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.

```
    MPI_Sendrecv(
      &send_val, 1, MPI_INT, next, 0,
      &recv_val, 1, MPI_INT, prev, 0,
      MPI_COMM_WORLD, MPI_STATUS_IGNORE
    );
```

Listing 1: Implemetation from file ring_sum.c

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.

```
    // TODO: set the dimensions of the processor grid and periodic boundaries in both dimensions
    dims[0] = 4;
    dims[1] = 4;
    periods[0] = 1; // indica se periodico, 0 no, 1 si'
    periods[1] = 1;

    // TODO: Create a Cartesian communicator (4*4) with periodic boundaries (we do not allow
    // the reordering of ranks) and use it to find your neighboring
    // ranks in all dimensions in a cyclic manner.
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm_cart);

    // TODO: find your top/bottom/left/right neighbor using the new communicator, see
    ↪ MPI_Cart_shift()
    // rank_top, rank_bottom
    // rank_left, rank_right
    MPI_Cart_shift(comm_cart, 0, 1, &rank_top, &rank_bottom);
    MPI_Cart_shift(comm_cart, 1, 1, &rank_left, &rank_right);

```

```c
    //  TODO: create derived datatype data_ghost, create a datatype for sending the column, see MPI_Type_vector() and MPI_Type_commit()
    // data_ghost
    MPI_Type_vector(SUBDOMAIN, 1, DOMAINSIZE, MPI_DOUBLE, &data_ghost);
    MPI_Type_commit(&data_ghost);

    //  TODO: ghost cell exchange with the neighbouring cells in all directions
    //  use MPI_Irecv(), MPI_Send(), MPI_Wait() or other viable alternatives

    //  to the top
    MPI_Sendrecv(&data[1 * DOMAINSIZE + 1], SUBDOMAIN, MPI_DOUBLE, rank_top, 0,
                 &data[0 * DOMAINSIZE + 1], SUBDOMAIN, MPI_DOUBLE, rank_top, 0,
                 comm_cart, &status);

    //  to the bottom
    MPI_Sendrecv(&data[(DOMAINSIZE - 2) * DOMAINSIZE + 1], SUBDOMAIN, MPI_DOUBLE,
                 rank_bottom, 1, &data[(DOMAINSIZE - 1) * DOMAINSIZE + 1],
                 SUBDOMAIN, MPI_DOUBLE, rank_bottom, 1, comm_cart, &status);

    //  to the left
    MPI_Sendrecv(&data[1 * DOMAINSIZE + 1], 1, data_ghost, rank_left, 2,
                 &data[1 * DOMAINSIZE + 0], 1, data_ghost, rank_left, 2,
                 comm_cart, &status);

    //  to the right
    MPI_Sendrecv(&data[1 * DOMAINSIZE + (DOMAINSIZE - 2)], 1, data_ghost,
                 rank_right, 3, &data[1 * DOMAINSIZE + (DOMAINSIZE - 1)], 1,
                 data_ghost, rank_right, 3, comm_cart, &status);


    if (rank==9) {
        printf("data of rank 9 after communication\n");
        for (j=0; j<DOMAINSIZE; j++) {
            for (i=0; i<DOMAINSIZE; i++) {
                printf("%4.1f ", data[i+j*DOMAINSIZE]);
            }
            printf("\n");
        }
    }

    // Free MPI resources (e.g., types and communicators)
    // TODO
    MPI_Type_free(&data_ghost);
    MPI_Comm_free(&comm_cart);
```

Listing 2: 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]