



The Message Passing Interface (MPI): Parallelism on Distributed CPUs

<http://mpi-forum.org>
<https://www.open-mpi.org/>



Oregon State University
Mike Bailey
 mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

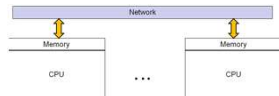




Diagram illustrating MPI architecture: Multiple nodes (CPU and Memory) connected via a Network.



mpb - March 26, 2025


Why Two URLs?



<http://mpi-forum.org>
 This is the definitive reference for the MPI standard. Go here if you want to read the official specification, which, BTW, continues to evolve.

<https://www.open-mpi.org/>
 This consortium formed later. This is the open source version of MPI. If you want to start using MPI, I recommend you look here. This is the MPI that the COE systems use

<https://www.open-mpi.org/doc/v4.0/>
 This URL is also really good – it is a link to all of the MPI man pages



mpb - March 26, 2025

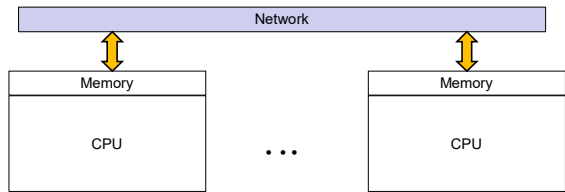
The Open MPI Consortium






mpb - March 26, 2025

MPI: The Basic Idea



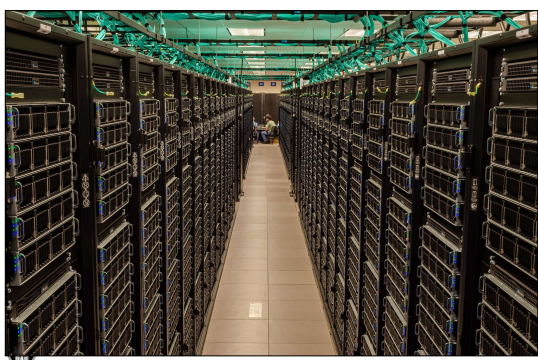
Programs on different CPUs coordinate computations by passing messages between each other

Note: Each CPU in the MPI "cluster" must be prepared ahead of time by having the MPI server code installed on it. It must then have that server code running and listening on its socket connection. Each MPI CPU must also have an integer ID assigned to it (called its *rank*).




mpb - March 26, 2025

This is how modern supercomputers work!

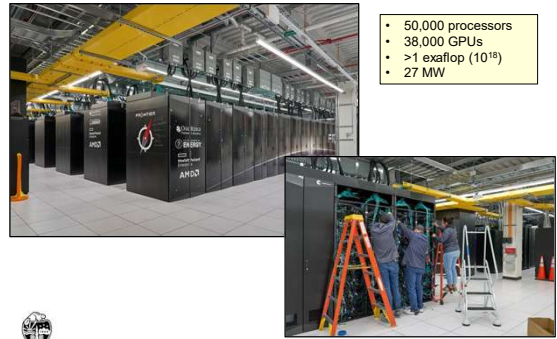


The Texas Advanced Computing Center's Frontera supercomputer




mpb - March 26, 2025

This is how modern supercomputers work!

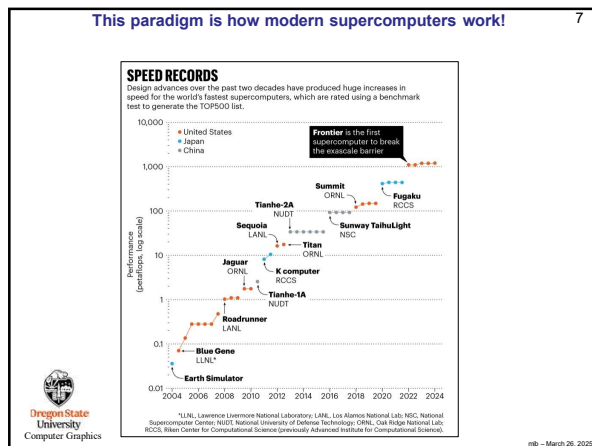


- 50,000 processors
- 38,000 GPUs
- >1 exaflop (10^{18})
- 27 MW

The Oakridge National Lab Frontier supercomputer



mpb - March 26, 2025



How to SSH to the COE MPI Cluster

ssh over to an MPI submission machine -- **submit-a** and **submit-b** will also work

```
flip3 151% ssh submit-c.hpc.engr.oregonstate.edu
```

submit-c 142% module load slurm
submit-c 143% module load openmpi

Type these right away to set your path correctly

BTW, you can find out more about the COE cluster here:
<https://it.engineering.oregonstate.edu/hpc>

The College of Engineering HPC cluster is a heterogeneous mix of over 130 servers providing 4,800 CPU cores, over 230 GPUs, and over 50 TB of total RAM. The systems are connected via gigabit ethernet and Infiniband. Most of the latest servers utilize Mellanox EDR or HDR Infiniband network connection. The cluster also has access to 500TB global scratch space. The CoE HPC Cluster is rated at over 2,200 peak TFLOPS (double-precision).

