KOÇ UNIVERSITY

Communication Optimizations

Didem Unat

COMP 429/529 Parallel Programming

Communication Optimizations

- Reduce the number of messages or total message size
  - Reduces communication cost
- Hide communication
  - Hide the communication overhead behind useful computation
- Avoid communication
  - Doesn’t perform communication at the expense of more computation
- Combine (aggregate messages)
  - Instead of sending many small messages, send few large messages
- These methods can be combined

Reducing Communication Cost

- Reformulate MPI applications to reduce communication cost
  - Major restructuring of a program
  - Code can become very messy
  - Split phase coding with non-blocking MPI calls

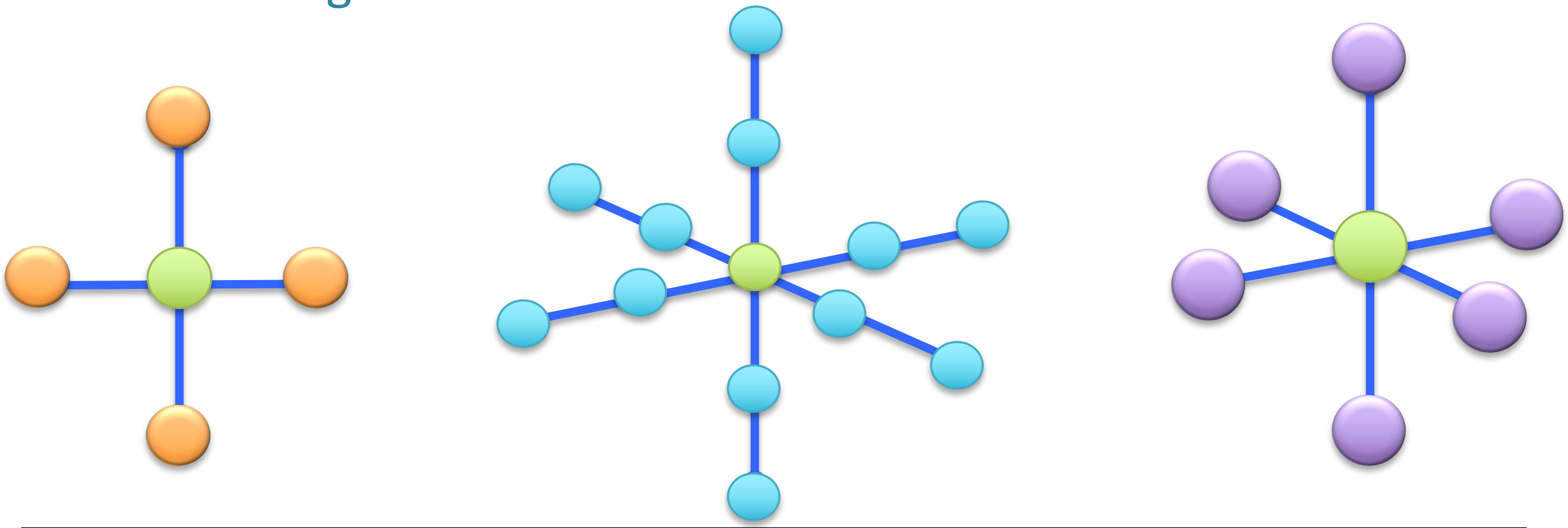
Split-Phased Execution

- Phase 1: initiate communication with Irecv(), Isend()
- Phase 2: synchronize Wait()
- Perform unrelated computations between the two phases
- Each pending irecv()s and isend()s must have a distinct buffer

OVERLAP	NO OVERLAP
Irecv(x,req) ISend(..) Compute(y) Wait(req) Compute(x)	Irecv(x) ISend(…) Wait(x) Compute(x) Compute(y)

Stencil Shape

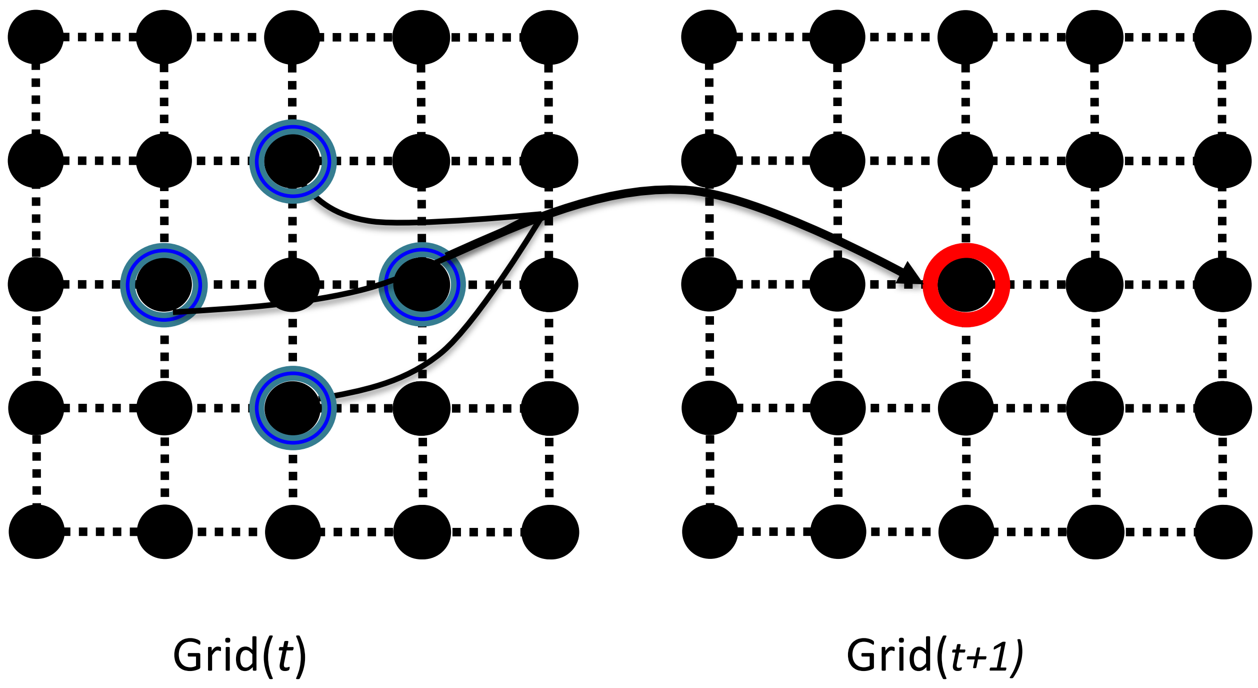
- Stencil shape can be different depending on the numerical algorithm used
- Here 5-point in 2D, 13-point and 7-point in 3D are shown
  - Next value of green point depends on the current value in the nearest neighbors



```
//image smoothing example using 5-point stencil
for iter = 1 : nSmooth
  for (i,j) in (0:N-1, 0:N-1)
       Imgnew [i,j] = (Img[i-1,j]+Img[i+1,j]+Img[i,j-1]+Img[i, j+1])/4
 Swap(Imgnew,Img)
```

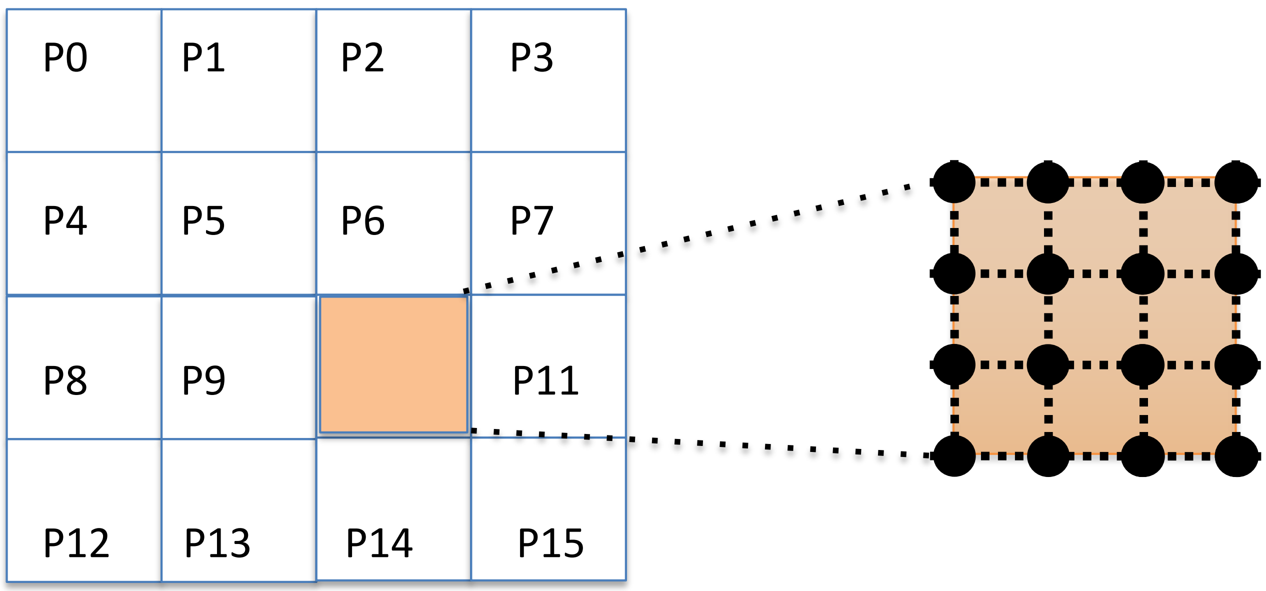
Cell Update

- Since we are updating the next value of a cell from its neighbors, typically we need to keep two grids around
  - Current and Next Grid
- There are techniques to implement with one grid, they are called update-in-place methods



