SUMMA

Scalable Universal Matrix Multiplication Algorithm

Naïve matrix multiply

```
For i = 0 to n
    For j = 0 to n
        For k = 0 to n
            C[i,j] += A[i,k]*B[k,j]
```

Calculates n^2 dot products (inner products)

C[i,j] = A[i,:]*B[:,j]

Naïve matrix multiply

What happens if we switch the order?

```
For k = 0 to n
    For i = 0 to n
        For j = 0 to n
            C[i,j] += A[i,k]*B[k,j]
```

Naïve matrix multiply

What happens if we switch the order?

```
For k = 0 to n
    For i = 0 to n
        For j = 0 to n
            C[i,j] += A[i,k]*B[k,j]
```

Calculates n outer products

Outer product

$$\begin{vmatrix} a \\ b \\ c \end{vmatrix} * \quad [\ e \quad f \quad g\ ] \quad = \begin{vmatrix} a*e & a*f & a*g \\ b*e & b*f & b*g \\ c*e & c*f & c*g \end{vmatrix}$$

Introducing SUMMA

- Processors arranged in grid, P(i,j)
- Example, 2x2 grid:

P(1,1)	P(1,2)
P(2,1)	P(2,2)

A

x

P(1,1)	P(1,2)
P(2,1)	P(2,2)

B

=

P(1,1)	P(1,2)
P(2,1)	P(2,2)

C

Introducing SUMMA

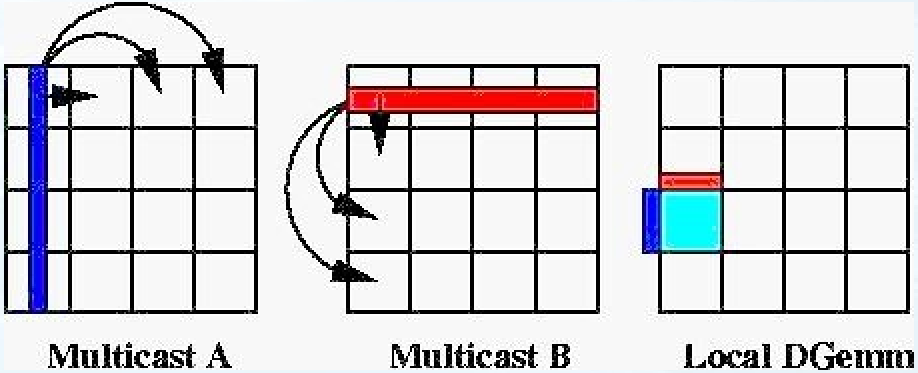
- Let A=mxk, B=kxn => C=mxn
- Let each process do k outer products

How do we handle the communication?

SUMMA

- 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

Image credit: Stephen J. Fink



SUMMA

Complete algorithm. On each process P(i,j):

For k = 0...n-1

Bcast column k of A (a_i) within row i

Bcast row k of B (b_j) within column j

Do C += outer product (a_i,b_j)

Communication cost

- What's the cost of communication? Let α be the startup cost of a message, and β be the bandwidth

Communication cost

- What's the cost of communication? Let α be the startup cost of a message, and β be the bandwidth
- Bcast among p processes takes $\log(p)*(\alpha+\beta*s)$ time, where s is the size of the message

Communication cost

- Bcast among p processes takes log(p) ($\alpha+\beta s$) time, where s is the size of the message
- For each k, there are one Bcast along columns and one Bcast along rows
- Each partial column has size m/r, each partial row has size n/c

Communication cost

- Bcast among p processes takes $\log(p)(\alpha+\beta s)$ time, where s is the size of the message
- Total: $k*(\log(c)(\alpha+\beta*m/r)+\log(r)(\alpha+\beta*n/c))$
- Not very efficient
  - Lots of messages

Improvements?

- Can we reduce the communication cost somehow?

Improvements?

- Can we reduce the communication cost somehow?
- Obvious improvement: Instead of broadcasting single rows and columns, do block rows and columns.

Improvements?

