An Interface for Integrated MPI Correctness Checking

ParCOMA
Parallel Programming for Multi-core Architectures
www.parma-itea2.org

ParCO 2009: Hilbrich, Jurenz, Mix, Knüpfer, Brunst, Müller, Nagel
Content

• Introduction to MPI Correctness Checking
• Motivation
• UniMCI
• Marmot and UniMCI
• VampirTrace and UniMCI
• Example
• Performance Results
Various runtime MPI Correctness Tools exist:
- E.g.: MPI-Check, ISP, MPIDD, Umpire, and Marmot

Detect usage error of MPI, e.g.:

```c
MPI_Type_contiguous(2, MPI_INT, &newtype);
MPI_Send(buf, count, newtype, target, tag ,MPI_COMM_WORLD);
```

These tools intercept MPI calls to perform checks:
- Possible by using the MPI Profiling Interface

Tool is linked to application as a library:
Motivation

– A multitude of MPI tools exists
– Sometimes users need multiple tools at once
  • E.g.: Correlate an MPI usage error with an event trace
    => Requires an MPI correctness and a trace tool
– Users are reluctant to compile, link, and run multiple times
– Correctness tools usually provide no detailed history for detected errors
– We address:
  • Tool combinations where MPI correctness tools are utilized
  • How to execute both MPI tools at the same time
  • How to combine tool outputs
UniMCI – Overview

- UniMCI: Universal MPI Correctness Interface
- Provides MPI correctness checking to MPI tools, independent of the actual MPI correctness tool
- Allows MPI tools to utilize all MPI correctness tools that support UniMCI:

Without UniMCI

Host Tool

Guest Tool 1 (Correctness Tool)

Guest Tool N

One tool integration for each correctness tool

With UniMCI

Host Tool

UniMCI

Guest Tool 1 (Correctness Tool)

Guest Tool N

One integration with UniMCI
UniMCI – Multiple MPI Tools Issue

– MPI Profiling Interface may only be used by one tool
   ➢ Host and guest tool can’t both intercept MPI calls
– UniMCI offers two solutions

(A) Name-shifted Interface
  • Host tool calls extra functions provided by UniMCI
  • E.g.:

    ```
    MPI_X(...) {
        UniMCI_check_pre__MPI_X(...) 
        P_MPI_X(...) 
        UniMCI_check_post__MPI_X(...) 
    }
    ```

(B) Usage of P\(^n\)MPI
  • P\(^n\)MPI forwards to both tools:
  • Currently as prototype
UniMCI – Correctness Events

– Host tool needs to handle detected MPI usage errors
– Errors need to be returned before they cause a crash
– UniMCI uses a simple message record with:
  • Event Type (Note, Warning, Error)
  • Info about the MPI call causing the event
  • Textual error description
  • A reference to the MPI Standard (optional)
  • List of involved ranks (for non local errors)
– Messages retrieved as follows:
  • Name-shifted interface: host queries after the check calls
  • P^nMPI: host is informed via a callback function
Marmot and UniMCI

– First correctness tool with UniMCI support
– Marmot primarily checks for process local errors
  • Invalid arguments
  • Non-portable constructs
  • Erroneous usage of MPI resources (e.g. requests)
– Global checks executed on a dedicated process
  • Disabled to provide a more lightweight Marmot

![Diagram of Marmot and UniMCI interfaces]
VampirTrace and UniMCI

– VampirTrace: a performance event tracing tool
– VT uses UniMCI to highlight correctness events
– UniMCI installation detected with “unimci-config” tool:
  • Gives information about required compilers, flags, and libraries
– VT uses the name-shifted interface
– Easy integration, as most VT wrappers generated
– Correctness messages are stored as markers
– Markers are displayed in VampirServer
Example – Overview

- **Artificial example with a correctness error**
- **Communication pattern:**

  Receives an index used in the next send call

  **Uses index to select datatype:**

  ```c
  MPI_Send (... types[index], ...);
  ```

- Repeated in multiple iterations
- Afterwards slaves send further message to master
Example – Execution

– Execution on an Altix4700 with SGI MPT:

$ mpirun -np 3 example.exe
> I am rank 2 of 3 processes
> I am rank 1 of 3 processes
> I am rank 0 of 3 processes
> Rank 1 of 3 finished!
> MPI: On host neptun, Program /.../example.exe, Rank 0, Process 8570
> received signal SIGSEGV(11)

– With Marmot:

- Send call from Master that uses index
- Obvious: index is wrong
- But, why? Marmot provides no insight!
- Problem: correctness tool provides no history of the events that lead to an error!
Example – VT+UniMCI+Marmot

– Still crashes with VT, but trace contains details:
Performance Results

– Performance tests with SPEC MPI2007 to:
  • Demonstrate applicability to various applications
  • Provide data on performance overheads
Conclusions

– UniMCI provides an interface for MPI correctness checking
– Usage with name-shifted interface or P^nMPI
– Marmot implements UniMCI
– VampirTrace uses UniMCI to provide MPI checking
– Visualization of correctness events with markers
– Performance impact of VT + UniMCI + Marmot combination acceptable for a range of application