Revisions to MPAS block decomposition routines

January 30, 2012

Contents

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 last item, namely, the extensions to the block decomposition module that will be necessary for supporting multiple blocks per task in other infrastructure modules.

For a broader scope of this project, the intent with these five previously detailed tasks is to provide the capabilities within MPAS to support PIO and simulations where the number of blocks in a decomposition are not equal to the number of MPI tasks. For example, a simulation could run on 16 processors with a total of 64 blocks, as opposed to the current framework where only 16

blocks can run at 16 processors.

After these tasks are implemented shared memory parallism can be implemented at the core level to (hopefully) improve performance, but also allow greater flexibility in terms of the parallel infrastructre of MPAS.

As a rough timeline, these 5 tasks are planned to be completed by the end of February, 2012.

Chapter 2

Requirements

The changes to the block decomposition module should enable an MPI task to get a list of it's owned cells, as well as the block number each of those cells lives on within it's task.

- The user must be able to specify the number of blocks in a simulation.
- Block decomposition modules must provide information describing the cell and block relationship for a given MPI task.
- Block decomposition modules need to be flexible enough to support multiple methods of acquiring a decomposition.
- Block decomposition modules need to support a different number of blocks than MPI tasks, even when they are not evenly divisible.
- Block decomposition modules should provide an interface to map a global block number to local block number, and owning processor number.

Chapter 3

Design

We propose several changes to the block decomposition module in order to support multiple blocks per MPI task. Currently, in order to support the case where there are multiple blocks per MPI task a namelist parameter needs to be added that will allow these two values to differ.

First, changes to the namelist.input file include a new section called decomposition. This section will include four parameters. The first being config_number_of_blocks which is an integer representation of the number of blocks a run should use. Second is config_block_decomp_file_prefix, which represents the path and prefix (before the .N) of the file for the block decomposition. Third is config_proc_decomp_file_prefix representing the path and prefix (before the .Np) to the file for the processor decomposition. Finally is config_explicit_proc_decomp which is logical and tells MPAS to use the config_proc_decomp_file for the distribution of blocks, or to use the built in method of distributing the blocks to processors.

config_number_of_blocks option will have a default value of 0. A value of 0 in this field means there should be nProcs blocks, or one block for every MPI task, which is the default behavior currently. config_block_decomp_file_prefix is read by default and config_proc_decomp_file_prefix is only read for external block assignment, as will be described later.

Inside mpas_block_decomp.F, the mpas_block_decomp_cells_for_proc needs to be changed. To not only read in all cells in all blocks, but also their block numbers.

The meaning of the contents of graph.info.part.N needs to change from the processor ID that owns a cell, to the global block number for a cell. This means the file that is read in with have N = config_number_of_blocks.

Given a graph.info.part.N file, the global block number needs to be mapped into both an owning processor number, and a local block id. The local block id does not need to be computed within mpas_block_decomp_cells_for_proc as long as the mapping is available or known.

The api for mpas_block_decomp_cells_for_proc will change from

```
subroutine mpas_block_decomp_cells_for_proc(dminfo, &
           partial_global_graph_info, local_cell_list)
```

to

```
subroutine mpas_block_decomp_cells_for_proc(dminfo, &
           partial_global_graph_info, local_cell_list, block_id, &
                 block_start, block_count)
```

where local_cell_list is a list of cells owned by a processor that is sorted by local block id, block_id is a list of global block id's that an MPI task owns, block_start is a list of offsets in local_cell_list for the contiguous cells a block owns, and block_count is a the number of cells each block owns.

mpas_block_decomp_cells_for_proc will perform the same regardless of number of processors to enable the use of multiple blocks on a single processor.

The block_type data structure will be extended to include the local block id. This can help make dynamic load balancing easier for implementation at a later time. This will change

```
   type block_type

#include "block_group_members.inc"

      integer :: blockID   ! Unique global ID number for this block

      type (domain_type), pointer :: domain

      type (parallel_info), pointer :: parinfo

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

to

```
   type block_type

#include "block_group_members.inc"

      integer :: blockID   ! Unique global ID number for this block
      integer :: localBlockID

      type (domain_type), pointer :: domain

      type (parallel_info), pointer :: parinfo

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

There will be three additions to the public interface of mpas_block_decomp.F The first adds the routine mpas_get_local_block_id which takes has three arguments. As input it takes the domain information and the global block number, and as output it provides the local block number on a processor. This allows other parts of MPAS to determine what local block number a global block is, even if it's on another processor.

