Revision

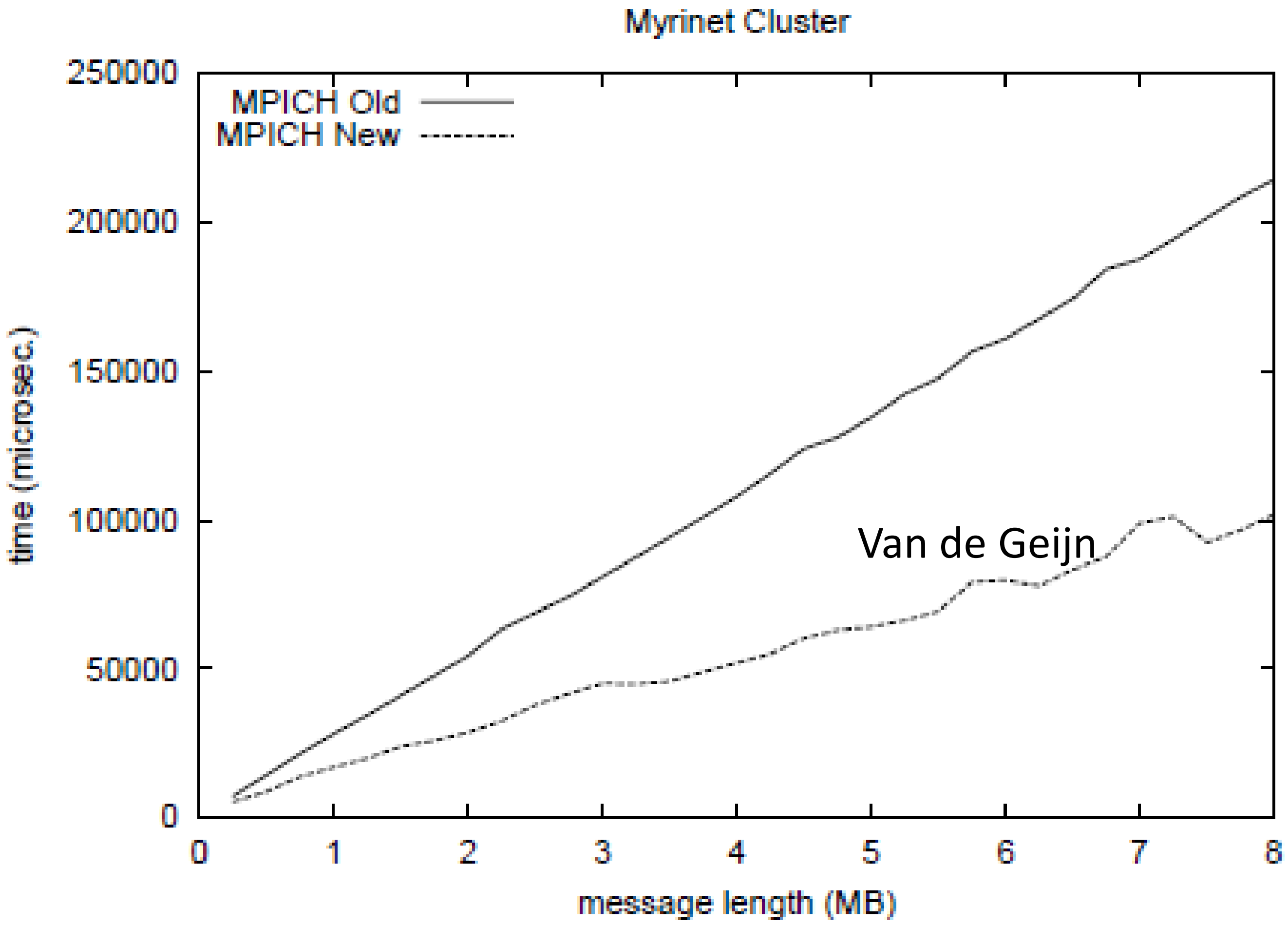
Lecture 12

February 14, 2024

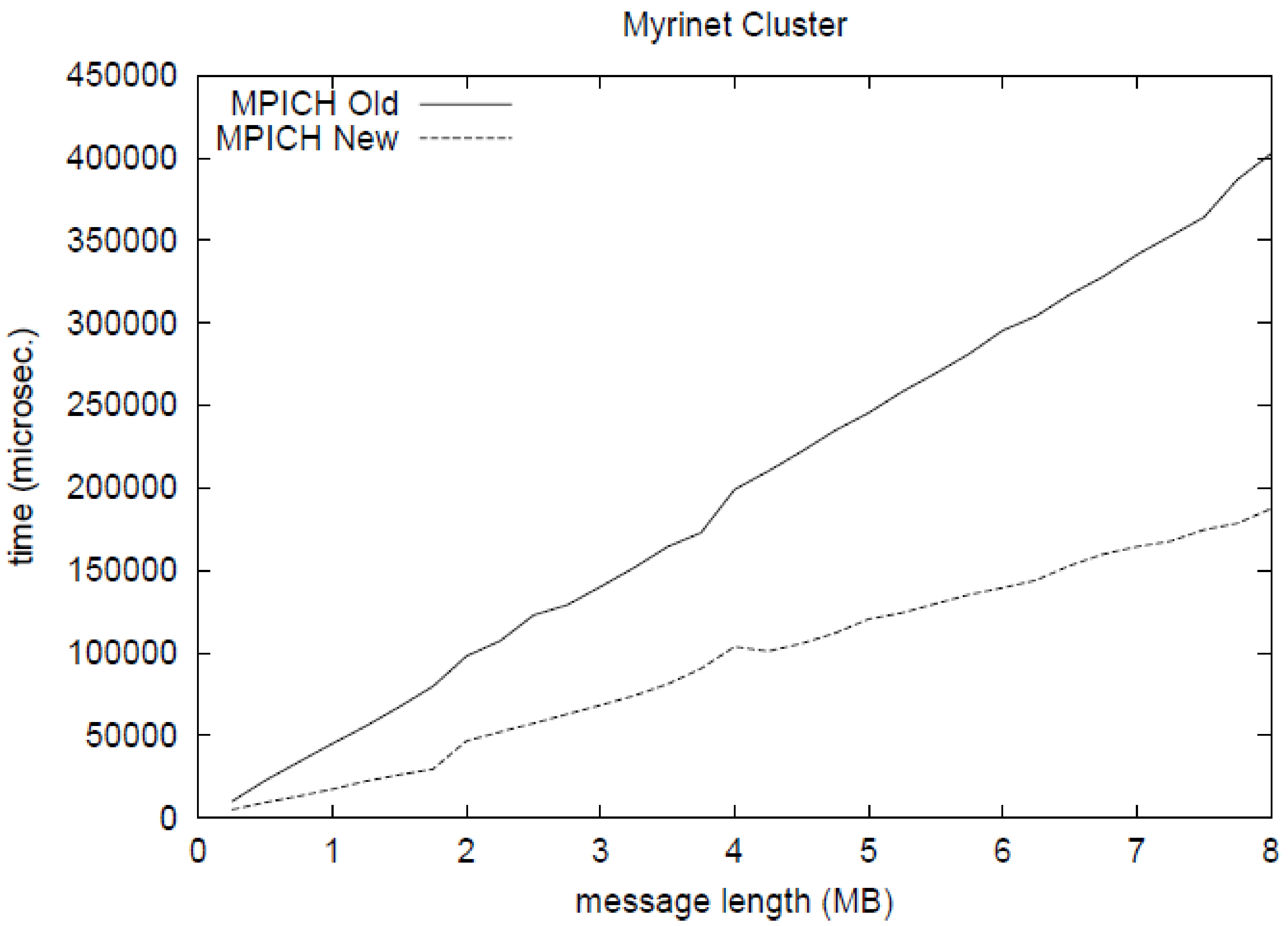
Broadcast Algorithms in MPICH

- Short messages
  - < MPIR_CVAR_BCAST_SHORT_MSG_SIZE
  - Binomial
- Medium messages
  - Scatter + Allgather (Recursive doubling)
- Large messages
  - > MPIR_CVAR_BCAST_LONG_MSG_SIZE
  - Scatter + Allgather (Ring)

