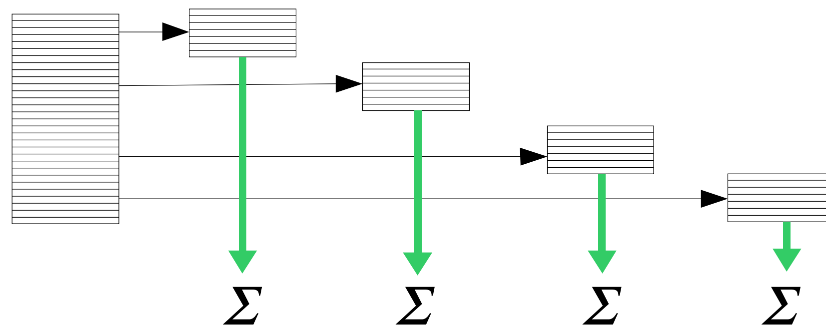
What is Parallel Computing?

- A mechanism for <u>speeding up</u> computation
- Multiple processes work together to solve a problem
- Multiple processors allow multiple processes to run at the same time (in parallel)

Example: compute sum of 1M numbers

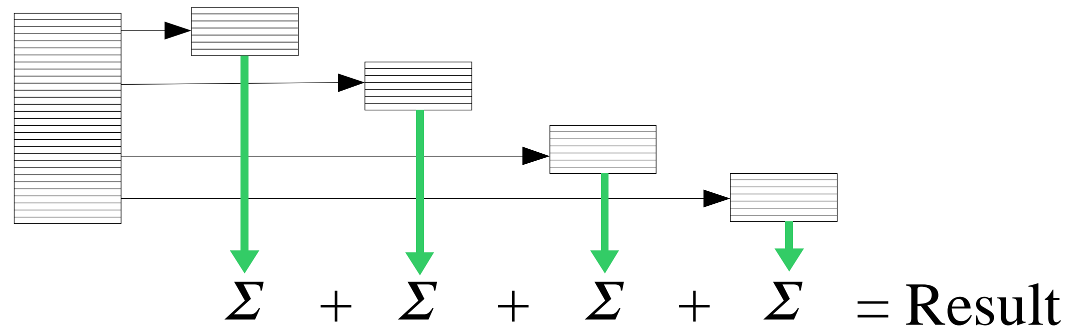
with 4 processors - runs 4 times faster!



copyright © 2004 Walter B. Ligon III, all rights reserved

Interesting Parallel Programs Need Communication

- Summarize or combine results
- Propagate updates across processors
- Distribute work to processors
- Handle boundary conditions



copyright © 2004 Walter B. Ligon III, all rights reserved

Two Ways to Communicate

- Pass Messages
  - Explicitly send
  - Explicitly receive
- Share Memory
  - Read and write shared variables
    - Data implicitly passed between processes
  - Explicitly control access to shared variables
    - Prevent inconsistent state
    - Prevent race conditions

copyright © 2004 Walter B. Ligon III, all rights reserved

Message Passing Semantics

- Primary functions:
  - Send Data (what data, where to send it)
  - Receive Data (who to receive from, where to put it)
- Many subtle shades to consider:
  - synchronization
  - buffering
  - naming
  - data size and type

copyright © 2004 Walter B. Ligon III, all rights reserved

Synchronization

- *blocking* - operation might block until other task makes progress
- *non-blocking* - operation will not block, but might fail if it must otherwise wait
- *asynchronous* - occurs "in the background" concurrently with main thread
- *synchronous* - requires that both sender and receiver read send/receive before either can complete

copyright © 2004 Walter B. Ligon III, all rights reserved

Buffering

- *No buffering* - requires synchronous comm
- *Infinite buffering* - make non-blocking
- *Partial buffering* - might block, might not
- *Explicit buffering* - user guarantees enough buffer (thus non-blocking)

copyright © 2004 Walter B. Ligon III, all rights reserved

Naming

- *Direct* - processes named each other directly
- *Indirect* - send and receive via a “mailbox”
- *Symbolic* - processes refer to each other with a symbol or logical number
- *Symmetric* - both processes must name the other
- *Asymmetric* - sender must name destination, receiver receives sender's ID

