Scalable Remote Memory Access Halo Exchange with Reduced Synchronization Cost

Maciej Szpindler
m.szpindler@icm.edu.pl

University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling
http://www.icm.edu.pl

Agenda

- Introduction and motivation
- Problem definition
- Synchronization costs
- Migration steps
- Preliminary results
- Summary

2016, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

Introduction and context

- This presentation describes approach on migration from message passing to remote memory operations in complex MPI code
- The work is in-progress and results are not final

- Majority of well established scientific codes use message passing for communication
- MPI-3 (current standard version, last update 2015) introduced redesigned RMA (one-sided) communication model
  - Most of modern HPC hardware support RMA natively
  - Cray (XC line) support RMA with DMAPP interface
- How application can benefit from these mechanisms?

Site Information

- Site ID **WARSAWU**, joined CUG (again*) in 2016
- Interdisciplinary Centre for Mathematical and Computational Modelling (**ICM**) is computational and data sciences research institution at the University Warsaw, Poland.
- Maintains HPC centre for academic users
- Hosts Cray XC40 (approx. 1000 nodes, 24-core nodes)

- In operation from March 2016
- System name: okeanos

* Long-ago hosted Y-MP, J90, SV1, X1 Cray machines

Motivation

- System evaluation – programming environment and libraries
- MPI-3 RMA seems mature and portable
  - Not really usable in previous implementations (MPI-2)
- Would one-sided communication provide:
  - Performance benefits (versus classic message passing)?
  - Scalability?
- How much work is needed to enable it in large application code? Is it universal?
- It is *believed* that RMA would not deliver better performance than message passing
  - Is it true (at least for selected case)?
  - Are there bottlenecks that can be addressed?

Current state
RMA – tools used

- MPICH3 – reference implementation of MPI-3
- Cray MPI
  - Is based on MPICH
  - Use DMAPP layer for one-sided operations
  - „Thread hot” (I was not aware)
- Other implementations
  - foMPI: fast one-sided implementation (research implementation, ETH Zurich)
  - foMPI-NA: implementation of the Notified Access mechanism on top of the foMPI

Problem definition

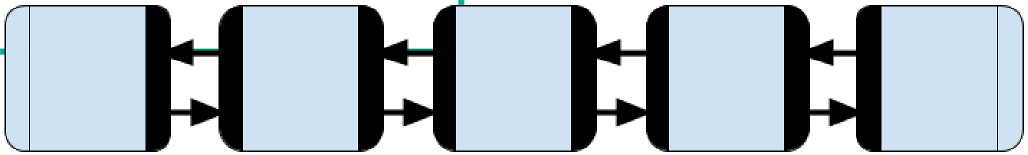
- Given halo exchange implementation with message passing
- Fortran module, part of the large code
  - Unified Model* is a code of interest, Fortran90
  - OpenMP have not been addressed
- Is it possible, and practical, to change only the module to enable RMA?
  - Need to preserve structures, buffers, etc.
  - Additional initialization only at the module scope
- Similar approach can be applied to other codes in that field
  - Performance advances will be translated to other weather code:
  - EULAG (open-source, http://www2.mmm.ucar.edu/eulag/)

* MetOffice proprietary, used under license agreement

Halo exchange scheme

- Each process exchanges columns on its data boundary with neighbors
- EW and NS exchanges are separated (different procedures)
- For LAM usage processes on boundaries are asymmetric in number of neighbors
- Exchanges are local only
- Original implementation:

```
CALL MPI_Irecv(recv_buffer, halo_size, MPI_REAL8,
     & neighbour, tag, ew_comm, ...)
! Prepare send_buffer
CALL MPI_Isend(send_buffer, halo_size, MPI_REAL8,
     & neighbour, tag, ew_comm, ...)
CALL MPI_Waitall(...)
```



EW exchange flow

Synchronization costs

- Ideally MPI data exchange requires*:
- For message passing – transmission and synchronization coupled in send/recv pair
  - Eager protocol: **single** transaction (plus additional matching)
  - Randevous protocol: at least **three** transactions
- For remote memory access (RMA) – transfer and synchronization is decoupled
  - At least **three** transactions for transfer (put or get) and synchronization

* Taken from: Belli, Hoefler, *Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization*