Oregon State University
Computer Graphics

mp - March 26, 2025

Compiling and Running

```
mpicc -o program program.c ...
```

or

```
mpic++ -o program program.cpp ...
```

Warning – use *mpic++* and *mpixec*.

Don't use g++.
Don't run by just typing the name of the executable!

```
mpixec -mca btl self,tcp -np 4 program
```

of CPUs (processors) to use

All distributed CPUs execute the same program at the same time

Oregon State University
Computer Graphics

mp - March 26, 2025

Running with a *bash* Batch Script

submit.bash:

```
#!/bin/bash
#SBATCH -J Fourier
#SBATCH -A cs475-575
#SBATCH -p classmpitest
#SBATCH -N 8 # number of nodes
#SBATCH -n 8 # number of tasks
#SBATCH -o mpiproject.out
#SBATCH -e mpiproject.err
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=joeparallel@oregonstate.edu
module load openmpi
mpic++ mpiproject.cpp -o mpiproject -lm
mpixec -mca btl self,tcp -np 4 ./mpiproject
```

Your Job Name

This is the partition name that we use for our class when debugging and testing your program. Use **classmpifinal** for taking your final performance numbers.

These 3 lines are bash code

```
submit-c 143% sbatch submit.bash
```

Submitted batch job 258759

Oregon State University
Computer Graphics

mp - March 26, 2025

What is the Difference Between the Partitions *classmpitest* and *classmpifinal*?

classmpitest lets your program get into the system sooner, but it might be running alongside other jobs, so its performance might suffer. But, you don't care because you are just compiling and debugging, not taking performance numbers for your report.

classmpifinal makes your program wait in line until it can get dedicated resources so that you get performance results that are much more representative of what the machines can do, and thus are worthy to be listed in your report.

Oregon State University
Computer Graphics

mp - March 26, 2025

Auto-Notifications via Email

```
#SBATCH --mail-user=joeparallel@oregonstate.edu
```

You don't have to ask the system to email information to you, but if you do, **please be sure you spell your own email address correctly!**

Our IT people are getting *really* tired of fielding the bounced emails when people misspell their own email address.


Oregon State University
Computer Graphics

mp - March 26, 2025

Use slurm's *scancel* if your Job Needs to Be Killed

```
submit-c 143% sbatch_submit.bash
Submitted batch job 258759
```

```
submit-c 144% scancel 258759
```



mpb - March 26, 2025

Setting Up and Finishing MPI

```
#include <mpi.h>

int
main( int argc, char *argv[] )
{
    ...


    MPI_Init( &argc, &argv );

    ...

    MPI_Finalize( );
    return 0;
}
```

You don't need to process command line arguments if you don't want to. You can just call `MPI_Init()` as:


```
MPI_Init( NULL, NULL );
```



mpb - March 26, 2025

MPI Follows a Single-Program-Multiple-Data (SPMD) Model

A **communicator** is a collection of CPUs that are capable of sending messages to each other



Oh, look, a communicator of Corvallis deer!

Oh, look, a communicator of Corvallis turkeys!

This requires MPI server code getting installed on all those CPUs. That code then needs to be running and listening on a socket connection. Only an administrator can do this.

Getting information about our place in the **communicator**:


```
int numCPUs; // total # of cpus involved
int me; // which one I am

MPI_Comm_size( MPI_COMM_WORLD, &numCPUs );
MPI_Comm_rank( MPI_COMM_WORLD, &me );
```

Size, i.e., how many altogether?

Rank, i.e., which one am I?

It is then each CPU's job to figure out what piece of the overall problem it is responsible for and then go do it.



mpb - March 26, 2025

A First Test of MPI

```
#include <stdio.h>
#include <math.h>
#include <mpi.h>

#define THEBOSS 0


int
main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );

    int numCPUs; // total # of cpus involved
    int me; // which one I am

    MPI_Comm_size( MPI_COMM_WORLD, &numCPUs );
    MPI_Comm_rank( MPI_COMM_WORLD, &me );

    if( me == THEBOSS )
        fprintf( stderr, "Rank %d says that we have a Communicator of size %d\n", THEBOSS, numCPUs );
    else
        fprintf( stderr, "Welcome from Rank %d\n", me );

    MPI_Finalize( );
    return 0;
}
```




mpb - March 26, 2025

```
submit-c 165% mpxexec -np 16 ./first
Welcome from Rank 13
Welcome from Rank 15
Welcome from Rank 3
Welcome from Rank 7
Welcome from Rank 5
Welcome from Rank 8
Welcome from Rank 9
Welcome from Rank 11
Rank 0 says that we have a Communicator of size 16
Welcome from Rank 1
Welcome from Rank 12
Welcome from Rank 14
Welcome from Rank 6
Welcome from Rank 2
Welcome from Rank 10
Welcome from Rank 4
```

```
submit-c 166% mpxexec -np 16 ./first
Welcome from Rank 1
Welcome from Rank 5
Welcome from Rank 7
Welcome from Rank 9
Welcome from Rank 11
Welcome from Rank 13
Welcome from Rank 15
Rank 0 says that we have a Communicator of size 16
Welcome from Rank 2
Welcome from Rank 3
Welcome from Rank 4
Welcome from Rank 6
Welcome from Rank 8
Welcome from Rank 12
Welcome from Rank 14
Welcome from Rank 10
```

