NORTHWEST INSTITUTE *for* ADVANCED COMPUTING

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle for the U.S. Department of Energy

UNIVERSITY of WASHINGTON

AMATH 483/583
High Performance Scientific Computing

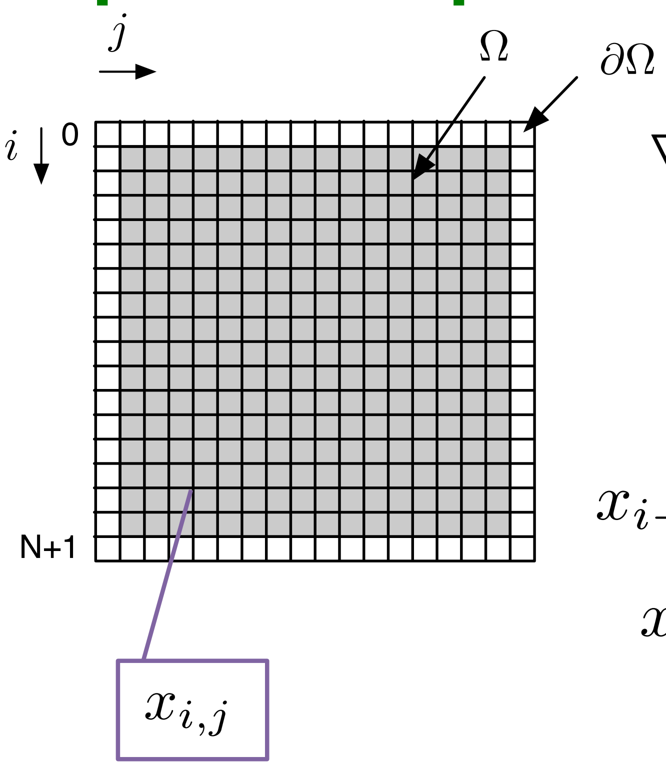
Lecture 20:
Advanced Message Passing, Cannon's Algorithm

Andrew Lumsdaine
Northwest Institute for Advanced Computing
Pacific Northwest National Laboratory
University of Washington
Seattle, WA

Administrative

- Fill out course evaluations!

Laplace's Equation on a Regular Grid



$$\begin{aligned} \nabla^2 \phi &= 0 \quad \text{on } \Omega \\ \phi &= f \quad \text{on } \partial\Omega \end{aligned}$$

Discretization

$$x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j} = 0$$

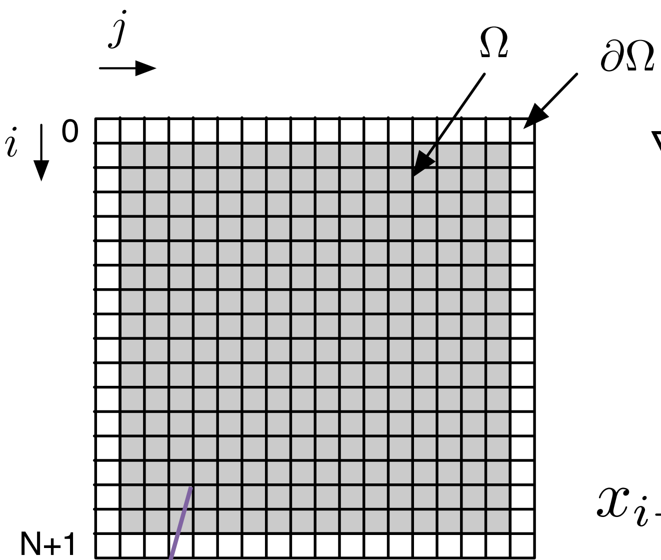
$$x_{i,j} = (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1})/4$$

The value of each point on the grid

The average of its neighbors

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & & \\ -1 & \ddots & \ddots & \ddots & \ddots & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 & \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots & \\ & \ddots & \ddots & \ddots & \ddots & -1 & \\ & & -1 & \cdots & -1 & 4 & \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

Jacobi Iteration



$$\begin{aligned} \nabla^2 \phi &= 0 \quad \text{on } \Omega \\ \phi &= f \quad \text{on } \partial\Omega \end{aligned}$$

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

Discretization

$$x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j} = 0$$

$$x_{i,j}^{k+1} = (x_{i-1,j}^k + x_{i+1,j}^k + x_{i,j-1}^k + x_{i,j+1}^k)/4$$

$x_{i,j}$

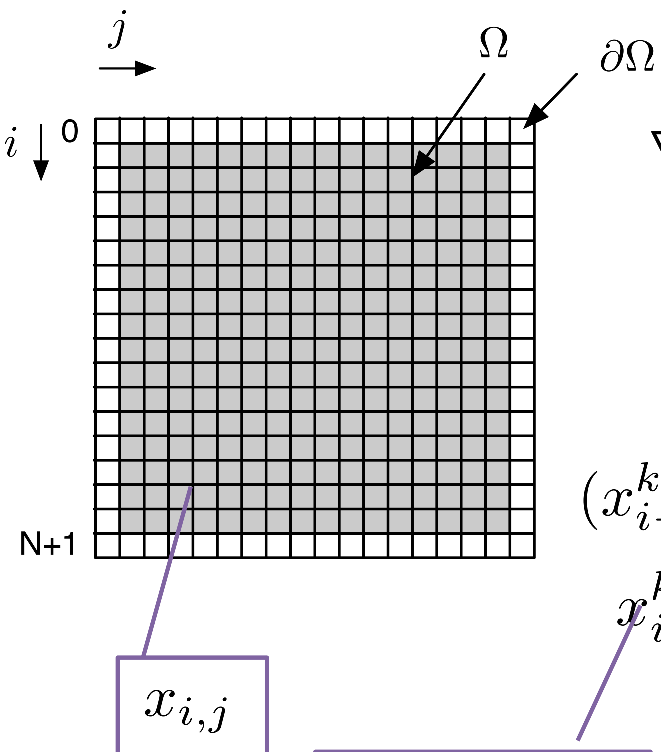
Iteration k+1

The value of each point on the grid

The average of its neighbors

Iteration k

Jacobi Iteration



$$\begin{aligned} \nabla^2 \phi &= 0 \quad \text{on } \Omega \\ \phi &= f \quad \text{on } \partial\Omega \end{aligned}$$

Discretization

$$(x^k_{i-1,j} + x^k_{i+1,j} + x^k_{i,j-1} + x^k_{i,j+1}) - 4x^{k+1}_{i,j} = 0$$

$$x^{k+1}_{i,j} = (x^k_{i-1,j} + x^k_{i+1,j} + x^k_{i,j-1} + x^k_{i,j+1})/4$$

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

Iteration k+1

The value of each point on the grid

The average of its neighbors

Iteration k

Jacobi Iteration

$Ax = b$

$$4x_{i,j} - (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}) = 0$$

A

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \end{bmatrix}$$

$A = M - N$

$$4x_{i,j}^{k+1} - (x_{i-1,j}^k + x_{i+1,j}^k + x_{i,j-1}^k + x_{i,j+1}^k) = 0$$

$$\frac{1}{h^2}\begin{bmatrix} 4 & 0 & \cdots & 0 & & \\ 0 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ & & 0 & \cdots & 0 & 4 \end{bmatrix}\begin{bmatrix} x_0^{k+1} \\ x_1^{k+1} \\ x_2^{k+1} \\ \vdots \end{bmatrix} - \frac{1}{h^2}\begin{bmatrix} 0 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 0 \end{bmatrix}\begin{bmatrix} x_0^k \\ x_1^k \\ x_2^k \\ \vdots \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \end{bmatrix}$$

M N

Jacobi Iteration

$$Ax = b$$

$$A$$

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \end{bmatrix}$$

$$4x_{i,j} - (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}) = 0$$

$$A = M - N$$

$$4x_{i,j}^{k+1} - (x_{i-1,j}^k + x_{i+1,j}^k + x_{i,j-1}^k + x_{i,j+1}^k) = 0$$

$$Mx^{k+1} - Nx^k = b$$

$$x_{i,j}^{k+1} = \frac{1}{4}(x_{i-1,j}^k + x_{i+1,j}^k + x_{i,j-1}^k + x_{i,j+1}^k)$$

$$x^{k+1} = M^{-1}(Nx^k + b)$$

Average of neighbors

Still a stencil application

class Grid

```
class Grid {

public:
  explicit Grid(size_t x, size_t y)                    :
      xPoints(x+2), yPoints(y+2), arrayData(xPoints*yPoints) {}

        double &operator()(size_t i, size_t j)
            { return arrayData[i*yPoints + j]; }
  const double &operator()(size_t i, size_t j) const
            { return arrayData[i*yPoints + j]; }

  size_t numX() const { return xPoints; }
  size_t numY() const { return yPoints; }

private:
  size_t xPoints, yPoints;
  std::vector<double> arrayData;
};
```

Grid is a 2D array

Constructor

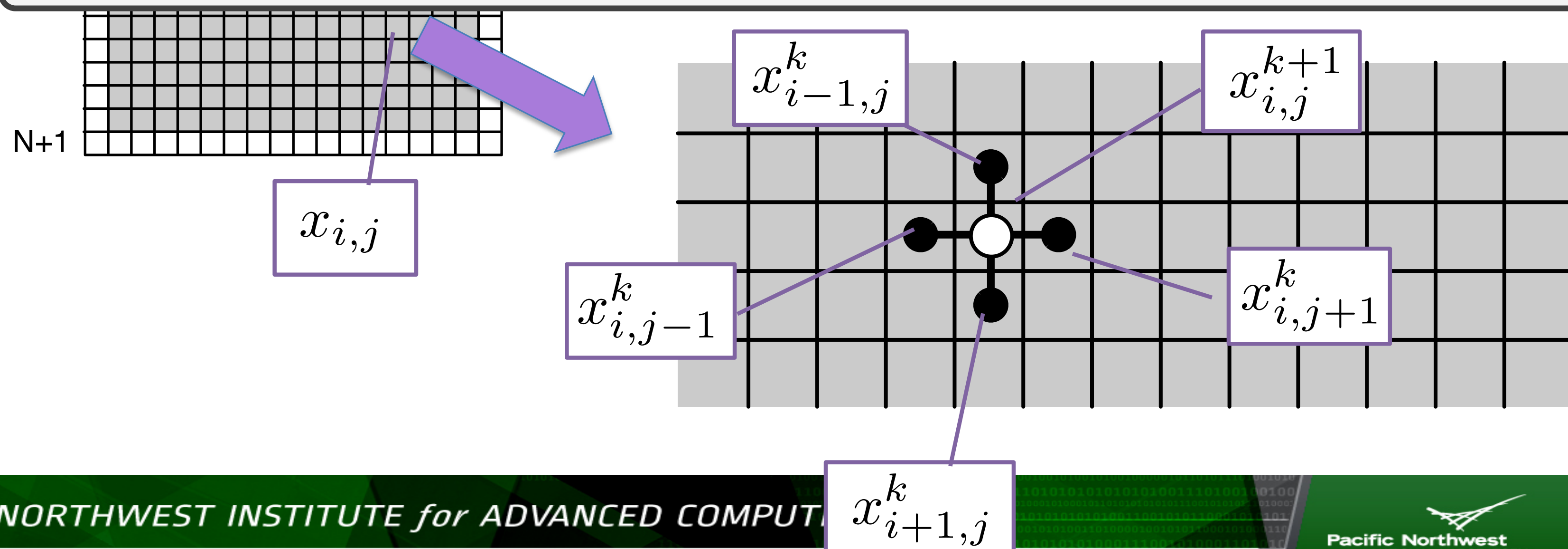
Accessor

Storage

Iterating for a solution

Claim: We only ever need two grids

```
while (! converged()) {
  for (size_t k = 0; k < max_k; ++k) {
    for (size_t i = 1; i < N+1; ++i)
      for (size_t j = 1; j < N+1; ++j)
        x[k+1](i,j) = (x[k](i-1,j) + x[k](i+1,j) + x[k](i,j-1) + x[k](i,j+1))/4.0;
}
```