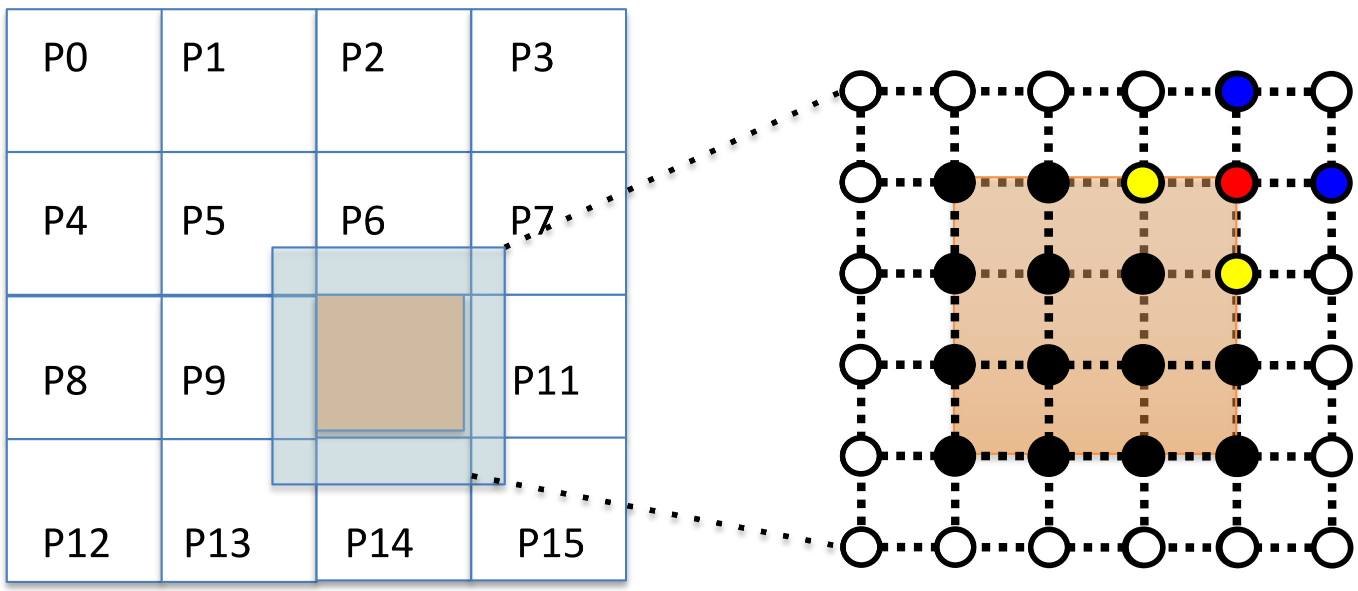
Data Dependences

- Each process has its own data partition
- Do we need any data points from neighboring processes?



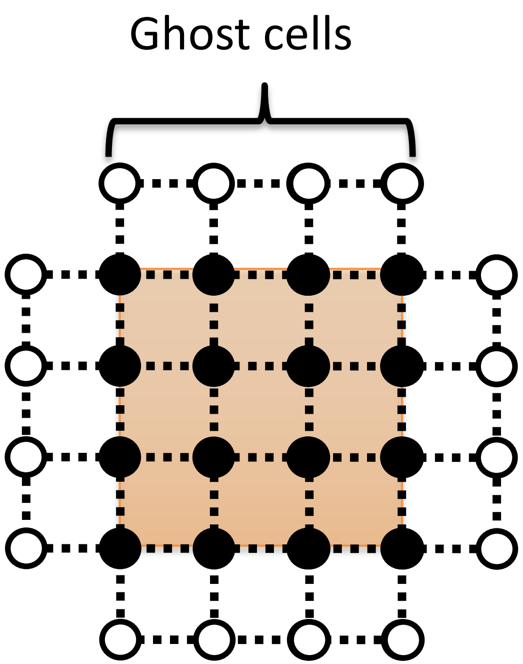
Data Dependences

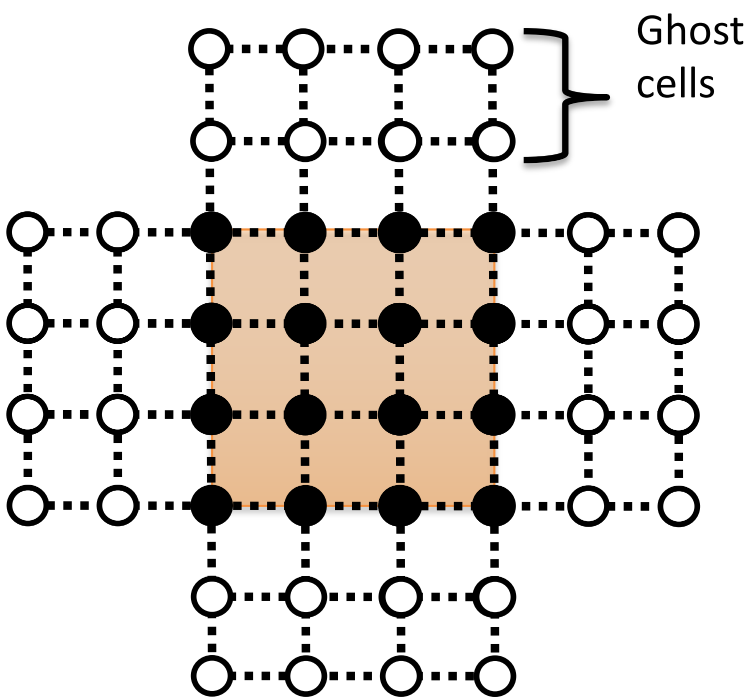
- In order to update red, we need yellow points (local) and blue points from other processes (remote)
  - Too expensive to communication individual array values one at a time
  - “Ghost” cells hold a copies off-process values



Ghost Cells (Halos)

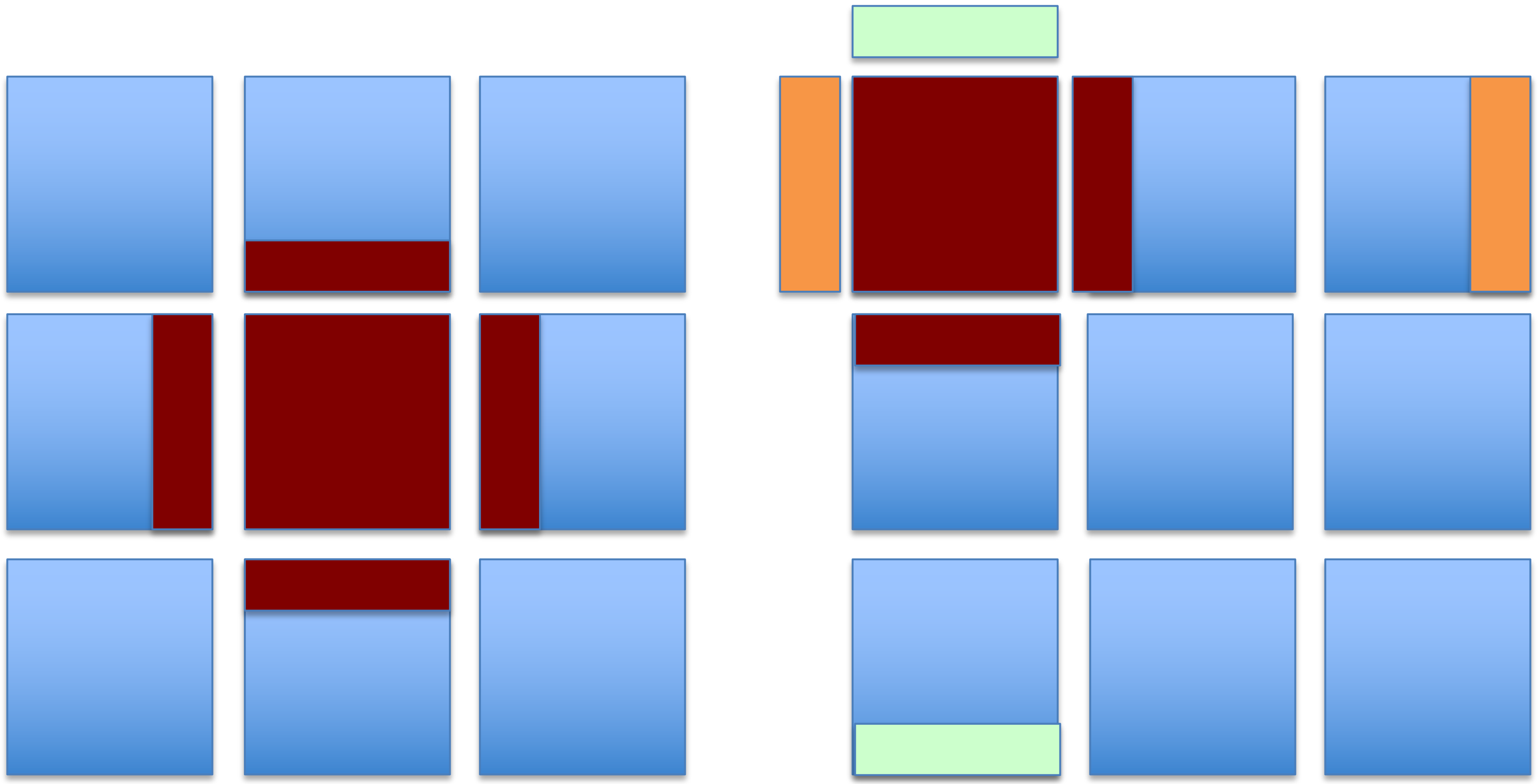
- Ghost cells are allocated for the information needed from other processes
  - Thus each processor allocates $(N+ 2*ghosts)^{d}$ space to accommodate ghost region
  - Depth of ghost region depends on the numerical method used





