CIS 431/531 Intro to Parallel Computing

Collectives & Message Passing Interface (MPI)

Collectives

Collective operations deal with a **collection** of data as a whole, rather than as **separate** elements

Collective patterns include

- Reduce (Parallel Patterns - parallel control patterns)
- Scan (Parallel Patterns - parallel control patterns)
- Scatter (Parallel Patterns - parallel data management patterns)
- Gather (Parallel Patterns - parallel data management patterns)

Reduce

Reduce is used to combine a collection of elements into **one** summary value

A combiner function combines elements pairwise - it only needs to be **associative** to be parallelizable

$A + (B + C) = (A + B) + C$

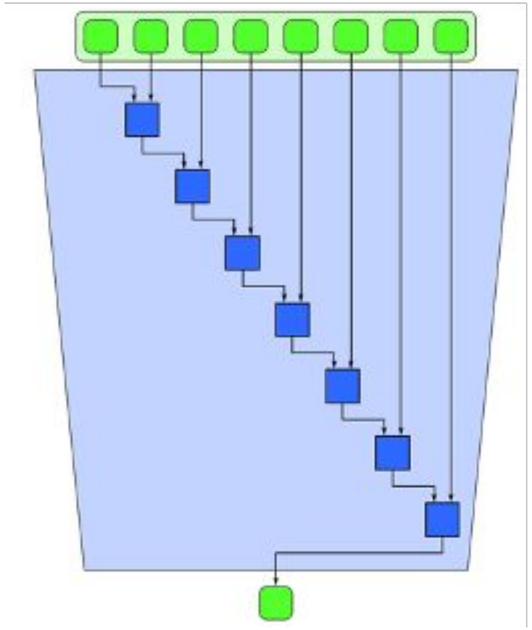
$A + B + C + D = (A + B) + (C + D)$

Examples of combiner functions

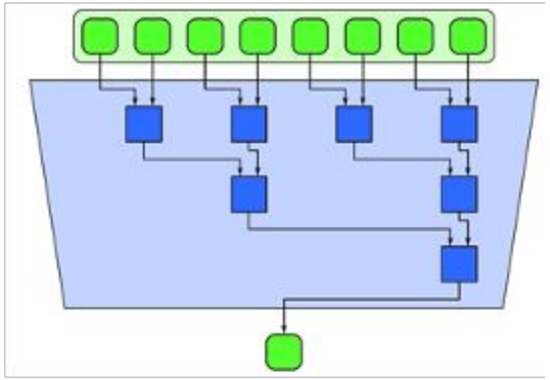
- Add (e.g., prefix-sum)
- Multiplication
- Max/min

Reduce

Serial Reduction

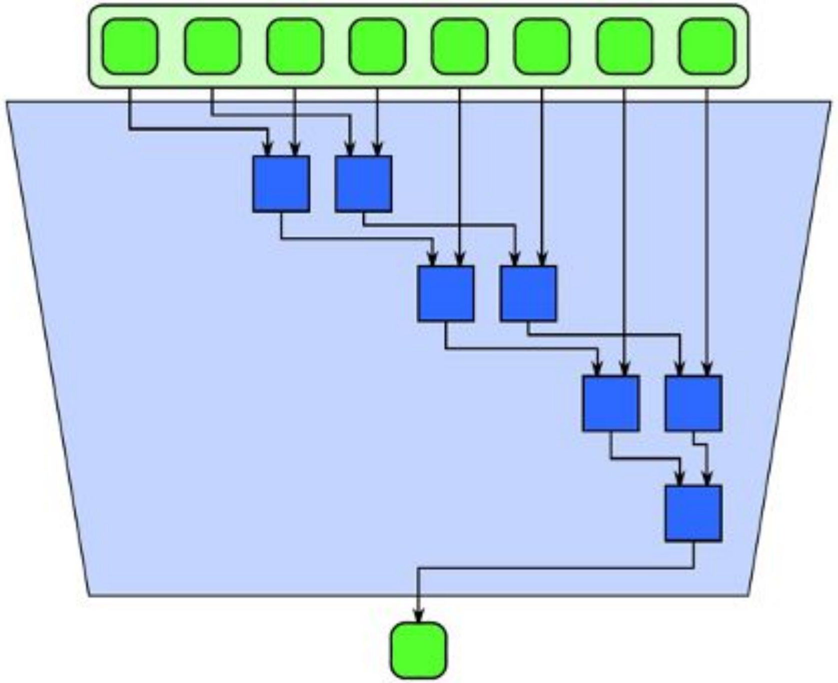


Parallel Reduction



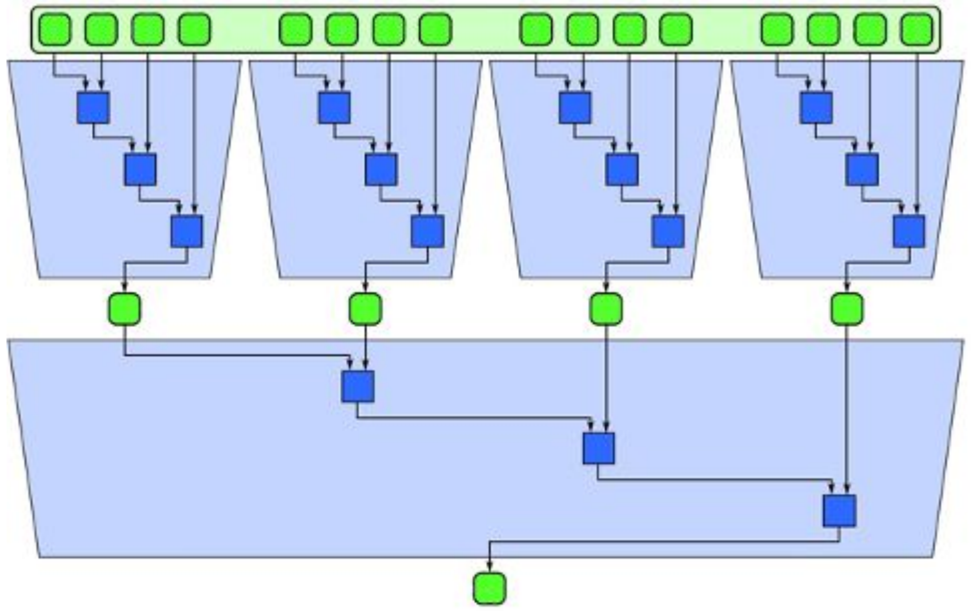
Reduce

Vectorization (serially)



