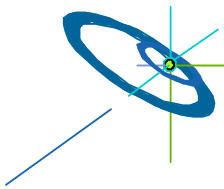
Correctness Checking of MPI One-Sided Communication Using MARMOT

Bettina Krammer,
Michael Resch

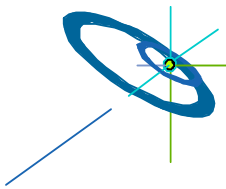
HLRS (High Performance Computing Center Stuttgart)
Allmandring 30

D-70550 Stuttgart
http://www.hlrs.de



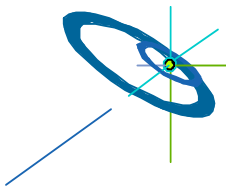
Overview

- Motivation
- Related Work
- MARMOT
- MPI 1-Sided Communication
- Conclusions



Motivation – Problems of Parallel Programming

- All problems of serial programming
- Additional problems:
  - Increased complexity
  - New parallel problems:
    - **deadlocks**
    - **race conditions**
    - **Irreproducibility**
- Portability issues: MPI standard leaves some decisions to implementors, e.g.
  - Message buffering for send/recv operations
  - Synchronising collective calls
  - Implementation of “opaque objects”

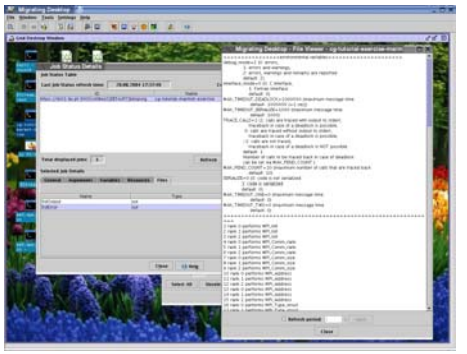


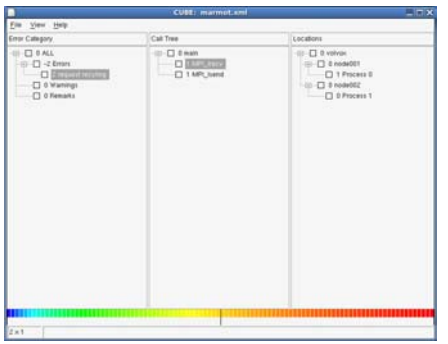
Related Work

- Parallel Debuggers, e.g. totalview, DDT, p2d2
- MPI implementations offer (limited) support, e.g.
  - NEC Collectives Verification Library
    - **Only for NEC MPI/SX, MPI/EX**
  - mpich2 profiling library for collective functions
    - **Checks correctness e.g. of collective calls and datatypes**
    - **Portable library**
- Special verification tools
  - MPI-CHECK
    - **Limited to Fortran, arguments checks, deadlock detection, ...**
  - Umpire
    - **First version limited to shared memory platforms, now also Distributed memory version**
    - **Checks for send/recv pairs, collective calls, ...**
  - MARMOT
- Other tools (post-mortem analysis of trace file, e.g. IMC)

What is MARMOT?

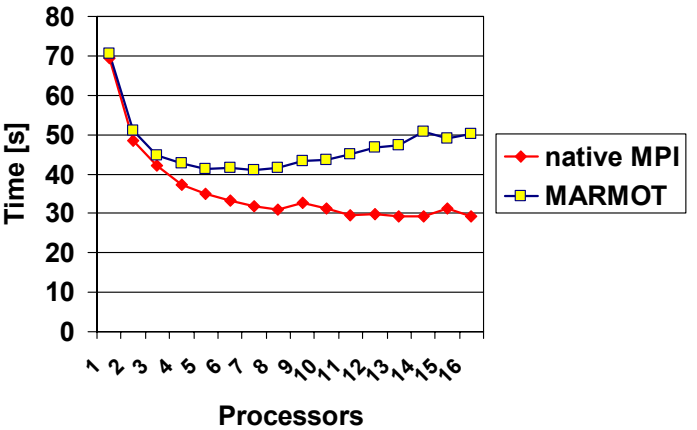
- Automatic runtime analysis of the MPI application:
  - Detect incorrect use of MPI
  - Detect non-portable constructs
  - Detect possible race conditions and deadlocks
- MARMOT does not require source code modifications, just relinking
- C and Fortran binding of MPI -1.2 is supported, also C++ and mixed C/Fortran code, extensions to MPI-2
- Tool makes use of the so-called *profiling interface* PMPI:
  - all MPI resources are mapped to MARMOT resources
  - One additional process (MARMOT's "debug server")
- Environment variables for tool behavior and output (report of errors, warnings and/or remarks, trace-back, etc.)
- Output: human-readable logfile (or other formats)



