# High Performance Scientific Computing
# MPI III.

SciNet HPC Consortium
University of Toronto

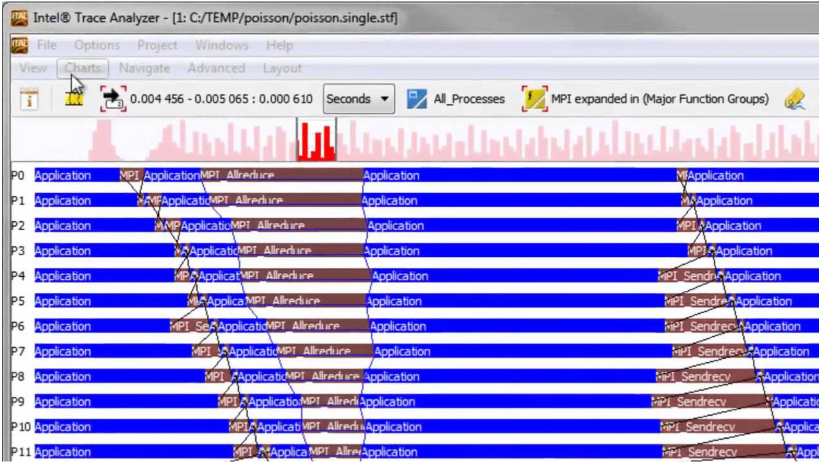
Winter 2014

# Message Passing Interface (MPI)

## Review: MPI I. & II.

- MPI: Library for sending messages
- MPI Basics (MPI_Init, size, rank, MPI_COMM_WORLD)
- MPI utils (mpirun, mpic++)
- Pairwise Communication
  - Send & Recv
  - Deadlocks
  - Sendrecv
- Collectives (MPI_Allreduce)
- Domain Decomposition

# MPI: Blocking



Intel® Trace Analyzer - [1: C:/TEMP/poisson/poisson.single.stf]

# Message Passing Interface (MPI)

## Non-Blocking Communications

- Mechanism for overlapping/interleaving communications and useful computations
- Avoid deadlocks
- Can avoid system buffering, memory-to-memory copying and improve performance

# Message Passing Interface (MPI)

## Non-Blocking Communications

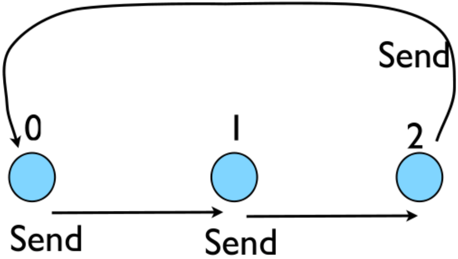
- Mechanism for overlapping/interleaving communications and useful computations
- Avoid deadlocks
- Can avoid system buffering, memory-to-memory copying and improve performance

## Non-Blocking: MPI_Isend, MPI_Irecv

- Returns immediately, posting request to system to initiate communication.
- However, communication is not completed yet.
- Cannot tamper with the memory provided in these calls until the communication is completed.

# MPI: Send Left, Receive Right with Periodic BC's

Send in a loop



# MPI: Send Left, Receive Right with Periodic BC's

```
{
  ...
  //Pass to left
  left = rank-1;
  if (left <0) left = size-1; // Periodic BC
  right = rank+1;
  if (right >= size) right =0; // Periodic BC
  msgsent = rank*rank;
  msgrcvd = -999.;
  ...
}
```

# MPI: Send Left, Receive Right with Periodic BC's - fixed

```
{
  ...
  //Even/odd message passing to avoid deadlock
  if ((rank % 2) == 0) {
    ierr = MPI_Ssend(&msgsent, 1, MPI_DOUBLE, right, tag,
    MPI_COMM_WORLD);
    ierr = MPI_Recv(&msgrcvd, 1, MPI_DOUBLE, left, tag,
    MPI_COMM_WORLD, &rstatus);
  } else {
    ierr = MPI_Recv(&msgrcvd, 1, MPI_DOUBLE, left, tag,
    MPI_COMM_WORLD, &rstatus);
    ierr = MPI_Ssend(&msgsent, 1, MPI_DOUBLE, right, tag,
    MPI_COMM_WORLD);
  }
}
```

