

A critical analysis of the MPI-3 RMA interface

Jeff Hammond

Leadership Computing Facility
Argonne National Laboratory

12 November 2009



The current interface

`MPI_RMA_xfer(rma_optype, origin_addr, origin_count, origin_datatype, target_mem, target_disp, target_count, target_datatype, target_rank, comm, RMA_Attributes, request)`

IN	<code>rma_optype</code>	<code>RMA_PUT</code> , <code>RMA_GET</code> , <code>RMA_ACC...</code>
IN	<code>origin_addr</code>	The local address
IN	<code>origin_count</code>	the number of entries
IN	<code>origin_datatype</code>	datatype of each entry in origin buffer
IN	<code>target_mem</code>	structure representing the target memory being accessed
IN	<code>target_disp</code>	displacement from start of target buffer represented by <code>target_mem</code>
IN	<code>target_count</code>	number of entries in target buffer
IN	<code>target_datatype</code>	datatype of each entry in target buffer
IN	<code>target_rank</code>	rank of target
IN	<code>comm</code>	communicator
IN	<code>rma_attributes</code>	the attributes of this RMA operation
OUT	<code>request</code>	communication request

Additional features not yet specified

- symmetric memory registration across communicator
- pair-wise memory registration between ranks
- (memory de-registration)

Do we need a second interface for lower latency?

A second, stripped-down, interface was proposed and discussed on the RMA list in September.

These issues have been analyzed:

- communication and datatype lookup
- memory registration
- put/get versus accumulation, contiguous versus strided
- remote agents (communication threads to handle some or all RMA operations to ensure one-sided progress) and assertions to disable them

Motivation and objectives

MPI-3 RMA should:

- ① be programmable by average MPI users
- ② not decrease performance of other features of MPI
- ③ be as efficient as possible across the spectrum of its features
- ④ be rich enough to implement Global Arrays. . .
- ⑤ be implementable on current and future architectures independent of hardware support for RMA

For comparison, ARMCI clearly satisfies (2), (3) and (4).

Compliance with (5) is a matter of debate while (1) is clearly an issue.

Acknowledgments

- Pavan Balaji (Argonne) — explained MPICH internals for datatypes, etc. and why second interface is not necessary to reduce latency.
- Dick Treumann (IBM) — raised the issue of remote agents and assertions therewith.
- Keith Underwood (Intel) — raised issues of interface complexity and hardware support.
- Vinod Tipparaju (ORNL) — commented throughout.

All errors are mine.

Does communicator/datatype lookup matter?

MPI_RMA_xfer(rma_optype, origin_addr, origin_count, origin_datatype, target_mem, target_disp, target_count, target_datatype, target_rank, comm, RMA_Attributes, request)

IN	rma_optype	RMA_PUT, RMA_GET, RMA_ACC...
IN	origin_addr	The local address
IN	origin_count	the number of entries
IN	origin_datatype	datatype of each entry in origin buffer
IN	target_mem	structure representing the target memory being accessed
IN	target_disp	displacement from start of target buffer represented by target_mem
IN	target_count	number of entries in target buffer
IN	target_datatype	datatype of each entry in target buffer
IN	target_rank	rank of target
IN	comm	communicator
IN	rma_attributes	the attributes of this RMA operation
OUT	request	communication request

Does communicator/datatype lookup matter?

Easy case for communicator:

- pointer dereference
- bit manipulation
- switch

Easy case for datatype:

- bit manipulation for built-ins
- no remote verification necessary

Hard cases:

- Second interface would not address these
- Remote user-defined datatype confirmation may prevent progress

Memory registration and direct access of local memory?

MPI_RMA_xfer(rma_optype, origin_addr, origin_count, origin_datatype, target_mem, target_disp, target_count, target_datatype, target_rank, comm, RMA_Attributes, request)

IN	rma_optype	RMA_PUT, RMA_GET, RMA_ACC...
IN	origin_addr	The local address
IN	origin_count	the number of entries
IN	origin_datatype	datatype of each entry in origin buffer
IN	target_mem	structure representing the target memory being accessed
IN	target_disp	displacement from start of target buffer represented by target_mem
IN	target_count	number of entries in target buffer
IN	target_datatype	datatype of each entry in target buffer
IN	target_rank	rank of target
IN	comm	communicator
IN	rma_attributes	the attributes of this RMA operation
OUT	request	communication request

Memory registration and direct access of local memory?