copyright © 2004 Walter B. Ligon III, all rights reserved

Data Size and Type

- Fixed message size
- Variable message size
- Infinite data stream
- Bytes only
- Complex types
- Non-contiguous access

copyright © 2004 Walter B. Ligon III, all rights reserved

Collective Message Passing

- Involves a group of processes
  - manage process naming
  - manage process groups
- Compute while communicating
  - Summarizing, Searching
- Re-arrange data
  - Distribute data
  - Gather data
  - Move data around

copyright © 2004 Walter B. Ligon III, all rights reserved

Message Passing Systems

- TCP/IP
- UDP/IP
- GM (Myrinet)
- VIA
- PVM
- MPI
  - MPI 1.1
  - MPI 2.0

copyright © 2004 Walter B. Ligon III, all rights reserved

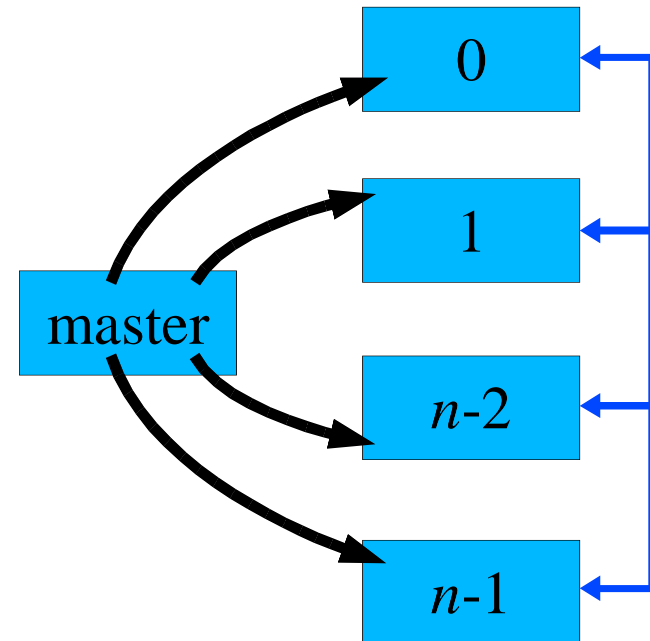
The MPI Interface

- MPI is an *interface* standard
  - Is **not** a specific implementation
  - Does not specify much about processes
- MPI designed for parallel computing
  - Not very good for general purpose messaging
- Two “levels” of implementation:
  - MPI 1.1: basic level
  - MPI 2.0: more advanced features

copyright © 2004 Walter B. Ligon III, all rights reserved

An MPI Job

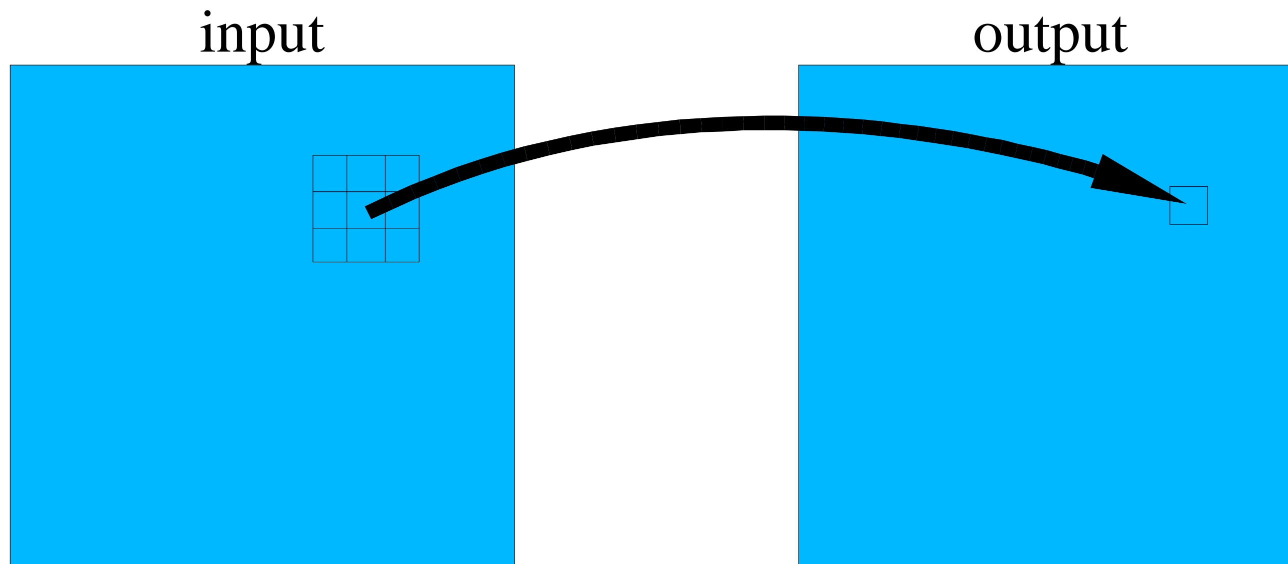
- A Job creates *n* copies of your program (tasks)
- One process stays on the master node to manage IO
- Tasks can send/receive messages to/from the other tasks



