Semantics Driven Dynamic Partial-order Reduction of MPI-based Parallel Programs

Robert Palmer
Intel Validation Research Labs, Hillsboro, OR
(work done at the Univ of Utah as PhD student)

Ganesh Gopalakrishnan
Robert M. Kirby

School of Computing
University of Utah

Supported by:
Microsoft HPC Institutes
NSF CNS 0509379
MPI is the de-facto standard for programming cluster machines

Our focus: Eliminate Concurrency Bugs from HPC Programs!

An Inconvenient Truth: Bugs $\rightarrow$ More CO$_2$, Bad Numbers!

(BlueGene/L - Image courtesy of IBM / LLNL)

(Image courtesy of Steve Parker, CSAFE, Utah)
So many ways to eliminate MPI bugs …

- **Inspection**
  - Difficult to carry out on MPI programs (low level notation)

- **Simulation Based**
  - Run given program with manually selected inputs
  - Can give poor coverage in practice

- **Simulation with runtime heuristics to find bugs**
  - Marmot: Timeout based deadlocks, random executions
  - Intel Trace Collector: Similar checks with data checking
  - TotalView: Better trace viewing – still no “model checking” (?)
  - We don’t know if any formal coverage metrics are offered

- **Model Checking Based**
  - Being widely used in practice
  - Can provide superior debugging for reactive bugs
  - Has made considerable strides in abstraction (data, control)
Our Core Technique: Model Checking

Why model checking works in practice:
* It applies *Exhaustive Analysis*, as opposed to *Incomplete Analysis*
* It relies on *Abstraction* (both manual, and automated)

*Exhaustive analysis* of *suitably abstracted* systems helps catch more bugs than *incomplete analysis* of *unabstrackted systems*
[ Rushby, SRI International ]
Model Checking Approaches for MPI

- **MC Based On “Golden” Semantics of MPI**
  - Limited Subsets of MPI / C Translated to TLA+ (FMICS 2007)
  - Limited C Front-End with Slicing using Microsoft Phoenix

- **Hand Modeling / Automated Verif. in Executable Lower Level Formal Notations**
  - Modeling / Verif in Promela (Siegel, Avrunin, et.al. – several papers)
  - Non-Blocking MPI Operations in Promela + C (Siegel)
  - Limited Modeling in LOTOS (Pierre et.al. – in the 90’s)

- **Modeling in MPI / C – Automatic Model Extraction**
  - Limited Conversion to Zing (Palmer et.al. – SoftMC 05)
  - Limited Conversion to MPIC-IR (Palmer et.al. – FMICS 07)

- **Direct Model Checking of Promela / C programs**
  - Pervez et.al. using PMPI Instrumentation – EuroPVM / MPI
  - Demo of One-Sided + a Few MPI Ops (Pervez etal, EuroPVM / MPI 07)
Model Checking Approaches for MPI

1. MC Based On “Golden” Semantics of MPI
   1. Limited Subsets of MPI / C Translated to TLA+ (FMICS 2007)
   2. Limited C Front-End with Slicing using Microsoft Phoenix

2. Hand Modeling / Automated Verif. in Executable Lower Level Formal Notations
   1. Modeling / Verif in Promela (Siegel, Avrunin, et.al. – several papers)
   2. Non-Blocking MPI Operations in Promela + C (Siegel)
   3. Limited Modeling in LOTOS (Pierre et.al. – in the 90’s)

3. Modeling in MPI / C – Automatic Model Extraction
   1. Limited Conversion to Zing (Palmer et.al. – SoftMC 05)
   2. Limited Conversion to MPIC-IR (Palmer et.al. – FMICS 07)

4. Direct Model Checking of Promela / C programs
   1. Pervez et.al. using PMPI Instrumentation – EuroPVM / MPI
   2. Demo of One-Sided + a Few MPI Ops (Pervez etal, EuroPVM / MPI 07)

THIS PAPER : Explain new DPOR Idea Underlying 3.2, 4.2
The Importance of Partial Order Reduction During Model Checking

- With 3 processes, the size of an interleaved state space is $p^s=27$

- Partial-order reduction explores representative sequences from each equivalence class

- Delays the execution of independent transitions
The Importance of Partial Order Reduction for Model Checking

- With 3 processes, the size of an interleaved state space is $p^s=27$

- Partial-order reduction explores representative sequences from each equivalence class

- Delays the execution of independent transitions

- In this example, it is possible to "get away" with 7 states (one interleaving)
POR in the presence of FIFO Channels...

