MPI Application Development with MARMOT

Bettina Krammer
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Matthias Müller
University of Dresden
Centre for Information Services and High Performance Computing (ZIH)
www.tu-dresden.de/zih

Outline

• Motivation
• Approaches and Tools
• Examples
• Future work and comparison with other tools
• Conclusion
Motivation

Motivation - Problems of Parallel Programming

• Additional problems:
  – Increased complexity
  – New parallel problems:
    • deadlocks
    • race conditions
    • Irreproducibility
• Portability issues: MPI standard leaves some decisions to implementers, e.g.
  – Message buffering for send/recv operations
  – Synchronising collective calls
  – Implementation of “opaque objects”
(Parallel) Debuggers

- Most vendor debuggers have some support
- Debugging MPI programs with a “serial” debugger is hard but possible
  - MPICh supports debugging with gdb attached to one process
  - manual attaching to the processes is possible
- Commercial debuggers: Totalview, DDT
Special MPI tools & libraries

- 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 of collective calls and datatypes
    - Portable library
- Tools specially dedicated to analysis of MPI applications:
  - MPI-Check
  - Umpire
  - IMC
  - Marmot

MARMOT
MPI Analysis and Checking Tool

Bettina Krammer

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de
What is MARMOT?

- Tool for the development of MPI applications
- Automatic runtime analysis of the 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
- Development is still ongoing (not every possible functionality is implemented yet…)
- Tool makes use of the so-called profiling interface

Design of MARMOT

Application or Test Program → Profiling Interface → MARMOT core tool → MPI library → Debug Server (additional process)
Examples of Server Checks: verification between the nodes, control of program

- Everything that requires a global view
- Control the execution flow, trace the MPI calls on each node throughout the whole application
- Signal conditions, e.g. deadlocks (with traceback on each node.)
- Check matching send/receive pairs for consistency
- Check collective calls for consistency
- Output of human readable log file

Examples of Client Checks: verification on the local nodes

- Verification of proper construction and usage of MPI resources such as communicators, groups, datatypes etc., for example
  - Verification of MPI_Request usage
    - invalid recycling of active request
    - invalid use of unregistered request
    - warning if number of requests is zero
    - warning if all requests are MPI_REQUEST_NULL
  - Check for pending messages and active requests in MPI_Finalize
- Verification of all other arguments such as ranks, tags, etc.
Availability of MARMOT

- Tests on different platforms, using different compilers (Intel, GNU,...) and MPI implementations (mpich, lam, vendor MPIs,...), e.g.
  - IA32/IA64 clusters
  - Opteron clusters
  - Xeon EM64T clusters
  - IBM Regatta
  - NEC SX5,..., SX8

- Download and further information
  http://www.hlrs.de/organization/tsc/projects/marmot/

Examples
Example - Medical Application B_Stream

- Calculation of blood flow with 3D Lattice-Boltzmann method
- 16 different MPI calls:
  - MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Pack, MPI_Bcast, MPI_Unpack, MPI_Cart_create, MPI_Cart_shift, MPI_Cart_rank, MPI_Send, MPI_Recv, MPI_Barrier, MPI_Reduce, MPI_Sendrecv, MPI_Wtime, MPI_Finalize
- Around 6500 lines of code
- We use different input files that describe the geometry of the artery: tube, tube-stenosis, bifurcation

Example: B_Stream (blood flow simulation, tube)

- Tube geometry: simplest case, just a tube with about the same radius everywhere
- Running the application without/with MARMOT:
  
  mpirun -np 3 B_Stream 500. tube
  mpirun -np 4 B_Stream_marmot 500. tube

- Application seems to run without problems
Example: B_Stream (blood flow simulation, tube-stenosis)

- Tube-stenosis geometry: just a tube with varying radius
- Without MARMOT:
  $\text{mpirun -np 3 B\_Stream 500. tube-stenosis}$
- Application seems to be hanging
- With MARMOT:
  $\text{mpirun -np 4 B\_Stream\_marmot 500. tube-stenosis}$
- Deadlock found

---

Example: B_Stream (blood flow simulation, tube-stenosis)

*WARNING: all clients are pending!*
Example: B_Stream (blood flow simulation, tube-stenosis)

decklock

Node 0

timestamp= 9319: MPI_Sendrecv(*sendbuf, sendcount = 7220, 
    sendtype = MPI_DOUBLE, dest = 1, sendtag = 1, *recvbuf, 
    recvcount = 7220, recvtype = MPI_DOUBLE, source = 2, recvtag 
    = 1, comm = self-defined communicator, *status)
timestamp= 9321: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9328: MPI_Sendrecv(*sendbuf, sendcount = 7220, 
    sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf, 
    recvcount = 7220, recvtype = MPI_DOUBLE, source = 1, recvtag 
    = 1, comm = self-defined communicator, *status)
timestamp= 9322: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9324: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
timestamp= 9325: MPI_Bcast(*buffer, count = 3, datatype = 
    MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
Node 1

timestamp= 9322: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9324: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
timestamp= 9325: MPI_Bcast(*buffer, count = 3, datatype = 
    MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
Node 2

timestamp= 9323: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9326: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
timestamp= 9327: MPI_Bcast(*buffer, count = 3, datatype = 
    MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)