copyright © 2004 Walter B. Ligon III, all rights reserved

Example - Image Smoothing

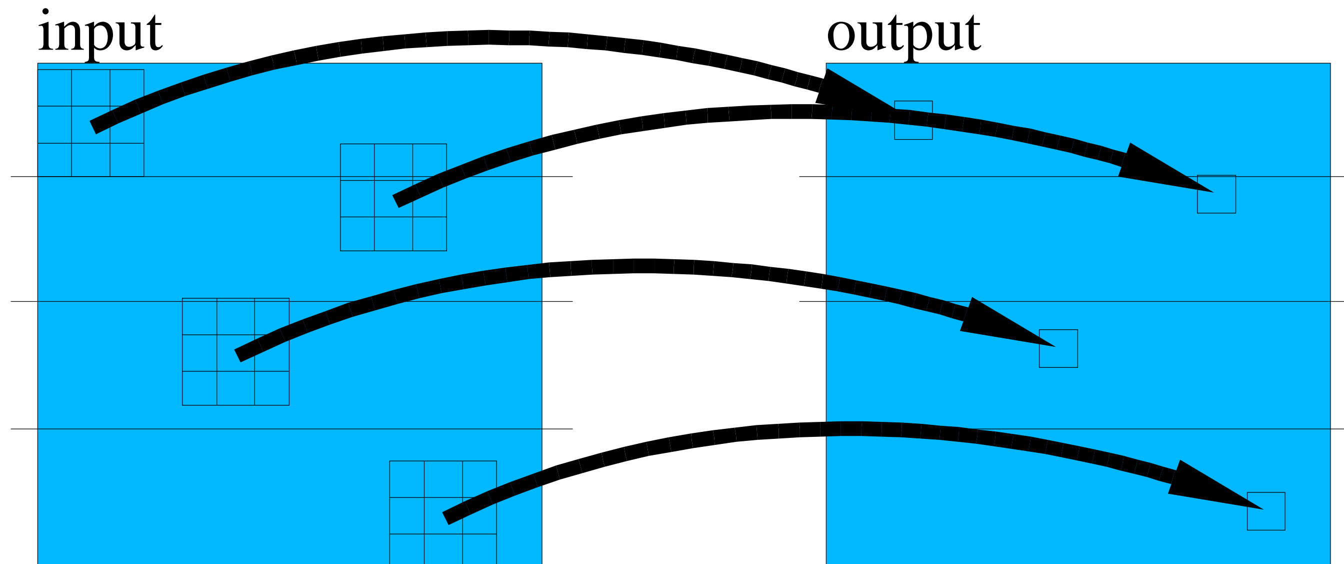
- Image data is modified to reduce noise
- Each pixel replaced by the average of the 8 surrounding pixels, and itself



