Lab 2:
Parallel Programming with MPI
A Distributed Data Structure

Jan Izquierdo
jan.izquierdo@
hpc1211

Laura LLorente
laura.llorente@
hpc1214

12/04/2024

1.1

1- Write the solution for 12 statements, identified as S1 . . . S12 in the source codes, that have some missing parameters in some calls to MPI primitives. Indicate the identifier of the statement and how you have filled it in.

par_data_struct.c
S1: MPI_Comm_rank(MPI_COMM_WORLD, &rank)
S2: MPI_Comm_size(MPI_COMM_WORLD, &size);
S3: MPI_Recv(xlocal[0], maxn , MPI_DOUBLE , rank - 1, 0, MPI_COMM_WORLD, &status);
S4: MPI_Send(xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1, MPI_COMM_WORLD);
S5: MPI_Recv(xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank + 1, 1, MPI_COMM_WORLD, &status);
S6: MPI_Reduce(&errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

par_data_struct_nonblocking.c
S7: MPI_Isend(&xlocal[maxn/P][0], maxn, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, &r[numreq++]);
S8: MPI_Irecv(&xlocal[0][0], maxn, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD, &r[numreq++]);
S9: MPI_Isend(&xlocal[1][0], maxn, MPI_DOUBLE, rank - 1, 1, MPI_COMM_WORLD, &r[numreq++]);
S10: MPI_Irecv(&xlocal[maxn/P + 1][0], maxn, MPI_DOUBLE, rank + 1, 1, MPI_COMM_WORLD, &r[numreq++]);
S11: MPI_Reduce(&errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

par_data_struct_sendreceive.c
S12: MPI_Sendrecv(xlocal[1], maxn, MPI_DOUBLE, prev_nbr, 1, xlocal[maxn/size+1], maxn, MPI_DOUBLE, next_nbr, 1, MPI_COMM_WORLD, &status);

1.2

1- Why do we need to use MPI Allreduce instead of MPI Reduce in all the codes in section 1.2?
To have the approximation wanted to be parallel (and so, all processes to have access to the reduced value) we need to have the solution of the reduced operation to be distributed to all processes and not only to the root process (as the MPI reduce does).

2- Alternatively, which other MPI call would be required if we had used MPI Reduce in all the codes in section 1.2?

MPI BcastWe use the following formula to balance the workload across the processes when they have to be distributed unevenly:
(rowsTotal / mpiSize) + (rowsTotal % mpiSize > mpiRank)

Each process receives a different amount of rows to handle, determined by the rank of the process. Because of the > mpiRank some processes will get +1 rows to handle and others will get +0, that way all the processes will get a similar load but no row is left unhandled.
We will need an extra operation in order to distribute the result from the root to the rest of the processes after using the MPI Reduce operation.

3- We have said that we have a halo so that some of the values in the matrix are read but not written during the computation. Inspect the code and explain which part of the code is responsible for defining such halo.

```
for (j=0; j<maxn; j++) {
xlocal[i_first-1][j] = -1;
xlocal[i_last+1][j] = -1;
}
```

In this part of the code, the halo region will be used to store reading data that does not need to be written in our computation (first and last rows of ‘xlocal’ filled with -1).

4- Explain with your own words how the load is balanced across the different processes when the number of rows is not evenly divided by P.

We use the following formula to balance the workload across the processes when they have to be distributed unevenly:
(rowsTotal / mpiSize) + (rowsTotal % mpiSize > mpiRank)

Each process receives a different amount of rows to handle, determined by the rank of the process. Because of the > *mpiRank* some processes will get +1 rows to handle and others will get +0, that way all the processes will get a similar load but no row is left unhandled.

5- Write the solution for the statements identified as S13 . . . S24 in the source codes, which have errors or some missing parameters in some calls to MPI primitives. Indicate the identifier of the statement and how you have filled it in:

Jacobi.c
S13: if (next_nbr >= size) next_nbr = MPI_PROC_NULL;
S14: MPI_Allreduce(&diffnorm, &globaldiffnorm, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

S15: MPI_Gather(xlocal[1], maxn /*send count(maxn?)*/, MPI_DOUBLE/*send_datatype*/,
        x, maxn * (maxn/size), MPI_DOUBLE/*recv_datatype*/,
        0/*root*/, MPI_COMM_WORLD);

Jacobi_nb.c
S16: MPI_Wait(&r[2], &status); //missing & in status
S17: MPI_Waitall(numreq, r, statuses); //numreq, MPI_Request variable, MPI_Status variable
S18: MPI_Wait(&r[3], &status);
S19: MPI_Iallreduce(&diffnorm, &globaldiffnorm, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, &r[0]);
S20: MPI_Igather(&diffnorm, maxn * (maxn/size),MPI_DOUBLE, x, maxn * (maxn/size), MPI_DOUBLE, 0, MPI_COMM_WORLD, &r[0]);

Jacobi_vr.c
S21: return (rowsTotal / mpiSize) + (rowsTotal % mpiSize > mpiRank);
S22: nrows = getRowCount(maxn, rank, size);
S23: MPI_Gather(&lcnt, 1, MPI_INT, recvcnts, 1, MPI_INT, 0, MPI_COMM_WORLD);
S24: MPI_Gatherv(xlocal[1], 1, MPI_DOUBLE, x, recvcnts, displs, MPI_DOUBLE, 0, MPI_COMM_WORLD);