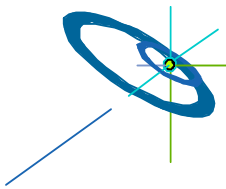


MARMOT

- Used for a number of Fortran and C applications
- Performance sufficient for most applications
- Tests on different platforms, using different compilers and MPI implementations, e.g.
  - IA32/IA64 clusters (Intel, GNU,...) with mpich, lam, vendor MPIs,...
  - IBM Regatta
  - NEC SX8
  - Cray, Hitachi,...
- Development is still ongoing (not every possible functionality is implemented yet…)



MPI One-sided Communication (Remote memory Access RMA)

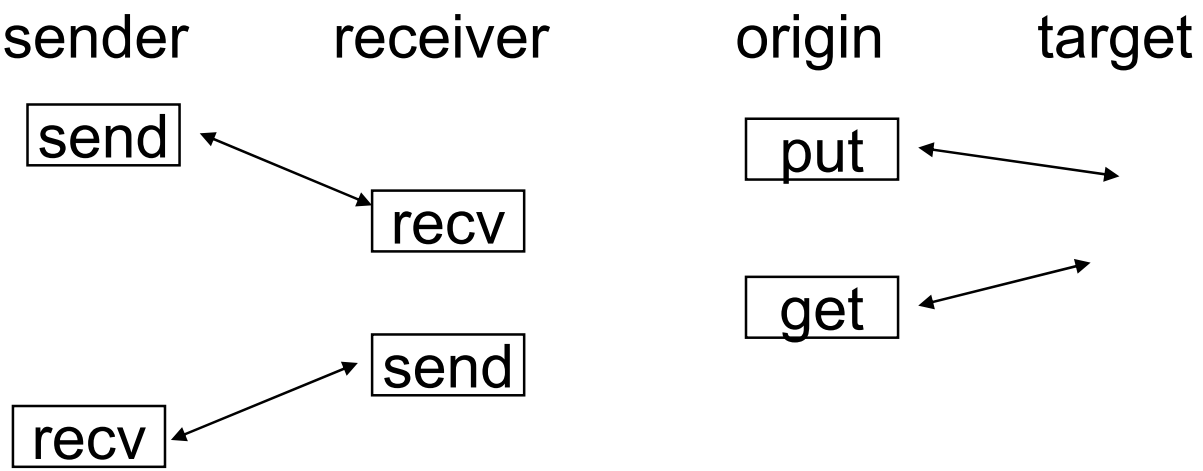


Bettina Krammer
7 Höchstleistungsrechenzentrum Stuttgart

HLRS

MPI Communication

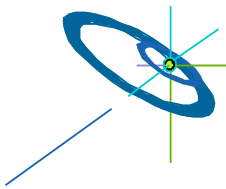
- MPI-1 supports 2-sided communication: Both sender and receiver processes must participate in the communication
- MPI-2 supports 1-sided communication: parameters for both the sender and receiver are specified by one process (origin)



Problems with RMA

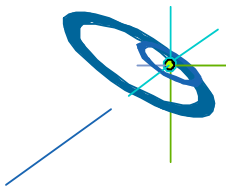
RMA was meant for situations where one process is in control of all send / recv parameters, e.g. when the receiver does not know the size/number/origin of incoming messages, **BUT**:

- Syntax of RMA to be understood first (~30 MPI calls)
- all RMA calls (get, put, accumulate) have to be within *synchronisation epochs* (active sync involves origin and target process)
- User must ensure that there are no conflicting memory accesses (and other pitfalls)



Problems with RMA

- Performance of RMA might be worse (depending on implementation)
  - synchronisation overhead
  - User might serialise the application by synchronisation (e.g. lock/unlock)
  - better performance might be achieved with MPI_ALLOC_MEM
  - better performance might be achieved with **assertions** (used with fence/start/post operations) but might be ignored by MPI implementation
- RMA hardly used by applications