copyright © 2004 Walter B. Ligon III, all rights reserved

Parallel Smoothing

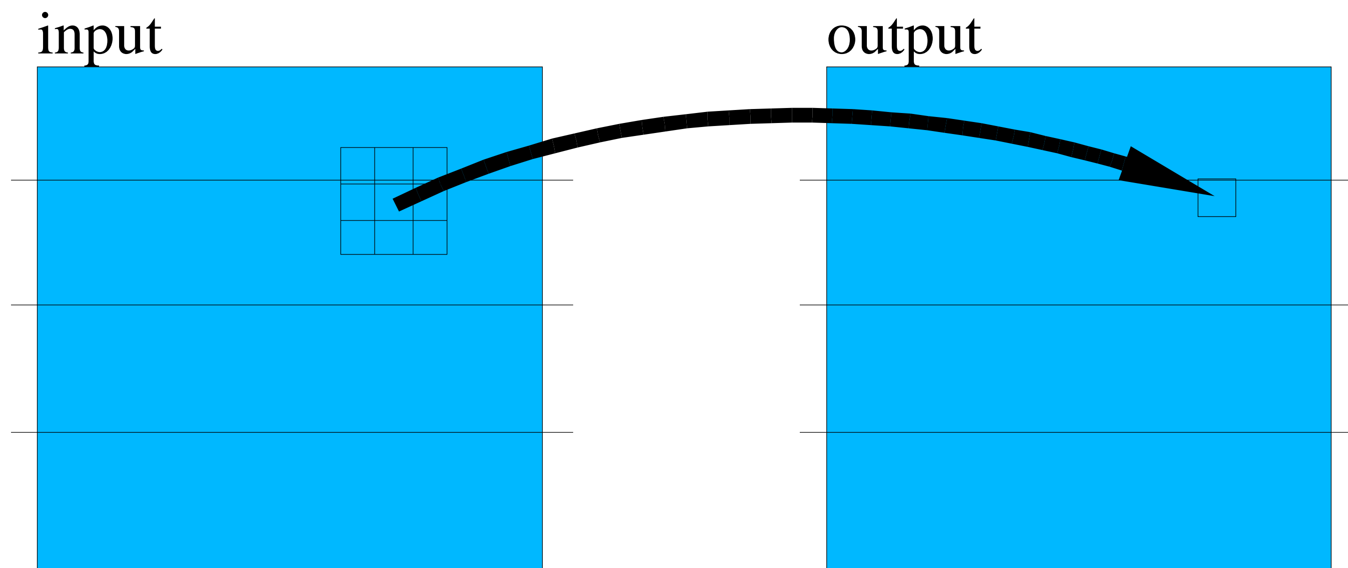
- Image data divided among tasks
- Each task smooths its portion



copyright © 2004 Walter B. Ligon III, all rights reserved

Border pixels need overlapping data

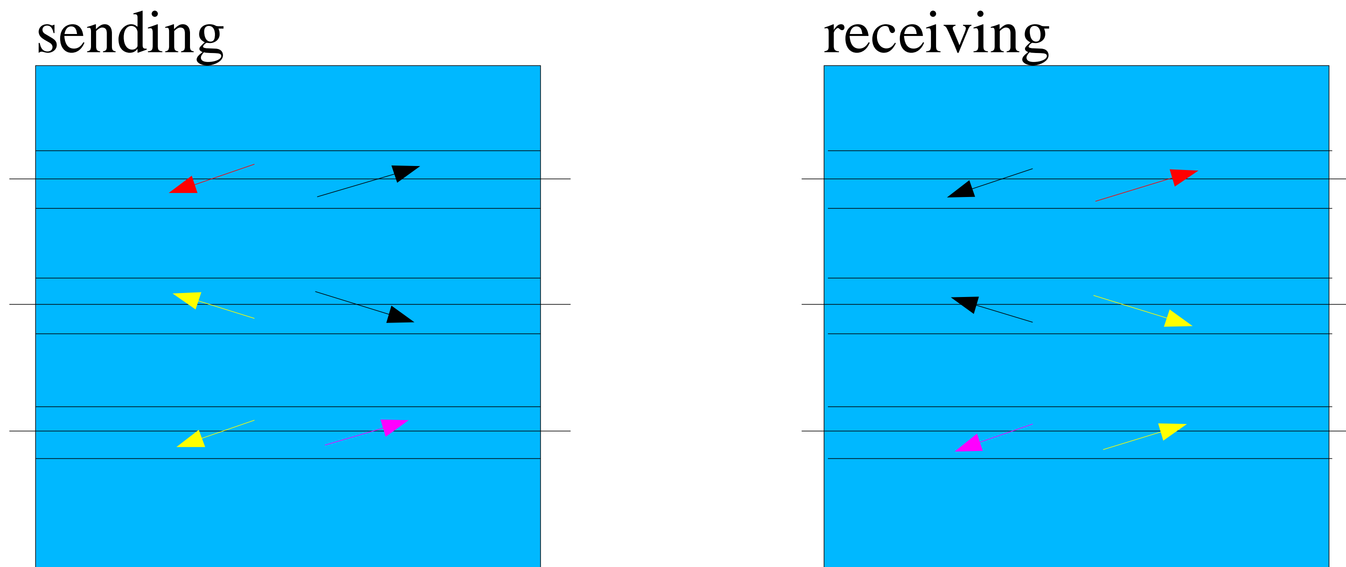
- Data is required from the other tasks
- Other tasks require data as well



