Scalable Remote Memory Access Halo Exchange with Reduced Synchronization Cost

Maciej Szpindler
ICM
University of Warsaw
Warsaw, Poland
Email: m.szpindler@icm.edu.pl

***Abstract*—Remote Memory Access (RMA) is popular technique for data exchange in the parallel processing. Message Passing Interface (MPI), ubiquitous environment for distributed memory programming, introduced improved model for RMA in the recent version of the standard. While RMA provides direct access to low-level high performance hardware, MPI one-sided communication enables various synchronization regimes including scalable group synchronization. This combination provides methods to improve performance of commonly used communication schemes in parallel computing. This work evaluates one-sided halo exchange implementation on the Cray XC40 system. Large numerical weather prediction code is studied. To address already identified overheads for RMA synchronization, recently proposed extension of Notified Access is considered. To reduce the cost of the most frequent message passing communication scheme, alternative RMA implementation is proposed. Additionally, to identify more scalable approaches, general active target synchronization, Notified Access modes of RMA and original message passing implementation are compared.**

***Keywords*-RMA, Parallel programming, Message passing**

I. Introduction

Remote Memory Access (RMA) is one of the principal techniques for data exchange in the parallel processing. It provides direct access to hardware communication features and enables high performance for data exchanges in distributed memory computing. While programming environments and computing hardware provide mature support for RMA it gains interest and popularity. It is a general approach for the Partitioned Global Address Space (PGAS) programming languages such as UPC and Co-array Fortran or Global Arrays library. Recently support for RMA based one-sided communications from Message Passing Interface (MPI) standard was extended as a part of MPI-3 effort [3]. These extensions of the RMA model include support for various levels of synchronization. This re-design addresses scalability challenges for massively parallel applications that use process based execution and one-sided communications.

Although scalable group synchronization functions are available in MPI-3 RMA model, there are overheads that may limit the performance of applications. These synchronization schemes are associated with additional message exchange. Specifically, this overheads are recognized for parallel applications using halo swapping pattern for communication. In this cases traditional two-sided message passing usually delivers better performance than one-sided schemes. Recently a proposal of the Notified Access that extends RMA synchronization primitives was described [1]. Proposed communication mechanism provides synchronization of remote memory accesses with less communication. This enables improved performance of RMA halo exchange patterns. It was already shown that this approach outperforms message passing implementations for select classes of problems including halo exchange schemes.

The aim of using RMA model is to benefit from fast remote direct memory access provided by specialized hardware. Combined with improved synchronization schemes, RMA provide high performance data exchange for stencil pattern computation and communication schemes that are core kernels for wide range of scientific applications. The motivation of this work is to experiment with described RMA schemes beyond typical communication performance benchmarks and to study performance on real complex scientific code. Spectrum of problems that are solved numerically with stencil scheme data processing is very wide. This work concentrates on the numerical weather prediction where stencil computations are heavily used. Incremental migration from legacy point-to-point message passing to one-sided halo exchange is analysed. The focus of the data exchange method is on scalability of the communication.

The one-sided halo exchange scheme with eventual notified access usage was evaluated on the Cray XC40 system. Data exchange was incrementally implemented in a large numerical weather prediction code. Transitional steps and implementation choices are discussed. The aim of this approach was to reduce the cost of the most frequent communication routines and to identify more scalable approaches. Part of message passing communication routines for halo swapping was isolated and switched to one-sided implementation. To find performance improvements for RMA implementation, Notified Access based synchronization was considered. Open source MPI-3 RMA library with Notified Access extension, described in [1], was tested.

RMA programming with MPI gains more interest with

MPI-3 standard adoption. Most of available MPI libraries provide support for the current generation of the standard. Performance of the common RMA data exchange schemes are found to have lower performance (in case of operation per second) comparing to traditional message passing. One of the efforts to answer identified limitations of MPI's RMA model is Notified Access proposal. Early implementations show improved performance that already exceed message passing. Although these benefits are demonstrated only for mini benchmark codes (including stencil scheme) as reported in [1]. Extending this performance improvements to more complex application codes is original contribution of the reported work.

Problem addressed with this work describes section II. General assumptions and applied methods are described in section III. Implementation details are discussed in section IV and results of its evaluation are discussed in section V.

II. Stencil scheme and halo exchange

Stencil scheme is common iterative procedure which updates given element of the matrix or array with a fixed pattern. Dependency of newly computed matrix element value with its neighbours values follows this pattern called stencil. Most finite difference methods for numerical differential equations are formulated as stencil schemes. In case when numerical solver is parallelized, input matrix (or domain) is decomposed into parts that are solved in parallel. Running stencil codes on distributed memory computers requires extended boundary elements to be cached and exchanged to satisfy stencil pattern locally. This elements of the sub-domain boundary are called halo. Halo exchange is often used jargon in computational sciences. It means data movement between two parallel processes holding neighbouring parts of the decomposed domain.

