

Dense Linear Algebra

David Bindel

20 Oct 2015

Matrix vector product

Simple $y = Ax$ involves two indices

$$y_i = \sum_j A_{ij}x_j$$

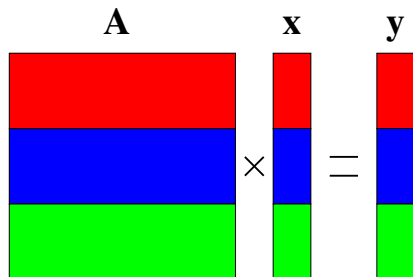
Can organize around either one:

```
% Row-oriented
for i = 1:n
    y(i) = A(i,:) * x;
end
```

```
% Col-oriented
y = 0;
for j = 1:n
    y = y + A(:,j) * x(j);
end
```

... or deal with index space in other ways!

Parallel matvec: 1D row-blocked



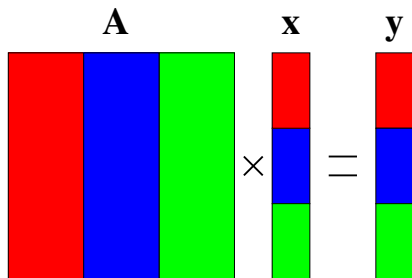
Receive broadcast x_0, x_1, x_2 into local x_0, x_1, x_2 ; then

$$\text{On } P_0: A_{00}x_0 + A_{01}x_1 + A_{02}x_2 = y_0$$

$$\text{On } P_1: A_{10}x_0 + A_{11}x_1 + A_{12}x_2 = y_1$$

$$\text{On } P_2: A_{20}x_0 + A_{21}x_1 + A_{22}x_2 = y_2$$

Parallel matvec: 1D col-blocked

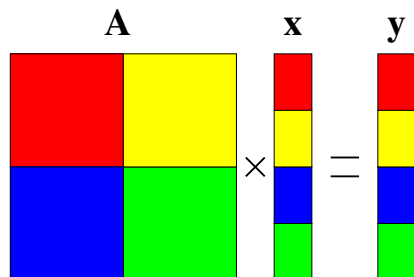


Independently compute

$$z^{(0)} = \begin{bmatrix} A_{00} \\ A_{10} \\ A_{20} \end{bmatrix} x_0 \quad z^{(1)} = \begin{bmatrix} A_{00} \\ A_{10} \\ A_{20} \end{bmatrix} x_1 \quad z^{(2)} = \begin{bmatrix} A_{00} \\ A_{10} \\ A_{20} \end{bmatrix} x_2$$

and perform reduction: $y = z^{(0)} + z^{(1)} + z^{(2)}$.

Parallel matvec: 2D blocked



- ▶ Involves broadcast *and* reduction
- ▶ ... but with subsets of processors

Parallel matvec: 2D blocked

Broadcast x_0, x_1 to local copies x_0, x_1 at **P0** and **P2**

Broadcast x_2, x_3 to local copies x_2, x_3 at **P1** and **P3**

In parallel, compute

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} z_0^{(0)} \\ z_1^{(0)} \end{bmatrix}$$

$$\begin{bmatrix} A_{02} & A_{03} \\ A_{12} & A_{13} \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} z_0^{(1)} \\ z_1^{(1)} \end{bmatrix}$$

$$\begin{bmatrix} A_{20} & A_{21} \\ A_{30} & A_{31} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} z_2^{(3)} \\ z_3^{(3)} \end{bmatrix}$$

$$\begin{bmatrix} A_{20} & A_{21} \\ A_{30} & A_{31} \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} z_2^{(3)} \\ z_3^{(3)} \end{bmatrix}$$

Reduce across rows:

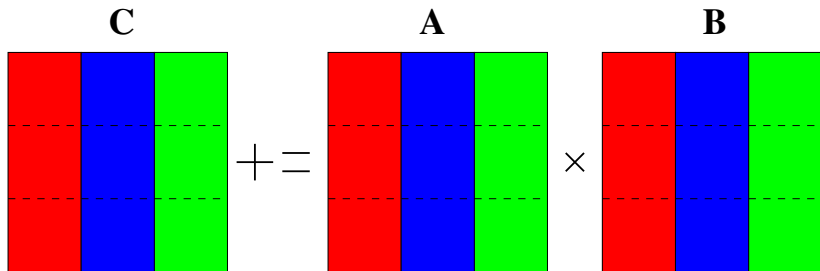
$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} z_0^{(0)} \\ z_0^{(0)} \\ z_1^{(0)} \\ z_1^{(0)} \end{bmatrix} + \begin{bmatrix} z_0^{(1)} \\ z_1^{(1)} \end{bmatrix}$$

$$\begin{bmatrix} y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} z_2^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ z_3^{(2)} \end{bmatrix} + \begin{bmatrix} z_2^{(3)} \\ z_3^{(3)} \\ z_2^{(3)} \\ z_3^{(3)} \end{bmatrix}$$

