BMI for XBeach

A short description of the implementation of BMI in parallel XBeach, and an use case. Code is available in https://svn.oss.deltares.nl/repos/xbeach/branches/willem-2016/wtrunk/src and https://svn.oss.deltares.nl/repos/xbeach/branches/willem-2016/wtrunk/bmi

Willem Vermin, May 2017

Introduction

In this document we describe an implementation of BMI for parallel XBeach and an use case using a Python main program.

A serial interface for BMI was already present: `xbeach_bmi.F90`. This interface was extended for parallel XBeach.

The use case, a Python main program `xbeachbmi.py` connecting two XBeach instances, is able to service serial XBeach as well as parallel XBeach. In both cases MPI is used for communication.

BMI for parallel XBeach

Parallel XBeach uses MPI for communication. For performance reasons, the processes are split into one output process, and one or more computing processes. The output process is only active when output is requested. In MPI terms, the rank of the output process is zero. The compute processes have ranks 1 and higher, depending on how XBeach has been started. Process 0 waits for a `MPI_Send` from process 1. Upon receiving this message, the data to be output is collected on process 0, whereupon the output is created and, in parallel, the compute processes continue computing. Input for the program is done exclusively by process 1. In general, this input is distributed among the compute processes. XBeach uses two communicators: one for all processes, and one for the compute processes.

Of course, in the serial version of XBeach, all output, input and computing is done by one process, and there is no MPI involved.

Implementation of xbeachbmi.py

An use case was constructed: one XBeach handling the total domain and another XBeach handling a sub domain.
The program expects to be started using `mpiexec`, and expects to parameters: `n_inner` and `n_outer`, respectively the number of processes to be used for handling the inner and outer case. Two helper scripts are available:

- runit: runs the simulation (assuming OpenMPI as MPI implementation. Trivial changes are needed to use MPICH)
- wrap: a simple wrapper for xbeachbmi.py

The interface to MPI is delivered by `mpi4py`.

A few definitions:

- inner: the subdomain
- outer: the total domain

The following communicators are used:

- 0: MPI_COMM_WORLD as a starting point to create other communicators
- 1: for the inner XBeach
- 2: for the compute processes of inner XBeach
- 3: for the outer XBeach
- 4: for the compute processes of outer XBeach
- 5: for the 1st processes of the compute communicators for inner and outer

Remarks:

- when using serial XBeach for inner and/or outer: 1,2 respectively 3,4 are not used.
- 5 is always used for communication between inner and outer

The decision to use serial or parallel XBeach depends on the value of `n_inner` and `n_outer`. When this value equals 1, serial XBeach is assumed, otherwise parallel XBeach.

After creating the communicators, the program communicates the values of the communicators to the XBeach instances, where after initialize is called.

Then some useful information is extracted from XBeach such as `x` and `y`. Some preparatory work is done: for example the computing of the kd-trees necessary for interpolation later on.

In the model, it is arranged that communication from outer to inner results in an update of the border of inner, and that communication from inner to outer results in an update of the whole inner area in outer. Because the grids of inner and outer are not the same, an interpolation is done using kd-trees.

Special attention was required:

- most entities are matrices with dimensions `(ny+1, nx+1)` (reversed from the Fortran convention in XBeach)
- 3-d entities. These are handled as a number of matrices of order `(ny+1, nx+1)`.
- 2-d entities, dimensioned as `(ny+1,2)`. After `get_var`, these are expanded to `(ny+1,nx+1)` with the extra elements equal to the values in the last column. Before `set_var`, these entities are chopped again to `(ny+1,2)`
- 1-d entities, dimensioned as `(ny+1)`. Basically, these get the same treatment as the previous.
- some entities come in pairs, for example `u,v`. After `get_var`, these are converted as in `postprocess.F90`. Before `set_var`, these are converted with a negative `alfaz`.
- the timings of the inner and outer process are different. Before exchange of data, it is tried to get the timings as equal as possible by introducing extra update steps for the lagging instance.

Implementation of BMI for parallel XBeach

The following BMI functions are defined in `xbeach_bmi.F90`:
`initialize, finalize, update, get_var_type, get_var_rank, get_var_shape, get_var, set_var, get_current_time, get_start_time, get_end_time`.

Normally, one starts with `initialize` and continues with the other functions. In the parallel XBeach case, we need to tell XBeach what MPI communicator to use. Therefore, `set_var` has been extended: when used with a special name `mpicomm`, XBeach will use the accompanying value as the MPI communicator for all it's processes. After initialize, XBeach will create a communicator for the compute processes, containing processes 1 and higher.

Process 1 is responsible for the actual communication of the values in `get_var` and `set_var`.

The following functions deserved special attention:

- `get_var`: originally, XBeach contained only code to collect distributed entities on process 0. Since this process is not involved in computations this process is not usable for BMI. Therefore XBeach has been made aware of the fact that collection can also be done to process 1. (Interested readers can search for the usage of the logical `w_only`. If `.true.`, collection has to be done on process 1 (process 0 in the compute communicator)).
- `set_var`: since XBeach was already designed to distribute entities from process 1, this posed no problem. Extra code was inserted to handle the special variable `mpicomm`.

Typical usage would be:
- `set_var('mpicomm',communicator)`: collective over all processes
- `initialize('params.txt')`: collective over all processes
  - process 0 will not return
- calls to `get_var, set_var, update,` and so on: collective over the compute processes
- `finalize()`: collective over the compute processes

Subroutine `xmpi_initialize (in xmpi.F90)` has been adapted: if MPI is already initialized, MPI_COMM_WORLD will not be used as communicator.

Extra parameters were introduced:
- `inner_lowx = 1`: cells with `ix == 1` will not be computed, it is assumed that these values are provided by some other means (BMI, for example)
- likewise for `inner_lowy, inner_highx` and `inner_highy`
- `inner = 1: inner_lowx, inner_highx, inner_lowy, inner_highy` are set to 1

For parallel Xbeach, these parameters are only valid for the processes managing the border areas. To keep the program readable, convenience variables `manage_lowx` etc. are determined. See for example usage: `boundaryconditions.F90`. The variables are defined in `spaceparams.F90`.