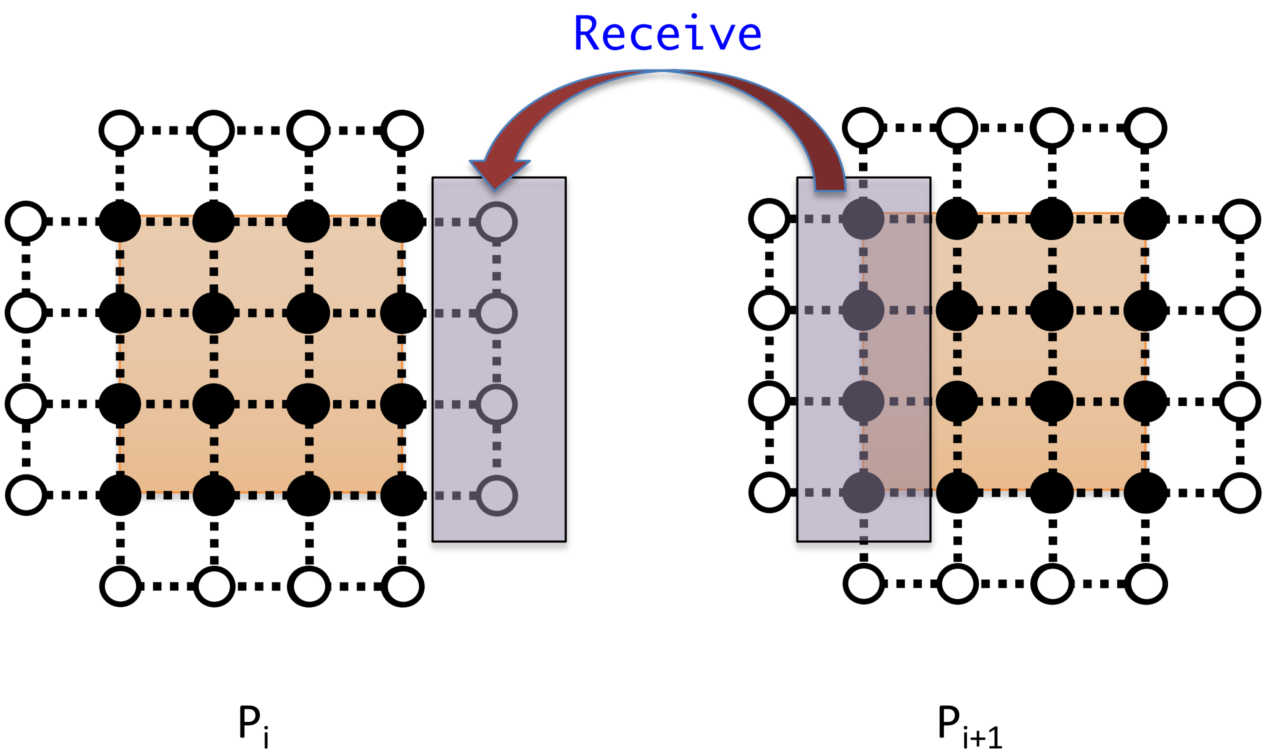
Boundary Updates

- An example of periodic boundary conditions:



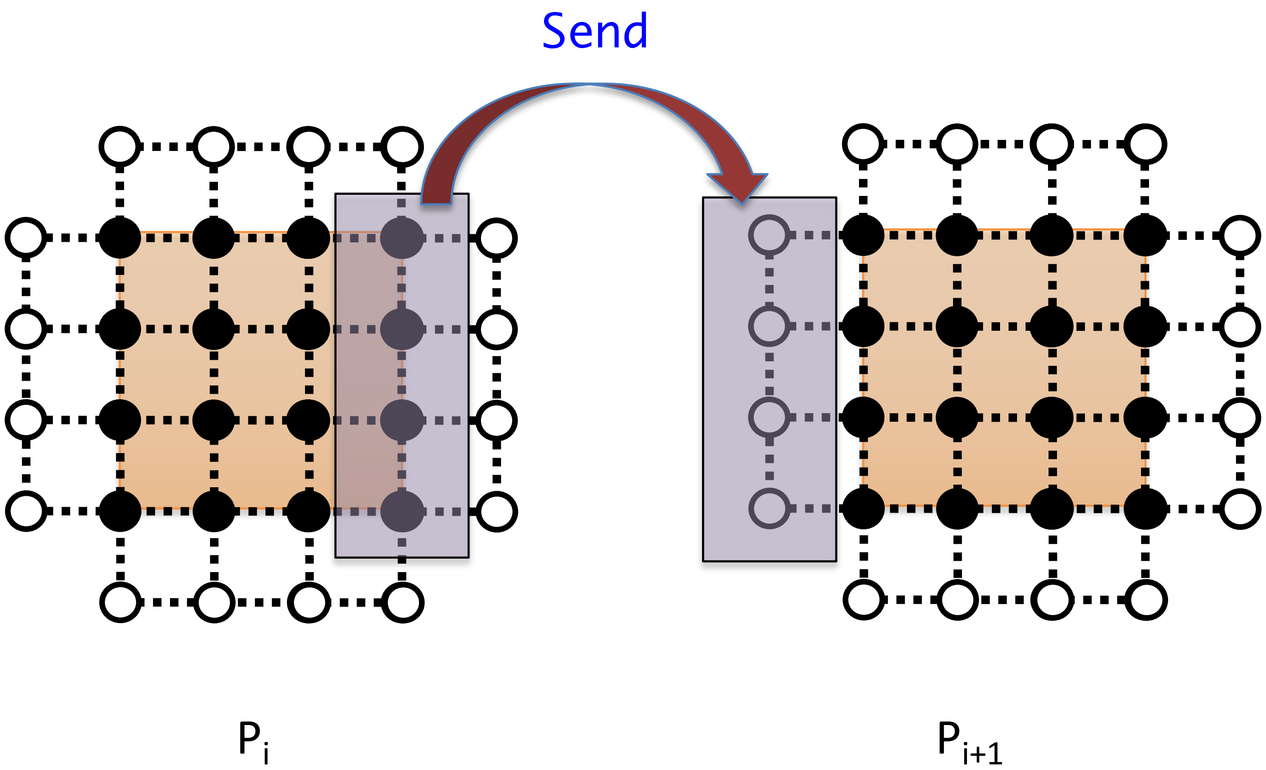
Ghost Cell Exchange- Receive

- Need to update ghosts every iteration so that a process has the latest values
- This is done with message exchange between processes