Example: B_Stream (blood flow simulation, tube-stenosis) – Code Analysis

main {
    ...
    num_iter = calculate_number_of_iterations();
    for (i=0; i < num_iter; i++) {
        computeBloodflow();
    }
    writeResults();
    // communicate results with neighbors
    MPI_Bcast(…);
    ...
    // exchange results with neighbors
    MPI_Sendrecv(...);
   (MPI_Barrier(...);
    // ERROR: it is not ensured here that all
    // procs do the same (maximal) number
    // of iterations
    if (radius < x) num_iter = 50;
    if (radius >= x) num_iter = 200;
    CalculateSomething();
    // exchange results with neighbors
    MPI_Sendrecv(...);
    MPI_Barrier(...);
    // communicate results with neighbors
    MPI_Bcast(…);

Be careful if you call functions with hidden MPI calls!
Example: B_Stream (blood flow simulation, bifurcation)

- Bifurcation geometry: forked artery
- Without MARMOT:
  
  \[
  \text{mpirun -np 3 B\_Stream 500. bifurcation}
  \]

  ...  
  
  Segmentation fault  
  (platform dependent if the code breaks here or not)

- With MARMOT:

  \[
  \text{mpirun -np 4 B\_Stream\_marmot 500. bifurcation}
  \]

- Problem found at collective call MPI\_Gather

Example calls on node 0:

\[
\begin{align*}
\text{timestamp} & = 9327: \text{MPI\_Bcast(*buffer, count = 3, datatype = MPI\_DOUBLE, root = 0, comm = MPI\_COMM\_WORLD)} \\
\text{timestamp} & = 9330: \text{MPI\_Bcast(*buffer, count = 3, datatype = MPI\_DOUBLE, root = 0, comm = MPI\_COMM\_WORLD)} \\
\text{timestamp} & = 9333: \text{MPI\_Gather(*sendbuf, sendcount = 266409, sendtype = MPI\_DOUBLE, *recvbuf, recvcount = 266409, recvtype = MPI\_DOUBLE, root = 0, comm = MPI\_COMM\_WORLD)}
\end{align*}
\]

Example calls on node 1:

\[
\begin{align*}
\text{timestamp} & = 9334: \text{MPI\_Gather(*sendbuf, sendcount = 258336, sendtype = MPI\_DOUBLE, *recvbuf, recvcount = 258336, recvtype = MPI\_DOUBLE, root = 0, comm = MPI\_COMM\_WORLD)} \\
\text{timestamp} & = 9336: \text{MPI\_Sendrecv(*sendbuf, sendcount = 13455, sendtype = MPI\_DOUBLE, dest = 0, sendtag = 1, *recvbuf, recvcount = 13455, recvtype = MPI\_DOUBLE, source = 0, recvtag = 1, comm = self-defined communicator, *status)}
\end{align*}
\]

\text{ERROR: Root 0 has different counts than rank 1 and 2}
MARMOT
Performance with real applications

- Air pollution modeling with STEM-II model
- Transport equation solved with Petrov-Crank-Nikolson-Galerkin method
- Chemistry and Mass transfer are integrated using semi-implicit Euler and pseudo-analytical methods
- 15500 lines of Fortran code
- 12 different MPI calls:
  - MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Type_extent, MPI_Type_struct, MPI_Type_commit, MPI_Type_hvector, MPI_Bcast, MPI_Scatterv, MPI_Barrier, MPI_Gatherv, MPI_Finalize.
5. Parallel debugging and TotalView

- STEM application on an IA32 cluster with Myrinet

Comparison with other approaches and Future Directions
Future Direction of MARMOT

- MARMOT will continue to be developed jointly by ZIH and HLRS

Future Directions

Functionality
- More checks
- MPI-2
- OpenMP/MPI

Usability
- GUI

Performance
- Scalability

Combination with other tools
- Debugger
- Vampir
Online vs. Offline Checking

- Offline Checking (Intel Message Checker):
  + History information available
  + Less intrusive at runtime
  - Large file space needed for trace file
  - Limited scalability of analysis
- Online Checking (Marmot):
  - Reduced performance of running application
  - Only limited history/time line information
  + No limit for the runtime of the program
  + The program is still „alive“ when the error is detected, further online analysis is possible

MPI implementation with checking

- Scalability and performance
- Avoids duplicating internal management tasks
- Avoids re-implementation of existing functionality
- No history/time line information
- Performance and reliability is always more important than checks
- Portability problems are not a major focus
Comparison of systems

<table>
<thead>
<tr>
<th></th>
<th>Online</th>
<th>Offline</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks</td>
<td>++</td>
<td>++</td>
<td>0</td>
</tr>
<tr>
<td>Scalability and Performance</td>
<td>+/-</td>
<td>+/-</td>
<td>++</td>
</tr>
<tr>
<td>Runtime scalability</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Timeline information</td>
<td>-</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Combination with other tools</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Conclusion

• Parallel programming in general and the MPI standard contains enough pitfalls to raise a need for checking/correctness/confidence tools
• Different tools and approaches with different advantages and disadvantages
• MARMOT is a freely available solution that has demonstrated its usefulness with various real applications
• A combination of tools will offer the best solution for the program developer
• Not every error can be detected by tools
Thanks for your attention