# MPI: Send Left, Receive Right with Periodic BC's - Sendrecv

```
{
  ...
  //Replace separate Send/Recv's with Sendrecv
  ierr = MPI_Sendrecv(&msgsent, 1, MPI_DOUBLE,
  right, tag, &msgrcvd, 1, MPI_DOUBLE, left, tag,
  MPI_COMM_WORLD, &rstatus);

  cout<<rank<<":  Sent "<<msgsent<<" and got "<<msgrcvd<<endl;
  ierr = MPI_Finalize();
  return 0;
}
```

# MPI: Send Left, Receive Right: Non-Blocking

```
{
  ...
  //Non-Blocking
  MPI_Request sendreq, recvreq;
  MPI_Status sendstat, recvstat;
  ierr = MPI_Isend(&msgsent, 1, MPI_DOUBLE, right, tag,
  MPI_COMM_WORLD,&sendreq);
  ierr = MPI_Irecv(&msgrcvd, 1, MPI_DOUBLE, left, tag,
  MPI_COMM_WORLD,&recvreq);

  // do other work here

  ierr = MPI_Wait(sendreq, sendstat); ierr =
  MPI_Wait(recvreq, recvstat);
}
```

# MPI: Non-Blocking Isend & Irecv

```
ierr = MPI_Isend(sendptr, count, MPI_TYPE,
destination,tag, Communicator, MPI_Request)
ierr = MPI_Irecv(rcvptr, count, MPI_TYPE,
source, tag,Communicator, MPI_Request)
```

- `sendptr/rcvptr`: pointer to message
- `count`: number of elements in ptr
- `MPI_TYPE`: one of MPI_DOUBLE, MPI_FLOAT, MPI_INT, MPI_CHAR, etc.
- `destination/source`: rank of sender/reciever
- `tag`: unique id for message pair
- `Communicator`: MPI_COMM_WORLD or user created
- `MPI_Request`: Identify comm operations

# MPI: Wait & Waitall

- Will block until the communication(s) complete

```
ierr = MPI_Wait(MPI_Request *, MPI_Status *)
ierr = MPI_Waitall(count, MPI_Request *, MPI_Status *)
```

- `MPI_Request`: Identify comm operation(s)
- `MPI_Status`: Status of comm operation(s)
- `count`: Number of comm operations(s)

# MPI: Test

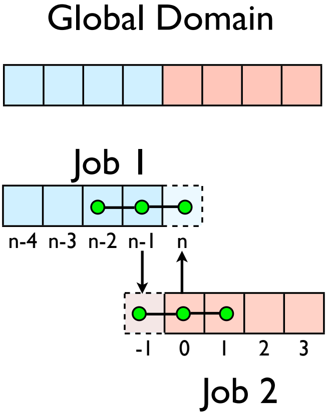
- Does not block, returns immediately
- Provides a mechanism for overlapping communication and computation

```
ierr = MPI_Test(MPI_Request *, flag, MPI_Status *)
```

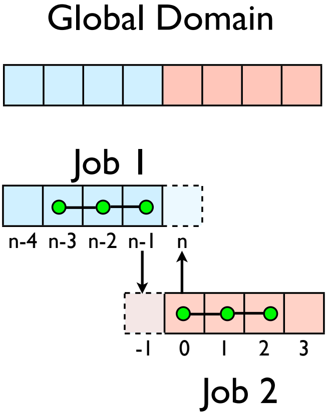
- `MPI_Request`: Identify comm operation(s)
- `MPI_Status`: Status of comm operation(s)
- `flag`: true if comm complete; false if not sent/recv yet

# Diffusion: Had to wait for communications to compute

- Could not compute end points without guardcell data
- All work halted while all communications occurred
- Significant parallel overhead



# Diffusion: *Had* to wait?



- But inner zones could have been computed just fine
- Ideally, would do inner zones work while communications is being done; then go back and do end points.