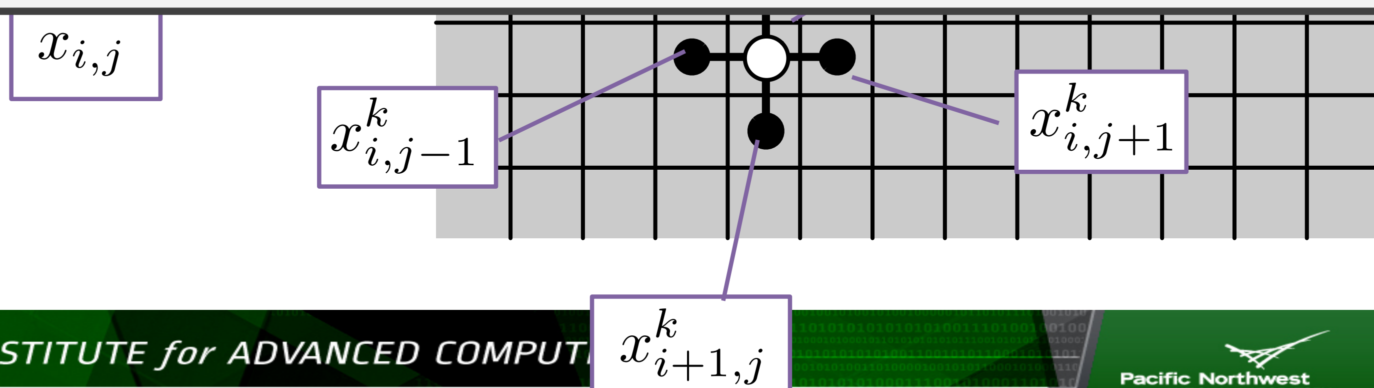
Iterating for a solution

```
while (! converged()) {
  for (size_t i = 1; i < N+1; ++i) {
    for (size_t j = 1; j < N+1; ++j) {
      xp(i,j) = (x(i-1,j) + x(i+1,j) + x(i,j-1) + x(i,j+1))/4.0;
    }
  }
  swap(xp, x);
}
```

Claim: We only ever need two grids

Make current the previous

Could copy instead, but…



Sequential

```
void jacobi(Grid& x, Grid& xp) {
  while (! converged()) {
    for (size_t i = 1; i < x.num_x()-1; ++i) {
      for (size_t j = 1; j < x.num_y()-1; ++j) {
        xp(i,j) = (x(i-1,j) + x(i+1,j) + x(i,j-1) + x(i,j+1))/4.0;
      }
    }
    swap(xp, x);
  }
}
```

Decomposition

Boundary

Boundary

One crucial difference

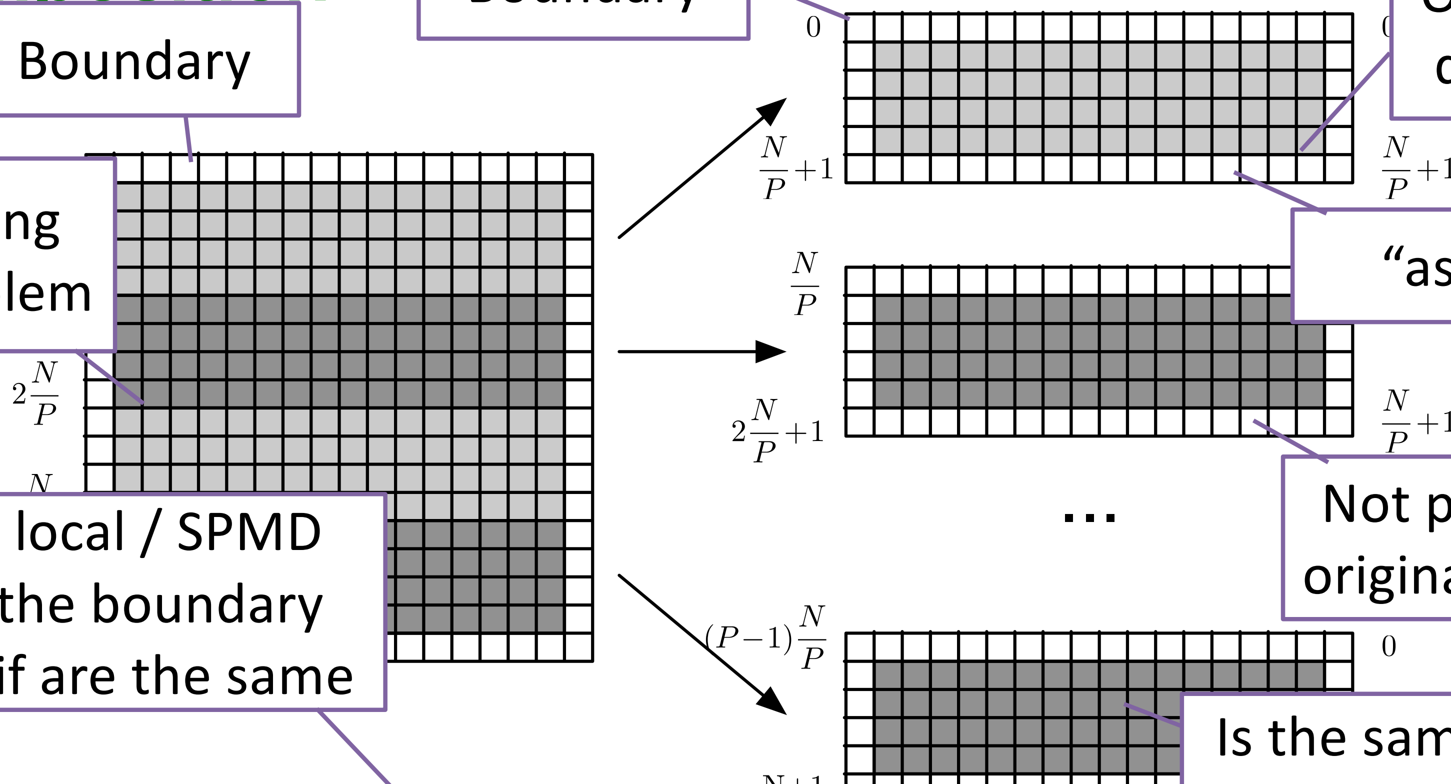
So solving this problem

"as-if"

Not part of the original problem

To the local / SPMD code, the boundary and as-if are the same

Is the same as solving lots of the same problem but smaller

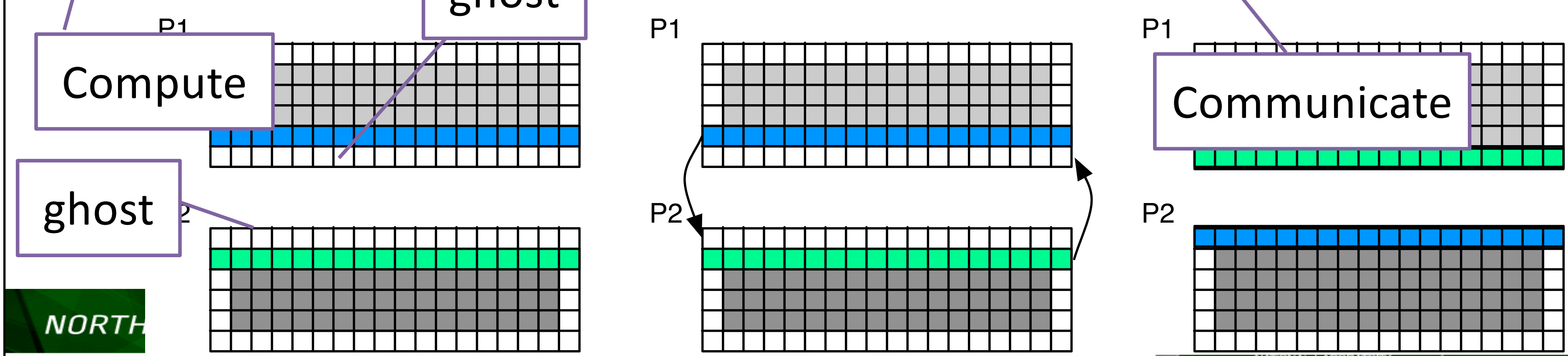



```
for (size_t i = 1; i < N/P+1; ++i)
  for (size_t j = 1; j < N+1; ++j)
    y(i,j) = (x(i-1,j) + x(i+1,j) + x(i,j-1) + x(i,j+1))/4.0;
```

Compute / Communicate

```
while (! converged()) {
  for (size_t i = 1; i < N+1; ++i)
    for (size_t j = 1; j < N+1; ++j)
      y(i,j) = (x(i-1,j) + x(i+1,j) + x(i,j-1) + x(i,j+1))/4.0;
  swap(x,y);
  make_as_if(x); // Communicate ghost cells
}
```

Standard terminology for as-if boundary is “ghost cell” or “halo”



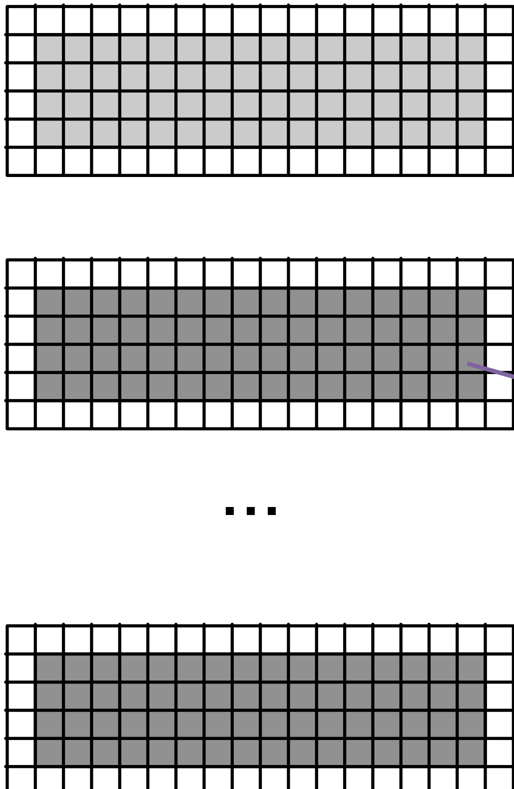
SPMD

```
void jacobi(Grid& x, Grid& xp) {
  while (! converged()) {
    for (size_t i = 1; i < x.num_x()-1; ++i) {
      for (size_t j = 1; j < x.num_y()-1; ++j) {
        xp(i,j) = (x(i-1,j) + x(i+1,j) + x(i,j-1) + x(i,j+1))/4.0;
      }
    }
    swap(xp, x);
  }
}
```

Or here

Communicate here

Decomposition



“myrank”

```
MPI::COMM_WORLD.Send(to myrank + 1)
MPI::COMM_WORLD.Send(to myrank − 1)
MPI::COMM_WORLD.Recv(from myrank − 1)
MPI::COMM_WORLD.Recv(from myrank + 1)
```

Which match?

Message sent ”up”

Received from below

```
MPI::COMM_WORLD.Send(to myrank + 1, uptag)
MPI::COMM_WORLD.Send(to myrank − 1, downtag)
MPI::COMM_WORLD.Recv(from myrank − 1, uptag)
MPI::COMM_WORLD.Recv(from myrank + 1, downtag)
```

Tags really Necessary?

Message sent ”up”

Received from below

Details

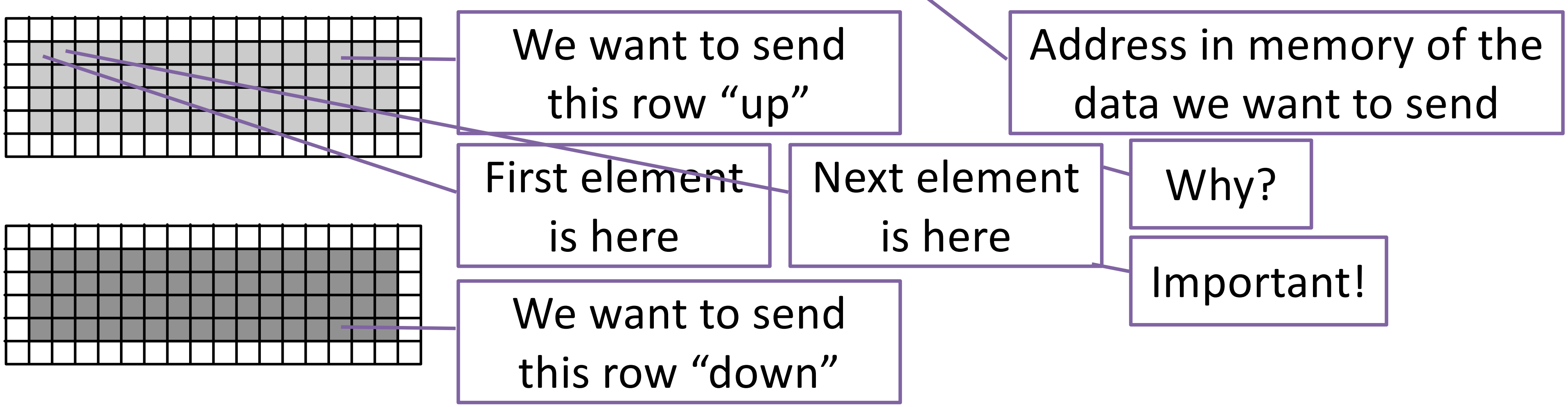
```
MPI::COMM_WORLD.Send(to myrank + 1)
MPI::COMM_WORLD.Send(to myrank - 1)
MPI::COMM_WORLD.Recv(from myrank - 1)
MPI::COMM_WORLD.Recv(from myrank + 1)
```

```
void Comm::Send(const void* buf, int count, const Datatype&
    datatype, int dest, int tag);
```

What are these actually?

