Università della Svizzera italiana

Institute of Computing
CI

High-Performance Computing Lab **Institute of Computing**

Student: Paolo Deidda Discussed with: FULL NAME

Solution for Project 4

HPC Lab — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

Contents

1. Task: Ring maximum using MPI [10 Points]

I detrmine each process message destination and source using its rank.

The communication can be implemented using the standard MPI functions `MPI_Send` and `MPI_Recv`, which take the following arguments:

- buffer address (to store sent or received data)
- number of elements (to send or receive)
- MPI datatype (of the elements to send or receive)
- destination (or source) rank (of the process to send to or receive from)
- message tag (to identify the message with an id)
- communicator (to identify the group of processes involved in the communication, useless in this case)

When using `MPI_Send` and `MPI_Recv` separately, the ranks must be split into two groups like even and odd ranks, otherwise all processes may block by attempting to send or receive at the same time.

Alternatively, we can use the `MPI_Sendrecv` function, which takes care of both sending and receiving in a single call, avoiding deadlocks.

```
21      MPI_Sendrecv(
22        &send_val, 1, MPI_INT, next, 0,
23        &recv_val, 1, MPI_INT, prev, 0,
24        MPI_COMM_WORLD, MPI_STATUS_IGNORE
25      );
```

Listing 1: Implemetation from file ring_sum.c

Once launched the job we can see the folloring output with `-ntasks=4` on the left and `-ntasks=8` on the right.

```
3  Process 0: Sum = 6
4  Process 2: Sum = 6
5  Process 1: Sum = 6
6  Process 3: Sum = 6
```

Listing 2: Output from file ring_57313.out

```
3  Process 7: Sum = 28
4  Process 5: Sum = 28
5  Process 6: Sum = 28
6  Process 3: Sum = 28
7  Process 0: Sum = 28
8  Process 1: Sum = 28
9  Process 2: Sum = 28
10 Process 4: Sum = 28
```

Listing 3: Output from file ring_57315.out

2. Task: Ghost cells exchange between neighboring processes [15 Points]

We first create a 4×4 Cartesian communicator with periodic boundaries in both directions. Using `MPI_Cart_shift` we retrieve the rank of the north, south, east and west neighbors so that the first and last ranks remain connected. The column ghost layer is described with a derived datatype built with `MPI_Type_vector`; it spans one element every `DOMAINSIZE` entries and covers the whole interior height of the tile.

With these ingredients we perform four `MPI_Sendrecv` calls, one per direction, each time sending the interior boundary (rows or columns that exclude the corner points) and receiving directly into the ghost layer. Because every transfer is expressed as a combined send/receive, no explicit ordering or buffering is needed to avoid deadlocks.

```
80      // TODO: set the dimensions of the processor grid and periodic boundaries in both dimensions
81      dims[0] = 4;
82      dims[1] = 4;
83      periods[0] = 1; // indica se periodico, 0 no, 1 si'
84      periods[1] = 1;
85
86      // TODO: Create a Cartesian communicator (4*4) with periodic boundaries (we do not allow
87      // the reordering of ranks) and use it to find your neighboring
88      // ranks in all dimensions in a cyclic manner.
89      MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm_cart);
90
91      // TODO: find your top/bottom/left/right neighbor using the new communicator, see
        ↪ MPI_Cart_shift()
92      // rank_top, rank_bottom
93      // rank_left, rank_right
94      MPI_Cart_shift(comm_cart, 0, 1, &rank_top, &rank_bottom);
95      MPI_Cart_shift(comm_cart, 1, 1, &rank_left, &rank_right);
96
97      //  TODO: create derived datatype data_ghost, create a datatype for sending the column, see
        ↪ MPI_Type_vector() and MPI_Type_commit()
98      // data_ghost
99      MPI_Type_vector(SUBDOMAIN, 1, DOMAINSIZE, MPI_DOUBLE, &data_ghost);
100     MPI_Type_commit(&data_ghost);
101
102     //  TODO: ghost cell exchange with the neighbouring cells in all directions
103     //  use MPI_Irecv(), MPI_Send(), MPI_Wait() or other viable alternatives
104
105     //  to the top
106     MPI_Sendrecv(&data[1 * DOMAINSIZE + 1], SUBDOMAIN, MPI_DOUBLE, rank_top, 0,
107                  &data[0 * DOMAINSIZE + 1], SUBDOMAIN, MPI_DOUBLE, rank_top, 0,
108                  comm_cart, &status);
109
110     //  to the bottom
111     MPI_Sendrecv(&data[(DOMAINSIZE - 2) * DOMAINSIZE + 1], SUBDOMAIN, MPI_DOUBLE,
112                  rank_bottom, 1, &data[(DOMAINSIZE - 1) * DOMAINSIZE + 1],
113                  SUBDOMAIN, MPI_DOUBLE, rank_bottom, 1, comm_cart, &status);
114
115     //  to the left
116     MPI_Sendrecv(&data[1 * DOMAINSIZE + 1], 1, data_ghost, rank_left, 2,
117                  &data[1 * DOMAINSIZE + 0], 1, data_ghost, rank_left, 2,
118                  comm_cart, &status);
119
120     //  to the right
121     MPI_Sendrecv(&data[1 * DOMAINSIZE + (DOMAINSIZE - 2)], 1, data_ghost,
122                  rank_right, 3, &data[1 * DOMAINSIZE + (DOMAINSIZE - 1)], 1,
123                  data_ghost, rank_right, 3, comm_cart, &status);
124
125
126     if (rank==9) {
127         printf("data of rank 9 after communication\n");
```

```
128         for (j=0; j<DOMAINSIZE; j++) {
129             for (i=0; i<DOMAINSIZE; i++) {
130                 printf("%4.1f ", data[i+j*DOMAINSIZE]);
131             }
132             printf("\n");
133         }
134     }
135 
136     // Free MPI resources (e.g., types and communicators)
137     // TODO
138     MPI_Type_free(&data_ghost);
139     MPI_Comm_free(&comm_cart);
```

Listing 4: Ghost layer exchange, ghost.c

3. Task: Parallelizing the Mandelbrot set using MPI [20 Points]

4. Task: Parallel matrix-vector multiplication and the power method [40 Points]

5. Task: Quality of the Report [15 Points]