Non-blocking Halo Exchanges: Requirements and Design

MPAS Development Team

February 21, 2013

Contents

Chapter 1

Summary

Within the MPAS framework, blocking halo exchanges are currently provided. These halo exchanges support the use of multiple blocks per MPI task, however an alternative version of halo exchanges might be preferred when using optimally distributed blocks. These alternative halo exchanges can be referred to as, non-blocking halo exchanges.

This document describes the requirements and implementation for a non-blocking halo exchange within the MPAS framework.

Chapter 2

Requirements

2.1 Requirement: Allow Non-blocking halo exchanges

Date last modified: 02/21/13 Contributors: (Doug Jacobsen, Michael Duda)

The MPAS framework should allow the use of non-blocking halo exchanges, to overlap computation and communication as much as possible.

2.2 Requirement: Allow interior and exterior block loops

Date last modified: 02/21/13 Contributors: (Doug Jacobsen, Michael Duda)

The MPAS framework should provide developers with a method of determining which blocks are interior or exterior. These blocks will be described in the implementation section.

Chapter 3

Design and Implementation

3.1 Implementation: Allow interior and exterior block loops

Date last modified: 02/21/13 Contributors: (Doug Jacobsen, Michael Duda)

While MPAS currently provides the ability for a single MPI task to have multiple computational blocks, these blocks can be arbitrarily distributed. This means the blocks don't have to be part of a larger contiguous block.

Although this is true, in order to overlap computation and communication a user would prefer to assign connecting blocks to an MPI task. This allows local copies to remove some of the time between the MPI send/recv and MPI waits within halo exchange routines.

Furthermore, if the blocks do create a larger contiguous block, they can additionally be flagged as interior or exterior. An interior block would be a block that only communicates with other blocks that live on the MPI task, while exterior blocks communicate with blocks that live on other MPI tasks and can't communicate with only local copies.

To enable this functionality, a logical flag isInterior is added to each block. When creating the exchange lists, this flag is set appropriately based on the previous definition.

Block loops can then be modified in the following way to loop over either interior or exterior blocks.

```
!Exterior block loop
block => domain % blocklist
do while (associated(block))
        if(.not. block % isInterior) then
                exterior block computations
        end if

        block => block % next
end do

!Interior block loop
block => domain % blocklist
do while (associated(block))
        if( block % isInterior) then
                interior block computations
        end if

        block => block % next
end do
```

3.2 Implementation: Non-blocking halo exchanges

Date last modified: 02/21/13 Contributors: (Doug Jacobsen, Michael Duda)

The currently provided halo exchanges are considered blocking, meaning once you begin a halo exchange you can't do anything else until the halo exchange is completed. In order to overlap computation and communication developers might like the ability to have non-blocking halo exchanges. The blocking halo exchanges currently have the following structure:

```
Initialize Halo Exchange:
        Build communication lists
        Allocate buffers
        Initiate non-blocking recv
        Fill send buffer
        Initiate non-blocking send

Handle local copies:
        Copy elements between blocks on
                a single MPI task

Finalize Halo Exchange:
        Wait for non-blocking recv to finish
        Unpack buffer
        Wait for non-blocking send to finish
        Destroy buffers and communication lists
```

Currently the communication lists contain the buffers, and persist only as long as the MPI task is within the halo exchange routine. Alternatively, communication lists can be attached to a field as below.

```
! Derived type for storing fields
type field1DInteger

   ! Back-pointer to the containing block
   type (block_type), pointer :: block

   ! Raw array holding field data on this block
   integer, dimension(:), pointer :: array

   ! Information used by the I/O layer
   type (io_info), pointer :: ioinfo        ! to be removed later
   character (len=StrKIND) :: fieldName
   character (len=StrKIND), dimension(:), pointer :: constituentNames => null()
   character (len=StrKIND), dimension(1) :: dimNames
   integer, dimension(1) :: dimSizes
   logical :: hasTimeDimension
   logical :: isSuperArray
   type (att_list_type), pointer :: attList => null()

   ! Pointers to the prev and next blocks for this field on this task
   type (field1DInteger), pointer :: prev, next

   ! Halo communication lists
   type (mpas_communication_list), pointer :: commSendList, commRecvList
   type (mpas_multihalo_exchange_list), pointer :: sendList
   type (mpas_multihalo_exchange_list), pointer :: recvList
   type (mpas_multihalo_exchange_list), pointer :: copyList
end type field1DInteger
```

Additionally, each of the phases from the current halo exchanges can be abstracted into it's own subroutine. This provides the following new interfaces:

```
mpas_dmpar_begin_exch_halo_field
mpas_dmpar_local_exch_halo_field
mpas_dmpar_end_exch_halo_field
```

Each of these subroutines is called exactly like the current blocking halo exchange routines, however they now create and work on communication lists attached to the first block's field that is to be communicated. This allows the communication list to persist, until destroyed. Even if the destruction of the communication list happens within another subroutine.

Using this new structure, and the previously described interior and exterior block flags a block loop with halo exchanges that look similar to:

```
block => domain % blocklist
do while(associated(block)
        call mpas_compute_velocity(block)
        call mpas_compute_divergence(block)
        block => block % next
end do

call mpas_dmpar_exch_halo_field(domain % blocklist % velocity)
call mpas_dmpar_exch_halo_field(domain % blocklist % divergence)
```

can now be broken up into two separate loops, as follows:

```
block => domain % blocklist
do while(associated(block)
        if ( .not. block % isInterior ) then
          call mpas_compute_velocity(block)
          call mpas_compute_divergence(block)
        end if
        block => block % next
end do

call mpas_dmpar_begin_exch_halo_field(domain % blocklist % velocity)
call mpas_dmpar_begin_exch_halo_field(domain % blocklist % divergence)

block => domain % blocklist
do while(associated(block)
        if ( block % isInterior ) then
          call mpas_compute_velocity(block)
          call mpas_compute_divergence(block)
        end if
        block => block % next
end do

call mpas_dmpar_local_exch_halo_field(domain % blocklist % velocity)
call mpas_dmpar_local_exch_halo_field(domain % blocklist % divergence)

call mpas_dmpar_end_exch_halo_field(domain % blocklist % velocity)
call mpas_dmpar_end_exch_halo_field(domain % blocklist % divergence)
```

Chapter 4

Testing

4.1 Testing and Validation: Non-blocking halo exchanges

Date last modified: 02/21/13
Contributors: (Doug Jacobsen, Michael Duda)

Simulations can be performed with blocking and non-blocking halo exchanges using the same decomposition. These simulations should be bit-reproducible regardless of core.