copyright © 2004 Walter B. Ligon III, all rights reserved

Tasks exchange border data

- Each task sends to the other tasks
- Each task receives from the other tasks
- When more tasks, each exchanges with 8 adjacent neighbors



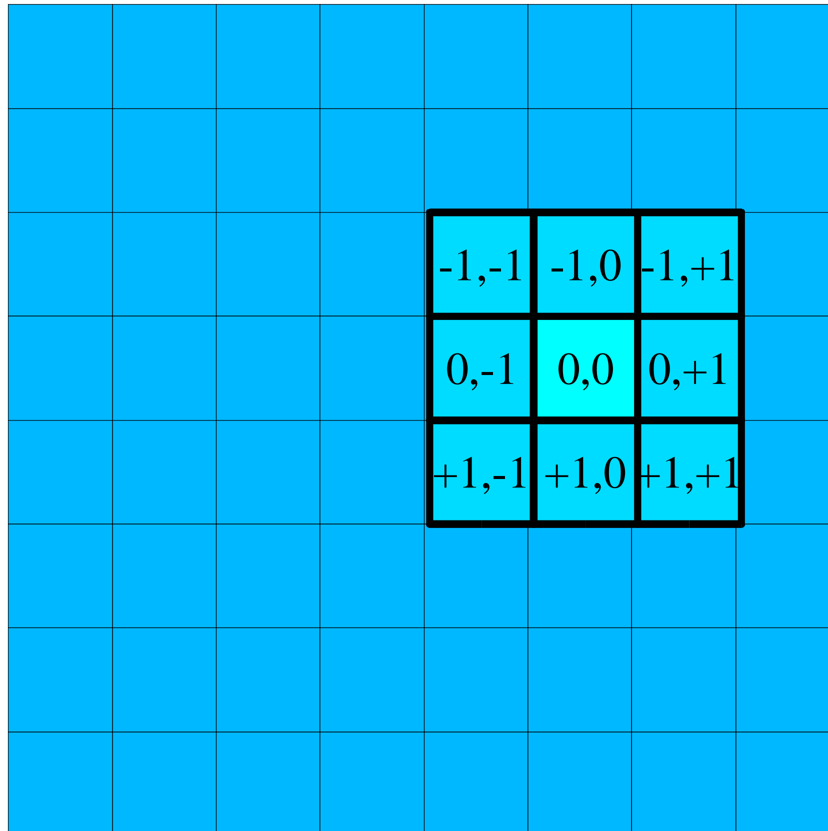
copyright © 2004 Walter B. Ligon III, all rights reserved

Code for Smoothing Program

```
for (r = 0; r < n; r++)
    for (c = 0; c < n; c++)
    {
        cnt = 0;
        sum = 0;
        for (rm = -1; rm < 2; rm++)
        {
            if (r+rm < 0 || r+rm >= n)
                continue;
            for (cm = -1; cm < 2; cm++)
            {
                if (c+cm < 0 || c+cm >= n)
                        continue;
                sum += input[r+rm][c+cm];
                cnt++;
            }
        }
        output[r][c] = sum / cnt;
    }
```


