Lab 3 Parallel Programming with MPI
Collective Communication (1)

1 Mục tiêu

- SV tìm hiểu và sử dụng các hàm collective communication trong thư viện MPI
- Một số hàmgiao tiếp nhóm SV cần tìm hiểu :
  - MPI_Bcast(), MPI_Scatter, MPI_Gather(), MPI_Barrier().
  - MPI_Scan(), MPI_Reduce(), MPI_Gatherv(), MPI_Scatterv() …
  - MPI_Reduce_scatter(), MPI_Allreduce …

-Link tham khảo:
https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines

2 Nội dung
2.1 Giới thiệu

- Sự giao tiếp giữa 1 nhóm process trong cùng communicator
- Mỗi process đều phải gọi hàm giao tiếp nhóm
- SV tìm hiểu xem mỗi hàm giao tiếp nhóm có chức năng gì và thực hiện các chương trình mẫu trong mục 2.2

2.2 Một số chương trình minh họa
2.2.1 Chương trình sử dụng MPI_Barrier():

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv){
      int i,rank,size;
      MPI_Init(&argc,&argv);
      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
      MPI_Comm_size(MPI_COMM_WORLD,&size);
      printf("Hello world, I have rank %d out of %d processes \n",rank,size);
      MPI_Finalize();
      return 0;
}
```

☺ *SV thử sử dụng hàm MPI_Barrier để điều khiển thứ tự xuất của các dòng hello world ! (với 20 processes)*

2.2.2 VD về tác dụng của hàm MPI_Barrier():

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    char filename[20];
    int rank, size;
    FILE *fp;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank < 2) { /* proc 0 and 1 only */
```

```
        sprintf(filename, "file_%d.out", rank);
        fp = fopen(filename, "w");
        fprintf(fp, "P%d: before Barrier\n", rank);
        fclose(fp);
    }
    MPI_Barrier(MPI_COMM_WORLD);
    if (rank < 2) { /* proc 0 and 1 only */
        sprintf(filename, "file_%d.out", (rank==0)?1:0 );
        fp = fopen(filename, "a");
        fprintf(fp, "P%d: after Barrier\n", rank);
        fclose(fp);
    }
    MPI_Finalize();
    return 0;
}
```

Sau khi chạy xong SV xem các file output **file_0.out** và **file_1.out** rồi cho nhận xét.

2.2.3 Chương trình sử dụng MPI_Bcast()

```
#include<mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]) {
      int rank;
      double param;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
      if(rank==5) {
            param=23.0;
      }
      MPI_Bcast(&param,1,MPI_DOUBLE,5,MPI_COMM_WORLD);
      printf("P:%d after broadcast parameter is %f\n",rank,param);
      MPI_Finalize();
}
```

2.2.4 Chương trình sử dụng MPI_Scatter()

```
#include <mpi.h>
#include <stdio.h>
int main( int argc, char* argv[] ) {
      int i;
      int rank, nproc;
      int isend[3], irecv;
      MPI_Init( &argc, &argv );
      MPI_Comm_size( MPI_COMM_WORLD, &nproc );
      MPI_Comm_rank( MPI_COMM_WORLD, &rank );
      if(rank == 0) {
            for(i=0; i<nproc; i++)
                  isend[i] = i+1;
      }
      MPI_Scatter(isend, 1, MPI_INT, &irecv, 1,MPI_INT, 0, MPI_COMM_WORLD);
      printf("irecv = %d\n", irecv);
      MPI_Finalize();
      return 0;
}
```

2.2.5 Chương trình sử dụng MPI_Gather()

```
#include <mpi.h>
int main( int argc, char* argv[] ) {
      int i;
      int rank, nproc;
      int isend, irecv[3];
```

```
        MPI_Init( &argc, &argv );
        MPI_Comm_size( MPI_COMM_WORLD, &nproc );
        MPI_Comm_rank( MPI_COMM_WORLD, &rank );
        isend = rank + 1;
        MPI_Gather( &isend, 1, MPI_INT, irecv, 1,MPI_INT, 0, MPI_COMM_WORLD);
        if(rank == 0) {
                 for(i=0; i<3; i++)
                          printf("irecv = %d\n", irecv[i]);
        }
        MPI_Finalize();
}
```

☺ *SV hãy viết các chương trình đơn giản như các ví dụ trên cho hàm MPI_Reduce !*

Cấu trúc của MPI_Reduce:

```
MPI_Reduce(void* send_data, void* recv_data, int count,
           MPI_Datatype datatype, MPI_Op op, int root,
           MPI_Comm communicator)
```

Một vài Operation của MPI_Reduce:

- `MPI_MAX` – Returns the maximum element.
- `MPI_MIN` – Returns the minimum element.
- `MPI_SUM` – Sums the elements.
- `MPI_PROD` – Multiplies all elements.
- `MPI_LAND` – Performs a logical "and" across the elements.
- `MPI_LOR` – Performs a logical "or" across the elements.
- `MPI_BAND` – Performs a bitwise "and" across the bits of the elements.
- `MPI_BOR` – Performs a bitwise "or" across the bits of the elements.
- `MPI_MAXLOC` – Returns the maximum value and the rank of the process that owns it.
- `MPI_MINLOC` – Returns the minimum value and the rank of the process that owns it.

3 Bài tập

SV hiện thực các bài tập theo hai cách:

1 Sử dụng các hàm point to point(MPI_Send, MPI_Recv).

2 Sử dụng các hàm giao tiếp nhóm.

3.1Viết chương trình nhân hai vector.

3.2 Viết chương trình nhân hai ma trận

SV sử dụng hàm **MPI_Wtime()** để đo thời gian thực hiện của 2 cách làm trên và cho nhận xét.