Old vs. New MPI_Bcast

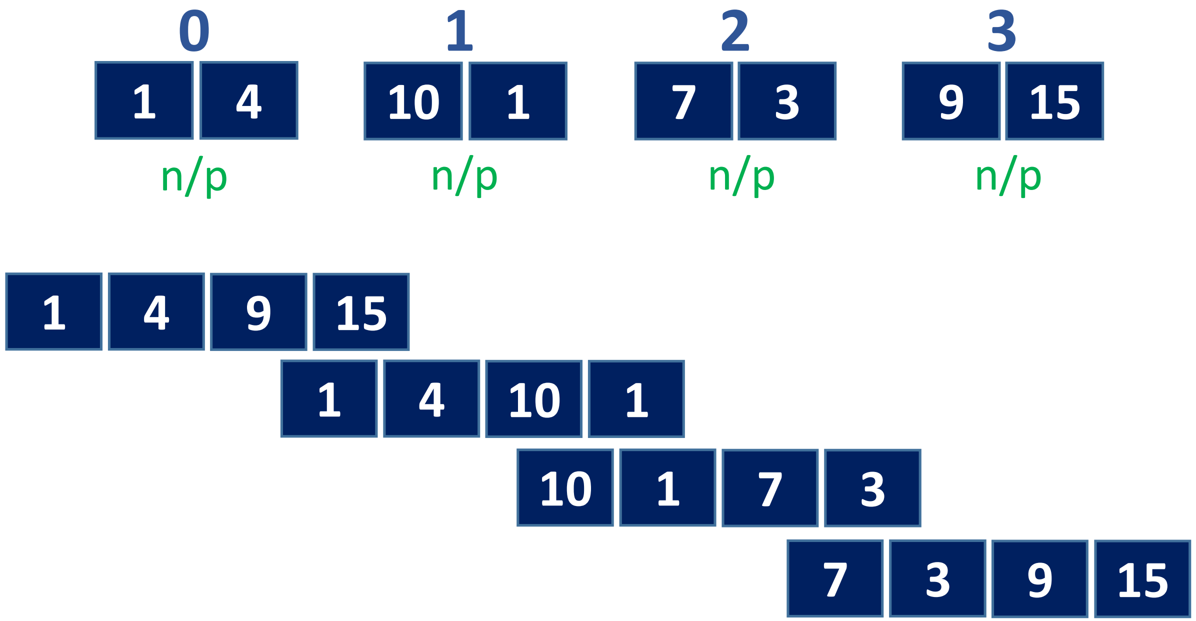


Reduce on 64 nodes



Allgather – Ring Algorithm

- Every process sends to and receives from everyone else
- Assume p processes and total n bytes
- Every process sends and receives n/p bytes
- Time
  - (p – 1) * (L + n/p*(1/B))
- How can we improve?



Non-blocking Point-to-Point

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)

- MPI_Wait (request, status)
- MPI_Waitall (count, request, status)

Many-to-one Non-blocking P2P

```
// send from all ranks to the last rank
start_time = MPI_Wtime ();
if (myrank < size-1)
{
  MPI_Send(arr, BUFSIZE, MPI_INT, size-1, 99, MPI_COMM_WORLD);
}
else
{
  int count, recvarr[size][BUFSIZE];
  for (int i=0; i<size-1; i++)
    MPI_Irecv(recvarr[i], BUFSIZE, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request[i]);
  MPI_Waitall (size-1, request, status);
}
time = MPI_Wtime () - start_time;

MPI_Reduce (&time, &max_time, 1, MPI_DOUBLE, MPI_MAX, size-1, MPI_COMM_WORLD);
if (myrank == size-1) printf ("Max time = %lf\n", max_time);
```

Non-blocking Performance

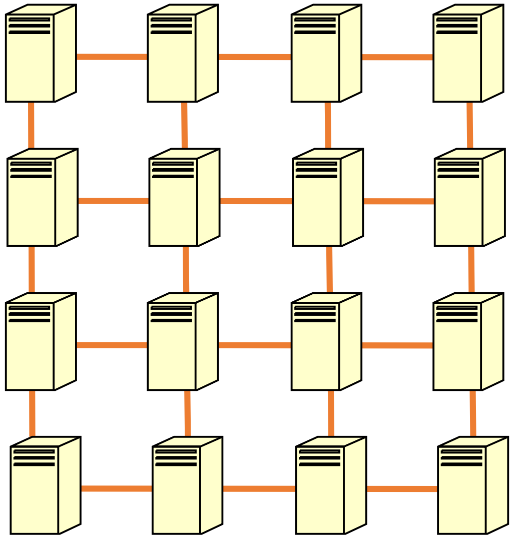
- Standard does not require overlapping communication and computation
- Implementation may use a thread to move data in parallel
- Implementation can delay the initiation of data transfer until “Wait”
- MPI_Test – non-blocking, tests completion, starts progress
- MPIR_CVAR_ASYNC_PROGRESS (MPICH)

Non-blocking Point-to-Point Safety

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)
- MPI_Wait (request, status)

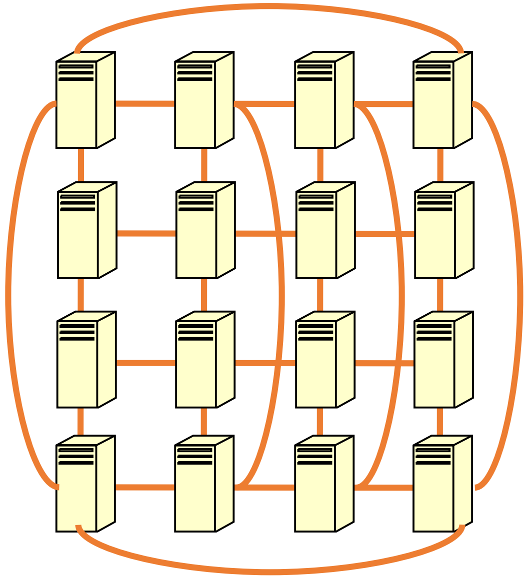
0	1	
MPI_Isend	MPI_Isend	Safe
MPI_Recv	MPI_Recv	

Mesh Interconnect



- Diameter $2(\sqrt{p} - 1)$
- Bisection width $\sqrt{p}$
- Cost $2(p - \sqrt{p})$