Details

```
void Comm::Send(const void* buf, int count, const Datatype&
     datatype, int dest, int tag);
```



Details

```
void Comm::Send(const void* buf, int count, const Datatype&
    datatype, int dest, int tag);
```

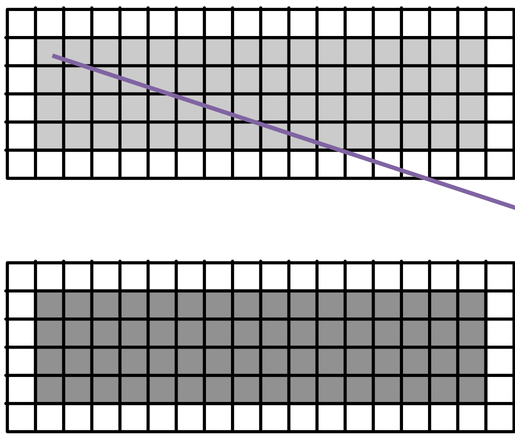
How many?

What type?

`x.num_y()`

`MPI::DOUBLE`

Address in memory of the data we want to send



First element is here

What is its address?

How do we access it?

`&x(1,1)`

`x(1,1)`

Address

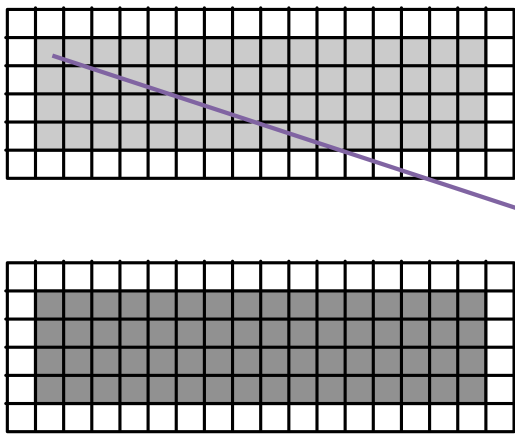
Details

```
void Comm::Send(const void* buf, int count, const Datatype&
    datatype, int dest, int tag);
```

How many? — `x.num_y()-2`

What type? — `MPI::DOUBLE`

Address in memory of the data we want to send



First element is here

What is its address?

How do we access it?

`&x(1,1)` — Address

`x(1,1)`

Alternatively

How many?

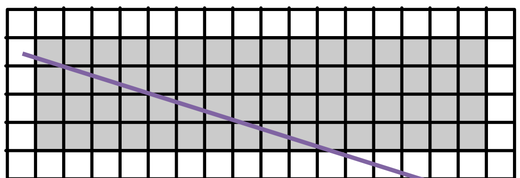
What type?

```
void Comm::Send(const void* buf, int count, const Datatype&
    datatype, int dest, int tag);
```

`x.num_y()`

`MPI::DOUBLE`

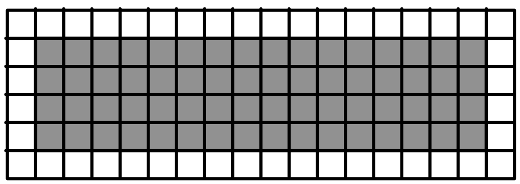
Address in memory of the data we want to send



First element is here

What is its address?

How do we access it?



`&x(1,0)`

`x(1,0)`

Address

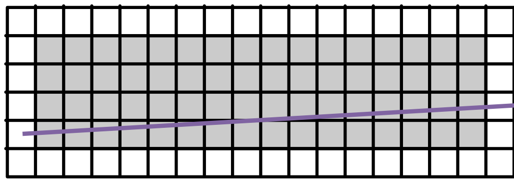
Sending "up"

May need const cast

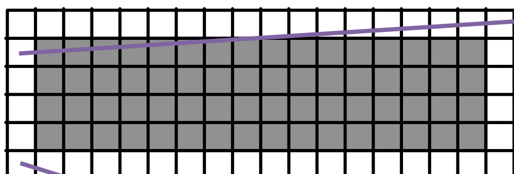
What is corresponding receive?

```
MPI::COMM_WORLD.Send(&x(1, 0)), x.num_y(), MPI::DOUBLE, myrank+1, uptag);
```

```
MPI::COMM_WORLD.Recv(&x(x.num_x()-1, 0)), x.num_y(), MPI::DOUBLE, myrank-1, uptag);
```



Send "down": First element is here



Receive "down": First element is here

Yes?

Need to handle top and bottom correctly

And not deadlock

First element is here

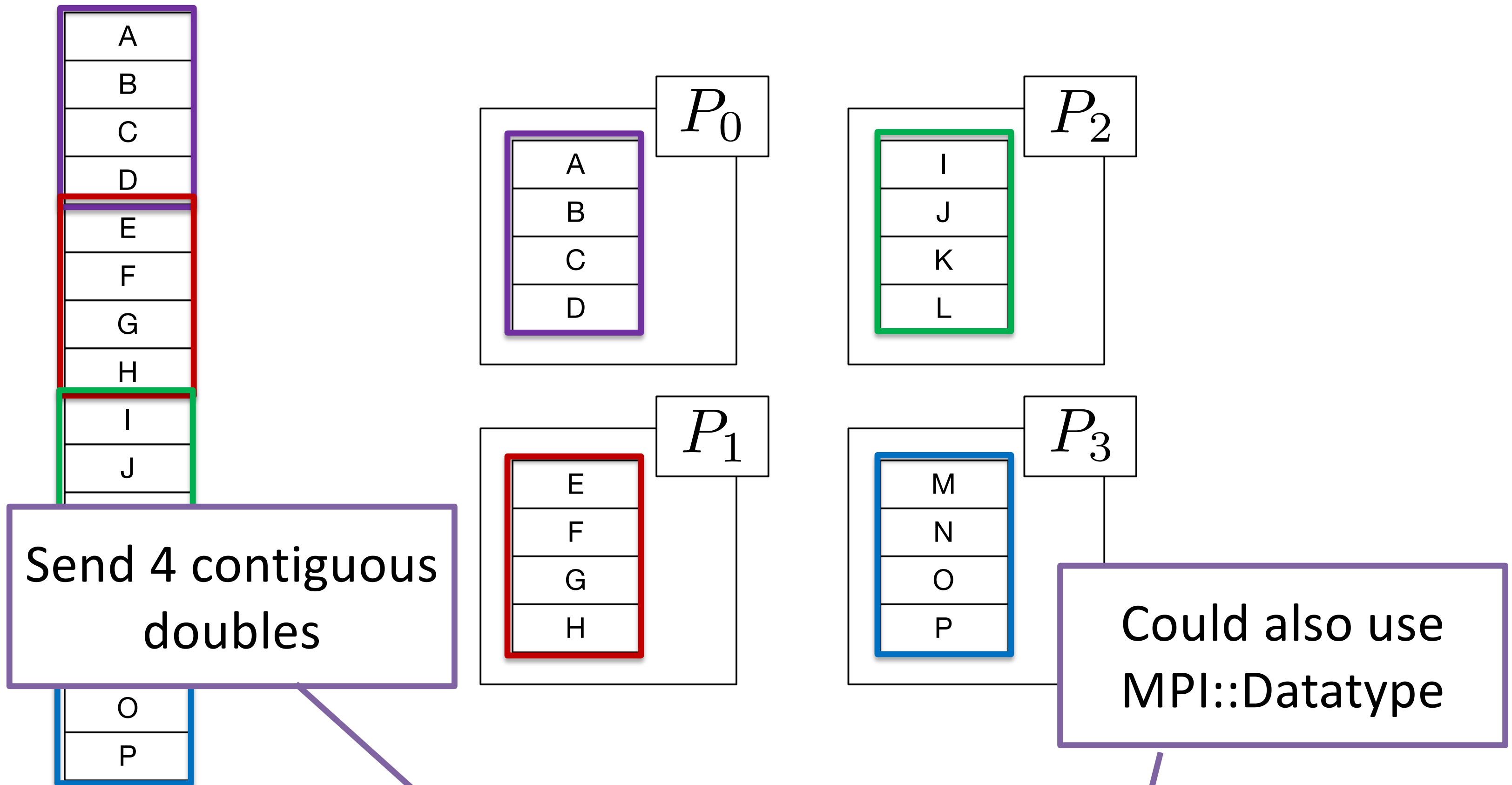
Same size and type

Same tag

Parallel Matrix-Matrix Multiply

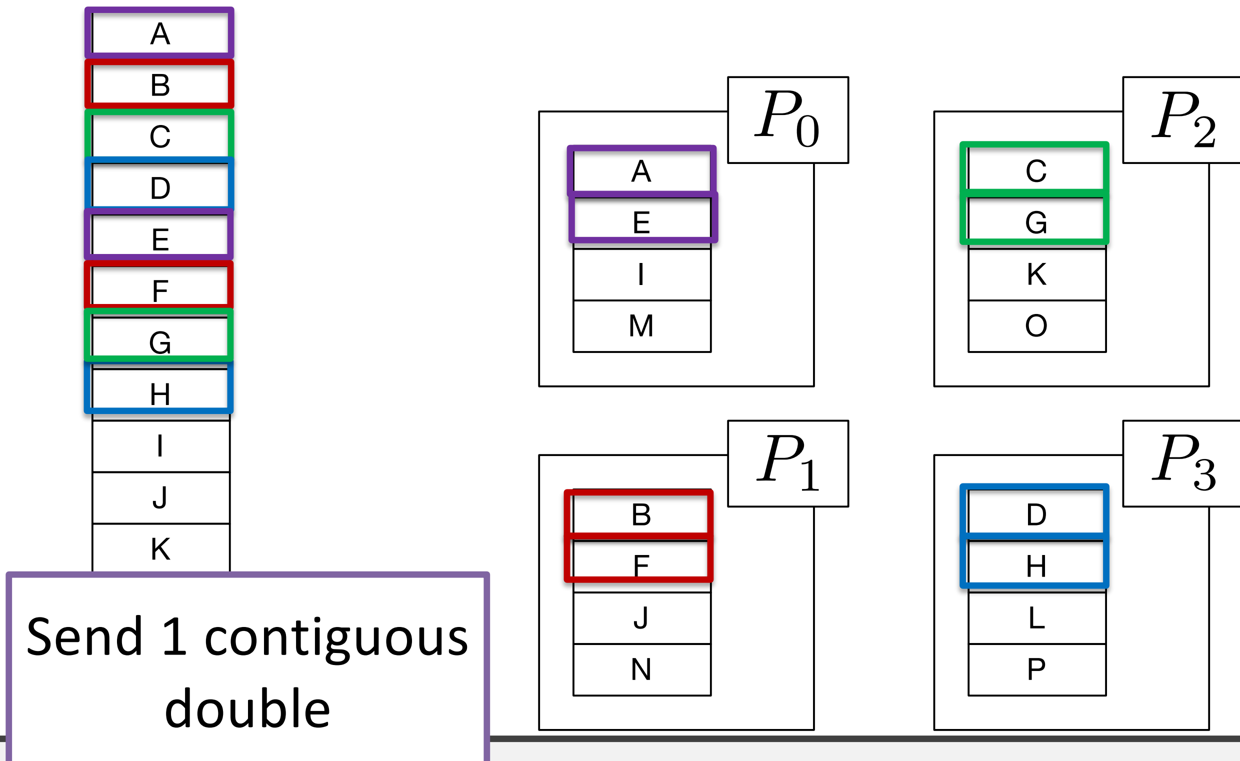
- Use block algorithm
- Partition matrix into blocks
- Assign blocks to processors
- Orchestrate communication and computation
- ***Owner computes***