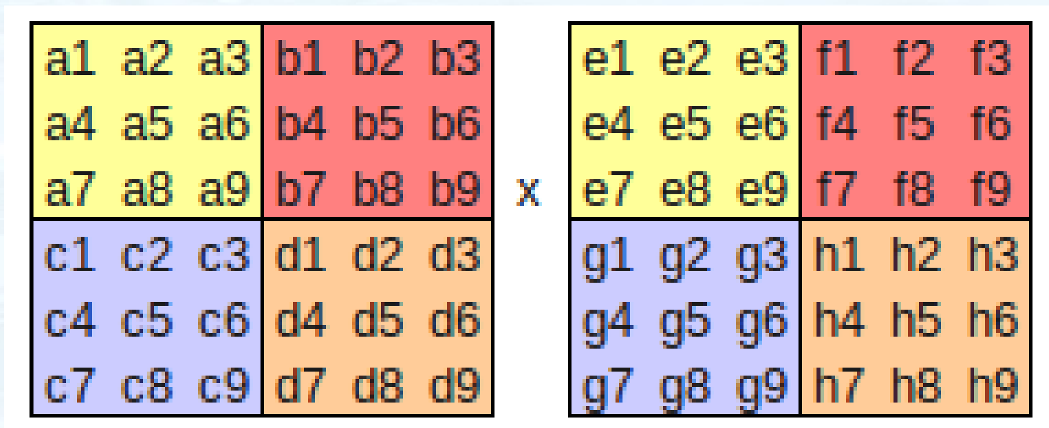
- Obvious improvement: Instead of broadcasting single rows and columns, do block rows and columns.
- Same amount of bytes communicated
- Fewer messages => less overhead
- More efficient (O(n^3) FLOPS, O(n^2) loads)

Pipelined SUMMA

- Another improvement:
- Instead of broadcast one row/column segment to all, pass multiple segments around in a ring
- Each process only communicates with neighbor => no broadcast => fewer messages

Pipelined SUMMA - Example

- Consider the following matrices A and B (color represents process rank)



Pipelined SUMMA - Example

- K=0: All processes multiply their first (block) column/row

*			*		
a1	a2	a3	**b1**	b2	b3
a4	a5	a6	**b4**	b5	b6
a7	a8	a9	**b7**	b8	b9
c1	c2	c3	**d1**	d2	d3
c4	c5	c6	**d4**	d5	d6
c7	c8	c9	**d7**	d8	d9

x

e1	**e2**	**e3**	**f1**	**f2**	**f3**	*
e4	e5	e6	f4	f5	f6	
e7	e8	e9	f7	f8	f9	
g1	**g2**	**g3**	**h1**	**h2**	**h3**	*
g4	g5	g6	h4	h5	h6	
g7	g8	g9	h7	h8	h9	

(for reference)

a1	a2	a3	b1	b2	b3
a4	a5	a6	b4	b5	b6
a7	a8	a9	b7	b8	b9
c1	c2	c3	d1	d2	d3
c4	c5	c6	d4	d5	d6
c7	c8	c9	d7	d8	d9

x

e1	e2	e3	f1	f2	f3
e4	e5	e6	f4	f5	f6
e7	e8	e9	f7	f8	f9
g1	g2	g3	h1	h2	h3
g4	g5	g6	h4	h5	h6
g7	g8	g9	h7	h8	h9

Pipelined SUMMA - Example

- K=1: All processes send their first (block) col in A to the right, their first (block) row in B down in a ring pattern

	*			*	
b1	**a2**	a3	a1	**b2**	b3
b4	**a5**	a6	a4	**b5**	b6
b7	**a8**	a9	a7	**b8**	b9
d1	**c2**	c3	c1	**d2**	d3
d4	**c5**	c6	c4	**d5**	d6
d7	**c8**	c9	c7	**d8**	d9

x

g1	g2	g3	h1	h2	h3	
e4	**e5**	**e6**	**f4**	**f5**	**f6**	*
e7	e8	e9	f7	f8	f9	
e1	e2	e3	f1	f2	f3	
g4	**g5**	**g6**	**h4**	**h5**	**h6**	*
g7	g8	g9	h7	h8	h9	

(for reference)

a1	a2	a3	b1	b2	b3
a4	a5	a6	b4	b5	b6
a7	a8	a9	b7	b8	b9
c1	c2	c3	d1	d2	d3
c4	c5	c6	d4	d5	d6
c7	c8	c9	d7	d8	d9

x

e1	e2	e3	f1	f2	f3
e4	e5	e6	f4	f5	f6
e7	e8	e9	f7	f8	f9
g1	g2	g3	h1	h2	h3
g4	g5	g6	h4	h5	h6
g7	g8	g9	h7	h8	h9