Torus Interconnect



- Diameter $2(\sqrt{p}/2)$
- Bisection width $2\sqrt{p}$
- Cost $2p$

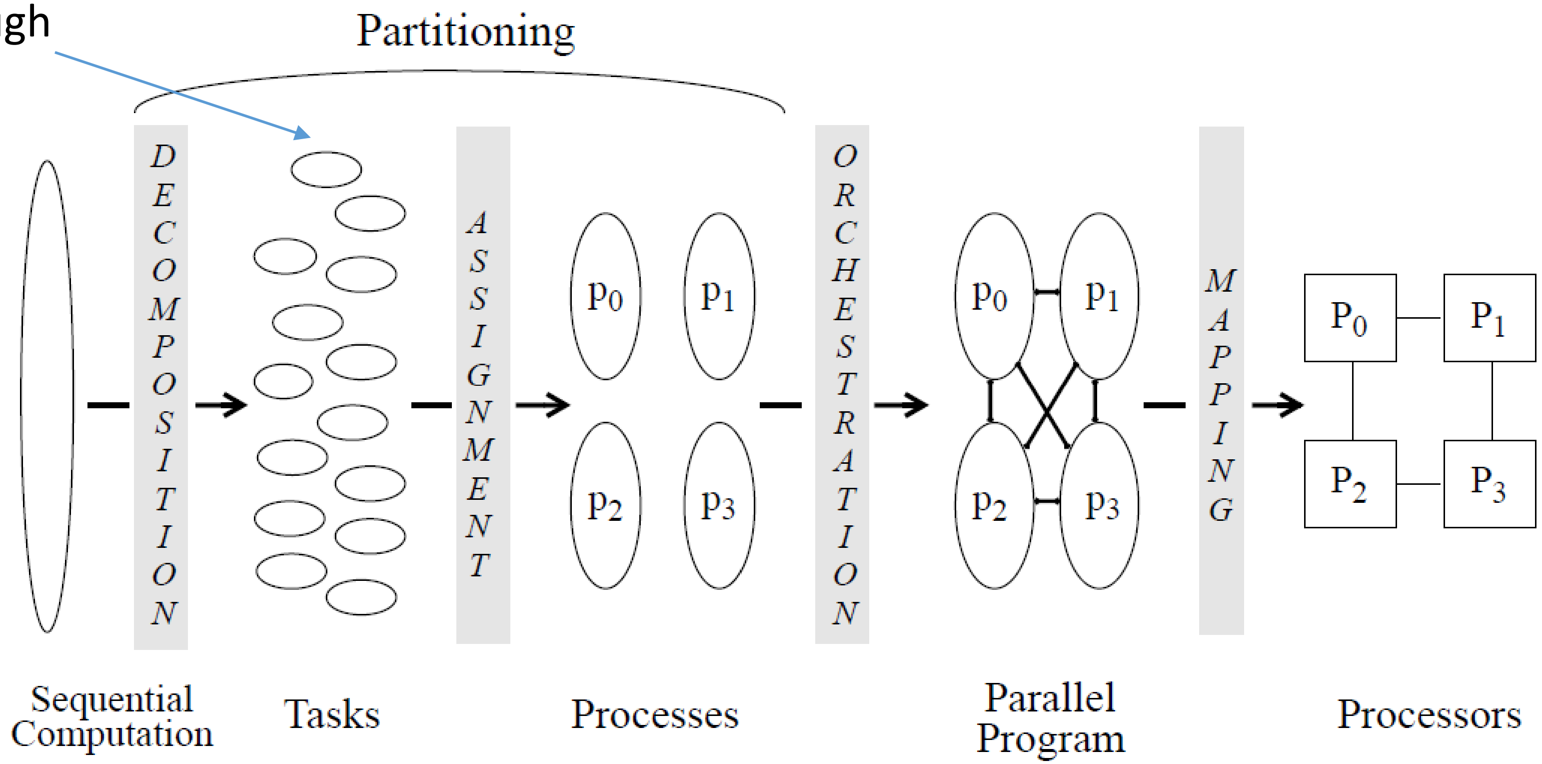
Parallelization

Parallelization Steps

1. *Decomposition* of computation into tasks
    - Identifying portions of the work that can be performed concurrently
2. *Assignment* of tasks to processes
    - Assigning concurrent pieces of work onto multiple processes running in parallel
3. *Orchestration* of data access, communication and synchronization among processes
    - Distributing the data associated with the program
    - Managing access to data shared by multiple processes
    - Synchronizing at various stages of the parallel program execution
4. *Mapping* of processes to processors
    - Placement of processes in the physical processor topology

Illustration of Parallelization Steps

Expose enough concurrency

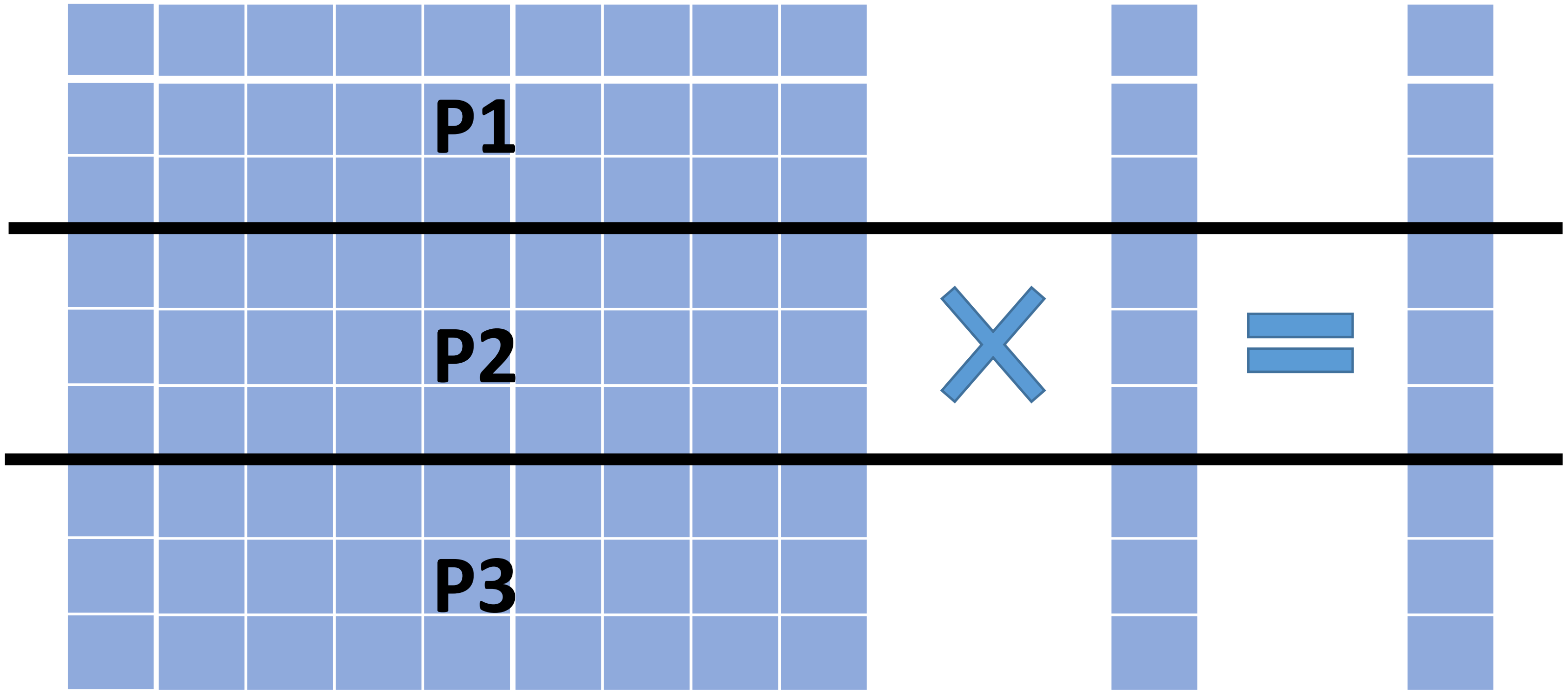


Source: Culler et al. book

Performance Goals

- Expose concurrency
- Reduce inter-process communications
- Load-balance
- Reduce synchronization
- Reduce idling
- Reduce management overhead
- Preserve data locality
- Exploit network topology

