Manual of the Parallel Geophysical Model Coupling Library (PGMCL)

Mathieu Dutour Sikirić

July 28, 2016

This fortran library is concerned with coupling between different geophysical models.

It helps in the matters of

1. Interpolation
2. Sending data between MPI parallelized models.

1 Introduction

Geophysical models were originally developped each for every domain. Possible examples of this are

1. Atmospheric model for forecasting wind, pressure, temperature, etc.
2. Circulation models for forecasting oceanic currents, free surface elevation, temperature, salinity, etc.
3. Wave models for forecasting wave height, wave direction, etc.
4. Ecological models for forecasting concentrations in carbon, oxygen, phosphate, nitrate, plankton, etc.
5. Sediment models for forecasting how the soil evolve in time.

The point is that all those models depend from each other, e.g. surface air pressure and wind are quite important for a circulation model, while a meteorological model can use the sea surface temperature for its forecasts.

Thus, we have the need for coupling models in order to get better forecasts. The library that we consider, the PGMCL actually helps doing such coupling. It should be completely clear that this requires a significant programming effort and that the library only makes it easier.

2 Assumptions and description of the library

The library assumes that the parallelization of the model is done by Message Passing Interface (MPI) and that the parallelization is geographic, i.e. some geographical region go to one node, some other to another node.

The model involved can have wildly different grids. They do not have to be the same, they do not have to be both of the same type, i.e. one can be unstructured triangular and the other can be a finite difference grid.

However, we assume that the interpolation is done linearly in order to use sparse matrices. The model allows to compute the matrices with first order linear interpolation but other interpolation matrices can be used also.

3 Repartition of the computing nodes

What we do is that we split the set of processors into several groups. Each model has its own group.

In term of MPI technology, we have a call to `MPI_COMM_SPLIT` that split the communicator in input (which is typically MPI_COMM_WORLD) into several subcommunicator. This splitting is done by the user. The programmer then simply has to replace his communicator by the new subcommunicator.

4 Indexes of nodes

The command for allocating the nodes is

```
SetComputationalNodes(ArrLocal, MyNbCompNode, eDestColor)
```

This is necessary because the node indexing in the subcommunicator has a priori no relation to the node ordering in the main communicator (i.e. MPI_COMM_WORLD).

5 matrix belonging

The user needs to create on each model a matrix of size $(NP, Nproc)$ with NP the number of nodes used by the model and $Nproc$ the number of processor of the model. The matrix must be of size `T_node_partition`.

If exchanging is between model A and B then both models need to know the data. And the command for that are:

```
CALL M2M_send_node_partition(ArrLocal, OCNid, MatrixBelongingWAV)
CALL M2M_recv_node_partition(ArrLocal, OCNid, MatrixBelongingOCN)
```

That is assuming we couple the ocean with the waves.

6 Interpolation arrays

The command for creating interpolation arrays is typically

```
 CALL SAVE_CreateInterpolationSparseMatrix_Parall(                  &
&    FileSave_OCNtoWAV_rho, mMat_OCNtoWAV_rho, DoNearest,           &
&    eGrid_ocn_rho, eGrid_wav,                                      &
&    WAV_COMM_WORLD, MatrixBelongingWAV)
```

This interpolates the coefficient of the matrix from the wave to the ocean model. The matrix is saved in a netcdf file that can be read in future runs automatically. The `eGrid_ocn_rho` and `eGrid_wav` are of type `T_grid` and contains the grids of the model.

The commands for sending and receiving the sparse matrices of interpolation arrays are:

```
CALL M2M_send_sparseMatrix(ArrLocal, OCNid, mMat_OCNtoWAV_rho)
CALL M2M_recv_sparseMatrix(ArrLocal, OCNid, mMat_WAVtoOCN_rho)
```

7 Input/Output arrays

Once we have the interpolation matrices and the matrix belonging we can easily compute the input/output arrays that allows to send and receive data.

This is done in two steps. First the creation of the array:

```
      CALL MPI_INTERP_GetSystemOutputSide(ArrLocal, OCNid, WAVid,       &
     &    MatrixBelongingOCN_rho, MatrixBelongingWAV,                   &
     &    mMat_OCNtoWAV_rho, TheArr_OCNtoWAV_rho)
```

Then the creation of the asynchrone array.

```
CALL MPI_INTERP_GetAsyncOutput_r8(TheArr_OCNtoWAV_rho,             &
&    3, TheAsync_OCNtoWAV_uvz)
```

3 is the number of variables read from the model.

Symmetrically for the inout, this is done that way:

```
CALL MPI_INTERP_GetSystemInputSide(ArrLocal, WAVid, OCNid,         &
&    MatrixBelongingWAV, MatrixBelongingOCN_rho,                    &
&    mMat_WAVtoOCN_rho, TheArr_WAVtoOCN_rho)
```

followed by

```
  CALL MPI_INTERP_GetAsyncInput_r8(TheArr_WAVtoOCN_rho,              &
  &    19, TheAsync_WAVtoOCN_stat)
```

8 The exchanges themselves

The exchanges are very similar to MPI calls except they are simpler.

So for the receiving is would go that way:

```
CALL MPI_INTERP_ARECV_3D_r8(TheAsync_OCNtoWAV_uvz, tag, A_wav_uvz)
```

with tag a MPI tag, in other word an arbitrary integer.

The assignation of variable could then be done that way:

```
DO IP=1,MNP
  Ucurr(IP) = A_wav_uvz(1,IP)
  Vcurr(IP) = A_wav_uvz(2,IP)
  Zeta (IP) = A_wav_uvz(3,IP)
END DO
```

For the sending, it would go that way:

```
DO IP=1,MNP
  A_wav_stat(1,IP) = Hwave(IP)
  .
  .
  .
  A_wav_stat(19,IP) = last interesting quantity
END DO
```

and the sending is done that way:

```
CALL MPI_INTERP_ASEND_3D_r8(TheAsync_WAVtoOCN_stat, 212, A_wav_stat)
```

That's it!

There are of course many details that I skipped. But the idea is that simple. If you have trouble, please send an email to `mathieu.dutour@gmail.com`