Migration setup

- System setup
  - 16 : 32 XC40 nodes partition, 24-core nodes
  - Cray compiler, Cray MPI
  - Not using HT (--hint=nomultithread)
- Application setup
  - Unified Model, vn10.1
  - Small regional domain (real case, not benchmark)
    - Grid size: 616x448x70, LAM
    - 100 s time step
    - Halo width: 5 grid points
  - Each model step requires about 170 halo exchanges in this configuration
  - Pure MPI setup

Migration steps

- **Step 0**: isolation of the module and use with dummy test
  - Original isend/irecv/waitall implementation is fine tuned
- **Step 1**: RMA implementation using Post-Start-Complete-Wait scheme (active target)
  - Used guidelines on synchronization mode from:

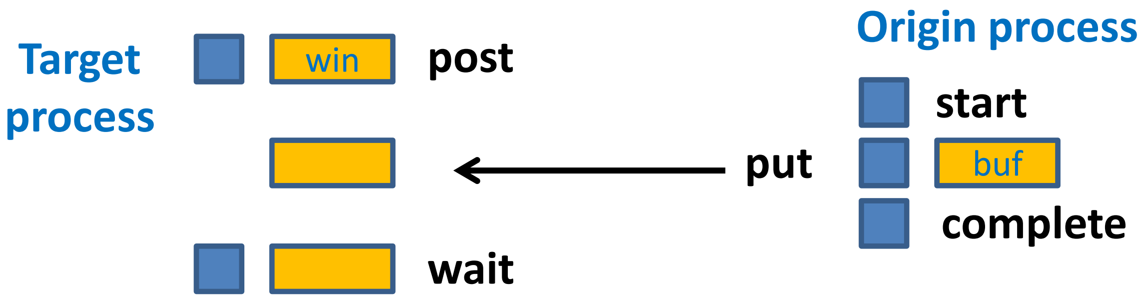
    Remote Memory Access Programming in MPI-3, Hoefler, Dinan, Thakur et al (paper); MPI Standard; Using Advanced MPI, Hoefler (book)
  - No any changes in module structure:
    - Halo buffers are used (automatic arrays)
    - RMA windows created with each halo exchange
  - Missing MPI RMA wrappers added (because of required interface library)

- Validation of results: reproducibility of output files
- Validation of performance: OSU Micro Bencharks, Intel PRK

2016, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

PSCW scheme

- The aim is to have fine-grained synchronization of RMA exchanges
  - Not in the communicator (row/col) scope
  - Only in pairs of neighbors processes




```
CALL MPI_Win_create(recv_buffer, halo_size*8, 8, info, ew_comm, win, ... )
! Initialize origin and target process groups
CALL MPL_Win_post(origin_group, MPL_MODE_NOSTORE, win,  ... )
! Prepare data to stored in remote window
CALL MPI_Win_start(target_group, 0, win, ... )
CALL MPI_Put(send_buffer, halo_size, MPI_REAL8, &
     neighbour, 0, halo_size, MPI_REAL8, win, ... )
CALL MPI_Win_complete(win, ... )
! Synchronize memory windows
CALL MPI_Win_wait(win, ... )
```


2016, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

1st Results

	EW AVG	EW MIN	EW MAX	NS AVG	NS MIN	NS MAX
Message Passing	0.97	0.69	1.20	1.74	0.25	2.45
RMA	1.89	1.61	2.19	7.58	5.90	9.59

- Collected total time (s) from short runs – 48 model steps on 16 XC nodes
  - Why it is that bad?
  - Benchmark codes on the same system performed similarly for RMA and message passing
  - Implementation is fairly the same as for benchmarks (OSU and PRK)

Migration steps cont.

- **Step 2**: introduction of dynamic windows
  - Improved memory (windows) management
  - Windows are initiated with module initialization
  - Buffers are attached to existing windows (this mode is dynamic)

- In the dynamic case relative addressees of memory regions need to be circulated across remote processes
  - Requires additional communication (single MPI_AINT)
  - … and synchronization
  - Sendrecv used, better choices possible
  - Not that scalable (in theory) as intended

2016, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

Results cont.

