MPI-1

January 22, 2021

Resources for MPI

- Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker and Jack Dongarra, MPI - The Complete Reference, Second Edition, Volume 1, The MPI Core.
- William Gropp, Ewing Lusk, Anthony Skjellum, Using MPI : portable parallel programming with the message-passing interface, 3rd Ed., Cambridge MIT Press, 2014.
- https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

Executing MPI programs

- Check `which mpicc` and `which mpiexec`
  - Update PATH environment variable
- Compile
  - mpicc –o filename filename.c
- Run
  - mpiexec –np 4 –f hostfile filename [Often “No such file” error]
  - mpiexec –np 4 –f hostfile ./filename
- Additional tips
  - Autogenerate “hostfile” (mandatory for assignment submission)
  - https://git.cse.iitk.ac.in/tusharag/NodeAllocator

Message Passing Paradigm

- Message sends and receives
- Explicit communication

Communication patterns

- Point-to-point
- Collective

MPI Basics

- Process is identified by rank
- Communicator specifies communication context

Message Envelope

Source

Destination

Communicator

Tag (0:MPI_TAG_UB)

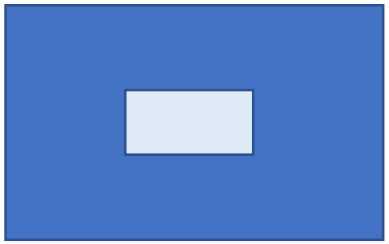
MPI Data Types

- MPI_BYTE
- MPI_CHAR
- MPI_INT
- MPI_FLOAT
- MPI_DOUBLE

Point-to-point Communication

Blocking send and receive

MPI_Send

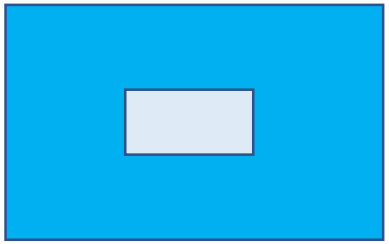


SENDER

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

Tags should match

MPI_Recv



RECEIVER

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

Simple Send/Recv Code

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

```
//From https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf Chap-3

#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
  char message[20];
  int myrank;
  MPI_Status status;
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  if (myrank == 0)    /* code for process 0 */
  {
    strcpy(message,"Hello, there");
    MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
    printf("received :%s\n", message);
  }
  MPI_Finalize();
  return 0;
}
~
~
```

DEMO – Simple Send Message

Homework

1. Use MPI_Comm_rank and MPI_Comm_size

   Output

   - Rank 1 of 2 received : Hello, there

2. Check whether this code runs on more than 2 processes or not?

MPI_Status

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

- Source rank
- Message tag
- Message length
  - MPI_Get_count

MPI_Get_count

```
MPI_Status status;

MPI_Init(&argc, &argv);
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
MPI_Comm_size( MPI_COMM_WORLD, &size );

if (myrank == 0)    /* code for process zero */
{
  MPI_Send(arr, numElements, MPI_INT, 1, 99, MPI_COMM_WORLD);
}
else if (myrank == 1)  /* code for process one */
{
  int count, recvarr[numElements];
  MPI_Recv(recvarr, numElements, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
  MPI_Get_count (&status, MPI_INT, &count);
  printf("Rank %d of %d received %d elements\n", myrank, size, count);

}
```

Output

- Rank 1 of 2 received 1000 elements

DEMO – MPI_Get_count

```c
#include <string.h>
#include "mpi.h"

#define numElements 1000

int main( int argc, char *argv[])
{
  int arr[numElements] = {0};
  int myrank, size;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  MPI_Comm_size( MPI_COMM_WORLD, &size );

  if (myrank == 0)    /* code for process zero */
  {
    MPI_Send(arr, numElements, MPI_INT, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process one */
  {
    int count, recvarr[numElements];
    MPI_Recv(recvarr, numElements, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
    MPI_Get_count (&status, MPI_INT, &count);
    printf("Rank %d of %d received %d elements\n", myrank, size, count);

  }

  MPI_Finalize();
  return 0;
}
```

30,0-1 Bot

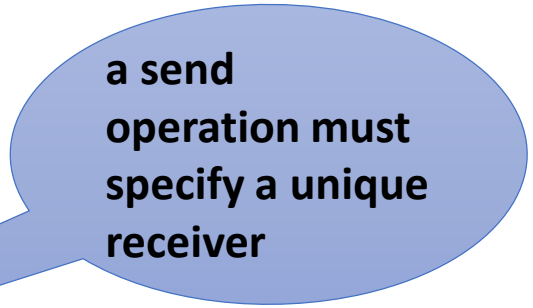
Timing Send/Recv

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
MPI_Comm_size(MPI_COMM_WORLD, &size);

