High Performance Computing for Science and Engineering II.

Exercise Set 2

Florian Mahlknecht

2020-03-18

1 Wave Equation

Listings 1 and 2 show the crucial parts of the implementation, for details please refer to the attached source code.

```
int nums[3] = {0,0,0};
int periodic[3] = {false, false, false};
MPI_Dims_create(size, 3, nums); // split the nodes automatically

MPI_Cart_create(MPI_COMM_WORLD, 3, nums, periodic, true, &cart_comm);
MPI_Comm_rank(cart_comm, &rank);
if (rank == 0) {
  std::cout << "Grid: " << nums[0] << ", " << nums[1] << ", " << nums[2] << std::endl;
}

MPI_Cart_shift(cart_comm, 0, 1, &rank_minus[0], &rank_plus[0]);
MPI_Cart_shift(cart_comm, 1, 1, &rank_minus[1], &rank_plus[1]);
MPI_Cart_shift(cart_comm, 2, 1, &rank_minus[2], &rank_plus[2]);
MPI_Cart_coords(cart_comm, rank, 3, coords);
```

Listing 1: Cartesian communicator creation

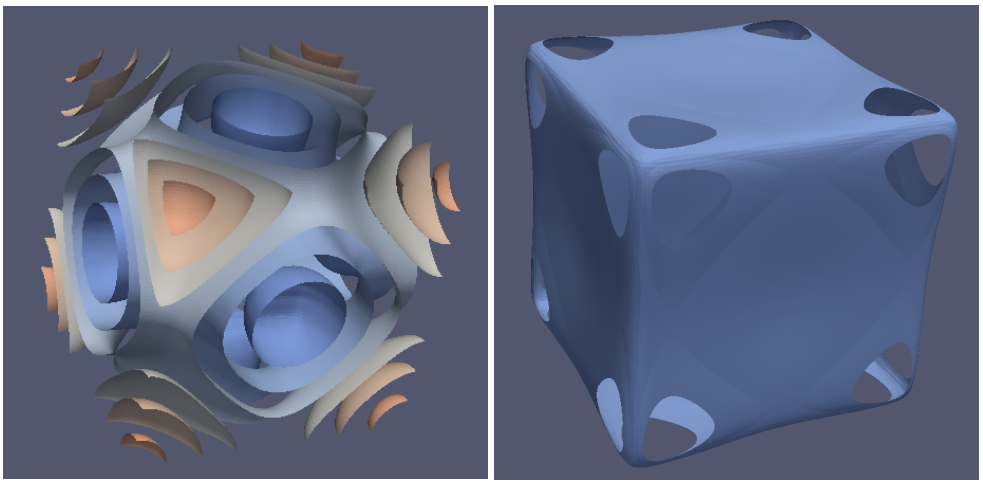
fmahlknecht@student.ethz.ch
19-945-351

```
MPI_Datatype SEND_FACE_PLUS [3];
MPI_Datatype SEND_FACE_MINUS[3];
MPI_Datatype RECV_FACE_PLUS [3];
MPI_Datatype RECV_FACE_MINUS[3];

// same for all
int sizes[3] = {N+2, N+2, N+2};
int subsizes[3][3] = {{1, N, N}, {N, 1, N}, {N, N, 1}};

/** SEND **/
// same for all faces
int sendStartsMinus[3] = {1,1,1};
int sendStartsPlus[3][3] = {{N, 1, 1}, {1, N, 1}, {1, 1, N}};

MPI_Type_create_subarray(3, sizes, subsizes[0], sendStartsMinus, /* ... */
/* ... */

/** RECEIVE **/
int recvStartsMinus[3][3] = {{0, 1, 1}, {1, 0, 1}, {1, 1, 0}};
int recvStartsPlus[3][3] = {{N + 1, 1, 1}, {1, N + 1, 1}, {1, 1, N + 1}};

/* ... */
```

Listing 2: Data type definition

The checksum has reliably been found to be `Checksum = 59235.8` for periodic boundary conditions and the initial conditions given in the problem statement, i.e. including $(z - 0.5)^2$ in the radius.

As discussed during the recitation, for the (updated) Dirichlet boundary conditions it is sufficient to extend the for loop in the initialization function, to apply also to the ghost cells. By setting periodic to false, as shown in listing 1, the communicator automatically returns `MPI_PROC_NULL`, which lets the respective cells unaffected and constant. The results can be visualized in Paraview, see fig. 1.



(a) Periodic (b) Dirichlet

Figure 1: Frame 21 visualized in paraview, for different boundary conditions

Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351

2 Cannon's algorithm

In listing 3 the implementation for the automatic topology detection. Note that left and right this time are on the 2nd axis, while up and down along the first.

```
MPI_Comm cart_comm;
int nums[2] = {p,p};
int periodic[2] = {true, true};
MPI_Cart_create(MPI_COMM_WORLD, 2, nums, periodic, true, &cart_comm);
MPI_Comm_rank(cart_comm, &rank);

int up, down, left, right;
MPI_Cart_shift(cart_comm, 1, 1, &left, &right);
MPI_Cart_shift(cart_comm, 0, 1, &up, &down);
int coords[2];
MPI_Cart_coords(cart_comm, rank, 2, coords);

int ranky = coords[0];
int rankx = coords[1];
```

Listing 3: Cannon's algorithm rank topology

For the submatrix a simple contiguous type was sufficient, as shown in listing 4

```
MPI_Datatype SUBMATRIX;
MPI_Type_contiguous(n*n, MPI_DOUBLE, &SUBMATRIX);
MPI_Type_commit(&SUBMATRIX);
/* ... */
MPI_Irecv(tmpA, 1, SUBMATRIX, right, 0, cart_comm, &request[0]);
MPI_Irecv(tmpB, 1, SUBMATRIX, down, 1, cart_comm, &request[1]);
/* ... */
```

Listing 4: Contiguous submatrix type

With this construct we get the same results as before on euler:

```
Running Matrix-Matrix Multiplication...
Verifying Result... Passed!
Execution time: 0.026s
GFlop/s: 81.2953
```


fmahlknecht@student.ethz.ch
19-945-351

3 Custom structs in MPI

Finally, listing 5 shows how the custom struct is implemented as a datatype in MPI.

```
struct particle
{
  int id;
  double x[3];
  bool state;
  double gamma;
};

/* ... */
particle p;

MPI_Aint p_lb, p_id, p_x, p_state, p_gamma, p_ub;
MPI_Get_address(&p, &p_lb);
MPI_Get_address(&p.id, &p_id);
MPI_Get_address(&p.x, &p_x);
MPI_Get_address(&p.state, &p_state);
MPI_Get_address(&p.gamma, &p_gamma);
MPI_Get_address(&p + 1, &p_ub);

MPI_Datatype MPI_PARTICLE;
int blocklens[] = {0, 1, 3, 1, 1, 0};
MPI_Aint offsets[] = {0, p_id - p_lb, p_x - p_lb,
                      p_state - p_lb, p_gamma - p_lb, p_ub - p_lb};
MPI_Datatype types[] = {MPI_LB, MPI_INT, MPI_DOUBLE,
                        MPI_CXX_BOOL, MPI_DOUBLE, MPI_UB};
MPI_Type_create_struct(6, blocklens, offsets, types, &MPI_PARTICLE);
MPI_Type_commit(&MPI_PARTICLE);

/* ... */
```

Listing 5: Custom struct datatype

With this construct we have a portable code, which indeed successfully sends the custom struct, giving the following output:

```
1
3.14159265
2.71828183
1.61803399
0
0.57721566
```


Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351