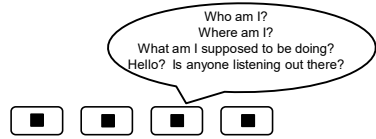
```
submit-c 167% mpxexec -np 16 ./first
Welcome from Rank 9
Welcome from Rank 11
Welcome from Rank 13
Welcome from Rank 7
Welcome from Rank 5
Welcome from Rank 8
Welcome from Rank 10
Welcome from Rank 15
Welcome from Rank 4
Welcome from Rank 6
Rank 0 says that we have a Communicator of size 16
Welcome from Rank 2
Welcome from Rank 3
Welcome from Rank 14
Welcome from Rank 12
```

```
submit-c 168% mpxexec -np 16 ./first
Welcome from Rank 13
Welcome from Rank 15
Welcome from Rank 7
Welcome from Rank 5
Welcome from Rank 9
Welcome from Rank 11
Welcome from Rank 1
Welcome from Rank 12
Welcome from Rank 14
Welcome from Rank 4
Welcome from Rank 2
Rank 0 says that we have a Communicator of size 16
Welcome from Rank 8
Welcome from Rank 10
Welcome from Rank 6
```





mpb - March 26, 2025

So, we have a group (a "communicator") of distributed CPUs. How do they communicate about what work they are supposed to do?



Example: You could coordinate the units of our DGX system using MPI

mpb - March 26, 2025

A Good Place to Start: MPI Broadcasting

MPI_Bcast(array, count, type, src, MPI_COMM_WORLD);


Address of the data to send from if you are the *src* node;
Address of the data to receive into if you are not

elements
MPI_CHAR
MPI_INT
MPI_LONG
MPI_FLOAT
MPI_DOUBLE
...

rank of the CPU doing the sending
MPI allows you to subset the CPUs you want to get involved in this operation. MPI_COMM_WORLD says "use them all, don't subset"

Broadcast
src node
src nodes

Both the sender and receivers need to execute **MPI_Bcast** – there is no separate receive function

 mp – March 26, 2025

MPI Broadcast Example

This is our heat transfer equation from before. Clearly, every CPU will need to know this value.

$$\Delta T = \frac{k}{(\rho C)} \left(\frac{T_{i-1} - 2T_i + T_{i+1}}{(\Delta x)^2} \right) \Delta t$$

```
int numCPUs;
int me;
float k_over_rho_c; // the THEBOSS node will know this value, the others won't (yet)


#define THEBOSS 0

MPI_Comm_size( MPI_COMM_WORLD, &numCPUs );
MPI_Comm_rank( MPI_COMM_WORLD, &me ); // how many are in this communicator // which one am I?


if( me == THEBOSS )
{
    << read k_over_rho_c from the data file >>
    MPI_Bcast( &k_over_rho_c, 1, MPI_FLOAT, THEBOSS, MPI_COMM_WORLD ); // send if I am THEBOSS, and receive if not
}
```


I am the THEBOSS: this identifies this call as a send

Broadcast
src node
src nodes

 mp – March 26, 2025

How Does this Work? Think Star Trek Wormholes!



 mp – March 26, 2025

Sending Data from One Source CPU to Just One Destination CPU

MPI_Send(array, numToSend, type, dst, tag, MPI_COMM_WORLD);

address of data to send from
elements (note: this is the number of elements, not the number of bytes!)

MPI_CHAR
MPI_INT
MPI_LONG
MPI_FLOAT
MPI_DOUBLE
...


rank of the CPU to send to
An integer or character to differentiate this transmission from any other transmission. I like to use chars.

MPI allows you to subset the CPUs you want to get involved in this operation. MPI_COMM_WORLD says "use them all, don't subset"

Rules:

- One message from a specific *src* to a specific *dst* cannot overtake a previous message from the same *src* to the same *dst*.
- MPI_Send() blocks until the transfer is far enough along that *array* can be destroyed or re-used.
- There are no guarantees on order from different *src*'s.

src node → dst node

 mp – March 26, 2025

Receiving Data in a Destination CPU from a Source CPU

MPI_Recv(array, maxCanReceive, type, src, tag, MPI_COMM_WORLD, &status);

address of data to receive into
elements we can receive, at most

MPI_CHAR
MPI_INT
MPI_LONG
MPI_FLOAT
MPI_DOUBLE
...


Rank of the CPU we are expecting to get a transmission from
An integer or character to differentiate what transmission we are looking for with this call (be sure this matches what the sender is sending!). I like to use chars.