Reduce

Tiling - break work into “chunks” and then reduce (serially)



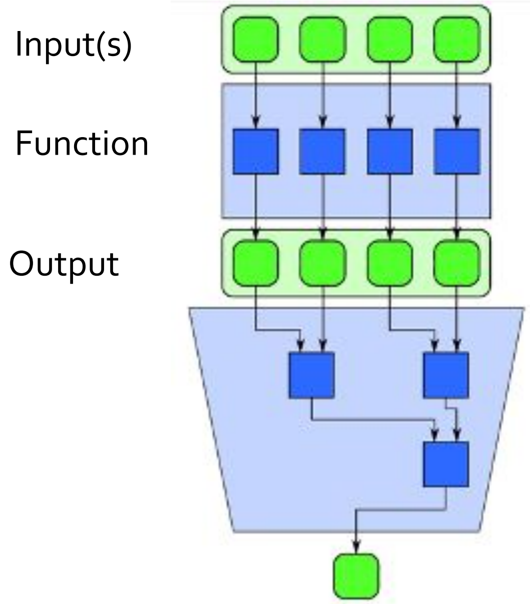
Map & Reduce

Map - perform a function over every element of a collection of data

Example - scaling an array by a constant or adding two arrays

We can also **combine** different patterns together

Example: Dot product



$$\mathbf{a} \cdot \mathbf{b} = \sum^{n-1} a_i b_i.$$

Prefix-sum

Is prefix sum a reduction?

Scan

The **scan** collective produces partial reduction of input sequence to generate a **new sequence**

Trickier to parallelize than reduce

Inclusive vs. exclusive

Inclusive - includes current element in partial reduction

Exclusive - excludes current element in partial reduction & partial reduction is of all prior elements (to the current element)

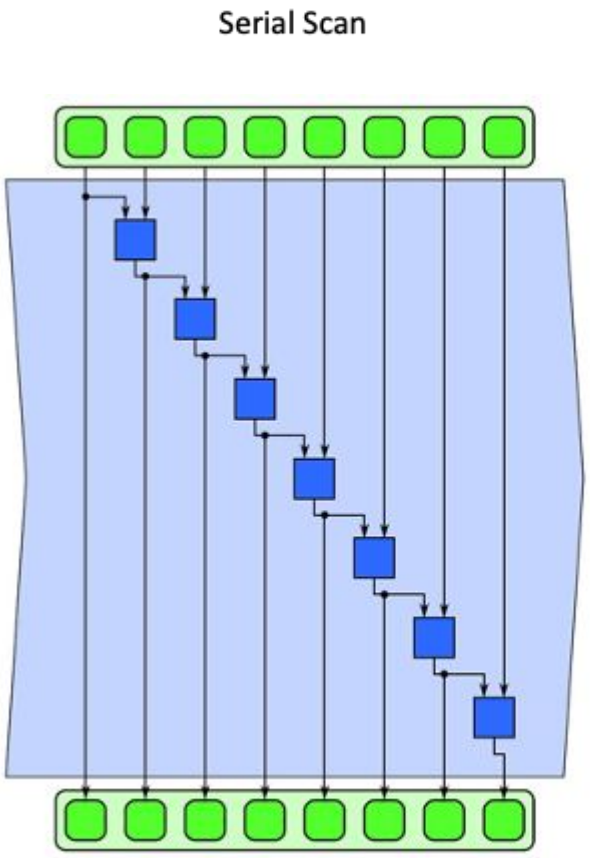
Is prefix-sum inclusive or exclusive?