Block Partitioning



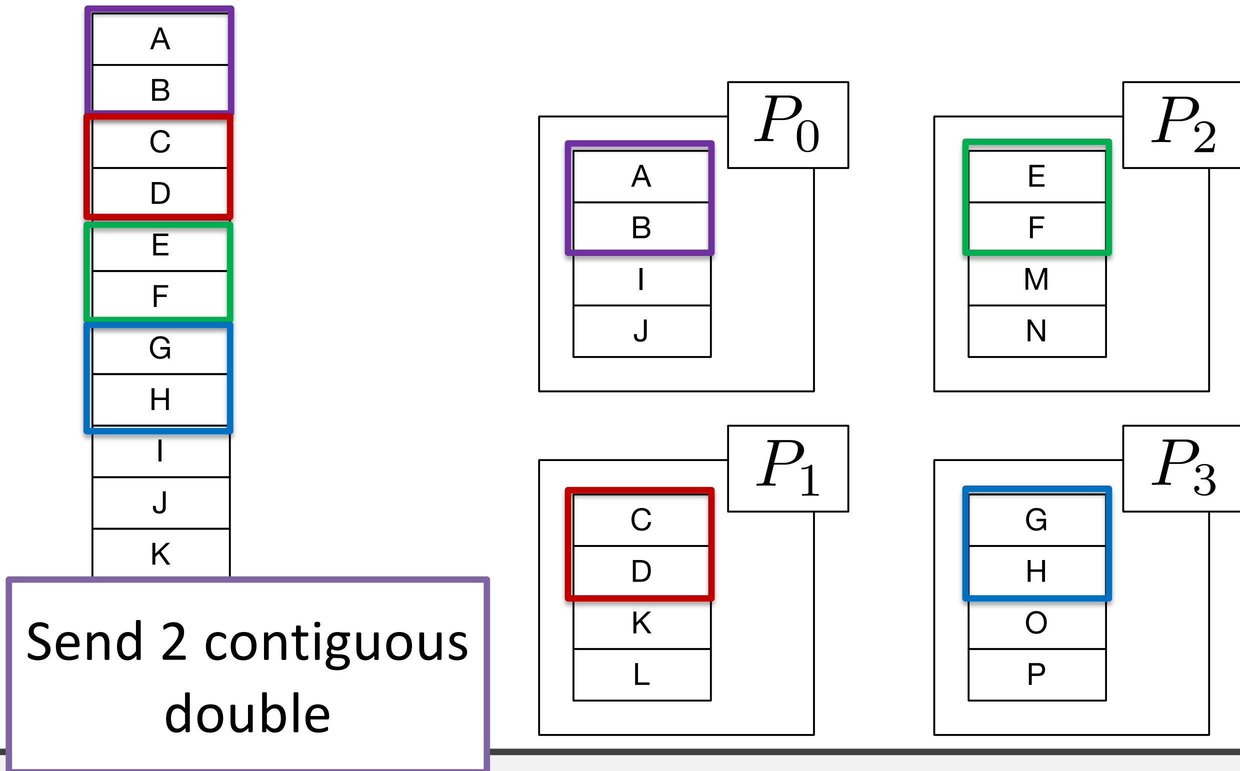

```
MPI::COMM_WORLD.Scatter(&x(0), 4, MPI::DOUBLE, &x(0), 4, MPI::DOUBLE, 0);
```

Cyclic Partitioning




```
for (size_t i = 0; i < 4; ++i) {
  MPI::COMM_WORLD.Scatter(&x(i*4), 1, MPI::DOUBLE, &x(i*4), 1, MPI::DOUBLE, 0);
}
```

Block Cyclic Partitioning




```
for (size_t i = 0; i < 2; ++i) {
  MPI::COMM_WORLD.Scatter(&x(i*8), 2, MPI::DOUBLE, &x(i*8), 2, MPI::DOUBLE, 0);
}
```

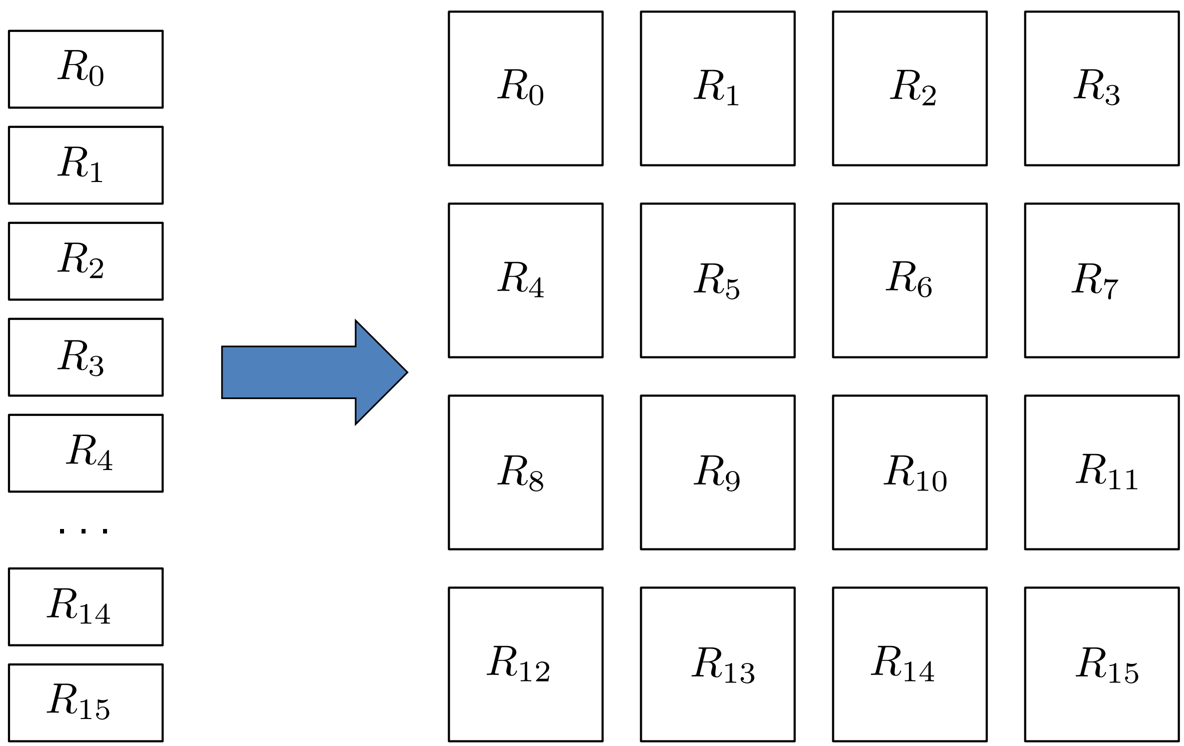
Block Matrix-Matrix Product

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

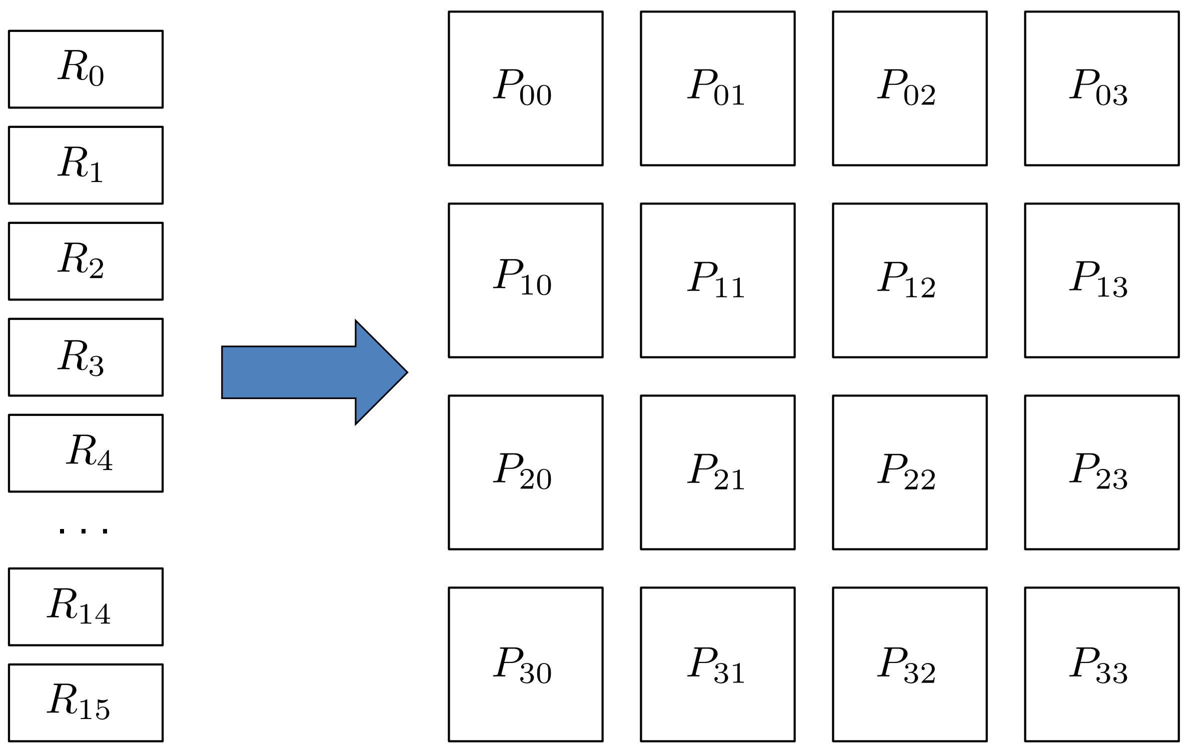
$$\begin{bmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \end{bmatrix}$$

$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

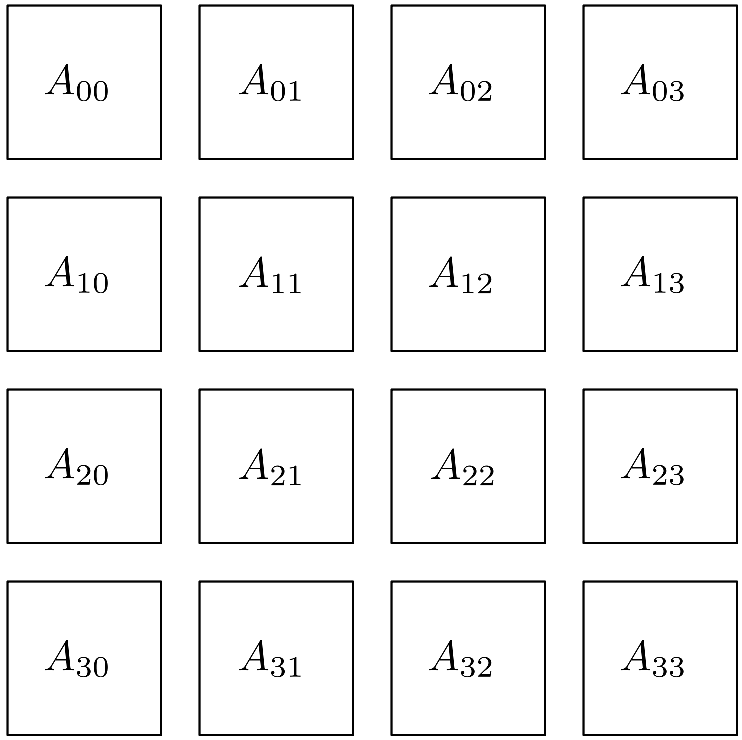
Processor Grid



Processor Grid



Matrix Block Partitioning



Matrix Block Partitioning

A_{00} B_{00}	A_{01} B_{01}	A_{02} B_{02}	A_{03} B_{03}
A_{10} B_{10}	A_{11} B_{11}	A_{12} B_{12}	A_{13} B_{13}
A_{20} B_{20}	A_{21} B_{21}	A_{22} B_{22}	A_{23} B_{23}
A_{30} B_{30}	A_{31} B_{31}	A_{32} B_{32}	A_{33} B_{33}