- Can do S, R, S, R
- Or S, S, R, R
- Prefer to do SR, SR (diagonal)

- This is what the “urgent” algorithm tries to do (Siegel)
Static POR Won’t Always Do (Flanagan and Godefroid, POPL 05)

- Action Dependence Determines COMMUTABILITY
  (POR theory is really detailed; it is more than commutability, but let’s pretend it is …)

- Depends on j == k, in this example

- Can be very difficult to determine statically

- Can determine dynamically
**Similar Situation Arises with Wildcards...**

- Dependencies may not be fully known, **JUST by looking at enabled actions**

- So Conservative Assumptions to be made (as in Urgent Algorithm)

- If not, Dependencies may be **Overlooked**

- The same problem exists with other "dynamic situations"
  - e.g. MPI_Cancel

---

**Proc P:** Send(to Q)

**Proc Q:** Recv(from *) Some Stmt

**Proc R:** Send(to Q)
POR in the presence of Wildcards…

Illustration of a Missed Dependency that would have been detected, had Proc R been scheduled first…

Proc P:
Send(to Q)

Proc Q:
Recv(from *)
Some Stmt

Proc R:
Send(to Q)
DPOR Exploits Knowledge of “Future” to Compute Dependencies More Accurately

Ample determined using “local” criteria

Add Red Process to “Backtrack Set”

This builds the “Ample set” incrementally based on observed dependencies

Blue is in “Done” set

Current State

Next move of Red process

Nearest Dependent Transition Looking Back

{ BT }, { Done }
How to define “Dependence” for MPI?

- No a Priori Definition of when Actions Commute
- MPI Offers MANY API Calls
- So need SYSTEMATIC way to define “Dependence”

CONTRIBUTION OF THIS PAPER:

- Define Formal Semantics of MPI
- Define Commutability Based on Formal Semantics
Spec of MPI\_Wait (Slide 1/2) – FMICS07

```c
1  MPI\_Wait(request, status, return, proc) ==
2    LET r == requests[proc][Memory[proc][request]] IN
3    /* Assert(initialized[proc] = "initialized", /* 200.10-200.12
4    "Error: MPI\_Wait called with proc not in initialized state.")
5    /* 41.32-41.39 The request handle is not the null handle.
6    */
7    Memory[proc][request] /= MPI\_REQUEST\_NULL
8    /* The request is active locally.
9    */
10   r.localactive
11   /* The message src is not null
12   */
13   r.message.src /= MPI\_PROC\_NULL
14   /* The message dest is not null
15   */
16   r.message.dest /= MPI\_PROC\_NULL
17   /* 41.32 - Blocks until complete
18   */
19   r.transmitted
20   /* The message was transmitted or
21   */
22   r.canceled
23   /* canceled by the user program or
24   */
25   r.buffered
26   /* buffered by the system
27   */
28   Memory’ =
29    [Memory EXCEPT ![proc] = /* 41.36
30    [* EXCEPT ![Status\_Canceled(status)] =
31    /* r.canceled
32    /* !not r.transmitted, /* 54.46
33    ![Status\_Count(status)] = r.message.numelements,
34    ![Status\_Source(status)] = r.message.src,
35    ![Status\_Tag(status)] = r.message.mstid,
36    ![Status\_Err(status)] = r.error,
37    ![request] = /* 41.32-41.35, 58.34-58.35
38    IF r.persist
39    THEN @
40    ELSE MPI\_REQUEST\_NULL]]
41    /*
42    */
43   r.message.src = MPI\_PROC\_NULL
44   /*
45    */
46   r.message.dest = MPI\_PROC\_NULL
47   /*
48    */
49   Memory’ = [Memory EXCEPT ![proc] = /* 41.36
50    [* EXCEPT ![Status\_Canceled(status)] = r.canceled,
51    ![Status\_Count(status)] = 0,
52    ![Status\_Source(status)] = MPI\_PROC\_NULL,
53    ![Status\_Tag(status)] = MPI\_ANY\_TAG,
54    ![Status\_Err(status)] = 0,
55    ![request] = /* 41.32-41.35, 58.34-58.35
56    IF r.persist
57    THEN @
58    ELSE MPI\_REQUEST\_NULL]]
```
Spec of MPI_Wait (Slide 2/2)

```c
16

\/* requests' = 
  IF r.match /= << >>
  THEN
    [requests EXCEPT ![proc] =  \* 58.34
      [@ EXCEPT
        ![Memory[proc][request]] =
        IF r.persist
          THEN
            IF requests[r.match[1]][r.match[2]].localactive
              THEN [@ EXCEPT ![.localactive = FALSE,
                  ![.globalactive = FALSE]
                ELSE [@ EXCEPT ![.localactive = FALSE]
                  ELSE IF requests[r.match[1]][r.match[2]].localactive
                    THEN [@ EXCEPT ![.localactive = FALSE,
                        ![.globalactive = FALSE,
                        ![.deallocated = TRUE]
                      ELSE [@ EXCEPT ![.localactive = FALSE,
                          ![.deallocated = TRUE]]]
                    ![r.match[1]] =
                      [@ EXCEPT ![r.match[2]] =
                        IF requests[r.match[1]][r.match[2]].localactive
                          THEN requests[r.match[1]][r.match[2]]
                          ELSE [@ EXCEPT ![.globalactive = FALSE]]
                        ELSE [requests EXCEPT ![proc] =  \* 58.34
                          [@ EXCEPT ![Memory[proc][request]] =
                            IF r.persist
                              THEN [@ EXCEPT ![.localactive = FALSE]
                              ELSE [@ EXCEPT ![.localactive = FALSE,
                                  ![.deallocated = TRUE]]]}
                        \*/ Memory[proc][request] = MPI_REQUEST_NULL  \* 41.40-41.41 The
                        \*/ Memory[proc][request] /= MPI_REQUEST_NULL  \* request handle is
                        \*/ lnot r.llocalactive  \* null or the request is not active
                        \*/ Memory' = [Memory EXCEPT ![proc] =  \* 41.36
                          [@ EXCEPT ![Status_Canceled(status)] = FALSE,
                          ![Status_County(status)] = 0,
                          ![Status_Source(status)] = MPI_ANY_SOURCE,
                          ![Status_Tag(status)] = MPI_ANY_TAG,
                          ![Status_Err(status)] = 0]]
                        \*/ UNCHANGED << requests >>
20  \*/ UNCHANGED << group, communicator, buFSIZE, message_buffer,
                initialized, collective >>
```
MPI Formal Specification Organization

- Requests
- Collective
- Context
- Group
- Communicator
- Point to Point Operations
- Collective Operations
- Constants
- MPI 1.1 API
Example: Challenge posed by a 5-line MPI program...

p0: { Irecv(rcvbuf1, from p1);
     Irecv(rcvbuf2, from p1); ... }

p1: { sendbuf1 = 6; sendbuf2 = 7;
     Issend(sendbuf1, to p0);
     Isend (sendbuf2, to p0); ... }

• In-order message delivery (rcvbuf1 == 6)

• Can access the buffers only after a later wait / test

• The second receive may complete before the first

• When Issend (synch.) is posted, all that is guaranteed is that Irecv(rcvbuf1,...) has been posted
One of our Litmus Tests

```c
#include "mpi.h"

int main(int argc, char** argv)
{
    int rank, size, data1, data2, data3, flag;
    MPI_Request req1, req2, req3;
    MPI_Status stat;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(rank == 0){
        data1 = 0;
        data2 = 0;
        MPI_Irecv(&data1, 1, MPI_INT, 1,
                  0, MPI_COMM_WORLD, &req1);
        MPI_Irecv(&data2, 1, MPI_INT, 1,
                  1, MPI_COMM_WORLD, &req2);
        MPI_Irecv(&data3, 1, MPI_INT, 1,
                  2, MPI_COMM_WORLD, &req3);
    } else {
        data1 = 7;
        data2 = 6;
        MPI_Isend(&data1, 1, MPI_INT, 0,
                  1, MPI_COMM_WORLD, &req1);
    }
    if(rank == 1){
        MPI_Wait(&req1, &stat);
        MPI_Isend(&data2, 1, MPI_INT, 0,
                  0, MPI_COMM_WORLD, &req2);
        MPI_Isend(&data3, 1, MPI_INT, 0,
                  2, MPI_COMM_WORLD, &req3);
    }
    else {
        MPI_Wait(&req2, &stat);
    }
    if(rank == 0){
        MPI_Wait(&req1, &stat);
    }
    else {
        MPI_Wait(&req2, &stat);
    }
    MPI_Finalize();
    return 0;
}
```
The Histrionics of FV for HPC (1)
The Histrionics of FV for HPC (2)
Error-trace Visualization in VisualStudio
This paper: Simplified Semantics (e.g. as shown by MPI_Wait)

