MCC: A dynamic verification tool for MCAPI
user applications

Subodh Sharma, UofU
Ganesh Gopalakrishnan, UofU
Eric Mercer, BYU
Jim Holt, Freescale

Supported by NSF CCF 0903408/0903491 and SRC 2009-TJ-1993/1994
Concurrent Space in Multicore Era

<table>
<thead>
<tr>
<th></th>
<th>S/W solution</th>
<th>H/W solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cloud/HPC Computing</strong></td>
<td>MPI, PVM, Mapreduce, MSCS, Dryad</td>
<td>Clusters, Vector machines, Supercomputers</td>
</tr>
<tr>
<td><strong>Desktop computing</strong></td>
<td>OOSD, Scripting, Pthreads, TBB, CT, OpenMP</td>
<td>Desktop machines</td>
</tr>
<tr>
<td><strong>Embedded Computing</strong></td>
<td>Lightweight counterparts of the above.</td>
<td>FPGA, DoC, SoC, etc.</td>
</tr>
</tbody>
</table>
Formalize Emerging Communications API in the Embedded Space

<table>
<thead>
<tr>
<th></th>
<th>S/W solution</th>
<th>H/W solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud/HPC Computing</td>
<td>MPI, PVM, Mapreduce, MSCS</td>
<td>Clusters, Vector machines, Supercomputers</td>
</tr>
<tr>
<td>Server/Desktop</td>
<td>OOSD, Scripting, Pthreads, TBB, CT, OpenMP</td>
<td>Desktop machines</td>
</tr>
<tr>
<td>computing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded Computing</td>
<td>Lightweight counterparts of the above.</td>
<td>FPGAs, DoC, SoC, etc.</td>
</tr>
</tbody>
</table>

- **Formalize Standards, build Query Oracle, Derive Tests**
- **Build Dynamic Formal Verifier for Applications**
- **Demonstrate and Evaluate Prototype Solutions**
MCAPI (Multicore Communication API)
Introduction
What is MCAPI (Multicore Communication API)?

- An API specification from MCA (Multicore Association)
  - Member companies – Freescale, Samsung, Intel, etc.

- To program embedded systems like mobile phones, PDAs, routers, servers, etc.

- Not restricted to SPMD (like MPI) or multi threaded style of programming.
Taxonomy for MCA APIs

**MCAPI – Communication API**
- Message based
- Packet/Scalar Channel based

**MTAPI – Task Management API**
- Specification work yet to begin
- Thread Pooling, Work Stealing queues e.g. CILK, TBB, TPL etc.

**MRAPI – Resource Management API**
- Semaphores, mutexes
- Shared memory segment allocation, deallocation

Inter API interactions
Example usage scenario of MCA APIs
MCAP Messages

- Message received at FIFO ordered receive queue

Node 1
- Port 1 <1,1>
- Port 2 <1,2>

To <2,1>

Node 2
- Port 1 <2,1>

Endpoint:
Node_id, Port_id
MC-API Connection Oriented Communication

- Sent over a connected channel
- \( <1,1> \) can communicate on ONE channel only

Node 1
- Port 1 \( <1,1> \)
- Port 2 \( <1,2> \)

Node 2
- Port 1 \( <2,1> \)

Node 3
- Port 1 \( <3,1> \)

ERROR!!

To \( <2,1> \)
To \( <3,1> \)
Our Contributions!
Early Engagement of FV in the MCAPI space

- To promote early adoption of the API
- To promote better programming practices
- To help avoid early pitfalls – e.g. unspecified behaviors
MCC - MCAPI Checker

MCAPIC Program → instrumentation → Instrumented Program

MCAPIC Library Wrapper

MCAPIC Program → instrumentation → Instrumented Program → compile → Executable

Executable

thread 1

thread n

Scheduler

RUNTIME

request/permit

request/permit

....
Related Work

• Tools in this domain work on directly work on unmodified sources - inspired by Verisoft.