Parallel matmul

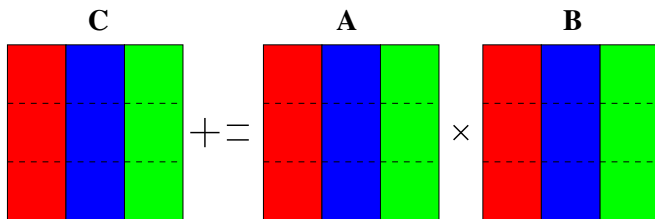
- ▶ Basic operation: $C = C + AB$
- ▶ Computation: $2n^3$ flops
- ▶ Goal: $2n^3/p$ flops per processor, minimal communication

1D layout



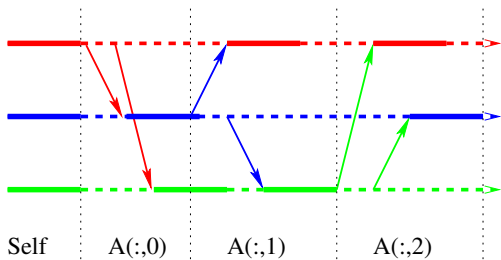
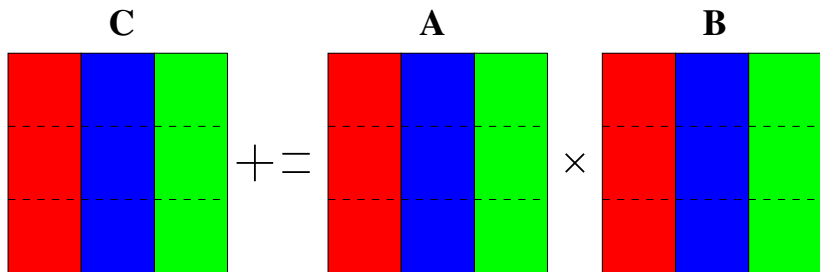
- ▶ Block MATLAB notation: $A(:, j)$ means j th block column
- ▶ Processor j owns $A(:, j)$, $B(:, j)$, $C(:, j)$
- ▶ $C(:, j)$ depends on *all* of A , but only $B(:, j)$
- ▶ How do we communicate pieces of A ?

1D layout on bus (no broadcast)



- ▶ Everyone computes local contributions first
- ▶ **P0** sends $A(:, 0)$ to each processor j in turn; processor j receives, computes $A(:, 0)B(0, j)$
- ▶ **P1** sends $A(:, 1)$ to each processor j in turn; processor j receives, computes $A(:, 1)B(1, j)$
- ▶ **P2** sends $A(:, 2)$ to each processor j in turn; processor j receives, computes $A(:, 2)B(2, j)$

1D layout on bus (no broadcast)



1D layout on bus (no broadcast)

```
C(:,myproc) += A(:,myproc)*B(myproc,myproc)
for i = 0:p-1
  for j = 0:p-1
    if (i == j)      continue;
    if (myproc == i) i
      send A(:,i) to processor j
    if (myproc == j)
      receive A(:,i) from i
      C(:,myproc) += A(:,i)*B(i,myproc)
    end
  end
end
end
```

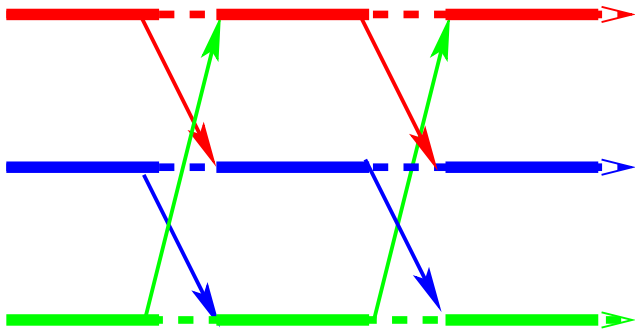
Performance model?