# MPI: Diffusion Non-Blocking

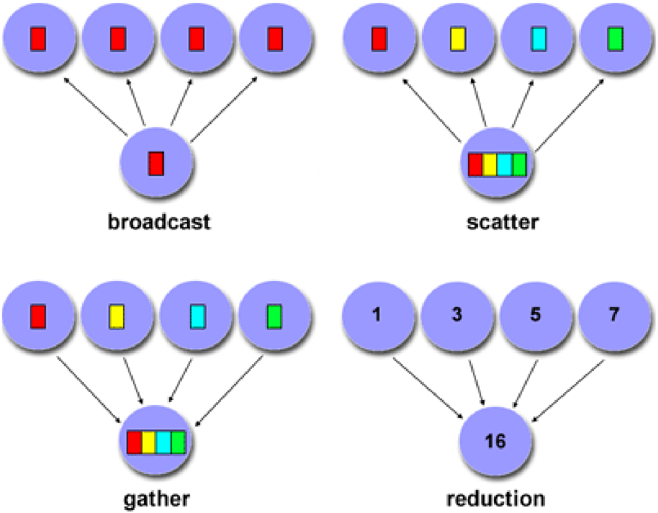
```
MPI_Request request[4];
MPI_Status status[4];
for (int step=0; step < nsteps; step++) {
  //Send Right Guardcell
  ierr = MPI_Isend(rho[n]], 1, MPI_DOUBLE, right,
  righttag, MPI_COMM_WORLD,&request[0]);
  ierr = MPI_Irecv(rho[0], 1, MPI_DOUBLE, left,
  righttag, MPI_COMM_WORLD,&reques[1]);
  //Send Left Guardcell
  ierr = MPI_Isend(rho[1]], 1, MPI_DOUBLE, right,
  lefttag, MPI_COMM_WORLD,&request[2]);
  ierr = MPI_Irecv(rho[n+1], 1, MPI_DOUBLE, left,
  lefttag, MPI_COMM_WORLD,&reques[3]);

  //Evolve timestep here
  ierr = MPI_Waitall(4, request, status);
}
```

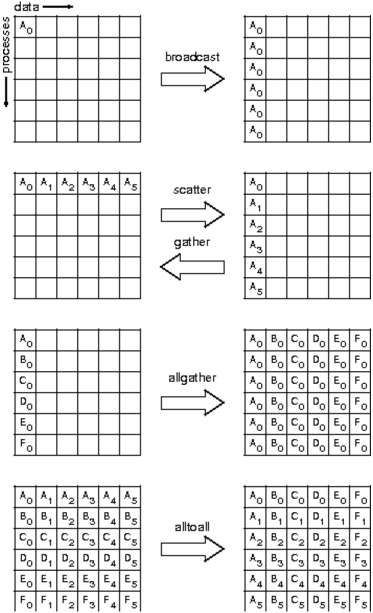
# MPI Collectives

- All processes in a group participate in communication, by calling the same function with matching arguments.
- Types:
  - Synchronization: MPI_Barrier
  - Data Movement: MPI_Bcast, MPI_Scatter, MPI_Gather, MPI_Alltoall
  - Collective Computation: MPI_Allreduce
- Collective routines are blocking

# MPI Collectives



# MPI Collectives



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

broadcast

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ |  |  |  |  |  |
| $A_0$ |  |  |  |  |  |
| $A_0$ |  |  |  |  |  |
| $A_0$ |  |  |  |  |  |
| $A_0$ |  |  |  |  |  |
| $A_0$ |  |  |  |  |  |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

scatter

gather

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ |  |  |  |  |  |
| $A_1$ |  |  |  |  |  |
| $A_2$ |  |  |  |  |  |
| $A_3$ |  |  |  |  |  |
| $A_4$ |  |  |  |  |  |
| $A_5$ |  |  |  |  |  |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ |  |  |  |  |  |
| $B_0$ |  |  |  |  |  |
| $C_0$ |  |  |  |  |  |
| $D_0$ |  |  |  |  |  |
| $E_0$ |  |  |  |  |  |
| $F_0$ |  |  |  |  |  |

allgather

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
| $B_0$ | $B_1$ | $B_2$ | $B_3$ | $B_4$ | $B_5$ |
| $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ |
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
| $E_0$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ |
| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ |

alltoall

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $A_0$ | $B_0$ | $C_0$ | $D_0$ | $E_0$ | $F_0$ |
| $A_1$ | $B_1$ | $C_1$ | $D_1$ | $E_1$ | $F_1$ |
| $A_2$ | $B_2$ | $C_2$ | $D_2$ | $E_2$ | $F_2$ |
| $A_3$ | $B_3$ | $C_3$ | $D_3$ | $E_3$ | $F_3$ |
| $A_4$ | $B_4$ | $C_4$ | $D_4$ | $E_4$ | $F_4$ |
| $A_5$ | $B_5$ | $C_5$ | $D_5$ | $E_5$ | $F_5$ |

