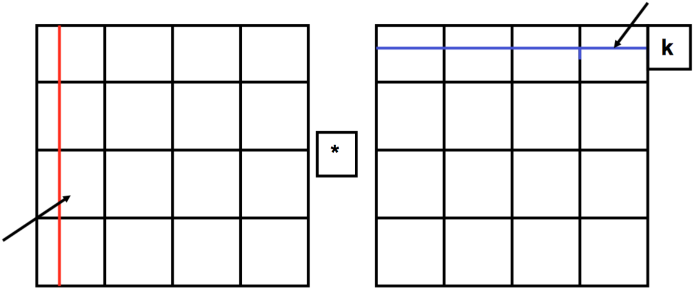
SUMMA:Scalable Universal Matrix Multiplication Algorithm

- Slightly less efficient, but simpler and easier to generalize.
- Uses a shift algorithm to broadcast

- The SUMMA algorithm computes n partial outer products:

$$\begin{aligned}&\text{for } k := 0 \text{ to } n-1\\ &\qquad C[:,:] += A[:,k] \cdot B[k,:]\end{aligned}$$

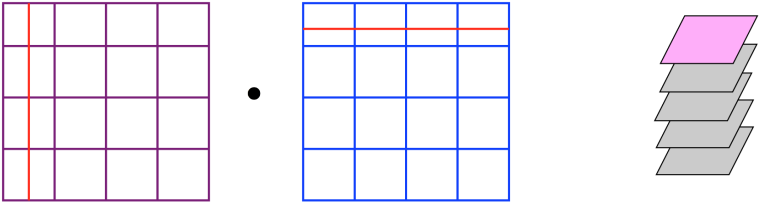
- Each row k of B contributes to the n partial outer products



- Compute the sum of n outer products
- Each row and column (k) of A and B generates a single outer product

Column vector $A[:, k]$ ($nx1$) and a vector $B[k, :]$ ($1xn$)

for $k := 0$ to $n - 1$
  $C[:, :] += A[:, k] \cdot B[k, :]$

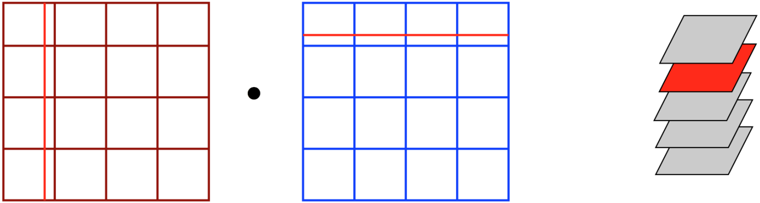


- Compute the sum of n outer products
- Each row and column (k) of A and B generates a single outer product

  $A[:, k+1] \cdot B[k+1, :]$

for $k := 0$ to $n-1$

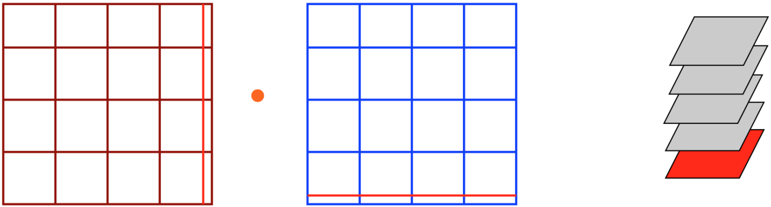
$C[:, :] += A[:, k] \cdot B[k, :]$

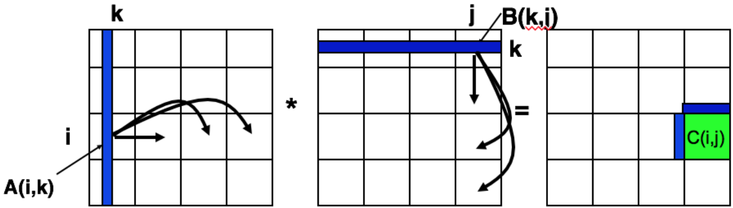


- Compute the sum of n outer products
- Each row and column (k) of A and B generates a single outer product

  $A[:, n-1] \cdot B[n-1, :]$

for $k := 0$ to $n-1$
    $C[:, :] += A[:, k] \cdot B[k, :]$





- For each k (between 0 and $n-1$),
- Owner of partial row k broadcasts that row along its process column
- Owner of partial column k broadcasts that column along its process row

$$C(i,j) = C(i,j) + \sum_k A(i,k) * B(k,j)$$

- Assume a p_r by p_c processor grid ($p_r = p_c = 4$ above)
  Need not be square

Example 3x3 SUMMA Algorithm

Consider multiplying 3x3 block matrices:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 2 \\ 2 & 0 & 3 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 2 & 9 \\ 4 & 4 & 5 \\ 4 & 2 & 6 \end{bmatrix}$$

- Owner of partial row 0 broadcasts that row along its process column and owner of partial column 0 broadcasts that column along its process row

	1	0	2
1	1	0	2
0	0	0	0
1	1	0	2

- Owner of partial row 1 broadcasts that row along its process column and owner of partial column 1 broadcasts that column along its process row

	2	0	3
2	4	0	6
1	2	0	3
1	2	0	3

- Owner of partial row 2 broadcasts that row along its process column and owner of partial column 2 broadcasts that column along its process row

	1	2	1
1	1	2	1
2	2	4	2
1	1	2	1

- When we sum all the entries we get the following matrix:

$$\begin{bmatrix} 6 & 2 & 9 \\ 4 & 4 & 5 \\ 4 & 2 & 6 \end{bmatrix}$$

Algorithm 3 SUMMA Algorithm

```
for k = 0 to n − 1 do
    for all i = 1 to p_r do
        owner of A(i, k) broadcasts it to whole processor row;
    end for
    for all j = 1 to p_c do
        owner of B(k, j) broadcasts it to whole processor column;
    end for
    Receive A(i, k) into Acol
    Receive B(k, j) into Brow
    C_myproc = C_myproc + Acol * Brow
end for
```

- We can also take $k = 0$ to $n/b - 1$ where b is the block size = cols in $A(i,k)$ and rows in $B(k,j)$

SUMMA Performance Model

- To simplify analysis only, assume $s = \sqrt{p}$

Algorithm 4 SUMMA Performance Model

for $k = 0$ to $n/b - 1$ **do**
  for all $i = 1$ to s **do**
    owner of $A(i, k)$ broadcasts it to whole processor row;
    $\%time = \log s * (\alpha + \beta * b * n/s)$, using a tree
  end for
  for all $j = 1$ to s **do**
    owner of $B(k, j)$ broadcasts it to whole processor column;
    $\%time = \log s * (\alpha + \beta * b * n/s)$, using a tree
  end for
  Receive $A(i, k)$ into Acol
  Receive $B(k, j)$ into Brow
  $C_{myproc} = C_{myproc}$ + Acol * Brow
  $\%time = 2 * (n/s)^2 * b$

Analysis of SUMMA

- $T(p) = 2 * \frac{n^3}{p} + \alpha * \log p * \frac{n}{b} + \beta * \log p * \frac{n^2}{s}$
- $E(p) = \frac{1}{(1 + \alpha * \log p * \frac{p}{(2 * b * n^2)} + \beta * \log p * \frac{s}{(2 * n)})}$

Where α is the start-up cost of a message, and β is the bandwidth

THANK YOU FOR YOUR ATTENTION TODAY!