copyright © 2004 Walter B. Ligon III, all rights reserved

Smoothing Program



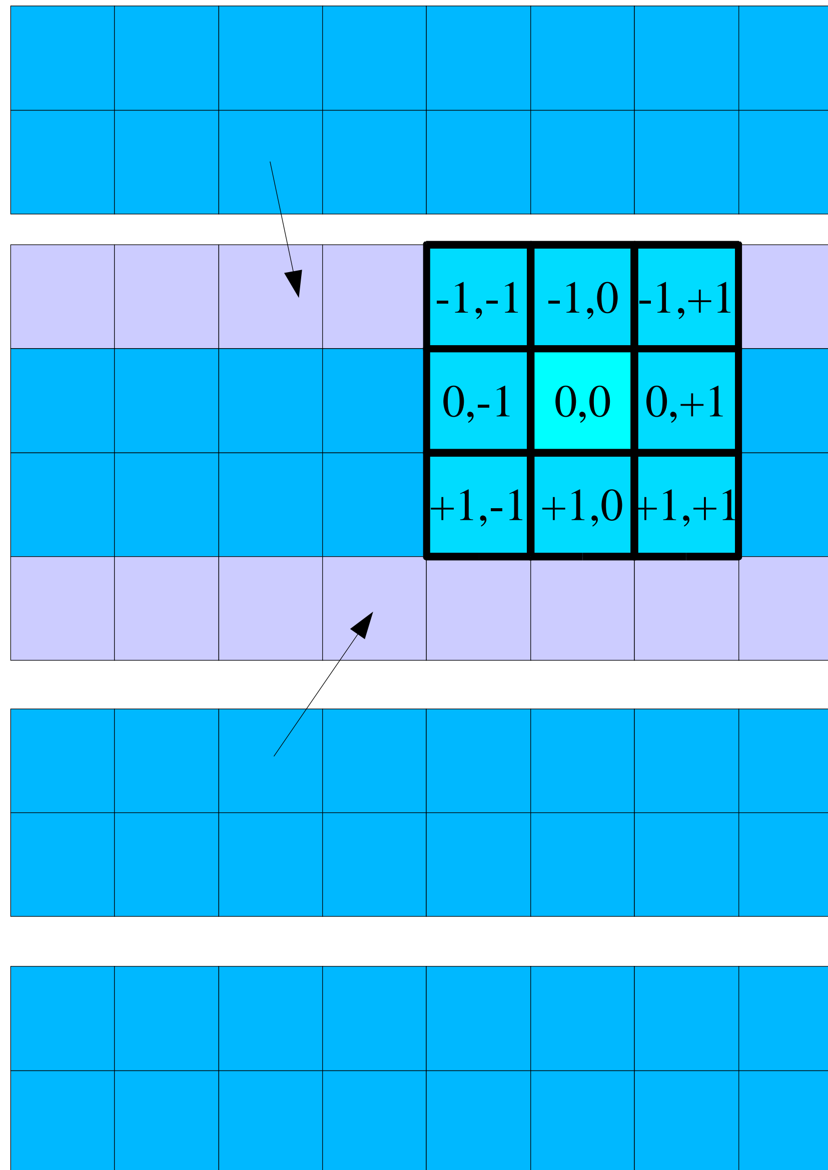
copyright © 2004 Walter B. Ligon III, all rights reserved

Dividing the Data

-1,-1	-1,0	-1,+1
0,-1	0,0	0,+1
+1,-1	+1,0	+1,+1

copyright © 2004 Walter B. Ligon III, all rights reserved

Border Cells



copyright © 2004 Walter B. Ligon III, all rights reserved

Code for Smoothing Program

```
exchange_borders(n, SIZE, RANK, input);
for (r = 1; r < (n/SIZE)+1; r++)
    for (c = 0; c < n; c++)
    {
        cnt = 0;
        sum = 0;
        for (rm = -1; rm < 2; rm++)
        {
            if (RANK == 0 && r+rm < 1 ||
                    RANK == SIZE-1 && r+rm > n/SIZE)
                continue;
            for (cm = -1; cm < 2; cm++)
            {
                if (c+cm < 0 || c+cm >= n)
                        continue;
                sum += input[r+rm][c+cm];
                cnt++;
            }
        }
        output[r][c] = sum / cnt;
    }
```


copyright © 2004 Walter B. Ligon III, all rights reserved

Exchange Code

```
exchange_borders(int n; int SIZE, int RANK, int input[][n])
{
   if (RANK < SIZE-1)
   {
      send(RANK+1, &input[1][0], n*sizeof(data));
      recv(RANK+1, &input[0][0], n*sizeof(data));
   }
   if (RANK > 0)
   {
      send(RANK-1, &input[n/size][0], n*sizeof(data));
      recv(RANK-1, &input[n/size+1][0], n*sizeof(data));
   }
}
```


copyright © 2004 Walter B. Ligon III, all rights reserved