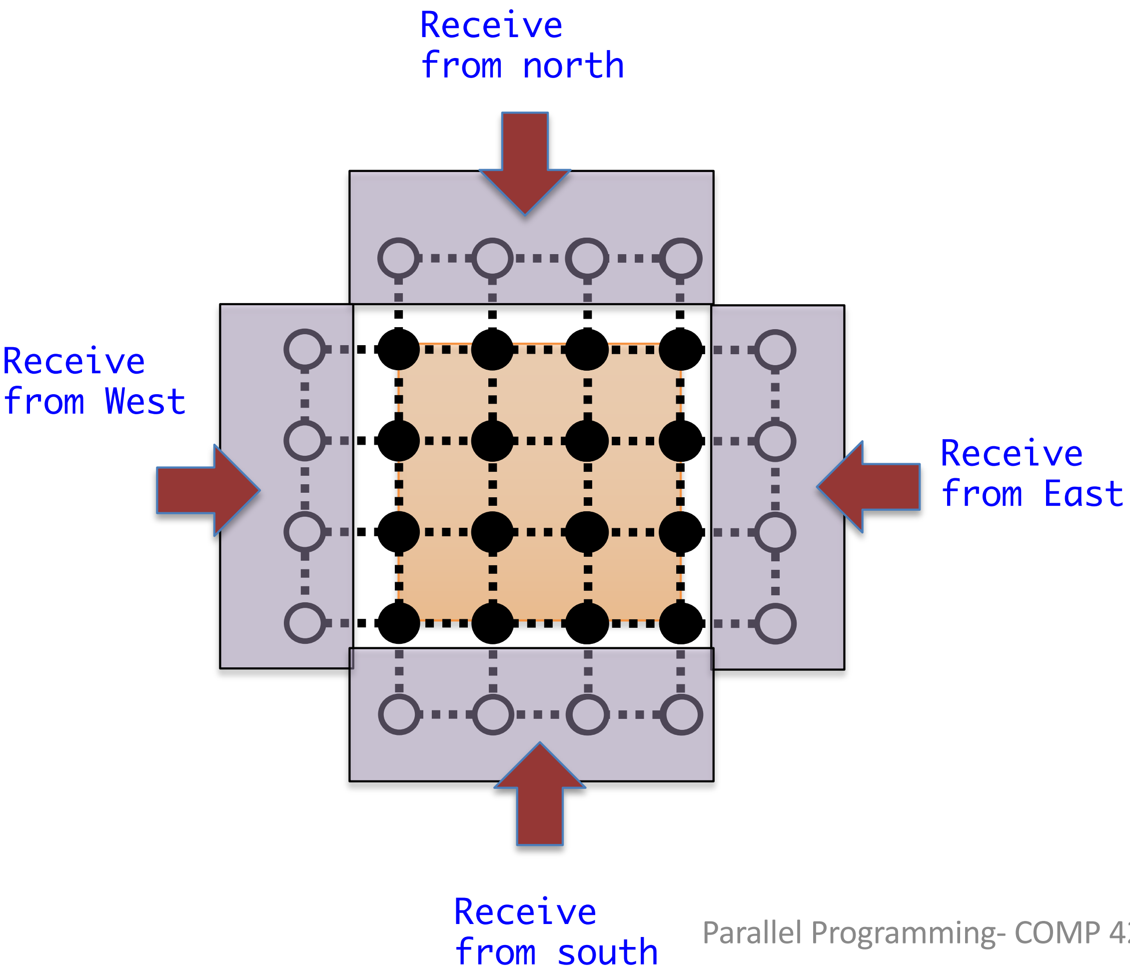
Ghost Cell Exchange- Send

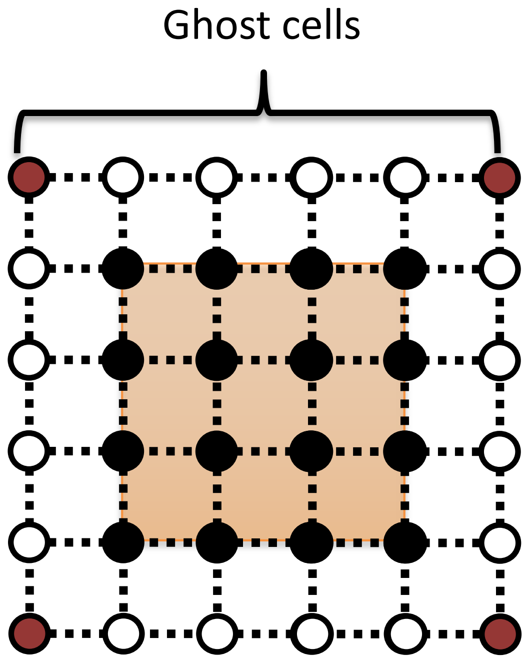
- Need to update ghosts every iteration so that a process has the latest values
- This is done with message exchange between processes
  - Note that the elements may not be contiguous in memory – need to pack a message at the source and unpack at the destination



Msg Exchange with All Neighbors

- How many processors involved in communication depends on the numerical algorithm and domain decomposition
  - In 2D, typically sends/receives involve 4 neighbors (N, S, W, E)
- How many neighbors are there for 3D problems?





If the computation uses the corner cells in the update, then need to exchange ghosts with 4 more processes: NE, NW, SE and SW

Communication Cost

- Communication performance can be a major factor in determining application performance
- **Message passing time = α + N/β**

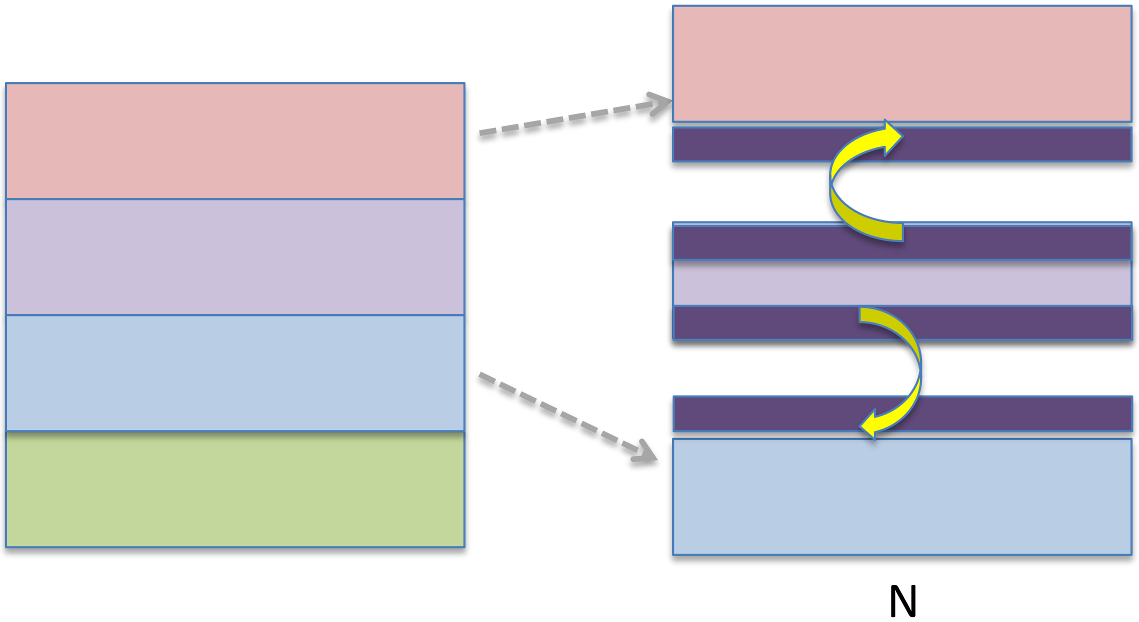
α = message startup time and overhead (latency)
β = network bandwidth (bytes/sec)
N = message size

- Short messages: startup term dominates
- Long messages: bandwidth term dominates

Communication Cost for 1D Geometries

- Assumptions *P* processors divides input size *N* evenly
  - 1 word = double precision floating point = 8 bytes
- For horizontal strips, data are contiguous
  - Row-major C arrays
- Message Size = 16N bytes (sent to North + sent to South) per processor

`Time(comm) = 2(α+8N/β) per processor`

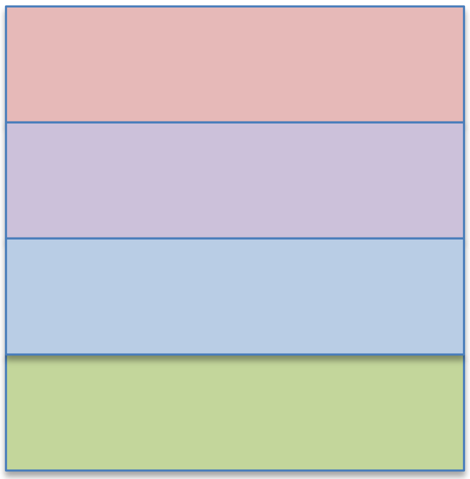


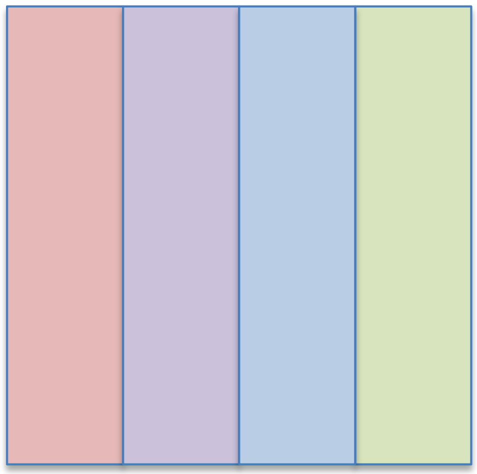
α :latency
β: bandwidth

Communication Cost for 1D Geometries

- For horizontal strips, data are contiguous
  - Row-major C arrays
- For vertical strips, data are not contiguous in memory
- This model doesn't take into account the message packing overhead

Ideal for
Row-major
C arrays



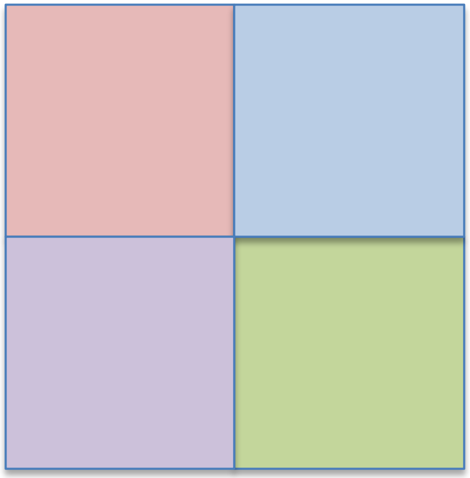


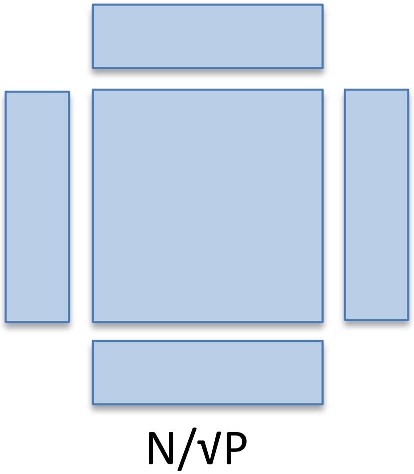
Ideal for
Column-major
arrays

Communication Cost for 2D Geometries

- Assumptions
  - $\sqrt{P}$ divides N evenly
  - 1 word = double precision floating pt. = 8 bytes
- Ignore the cost of packing/unpacking message buffers
- Message size sent in bytes for processor is $4*8N/\sqrt{P}$

$$\text{Time(comm)} = 4(\alpha+8N/(\beta\sqrt{P}))$$





Process Geometry

- Block decomposition is harder to implement than strip decomposition
- **Any good reasons why an application should support block decomposition?**

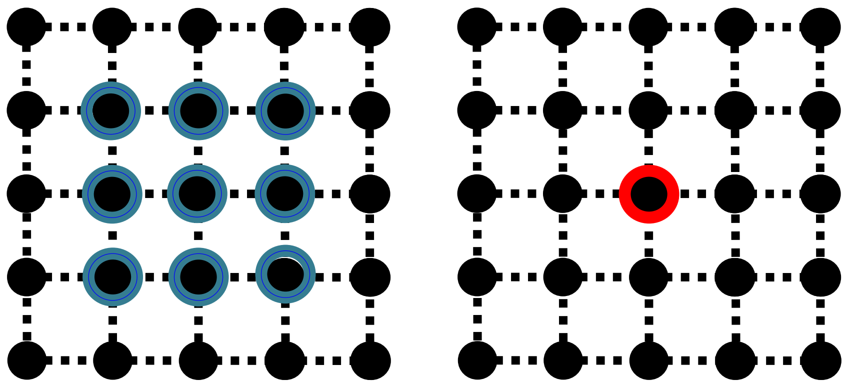
- Typically strip decomposition outperforms block decomposition resulting in lower communication times when

$$2(\alpha+8N/\beta) < 4(\alpha+8N/(\beta\sqrt{P}))$$

- Optimal process geometry depends on the N, β and α.