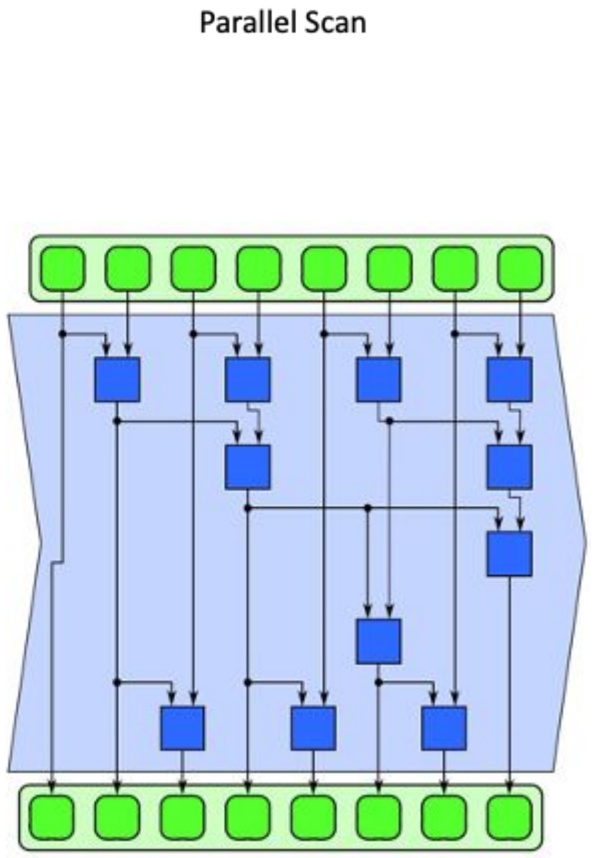
Scan

One (parallel) algorithm for **up sweep** and one for **down sweep**

- Down sweep - compute intermediate results
- Up sweep - compute reduction

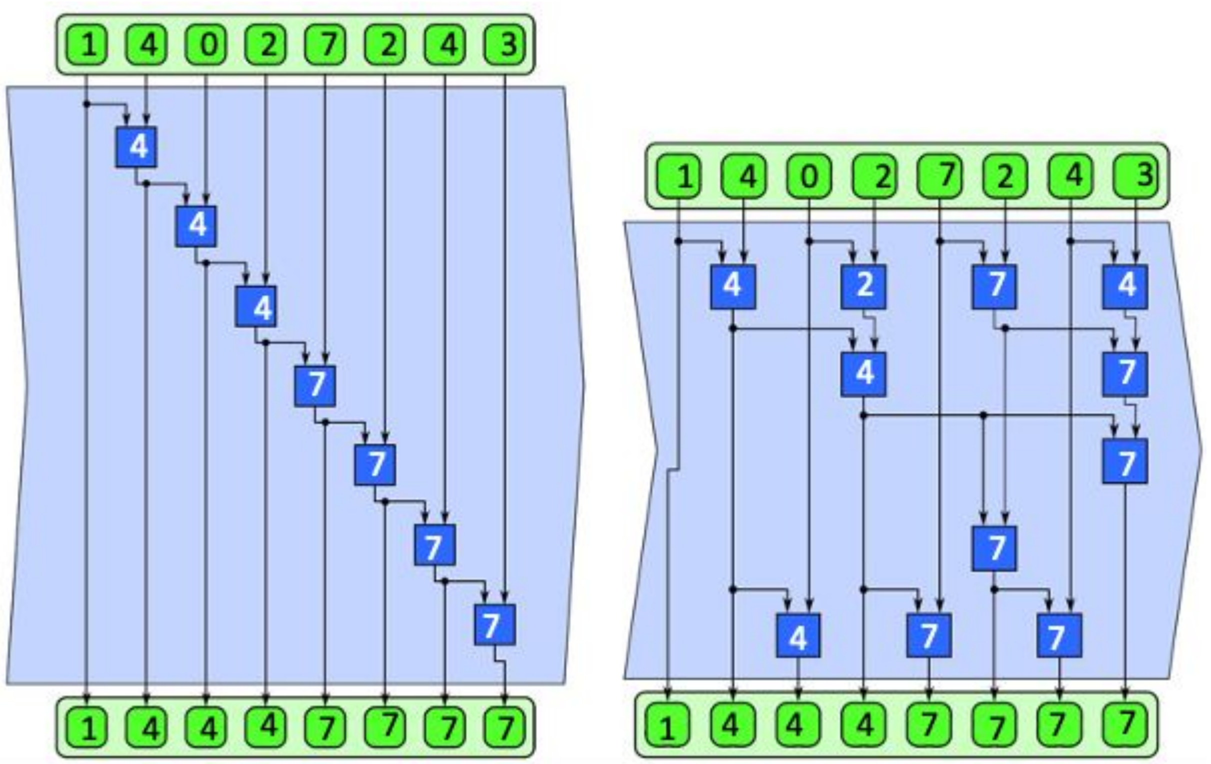
Scan (prefix-sum)