Type = MPI_Status

Rules:

- The receiver blocks waiting for data that matches what it declares to be looking for
- One message from a specific *src* to a specific *dst* cannot overtake a previous message from the same *src* to the same *dst*
- There are no guarantees on the order from different *src*'s
- The order from different *src*'s could be implied in the *tag*
- status* is type MPI_Status – the "&status" can be replaced with MPI_STATUS_IGNORE

src node → dst node

 mp – March 26, 2025

Example

Remember, this identical code runs on all CPUs:

```
int numCPUs;
int me;
#define MYDATA_SIZE 128
char mydata[ MYDATA_SIZE ];
#define THEBOSS 0

MPI_Comm_size( MPI_COMM_WORLD, &numCPUs );
MPI_Comm_rank( MPI_COMM_WORLD, &me );


if( me == THEBOSS ) // the primary
{
    for( int dst = 0; dst < numCPUs; dst++ )
    {
        if( dst != THEBOSS )
        {
            char InputData = "Hello, Beavers!";
            MPI_Send( InputData, strlen(InputData)+1, MPI_CHAR, dst, 'B', MPI_COMM_WORLD );
        }
    }
}
else // a secondary
{
    MPI_Recv( myData, MYDATA_SIZE, MPI_CHAR, THEBOSS, 'B', MPI_COMM_WORLD, MPI_STATUS_IGNORE );
    printf( " %s from rank # %d\n", in, me );
}
```

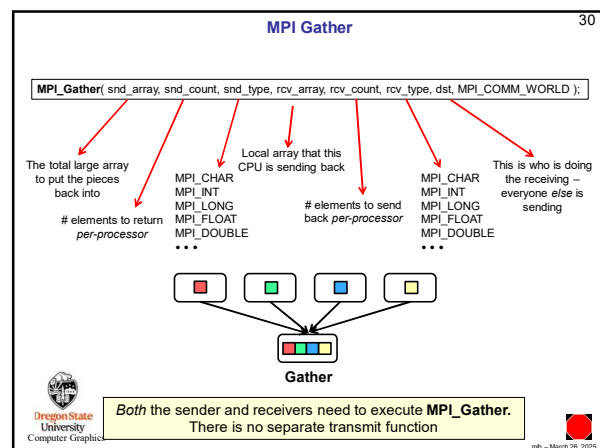
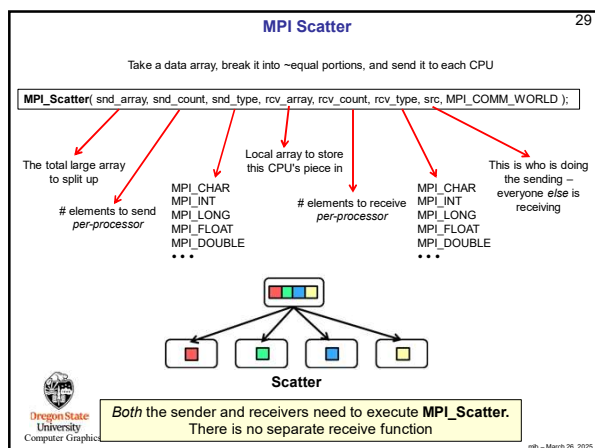
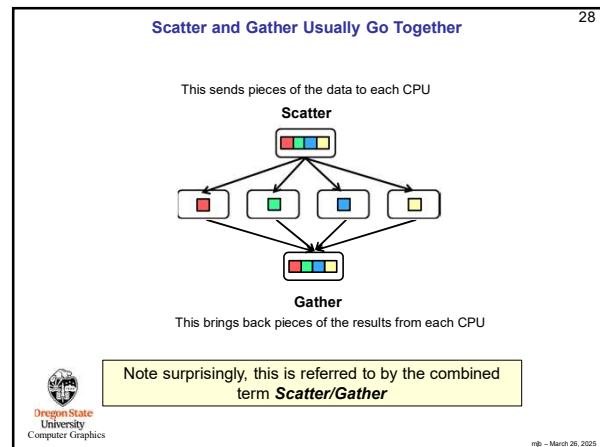
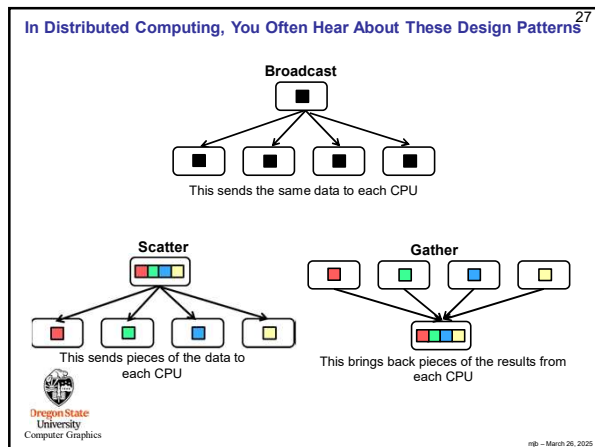
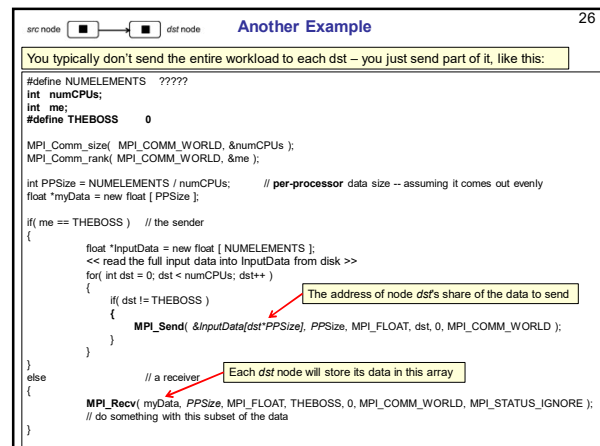
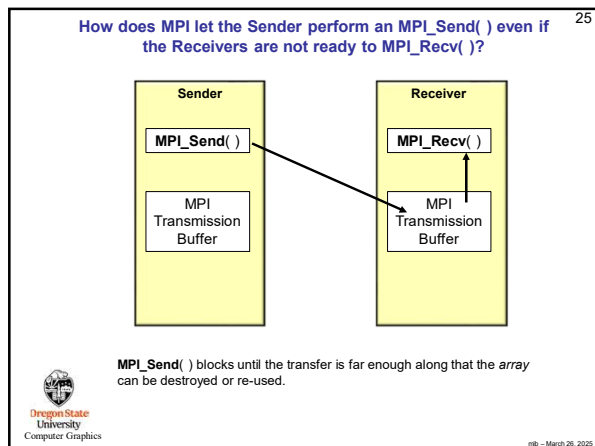
Be sure the receiving tag matches the sending tag