Numerical weather prediction (or atmospheric dynamics simulations) often use finite difference solvers and stencil schemes. To achieve accuracy and interesting simulation time scales often significant number of solver iterations are required. Moreover each iteration step usually use iterative method internally. This gives large number of halo exchanges that contributes to total simulation runtime. When scaling the solvers to thousands and hundreds of thousands of parallel processes is considered, halo exchange becomes limiting feature of the code.

In this work complex atmospheric code, the Unified Model is considered. It is proprietary of the MetOffice, UK meteorological service. Although specific code is studied, most of the methods, described here, are common for such class of codes. Methods described here are applicable to other codes as well. The code uses regular (or semi regular) grid and finite difference solvers. Its current dynamical solver is described in [5]. Two dimensional decomposition (horizontal) is used and each process holds atmospheric column sub-domain. It is written in Fortran 90 and has modular structure. The main interest of this work is on halo exchange module which is self contained. That allows to limit code intervention to that module only.

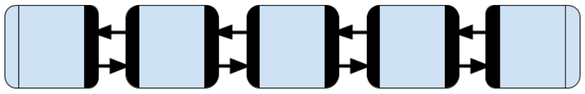
Figure 1. Schematic view of halo exchange in east-west direction

The module implements legacy and best performance MPI point-to-point communication. Message passings are divided into east-west and north-south exchanges and associated communicators are used. Schematic view of east-west halo exchange is depicted on figure 1. It is based on send and receive scheme between neighbouring processes in non-blocking mode:

```
 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(...)
```

III. Synchronization cost

One-sided communication depends on memory regions that are exposed for other processes for remote accesses. Owner of the remote memory is referred to as target process. MPI RMA model provides two general synchronization modes depending on the target process involvement. In active target synchronization mode, the target process explicitly participate in the communication exposing its memory while in passive target mode memory exposure is permanent. In passive target mode, the target process is not responsible for the synchronization.

Active target synchronization uses fence synchronization which is best suited for dynamic memory access patterns and general synchronization which provides more fine-grained memory exposure model. The latter general active target synchronization is often called scalable group synchronization. It allows to define the group of participating processes for each memory exposure epochs and relax synchronization to these groups only. This scheme is implemented with Post-Start-Complete-Wait (PSCW) sequence of RMA operations. This kind of memory access is considered best for computations that rely on relatively static communication schemes such as stencil pattern with halo exchange between neighbour processes.

Belli and Hoefler in [1] define stencil scheme as a case of generic producer-consumer communications. This communications require two basic steps: data transmission and process synchronization. In the message passing model, both steps are realized by the receive operation. In the RMA model these two steps are separated. In their work authors identify minimal cost of RMA synchronization as at least three message transactions, no matter which protocol is considered. Introduction of the Notified Access aims at reducing the number of message transactions required for synchronization. Details of this proposal are described in [1].

IV. Implementation

Following recommendations from [2] general active target synchronization (PSCW scheme) was chosen to implement RMA version of halo exchange. Two reference implementation of PSCW synchronization were considered: Ohio State University (OSU) Micro Benchmarks - put_latency test and Intel Parallel Research Kernels (PRK) [4] - Synch_p2p kernel. The latter one was studied in [1] as stencil code application for notified access performance evaluation. Incremental approach of the one-sided halo exchange module migration was used. That allowed to keep code accuracy monitored and to collect performance changes. Experimental implementation was required to preserve bit-reproducibility of results with original message passing version of the halo exchange.

Naive RMA implementation of the module simply updates message passing send-receive scheme with remote memory post-start-complete-wait scheme:

```
 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, ... )
```

To minimize changes in the module send and receive memory buffers are re-used. These buffers are used to create memory windows for remote accesses. That choice requires creation of a new window for each halo exchange as halo sizes are changing. First optimizations included memory windows and processes groups to be created once as communication is initialized. For this purpose dynamic memory windows are introduced. Every change of the exchange buffers requires now only attaching and detaching memory buffer to the window. That improves code structure and clarity but without significant effect on performance.

Improved version of the module targets at weak synchronization mode. In this case of PSCW scheme, start and complete operations can return while access to remote window is delayed unit corresponding post call is issued. This is allowed by the MPI standard but requires put operation to be buffered on the origin process. Such semantic may lead to more efficient synchronization, as suggested in [3].