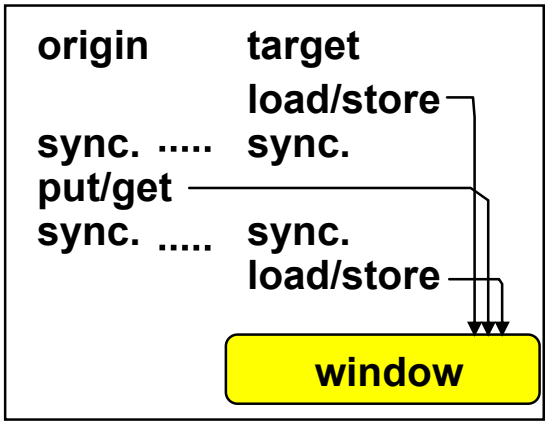


One-sided Operations (MPI-2, ch. 6 and others)

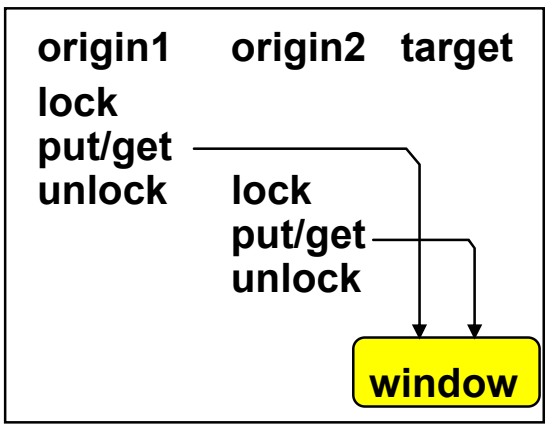
- Initialization (window specifies the region in memory that can be accessed by remote processes)
  - MPI_ALLOC_MEM, MPI_FREE_MEM
  - MPI_WIN_CREATE, MPI_WIN_FREE (collective)
- Remote Memory Access (nonblocking, similar to Send, Recv, Reduce)
  - MPI_PUT
  - MPI_GET
  - MPI_ACCUMULATE (operators from MPI-1, new REPLACE op)
- Synchronization (active/passive target sync)
  - MPI_WIN_FENCE (active sync, collective, Barrier-like, should be used before and after calls to put, get, and accumulate)
  - MPI_WIN_POST / MPI_WIN_START / MPI_WIN_COMPLETE / MPI_WIN_WAIT /MPI_WIN_TEST (active sync)
  - MPI_WIN_LOCK / MPI_WIN_UNLOCK (passive sync)
- Others (MPI_WIN_GET_GROUP, MPI_WIN_GET_ATTR,...)

Synchronization Calls

- Active target communication
  - communication paradigm similar to message passing model
  - target process participates only in the synchronization
  - fence or post-start-complete-wait

