MPI Tutorial

Dhanashree N P

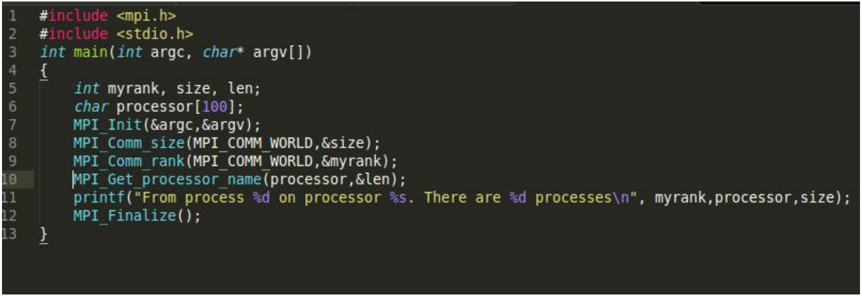
February 24, 2016

MPI - Message Passing Interface

- A standard for message passing library
- Efficient, Portable, Scalable, Vendor Independent
- C, Fortran
- Message Passing Parallel Programming Model
- MPI-3
- Distributed, Shared, Hybrid Memory
- Some implementations - OpenMPI, MPICH2, IBM Platform MPI etc.
- Different implementations support different versions and functionalities of the standard

Use MPI with C for Assignment 2

MPI Routines

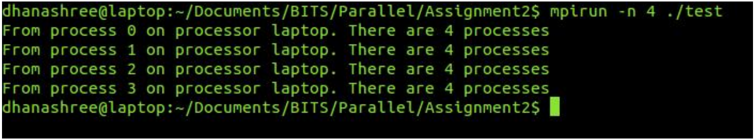


```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
    int myrank, size, len;
    char processor[100];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    MPI_Get_processor_name(processor,&len);
    printf("From process %d on processor %s. There are %d processes\n", myrank,processor,size);
    MPI_Finalize();
}
```

- MPI_Init(), MPI_Finalize()
- MPI_Comm_size(), MPI_Comm_rank()
- MPI_Get_processor_name()
- MPI_Abort(), MPI_Get_version(), MPI_Initialized()

Compilation and Program Execution

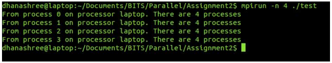
mpicc -o test.c test
mpirun -n 4 ./test



```
dhanashree@laptop:~/Documents/BITS/Parallel/Assignment2$ mpirun -n 4 ./test
From process 0 on processor laptop. There are 4 processes
From process 1 on processor laptop. There are 4 processes
From process 2 on processor laptop. There are 4 processes
From process 3 on processor laptop. There are 4 processes
dhanashree@laptop:~/Documents/BITS/Parallel/Assignment2$
```

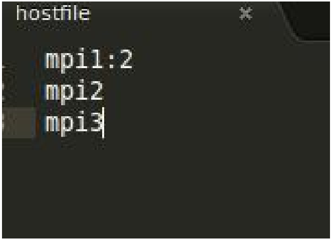
Multiple hosts

To run on multiple hosts
mpirun -n 4 -host hostname1,hostname2 ./test
mpirun -n 4 -hostfile filename ./test



```
dhanashree@laptop:~/Documents/BITS/Parallel/Assignment2$ mpirun -n 4 ./test
From process 0 on processor laptop. There are 4 processes
From process 1 on processor laptop. There are 4 processes
From process 2 on processor laptop. There are 4 processes
From process 3 on processor laptop. There are 4 processes
dhanashree@laptop:~/Documents/BITS/Parallel/Assignment2$
```

(a) etc hosts file



```
mpi1:2
mpi2
mpi3
```

(b) hostfile

Some Points to Note

- Only one MPI_Init and MPI_Finalize in a program
- Do not declare functions or variables starting with MPI_ or PMPI_ in the program

Communicators, Groups and Ranks

- A communication domain is the set of processes allowed to communicate with each other.
- MPI_Comm type variables eg. MPI_COMM_WORLD
- Parameter to all message passing primitives
- Each process belongs to many different communication domains
- Rank(task id) of the calling process is a unique integer identifier in the range 0 to n-1.

