MPI Basics

Chris Kauffman

CS 499: Spring 2016 GMU
Logistics

Reading: Grama Ch 6 + 4

- Ch 6: MPI basics
- Ch 4: Communication patterns

Assignment 1

- Posted, due Thu 2/4
- Groups of 2 permitted
- A few tips on broadcast Problem 5
- Questions?

Today

Begin discussion of MPI programming
Generic Send and Receive

Minimum required functionality to do distributed memory parallel computing:

send(void *sendbuf, int nelems, int dest)
receive(void *recvbuf, int nelems, int source)

Sample Use

1   P0                        P1
2
3   a = 100;                 receive(&a, 1, 0)
4   send(&a, 1, 1);         printf("%d\n", a);
5   a=0;

▷ Proc 0 sends a single integer to Proc 1
▷ Proc 0 then 0s that integer
▷ Proc 1 receives and prints the integer
Will typically write this as a single program which every processor runs.

```c
void exchange()
{
    int a = 100;
    int my_proc = get_processor_number();
    if(my_proc == 0){
        send(&a, 1, 1);
        a=0;
    }
    else if(my_proc == 1){
        receive(&a, 1, 0);
        printf("%d\n", a);
    }
}
```

- Function to identify proc number
- Branching on proc number to take different actions
Flavors Send/Receive

- Hardware/OS support for buffered communication tends to make things run faster
- Usually have function calls available to do `send()` (blocking) and `send_nonblocking()` but must have some hardware support for it
Blocked and Unbuffered

**Blocking/Unbuffered:** no extra buffer available to hold pending sends/receives so must wait, wait until message is sent to proceed. Blocked processors are idle, do no work, which cuts into speedup.
Danger

1 P0
2
3 send(&a, 1, 1); send(&a, 1, 0);
4 receive(&b, 1, 1); receive(&b, 1, 0);

Assuming send/receive blocked/unbuffered, what’s wrong with the above code?
Blocking with Buffers

Hardware buffer support, sender and receiver have a memory minion

No buffer support: sender interrupts receiver
receive(&a, 1, 1); receive(&a, 1, 0);
send(&b, 1, 1); send(&b, 1, 0);

▶ receive() always blocks until message is obtained
▶ Does the above code work even in the buffered setting?
Non-blocking Communication

- Takes a bit more work on the programming side
- Must explicitly ensure that transaction completes with function calls
  - `isend(data, dest, status)`: send w/o waiting
  - `ireceive(data, dest, status)`: receive w/o waiting
  - `wait(status)`: wait until a message has been sent or received before moving one
MPI: Message Passing Interface

- Standardized library of functions for C/C++/Fortran
- Communicate between processors in a distributed memory machine
- Open source implementations: MPICH, Open MPI
- Proprietary: Intel, Platform, IBM, Platform, Cray
- Typically geared for particular architecture
- May exploit specifics of a particular machine
MPI In a Nutshell: 6 Essential Functions

// Initializes MPI.
int MPI_Init(int *argc, char **argv);

// Terminates MPI.
int MPI_Finalize();

// Determines the number of processes.
int MPI_Comm_size(MPI_Comm comm, int *size);

// Determines the label of the calling process.
int MPI_Comm_rank(MPI_Comm comm, int *rank);

// Sends a message.
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
              int dest, int tag, MPI_Comm comm);

// Receives a message.
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status);
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[])
{
    int rank, size;
    MPI_Init (&argc, &argv); /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size); /* get number of processes */
    int i;
    for(i=0; i<1; i++){
        printf( "Hello world from process %d of %d\n", rank, size );
    }
    MPI_Finalize();
    return 0;
}

- Note the use of MPI_COMM_WORLD which is a predefined constant corresponding to all processors.
- Can also set up other communicators that correspond to subsets of processors
Compilation and Running

- Demo using openmpi implementation
- mpirun for interactive running
- mpirun -np 4 progr sets number of "processors" to 4

```
lila [test-code]%, mpicc -o hello hello.c
lila [test-code]%, ./hello
Hello world from process 0 of 1
lila [test-code]%, mpirun hello
Hello world from process 0 of 4
Hello world from process 1 of 4
Hello world from process 2 of 4
Hello world from process 3 of 4
lila [test-code]%, mpirun -np 2 hello
Hello world from process 0 of 2
Hello world from process 1 of 2
lila [test-code]%, mpirun -np 8 hello
Hello world from process 7 of 8
Hello world from process 0 of 8
Hello world from process 2 of 8
Hello world from process 3 of 8
Hello world from process 4 of 8
```
MPI Send and Recieve

```c
int a[10], b[10];
int partner = 1;
...

// Send contents of a to partner proc with tag=1
MPI_Send(a, 10, MPI_INT, partner, 1, MPI_COMM_WORLD);

// Receive message into b from partner proc with tag=1,
// ignore status of receipt
MPI_Recv(b, 10, MPI_INT, partner, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

Analyze the program send-receive-test.c
```
Tag Trouble

```c
int a[10], b[10], myrank;
MPI_Status status;
...
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
    MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
    MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
}
else if (myrank == 1) {
    MPI_Recv(b, 10, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Recv(a, 10, MPI_INT, 0, 1, MPI_COMM_WORLD);
}
```