Pipelined SUMMA - Example

- And so on

	*			*									
b1	**a2**	a3	a1	**b2**	b3		g1	g2	g3	h1	h2	h3	
b4	**a5**	a6	a4	**b5**	b6		**e4**	**e5**	**e6**	**f4**	**f5**	**f6**	*
b7	**a8**	a9	a7	**b8**	b9	x	e7	e8	e9	f7	f8	f9	
d1	**c2**	c3	c1	**d2**	d3		e1	e2	e3	f1	f2	f3	
d4	**c5**	c6	c4	**d5**	d6		**g4**	**g5**	**g6**	**h4**	**h5**	**h6**	*
d7	**c8**	c9	c7	**d8**	d9		g7	g8	g9	h7	h8	h9	

	*			*									
b1	**b2**	b3	a1	**a2**	a3		g1	g2	g3	h1	h2	h3	
b4	**b5**	b6	a4	**a5**	a6		**g4**	**g5**	**g6**	**h4**	**h5**	**h6**	*
b7	**b8**	b9	a7	**a8**	a9	x	g7	g8	g9	h7	h8	h9	
d1	**d2**	d3	c1	**c2**	c3		e1	e2	e3	f1	f2	f3	
d4	**d5**	d6	c4	**c5**	c6		**e4**	**e5**	**e6**	**f4**	**f5**	**f6**	*
d7	**d8**	d9	c7	**c8**	c9		e7	e8	e9	f7	f8	f9	

		*			*								
b1	b2	**a3**	a1	a2	**b3**		g1	g2	g3	h1	h2	h3	
b4	b5	**a6**	a4	a5	**b6**		g4	g5	g6	h4	h5	h6	
b7	b8	**a9**	a7	a8	**b9**	x	**e7**	**e8**	**e9**	**f7**	**f8**	**f9**	*
d1	d2	**c3**	c1	c2	**d3**		e1	e2	e3	f1	f2	f3	
d4	d5	**c6**	c4	c5	**d6**		e4	e5	e6	f4	f5	f6	
d7	d8	**c9**	c7	c8	**d9**		**g7**	**g8**	**g9**	**h7**	**h8**	**h9**	*

		*			*								
b1	b2	**b3**	a1	a2	**a3**		g1	g2	g3	h1	h2	h3	
b4	b5	**b6**	a4	a5	**a6**		g4	g5	g6	h4	h5	h6	
b7	b8	**b9**	a7	a8	**a9**	x	**g7**	**g8**	**g9**	**h7**	**h8**	**h9**	*
d1	d2	**d3**	c1	c2	**c3**		e1	e2	e3	f1	f2	f3	
d4	d5	**d6**	c4	c5	**c6**		e4	e5	e6	f4	f5	f6	
d7	d8	**d9**	c7	c8	**c9**		**e7**	**e8**	**e9**	**f7**	**f8**	**f9**	*

*			*										
b1	b2	b3	**a1**	a2	a3		**g1**	**g2**	**g3**	**h1**	**h2**	**h3**	*
b4	b5	b6	**a4**	a5	a6		g4	g5	g6	h4	h5	h6	
b7	b8	b9	**a7**	a8	a9	x	g7	g8	g9	h7	h8	h9	
d1	d2	d3	**c1**	c2	c3		**e1**	**e2**	**e3**	**f1**	**f2**	**f3**	*
d4	d5	d6	**c4**	c5	c6		e4	e5	e6	f4	f5	f6	
d7	d8	d9	**c7**	c8	c9		e7	e8	e9	f7	f8	f9	

(for reference)

a1	a2	a3	b1	b2	b3		e1	e2	e3	f1	f2	f3
a4	a5	a6	b4	b5	b6		e4	e5	e6	f4	f5	f6
a7	a8	a9	b7	b8	b9	x	e7	e8	e9	f7	f8	f9
c1	c2	c3	d1	d2	d3		g1	g2	g3	h1	h2	h3
c4	c5	c6	d4	d5	d6		g4	g5	g6	h4	h5	h6
c7	c8	c9	d7	d8	d9		g7	g8	g9	h7	h8	h9