Matrix Vector Multiplication – Decomposition



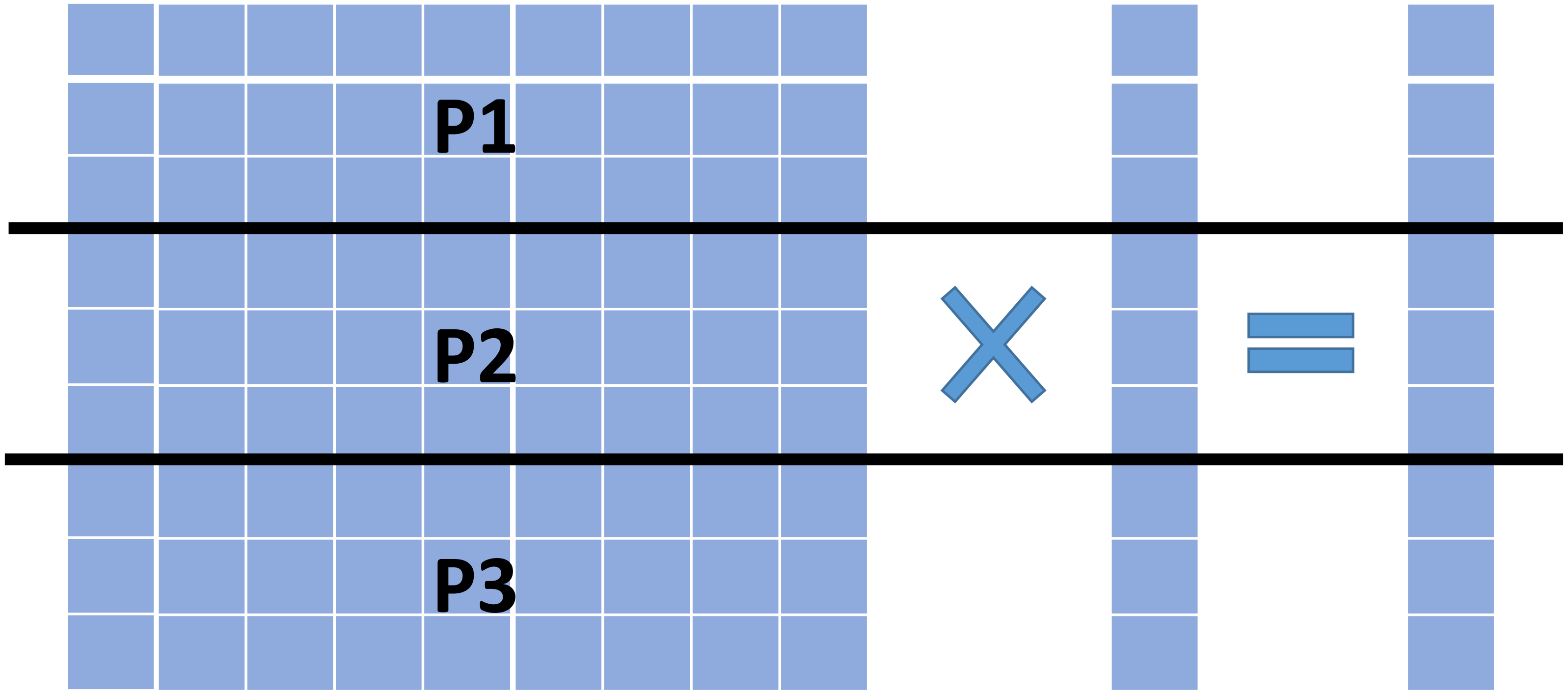
P = 3 ?

Decomposition

Identifying portions of the work that can be performed concurrently

Assignment

Matrix Vector Multiplication – Orchestration



P = 3

Decomposition

Assignment

Orchestration

- Allgather/Bcast
- Scatter
- Gather

- Initial communication
    - Distribute (read by process 0) or parallel reads
- Final communication