The tag to expect

The tag to label this transmission with

You are highly discouraged from sending to yourself. Because both the send and receive are capable of blocking, the result could be deadlock.

 mp – March 26, 2025



Remember This? It's Baaaaaack as a complete Scatter/Gather Example

CPU #0 CPU #1 CPU #2 CPU #3

The **Compute : Communicate Ratio** still applies, except that it is even more important now because there is much more overhead in the Communicate portion.

This pattern of breaking a big problem up into pieces, sending them to different CPUs, computing on the pieces, and getting the results back is very common. That's why MPI has its own scatter and gather functions.

31

mp - March 26, 2025

heat.cpp, I

```
#include <stdio.h>
#include <math.h>
#include <mpi.h>

const float RHO = 8050.;
const float C = 0.466;
const float K = 20.;
float k_over_rho_c = K / (RHO*C); // units of m^2/sec NOTE: this cannot be a const!
// K / (RHO*C) = 5.33x10^-6 m^2/sec

const float DX = 1.0;
const float DT = 1.0;

#define THEBOSS 0

#define NUMELEMENTS (8*1024*1024)
#define NUM_TIME_STEPS 4
#define DEBUG false

float * NextTemps; // per-processor array to hold computer next-values
int NumCpus; // total # of cpus involved
int PPSize; // per-processor local array size
float * PPTemps; // per-processor local array temperature data
float * TempData; // the overall NUMELEMENTS-big temperature data

void DoOneTimeStep( int );
```

32

mp - March 26, 2025

heat.cpp, II

```
int
main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );

    int me; // which one I am

    MPI_Comm_size( MPI_COMM_WORLD, &NumCpus );
    MPI_Comm_rank( MPI_COMM_WORLD, &me );

    // decide how much data to send to each CPU:
    PPSize = NUMELEMENTS / NumCpus; // assuming it comes out evenly
    PPTemps = new float [PPSize]; // all CPUs now have this uninitialized Local array
    NextTemps = new float [PPSize]; // all CPUs now have this uninitialized local array too

    // broadcast the constant:
    MPI_Bcast( (void *)&k_over_rho_c, 1, MPI_FLOAT, THEBOSS, MPI_COMM_WORLD );
```

Broadcast

33

mp - March 26, 2025

heat.cpp, III

```
if( me == THEBOSS ) // this is the data-creator
{
    TempData = new float [NUMELEMENTS];
    for( int i = 0; i < NUMELEMENTS; i++ )
        TempData[i] = 0.;
    TempData[NUMELEMENTS/2] = 100.;

    MPI_Scatter( TempData, PPSize, MPI_FLOAT, PPTemps, PPSize, MPI_FLOAT,
        THEBOSS, MPI_COMM_WORLD );
```

34

mp - March 26, 2025

heat.cpp, IV

```
// all the PPTemps arrays have now been filled
// do the time steps:

double time0 = MPI_Wtime();

for( int steps = 0; steps < NUM_TIME_STEPS; steps++ )
{
    // do the computation for one time step:
    DoOneTimeStep( me );

    // ask for all the data:
    #ifdef WANT_EACH_TIME_STEPS_DATA_BACK
        MPI_Gather( PPTemps, PPSize, MPI_FLOAT, TempData, PPSize, MPI_FLOAT,
            THEBOSS, MPI_COMM_WORLD );
    #endif
}

#ifdef WANT_EACH_TIME_STEPS_DATA_BACK
    MPI_Gather( PPTemps, PPSize, MPI_FLOAT, TempData, PPSize, MPI_FLOAT,
        THEBOSS, MPI_COMM_WORLD );
#endif

double time1 = MPI_Wtime();
```

35

mp - March 26, 2025

heat.cpp, V

```
if( me == THEBOSS )
{
    double seconds = time1 - time0;
    double performance =
        (double)NUM_TIME_STEPS * (double)NUMELEMENTS / seconds / 1000000.;
    // mega-elements computed per second
    printf( stderr, "%3d, %10d, %8.2f\n", NumCpus, NUMELEMENTS, performance );
}

MPI_Finalize();
return 0;
```

