Nonblocking Communications

Ned Nedialkov

Department of Computing and Software
McMaster University, Hamilton, Ontario
Canada

Introduction

Nonblocking communications are useful for overlapping communication with computation

That is, compute while communicating data

A nonblocking operation requests the MPI library to perform an operation (when it can)

Nonblocking operations do not wait for any communication events to complete

Nonblocking send and receive: return almost immediately

The user can modify a send [resp. receive] buffer only after send [resp. receive] is completed

There are "wait" routines to figure out when a nonblocking operation is done

MPI_Isend

Performs a nonblocking send

```
int MPI_Isend(void* buf, int count, MPI_Datatype datatype,
              int dest, int tag, MPI_Comm comm,
              MPI_Request *request)
```

buf starting address of buffer

count number of entries in buffer

datatype data type of buffer

dest rank of destination

tag message tag

comm communicator

request communication request (out)

MPI_Irecv

Performs a nonblocking receive

```
int MPI_Irecv(void* buf, int count, MPI_Datatype datatype,
              int source, int tag, MPI_Comm comm,
              MPI_Request *request)
```

buf starting address of buffer (out)

count number of entries in buffer

datatype data type of buffer

source rank of source

tag message tag

comm communicator

request communication request (out)

Wait routines

```
int MPI_Wait (MPI_Request *request, MPI_Status *status)
```

Waits for MPI_Isend or MPI_Irecv to complete

request request (in), which is out parameter in MPI_Isend and MPI_Irecv

status status (out)

Other routines include

MPI_Waitall waits for all given communications to complete

MPI_Waitany waits for any of given communications to complete

MPI_Test tests for completion of send or receive

MPI_Testany tests for completion of any previously initiated communication

Example

From `http://www.llnl.gov/computing/tutorials/mpi/samples/C/mpi_ringtopo.c`

```
/* nonb.c */
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int numtasks, rank, next, prev,
    buf[2], tag1=1, tag2=2;

  tag1=tag2=0;
  MPI_Request reqs[4];
  MPI_Status stats[4];

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

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
  prev = rank-1;
  next = rank+1;

  if (rank == 0)              prev = numtasks - 1;
  if (rank == numtasks - 1)   next = 0;

  MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1,
            MPI_COMM_WORLD, &reqs[0]);
  MPI_Irecv(&buf[1], 1, MPI_INT, prev, tag2,
            MPI_COMM_WORLD, &reqs[1]);

  MPI_Isend(&rank, 1, MPI_INT, next, tag2,
            MPI_COMM_WORLD, &reqs[2]);
  MPI_Isend(&rank, 1, MPI_INT, next, tag1,
            MPI_COMM_WORLD, &reqs[3]);

  MPI_Waitall(4, reqs, stats);
  printf("Task %d communicated with tasks %d & %d\n",
         rank,prev,next);

  MPI_Finalize();
  return 0;
}
```

Final remarks

Nonblocking send can be posted whether a matching receive has been posted or not

Send is completed when data has been copied out of send buffer

Nonblocking send can be matched with blocking receive and vice versa

Communications are initiated by sender

A communication will generally have lower overhead if a receive buffer is already posted when a sender initiates a communication