Revisions to MPAS block decomposition routines

May 2, 2012

Contents

Chapter 1

Introduction

Previously several projects have been worked on with the end goal of supporting multiple blocks. These tasks are described below.

1. Update/extend the fundamental derived types in mpas_grid_types.F. In order for other parts of the infrastructure to handle multiple blocks per task in a clean way, we'll need to be able to pass a head pointer to a field into a routine, and have that routine loop through all blocks for that field, with information about which cells/edges/vertices in that field need to be communicated.

2. Decide on a new MPAS I/O abstraction layer, which will provide a high-level interface to the PIO layer for the rest of MPAS. This layer should work with blocks of fields, and make it possible to define an arbitrary set of I/O streams at run-time.

3. Add a new module to parse a run-time I/O configuration file that will describe which fields are read or written to each of the I/O streams that a user requests via the file. This module will make calls to the new MPAS I/O layer to register the requested fields for I/O in the requested streams.

4. Update the mpas_dmpar module to support communication operations on multiple blocks per task. This will likely involve revising the internal data structures used to define communication of cells between tasks, and also require revisions to the public interface routines themselves.

5. Modify the block_decomp module to enable a task to get a list of cells in more than one block that it is to be the owner of. Implemented in the simplest way, there could simply be a namelist option to specify how many blocks each task should own, and the block_decomp module could look for a graph.info.part.n file, with n=num_blocks_per_task*num_tasks, and assign blocks k, 2k, 3k, ..., num_blocks_per_task*k to task k.

This document related to tying all of these projects together, and allowing MPAS to run with multiple blocks per MPI process.

As an example of what running with multiple blocks means, currently MPAS is allowed to run with 1 block per process. This means prior to running MPAS a decomposition of cells needs to be determined. This decomposition tells each MPI process which cells it owns. So, under the current framework the number of decompositions have to be equal to the number of MPI tasks used to run the job.

After this project is completed, a user should be able to specify the use of a decomposition that does not have an equal number of blocks as the number of processors the job is run with. Typically this would be used to allow more than one block per processor, but could theoretically be used to run having some processors not have any blocks.

The goal with this project is to allow exploration of OpenMP directives, and test performance using different block sizes. Allowing multiple blocks per processor could increase cache reuse and allow a performance increase.

This work is currently being performed under branches/omp_blocks/multiple_blocks and all source code changes can be examined there.

Chapter 2

Requirements

There are significant changes to MPAS' framework that have to be made in order to support multiple blocks. A list of requirements that determine these changes are listed below.

- Block creation must be robust, and handle an arbitrary number of blocks per processor.
- Blocks should be created using the derived data types created in an earlier project, utilizing the field data types.
- Block creation routines should be created with an arbitrary number of halos assumed, although the default is currently two.
- All block communication routines should be able to handle shared memory copies.
- Exchange list creation should be performed at the block/field level.
- A new module should be setup to handle the management of blocks.

Chapter 3

Design

Only a small amount of design has been completed thus far. So, all information in this section should be regarded as a work in progress for now.

The current prototyping efforts have determined the following routines which require changes to their infrastructure.

```
mpas_dmpar_alltoall_field
mpas_dmpar_exch_halo_field
mpas_dmpar_get_owner_list
mpas_input_state_for_domain
```

Within the allToAll and exch_halo routines, loops over multiple blocks need to be added where they are not currently in place. Also, shared memory copies within local blocks need to be added.

The whole structure of mpas_dmpar_get_owner_list has to change in order to support multiple blocks. This routine currently builds the exchange lists for a single block and has global communications. In order to handle the creation of exchange lists with multiple blocks, this routine will be rewritten to use the field data types. This way each field will be a linked list, consisting of the fields from each block. The downside to this change, is that now the routine requires the setup of basic block types (really just the blockID number), and block fields (like indexToCellID) prior to calling this. However, this keeps the data types used in the creation of these routines in line with how the rest of MPAS deals with fields.

mpas_input_state_for_domain also has to be modified in order to setup the fields and blocks as required for mpas_dmpar_get_owner_list, and to create multiple blocks. Currently it is writing under the assumption that only one block per process exists, and has that single block hard-coded as the only one that gets created.

In addition to the currently in place changes that need to be made, several routines need to be added. Within the current prototyping work, a routine has been created to determine the all cell indices for a block, including all halo cells. It has been written under the assumption that there could be an arbitrary number of halos. This routine is called

```
mpas_get_halo_cells_and_exchange_lists(dminfo, nHalos,
                indexToCellID_0Halo, nEdgesOnCell_0Halo,
                cellsOnCell_0Halo, indexToCellID_nHalos,
                nEdgesOnCell_nHalos, cellsOnCell_nHalos)
```

At the time of writing this document, this routine can be seen within the src/framework/mpas_block_decomp.F module, but this may change when a new module is created.

One general change that has to be made in order to support these field data types being used in the input stage of MPAS is the addition of a deallocate field routine. This routine would be used to deallocate all fields within a field linked list. It is used when a field is created that's not a member of a block, so calling mpas_deallocated_block would not destroy all the memory associated with the field.

In addition to the changes listed here, routines still need to be determined to create a list of vertices and edges for a block and all it's halos, as well as their respective exchange lists. After the list of cells, vertices, and edges are complete for a block the IO read fields can be called to setup the fields within each block. Finally, the global indices within a block need to be modified to be local indices.

As mentioned in the requirements section, a large portion of these changes might be pushed into a new module. This new module would be written to handle the management of blocks. The proposed name would be mpas_block_manager.

Chapter 4

Testing

The end goal from this project is to provide a framework that allows bit-for-bit reproduction of data using an arbitrary combination of blocks and processor numbers.

Using this goal to define a testing strategy implies a good test would be exploring bit-for-bit reproduction of output data using the three following simulations:

- Current trunk simulation run with 8 processors and 8 blocks
- Finished branch simulation run with 8 processors and 8 blocks
- Finished branch simulation run with 1 processor and 8 blocks
- Finished branch simulation run with 2 processors and 8 blocks

If all of these simulations produce bit-for-bit output then the project would be deemed as completed.