Matrix Block Partitioning

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

Matrix Block Partitioning

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$ (Owner computes)

$$C_{21} = A_{20} B_{01} + A_{21} B_{11} + A_{22} B_{21} + A_{23} B_{31}$$

Matrix Block Partitioning

$$C_{IJ} = \sum_{K} A_{IK} B_{KJ}$$ (Owner computes)

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

Matrix Block Partitioning

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$ (Owner computes)

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

Matrix Block Partitioning

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$ (Owner computes)

$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

Matrix Block Partitioning

$$C_{IJ} = \sum_{K} A_{IK} B_{KJ}$$ (Owner computes)

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

Matrix Block Partitioning

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{10} B_{10}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{12}	C_{13} A_{13} B_{13}
C_{20} A_{20} B_{20}	C_{21} A_{21} B_{21}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{23}
C_{30} A_{30} B_{30}	C_{31} A_{31} B_{31}	C_{32} A_{32} B_{32}	C_{33} A_{33} B_{33}

$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

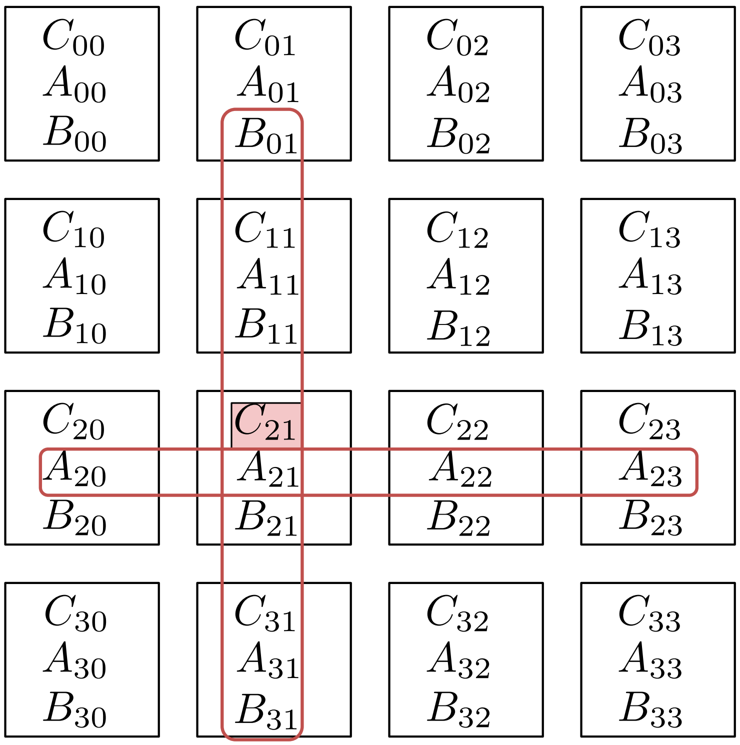
$$C_{IJ} = \sum_K A_{IK}B_{KJ}$$ (Owner computes)

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J

Cannon's Algorithm



$$C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}$$

$$C_{IJ} = \sum_K A_{IK}B_{KJ}$$

- At each step K, arrange for

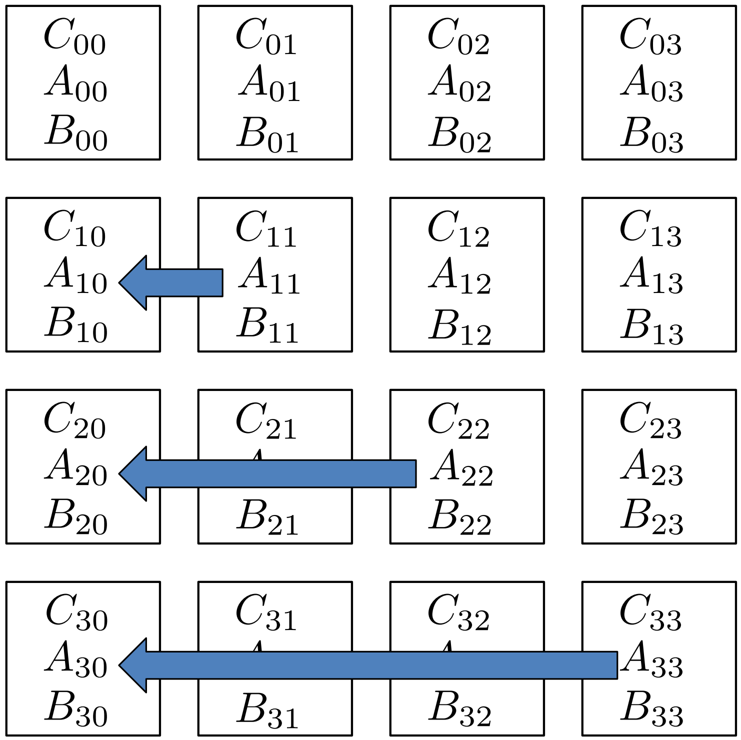
  $A_{I,(I+J+K)}$ $\quad$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: Setup (K = 0)



$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: Setup (K = 0)

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{01}	C_{02} A_{02} B_{02}	C_{03} A_{03} B_{03}
C_{10} A_{11} B_{10}	C_{11} A_{12} B_{11}	C_{12} A_{13} B_{12}	C_{13} A_{10} B_{13}
C_{20} A_{22} B_{20}	C_{21} A_{23} B_{21}	C_{22} A_{20} B_{22}	C_{23} A_{21} B_{23}
C_{30} A_{33} B_{30}	C_{31} A_{30} B_{31}	C_{32} A_{31} B_{32}	C_{33} A_{32} B_{33}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

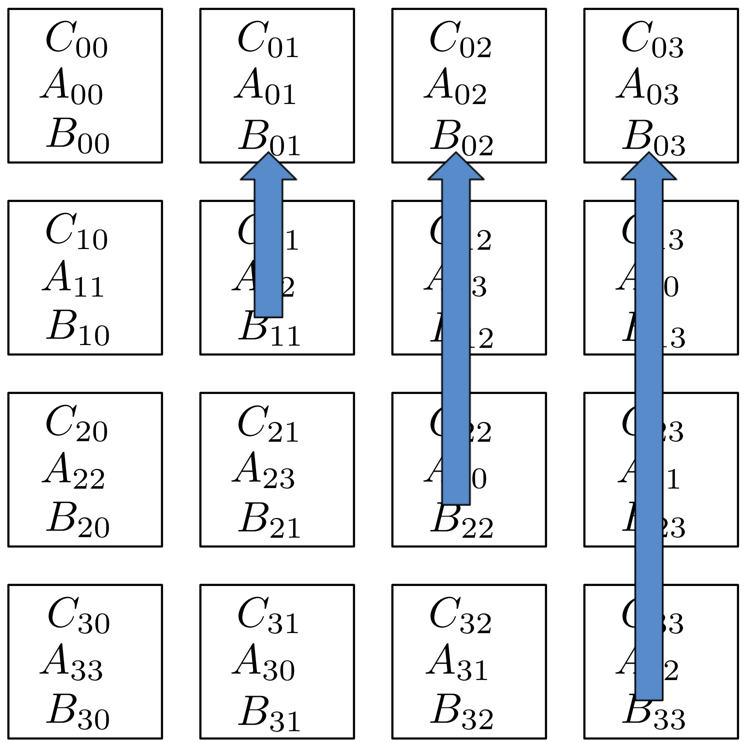
  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: Setup (K = 0)



$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

$$A_{I,(I+J+K)} \qquad B_{(I+J+K),J}$$

  to be on processor I,J

- Compute

$$C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$$

Cannon's Algorithm: Setup

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{11}	C_{02} A_{02} B_{22}	C_{03} A_{03} B_{33}
C_{10} A_{11} B_{10}	C_{11} A_{12} B_{21}	C_{12} A_{13} B_{32}	C_{13} A_{10} B_{03}
C_{20} A_{22} B_{20}	C_{21} A_{23} B_{31}	C_{22} A_{20} B_{02}	C_{23} A_{21} B_{13}
C_{30} A_{33} B_{30}	C_{31} A_{30} B_{01}	C_{32} A_{31} B_{12}	C_{33} A_{32} B_{23}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J
- Compute

$$C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$$

Cannon's Algorithm: K = 0

C_{00} A_{00} B_{00}	C_{01} A_{01} B_{11}	C_{02} A_{02} B_{22}	C_{03} A_{03} B_{33}
C_{10} A_{11} B_{10}	C_{11} A_{12} B_{21}	C_{12} A_{13} B_{32}	C_{13} A_{10} B_{03}
C_{20} A_{22} B_{20}	C_{21} A_{23} B_{31}	C_{22} A_{20} B_{02}	C_{23} A_{21} B_{13}
C_{30} A_{33} B_{30}	C_{31} A_{30} B_{01}	C_{32} A_{31} B_{12}	C_{33} A_{32} B_{23}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

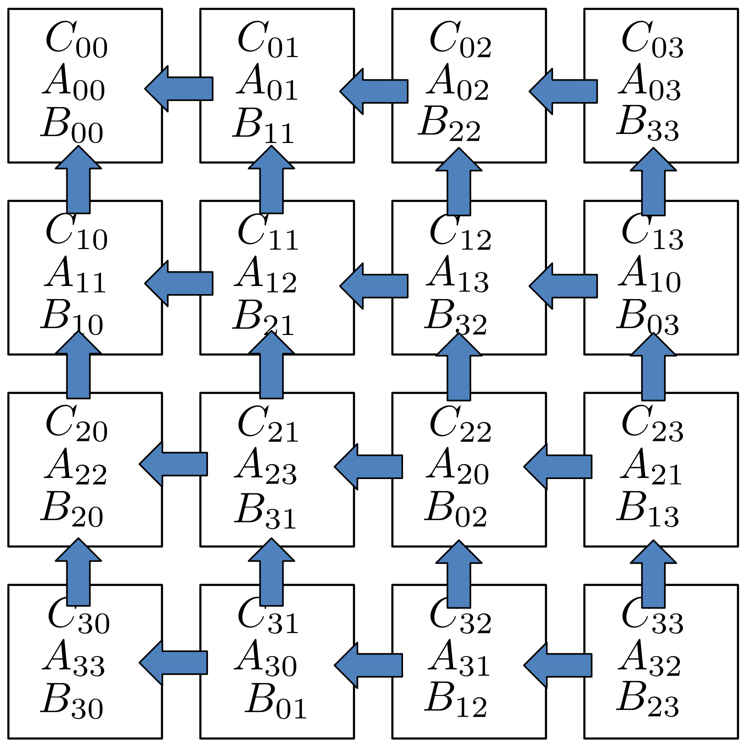
  $A_{I,(I+J+K)}$ $\quad$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: K = 1



$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $\quad$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: K = 1

C_{00} A_{01} B_{10}	C_{01} A_{02} B_{21}	C_{02} A_{03} B_{32}	C_{03} A_{00} B_{03}
C_{10} A_{12} B_{20}	C_{11} A_{13} B_{31}	C_{12} A_{10} B_{02}	C_{13} A_{11} B_{13}
C_{20} A_{23} B_{30}	C_{21} A_{20} B_{01}	C_{22} A_{21} B_{12}	C_{23} A_{22} B_{23}
C_{30} A_{30} B_{00}	C_{31} A_{31} B_{11}	C_{32} A_{32} B_{22}	C_{33} A_{33} B_{33}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

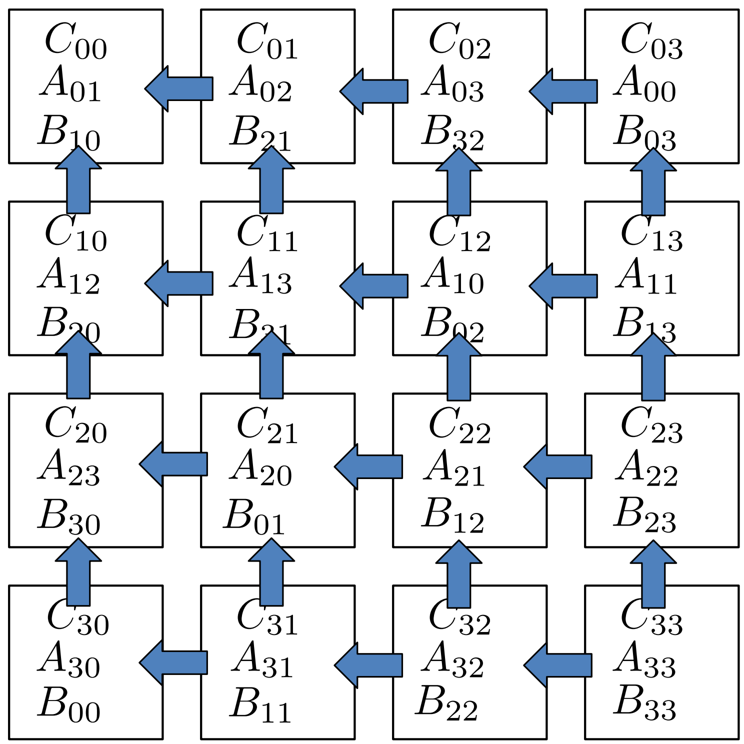
  $A_{I,(I+J+K)}$ $\quad$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: K = 2