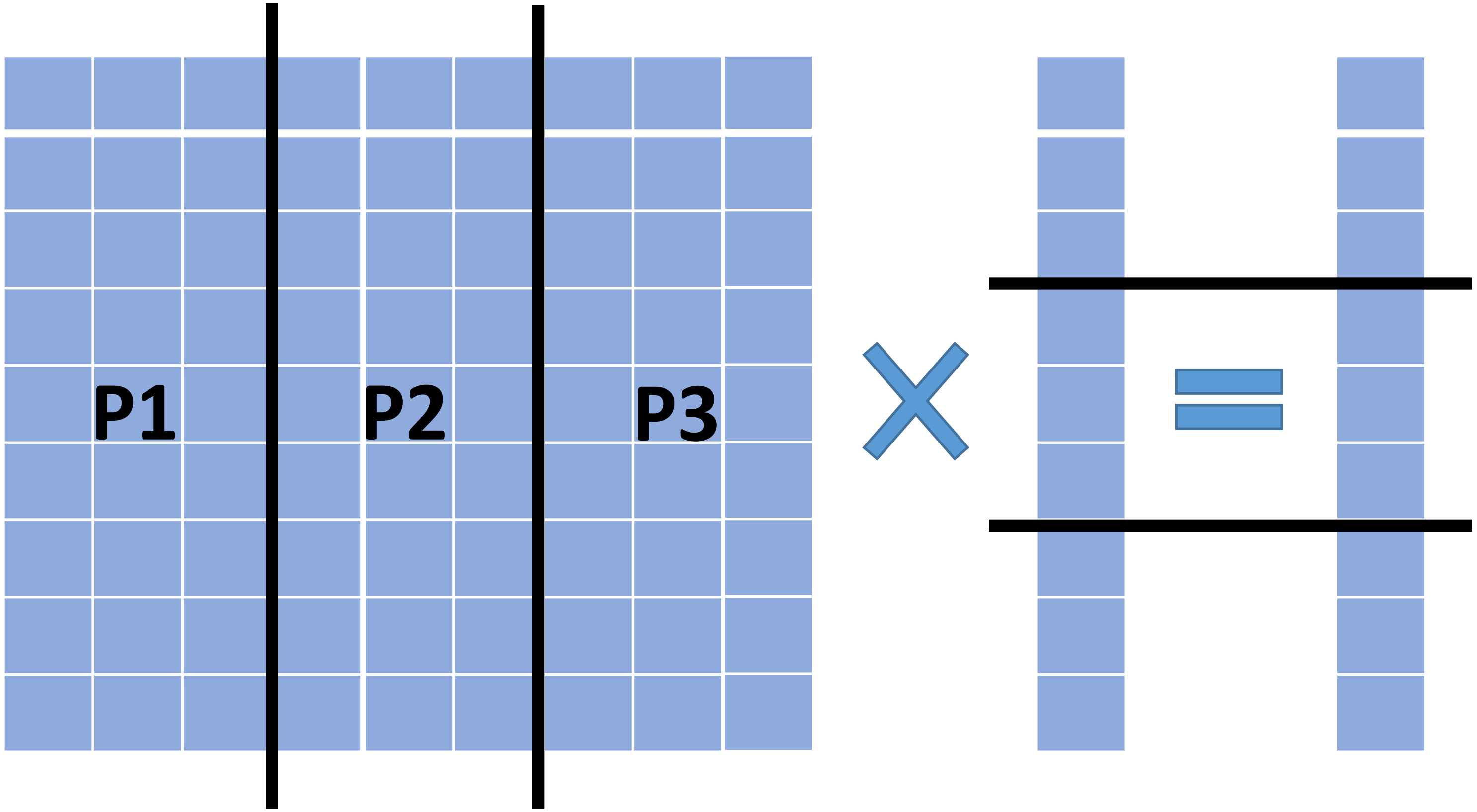
Distribute using Bcast vs. Allgather

```
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
MPI_Comm_size( MPI_COMM_WORLD, &commsize );

time = MPI_Wtime();
MPI_Bcast (buf, N, MPI_FLOAT, 0, MPI_COMM_WORLD);
etime = MPI_Wtime() - time;
MPI_Reduce (&etime, &maxtime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
if (!myrank) printf ("Time to bcast: %11.3lf\n", maxtime);

int count = N/commsize;
time = MPI_Wtime();
MPI_Allgather (buf, count, MPI_FLOAT, recvbuf, count, MPI_FLOAT, MPI_COMM_WORLD);
etime = MPI_Wtime() - time;
MPI_Reduce (&etime, &maxtime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
if (!myrank) printf ("Time to allgather: %7.3lf\n", maxtime);
```

```
class for i in `seq 1 3` ; do mpirun -np 10 -hosts csews2,csews5,csews20 ./bcast-allgather 10000; echo ; done
Time to bcast:       0.014
Time to allgather:   0.021

Time to bcast:       0.018
Time to allgather:   0.009

Time to bcast:       0.012
Time to allgather:   0.007

class for i in `seq 1 3` ; do mpirun -np 10 -hosts csews2,csews5,csews20 ./bcast-allgather 100000; echo ; done
Time to bcast:       0.034
Time to allgather:   0.011

Time to bcast:       0.027
Time to allgather:   0.023

Time to bcast:       0.026
Time to allgather:   0.011

class for i in `seq 1 3` ; do mpirun -np 10 -hosts csews2,csews5,csews20 ./bcast-allgather 1000000; echo ; done
Time to bcast:       0.187
Time to allgather:   0.347

Time to bcast:       0.176
Time to allgather:   0.111

Time to bcast:       0.155
Time to allgather:   0.112
```

```
class for i in `seq 1 3` ; do mpirun -np 10 -hosts csews2,csews5,csews20 ./bcast-allgather 1000000; echo ; done
Time to bcast:       0.187
Time to allgather:   0.347

Time to bcast:       0.176
Time to allgather:   0.111

Time to bcast:       0.155
Time to allgather:   0.112

class for i in `seq 1 3` ; do mpirun -np 10 -hosts csews2,csews5,csews20 ./bcast-allgather 10000000; echo ; done
Time to bcast:       1.421
Time to allgather:   1.121

Time to bcast:       1.618
Time to allgather:   1.282

Time to bcast:       1.674
Time to allgather:   1.583

class for i in `seq 1 3` ; do mpirun -np 10 -hosts csews2,csews5,csews20 ./bcast-allgather 100000000; echo ; done
Time to bcast:      18.061
Time to allgather:  15.616

Time to bcast:      23.447
Time to allgather:  17.005

Time to bcast:      16.875
Time to allgather:  11.085
```

Matrix Vector Multiplication – Column-wise Decomposition

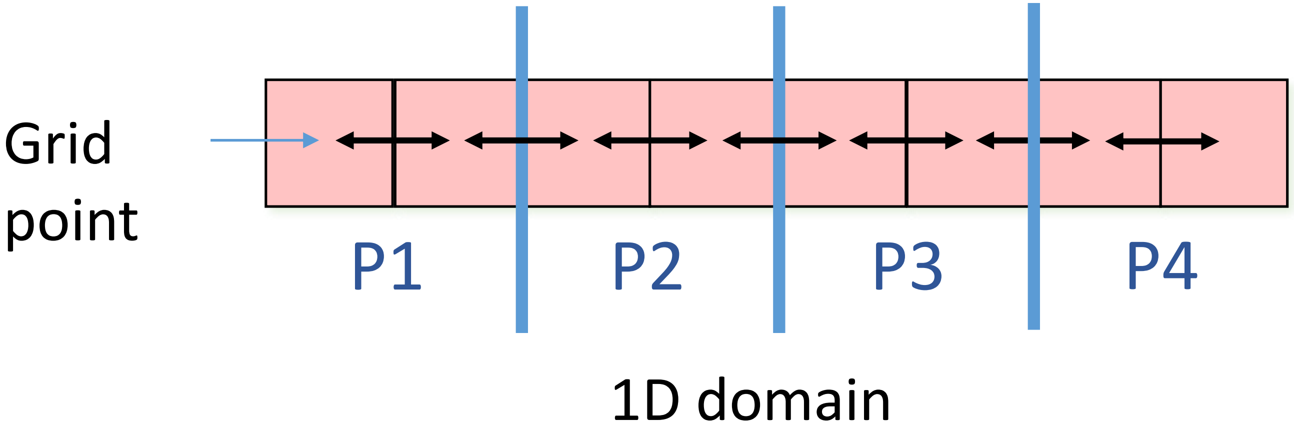


Decomposition
Assignment
Orchestration

- Reduce

Row-wise vs. column-wise partitioning?

1D Domain Decomposition



N grid points
P processes
N/P points per process

#Communications?
2
#Computations?
N/P

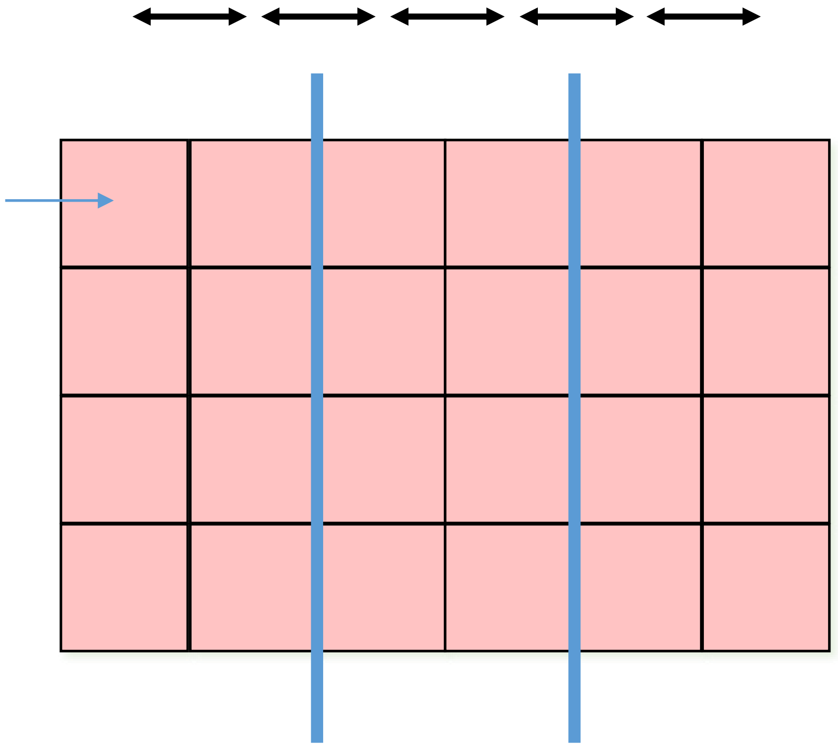
Nearest neighbor communications

2 sends()
2 recvs()