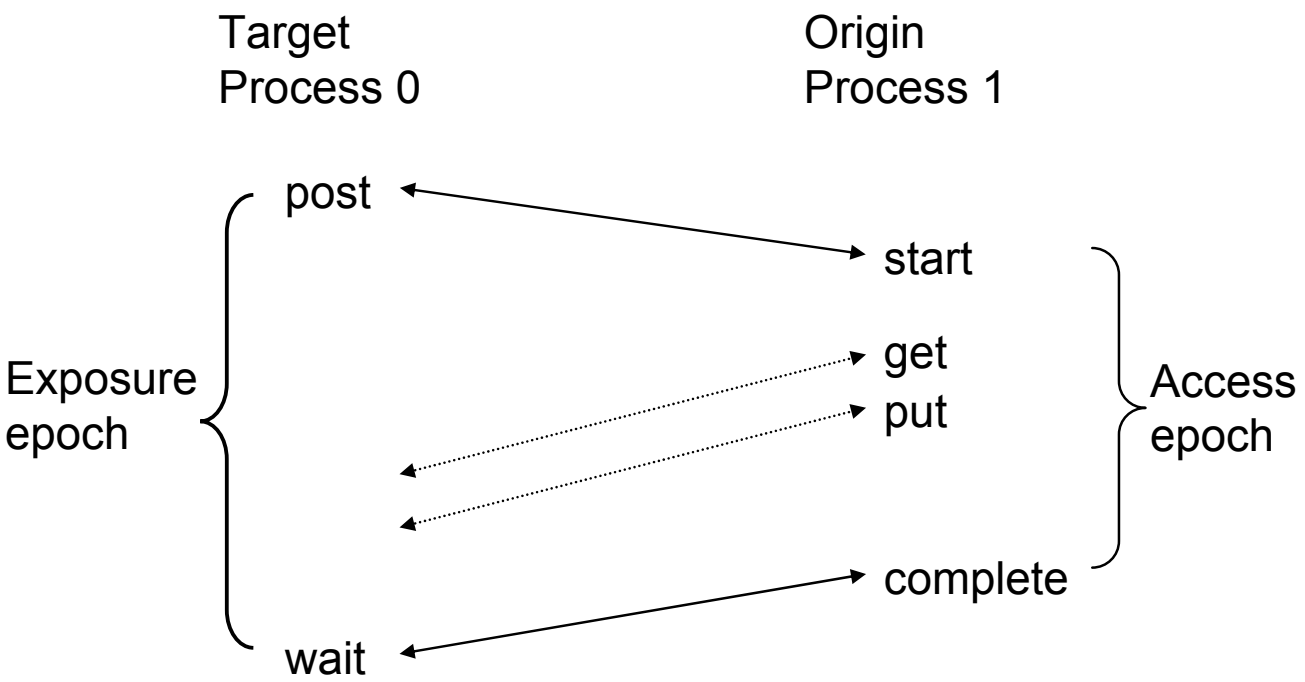


- Passive target communication
  - communication paradigm closer to shared memory model
  - only the origin process is involved in the communication
  - lock/unlock



Example: Start/Complete and Post/Wait

- Used for active target communication with weak synchronization

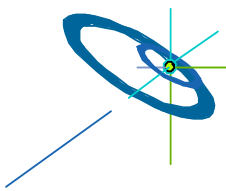


Pitfalls

- Correct construction/destruction of resources
  - create windows, e.g. window size is nonnegative integer, etc.,
  - free window, e.g. window valid, no pending RMA calls
- Collective calls
  - windows creation
  - fence synchronisation

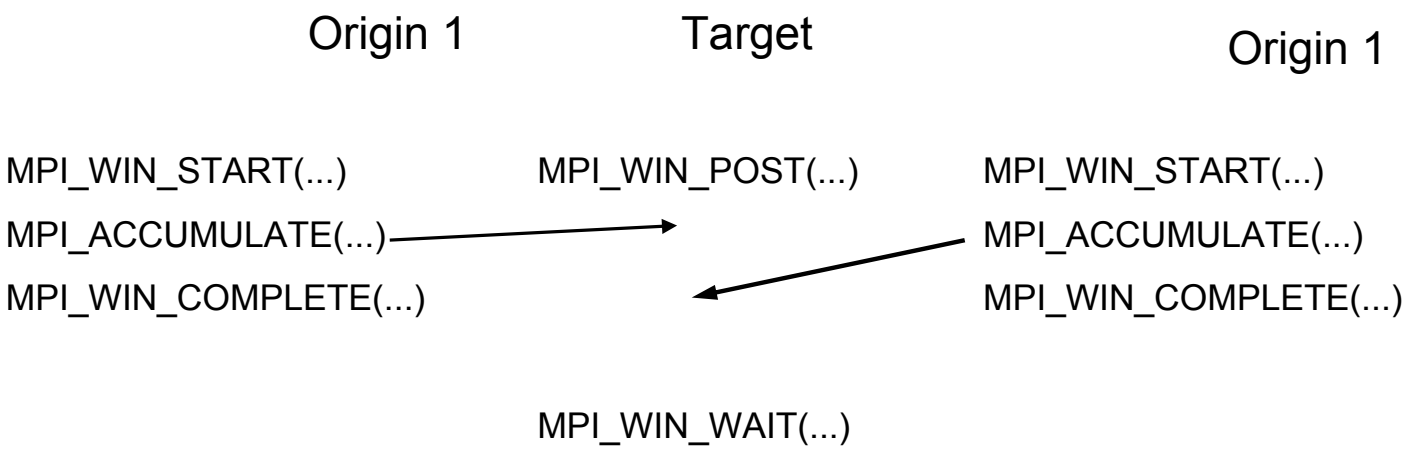
Pitfalls

- RMA calls (get, put, accumulate): correctness of arguments such as counts, datatypes, ranks
- Conflicting RMA get/put or local load/store accesses to the same memory location
- several accumulate operations may update same location in memory (in some serial order)