\[ \Sigma(c, p) \]
\[ p(i) = (l, g) \]
\[ \land \text{proc}(l(vars(pc))) = \text{'wait e'} \]
\[ \land E[e, p_i] = 0 \lor \text{Completed}(g(E[e, p_i])) \]

\[ \Sigma(c, p)[ i \mapsto ( l[vars(pc) \mapsto \text{next}(l(vars(pc))), \]
\[ vars(e) \mapsto 0])] \]

\[ \Sigma(c, p) \land \]
\[ p(i) = (l_i, g_i) \land \text{proc}(l_i(vars(pc))) = \text{'wait e'} \]
\[ \land E[e, p_i] \in \text{Dom}(g_i) \land \]
\[ \exists j : p(j) = (l_j, g_j) \land \]
\[ \exists k : \text{Match}(g_j(k), g_i(E[e, p_i])) \land \]
\[ \forall m < k : \neg \text{Match}(g_j(m), g_i(E[e, p_i])) \]

\[ \Sigma(c, p)[ i \mapsto ( l_i[vars(pc) \mapsto \text{next}(l_i(vars(pc))), \]
\[ vars(e) \mapsto 0], \]
\[ g_i[E[e, p_i] \mapsto g_i(E[e, p_i])[false/true]] \]
\[ j \mapsto ( l_j, g_j[k \mapsto g_j(k)[false/true]]))] \]
Independence Theorems based on Formal Semantics of MPI Subset

1. Local actions (Assignment, Goto, Alloc, Assert) are independent of all transitions of other processes.

2. Barrier actions (Barrier_init, Barrier_wait) are independent of all transitions of other processes.

3. Issend and Irecv are independent of all transitions of other processes except Wait and Test.

4. Wait and Test are independent of all transitions of other processes except Issend and Irecv.
Executable Formal Specification and MPIC Model Checker Integration into VS

- Visual Studio 2005
- Verification Environment
- Phoenix Compiler
- MPIC IR
- TLA+ MPI Library Model
- TLA+ Prog. Model
- TLC Model Checker
- MPIC Program Model
- MPIC Model Checker

FMICS 07

PADTAD 07
A Simple Example:
e.g. mismatched send/recv causing deadlock

/* Add-up integrals calculated by each process */
if (my_rank == 0) {
    total = integral;
    for (source = 0; source < p; source++) {
        MPI_Recv(&integral, 1, MPI_FLOAT, source,
                  tag, MPI_COMM_WORLD, &status);
        total = total + integral;
    }
} else {
    MPI_Send(&integral, 1, MPI_FLOAT, dest,
             tag, MPI_COMM_WORLD);
}
Partial Demo of DPOR Tool for MPIC
So, the whole story (i.e. Conclusions)…

- Preliminary Formal Semantics of MPI in Place (50 point-to-point functions)
- Can Model-Check this Golden Semantics
- About 5 of these 30 have a more rigorous characterization thru Independence Theorems
- For MPI Programs using These MPI functions, we have a DPOR based model checker MPIC
- Integrated in the VS Framework with MPI-TLC also

- Theory Expected to Carry Over into In-Situ Dynamic Partial Order Reduction (model-check without model building – EuroPVM / MPI 2007)
Questions ?

The verification environment is downloadable from

http://www.cs.utah.edu/formal_verification/mpic

It is at an early stage of development
Answers!

1. We are extending it to Collective Operations
   - lesson learned from de Supinski

2. We may perform Formal Testing of MPI Library Implementations based on the Formal Semantics

3. We plan to analyze mixed MPI / Threads

4. That is a very good question – let’s talk!