Communication to computation ratio=2P/N

1D Domain Decomposition

Grid point



2D domain

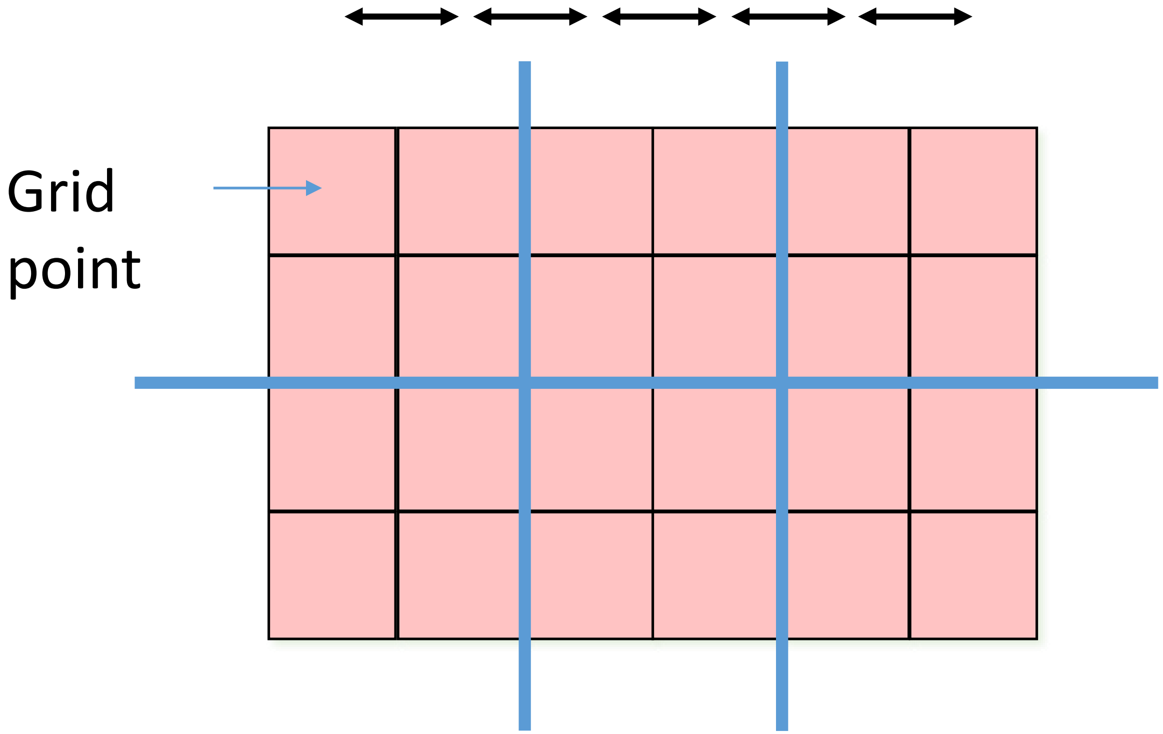
N grid points
P processes
N/P points per process

#Communications?
$2\sqrt{N}$ (assuming square grid)

#Computations?
N/P (assuming square grid)

Communication to computation ratio=?