• Tools control the scheduling to achieve the goal of coverage guarantees

• Examples are CHESS from Microsoft, Inspect and ISP from Univ of Utah, etc.
Unique features of MCC

• External schedulers may not be able to exercise control over runtime.
  • Novel way of enforcing deterministic match of transitions at runtime

• Instruments Pthread calls (i.e. would support hybrid programs written in future using MCAPI)
**MC-API API calls supported by MCC**

<table>
<thead>
<tr>
<th>Call Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize/Finalize</td>
<td>Sets and deletes the environment. Must be called by each communicating node.</td>
</tr>
<tr>
<td>Create/Delete endpoint</td>
<td>API calls to manage creation/deletion of endpoints on the owner node</td>
</tr>
<tr>
<td>Get_endpoint</td>
<td>A blocking call to get the address of a remote endpoint.</td>
</tr>
<tr>
<td>Send (sndEp, rcvEp, ..)</td>
<td>Sends a data-payload from a sending endpoint to a receiving endpoint. API provides blocking and non blocking versions.</td>
</tr>
<tr>
<td>Recv (rcvEp, ..)</td>
<td>Receives a data payload from the receiving endpoint. API provides blocking non-blocking versions.</td>
</tr>
<tr>
<td>Wait (reqHndl, ..), Test(reqHndl,..)</td>
<td>Checks the completion of the request. Wait – Blocking; Test – Non blocking</td>
</tr>
</tbody>
</table>
Bugs in the application space?
Why Dynamic Analysis?
Why Deterministic Execution Control?
Non-deterministic MCAPI Receive Calls

- The recv() call is passed with only receiving endpoint as a parameter.
- The receiving endpoint extracts message from the FIFO receive queue.

```
T0
...
T1
...
T2
...
```

Send(from e1 to e2, d1)  Send(from e1 to e2, d2) Recv( from e2 in d3)

If( d3 == 22)  
  Recv(from e2)  
Else  
  ERROR

The send that matches the recv may decide the correctness !!

POTENTIAL BUG
Mismatched Parameters

```
T0
---
Create_Ep (0,1)
R(<0,1>)

T1
---
Get_Ep(<0,2>)
S(1,<0,2>)
```

"get_endpoint" requested for a non-existent endpoint -- ERROR
Challenge: Exponential Interleavings

Dependent MCAPI calls

Only these 2 are RELEVANT!!!
Dynamic Interleaving Reduction

- Dynamic reduction
  - Transition dependency at runtime
  - precise information \(\rightarrow\) effective reduction

Dynamically:
- Discover all potential senders
- Match \texttt{Recv} with each sender
- Recurse through all such configurations

\text{E1} == \text{E2} == \text{E3}?
If yes, extra interleaving
Relevant interleaving exploration by MCC scheduler

T0  T1  T2

S(\text{ep1, ep3})  S(\text{ep2, ep3})  R(\text{ep3})

Intra happens-Before Edge

R(\text{ep3})
Relevant interleaving exploration by MCC scheduler

Interleaving 1

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (ep1, ep3)</td>
<td>S (ep2, ep3)</td>
<td>R (ep3)</td>
</tr>
</tbody>
</table>

Issue to the runtime

Match sets are computed at "decision points"

Set of matching transitions

All threads are blocked
Relevant interleaving exploration by MCC scheduler

Interleaving 1

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(ep1,ep3)</td>
<td>S(ep2,ep3)</td>
<td>R(ep3)</td>
</tr>
</tbody>
</table>

Issue to the runtime
Relevant interleaving exploration by MCC scheduler

Interleaving 2

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (ep1, ep3)</td>
<td>S (ep2, ep3)</td>
<td>R (ep3)</td>
</tr>
</tbody>
</table>

Enabled Transitions: S(ep1, ep3), S(ep2, ep3), R (ep3)

Match Set: <S(ep2, ep3), R (ep3)>
Relevant interleaving exploration by MCC scheduler

Interleaving 2

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (ep1, ep3)</td>
<td>S (ep2, ep3)</td>
<td>R (ep3)</td>
</tr>
</tbody>
</table>

Issue to the runtime

Enabled Transitions: S(ep2, ep3), R (ep3)