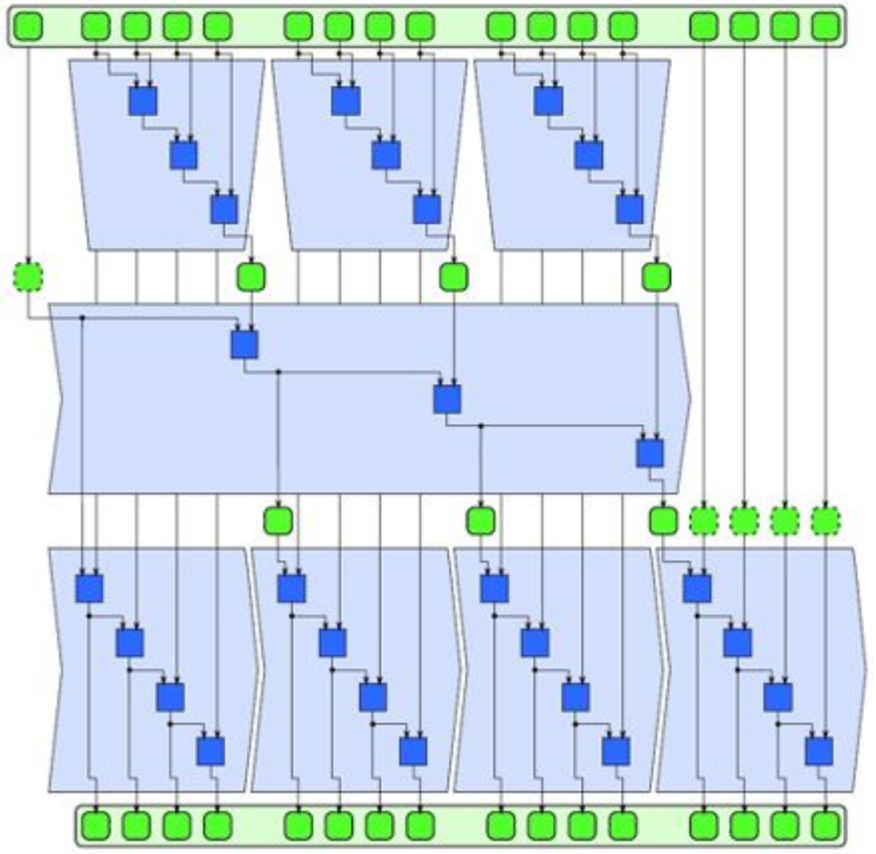
Scan

Calculating the max value



Scan

Three-phase scan with tiling



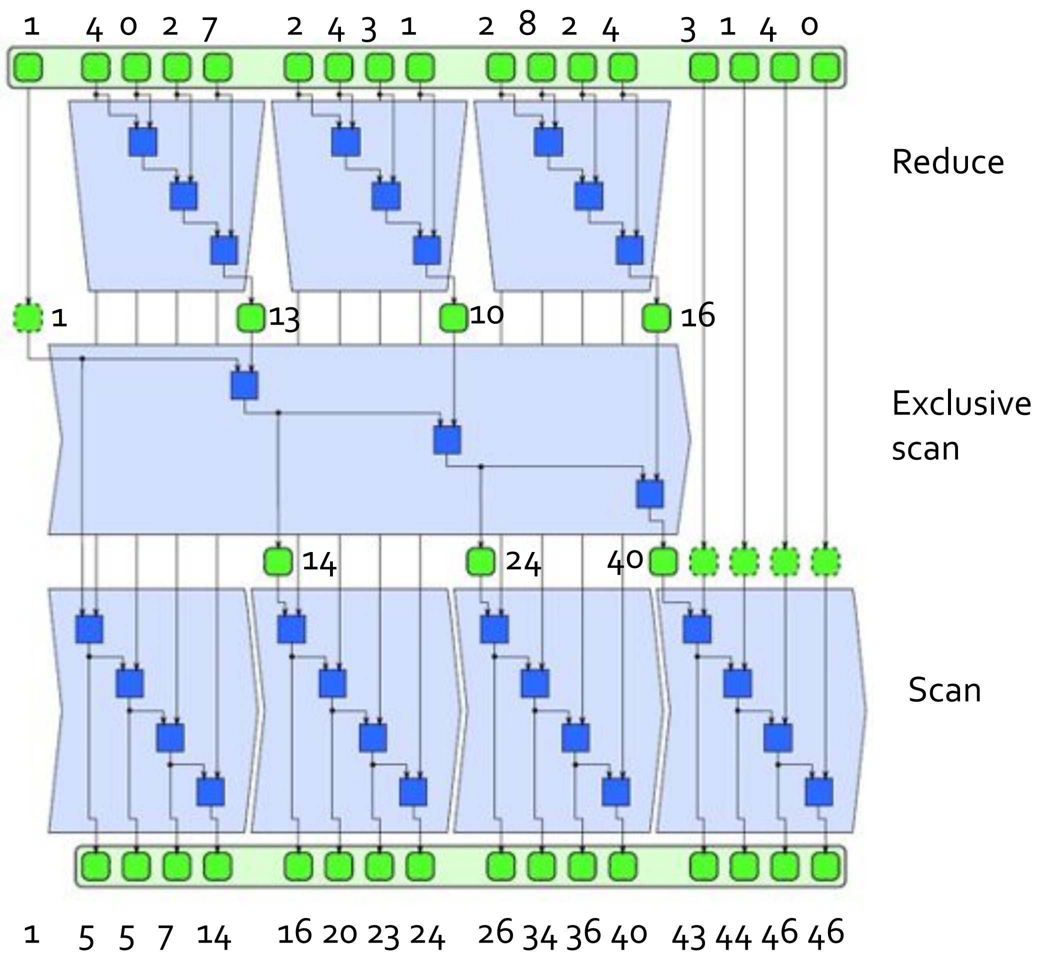
Reduce

Exclusive scan

Scan

Scan

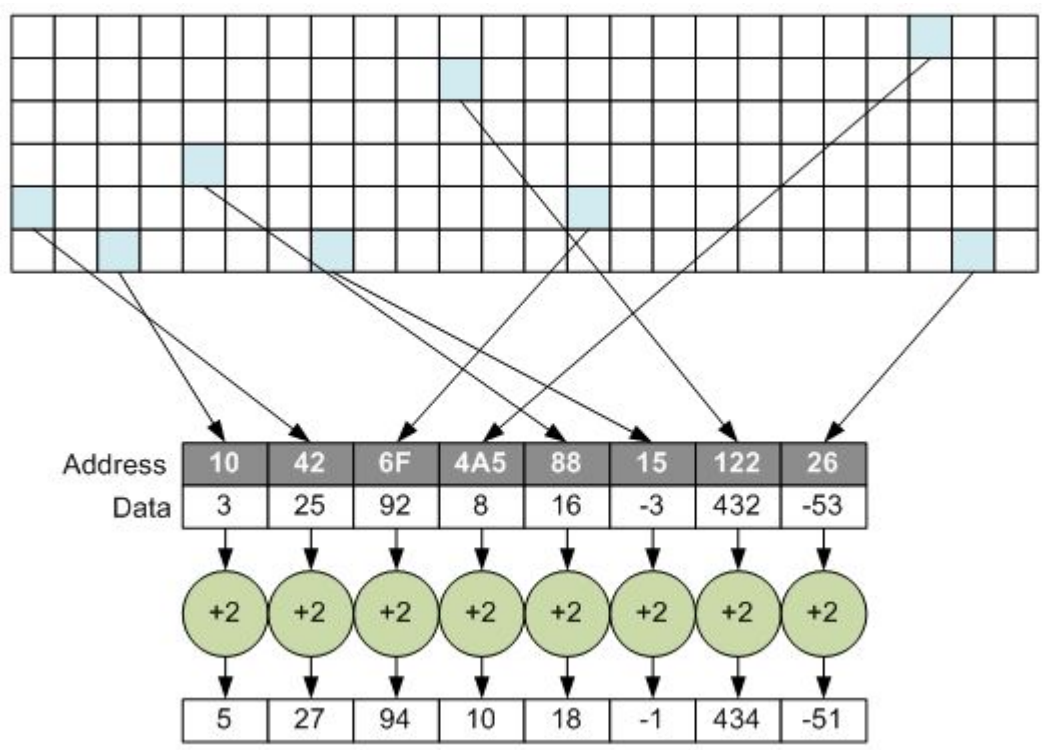
Three-phase scan with tiling



Scatter & Gather

Gather - **collecting** a bunch of randomly located data and putting them together in a **packed** form

Requires a sequence of random reads (but consecutive writes)