36

mp - March 26, 2025

DoOneTimeStep, I

37

```
// read from PerProcessorData[ ], write into NextTemps[ ]
void
DoOneTimeStep( int me )
{
    MPI_Status status;

    // send out the left and right end values:
    // (the tag is from the point of view of the sender)
    if( me != 0 ) // i.e., if I'm not the first group on the left
    {
        // send my PPTemps[0] to me-1 using tag 'L'
        MPI_Send( &PPTemps[0], 1, MPI_FLOAT, me-1, 'L', MPI_COMM_WORLD );
        if( DEBUG ) fprintf( stderr, "%3d sent 'L' to %3d\n", me, me-1 );
    }

    if( me != NumCpus-1 ) // i.e., not the last group on the right
    {
        // send my PPTemps[PPSize-1] to me+1 using tag 'R'
        MPI_Send( &PPTemps[PPSize-1], 1, MPI_FLOAT, me+1, 'R', MPI_COMM_WORLD );
        if( DEBUG ) fprintf( stderr, "%3d sent 'R' to %3d\n", me, me+1 );
    }
}
```

Oregon State University
Computer Graphics

mp - March 26, 2025

DoOneTimeStep, II

38

```
float left = 0.;
float right = 0.;

if( me != 0 ) // i.e., if I'm not the first group on the left
{
    // receive my "left" from me-1 using tag 'R'
    MPI_Recv( &left, 1, MPI_FLOAT, me-1, 'R', MPI_COMM_WORLD, &status );
    if( DEBUG ) fprintf( stderr, "%3d received 'R' from %3d\n", me, me-1 );
}

if( me != NumCpus-1 ) // i.e., not the last group on the right
{
    // receive my "right" from me+1 using tag 'L'
    MPI_Recv( &right, 1, MPI_FLOAT, me+1, 'L', MPI_COMM_WORLD, &status );
    if( DEBUG ) fprintf( stderr, "%3d received 'L' from %3d\n", me, me+1 );
}
}
```

Oregon State University
Computer Graphics

mp - March 26, 2025

Sharing Values Across the Boundaries

39

```
1 sent 'L' to 0
1 sent 'R' to 2
2 sent 'L' to 1
2 sent 'R' to 3
2 received 'R' from 1
0 sent 'R' to 1
0 received 'L' from 1
1 received 'R' from 0
1 received 'L' from 2
3 sent 'L' to 2
3 received 'R' from 2
2 received 'L' from 3
```

Oregon State University
Computer Graphics

mp - March 26, 2025

DoOneTimeStep, III

40

```
// first element on the left (0):
{
    float dtemp = ( k_over_rho_c *
        ( left - 2.*PPTemps[0] + PPTemps[1] ) / ( DX*DX ) ) * DT;
    NextTemps[0] = PPTemps[0] + dtemp;
}

// all the nodes in the middle:
for( int i = 1; i < PPSize-1; i++ )
{
    float dtemp = ( k_over_rho_c *
        ( PPTemps[i-1] - 2.*PPTemps[i] + PPTemps[i+1] ) / ( DX*DX ) ) * DT;
    NextTemps[i] = PPTemps[i] + dtemp;
}

// last element on the right (PPSize-1):
{
    float dtemp = ( k_over_rho_c *
        ( PPTemps[PPSize-2] - 2.*PPTemps[PPSize-1] + right ) / ( DX*DX ) ) * DT;
    NextTemps[PPSize-1] = PPTemps[PPSize-1] + dtemp;
}
}
```