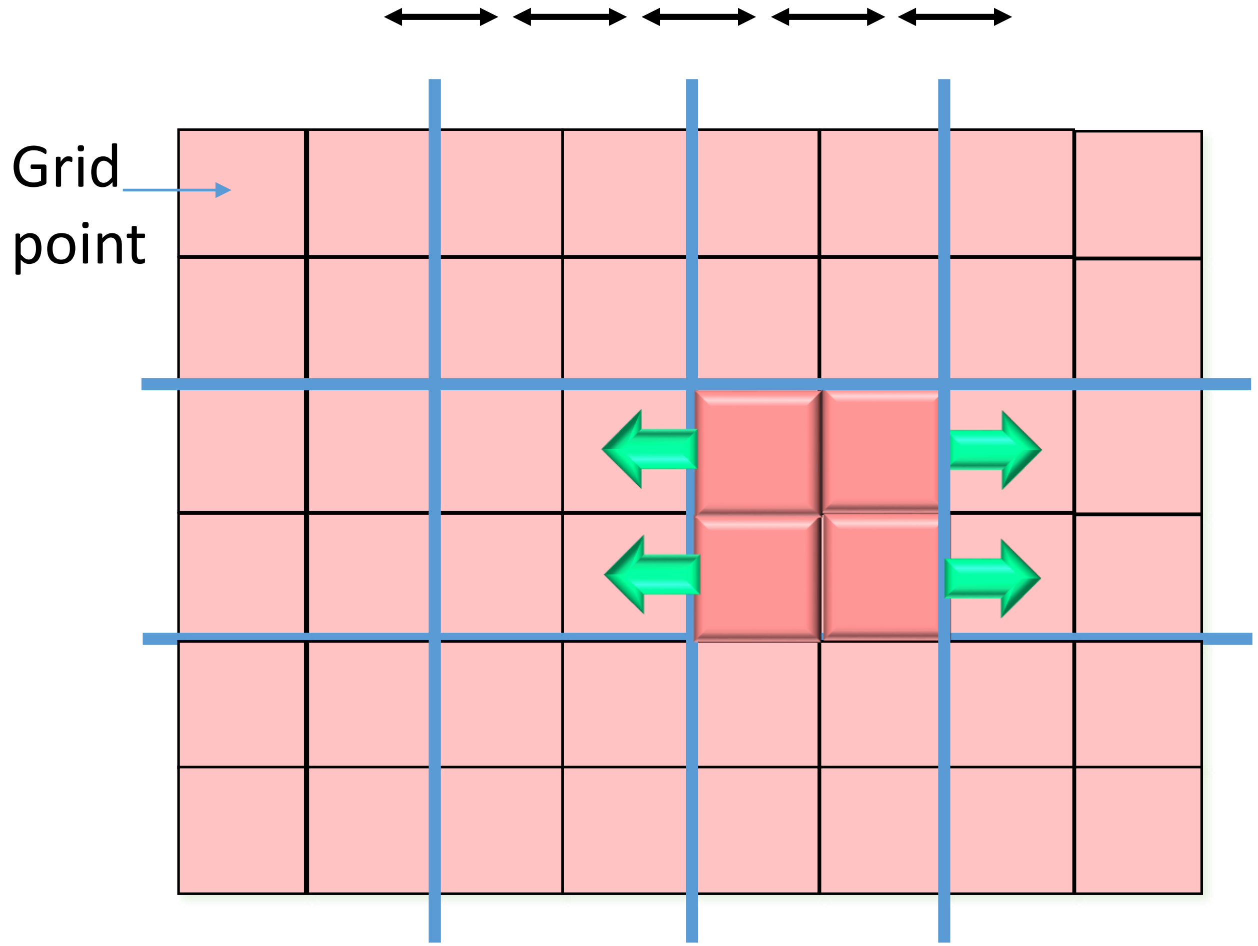
2D Domain decomposition



N grid points (√N x √N grid)
P processes (√P x √P grid)
N/P points per process

\+ Several parallel communications
\+ Lower communication volume/process

2D Domain decomposition



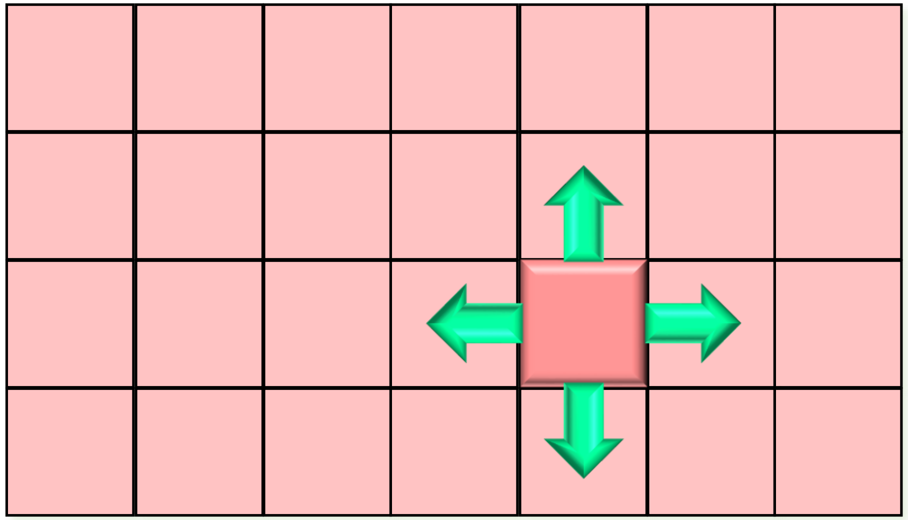
2 Sends()
2 Recvs()

#Communications?
$2\sqrt{N}/\sqrt{P}$ (assuming square grid)

#Computations?
N/P (assuming square grid)

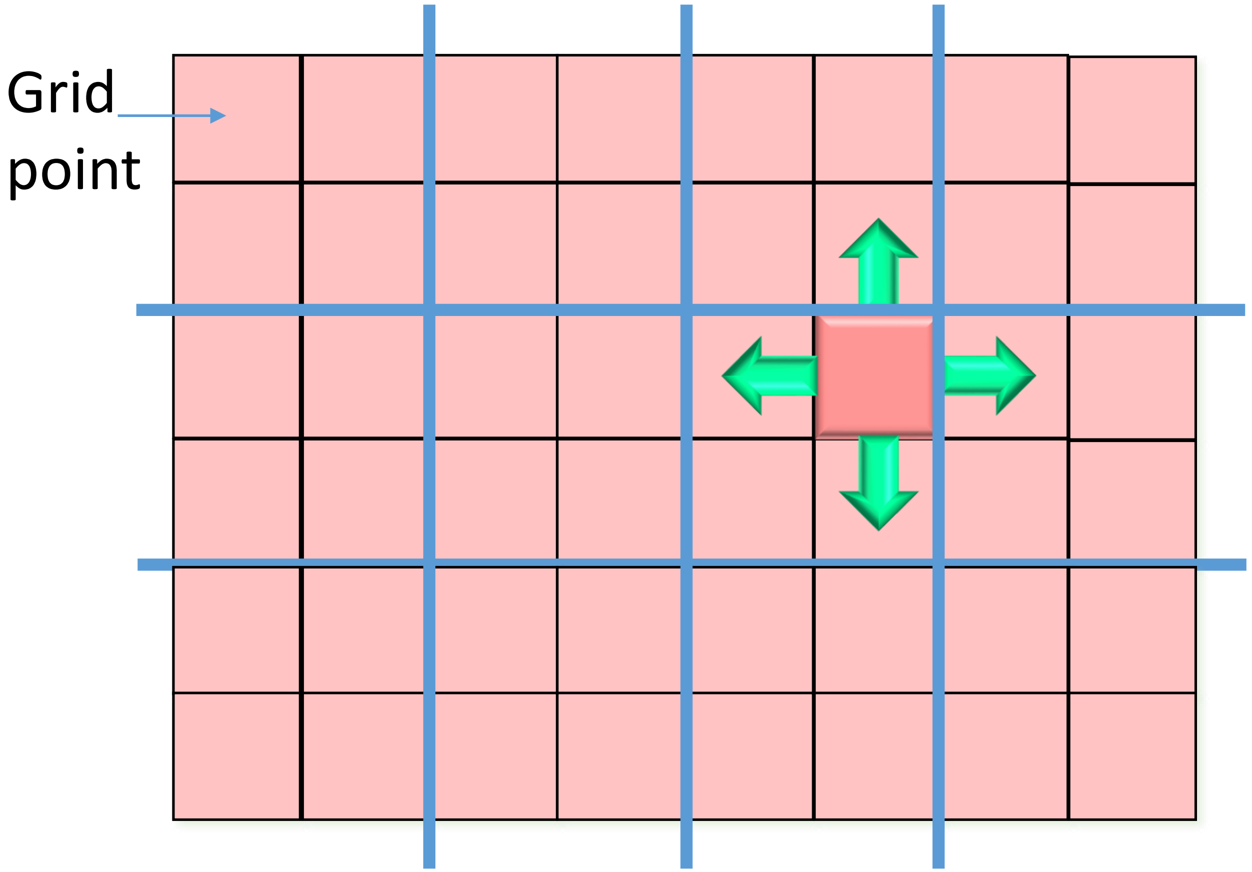
Communication to computation ratio=?

Stencils



Five-point stencil

2D Domain decomposition



4 Sends()
4 Recvs()

N grid points (√N x √N grid)
P processes (√P x √P grid)
N/P points per process

#Communications?