	EW AVG	EW MIN	EW MAX	NS AVG	NS MIN	NS MAX
Message Passing	0.97	0.69	1.20	1.74	0.25	2.45
Initial RMA	1.89	1.61	2.19	7.58	5.90	9.59
RMA dynamic	**1.09**	**0.74**	**1.29**	**2.11**	**0.56**	**2.75**

- Refined a lot – why really?
- Additional sendrecvs seem not affect performance that much

More migration …

- **Step 3**: introduction of dynamic windows
  - Try another MPI RMA implementation
  - foMPI (ETH Zurich research implementation)
  - For isolated dummy test (Step 0) works quite fine
    - Memory for remote access need to be aligned (how to achieve this for automatic arrays in Fortran?)
    - Significantly long startup time, but performance is good

Even more migration …

- **Step 4**: introduction of notified access
  - Idea* implemented in foMPI(-NA)
  - Targets at lower RMA synchronization cost – at two transactions per data exchange
  - Different communication scheme:

```
CALL foMPI_Win_create(recv_buffer, halo_size*8, 8, info, ew_comm, win, ... )
! Initialize notification requests
CALL foMPI_Notify_init(win, neighbour, tag, count, req,  ...)
! Prepare data to stored in remote window
CALL foMPI_Put_notify(send_buffer, halo_size, MPI_REAL8, &
     neighbour, 0, halo_size, MPI_REAL8, win, tag, ...)
! Synchronize memory windows
CALL foMPI_Start(req, ...)
CALL foMPI_Wait(req, stat, ...)
CALL foMPI_Win_flush(neighbour, win, ...)
```

* Belli and Hoefler. Notified access: Extending remote memory access programming models for producerconsumer synchronization. In Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, pages 871–881. IEEE, 2015.

Even more migration … failed

- **Step 4**: introduction of notified access
  - Idea implemented in foMPI(-NA)
  - Targets at lower RMA synchronization cost – at two transactions per data exchange
  - Cray DMAPP implementation specific (although work only with GNU compiler)

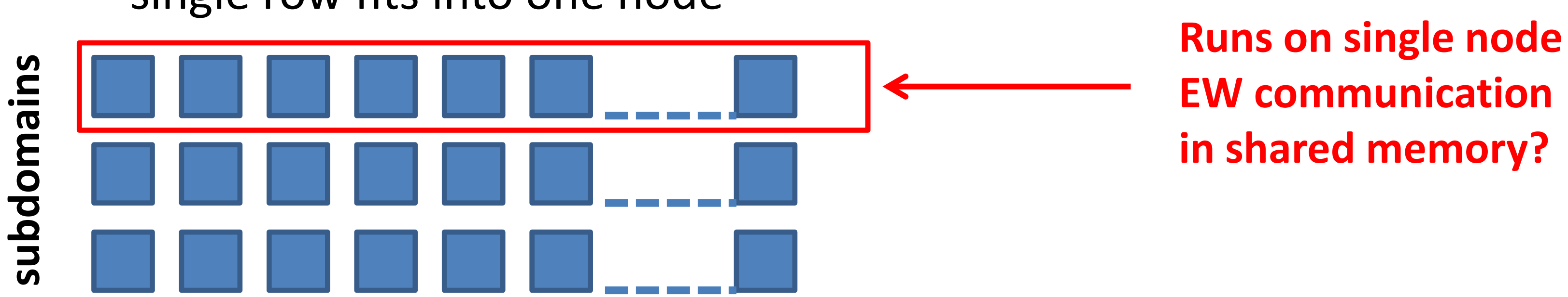
- **Step 4&5**: failed on the complete code
  - Segmentation faults
  - Difficult to debug

Does it scale?

- Rewinding to Step 3 (best working)
  - Doubled the number of nodes (to 32), problem size states the same
  - Results are not encouraging

	EW AVG	EW MIN	EW MAX	NS AVG	NS MIN	NS MAX
Message Passing	0.57 -41%	0.40 -36%	2.60 -42%	1.92 10%	0.81 224%	2.60 6%
RMA	1.76 64%	1.51 104%	2.10 54%	7.56 258%	6.43 1050%	8.22 199%

  - Decomposition of 24x32 should provide significant speedup while single row fits into one node



2016, University of Warsaw
Interdisciplinary Centre for Mathematical and Computational Modelling

Conclusions

- Case study for message-passing to one-sided migration
  - Simple halo exchange scheme with PSCW active target synchronization
  - Migrating single module of the complex code
- PSCW approach
  - Naive implementation produces poor performance
  - Memory windows usage can improve performance a lot for small number of nodes
  - Not scales for larger node counts, in preliminary tests
  - More refinement - deeper intervention in code structure required
- Notified Access
  - Interesting alternative for improved RMA synchronization costs
  - foMPI implementation tested
  - Works for synthetic test but fails for real, complex code