// initialize data
for (int i=0; i<count; i++)
 buf[i] = myrank+i;

sTime = MPI_Wtime();
if (myrank == 0)
 MPI_Send (buf, count, MPI_INT, 1, 1, MPI_COMM_WORLD);
else
 if (myrank == 1)
  MPI_Recv (buf, count, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
eTime = MPI_Wtime();
time = eTime - sTime;
```

DEMO – Timing

```
  MPI_Status status;
  double sTime, eTime, time;

  MPI_Init(&argc, &argv);

  int count = atoi (argv[1]);
  int buf[count];

  MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // initialize data
  for (int i=0; i<count; i++)
   buf[i] = myrank+i;

  sTime = MPI_Wtime();
  if (myrank == 0)
   MPI_Send (buf, count, MPI_INT, 1, 1, MPI_COMM_WORLD);
  else
   if (myrank == 1)
    MPI_Recv (buf, count, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
  eTime = MPI_Wtime();
  time = eTime - sTime;

  printf ("%lf\n", time);

  MPI_Finalize();
  return 0;

}

class $ cat send.c
```

Many-to-one Sends




```
if (myrank)
{
  MPI_Send(arr, 20, MPI_INT, 0, myrank, MPI_COMM_WORLD);
}
else
{
  int count, recvarr[size][20];
  for (int i=1; i<size; i++)
  {
    MPI_Recv(recvarr[i], 20, MPI_INT, i, i, MPI_COMM_WORLD, &status); // slower
    printf("Rank %d of %d received from rank %d with tag %d\n", myrank, size, status.MPI_SOURCE, status.MPI_TAG);
  }
}
```

- MPI_Send (Blocking send)
- MPI_Recv (Blocking receive)

MPI_ANY_*

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

- MPI_ANY_SOURCE
  - Receiver may specify wildcard value for source
- MPI_ANY_TAG
  - Receiver may specify wildcard value for tag

Receive Out-of-order

```
if (myrank)
{
  MPI_Send(arr, 20, MPI_INT, 0, myrank, MPI_COMM_WORLD);
}
else
{
  int count, recvarr[size][20];
  for (int i=1; i<size; i++)
  {
    //MPI_Recv(recvarr[i], 20, MPI_INT, i, i, MPI_COMM_WORLD, &status);       // may be slower
    MPI_Recv(recvarr[i], 20, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    printf("Rank %d of %d received from rank %d with tag %d\n", myrank, size, status.MPI_SOURCE, status.MPI_TAG);
  }
}
```

Rectify the receive buffer location

No specific order of output

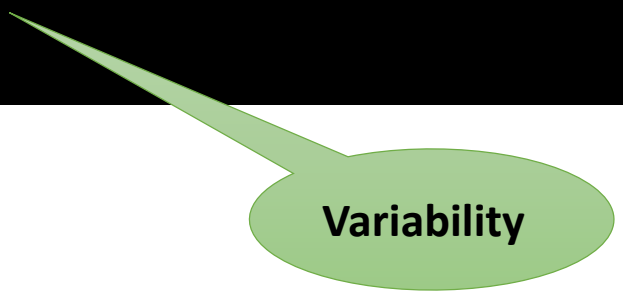
DEMO

One-to-many Sends

```
sTime = MPI_Wtime();
if (myrank == 0)
 for (int r=1; r<size; r++)
  MPI_Send (buf, count, MPI_INT, r, r, MPI_COMM_WORLD);
else
 MPI_Recv (buf, count, MPI_INT, 0, myrank, MPI_COMM_WORLD, &status);
eTime = MPI_Wtime();
time = eTime - sTime;

// obtain max time
MPI_Reduce (&time, &maxTime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
if (!myrank) printf ("%lf\n", maxTime);
```

Output

