Deterministic Replay for MCAP1 Programs

Mohamed Elwakil and Zijiang James Yang

Department of Computer Science
Western Michigan University

PADTAD’11
Toronto, ON, Canada
July 17, 2011
The Megahertz Race

Intel CPU Trends

(sources: Intel, Wikipedia, K. Olukotun)

Credit: National Instruments
Multicore Processors

Intel Core 2 Duo (Dual Core) - 2006

Intel Core i7 (Quad Core) - 2008
Concurrent Programs
Shared-memory and Message-passing

- Task 1
- Task 2
- Task 3
Shared-memory vs. Message-passing

• Message-passing:
  – “that other model for concurrent programming”
  – When shared-memory is not possible.
Shared-memory vs. Message-passing

• Message-passing:
  – “that other model for concurrent programming”
  – When shared-memory is not possible.

• A Case Study of Shared-memory and Message-passing [MIT 1995]:
  • Message-passing is 52% faster when processes>4
  • When using fast communication channel.
  • Locks and barriers slows down shared-memory.
Shared-memory vs. Message-passing

- **Message-passing:**
  - “that other model for concurrent programming”
  - When shared-memory is not possible.

- A Case Study of Shared-memory and Message-passing [MIT 1995]:
  - Message-passing is **52%** faster when processes>4
  - When using fast communication channel.
  - Locks and barriers slows down shared-memory.

- The case for message passing on many-core chips [Intel & UIUC 2010]:
  - “Message-passing is more suitable for the multicore era”.
  - Coherence Wall
New API’s

• MCA MCAPI, 2008:
  – MCA: Intel, AMD, Samsung, Siemens, NI, TI, CMU, UoH.
  – General purpose message-passing API
  – Lightweight, assumes fast and reliable interconnects.
  – Two implementations: MCA’s, Mentor Graphics

• Intel RCCE:
  – Message-passing API for Intel SCC (48 cores).

• API’s and language-extensions are necessary, but not sufficient.
Debugging Message-passing Programs

• Challenging
  – Due to inherent non-determinism

• Neglected!
  – Most work on shared-memory.

• We can help!
  – Techniques and tools for supporting message-passing programs debugging.

• MCAPI
MCAPI

- **MCAPI Node**
  - Process, thread...

- **MCAPI Endpoint**
  - Communication termination point.
  - A tuple of <node id, port id>
  - Has a FIFO buffer

- **API:**
  - Node initialization and finalization.
  - Endpoint creation and deletion.
  - Sending and receiving messages
## MCAPI API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send (Data, Src_EP, Dest_EP)</td>
<td>Sends data from a source endpoint to a destination endpoint.</td>
</tr>
<tr>
<td>Recv (&amp;Data, Recv_EP)</td>
<td>Retrieves a message from an endpoint buffer.</td>
</tr>
<tr>
<td>Send_i (Data, Src_EP, Dest_EP, &amp;Request)</td>
<td>Non-blocking calls.</td>
</tr>
<tr>
<td>Recv_i (&amp;Data, Recv_EP, &amp;Request)</td>
<td></td>
</tr>
<tr>
<td>Wait (Request)</td>
<td>Checks the completion of the request.</td>
</tr>
<tr>
<td>Test (Request)</td>
<td></td>
</tr>
<tr>
<td>Waitany (Requests)</td>
<td></td>
</tr>
</tbody>
</table>
MC-API sources of non-determinism

- Promiscuous `Recv`
- Non-blocking `Test`
- Blocking `Waitany`

```c
N0:
Send(Data, N0_EP, N2_EP);

N1:
Send(Data2, N1_EP, N2_EP);
while (!Test(Request))
N2{A++;}
Recv(&A, N2_EP);
Recv(&B, N2_EP);

Index=Waitany(Reqs);
Func(Index);
```
Reproducing Bugs - Deterministic Replay

- A controlled execution that is logically equivalent to a previous execution of interest.

Recording:

Replay:

- Recording Environment
- Program
- Output
- Trace
- Input

- Replay Environment
- Program
- Output
- Input
Replay Approaches

• Data Replay
  – Record message contents
  – Simulate receiving a message

• Order Replay
  – Record non-deterministic operations outcome.
  – Enforce these outcomes during replay.
During Runtime

Instrumented Program

dr_XXX()

... mcapi_XXX(); ...

} DR-MCAPI

mcapi_XXX()

... ...} MCAPI

mcapi_XXX()
Receiver-based Order Replay

• Recording:
  – When a function call F, retrieves message M:
    • H=Hash(M)
    • Msg(F)=H

• Replay:
  – When a function call F is invoked
    • Find H=Msg(F)
    • Only message M with Hash(M)=H is returned to F.
## Recording in Receiver-based Order Replay

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A = 10);</td>
<td>(X = \text{Recv}());</td>
<td>(B = 1);</td>
<td>(U = \text{Recv}());</td>
</tr>
<tr>
<td>(\text{Send}(A,N2); // M0)</td>
<td>(Y = \text{Recv}());</td>
<td>(\text{Send}(B,N2); // M3)</td>
<td>(W = \text{Recv}());</td>
</tr>
<tr>
<td>(\text{Send}(A,N4); // M1)</td>
<td>(Z = X - Y);</td>
<td>(\text{Send}(B,N4); // M4)</td>
<td>(O = \text{Recv}());</td>
</tr>
<tr>
<td></td>
<td>(\text{Send}(Z,N4); // M2)</td>
<td></td>
<td>(\text{Assert}(U &gt; 0);)</td>
</tr>
</tbody>
</table>

### Trace

<table>
<thead>
<tr>
<th>Node 2</th>
<th>Node 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:2/1/H0</td>
<td>R:4/1/H1</td>
</tr>
<tr>
<td>R:2/2/H3</td>
<td>R:4/2/H2</td>
</tr>
<tr>
<td></td>
<td>R:4/3/H4</td>
</tr>
</tbody>
</table>
### In Receiver-based Order Rule

**Node 2**

- `A=10;`
- `Send(A,N2); // M0`
- `Y=Recv();`
- `Send(A,N4 );// M1`
- `Z=X-Y;`
- `Send(Z,N4); // M2`

**Node 4**

- `B=1;`
- `Send(B,N2); // M3`
- `Send(B,N4 );// M4`
- `O=Recv(); Assert(O>0);`

**Trace**

- **Node 2**
  - R:2/1/H0
  - R:2/2/H3
- **Node 4**
  - R:4/1/H1
  - R:4/2/H2
  - R:4/3/H4

**ReceivedMessages**

- M3 is already in ReceivedMessages
- M4 is already in ReceivedMessages
- M2 is already in ReceivedMessages
Other sources of non-determinism

• **Waitany (Request *Reqs)**
  – Store the index returned during the recording phase.
  – Force the request associated with that index to complete and ignore other requests, during the replay phase.

• **Test(Request Req)**
  – Let $x$ be the number of failed invocations during the recording phase.
  – Automatically return false $x$ then force the request to complete.
Runtime Overhead

**BT**

**CG**

**TN**

**Bully**
Related Work

• An Integrated Record & Replay Mechanism for Nondeterministic Message Passing Programs [2001]
  – 3x runtime overhead

• Parallel Program Debugging based on Data-Replay [2005].

• MPIWiz [2009]
  – Doesn't handle MPI_Test
  – MPI-specific
Conclusion

• Message-passing software development will be as important as shared-memory software
  – Debugging support will encourage programmers to make the switch

• Future Work:
  – Trace compression
  – Check-pointing
Acknowledgements

• This research is supported in part by:
  – NSF Grant CCF-0811287
  – ONR Grant N000140910740
Questions?