Scatter & Gather

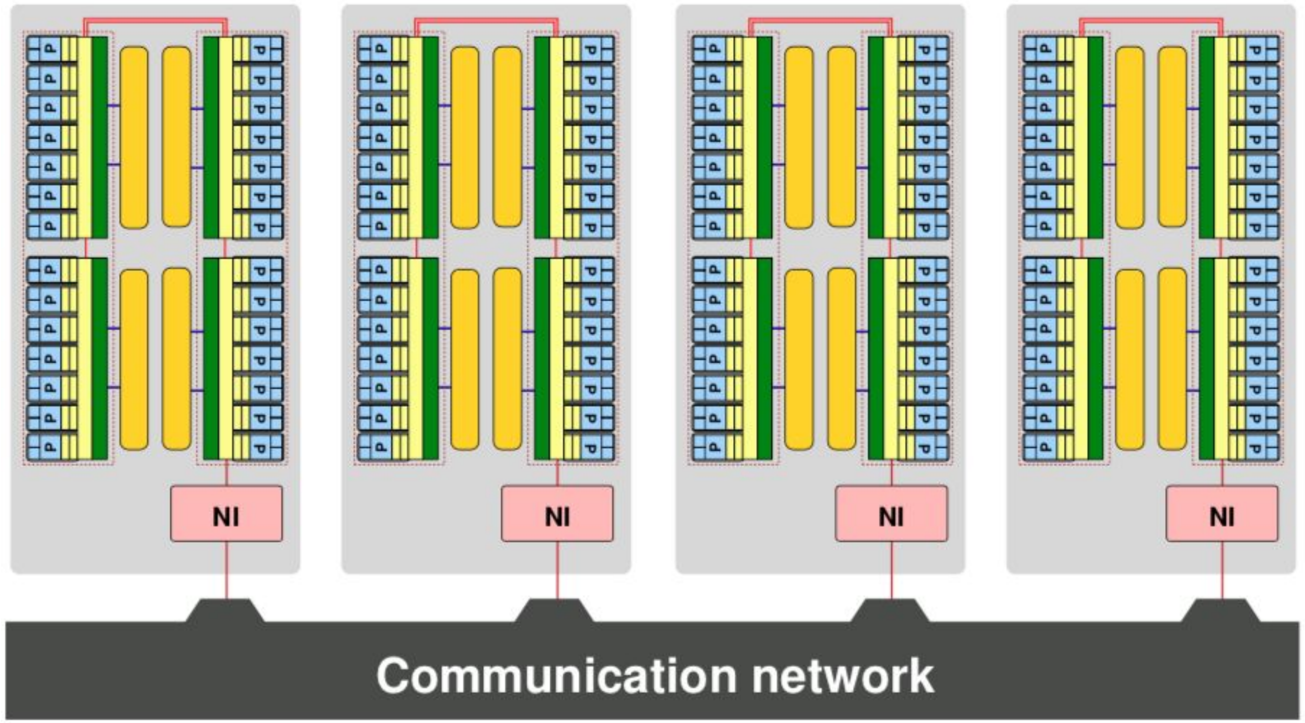
Scatter - inverse of gather, **write** to random locations from consecutive/packed address locations

Requires a sequence of random writes (but consecutive reads)

Will these operations be efficient on modern memory systems? Why or why not?

Questions?

Distributed-memory systems



How do the nodes communicate?

MPI

Specification for the message passing library

- Different vendors have different implementations
  - Different MPI library implementations support different MPI versions/features
- Objectives
  - Practical
  - Portable
  - Efficient
  - Flexible

Latest MPI version - 4.1 (approved Nov. 2, 2023)

MPI Libraries

MPI Library	System	Compilers
MVAPICH	Linux clusters	GNU, Intel, PGI, Clang
OpenMPI	Linux clusters	GNU, Intel, PGI, Clang
Intel MPI	Linux clusters	Intel, GNU
IBM BG/Q MPI	BG/Q Clusters	IBM, GNU
IBM Spectrum MPI	Coral and Summit	IBM, GNU, PGI, Clang

Module

```
[jeec@talapas-ln1 ~]$ module list

Currently Loaded Modules:
  1) slurm/19.05   2) intel/17   3) openmpi/2.1   4) mkl   5) cuda/9.2
```

Compiling MPI Code

```
mpic++ -c -g -W -Wall -std=c++14 -fopenmp -DDEBUG=1 ping_pong.cc -o ping_pong.o
mpic++ -g -W -Wall -std=c++14 -fopenmp -DDEBUG=1 -o pp ping_pong.o
```

Executing MPI Code

```
mpi_test.batch
#!/bin/bash
#SBATCH --account=cis431_531                         ### your charge account
#SBATCH --partition=compute                          ### queue to submit to
#SBATCH --job-name=mpi_test                     ### job name
#SBATCH --output=output/mpi_test_%A.out ### file in which to store job stdout
#SBATCH --error=output/mpi_test_%A.err  ### file in which to store job stderr
#SBATCH --time=5                                ### wall-clock time limit, in minutes
#SBATCH --mem=64000M                            ### memory limit per node, in MB
#SBATCH --nodes=2                               ### number of nodes to use
#SBATCH --ntasks-per-node=28                    ### number of tasks to launch per node
#SBATCH --cpus-per-task=1                       ### number of cores for each task

mpirun -np $SLURM_NTASKS ./merge 100000000 1000
```

Simple Linux Utility for Resource Management (**SLURM**)

open source cluster management and job scheduling system

MPI Task Placement

Two popular strategies for parallelizing code using MPI

Exclusively MPI

Parallelize the code at the MPI task level such that each MPI task is assigned to one core

For example, if you are running an application on 3 nodes, each with 28 cores, you can create 3x28 = 84 MPI tasks, each task running on a single core

```
#SBATCH --nodes=3                 ### number of nodes to use
#SBATCH --ntasks-per-node=28      ### number of tasks to launch per node
#SBATCH --cpus-per-task=1         ### number of cores for each task
```

MPI+OpenMP **Hybrid** Method

Create 1 MPI task per node

Within each node, use OpenMP to parallelize the code

```
#SBATCH --nodes=3                 ### number of nodes to use
#SBATCH --ntasks-per-node=1       ### number of tasks to launch per node
#SBATCH --cpus-per-task=28        ### number of cores for each task
```