# MPI Collectives: Barrier

- Blocks calling process until all group members have called it.
- Decreases performance. Try to avoid using it explicitly.

```
ierr = MPI_Barrier(Comm)
```

- `Communicator`: MPI_COMM_WORLD or user created

# MPI Collectives: Broadcast

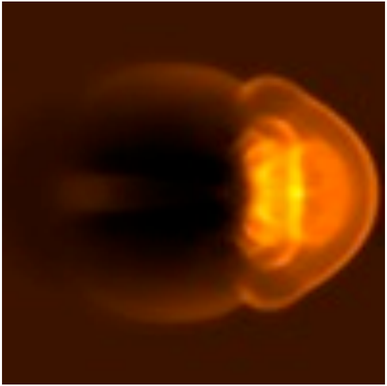
- Broadcasts a message from process with rank “root” to all processes in group, including itself.
- Amount of data sent must be equal to amount of data received.

```
ierr = MPI_Bcast(void *buf, count, MPI_Type, root, Comm)
```

- **buf**: buffer of data to send/recv
- **count**: number of elements in buf
- **MPI_TYPE**: one of MPI_DOUBLE, MPI_FLOAT, MPI_INT, MPI_CHAR, etc.
- **root**: “root” processor to send from
- **Communicator**: MPI_COMM_WORLD or user created

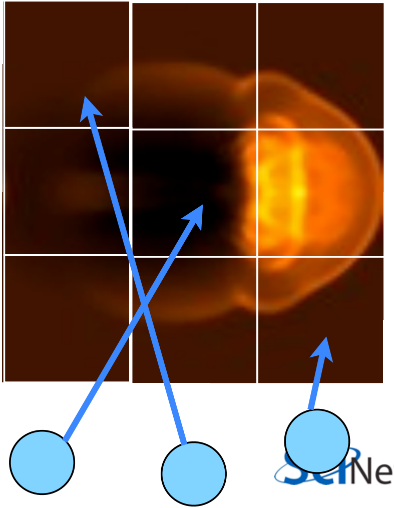
# MPI-IO

- Would like the new, parallel version to still be able to write out single output files.
- But at no point does a single processor have entire domain...



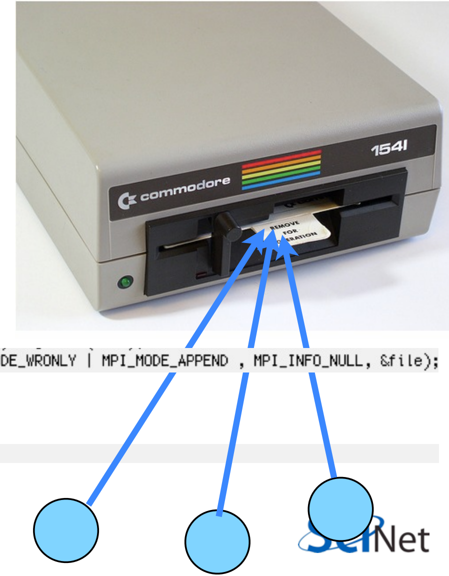
# Parallel I/O

- Each processor has to write its own piece of the domain..
- without overwriting the other.
- Easier if there is global coordination



# MPI-IO

- Uses MPI to coordinate reading/writing to single file



```
ierr = MPI_File_open(MPI_COMM_WORLD,filename, MPI_MODE_WRONLY | MPI_MODE_APPEND , MPI_INFO_NULL, &file);
```

...stuff...

```
ierr = MPI_File_close(&file);
```

- Coordination -- *collective* operations.

