Redesign of MPAS data structures

December 20, 2011

Contents

1 Introduction **2**

2 Requirements **3**

3 Design **4**

Chapter 1

Introduction

In order to support multiple blocks of cells per MPI task, there are a number of development issues that need to be addressed

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 concerns the first item, namely, the extensions to the derived data types that will be necessary for supporting multiple blocks per task in other infrastructure modules.

Chapter 2

Requirements

The changes to the derived data types used throughout the MPAS infrastructure and cores should enable I/O and communication routines to elegantly handle multiple blocks per MPI task.

- Routines must be able to traverse the list of blocks for any field that is owned by a task without having to explicitly dereference that field by name; this ensures that infrastructure routines can remain generic, in the sense that they never contain hard-wired references to fields.
- A block for a field must be able to access the parallel information (halo communication lists, as well as MPI communicator) that it is associated with so that such information is never explicitly passed with the field to infrastructure subroutines. This requirement will simplify the argument lists for infrastructure routines, and eliminate the possibility that a user might pass mismatched or invalid communication information for a field to an infrastructure routine.

Chapter 3

Design

We propose to extend the existing data structures in MPAS with additional pointers between fields of the same type, and also with pointers within fields to the appropriate communication lists for that field. The proposed changes are highlighted in the field type definition below; other type definitions are given for reference, and the figure below illustrates graphically the hierarchy of DDTs.

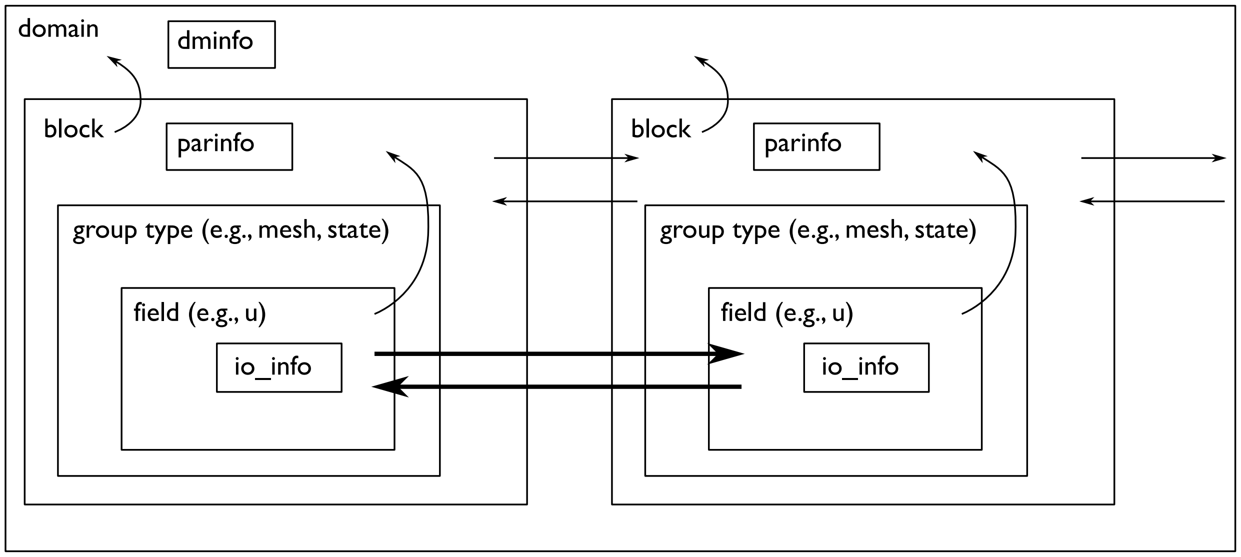

Figure 3.1: An illustration of the current DDT structure: existing back-pointers and previous/next pointers are drawn in light arrows; new pointers to be added between field blocks are drawn heavy arrows.

Besides the simple additions to the field types (field1DReal, field2DReal, etc.) described here, we will also need to extend the infrastructure routines that deal with allocating, deallocating, and reading in fields to be aware that there may be multiple blocks for a field; however, this work will be addressed by work on items 2, 4, and 5 identified in the Introduction to this document.

```
! Derived type for storing list of blocks from a domain
! to be handled by a process
type domain_type
   type (block_type), pointer :: blocklist

   ! Also store parallelization info here
   type (dm_info), pointer :: dminfo
end type domain_type
```

```
type dm_info
   integer :: nprocs, my_proc_id, comm, info
end type dm_info
```

```
! Derived type for storing part of a domain; used as a basic
! unit of work for a process
type block_type

#include "block_group_members.inc"

   type (domain_type), pointer :: domain

   type (parallel_info), pointer :: parinfo

   type (block_type), pointer :: prev, next
end type block_type
```

```
! Type for storing (possibly architecture specific) information
! concerning parallelism
type parallel_info
   type (exchange_list), pointer :: cellsToSend
   type (exchange_list), pointer :: cellsToRecv
   type (exchange_list), pointer :: edgesToSend
   type (exchange_list), pointer :: edgesToRecv
   type (exchange_list), pointer :: verticesToSend
   type (exchange_list), pointer :: verticesToRecv
end type parallel_info
```

```
! Derived type for storing fields
type field3DReal
   type (block_type), pointer :: block
   real (kind=RKIND), dimension(:,:,:), pointer :: array
   type (io_info), pointer :: ioinfo
   type (field3DReal), pointer :: prev, next
   type (exchange_list), pointer :: sendList
   type (exchange_list), pointer :: recvList
end type field3DReal
```

```fortran
! Derived type describing info for doing I/O specific to a field
type io_info
   character (len=1024) :: fieldName
   integer, dimension(4) :: start
   integer, dimension(4) :: count
   logical :: input
   logical :: sfc
   logical :: restart
   logical :: output
end type io_info
```