$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: K = 2

C_{00} A_{02} B_{20}	C_{01} A_{03} B_{31}	C_{02} A_{00} B_{02}	C_{03} A_{01} B_{13}
C_{10} A_{13} B_{30}	C_{11} A_{10} B_{01}	C_{12} A_{11} B_{12}	C_{13} A_{12} B_{23}
C_{20} A_{20} B_{00}	C_{21} A_{21} B_{11}	C_{22} A_{22} B_{22}	C_{23} A_{23} B_{33}
C_{30} A_{31} B_{10}	C_{31} A_{32} B_{21}	C_{32} A_{33} B_{32}	C_{33} A_{30} B_{03}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

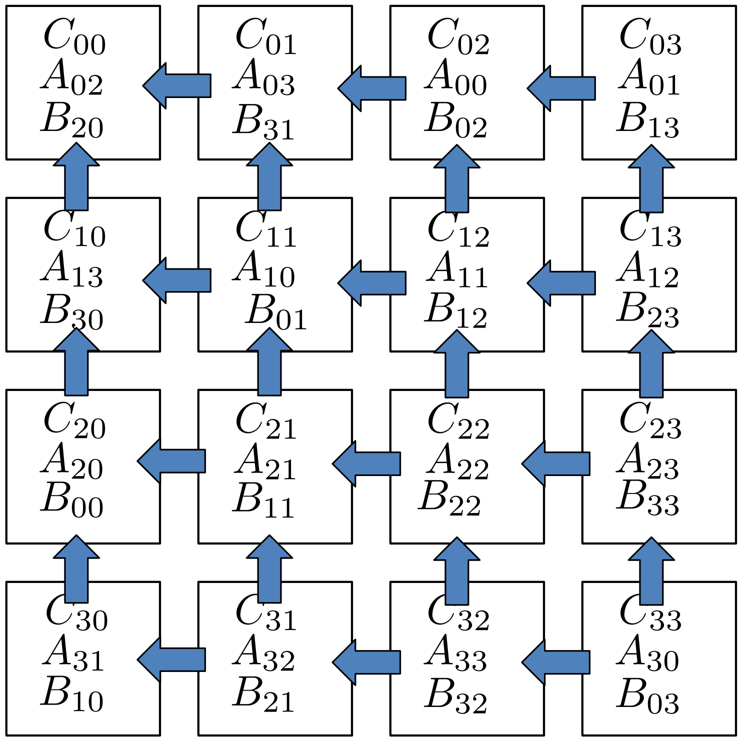
  $A_{I,(I+J+K)}$ $\qquad$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: K = 3



$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Cannon's Algorithm: K = 3

C_{00} A_{03} B_{30}	C_{01} A_{00} B_{01}	C_{02} A_{01} B_{12}	C_{03} A_{02} B_{23}
C_{10} A_{10} B_{00}	C_{11} A_{11} B_{11}	C_{12} A_{12} B_{22}	C_{13} A_{13} B_{33}
C_{20} A_{21} B_{10}	C_{21} A_{22} B_{21}	C_{22} A_{23} B_{32}	C_{23} A_{20} B_{03}
C_{30} A_{32} B_{20}	C_{31} A_{33} B_{31}	C_{32} A_{30} B_{02}	C_{33} A_{31} B_{13}

$$C_{IJ} = \sum_K A_{IK} B_{KJ}$$

- At each step K, arrange for

  $A_{I,(I+J+K)}$ $B_{(I+J+K),J}$

  to be on processor I,J

- Compute

  $C_{IJ} \mathrel{+}= A_{I,(I+J+K)} \times B_{(I+J+K),J}$

Implementation

- Two-D decomposition of matrices A, B, C
- Move A and B to starting positions
  - Local matrix-matrix product
  - Shift left
  - Shift up
- Move A and B back to initial distributions

MPI Mental Model

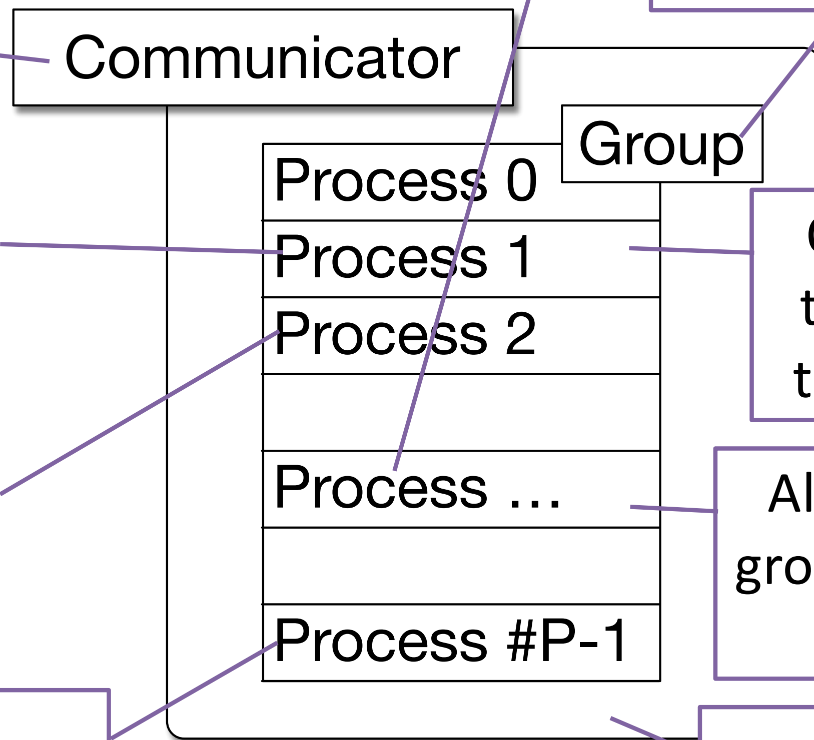
All MPI communication takes place in the context of an ***MPI Communicator***

An MPI Group translates from ***rank*** in the group to actual process

We us the index (***rank***) of a process in the group to identify other processes

The ***size*** of a communicator is the size of the group

Processes can query for size and for their own rank in group



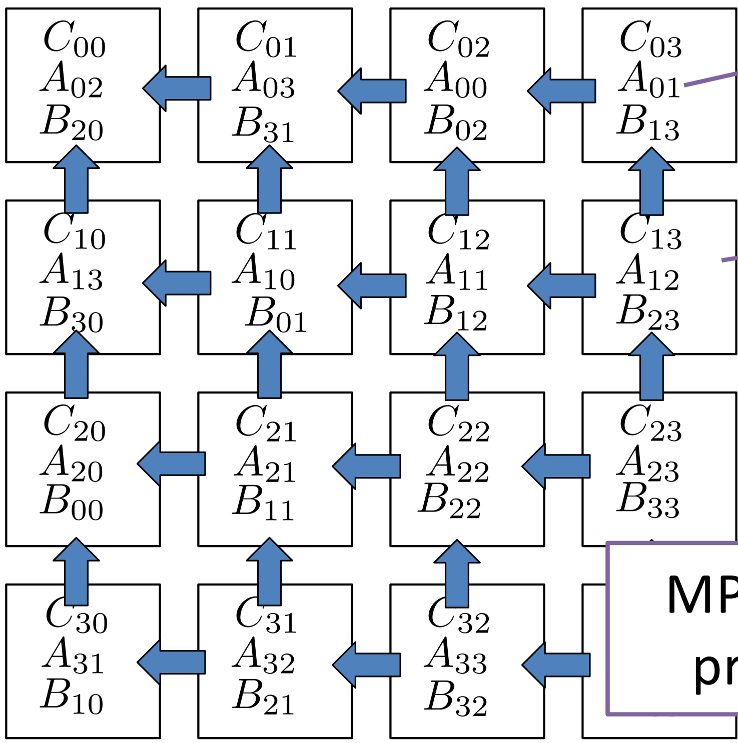
An MPI Communicator contains an ***MPI Group***