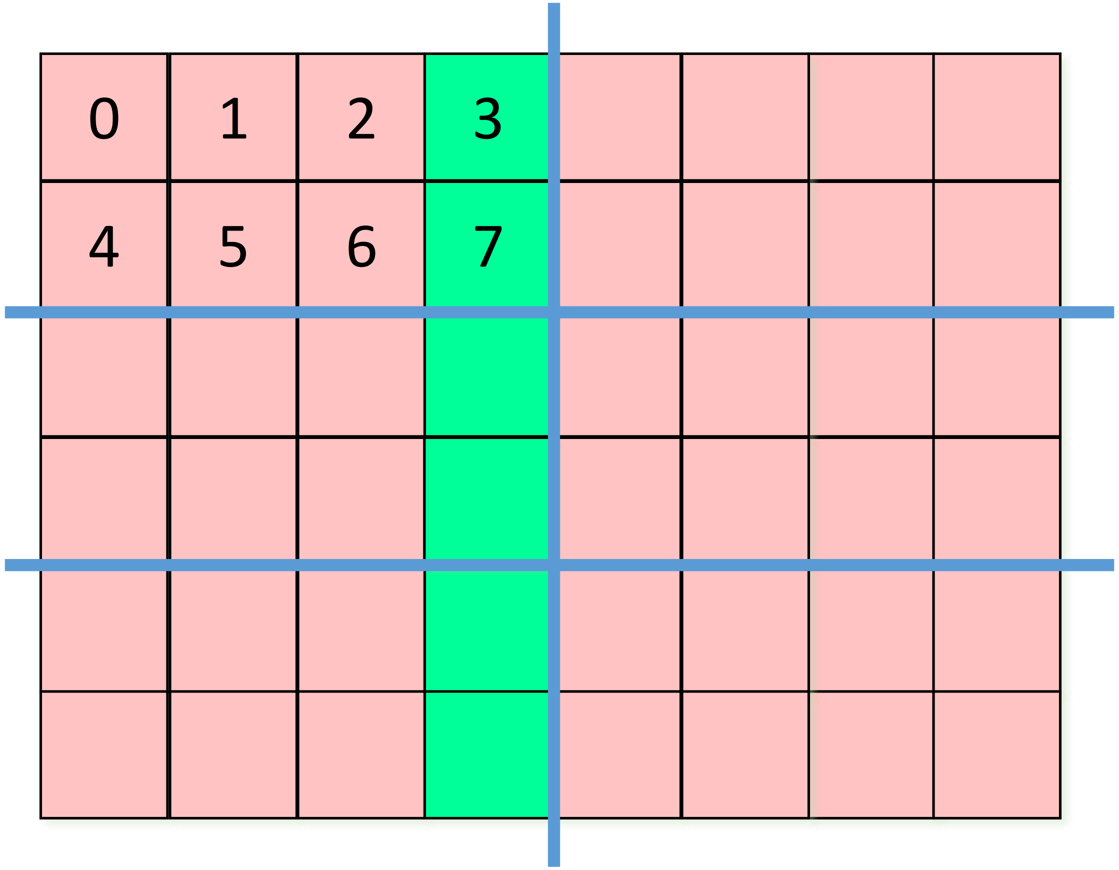
4√N/√P (assuming square grid)

#Computations?

N/P (assuming square grid)

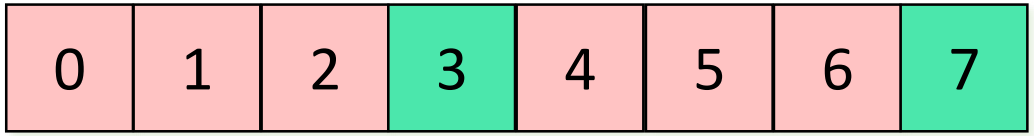
Communication to computation ratio=?

Send / Recv

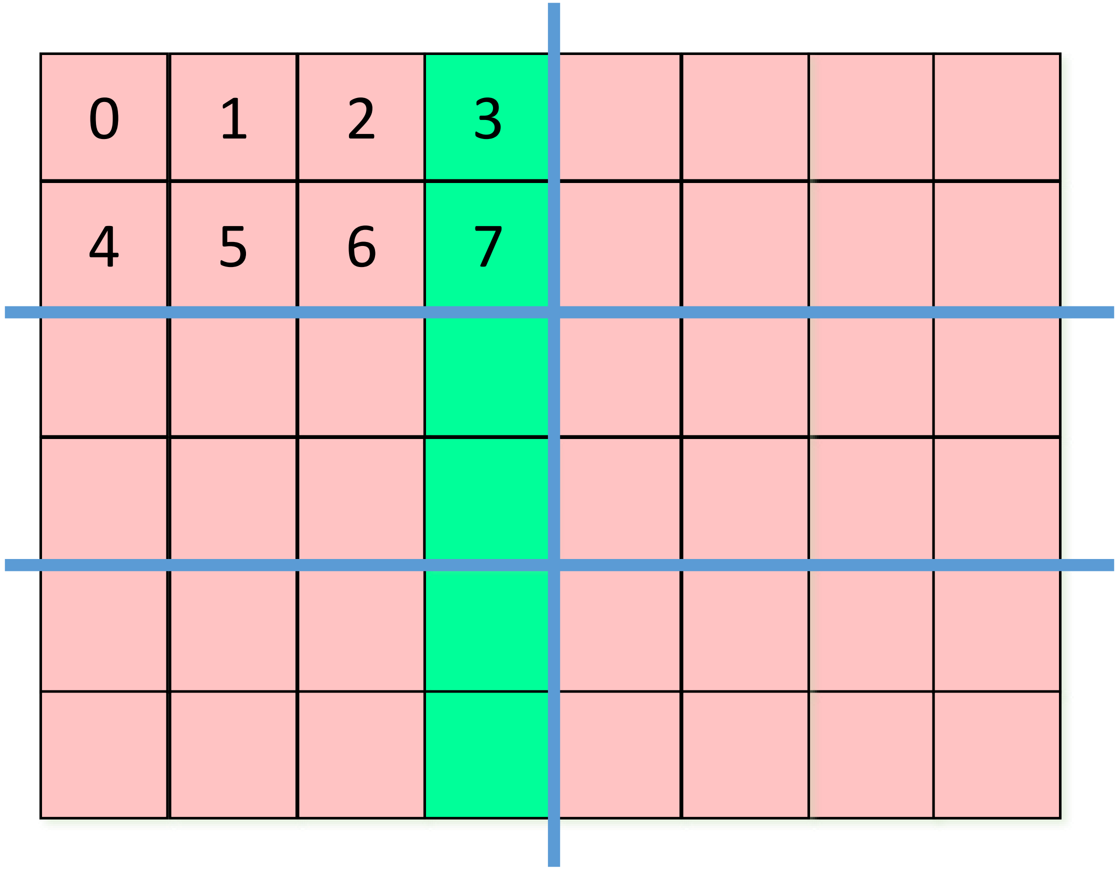


MPI_Send

MPI_Recv

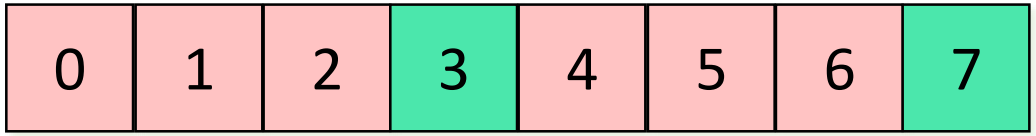


Send / Recv



MPI_Pack (buf)
MPI_Send (buf)

MPI_Recv (buf)
MPI_Unpack (buf)



MPI_Pack

```c
MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
MPI_Comm_size(MPI_COMM_WORLD, &size);

// initialize data
for (int i=0; i<M; i++)
 for (int j=0; j<N; j++)
  array2D[i][j] = myrank+i+j;

sTime = MPI_Wtime();
if (myrank == 0) {
 // pack the last element of every row (N ints)
 for (int j=0; j<N; j++) {
   MPI_Pack (&array2D[j][M-1], 1, MPI_INT, buffer, 400, &position, MPI_COMM_WORLD);
   printf ("packed %d %d\n", j, position);
 }
 MPI_Send (buffer, position, MPI_PACKED, 1, 1, MPI_COMM_WORLD);
}
else {
 // receive N ints
 if (myrank == 1)
  MPI_Recv (buffer, count, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
 // verify
 MPI_Get_count (&status, MPI_INT, &count);
}
eTime = MPI_Wtime();
time = eTime - sTime;

printf ("%lf\n", time);
```

int MPI_Pack (const void *inbuf, int incount, MPI_Datatype datatype, void **outbuf*, int outsize, int **position*, MPI_Comm comm)