Oregon State University
Computer Graphics

mp - March 26, 2025

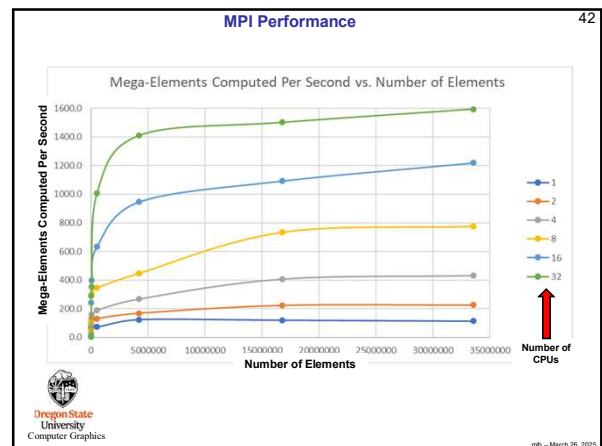
DoOneTimeStep, IV

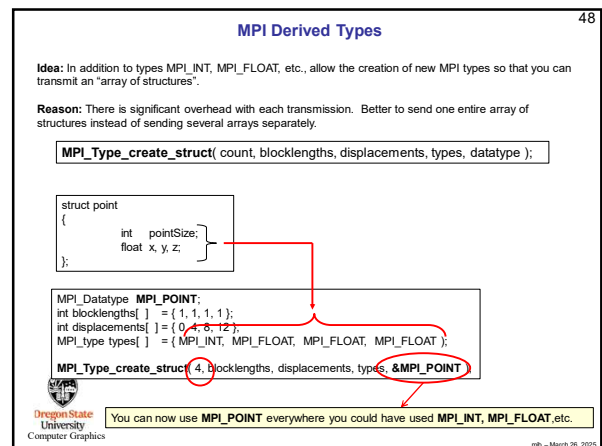
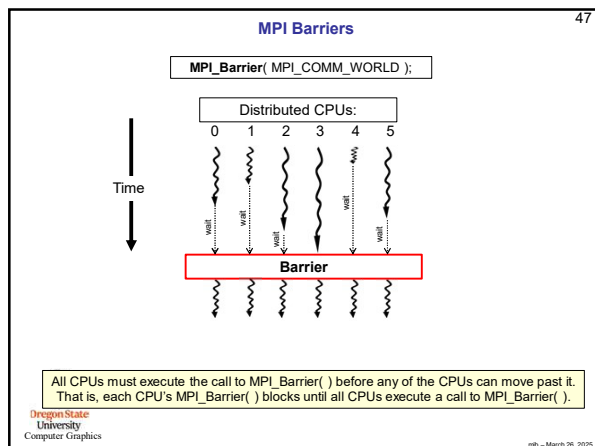
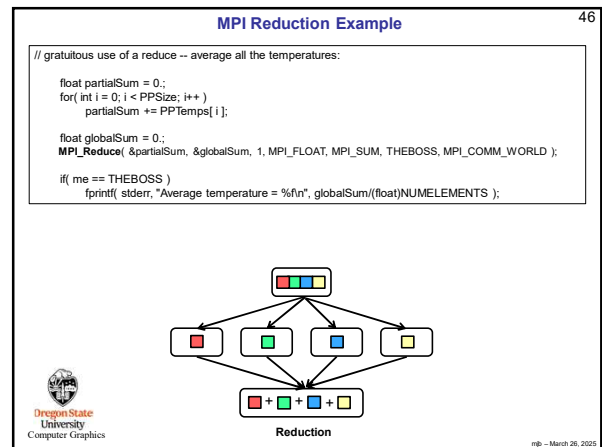
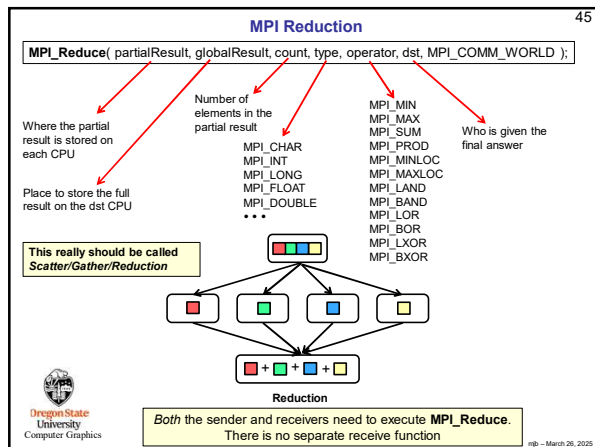
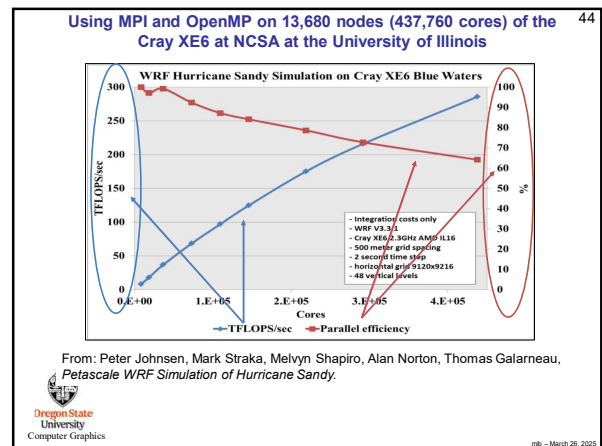
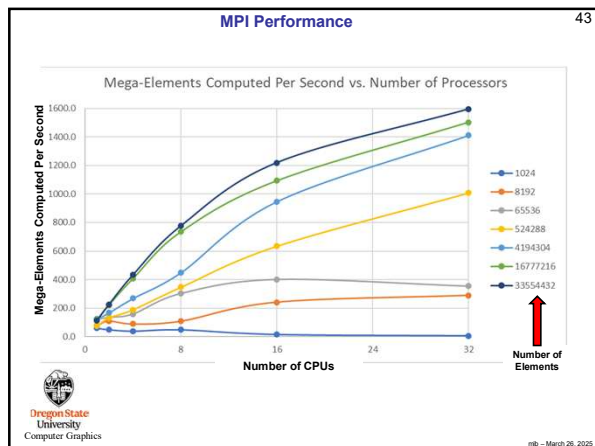
41

```
// update the local dataset:
for( int i = 0; i < PPSize; i++ )
{
    PPTemps[i] = NextTemps[i];
}
}
```

Oregon State University
Computer Graphics

mp - March 26, 2025





MPI Timing

49

```
double MPI_Wtick( );
```

Returns the resolution of the clock, in seconds.

```
double MPI_Wtime( );
```

Returns the time, in seconds, since "some time in the past".