The second addition to the public interface is the subroutine mpas_get_owning_proc. This subroutine takes as input the domain information and the global block number, and as output provides the MPI task number that owns the block. This allows other parts of MPAS to determine from a block number which MPI task it needs to communicate with to read/write this block.

The third public routine is called mpas_get_blocks_per_proc. This routine takes as input the domain information and a processor number. On output blocks_per_proc contains the number of blocks a processor owns.

In addition to the ad-hoc method of determining which blocks belong to which processors, a file based method will be added. This method will be toggelable by a namelist option named config_explicit_proc_decomp, which will be logical. If this option is true, a file (config_proc_decomp_file_prefix.N) will be provided, where N is the number of processors. This file will have number of blocks lines, and each line will say what processor should own the block. This file can be created using metis externally.

Chapter 4

Implementation

Implementation of the mpas_get_blocks_per_proc subroutine is as follows:

```
subroutine mpas_get_blocks_per_proc(dminfo, proc_number, blocks_per_proc)
  type(domain_info), intent(in) :: dminfo
  integer, intent(in) :: proc_number
  integer, intent(out) :: blocks_per_proc
  integer :: blocks_per_proc_min, even_blocks, remaining_blocks

  blocks_per_proc_min = config_number_of_blocks / dminfo % nprocs
  remaining_blocks = config_number_of_blocks - &
         (blocks_per_proc_min * dminfo % nprocs)
  even_blocks = config_number_of_blocks - remaining_blocks

  blocks_per_proc = blocks_per_proc_min
  if(proc_number .le. remaining_blocks) then
    blocks_per_proc = blocks_per_proc + 1
  end if
end subroutine mpas_get_blocks_per_proc
```

Implementation of the mpas_get_local_block_id is as follows:

```
subroutine mpas_get_local_block_id(dminfo, &
           global_block_number, local_block_number)
  type(domain_info), intent(in) :: dminfo
  integer, intent(in)  :: global_block_number
  integer, intent(out) :: local_block_number
  integer :: blocks_per_proc_min, even_blocks, remaining_blocks

  blocks_per_proc_min = config_number_of_blocks / dminfo % nprocs
  remaining_blocks = config_number_of_blocks - &
        (blocks_per_proc_min * dminfo % nprocs)
  even_blocks = config_number_of_blocks - remaining_blocks

  if(global_block_number > even_blocks) then
       local_block_number = blocks_per_proc_min - 1
  else
    local_block_number = mod(global_block_id, blocks_per_proc_min)
  end if
end subroutine mpas_get_local_block_id
```

Implementation of the mpas_get_owning_proc routine is as follows:

```
subroutine mpas_get_owning_proc(dminfo, &
           global_block_number, owning_proc)
  type(domain_info), intent(in) :: dminfo
  integer, intent(in) :: global_block_number
  integer, intent(out) :: owning_proc
  integer :: blocks_per_proc_min, even_blocks, remaining_blocks

  blocks_per_proc_min = config_number_of_blocks / dminfo % nprocs
  remaining_blocks = config_number_of_blocks - &
        (blocks_per_proc_min * dminfo % nprocs)
  even_blocks = config_number_of_blocks - remaining_blocks

  if(global_block_number > even_blocks) then
    owning_proc = global_block_number - even_blocks
  else
    owning_proc = global_block_number / blocks_per_proc_min
  end if
end subroutine mpas_get_owning_proc
```

Chapter 5

Testing

Only limited testing can be performed on this task. Since this task alone doesn't allow the use of multiple blocks the only testing that can really be performed is to provide a mis-matched number of blocks and MPI tasks and verify the block decomposition routines provide the correct block numbers for a processor and put the cells in their correct block numbers.