Only processes in the group can use the communicator

All processes in the group see an identical communicator

Behavior is ***as if*** it were global and shared

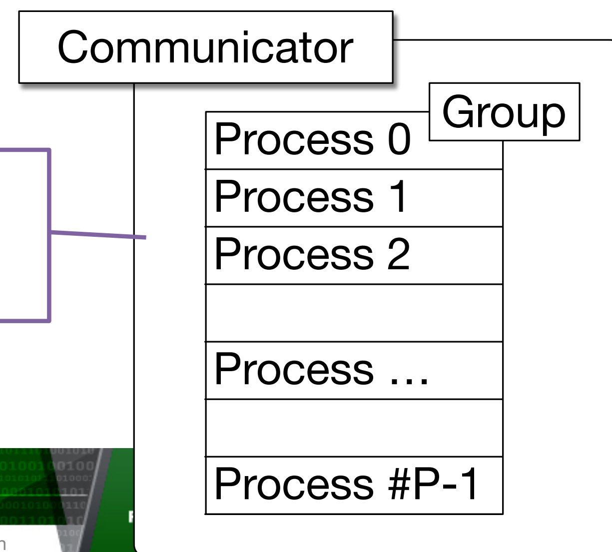
Shifting North, East, West, South



This is a useful way to reason about the algorithm

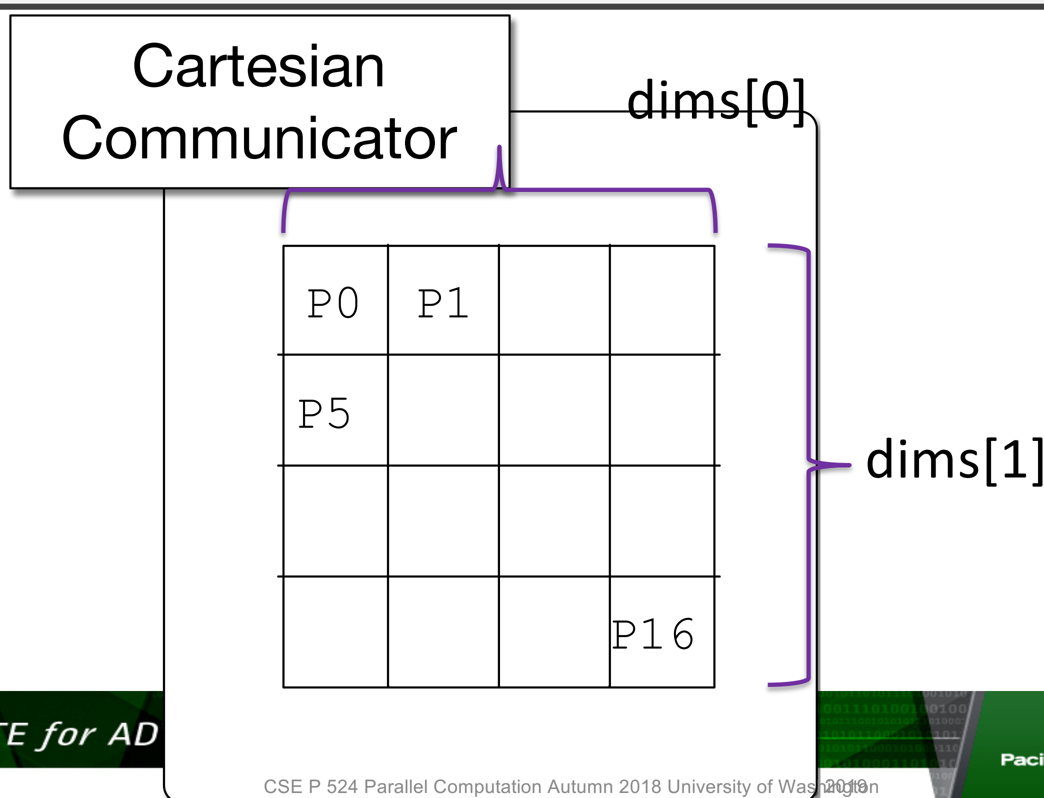
Also turns out to be efficient

MPI communicator has processes in an array



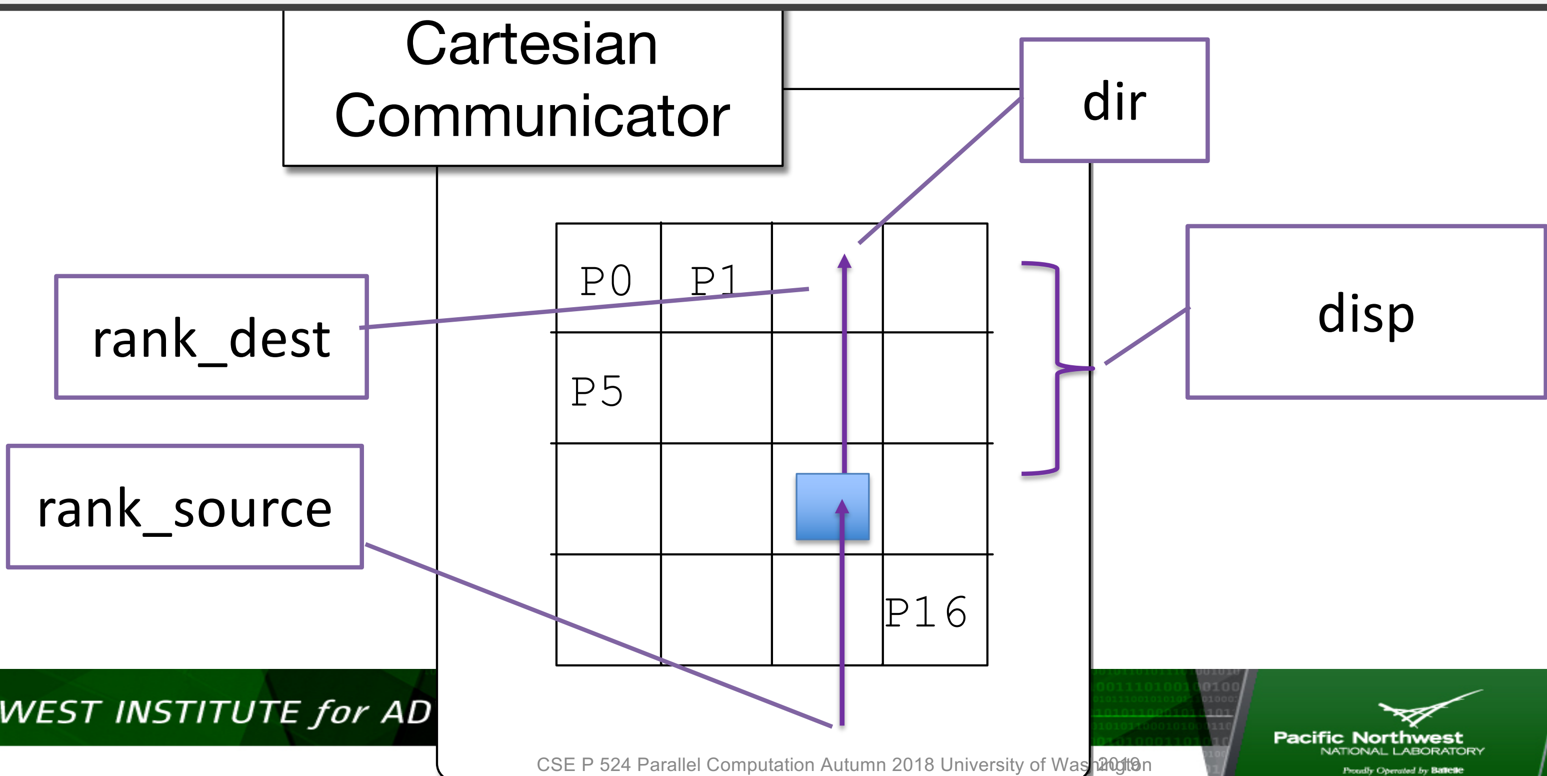
Cartesian Communicator

```
Cartcomm Intracomm.Create_cart(int ndims, int dims[], const bool periods[],
↪ bool reorder) const
```



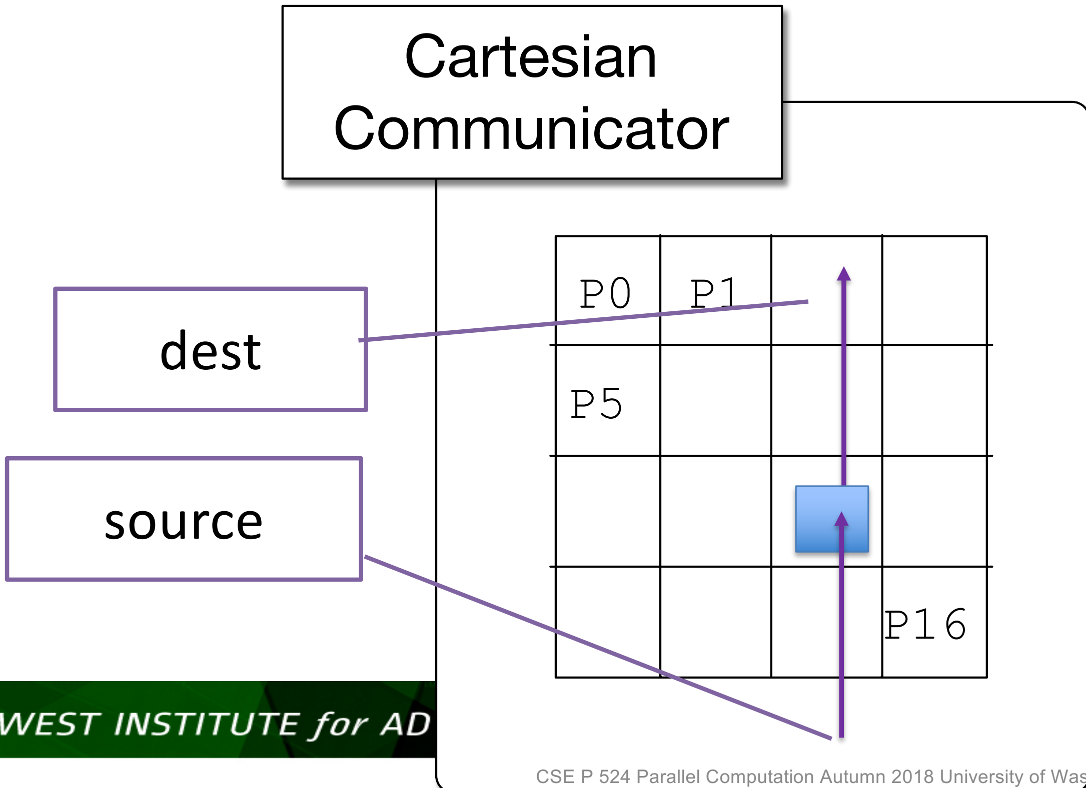
Cartesian Communicator

```
void Cartcomm::Shift(int direction, int disp, int& rank_source,
↪  int& rank_dest) const
```



Cartesian Communicator

```
void Comm::Sendrecv_replace(void* buf, int count, const Datatype& datatype,
↪  int dest, int sendtag, int source, int recvtag) const
```



Implementation

```
void cannonMultiplyMV(const Matrix& A, const Matrix& B, Matrix& C) {
  size_t mysize = MPI::COMM_WORLD.Get_size();

  // Set up grid topology and a grid (Cartesian) communicator
  int dims[2] = { (int) std::sqrt(mysize), (int) std::sqrt(mysize) };
  bool periods[2] = { true, true };

  MPI::Cartcomm gridComm = MPI::COMM_WORLD.Create_cart(2, dims, periods, true);
  size_t myrank = gridComm.Get_rank();

  int mycoords[2];
  gridComm.Get_coords(myrank, 2, mycoords);

  int northRank, eastRank, westRank, southRank;
  gridComm.Shift(0, -1, westRank,  eastRank);
  gridComm.Shift(1, -1, southRank, northRank);

  // Move A and B where they need to be to start
  int shiftSource, shiftDest;
  gridComm.Shift(0, -mycoords[0], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                            MPI::DOUBLE, shiftDest, 314, shiftSource, 314);

  gridComm.Shift(1, -mycoords[1], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*B.numCols(),
                            MPI::DOUBLE, shiftDest, 314, shiftSource, 315);


  // Main loop
  for (int k = 0; k < dims[0]; ++k) {
    hoistedCopyBlockedTiledMultiply2x2(A, B, C); // Local block matmat

    gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                              MPI::DOUBLE, westRank,  316, eastRank,  316);
    gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*A.numCols(),
                              MPI::DOUBLE, northRank, 317, southRank, 317);
  }

  // Restore A and B to initial distribution
  gridComm.Shift(0, +mycoords[0], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                            MPI::DOUBLE, shiftDest, 318, shiftSource, 318);

  gridComm.Shift(1, +mycoords[1], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*B.numCols(),
                            MPI::DOUBLE, shiftDest, 319, shiftSource, 319);

  gridComm.Free();
}
```

Implementation

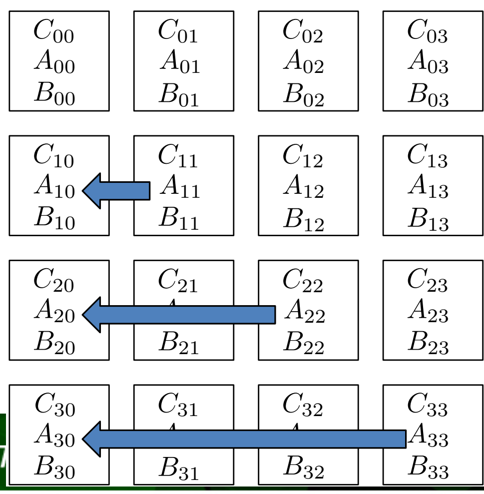
```
void cannonMultiplyMV(const Matrix& A, const Matrix& B, Matrix& C) {
  size_t mysize = MPI::COMM_WORLD.Get_size();

  // Set up grid topology and a grid (Cartesian) communicator
  int dims[2] = { (int) std::sqrt(mysize), (int) std::sqrt(mysize) };
  bool periods[2] = { true, true };

  MPI::Cartcomm gridComm = MPI::COMM_WORLD.Create_cart(2, dims, periods, true);
  size_t myrank = gridComm.Get_rank();

  int mycoords[2];
  gridComm.Get_coords(myrank, 2, mycoords);

  int northRank, eastRank, westRank, southRank;
  gridComm.Shift(0, -1, westRank,  eastRank);
  gridComm.Shift(1, -1, southRank, northRank);

  // Move A and B where they need to be to start
  int shiftSource, shiftDest;
  gridComm.Shift(0, -mycoords[0], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                            MPI::DOUBLE, shiftDest, 314, shiftSource, 314);

  gridComm.Shift(1, -mycoords[1], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*B.numCols(),
                            MPI::DOUBLE, shiftDest, 314, shiftSource, 315);


  // Main loop
  for (int k = 0; k < dims[0]; ++k) {
    hoistedCopyBlockedTiledMultiply2x2(A, B, C); // Local block matmat

    gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                              MPI::DOUBLE, westRank,  316, eastRank,  316);
    gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*A.numCols(),
                              MPI::DOUBLE, northRank, 317, southRank, 317);
  }

  // Restore A and B to initial distribution
  gridComm.Shift(0, +mycoords[0], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                            MPI::DOUBLE, shiftDest, 318, shiftSource, 318);

  gridComm.Shift(1, +mycoords[1], shiftSource, shiftDest);
  gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*B.numCols(),
                            MPI::DOUBLE, shiftDest, 319, shiftSource, 319);

  gridComm.Free();
}
```