Match Set: <S(ep1, ep3), R( ep3)>
Relevant interleaving exploration by MCC scheduler

Interleaving 2

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (ep1,ep3)</td>
<td>S (ep2,ep3)</td>
<td>R (ep3)</td>
</tr>
</tbody>
</table>

**Error Caught**

Issue to the runtime

R (ep3)
Enforcing a deterministic runtime match

- For non-blocking requests, there is an additional problem
  - Runtime communication race even after a scheduler decides deterministically.
Enforcing a deterministic runtime match

T0

nbS (ep1,ep3) → nbS (ep2,ep3) → nbR (ep3)

T1

RACE

T2

Match-set 1

Match-set 2

nbR (ep3) → nbR (ep3)

Not finished yet!!
Enforcing a deterministic runtime match

Solution 1 to enforce a deterministic match.

- Addition of Probing function to the MCAPI library
  - Probes an endpoint’s receive queue for a message
  - Returns TRUE if the message is found

- Scheduler issues the next match-set only when the probe function returns TRUE.
Enforcing a deterministic runtime match

Probing Solution

\[ \begin{align*}
T0: & \quad nbS(\text{ep1, ep3}) \\
T1: & \quad nbS(\text{ep2, ep3}) \quad \text{matched} \quad nbR(\text{ep3})
\end{align*} \]
Enforcing a deterministic runtime match

Probing Solution

T0: nbS (ep1,ep3) -> matched
T1: nbS (ep2,ep3) -> nbR (ep3)
T2: nbR (ep3) -> Receive Queue

Runtime -> MCC Scheduler

Probes
Enforcing a deterministic runtime match

Probing Solution

T0

nbS (ep1,ep3)

T1

nbS (ep2,ep3)

nbR (ep3)

T2

Receive Queue

nbR (ep3)

Probes

Runtime

MCC Scheduler
Enforcing a deterministic runtime match

Probing Solution

T0  T1  T2

nbS (ep1,ep3)  nbS (ep2,ep3)  nbR (ep3)

Receive Queue

nbR (ep3)
Enforcing a deterministic runtime match

Solution 2 to enforce a deterministic match.

- Scheduler induces a “test” call
- Scheduler spin-loops on the request handle until the successful completion of the “test” call.

We opt Solution 2 in our work as it is non-intrusive.
Enforcing a deterministic runtime match

Dummy “wait” Solution

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbS (ep1,ep3)</td>
<td>nbS (ep2,ep3)</td>
<td>nbR (ep3)</td>
<td></td>
</tr>
</tbody>
</table>

Matched

Enabled Transitions: Send(ep1, ep3), Send(ep2, ep3), Recv (ep3)

Match Set:
- <Send(ep2, ep3), Recv (ep3)>
- <Send(ep1, ep3), Recv (ep3)>
Enforcing a deterministic runtime match

Dummy “wait” Solution

\[
\begin{align*}
\text{T0} & \quad \text{T1} & \quad \text{T2} \\
\text{nbS (ep1,ep3)} & \quad \text{nbS (ep2,ep3)} & \quad \text{nbR (ep3)}
\end{align*}
\]

Runtime

MCC Scheduler

Test ()

nbR (ep3)
Enforcing a deterministic runtime match

Dummy “wait” Solution

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbS (ep1,ep3)</td>
<td>IS (ep2,ep3)</td>
<td>nbR (ep3)</td>
</tr>
</tbody>
</table>

matched

Test()

Runtime

MCC Scheduler
Concluding remarks and future work

- **Code is currently being developed**
  - MCC currently supports blocking and non-blocking connectionless API constructs
  - Checks for safety assertion violations and Deadlocks
  - Porting concurrency benchmarks into MCAPI – Eg. Rodinia Benchmarks, BSS use case
  - MCC will be tested on these benchmarks.

- **Steadily improve MCC**
  - Support for connection-oriented API calls, sanity checks etc.
  - Accommodate “non-determinism” in the shared memory space.
Thank You!

www.cs.utah.edu/formal_verification