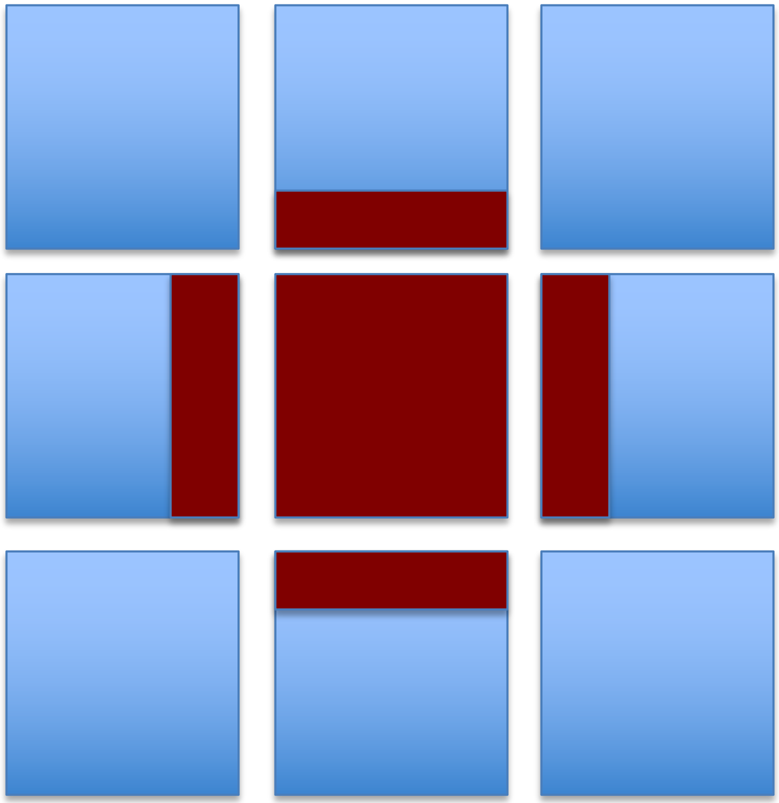
Question

```
for (i=1:N)
    for (j=1:N)
        Unew[i,j] = (-20*U[i,j] + 4*[U[i,j+1]+U[i,j-1] + U[i+1,j]+U[i-1,j]) +
                        U[i+1,j+1] + U[i+1,j-1] + U[i-1,j+1] + U[i-1,j-1]) / (6*h)
swap(U, Unew)
```



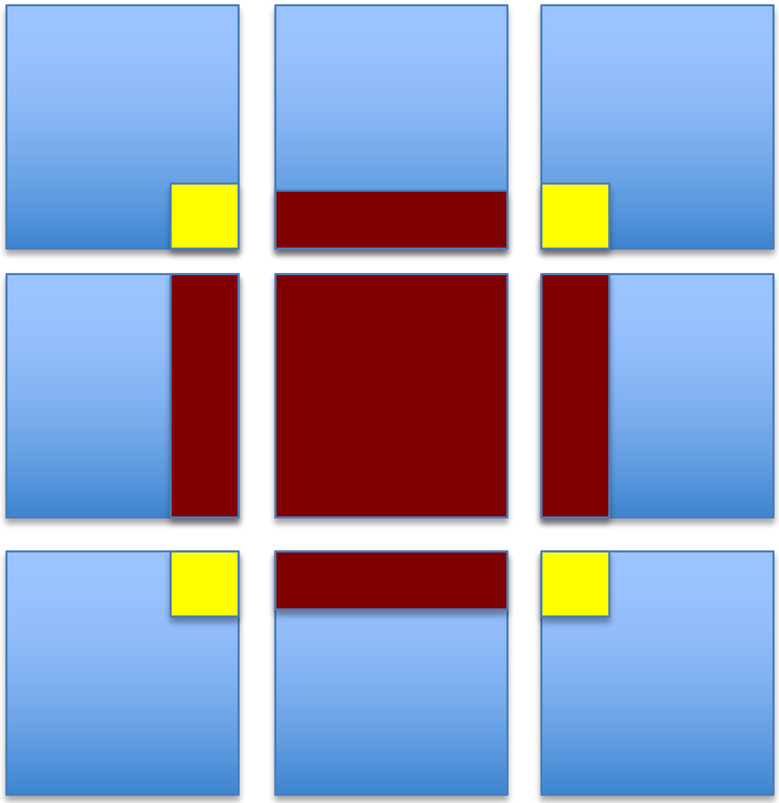
- 9-point stencil
- Case A: Cost model for ghost cell exchange with 8 neighbors
- Case B: Cost model for ghost cell exchange with 4 neighbors