# MPI-IO: Example

```
{
  ...
  MPI_Offset offset = (msgsize*rank);
  MPI_File file;
  MPI_Status stat;

  MPI_File_open(MPI_COMM_WORLD, "helloworld.txt",
  MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL,
  &file);

  MPI_File_seek(file, offset, MPI_SEEK_SET);
  MPI_File_write(file, msg, msgsize, MPI_CHAR, &stat);
  MPI_File_close(&file);
  ...
}
```

# MPI-IO: Example

```
{
  ...
  MPI_Offset offset = (msgsize*rank);
  MPI_File file;
  MPI_Status stat;

  MPI_File_open(MPI_COMM_WORLD, "helloworld.txt",
  MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL,
  &file);

  //Collective Coordinated Write
  MPI_File_write_at_all(file, offset, msg, msgsize,
  MPI_CHAR, &stat);
  MPI_File_close(&file);
  ...
}
```

# MPI-IO: MPI_File_open

- MPI_File_open

```
ierr = MPI_File_open(communicator, filename, mode,
MPI_Info, MPI_File); ierr = MPI_File_close(MPI_File);
```

- `communicator`: MPI_COMM_WORLD or user created
- `char * filename`: character string filename
- `int mode`: Access modes, MPI_MODE_CREATE, MPI_MODE_WRONLY, MPI_MODE_RDWR, etc.
- `MPI_Info`: extra info or MPI_INFO_NULL
- `MPI_File`: MPI file handle

# MPI-IO: MPI_File_write_at_all

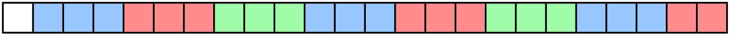
- Collective operation across all Comm processors

```
ierr = MPI_File_write_at_all(MPI_File, MPI_Offset,buffer,
count, MPI_Type, MPI_Status)
```

- `MPI_File`: MPI file handle
- `MPI_Offset`: MPI file offset location
- `void * buffer`: buffer of data to write
- `int count`: number of elements in ptr
- `MPI_TYPE`: one of MPI_FLOAT, MPI_INT, MPI_CHAR, etc.
- `MPI_Request`: Identify comm operations

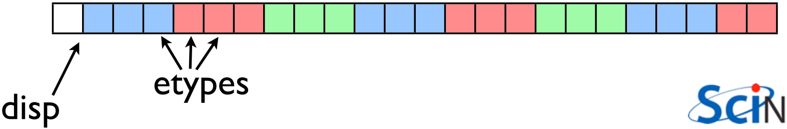
# MPI-IO File View

- Each process has a view of the file that consists of only of the parts accessible to it.
- For writing, hopefully non-overlapping!
- Describing this - how data is laid out in a file - is very similar to describing how data is laid out in memory...



# MPI-IO File View

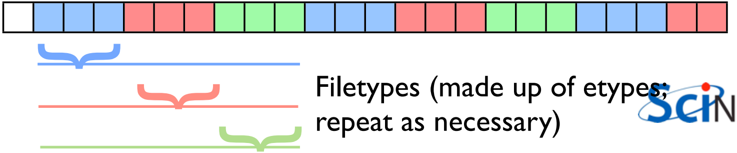
- int MPI_File_set_view(
  MPI_File fh,
  MPI_Offset disp, /* displacement in *bytes* from start */
  MPI_Datatype etype, /* elementary type */
  MPI_Datatype filetype, /* file type; prob different for each proc */
  char *datarep, /* ‘native’ or ‘internal’ */
  MPI_Info info) /* MPI_INFO_NULL for today */



# MPI-IO File View

- int MPI_File_set_view(

```
    MPI_File fh,
    MPI_Offset disp,          /* displacement in bytes from start */
    MPI_Datatype etype,       /* elementary type */
    MPI_Datatype filetype,    /* file type; prob different for each proc */
    char *datarep,            /* 'native' or 'internal' */
    MPI_Info info)            /* MPI_INFO_NULL */
```



Filetypes (made up of etypes; repeat as necessary)

# MPI-IO File Write

-
  ```
  int MPI_File_write_all(
      MPI_File fh,
      void *buf,
      int count,
      MPI_Datatype datatype,
      MPI_Status *status)
  ```

Writes (_all: collectively) to part of file within view.