1D layout on bus (no broadcast)

No overlapping communications, so in a simple $\alpha - \beta$ model:

- ▶ $p(p - 1)$ messages
- ▶ Each message involves n^2/p data
- ▶ Communication cost: $p(p - 1)\alpha + (p - 1)n^2\beta$

1D layout on ring



- ▶ Every process j can send data to $j + 1$ simultaneously
- ▶ Pass slices of A around the ring until everyone sees the whole matrix ($p - 1$ phases).

1D layout on ring

```
tmp = A(myproc)
C(myproc) += tmp*B(myproc,myproc)
for j = 1 to p-1
    sendrecv tmp to myproc+1 mod p,
              from myproc-1 mod p
    C(myproc) += tmp*B(myproc-j mod p, myproc)
```

Performance model?

1D layout on ring

In a simple $\alpha - \beta$ model, at each processor:

- ▶ $p - 1$ message sends (and simultaneous receives)
- ▶ Each message involves n^2/p data
- ▶ Communication cost: $(p - 1)\alpha + (1 - 1/p)n^2\beta$

Outer product algorithm

Serial: Recall outer product organization:

```
for k = 0:s-1
    C += A(:,k)*B(k,:);
end
```

Parallel: Assume $p = s^2$ processors, block $s \times s$ matrices.
For a 2×2 example:

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00}B_{00} & A_{00}B_{01} \\ A_{10}B_{00} & A_{10}B_{01} \end{bmatrix} + \begin{bmatrix} A_{01}B_{10} & A_{01}B_{11} \\ A_{11}B_{10} & A_{11}B_{11} \end{bmatrix}$$

- ▶ Processor for each $(i, j) \implies$ parallel work for each k !
- ▶ Note everyone in row i uses $A(i, k)$ at once, and everyone in row j uses $B(k, j)$ at once.

Parallel outer product (SUMMA)

```
for k = 0:s-1
  for each i in parallel
    broadcast A(i,k) to row
  for each j in parallel
    broadcast A(k,j) to col
  On processor (i,j), C(i,j) += A(i,k)*B(k,j);
end
```

If we have tree along each row/column, then

- ▶ $\log(s)$ messages per broadcast
- ▶ $\alpha + \beta n^2/s^2$ per message
- ▶ $2 \log(s)(\alpha s + \beta n^2/s)$ total communication
- ▶ Compare to 1D ring: $(p-1)\alpha + (1-1/p)n^2\beta$

Note: Same ideas work with block size $b < n/s$

Cannon's algorithm

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00}B_{00} & A_{01}B_{11} \\ A_{11}B_{10} & A_{10}B_{01} \end{bmatrix} + \begin{bmatrix} A_{01}B_{10} & A_{00}B_{01} \\ A_{10}B_{00} & A_{11}B_{11} \end{bmatrix}$$

Idea: Reindex products in block matrix multiply

$$\begin{aligned} C(i, j) &= \sum_{k=0}^{p-1} A(i, k)B(k, j) \\ &= \sum_{k=0}^{p-1} A(i, k + i + j \bmod p) B(k + i + j \bmod p, j) \end{aligned}$$

For a fixed k , a given block of A (or B) is needed for contribution to *exactly one* $C(i, j)$.

Cannon's algorithm

```
% Move A(i, j) to A(i, i+j)
for i = 0 to s-1
    cycle A(i, :) left by i

% Move B(i, j) to B(i+j, j)
for j = 0 to s-1
    cycle B(:, j) up by j

for k = 0 to s-1
    in parallel;
        C(i, j) = C(i, j) + A(i, j)*B(i, j);
    cycle A(:, i) left by 1
    cycle B(:, j) up by 1
```

Cost of Cannon

- ▶ Assume 2D torus topology
- ▶ Initial cyclic shifts: $\leq s$ messages each ($\leq 2s$ total)
- ▶ For each phase: 2 messages each ($2s$ total)
- ▶ Each message is size n^2/s^2
- ▶ Communication cost: $4s(\alpha + \beta n^2/s^2) = 4(\alpha s + \beta n^2/s)$
- ▶ This communication cost is optimal!
... but SUMMA is simpler, more flexible, almost as good