Notified Access version is standard MPI incompatible but aims at better scalability. Including Notified Access RMA does not require much changes. What need to be modified is initialization of notification requests objects with a call to MPI_Notify_init and usage of MPI_Put_notify (these functions are not a part of MPI Standard). More difficult is switching to MPI library that supports Notified Access functions. The only available implementation of such library is foMPI-NA from ETH Zurich, described in [1]. This library is a light-weight MPI implementation and is a research prototype.

V. Results

This section summarises experimental results gathered on the Cray system. While the first steps of the RMA implementation of halo exchange were more targeted on correct data exchange and synchronization that not change model results, successive migration steps were focused on performance improvement.

A. System and application setup

Experiments were run on Cray XC40 system with 24-core Intel Haswell nodes connected with a Dragonfly Aries network. Cray programming environment was used with: PrgEnv-cray (5.2.82), cce (8.4.5), cray-mpich (7.3.2) and craype (2.5.3).

Code of the Unified Model was based on version 10.1. Evaluation was using real model setup with a regular grid of dimension 448x616x70.

foMPI-NA library version 0.2.2 (first release) was used.

B. Evaluation results

The model code was running in the same configuration for each test. Each step of the dynamical solver requires, in the test case, at average 170 full halo exchanges in each of east-west (EW) and north-south (NS) direction. As expected naive implementation of the RMA halo exchange performs worse that message passing version due to windows initialization in every exchange and synchronization costs. Results of total runtime of 48 model steps, with average, minimal and maximal time over processes running on 16 XC nodes are shown in table I.

Table I
Results for the first migration step, total time in seconds

	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
One sided	1.89	1.61	2.19	7.58	5.90	9.59

First update of the RMA implementation introduce dynamic windows. In this mode, memory windows are dynamically attached to already allocated memory buffers. This approach allows to eliminate window initialization costs. Still it is requires to compute local buffer relative addresses and to circulate them across remote processes. Despite additional synchronization, this update much improved implementation. Analogous test results for 48 model steps are shown in table II. For the given node number (16 24-core nodes), final performance of the PSCW scheme is still slower that original message passing implementation. For the east-west (EW) exchange the implemented scheme is comparable in performance while for the north-south (NS) part has more complicated pattern and is significantly slower.

Table II
RESULTS FOR VERSION WITH DYNAMIC WINDOWS, TOTAL TIME IN SECONDS

	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
One sided	1.09	0.74	1.29	2.11	0.56	2.75

Second update of the RMA implementation employ foMPI fast one-sided MPI library. In a first step it switches from Cray MPI (MPICH derivative) to foMPI (ETH research implementation) during code building. Next step is overloading MPI RMA calls with specific foMPI calls (basically changing `MPI_` prefix to `foMPI_` in one-sided calls). This change enables use of direct DMAPP implementation of the RMA functions.

Final update use Notified Access mechanism for RMA halo exchange. It follows the 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, ...)
```

Both implementations with foMPI and foMPI-NA improve performance on the isolated halo exchange kernel. Complete model application fails when running with foMPI. This issue still needs to be investigated.

VI. CONCLUSION

This paper describes early work on substituting message passing halo exchange with RMA version in a complex atmospheric application. Described migration from MPI message passing to RMA one-sided model is applied to the Unified Model code and evaluated on Cray XC40 system. Different implementations based on MPI-3 RMA model are discussed. Performance of described RMA implementation, on a given system partition, is comparable with orginal message passing approach. Reported work is on early stage, issues affecting scalability are identified but not addressed. Notified Access approach is not successfully adopted while is not fully supported on the target system.

ACKNOWLEDGMENT

The code of the Unified Model is used under the licence agreement between the MetOffice and the University of Warsaw.

REFERENCES

[1] Roberto Belli and Torsten Hoefler. Notified access: Extending remote memory access programming models for producer-consumer synchronization. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 871–881. IEEE, 2015.

[2] Torsten Hoefler, James Dinan, Rajeev Thakur, Brian Barrett, Pavan Balaji, William Gropp, and Keith Underwood. Remote Memory Access Programming in MPI-3. *ACM Transactions on Parallel Computing*, 2(2):9, 2015.

[3] MPI Forum. MPI: A Message-Passing Interface Standard. Version 3.1, June 4 2015. Available at: `http://www.mpi-forum.org`.

[4] Rob F Van der Wijngaart and Timothy G Mattson. The Parallel Research Kernels. In *HPEC*, pages 1–6, 2014.

[5] Nigel Wood, Andrew Staniforth, Andy White, Thomas Allen, Michail Diamantakis, Markus Gross, Thomas Melvin, Chris Smith, Simon Vosper, Mohamed Zerroukat, et al. An inherently mass-conserving semi-implicit semi-lagrangian discretization of the deep-atmosphere global non-hydrostatic equations. *Quarterly Journal of the Royal Meteorological Society*, 140(682):1505–1520, 2014.