- Tags must be honored on receive
- Above code may deadlock if not buffered due to the misordering of tags
- Mostly we will not deal with tags (tag=1)
Issues with Untyped Data in MPI

// Sends a message.
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);

// Receives a message.
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status);

- Type of buffer is always untyped (void* buf)
- To try to get at slightly better safety, MPI has standard datatypes

  MPI_CHAR     signed char
  MPI_INT      signed int
  MPI_LONG     signed long int
  MPI_FLOAT    float
  MPI_DOUBLE   double
  MPI_BYTE     Last two used for sending
  MPI_PACKED   structure arrays

Unsigned types also available
Exercise: Heat Transfer

- Discuss conversion of the following HW1 code to an MPI version
- How is data in H divided up?
- Is communication required?
- How would one arrange MPI_Send / MPI_Recv calls?

```c
// Simulate the temperature changes for internal cells
for(t=0; t<max_time-1; t++){
    for(p=1; p<width-1; p++){
        double left_diff = H[t][p] - H[t][p-1];
        double right_diff = H[t][p] - H[t][p+1];
        double delta = -k*(left_diff + right_diff );
        H[t+1][p] = H[t][p] + delta;
    }
}
```
Some Patterns that occur in the problem

- Pair exchange of items: made easier with MPI_sendrecv
- Collecting final output for display: MPI_Gather
Exchange: Sendrecv for exchanging data between pairs

```
{
    double send[10], recv[10]; int partner;
    if(procid % 2 == 1 ){ // odd procs send left, receive left
        partner = procid-1;
        MPI_Send(send, 10, MPI_DOUBLE, partner, 1, MPI_COMM_WORLD);
        MPI_Recv(recv, 10, MPI_DOUBLE, partner, 1, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
    }
    else{ // even procs receive right, send right
        partner = procid+1;
        MPI_Recv(recv, 10, MPI_DOUBLE, partner, 1, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
        MPI_Send(send, 10, MPI_DOUBLE, partner, 1, MPI_COMM_WORLD);
    }
}
{ // Sendrecv simplifies this pattern
    double send[10], recv[10]; int partner;
    partner = (procid % 2 == 1) ? procid-1 : procid+1;
    MPI_Sendrecv(send, 10, MPI_DOUBLE, partner, 1,
        recv, 10, MPI_DOUBLE, partner, 1,
        MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
Take Care: Pair exchange can hang

```c
{  double send[10], recv[10]; int partner;
  partner = (procid % 2 == 1) ? procid-1 : procid+1;
  MPI_Sendrecv(send, 10, MPI_DOUBLE, partner, 1,
                recv, 10, MPI_DOUBLE, partner, 1,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

- With 9 processors, logic is broken
- Proc 8 will wait to communicate with a partner that doesn’t exist
- Program never terminates
Gather

Every processor has computed columns

One processor (usually procid 0) needs to gather all of the data

Everyone calls MPI_Gather()
MPI_Gather Sample

Use of Gather

// Preamble for any code
MPI_Comm comm = MPI_COMM_WORLD;
int sendarray[100];
int procid, total_procs, *rbuf;
...
// Only proc 0 needs space for all
// data
if(procid == 0) {
    rbuf = malloc(total_procs*100*
    sizeof(int));
}

// Everyone calls gather
// proc 0 gets all data eventually
MPI_Gather(sendarray, 100, MPI_INT,
    rbuf, 100, MPI_INT,
    0, comm);

Equivalent Non-Gather Code

if(rank == 0){
    for(i=0; i<100; i++){
        rbuf[i] = sendarray[i];
    }
    for(i=1; i<total_procs; i++){  
        int *rloc = &rbuf[i*100];
        MPI_Recv(rloc, 100,
            MPI_INT, i,
            tag, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
    }
}
else{
    MPI_Send(sendarray, 100,
        MPI_INT, 0,
        tag, MPI_COMM_WORLD);
}