```
class $ for i in `seq 1 5`; do mpirun -np 4 ./onetomanysends 100 ; done
0.000032
0.000014
0.000013
0.000013
0.000012
class $ for i in `seq 1 5`; do mpirun -np 4 ./onetomanysends 1000000 ; done
0.002296
0.002655
0.002453
0.002084
0.002210
class $ for i in `seq 1 5`; do mpirun -np 8 ./onetomanysends 1000000 ; done
0.004122
0.005187
0.004641
0.004474
0.004451
class $ for i in `seq 1 5`; do mpirun -np 8 -hosts csews1:2,csews10:2,csews20:2,csews30:2 ./onetomanysends 1000000 ; done
0.205539
0.215736
0.217059
0.214883
0.208585
class $
```



Communication Subsystem

- Communication using sockets (one of the options)
- MPI handles communications, progress etc.
- Communication channels determine performance
- Shared-memory queue for intranode messaging
  - Per-process send/recv queue

Send queue Recv queue

Reference: Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem by Buntinas et al.

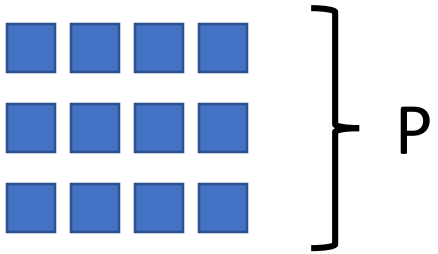
Sum of squares of N numbers

Serial

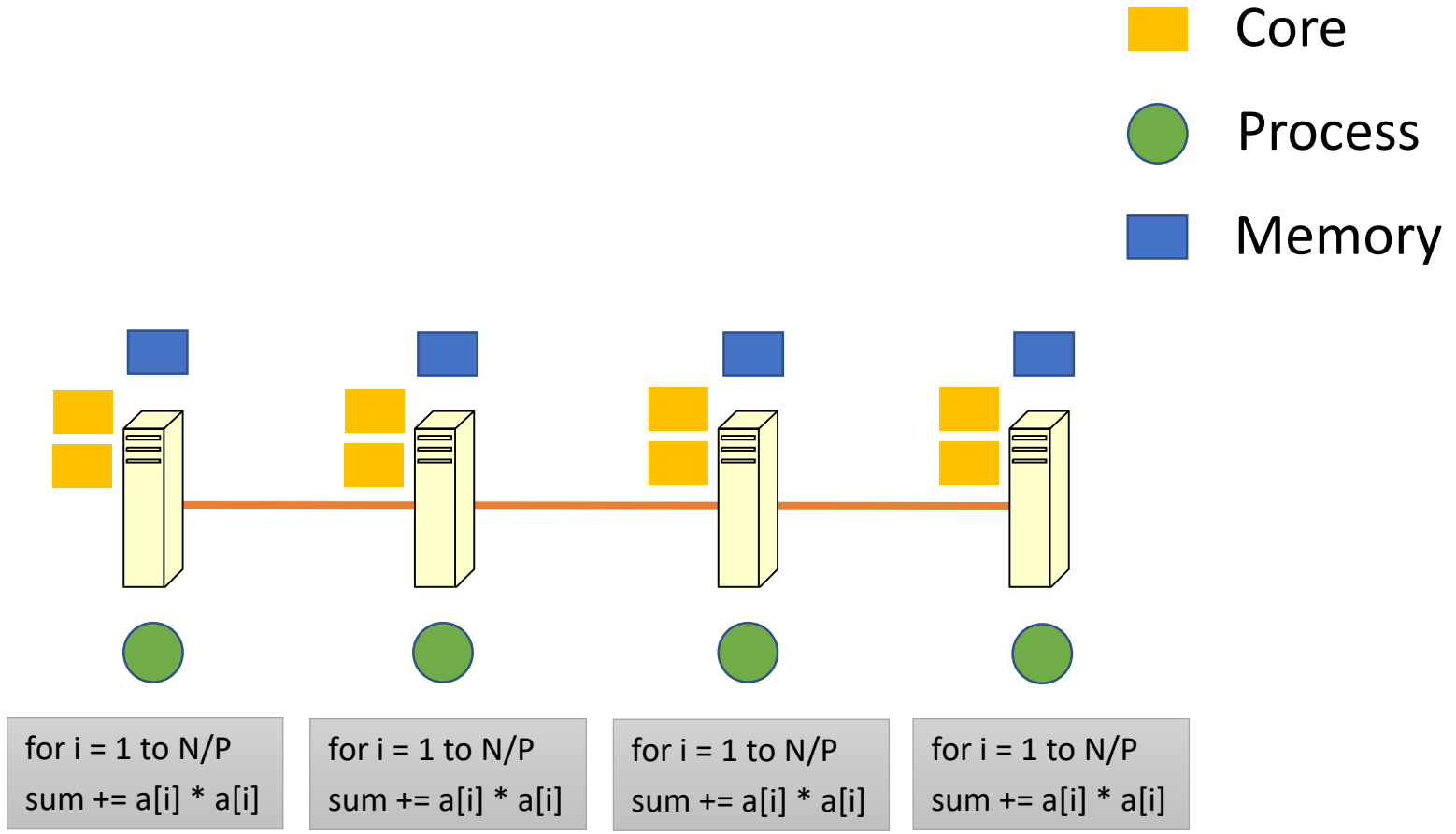
```
for i = 1 to N
sum += a[i] * a[i]
```

Parallel