Speedup and efficiency

Recall

$$\text{Speedup} := t_{\text{serial}}/t_{\text{parallel}}$$

$$\text{Efficiency} := \text{Speedup}/p$$

Assuming no overlap of communication and computation, efficiencies are

$$\begin{array}{ll} \text{1D layout} & \left(1 + O\left(\frac{p}{n}\right)\right)^{-1} \\ \text{SUMMA} & \left(1 + O\left(\frac{\sqrt{p} \log p}{n}\right)\right)^{-1} \\ \text{Cannon} & \left(1 + O\left(\frac{\sqrt{p}}{n}\right)\right)^{-1} \end{array}$$

Review: Parallel matmul

- ▶ Basic operation: $C = C + AB$
- ▶ Computation: $2n^3$ flops
- ▶ Goal: $2n^3/p$ flops per processor, minimal communication
- ▶ Two main contenders: SUMMA and Cannon

Outer product algorithm

Serial: Recall outer product organization:

```
for k = 0:s-1
    C += A(:,k)*B(k,:);
end
```

Parallel: Assume $p = s^2$ processors, block $s \times s$ matrices.
For a 2×2 example:

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00}B_{00} & A_{00}B_{01} \\ A_{10}B_{00} & A_{10}B_{01} \end{bmatrix} + \begin{bmatrix} A_{01}B_{10} & A_{01}B_{11} \\ A_{11}B_{10} & A_{11}B_{11} \end{bmatrix}$$

- ▶ Processor for each $(i, j) \implies$ parallel work for each $k!$
- ▶ Note everyone in row i uses $A(i, k)$ at once, and everyone in row j uses $B(k, j)$ at once.

Parallel outer product (SUMMA)

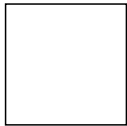
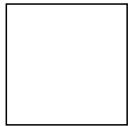
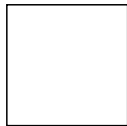
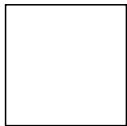
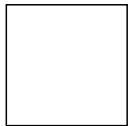
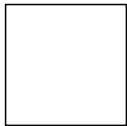
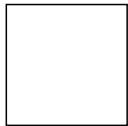
```
for k = 0:s-1
  for each i in parallel
    broadcast A(i,k) to row
  for each j in parallel
    broadcast A(k,j) to col
  On processor (i,j), C(i,j) += A(i,k)*B(k,j);
end
```

If we have tree along each row/column, then

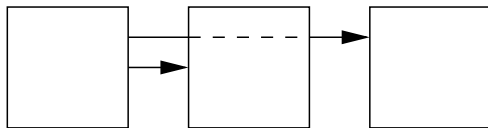
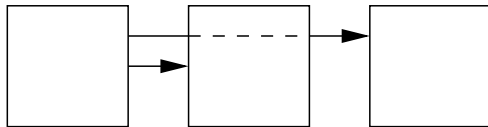
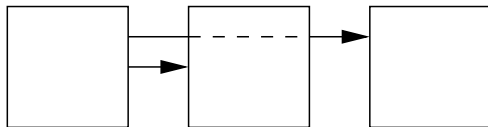
- ▶ $\log(s)$ messages per broadcast
- ▶ $\alpha + \beta n^2/s^2$ per message
- ▶ $2 \log(s)(\alpha s + \beta n^2/s)$ total communication
- ▶ Compare to 1D ring: $(p-1)\alpha + (1-1/p)n^2\beta$

Note: Same ideas work with block size $b < n/s$

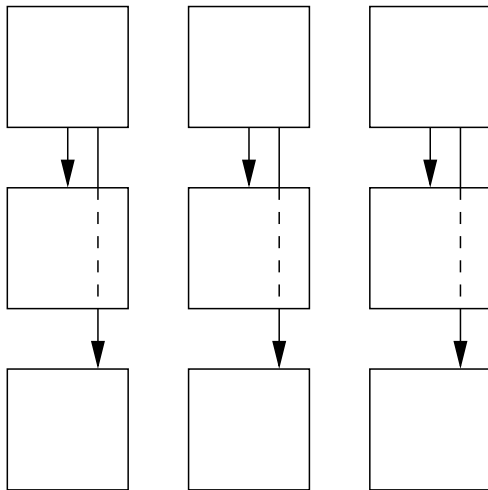
SUMMA



SUMMA



SUMMA



Parallel outer product (SUMMA)

If we have tree along each row/column, then

- ▶ $\log(s)$ messages per broadcast
- ▶ $\alpha + \beta n^2/s^2$ per message
- ▶ $2 \log(s)(\alpha s + \beta n^2/s)$ total communication

Assuming communication and computation can potentially overlap *completely*, what does the speedup curve look like?