Unicast Communication Primitives

Types of Operations

- Synchronous, Blocking, Non-blocking, Buffered, Combined etc.
- Blocking v/s Non-blocking
- System buffer v/s Application buffer
- Order and Fairness

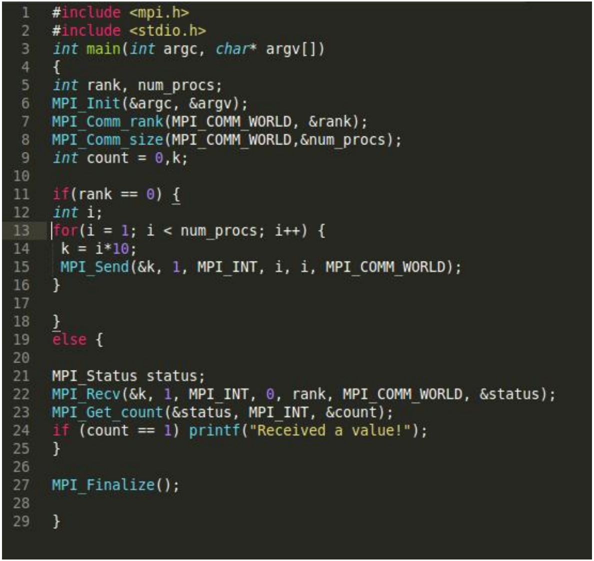
Unicast Communication Primitives

- Blocking - MPI_Send(), MPI_Recv()
  MPI_Send(buffer,count,type,dest,tag,comm)
  MPI_Recv(buffer,count,type,source,tag,comm,status)

- Non-Blocking- MPI_ISend(), MPI_IRecv()
  MPI_Isend(buffer,count,type,dest,tag,comm,request)
  MPI_Irecv(buffer,count,type,source,tag,comm,request)

  buffer - reference to data that has to be sent/received.
  count- number of data elements of a particular type.
  source - rank of sender, dest - rank of receiver.
  tag - MPI_ANY_TAG or any non-negative integer
  comm - by default MPI_COMM_WORLD

*****Make Sure to avoid deadlocks!*****



```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
int rank, num_procs;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD,&num_procs);
int count = 0,k;

if(rank == 0) {
int i;
for(i = 1; i < num_procs; i++) {
 k = i*10;
 MPI_Send(&k, 1, MPI_INT, i, i, MPI_COMM_WORLD);
}

}
else {

MPI_Status status;
MPI_Recv(&k, 1, MPI_INT, 0, rank, MPI_COMM_WORLD, &status);
MPI_Get_count(&status, MPI_INT, &count);
if (count == 1) printf("Received a value!");
}

MPI_Finalize();

}
```

Unicast Communication Routines

- Other flavors - Blocking - MPI_Ssend(), MPI_Bsend(), MPI_Rsend()
- Attaching a buffer - size in bytes
  MPI_Buffer_attach (&buffer,size)
  MPI_Buffer_detach (&buffer,size)
- Send a message and post a receive before blocking
  MPI_Sendrecv (&sendbuf,sendcount,sendtype,dest,sendtag, &recvbuf,recvcount,recvtype,source,recvtag, comm,&status)
- Wait functions - MPI_Wait, MPI_Waitany, MPI_Waitall, MPI_Waitsome
- Other flavors - Non-Blocking - MPI_Issend(), MPI_Ibsend(), MPI_Irsend()
- Status check functions - MPI_Test, MPI_Testany, MPI_Testall, MPI_Testsome

Collective Communication and Computation Routines

The unicast routines were primarily for communication purposes. Some of the collective operations support computations. All the processes that are part of a communicator has to participate in collective communication.

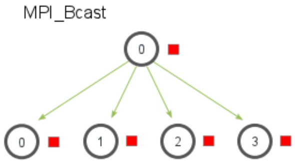
Three types of collective operations:

- Synchronization - wait till all members have reached the synchronization point
- Data movements - broadcast, scatter, gather
- Computations - reductions

These can be used with MPI primitive data types.