```
for i = 1 to N/P
sum += a[i] * a[i]
collate result
```



Sum of Squares



SIMD Parallelism

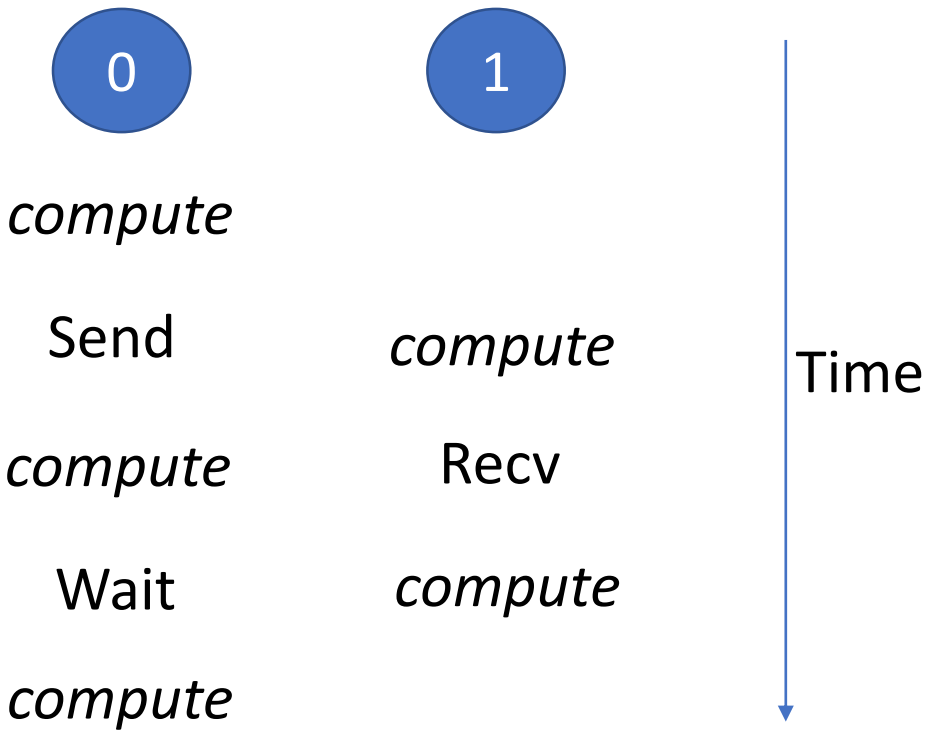
```
for i = N/P * rank ; i < N/P * (rank+1) ; i++
    localsum += a[i] * a[i]
Collect localsum, add up at one of the ranks
```

Output

```
class $ mpirun -np 18 -f hostfile ./parsum 10000000 | sort -k3n
6: Time: 0.003012 0.060570
7: Time: 0.003012 0.102022
10: Time: 0.003028 0.222636
8: Time: 0.003035 0.144442
9: Time: 0.005718 0.177058
11: Time: 0.005723 0.255620
2: Time: 0.006853 0.012692
3: Time: 0.006865 0.014576
14: Time: 0.007570 0.370506
12: Time: 0.007571 0.287444
17: Time: 0.007571 0.489290
13: Time: 0.007579 0.324584
15: Time: 0.007589 0.408255
16: Time: 0.007607 0.446781
1: Time: 0.008052 0.003379
5: Time: 0.008055 0.010920
0: Time: 0.008059 0.491419
4: Time: 0.008060 0.009140
class $ cat hostfile
csews10:6
csews11:6
csews12:6
csews13:6
csews14:6
csews15:6
csews16:6
csews17:6
csews18:6
class $
```

```
class $ mpirun -np 18 -f hostfile ./parsum 10000000 | sort -k4n
2: Time: 0.008899 0.001971
3: Time: 0.008892 0.004043
1: Time: 0.007026 0.004272
4: Time: 0.006996 0.013603
5: Time: 0.007045 0.015271
6: Time: 0.007582 0.043268
7: Time: 0.007561 0.083797
8: Time: 0.007579 0.124421
9: Time: 0.007561 0.162050
10: Time: 0.010233 0.190928
11: Time: 0.010234 0.229928
12: Time: 0.007656 0.276896
13: Time: 0.007631 0.317010
14: Time: 0.010614 0.344311
15: Time: 0.010599 0.386936
16: Time: 0.007566 0.433467
17: Time: 0.007586 0.469213
0: Time: 0.007014 0.480969
class $ cat hostfile
csews10:6
csews11:6
csews12:6
csews13:6
csews14:6
csews15:6
csews16:6
csews17:6
csews18:6
class $
```

Computation Communication Overlap



Non-blocking Point-to-Point

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)

- MPI_Wait (request, status)
- MPI_Waitall (count, request, status)

Code

http://web.cse.iitk.ac.in/users/pmalakar/cs633/2020-21/code/Jan19-22.tar.gz

Next lecture will be uploaded on Jan 29 (Tuesday is a holiday)