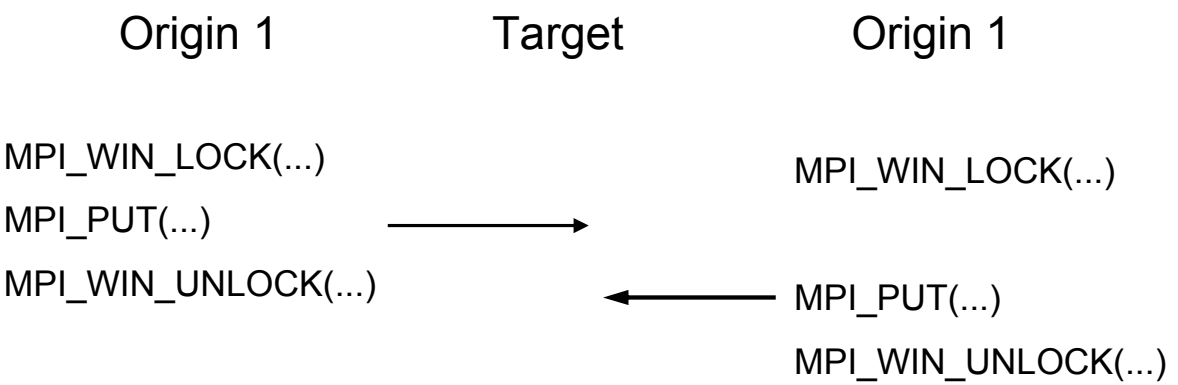
Pitfalls

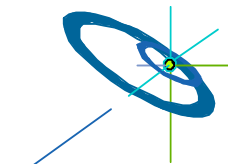
- Matching access/exposure epochs: post/wait/test and start/complete pairs (may also be nested)



Pitfalls

- Matching lock/unlock pairs (shared or exclusive locks)
- Erroneous to call lock on a window in an exposure epoch (on a window that has called post but not wait)

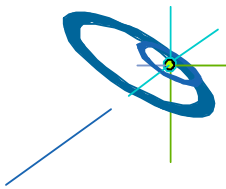




Pitfalls

- Legal assert values (bit vectors of zero or more of) may be used for performance optimisation in start, post, fence, lock calls, e.g.
  - MPI_WIN_LOCK: MPI_MODE_NOCHECK (no conflicting locks from other processes)
  - MPI_WIN_POST: MPI_MODE_NOCHECK (no matching start calls yet on origin processes) , MPI_MODE_NOSTORE (no local stores / get / recv calls since last sync), MPI_MODE_NOPUT (no update by put / accumulate after post until next wait sync)

    NOCHECK must be specified only if specified in each matching start

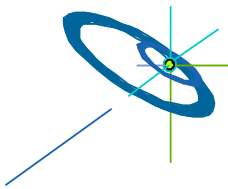


Conclusion

- MPI Programming offers a lot of pitfalls...
- ...but there are tools to help developers

- MARMOT currently supports MPI 1.2 for C and Fortran binding
- Extensions to MPI-2
- Development ongoing
- Some of the already existing functionality can be easily extended to cover MPI 1-sided communciation
- Not all errors user may possibly make can be detected.

- More information
  http://www.hlrs.de/organization/amt/projects/marmot/



Future Directions

Functionality

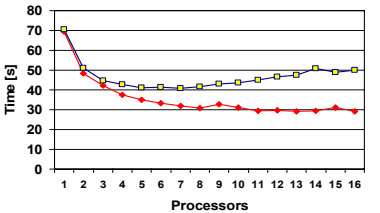
-More checks

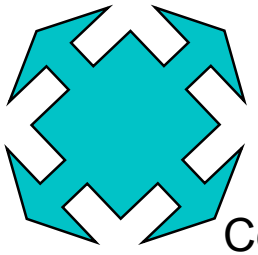
-MPI-2

-OpenMP/MPI

Performance

- Scalability





Usability

- GUI

Combination with other tools

-Debugger

-Vampir