Basics

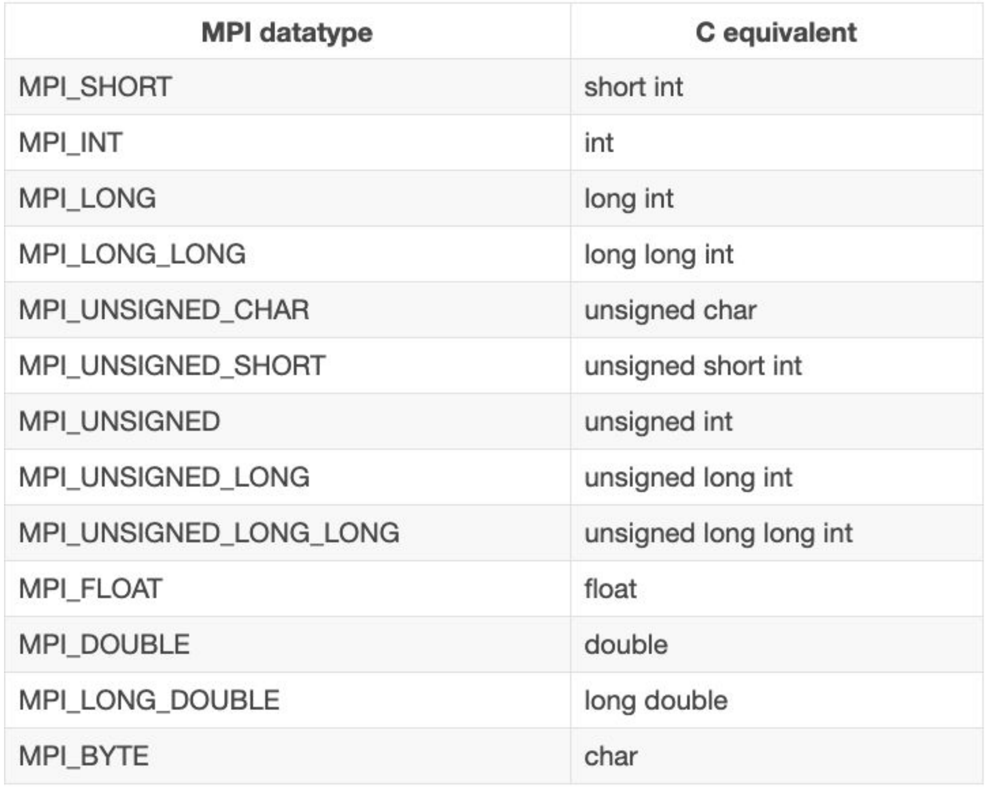
```
int MPI_Send(const void *buf,
             int count,
             MPI_Datatype datatype,
             int dest,
             int tag,
             MPI_Comm comm)

int MPI_Recv(void *buf,
             int count,
             MPI_Datatype datatype,
             int source,
             int tag,
             MPI_Comm comm,
             MPI_Status * status)
```

These are blocking communication primitives - functions blocks until function is successfully completed

Basics

Elementary MPI Datatypes



MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

Simple Point-to-Point Communication

```
// Initialize MPI
MPI_Init(NULL, NULL);

// Current rank's ID
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
// Total number of ranks
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

… Some code …

// Finish MPI
MPI_Finalize();
```

Simple Point-to-Point Communication

```
cout << "My rank is " << world_rank << endl;
cout << "My world is " << world_size << endl;

int number;
// If my rank is 0, send data
if(world_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
// If my rank is 1, receive data
} else if(world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    cout << "Process 1 received number " << number <<
            " from process 0" << endl;
}
```

Remember...

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int
tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
MPI_Comm comm, MPI_Status * status)
```

Simple Point-to-Point Communication

```
My rank is 0
My world is 2
My rank is 1
My world is 2
Process 1 received number -1 from process 0
```

Simple Point-to-Point Communication

```
// Assume 2 ranks
    int pp_count = 0;
    int partner_rank = (world_rank + 1) % 2;
    while(pp_count < PP_MAX) {
        if(world_rank == (pp_count % 2)) {
            pp_count++;
            MPI_Send(&pp_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
            cout << world_rank << " sent and incremented pp_count " << pp_count
                 << " to " << partner_rank << endl;
        } else {
            MPI_Recv(&pp_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
                     MPI_STATUS_IGNORE);
            cout << world_rank << " received pp_count " << pp_count << " from "
                 << partner_rank << endl;
        }
        cout << "My rank is " << world_rank << " and I have " << pp_count
             << endl;
    }
```

Simple Point-to-Point Communication

```
0 sent and incremented pp_count 1 to 1
My rank is 0 and I have 1
0 received pp_count 2 from 1
My rank is 0 and I have 2
0 sent and incremented pp_count 3 to 1
My rank is 0 and I have 3
...
0 sent and incremented pp_count 9 to 1
My rank is 0 and I have 9
0 received pp_count 10 from 1
My rank is 0 and I have 10

1 received pp_count 1 from 0
My rank is 1 and I have 1
1 sent and incremented pp_count 2 to 0
My rank is 1 and I have 2
1 received pp_count 3 from 0
My rank is 1 and I have 3
...
1 received pp_count 9 from 0
My rank is 1 and I have 9
1 sent and incremented pp_count 10 to 0
My rank is 1 and I have 10
```

Simple Point-to-Point Communication

What is happening here?

