Read Modify Write for MPI

Howard Pritchard (Cray)
Galen Shipman (ORNL)

Motivation

- Expose underlying hardware mechanisms (atomic updates)
- Provide functionality missing in current RMA semantics
  - Applications use this functionality via different APIs (not MPI)

MPI_RMW

```
MPI_Rmw(void *operand_addr, int count,
        MPI_Datatype datatype, void *result_addr,
        int target_rank, MPI_Aint target_disp,
        int assert, MPI_RMW_Op op,
        MPI_Win win)
```

- The target address is updated according to the specified operation using the value at operand_addr as the operand to the operation.
- result_addr is updated with the value at the target address prior to the update.
- Concurrent RMW operations specifying the same (or overlapping) target address are allowed and the updates occur as if the operations occurred in some order. -> Needs further discussion
- By default the operation is non-blocking with the same completion semantics as other RMA operations (assert can be used to modify these semantics and is discussed later)
- Note that operand_addr and target must be the same datatype and count

MPI_RMW_OP

MPI_RMW_INC	increment
MPI_RMW_PROD	product
MPI_RMW_SUM	sum
MPI_RMW_LAND	logical and
MPI_RMW_LOR	logical or
MPI_RMW_LXOR	logical xor
MPI_RMW_BAND	binary and
MPI_RMW_BOR	binary or
MPI_RMW_BXOR	binary xor
MPI_RMW_SWAP	swap value

MPI_RMW_OP

- The predefined MPI_REDUCE operations are the same except for MPI_RMW_INC (special case of MPI_RMW_SUM)
- We can reuse the MPI_REDUCE operations but...
  - Can efficiencies be had (reduced latency) by separating these?
  - Some of these operations are available in hardware, does separating the operations ease exposing of these hardware features?

MPI_RMW2

```
MPI_RMW2(void *operand_addr, int count,
         MPI_Datatype datatype, void *operand2_addr,
         void *result_addr, int target_rank,
         MPI_Aint target_disp, int assert,
         MPI_RMW_2_Op op, MPI_Win win)
```

- The target address is updated according to the specified operation using the values at operand_addr and operand2_addr as the operands to the operation.
- result_addr is updated with the value at the target address prior to the update.
- Concurrent RMW operations specifying the same (or overlapping) target address are allowed and the updates occur as if the operations occurred in some order. -> Needs further discussion
- By default the operation is non-blocking with the same completion semantics as other RMA operations (assert can be used to modify these semantics and is discussed later)
- Note that operand_addr, operand2_addr and target must be the same datatype and count

MPI_RMW2_OP

MPI_RMW2_MASK_SWAP	swap masked bits
MPI_RMW2_COMP_SWAP_LT	compare and swap (<)
MPI_RMW2_COMP_SWAP_LE	compare and swap (<=)
MPI_RMW2_COMP_SWAP_E	compare and swap (==)
MPI_RMW2_COMP_SWAP_GE	compare and swap (>=)
MPI_RMW2_COMP_SWAP_GT	compare and swap (>)
MPI_RMW2_COMP_SWAP_NE	compare and swap (!=)

MPI_RMW2_COMP_SWAP

- Compares the comperand (operand_addr) with the target value if comperand is (<, <=, >, >=, ==, !=) then the target value is updated with the swaperand (operand2_addr) value.
- Result address is updated with the target value prior to the compare/swap.

MPI_RMW2_MASK_SWAP

- Updates the values of target specified in the maskerand (operand_addr) with the corresponding values in the swaperand (operand2_addr).
- Result address is updated with the target value prior to the mask/swap.

A word about epochs

- By default completion semantics follow those of other RMA operations
- When assert==MPI_MODE_PASSIVE_EPOCH_IMPLICIT then MPI_RMW/2 is equivalent to:

```
MPI_Win_lock(MPI_LOCK_EXCLUSIVE, target_rank, 0, win);
MPI_Rmw(… target_rank … win);
MPI_Win_unlock(target_rank, win);
```

- When assert==MPI_MODE_PASSIVE_EPOCH_IMPLICIT | MPI_MODE_NOCHECK the MPI library knows that the window lock at the target need not be acquired

Isn’t this MPI_ACCUMULATE?

- MPI_ACCUMULATE allows combining data movement (PUT) and MPI_REDUCE operations
- MPI_ACCUMULATE does not:
  - “return” the value of the address prior to the update
  - Allow multiple operands for comp/swap mask/swap operations
  - Provide “alternative” epoch semantics
- MPI_RMW/2 does not:
  - Allow differing datatypes for source/target (even datatypes derived from the same predefined datatype)
  - Allow derived datatypes

Open Issues

- Do we need to limit to count == 1?
- What guarantees can we make regarding concurrent “conflicting” RMW operations?