An MPI Halo-Cell Implementation for Zero-Copy Abstraction

EuroMPI 2015
Runtime and Programming Models
September 21-23, Bordeaux

Jean-Baptiste Besnard (1), Allen Malony (2), Sameer Shende (1), Marc Pérache (3), Patrick Carribault (3) and Julien Jaeger (3)

1. ParaTools SAS, Bruyères-le-Châtel
2. ParaTools Inc, Eugene USA
3. CEA, DAM, DIF F91297 Arpajon France

jbbesnard@paratools.fr
HPC machines are rapidly shifting to higher concurrency

- Now gathering millions of cores
- Intra-node parallelism is rapidly increasing (several hundred threads) (Xeon Phi / KNL)
- This with a smaller memory per thread

It is well acknowledged that applications will have to evolve in order to take advantage of such architectures. MPI + X being often referred to as a potential solution.
But what does it mean…

What is this \( X \)?

<table>
<thead>
<tr>
<th>Distributed Memory</th>
<th>Shared-Memory</th>
<th>Accelerators</th>
<th>Logical Address Spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI and optimized intra-node communications</td>
<td>OpenMP, Cilk, TBB, Pthreads, …</td>
<td>GPUs, FPGAs</td>
<td>PGAS, DSM</td>
</tr>
</tbody>
</table>

There are several alternatives: 
\( MPI + \) OpenMP, \( MPI + \) GPU, \( MPI + \) PGAS, ….
But what does it mean…

- Why MPI is not sufficient? Why do we need this $X$?

In our paper, we propose to model this limitation when considering domain-splitting in distributed memory context.

We show that distributed memory poses problems of:
- Memory due to domain replication
- Communication overhead and therefore scalability

Then, we propose an MPI level abstraction solving these issues for domain splitting by providing the advantages of shared-memory programming.
We consider the case where computation is done over a distributed domain (often as a stencil) creating dependencies between cells structured as a mesh.

----> This covers a wide range of applications (not all)
It is common knowledge for all MPI programmers that such domain splitting requires halo/ghost cells on local domain boundaries.

Is it possible to provide a simple model of the halo-cells? What is the performance impact for common topologies?

Yes (first part of our paper)
Halo-Cell Model (1/6)

To derive this model, we considered wrapped-around meshes (tori) instead of regular ones in order to have a regular mesh layout (no border effect).
These regular topologies are nonetheless completely representative of unwrapped ones dealing with the level of connectivity between distributed areas.
Halo-Cell Model (3/6)

- \( n \): Number of cells
- \( C \): Number of halo layers
- \( d \): Mesh dimension
- \( l \): Characteristic length of the topology

\[
l(n, d) = n \frac{1}{d}
\]

\[
N_g(n, d) = (l(n, d) + 2C)^d - l(n, d)^d = \left(n \frac{1}{d} + 2C\right)^d - n
\]

« Subtract a mesh without halo-cells to a mesh with a characteristic length increased of 2C. »
Halo-Cell Model (4/6)

<table>
<thead>
<tr>
<th>$d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_g(p, n, C)$</td>
<td>$2pC$</td>
<td>$4pC(\sqrt{\frac{n}{p}} + C)$</td>
<td>$2pC\left(3 \frac{n}{p}^{\frac{2}{3}} + 6C \frac{n}{p}^{\frac{1}{3}} + 4C^2\right)$</td>
</tr>
</tbody>
</table>

$2 \times 3$ (processes) $\times$ 1 (layer) = 6 halo cells
Halo-Cell Model (5/6)

2D

\[ 4pC \left( \sqrt{\frac{n}{p}} + C \right) \]

\[ = 4.4.1 \left( \sqrt{\frac{16}{4}} + 1 \right) \]

\[ = 4.4.1 \left( \sqrt{4} + 1 \right) \]

\[ = 4.4.3 \]

\[ = 48 = (4 \times 12) \text{ halo cells} \]
Halo-Cell Model (6/6)

2D (two layers)

\[ 4pC\left(\sqrt{\frac{n}{p}} + C\right) \]

= \[ 4.4.2(\sqrt{16/4} + 2) \]

= \[ 4.4.2(\sqrt{4} + 2) \]

= \[ 4.4.2.4 \]

= \[ 128 = (4 * 32) \text{ halo cells} \]
Halo-Cells and Performance (1/4)

\[
S(n, p) = \frac{\frac{s(n)}{p}}{s(n)} + \text{comm}(n, p)
\]

Starting from the well-known speedup equation, it can be seen that strong-scaling speedup is bounded by communications which are directly linked to the number of halo-cells.

\(\rightarrow\) Computation time should be much larger than communication time. There should be more local cells than halo cells with a complex computation.

However, this ratio changes with \(p\) (strong scaling)
If we now consider the weak-scaling model, we have n/p which is a constant as is the ghost cell ratio.

Communication cost has then to be independent of the number of processes, in order to allow weak-scaling. Which is true for regular decomposition?
When doing weak-scaling, it is desirable to limit the ghost-cell ratio in order to completely hide communication costs.

However, memory per thread is decreasing:

In 3D, if you want 1% of ghost cells with one layer, you need 1.64 GB of memory (for 8 bytes cells). Compare it to the 34 MB / Thread on a Xeon Phi.
Halo-Cells and Performance (4/4)

The graph illustrates the relationship between the number of cells per memory area and the ghost cell ratio for different dimensional cases.

- **1D C=1**
- **2D C=2**
- **3D C=3**

The ghost cell ratio is plotted on the y-axis, while the number of cells per memory area is on the x-axis. Different lines represent different dimensional cases with specific ghost cell ratios indicated at various points on the graph.
Hybrid Approach

Intra-node parallelism is then a direct way of reducing the ghost cell ratio and then improving scalability by overcoming the per thread memory limitation.