```
int token;
if(world_rank != 0) {
    MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
} else {
    token = -1;
}

MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
         MPI_COMM_WORLD);

if(world_rank == 0) {
    MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    cout << "Process " << world_rank << " received token " << token
         << " from process " << world_size - 1 << endl;
}
```

Simple Point-to-Point Communication

With 4 nodes:

```
Process 0 received token -1 from process 3
```

Dynamic Receive

MPI Status

- You can ignore it with MPI_STATUS_IGNORE
- Pass in a structure which will be populated with information about the receive operation after completion
  - Rank of the sender
  - Tag of the message
  - Length of the message (using MPI_Get_count function)
- Why?
  - MPI_Recv can actually take MPI_ANY_SOURCE and MPI_ANY_TAG for those parameters
  - In these cases, MPI status is the only way to figure out the rank and tag

Dynamic Receive

```
MPI_Status stat;
if(rank){
    int someval = 0;
    MPI_Sendrecv(&someval, 1, MPI_INT, 0, 1, &recvbuf, 1, MPI_INT, 0, MPI_ANY_TAG,
                 MPI_COMM_WORLD, &stat);
} else {
    int someotherval = 1;
    MPI_Recv(&recvbuf, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
    MPI_Send(&someotherval, 1, MPI_INT, stat.MPI_SOURCE, someotherval, MPI_COMM_WORLD);
}
```

Any problems?

Dynamic Receive

Sending arbitrary amount of data

```
const int MAX_NUMBER = 100;
int numbers[MAX_NUMBER];
int number_amount;
if (world_rank == 0) {
    // Pick a random amount of integers to send to process one
    srand(time(NULL));
    number_amount = (rand() / (float)RAND_MAX) * MAX_NUMBER;

    // Send the amount of integers to process one
    MPI_Send(numbers, number_amount, MPI_INT, 1, 0, MPI_COMM_WORLD);
    cout << "0 sent " << number_amount << " numbers to 1" << endl;
...
```

Dynamic Receive

```
} else if (world_rank == 1) {
    MPI_Status status;

    // First, use probe to get status
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);

    // After receiving the status, check to determine
    // how many numbers were actually received
    MPI_Get_count(&status, MPI_INT, &number_amount);

    // Allocate a buffer to hold the incoming numbers
    int* number_buf = (int*) malloc(sizeof(int) * number_amount);

    // Receive at most MAX_NUMBER from process zero
    MPI_Recv(number_buf, number_amount, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);


    // Print off the amount of numbers, and also print additional
    // information in the status object
    cout << "1 dynamically received " << number_amount
         << " numbers from 0. Message"
         << " source = " << status.MPI_SOURCE
         << ", tag = " << status.MPI_TAG << endl;

    free(number_buf);
}
```

Dynamic Receive

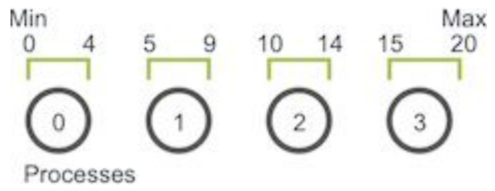
```
0 sent 72 numbers to 1
1 dynamically received 72 numbers from 0. Message source = 0, tag = 0
```

Example - Random Walk

Given a min and max, a random walker W takes S random walks of arbitrary lengths to the right on a line

If W goes out of bounds, W wraps around

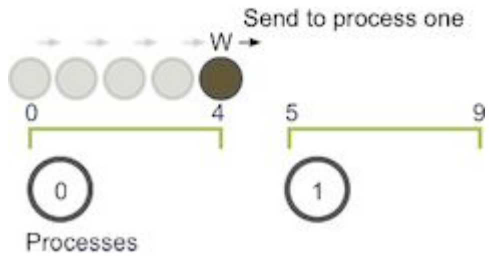
Example - Random Walk



Assuming min of 0 and max of 20 with 4 processes

W takes a single walk of size 6 to the right, starting from position 0

At step 4, it goes out of bounds for process 0



Process 0 must now communicate this to process 1, and process 1 continues the walk

Example - Random Walk

Each process is in charge of their part of the domain (domain decomposition)

Each process initializes N walkers, each starting from the beginning of their local domain

Each process has two values - current position of the walker, and number of steps to take

Walkers are traversing through the domain and are passed to other processes until they complete their walk

Processes end when ALL walkers have finished

Example - Random Walk

```
 domain_size = atoi(argv[1]);
 max_walk_size = atoi(argv[2]);
 num_walkers_per_proc = atoi(argv[3]);


void decompose_domain(int domain_size, int world_rank,
                      int world_size, int* subdomain_start,
                      int* subdomain_size) {
    if (world_size > domain_size) {
        // Don't worry about this special case. Assume the domain
        // size is greater than the world size.
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
    *subdomain_start = domain_size / world_size * world_rank;
    *subdomain_size = domain_size / world_size;
    if (world_rank == world_size - 1) {
        // Give remainder to last process
        *subdomain_size += domain_size % world_size;
    }
}
```

Example - Random Walk

```
typedef struct {
    int location;
    int num_steps_left_in_walk;
} Walker;

void initialize_walkers(int num_walkers_per_proc, int max_walk_size,
                        int subdomain_start, int subdomain_size,
                        vector<Walker>* incoming_walkers) {
    Walker walker;
    for (int i = 0; i < num_walkers_per_proc; i++) {
        // Initialize walkers in the middle of the subdomain
        walker.location = subdomain_start;
        walker.num_steps_left_in_walk =
            (rand() / (float)RAND_MAX) * max_walk_size;
        incoming_walkers->push_back(walker);
    }
}
```

Example - Random Walk

```
void walk(Walker* walker, int subdomain_start, int subdomain_size,
          int domain_size, vector<Walker>* outgoing_walkers) {
    while (walker->num_steps_left_in_walk > 0) {
        if (walker->location == subdomain_start + subdomain_size) {
            // Take care of the case when the walker is at the end
            // of the domain by wrapping it around to the beginning
            if (walker->location == domain_size) {
                walker->location = 0;
            }
            outgoing_walkers->push_back(*walker);
            break;
        } else {
            walker->num_steps_left_in_walk--;
            walker->location++;
        }
    }
}
```

Example - Random Walk

```
void send_outgoing_walkers(vector<Walker>* outgoing_walkers,
                           int world_rank, int world_size) {
    // Send the data as an array of MPI_BYTEs to the next process.
    // The last process sends to process zero.
    MPI_Send((void*)outgoing_walkers->data(),
             outgoing_walkers->size() * sizeof(Walker), MPI_BYTE,
             (world_rank + 1) % world_size, 0, MPI_COMM_WORLD);

    // Clear the outgoing walkers
    outgoing_walkers->clear();
}
```

Example - Random Walk

```
void receive_incoming_walkers(vector<Walker>* incoming_walkers,
                              int world_rank, int world_size) {
    MPI_Status status;