5-point Stencil Case



$$Time(comm) = 4(\alpha + 8(N/\sqrt{P})/\beta)$$

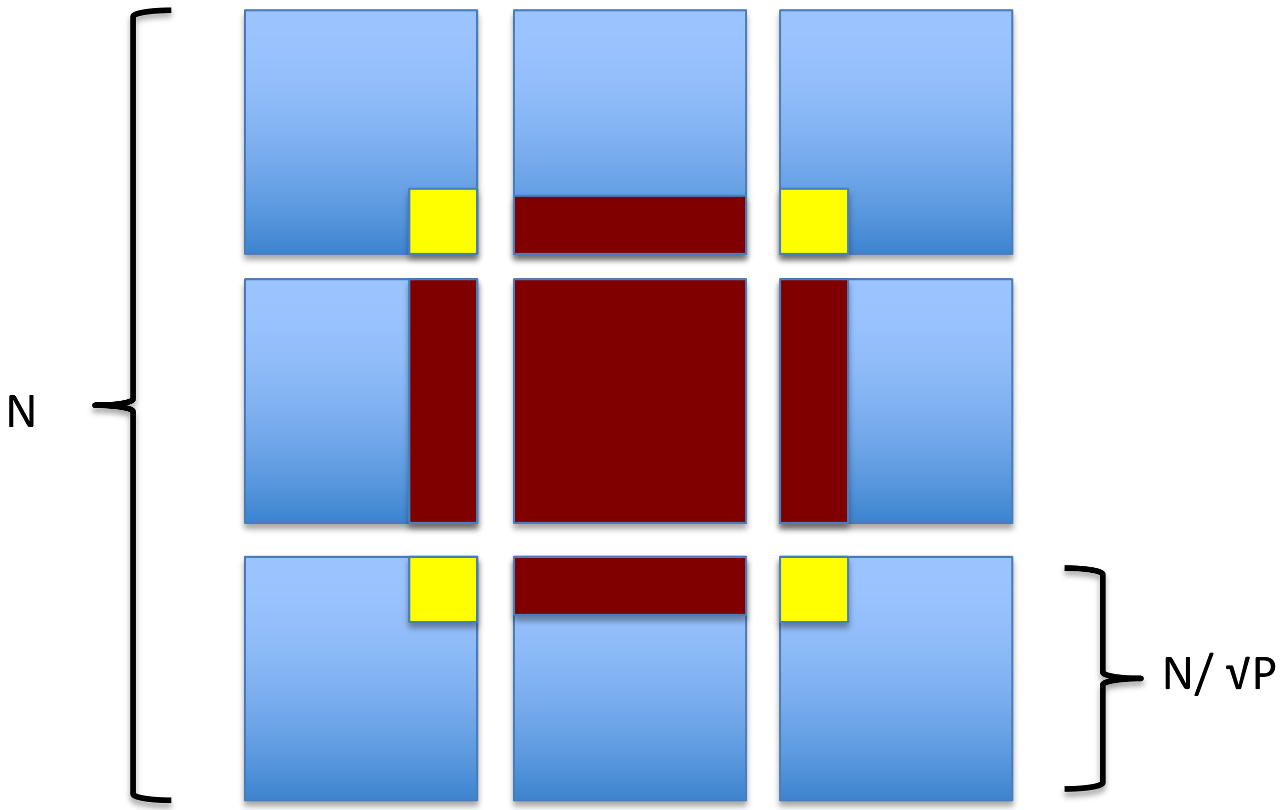
Case A: 9-point Stencil



Need to get data from 8 different neighbors

Input size N*N, number of processors is P

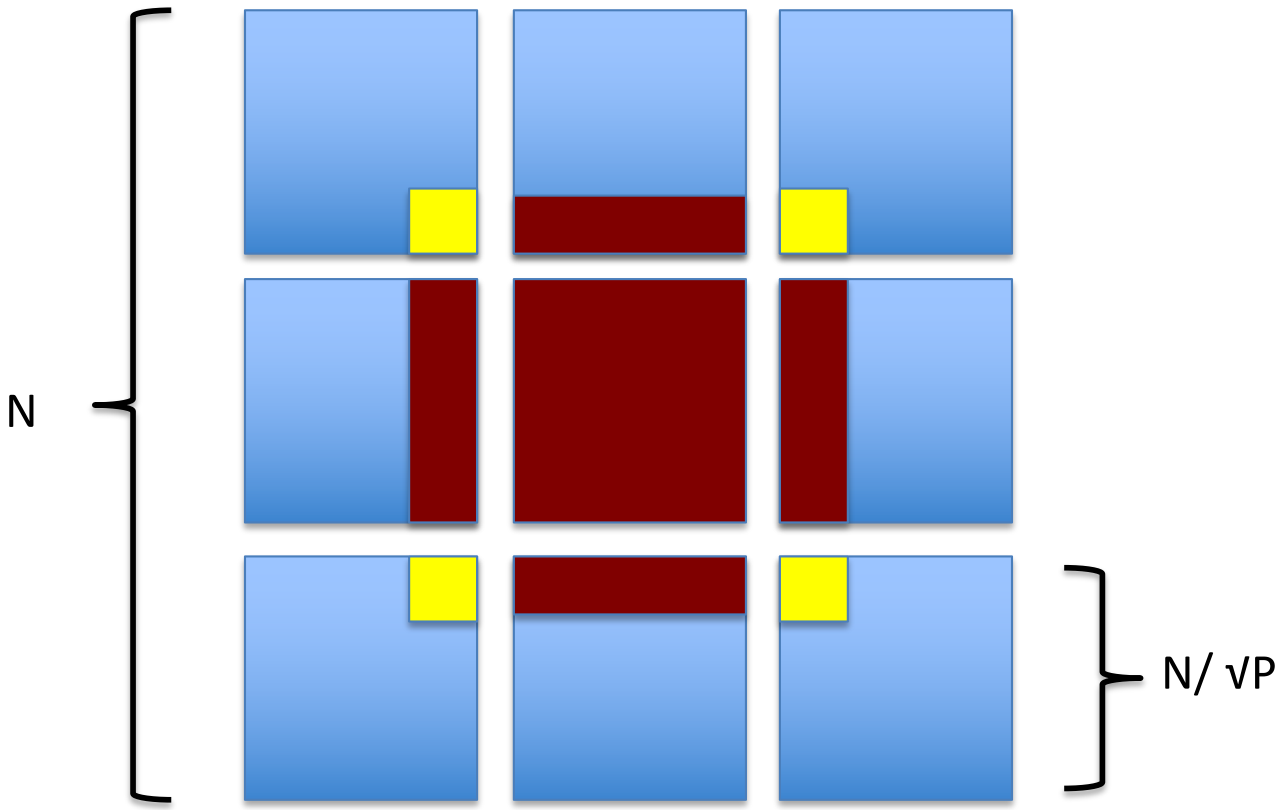
Case A: 9-point Stencil



Comm Cost$= 4\,(\alpha + 8\,(N / \sqrt{P}) / \beta\,) \qquad + 4(\alpha + 8 * 1/\beta)$

Double precision

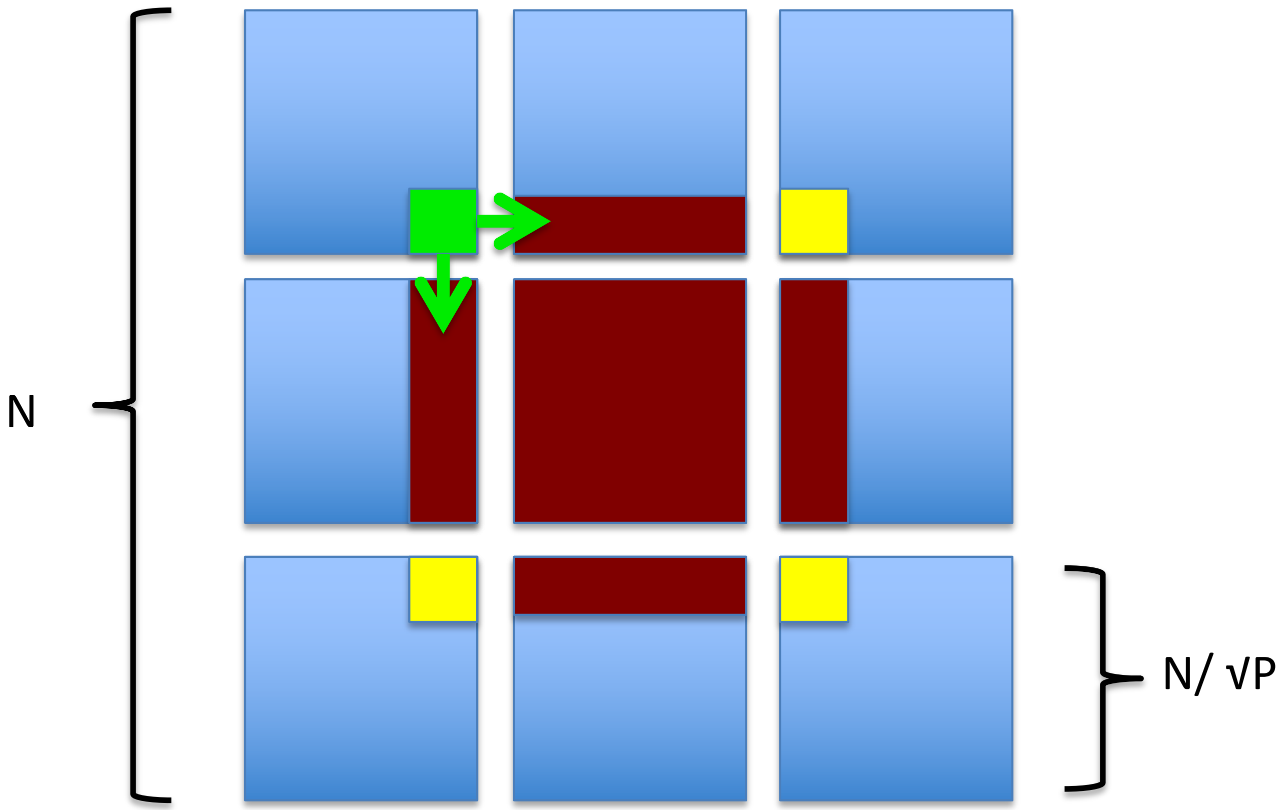
Case B: 9-point Stencil



Can we do better?

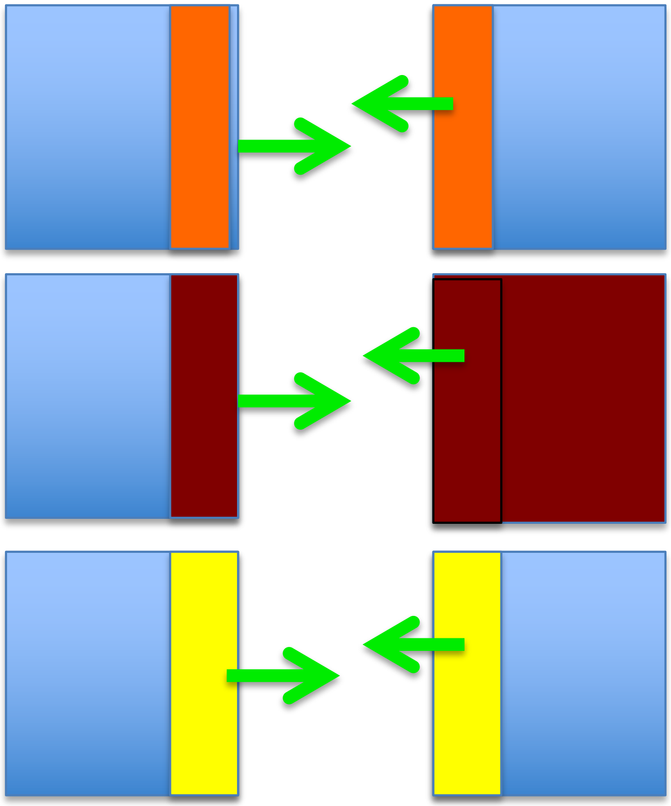
Exchange message with only 4 neighbors!

Case B: 9-point Stencil



Note that the green point is sent to two of my neighbors!
Why don’t I get that point from my North or West neighbor rather than getting it from NW neighbor?