Warning: the clocks on the different CPUs are not guaranteed to be synchronized!



Oregon State
University
Computer Graphics

mp - March 26, 2025

MPI Status-Checking

50

Some MPI calls have a **&status** in their argument list.

The **status** argument is declared to be of type **MPI_Status**, which is actually a struct:

```
typedef struct _MPI_Status
{
    int MPI_SOURCE;
    int MPI_TAG;
    int MPI_ERROR;
} MPI_Status;
```

- MPI_SOURCE is the rank of the node who sent this
- MPI_TAG is the tag used during the send
- MPI_ERROR is the error number that occurred

Example:

```
MPI_Status status;
MPI_Recv( myData, MYDATA_SIZE, MPI_CHAR, THEBOSS, MPI_ANY_TAG, MPI_COMM_WORLD,
&status );
```

```
printf( stderr, "Rank = %d, Tag = %d, Error Code = %d\n",
status.MPI_SOURCE, status.MPI_TAG, status.MPI_ERROR );
```



Oregon State
University
Computer Graphics

mp - March 26, 2025

MPI Error Codes

51

MPI_SUCCESS	No error	MPI_ERR_KEYVAL	Invalid keyval has been passed
MPI_ERR_BUFFER	Invalid buffer pointer	MPI_ERR_NO_MEM	MPI_ALLOC_MEM failed because memory is exhausted
MPI_ERR_COUNT	Invalid count argument	MPI_ERR_SAME	Invalid two passed to MPI_FREE_MEM
MPI_ERR_TYPE	Invalid datatype argument	MPI_ERR_INFO_KEY	Key longer than MPI_MAX_INFO_KEY
MPI_ERR_TAG	Invalid tag argument	MPI_ERR_INFO_VALUE	Value longer than MPI_MAX_INFO_VAL
MPI_ERR_COMM	Invalid communicator	MPI_ERR_INFO_NOKEY	Invalid key passed to MPI_INFO_DELETE
MPI_ERR_RANK	Invalid rank	MPI_ERR_SPAWN	Error in spawning processes
MPI_ERR_REQUEST	Invalid request (handle)	MPI_ERR_PORT	Invalid port name passed to MPI_COMM_CONNECT
MPI_ERR_ROOT	Invalid root	MPI_ERR_SERVICE	Invalid service name passed to MPI_UNPUBLISH_NAME
MPI_ERR_GROUP	Invalid group	MPI_ERR_NAME	Invalid service name passed to MPI_LOOKUP_NAME
MPI_ERR_OP	Invalid operation	MPI_ERR_WIN	Invalid win argument
MPI_ERR_TOPOLOGY	Invalid topology	MPI_ERR_SIZE	Invalid size argument
MPI_ERR_DIMS	Invalid dimension argument	MPI_ERR_DISP	Invalid disp argument
MPI_ERR_ARG	Invalid argument of some other kind	MPI_ERR_INFO	Invalid info argument
MPI_ERR_UNKNOWN	Unknown error	MPI_ERR_LOCKTYPE	Invalid locktype argument
MPI_ERR_TRUNCATE	Message truncated on receive	MPI_ERR_ASSERT	Invalid assert argument
MPI_ERR_OTHER	Known error not in this list	MPI_ERR_RMA_CONFLICT	Conflicting accesses to window
MPI_ERR_INTERR	Internal MPI (implementation) error	MPI_ERR_RMA_SYNC	Wrong synchronization of RMA calls
MPI_ERR_NI_STATUS	Error code is in name		
MPI_ERR_PENDING	Pending request		
MPI_ERR_FILE	Invalid file handle		
MPI_ERR_NOT_SAME	Collective argument not identical on all processes, or collective routines called in a different order by different processes		
MPI_ERR_AMODE	Error related to the amode passed to MPI_FILE_OPEN		
MPI_ERR_UNSUPPORTED_DATAREP	Unsupported datatype passed to MPI_FILE_SET_VIEW		
MPI_ERR_UNSUPPORTED_OPERATION	Unsupported operation, such as seeking on a file which supports sequential access only		
MPI_ERR_NO_SUCH_FILE	File does not exist		
MPI_ERR_FILE_EXISTS	File exists		
MPI_ERR_BAD_FILE	Invalid file name (e.g., path name too long)		
MPI_ERR_ACCESS	Permission denied		
MPI_ERR_NO_SPACE	Not enough space		
MPI_ERR_QUOTA	Quota exceeded		
MPI_ERR_READ_ONLY	Read-only file or file system		
MPI_ERR_FILE_ALREADY	File operation could not be completed, as the file is currently open by some process		
MPI_ERR_CUP_DATAREP	Conversion functions could not be registered because a data representation identifier that was already defined was passed to MPI_REGISTER_DATAREP		
MPI_ERR_CONVERSION	An error occurred in a user supplied data conversion function.		
MPI_ERR_IO	Other I/O error		
MPI_ERR_LASTCODE	Last error code		



Oregon State
University
Computer Graphics

mp - March 26, 2025