    // Receive from the process before you.
    int incoming_rank =
        (world_rank == 0) ? world_size - 1 : world_rank - 1;
    MPI_Probe(incoming_rank, 0, MPI_COMM_WORLD, &status);
    // Resize your incoming walker buffer
    int incoming_walkers_size;
    MPI_Get_count(&status, MPI_BYTE, &incoming_walkers_size);
    incoming_walkers->resize(
        incoming_walkers_size / sizeof(Walker));
    MPI_Recv((void*)incoming_walkers->data(), incoming_walkers_size,
             MPI_BYTE, incoming_rank, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
}
```

Example - Random Walk

1. Initialize the walkers.
2. Progress the walkers with the `walk` function.
3. Send out any walkers in the `outgoing_walkers` vector.
4. Receive new walkers and put them in the `incoming_walkers` vector.
5. Repeat steps two through four until all walkers have finished.

Example - Random Walk

```
decompose_domain(domain_size, world_rank, world_size,
                 &subdomain_start, &subdomain_size);
initialize_walkers(num_walkers_per_proc, max_walk_size,
                   subdomain_start, subdomain_size,
                   &incoming_walkers);

while (!all_walkers_finished) { // Determine walker completion later
    // Process all incoming walkers
    for (int i = 0; i < incoming_walkers.size(); i++) {
        walk(&incoming_walkers[i], subdomain_start, subdomain_size,
             domain_size, &outgoing_walkers);
    }
    // Send all outgoing walkers to the next process.
    send_outgoing_walkers(&outgoing_walkers, world_rank,
                          world_size);
    // Receive all the new incoming walkers
    receive_incoming_walkers(&incoming_walkers, world_rank,
                             world_size);
}
```

Example - Random Walk

Problems?

Example - Random Walk

Problems?

    Will they deadlock?

Example - Random Walk

Problems?

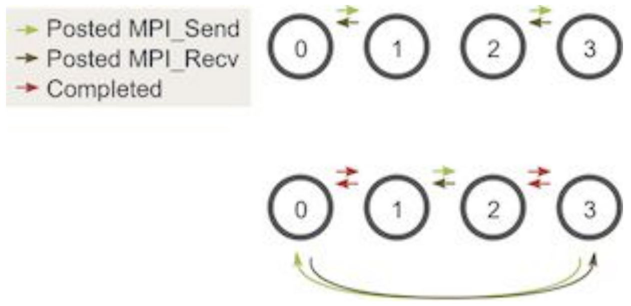
- Will they deadlock?
  - Perhaps - MPI Send blocks (until the send buffer can be used for other purposes)
  - This generally means that the data in the buffer has been placed in some queue/buffer in the network to be sent out
  - Or it could mean MPI has saved it elsewhere (this is implementation dependent)

This code will LIKELY work

However, it may deadlock on some systems, so it is better to make sure there are no way for deadlock to occur - how?

Example - Random Walk

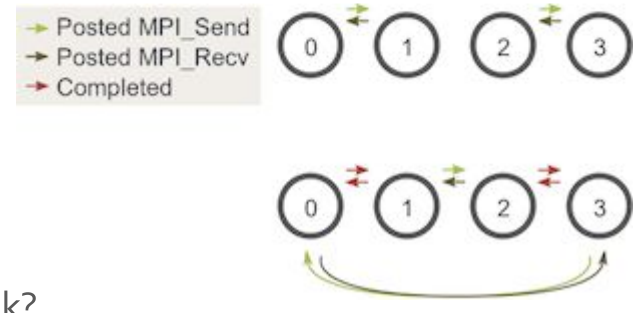
Even-numbered processes send, and odd numbered processes receive



Can it still deadlock?

Example - Random Walk

Even-numbered processes send, and odd numbered processes receive



Can it still deadlock?

Yes - if there is only 1 process

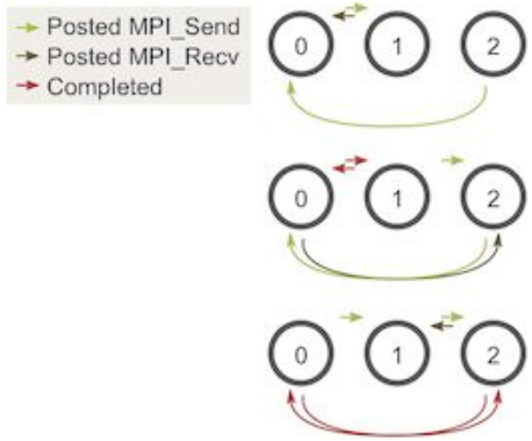
do not use send/recv when there is only one process

Example - Random Walk

What about if you have odd number of processes - will it still work?

Example - Random Walk

What about if you have odd number of processes - will it still work?



Example - Random Walk

How do we determine if the program has finished?

- Have process 0 keep track of unfinished walkers, and tell others to stop
- Requires additional communication

Example - Random Walk

How do we determine if the program has finished?

```
int maximum_sends_recvs = max_walk_size / (domain_size / world_size) + 1;
for (int m = 0; m < maximum_sends_recvs; m++) {
    // Process all incoming walkers
    for (int i = 0; i < incoming_walkers.size(); i++) {
        walk(&incoming_walkers[i], subdomain_start, subdomain_size,
             domain_size, &outgoing_walkers);
    }
    // Send and receive if you are even and vice versa for odd
    if (world_rank % 2 == 0) {
        send_outgoing_walkers(&outgoing_walkers, world_rank, world_size);
        receive_incoming_walkers(&incoming_walkers, world_rank, world_size);
    } else {
        receive_incoming_walkers(&incoming_walkers, world_rank, world_size);
        send_outgoing_walkers(&outgoing_walkers, world_rank, world_size);
    }
}
```

Example - Random Walk

Where is this used?

- Particle tracing
  - Used to visualize flow fields
  - Particles are inserted into the flow field and then traced along the flow using numerical integration
  - The traced particle can then be rendered for visualization
  - Load balancing is tricky - you don’t know where particles will end up

Questions