ACM40640 High Performance Comp Assignment 2

Ian Towey

04128591

June 29, 2018

Abstract

MPI code

Contents

0 Communication in a ring

File main_ring.c (see Appendix Listing 1 or accompanying file) demonstrates communication in a ring using MPI. Each processor sends a message to the its neighbour with rank one greater than its rank module the number of processors. The message sent along is the accumlated rank of the number of processors in the ring

e.g for a ring of size 5 starting at processor with rank 0

$$0 \xrightarrow[0]{\text{msg}=0} 1 \xrightarrow[0+1]{\text{msg}=1} 2 \xrightarrow[0+1+2]{\text{msg}=3} 3 \xrightarrow[0+1+2+3]{\text{msg}=6} 4 \xrightarrow[0+1+2+3+4]{\text{msg}=10} 0$$

e.g for a ring of size 6 starting at processor with rank 2

$$2 \xrightarrow[2]{\text{msg}=2} 3 \xrightarrow[2+3]{\text{msg}=5} 4 \xrightarrow[2+3+4]{\text{msg}=9} 5 \xrightarrow[2+3+4+5]{\text{msg}=14} 0 \xrightarrow[2+3+4+5+0]{\text{msg}=14} 1 \xrightarrow[2+3+4+5+0+1]{\text{msg}=15} 2$$

The origin processor is a commandline parameter, sample run of the program for a ring of size 6 starting at processor 2

```
$ cd /ichec/home/users/ph5xx10/PH504/assignment2/code/q1
$ module load dev intel/2015-u3
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ mpicc -g -o main main_ring.c -lm
$ mpirun -n 6 main 2

-->>2 sending '2' to 3

3 received: '2' from 2 3 sending '5' to 4

4 received: '5' from 3 4 sending '9' to 5

5 received: '9' from 4 5 sending '14' to 0

0 received: '14' from 5 0 sending '14' to 1

1 received: '14' from 0 1 sending '15' to 2

<<--2 received '15' from 1


Finished :: sum of ranks from 0 to 5 = 15
```

The program uses asynchronous communication between the processors to active this using MPI function MPI_Isend, MPI_Irecv, MPI_Wait

Listing 1: Asynchronous Communication Calls

```
1    fflush(stdout);
2    MPI_Isend(&value, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &
     s_request);
```

```
3     src = (num_processes == 1 ? my_rank : (my_rank == 0 ?
      num_processes - 1 : my_rank - 1));
4     MPI_Irecv(&value, 1, MPI_INT, src, tag,  MPI_COMM_WORLD, &
      r_request);
5     MPI_Wait(&r_request, &status);
6     printf("\n<<--%d received '%d' from %d\n\n", my_rank,value, src
      );
7     fflush(stdout);
```

1 Finding Determinent using Cramers Rule

File main_det.c calculates the determinent of a 5x5 matrix using cramers rule. 5 Threads are used to calculate the determinant where each thread calculates the determinant of a 4x4 matrix of the matrix A. The method then uses a recursive definition for the determinant of an n x n matrix, the minor expansion formula.

```
$ cd /ichec/home/users/ph5xx10/PH504/assignment2/code/q2
$ module load dev intel/2015-u3
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ mpicc -g -o main main_det.c -lm
$ mpirun -n 5 main

process 4, partial determinant -0.00386718112244897883
process 0, partial determinant -0.01440665145577631201
process 1, partial determinant -0.00361170218748875158
process 2, partial determinant -0.00415480520912068444
process 3, partial determinant -0.00409046542999916014
*******************************************************
****** Matrix  (n=5)                ***********
*******************************************************

1.000 -0.500 -0.333 -0.250 -0.200
-0.500 0.333 -0.250 -0.200 -0.167
-0.333 -0.250 0.200 -0.167 -0.143
-0.250 -0.200 -0.167 0.143 -0.125
-0.200 -0.167 -0.143 -0.125 0.111

*******************************************************
full determinant -0.03013080540483388525
*******************************************************
```

The processor figures out which 4x4 submatrix of the 5x5 matrix using its rank. The processor with rank 0, aggregates the minors of the determinent from the 5 processors using MPI_Reduce

Listing 2: MPI_Reduce

```
1     matrix *h = generate_matrix(N);
2     matrix *s = extract_submatrix(0,my_rank,h);
3     partialDeterminant = pow(-1,my_rank)*h->mat[0][my_rank]*
      determinant(s);
4     printf("process %d, partial determinant %.20f\n", my_rank,
      partialDeterminant);
5     fflush(stdout);
6     /* Reduce */
7     MPI_Reduce(&partialDeterminant,&fullDeterminant,1,MPI_DOUBLE,
      MPI_SUM,0,MPI_COMM_WORLD);
```

2 Finding Deadlock

Not attempted