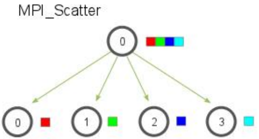
Collective Communication Routines

- MPI_Barrier(comm) - Each task executing this is blocked until all the other tasks of the same group have reached this point.
- MPI_Bcast(&buffer,count,datatype,root,comm) - The process with rank 'root' broadcasts to all other processes in the group

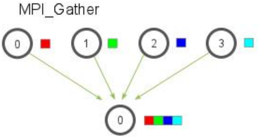


Collective Communication Routines

- MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm) - The process with rank 'root' broadcasts to all other processes in the group

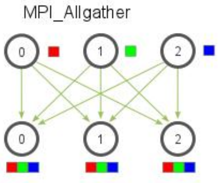


- MPI_Gather(&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm) - Reverse of scatter.



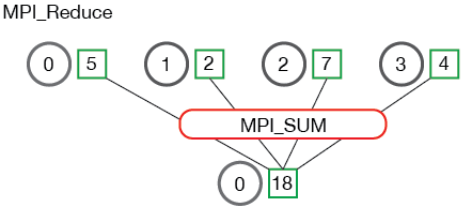
Collective Communication Routines

- MPI_Allgather(&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,comm) - Concatenation of data to all tasks in a group.



Collective Computation Routines

- MPI_Reduce(&sendbuf,&recvbuf,count,datatype,op,root,comm) - Applies a reduction operation on all tasks in the group and places the result in one task.

- MPI_Allreduce - collective computation and data movement operation
- MPI_MAX, MPI_MIN, MPI_SUM,MPI_PROD, MPI_BOR, MPI_BAND etc.

Data Types - MPI Datatype (C Data type)

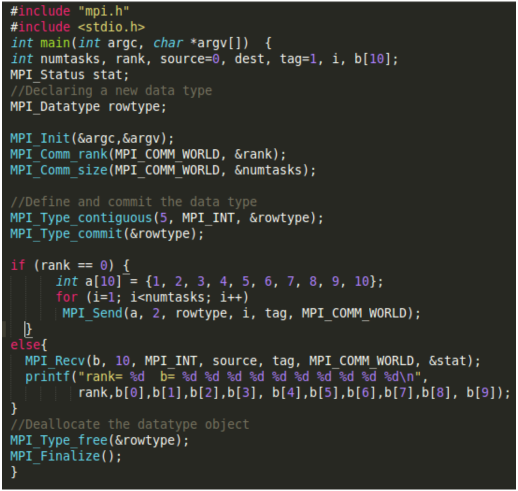
MPI_CHAR (signed char)
MPI_SHORT (signed short int)
MPI_INT (signed int)
MPI_LONG (signed long int)
MPI_UNSIGNED_CHAR (unsigned char)
MPI_UNSIGNED_SHORT (unsigned short int)
MPI_UNSIGNED_LONG (unsigned long int)
MPI_UNSIGNED (unsigned int)
MPI_FLOAT (float)
MPI_DOUBLE (double)
MPI_LONG_DOUBLE (long double)
MPI_BYTE
MPI_PACKED

Derived Data Types

The following are used for derived data type creation:

- Contiguous
- Vector
- Indexed
- Struct

Derived Data Types - Example using Contiguous



```c
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])  {
int numtasks, rank, source=0, dest, tag=1, i, b[10];
MPI_Status stat;
//Declaring a new data type
MPI_Datatype rowtype;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

//Define and commit the data type
MPI_Type_contiguous(5, MPI_INT, &rowtype);
MPI_Type_commit(&rowtype);

if (rank == 0) {
      int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
      for (i=1; i<numtasks; i++)
       MPI_Send(a, 2, rowtype, i, tag, MPI_COMM_WORLD);
  }
else{
  MPI_Recv(b, 10, MPI_INT, source, tag, MPI_COMM_WORLD, &stat);
  printf("rank= %d  b= %d %d %d %d %d %d %d %d %d %d\n",
         rank,b[0],b[1],b[2],b[3], b[4],b[5],b[6],b[7],b[8], b[9]);
}
//Deallocate the datatype object
MPI_Type_free(&rowtype);
MPI_Finalize();
}
```

References

Message Passing Interface(MPI) - https://computing.llnl.gov/tutorials/mpi/
Inter group communications - http://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node114.html
Tutorials - http://mpitutorial.com/tutorials/
Introduction to Parallel Computing by Ananth Grama et al. - Section 6.3 to Section 6.7