- Memory registration requirements vary from non-existent (sockets) to trivial (BGP) to important (IB).
- Pinning huge amounts of memory has negative side effects in some cases.
- Pinning many segments may lead to an expensive lookup to determine prior registration.
- On-the-fly (OTF) registration may be too much overhead. . .
- RMA handle binding can address registration issue.

Breakdown of a PUT operation

OTF registration only makes sense for larger buffers. Short messages should either (1) assert registration or (2) be implemented differently. Replacing `origin_addr` with `origin_mem` would solve implementation issues but place an increased burden on users.

Timing	Operation	Purpose
3830	DCMF_Put_register	register operation attributes
901	DCMF_Memregion_create	register buffer
6105	MPI_Sendrecv	exchange buffer registrations
5753	DCMF_Put	RMA operation
1128	DCMF_Messenger_advance	force local completion
358	DCMF_Memregion_destroy	unregister buffer

Timings are in processor cycles.

Some RMA operations require remote agency

MPI_RMA_xfer(rma_optype, origin_addr, origin_count, origin_datatype, target_mem, target_disp, target_count, target_datatype, target_rank, comm, RMA_Attributes, request)

IN	rma_optype	RMA_PUT, RMA_GET, RMA_ACC...
IN	origin_addr	The local address
IN	origin_count	the number of entries
IN	origin_datatype	datatype of each entry in origin buffer
IN	target_mem	structure representing the target memory being accessed
IN	target_disp	displacement from start of target buffer represented by target_mem
IN	target_count	number of entries in target buffer
IN	target_datatype	datatype of each entry in target buffer
IN	target_rank	rank of target
IN	comm	communicator
IN	rma_attributes	the attributes of this RMA operation
OUT	request	communication request

Some RMA operations require remote agency

- Contiguous PUT/GET may have hardware support, but striding or accumulation probably won't.
- Even if striding can be done without remote agency, it probably helps performance.
- Persistent or wake-able agents (threads?) may be used to handle non-contiguous transfer and accumulation.
- Many MPI implementations already use agents, but RMA may require more remote agents. Strided accumulate, which is critical to Global Arrays, is a challenge on BGP and XT.
- Assertions about remote agents are an implementation detail which shouldn't be in the standard.

Twelve arguments is a lot

MPI_RMA_xfer(rma_optype, origin_addr, origin_count, origin_datatype, target_mem, target_disp, target_count, target_datatype, target_rank, comm, RMA_Attributes, request)

IN	rma_optype	RMA_PUT, RMA_GET, RMA_ACC...
IN	origin_addr	The local address
IN	origin_count	the number of entries
IN	origin_datatype	datatype of each entry in origin buffer
IN	target_mem	structure representing the target memory being accessed
IN	target_disp	displacement from start of target buffer represented by target_mem
IN	target_count	number of entries in target buffer
IN	target_datatype	datatype of each entry in target buffer
IN	target_rank	rank of target
IN	comm	communicator
IN	rma_attributes	the attributes of this RMA operation
OUT	request	communication request

Twelve arguments is a lot

Binding option 1:

```
MPI_RMA_bind(rma_optype, origin_datatype, target_datatype,  
comm, rma_attributes, &handle)
```

```
MPI_RMA_xfer_handle(handle, origin_addr, origin_count,  
target_mem, target_disp, target_count, target_rank)
```

```
MPI_RMA_unbind(handle)
```

Binding option 2:

```
MPI_RMA_bind(rma_optype, origin_addr, max_origin_count,  
origin_datatype, target_datatype, comm, rma_attributes, &handle)
```

```
MPI_RMA_xfer_handle(handle, origin_count, target_mem,  
target_disp, target_count, target_rank)
```

```
MPI_RMA_unbind(handle)
```

Conclusions

- datatype and communicator lookup costs do not justify a second interface
- remote agents are implementation-specific but perhaps still worth consideration for exposure in the standard (via epochs?) if implicit wakeup upon memory registration is not sufficient
- binding addresses many issues but “best” binding interface is not clear
- should we apply same restrictions on MPI_Reduce?
- is explicit supporting for striding necessary?

Performance of RMA on BlueGene/P