- Reducing communication cost
- Limiting ghost-cell memory overhead while freeing memory for computation (hiding comms)
MPI Optimized Intra-Node Messaging

A lot of work has been done to optimize intra-node communications:

- SHM memory segments
- KNEM kernel module
- Or since Linux 3.2 Cross Memory Attach (CMA)
- Direct copy in thread-based MPI
- It is even possible to use the HCA to emit RDMA

Such approaches efficiently reduce node-local communication cost but do not reduce/remove the memory associated with halo cells which still has to be duplicated.
MPI Halo

We propose a Halo Cell abstraction providing the advantages of shared-memory models while remaining close to MPI semantics:

- Transparent use of larger memory areas
- Removal of memory duplications between tasks on the same node
- Removal of node-local communications (no copies — Zero copy)
- Support for computation outside of node boundaries (no mixing)
When doing a stencil, most applications use two meshes, one for « t » and another for « t+1 », approach required due to the spatial-dependency between cells.
MPI Halo Principle (2/4)

What if local cells (located on the same node) could be resolved as local pointers — no copies would be required. The source mesh being accessed in read-only is not necessary to duplicate data.
MPI Halo Principle (3/4)

Pointer exchanges allow mesh-switching
MPI Halo Principle (4/4)

Illustration of both inter-node and intra-node exchanges with MPI-Halo cells.
MPI Halo Example (1D splitting) (1/2)

/*----- Initialization (Done once) */
MPI_Halo local_left, local_right, left, right;
/* Name Cells and provide Layout */
MPIX_Halo_cell_init( &local_left, "Local Left" , MPI_INT, 1024 );
MPIX_Halo_cell_init( &local_right, "Local Right" , MPI_INT, 1024 );
MPIX_Halo_cell_init( &left, "Remote Right" , MPI_INT, 1024 );
MPIX_Halo_cell_init( &right, "Remote Left" , MPI_INT, 1024 );
/* Bind Cells */
MPI_Halo_ex ex;
MPIX_Halo_exchange_init( &ex );
MPIX_Halo_cell_bind_local( ex, local_left );
MPIX_Halo_cell_bind_local( ex, local_right );
MPIX_Halo_cell_bind_remote( ex, right, right_process, "Local Left" );
MPIX_Halo_cell_bind_remote( ex, left, left_process, "Local Right" );
/* Generate Communications */
MPIX_Halo_exchange_commit( ex );
MPI Halo Example (1D splitting) (2/2)

/*----- Compute Loop (Called at each time-step)*/
while( compute )
{
    /* Register local cell data */
    MPIX_Halo_cell_set( local_left, mesh );
    MPIX_Halo_cell_set( local_right, right_coll(mesh) );
    /* Start asynchronous communications */
    MPIX_Halo_iexchange( ex );
    /* ... Compute mesh center ... */
    MPIX_Halo_iexchange_wait( ex );
    /* Retrieve Ghost arrays */
    int* left_ghost, * right_ghost;
    MPIX_Halo_cell_get( left, (void**)&left_ghost );
    MPIX_Halo_cell_get( right, (void**)&right_ghost );
    /* ... Compute mesh boundaries ... */
    /* Swap Meshes */
    Mesh * tmp = mesh;
    mesh = oldmesh;
    oldmesh = tmp;
}
MPI Halo Interface

**MPI_Halo:**
- Automatic buffer abstraction (local or remote)
- Can be set to a value when local
- A pointer can be retrieved when remote
- Supports MPI data-types (packing abstraction)

**MPI_Halo_ex:**
- Build the communication scheme between MPI_Halo
- Buffers are named (no abstract offsets)
- An error is reported if the remote is not present
- No offset is passed to communication calls
  - **Boundaries have to be handled as particular case**
- Copy can still be forced when the remote is modified
Our test-case was the convolution of a 5616x3744 RBG image implemented in OpenMP, MPI-Halo, also forcing buffer allocation to behave like the classical ghost-cell approach. We tested this benchmark with various convolution kernel sizes.

Our MPI-Halo interface has been implemented in the MPC runtime which is a thread-based MPI, making node-level exchanges trivial (shared-memory). Nothing prevents the MPI Halo model to be ported to process-based MPI supposing a previous memory registration.
MPI Halo Performance Results (2/3)

Computation Time
Memory Usage

MPI Halo Performance Results (3/3)
Conclusion

Halo-Cell Model:
- Introduced a model of the halo-cell ratio
- Explained that scaling was highly impacted by this ratio
- Shown that distributed memory was hitting the per-thread memory barrier, encouraging hybrid models to achieve better ghost-cell ratios particularly for higher dimensions (3D with several layers).

MPI_Halo:
- Proposed an MPI based solution to the domain decomposition issue we exposed (buffer aliasing)
- Allows a clear definition of a communication scheme with static validation of buffer matching (size, name)
- Consistent with inter-node parallelism (unlike OpenMP)
An MPI Halo-Cell Implementation for Zero-Copy Abstraction

EuroMPI 2015
Runtime and Programming Models
September 21-23, Bordeaux

Jean-Baptiste Besnard (1), Allen Malony (2), Sameer Shende (1), Marc Pérache (3), Patrick Carribault (3) and Julien Jaeger (3)

1. ParaTools SAS, Bruyères-le-Châtel
2. ParaTools Inc, Eugene USA
3. CEA, DAM, DIF F91297 Arpajon France

jbbesnard@paratools.fr