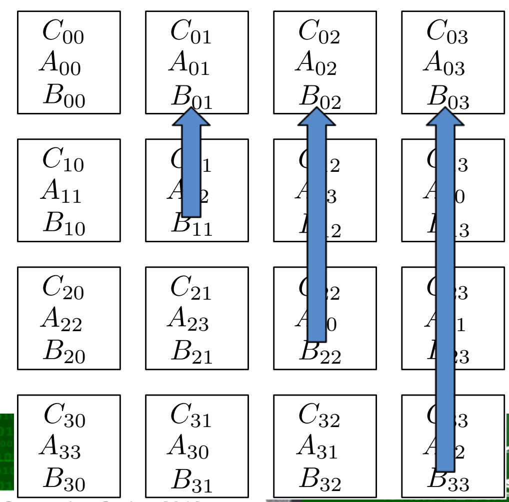
Implementation

```cpp
void cannonMultiplyMV(const Matrix& A, const Matrix& B, Matrix& C) {
  size_t mysize = MPI::COMM_WORLD.Get_size();

  // Set up grid topology and a grid (Cartesian) communicator
  int dims[2] = { (int) std::sqrt(mysize), (int) std::sqrt(mysize) };
  bool periods[2] = { true, true };

  MPI::Cartcomm gridComm = MPI::COMM_WORLD.Create_cart(2, dims, periods, true);
  size_t myrank = gridComm.Get_rank();

  int mycoords[2];
  gridComm.Get_coords(myrank, 2, mycoords);

  int northRank, eastRank, westRank, southRank;
  gridComm.Shift(0, -1, westRank,  eastRank);
  gridComm.Shift(1, -1, southRank, northRank);
```

Implementation

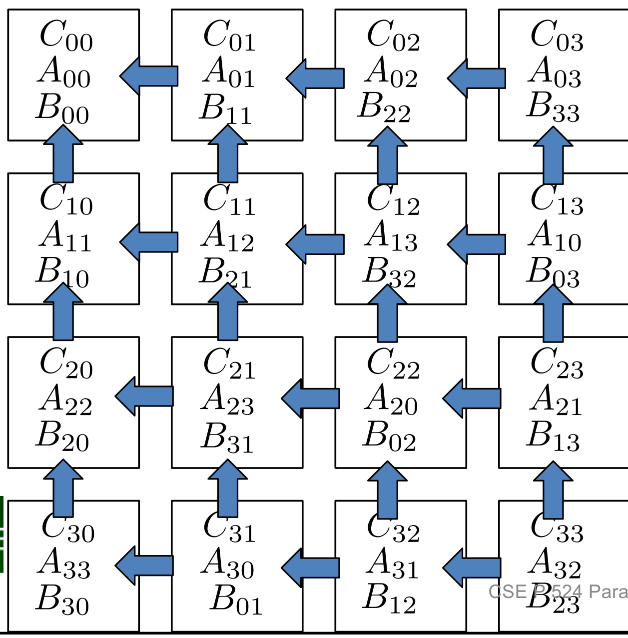
```
// Move A and B where they need to be to start
int shiftSource, shiftDest;
gridComm.Shift(0, -mycoords[0], shiftSource, shiftDest);
gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                          MPI::DOUBLE, shiftDest, 314, shiftSource, 314);

gridComm.Shift(1, -mycoords[1], shiftSource, shiftDest);
gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*B.numCols(),
                          MPI::DOUBLE, shiftDest, 314, shiftSource, 315);
```





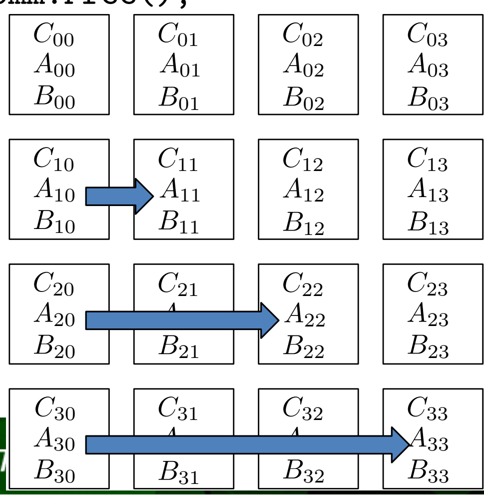
Implementation

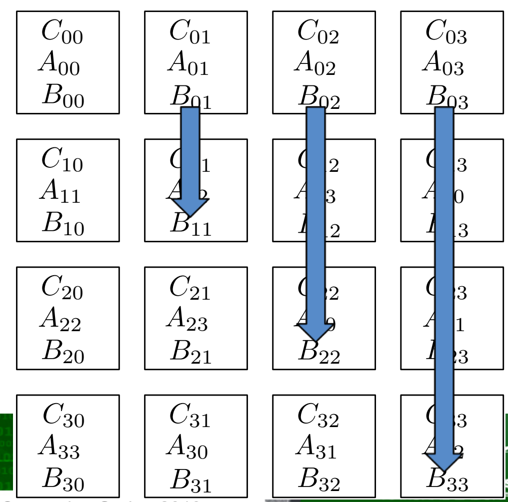
```
// Main loop
for (int k = 0; k < dims[0]; ++k) {
  hoistedCopyBlockedTiledMultiply2x2(A, B, C); // Local block matmat

  gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                            MPI::DOUBLE, westRank,  316, eastRank,  316);
  gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*A.numCols(),
                            MPI::DOUBLE, northRank, 317, southRank, 317);
}
```



Implementation

```
    // Restore A and B to initial distribution
    gridComm.Shift(0, +mycoords[0], shiftSource, shiftDest);
    gridComm.Sendrecv_replace(const_cast<double*>(&A(0,0)), A.numRows()*A.numCols(),
                              MPI::DOUBLE, shiftDest, 318, shiftSource, 318);

    gridComm.Shift(1, +mycoords[1], shiftSource, shiftDest);
    gridComm.Sendrecv_replace(const_cast<double*>(&B(0,0)), B.numRows()*B.numCols(),
                              MPI::DOUBLE, shiftDest, 319, shiftSource, 319);

    gridComm.Free();
}
```





Where do we go from here?

Clouds = Services

Amazon Web Services

Compute
- **EC2** – Virtual Servers in the Cloud
- **EC2 Container Service** – Run and Manage Docker Containers
- **Elastic Beanstalk** – Run and Manage Web Apps
- **Lambda** – Run Code in Response to Events

Storage & Content Delivery
- **S3** – Scalable Storage in the Cloud
- **CloudFront** – Global Content Delivery Network
- **Elastic File System** PREVIEW – Fully Managed File System for EC2
- **Glacier** – Archive Storage in the Cloud
- **Import/Export Snowball** – Large Scale Data Transport
- **Storage Gateway** – Integrates On-Premises IT Environments with Cloud Storage

Database
- **RDS** – Managed Relational Database Service
- **DynamoDB** – Predictable and Scalable NoSQL Data Store
- **ElastiCache** – In-Memory Cache
- **Redshift** – Managed Petabyte-Scale Data Warehouse Service

Networking
- **VPC** – Isolated Cloud Resources
- **Direct Connect** – Dedicated Network Connection to AWS
- **Route 53** – Scalable DNS and Domain Name Registration

Developer Tools
- **CodeCommit** – Store Code in Private Git Repositories
- **CodeDeploy** – Automate Code Deployments
- **CodePipeline** – Release Software using Continuous Delivery

Management Tools
- **CloudWatch** – Monitor Resources and Applications
- **CloudFormation** – Create and Manage Resources with Templates
- **CloudTrail** – Track User Activity and API Usage
- **Config** – Track Resource Inventory and Changes
- **OpsWorks** – Automate Operations with Chef
- **Service Catalog** – Create and Use Standardized Products
- **Trusted Advisor** – Optimize Performance and Security

Security & Identity
- **Identity & Access Management** – Manage User Access and Encryption Keys
- **Directory Service** – Host and Manage Active Directory
- **Inspector** PREVIEW – Analyze Application Security
- **WAF** – Filter Malicious Web Traffic

Analytics
- **EMR** – Managed Hadoop Framework
- **Data Pipeline** – Orchestration for Data-Driven Workflows
- **Elasticsearch Service** – Run and Scale Elasticsearch Clusters
- **Kinesis** – Work with Real-time Streaming data

Internet of Things
- **AWS IoT** BETA – Connect Devices to the cloud

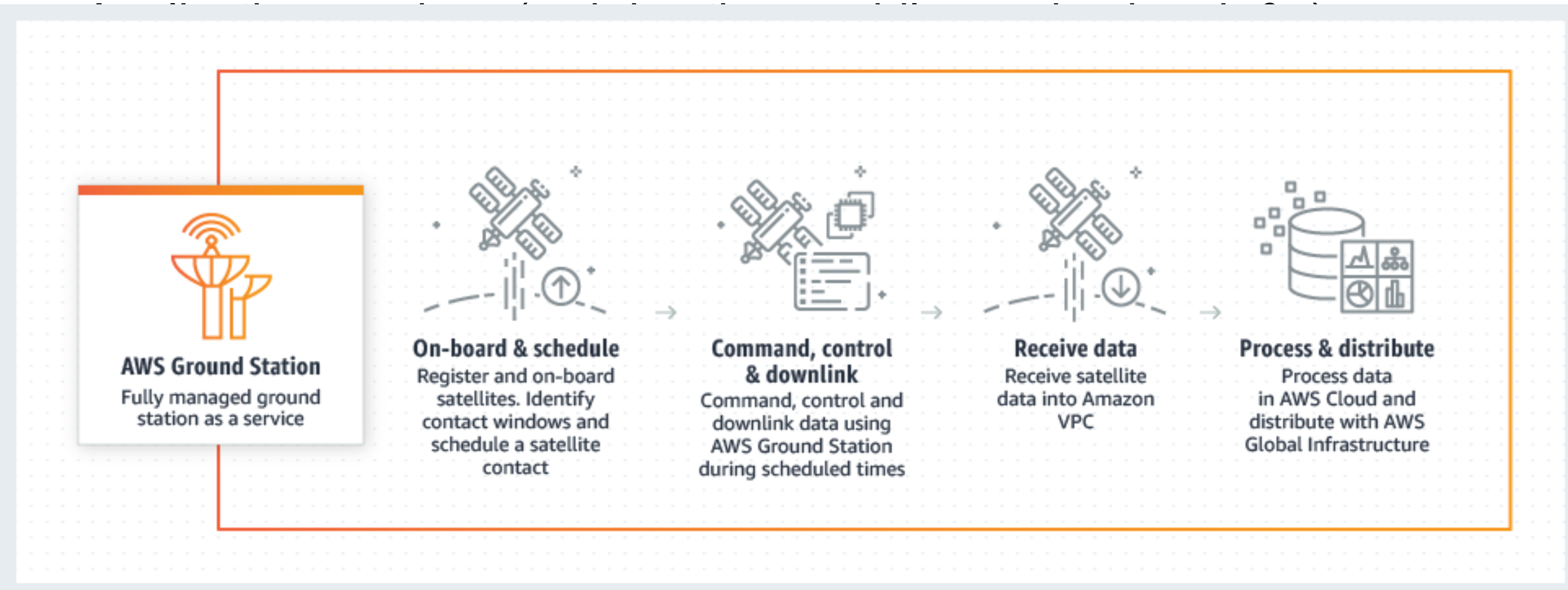
Mobile Services
- **Mobile Hub** BETA – Build, Test, and Monitor Mobile apps
- **Cognito** – User Identity and App Data Synchronization
- **Device Farm** – Test Android, Fire OS, and iOS apps on real devices in the Cloud
- **Mobile Analytics** – Collect, View and Export App Analytics
- **SNS** – Push Notification Service

Application Services
- **API Gateway** – Build, Deploy and Manage APIs
- **AppStream** – Low Latency Application Streaming
- **CloudSearch** – Managed Search Service
- **Elastic Transcoder** – Easy-to-use Scalable Media Transcoding
- **SES** – Email Sending Service
- **SQS** – Message Queue Service
- **SWF** – Workflow Service for Coordinating Application Components

Enterprise Applications
- **WorkSpaces** – Desktops in the Cloud
- **WorkDocs** – Secure Enterprise Storage and Sharing Service
- **WorkMail** PREVIEW – Secure Email and Calendaring Service

Services: On Demand Access

- Data Storage (blob, file, unstructured, SQL, &c)
- Computing (VM, cluster, GPU)



What's Next

- Machine learning
- Quantum computing
- 5G
- IoT / edge computing

Thank You!

- Be well
- Do good work
- Stay in touch

Creative Commons BY-NC-SA 4.0 License