Case B: 9-point Stencil



Showing only exchange with W neighbors.
Assume we are also exchanging with E neighbors in this step

Do message exchange **in two steps**.
First everyone sends ghost cells to W and E or (to N and S)

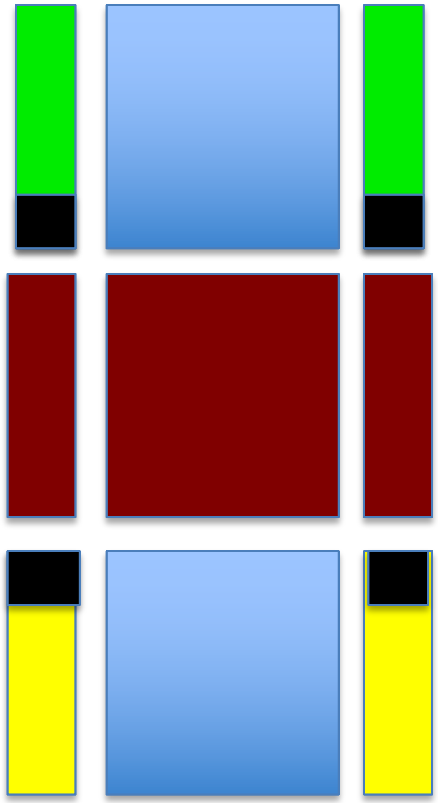
Cost of Step 1

Comm Cost$= 2\ (\alpha + 8\ (N / \sqrt{P}) / \beta\ )$

- 2 because we only communicated with E and W

Case B: 9-point Stencil

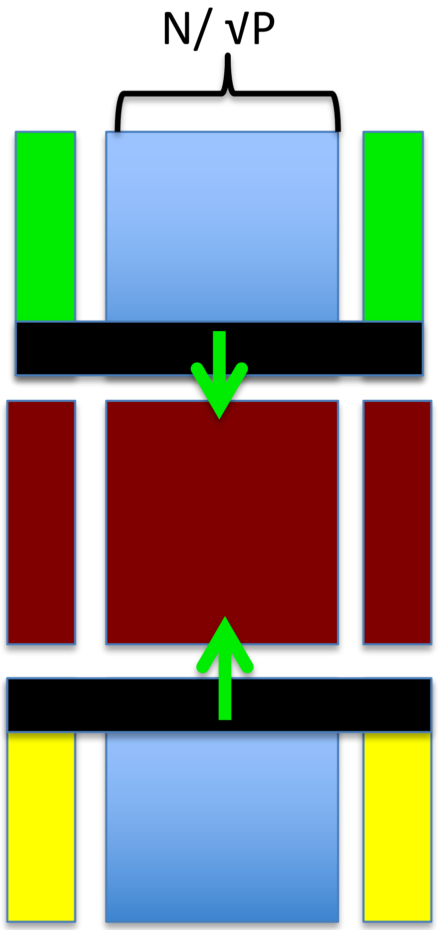
End of First Step, we have new ghost cells on our W and E sides. These include the corner cells that we needed!



In the second step, exchange with N and S but slightly larger data ($N/\sqrt{P} + 2$) that include the corner points

Case B: 9-point Stencil

End of First Step, we new ghost cells on our W and E sides. These include the corner cells that we needed!



Note that step 1 has to finish before we start step 2

In the second step, exchange with N and S but slightly larger data (N/√P + 2) that include the corner points

Step 2

*Comm Cost for Step 2= 2 (α +8 * (N / √P + 2) * 1/β)*

Two more elements

Then Combined Step 1 and 2

Comm Cost = *4 (α + 8 (N / √P) / β)   + 4 * 8 *1/ β*

Comparison: Case A - Case B = 4 α

thus case B is better

Pseudocode

```
while (t < tFinal)
{

      Compute(…);
      Step1:Post communication for East&West neighbors
      Wait for completion of Step1

      Step2:Post communication for South&North neighbors
      Wait for completion of Step2

      t++;
}
```

Note that there is no communication hiding here.

Hiding Communication Overhead

- Non-blocking communication calls can be used to hide communication overhead
  - Also called hiding communication latency

- When used correctly, these primitives are *capable of **overlapping communication overheads** with useful computations*.

- What is the maximum speedup you can get by hiding communication overhead?

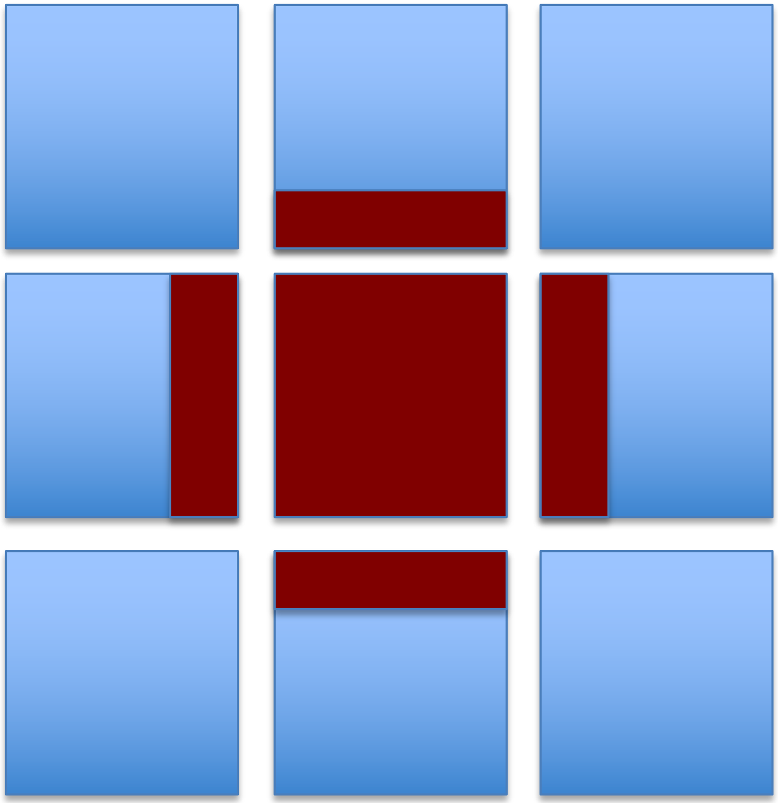
Nearest neighbor exchange in a ring topology with Non-Blocking Messages


```
#include "mpi.h"
#include <stdio.h>
main(int argc, char *argv[])  {
int numtasks, rank, next, prev, recvbuf[2], sendbuf[2], tag1=1, tag2=2;
MPI_Request reqs[4];
MPI_Status stats[4];

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

initBuffers(sendbuf);
prev = rank-1;
next = rank+1;
if (rank == 0)  prev = numtasks - 1;
if (rank == (numtasks - 1))  next = 0;

MPI_Irecv(&recvbuf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD, &reqs[0]);
MPI_Irecv(&recvbuf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD, &reqs[1]);

MPI_Isend(&sendbuf[0], 1, MPI_INT, prev, tag2, MPI_COMM_WORLD, &reqs[2]);
MPI_Isend(&sendbuf[1], 1, MPI_INT, next, tag1, MPI_COMM_WORLD, &reqs[3]);

{  do some work  }

MPI_Waitall(4, reqs, stats);

MPI_Finalize();
}
```

Do some useful work to hide the communication overhead. Work cannot depend on the message buffers!

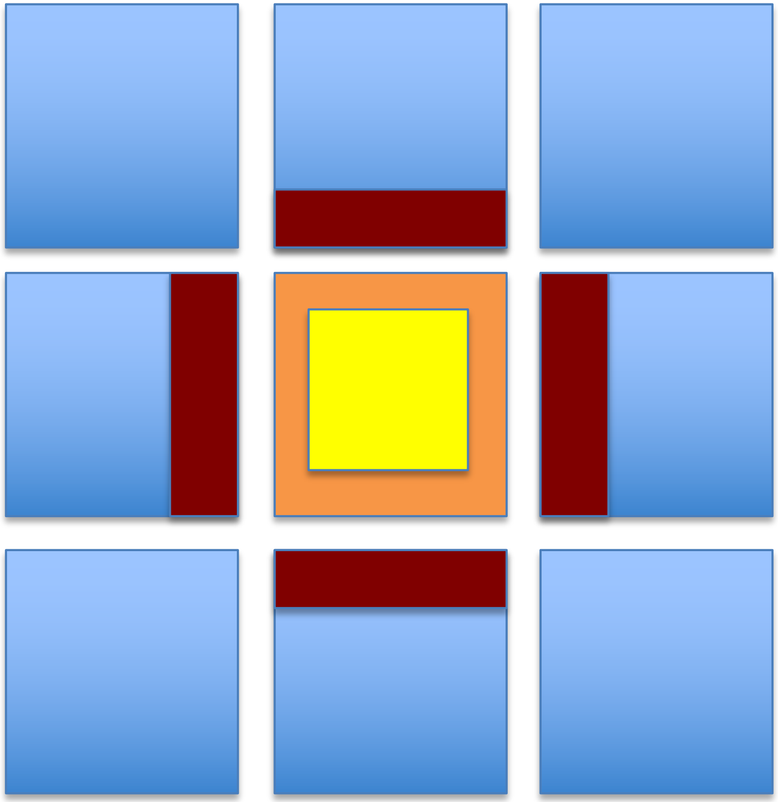
Communication Hiding

- How can we hide the communication in stencil computation?

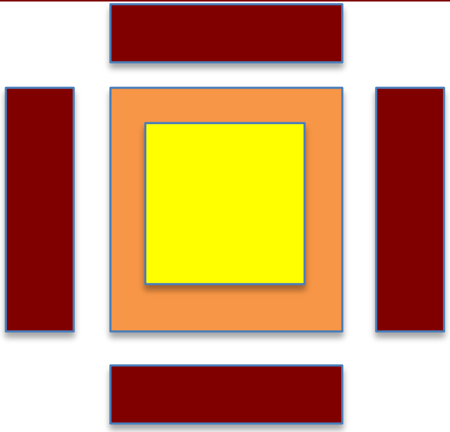


Communication Hiding

- Separate the computation as inner (yellow region) vs outer (orange)
- Outer region requires ghost cells from neighbors but
- Inner region doesn't depend on other processors



Pseudocode for Comm Hiding Version



```
while (t < tFinal)
{

        ComputeOrange(…);
        Post asynchronous comm with neighbors
        ComputeYellow(); // while waiting for comm
        Wait for completion of comm

        t++;
}
```

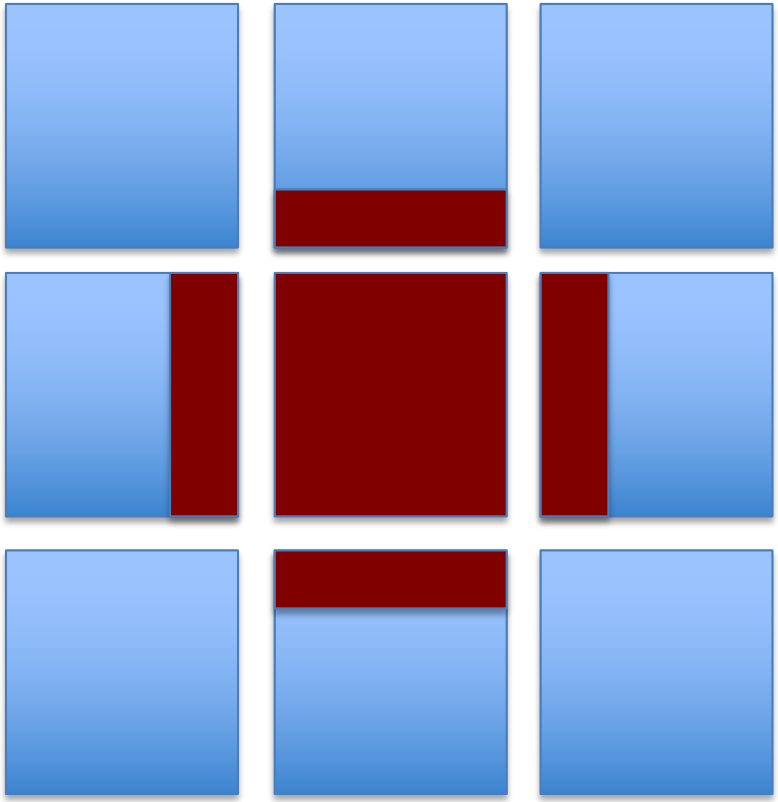
- We can even go more fine-grained and compute N, post comm for N, compute S, post comm for S, ….etc

Communication Avoiding

- Communication avoiding version of stencil method can be implemented by having more ghost cells on each side
  - For example, instead of 1 ghost cell, keep a deeper cost region e.g. 4 ghost cells
  - Exchange message every 4 iterations instead of every
  - This reduces the number of messages but not the message size
  - The goal is to compute for a longer period of time before performing a communication

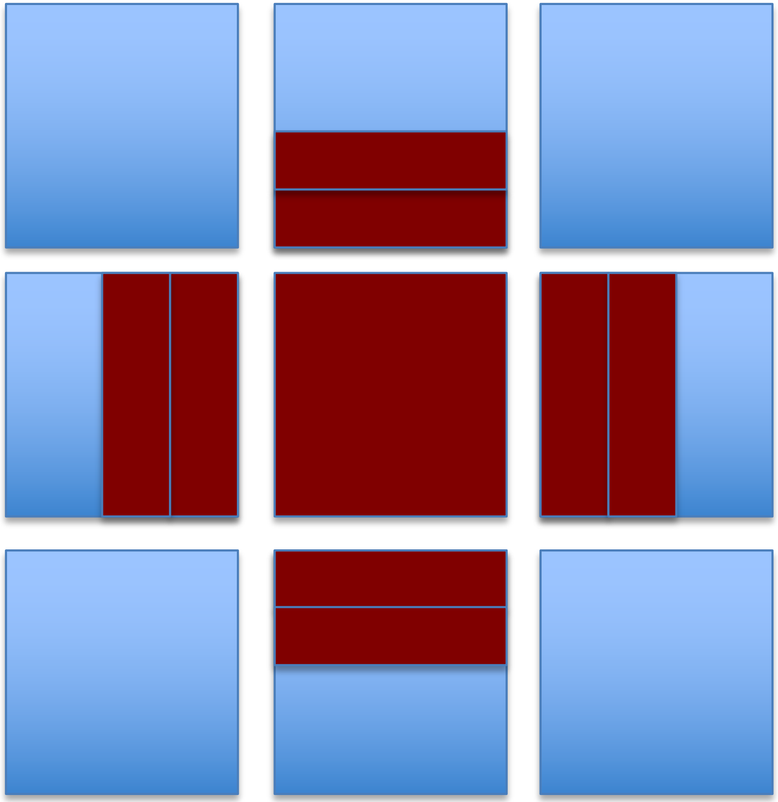
Communication Avoiding

- Ghost cells =1



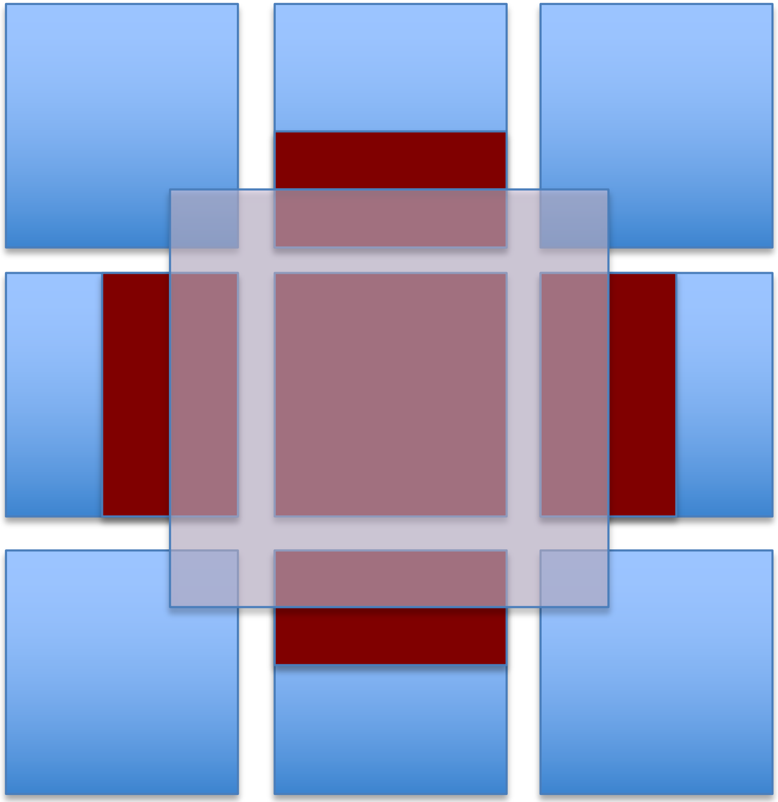
Communication Avoiding

- Ghost cells =2



Communication Avoiding

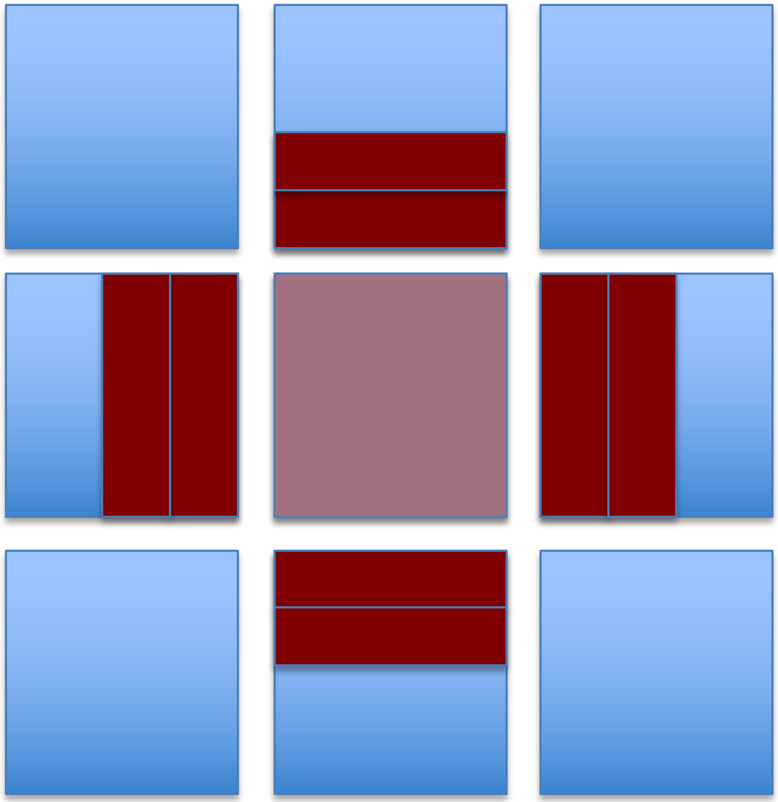
- Compute slightly larger area using the outermost ghost cells



- Note that we redundantly compute the borders because those elements are also computed by another rank

Communication Avoiding

- Compute slightly larger area using the outermost ghost cells and then shrink the area computed and use inner ghost cells



- This works! Why? Because data movement is a lot more expensive than computing

Acknowledgments

- These slides are inspired and partly adapted from
  - Metin Aktulga (Michigan State Univ.)
  - Scott Baden (UCSD)
  - The course book (Pacheco)
  - https://computing.llnl.gov/tutorials/mpi/