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

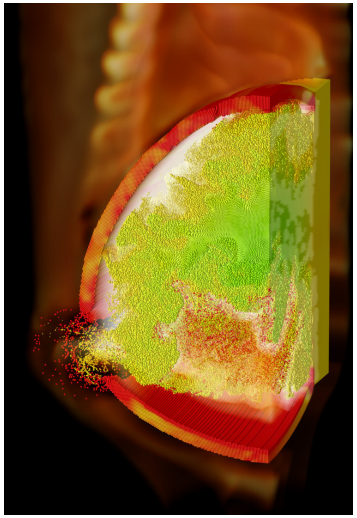
NSF CNS 0509379

THE UNIVERSITY OF UTAH

MPI is the de-facto standard for programming cluster machines

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

→



(Image courtesy of Steve Parker, CSAFE, Utah)

Our focus: ***Eliminate Concurrency Bugs from HPC Programs !***

An Inconvenient Truth: ***Bugs → More CO_2, Bad Numbers !***

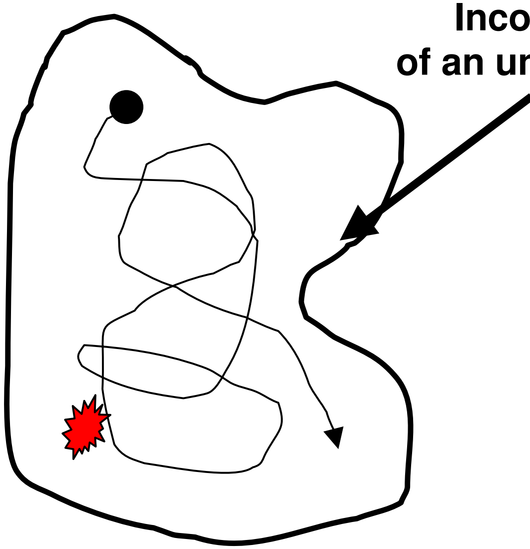
So many ways to eliminate MPI bugs …

- **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

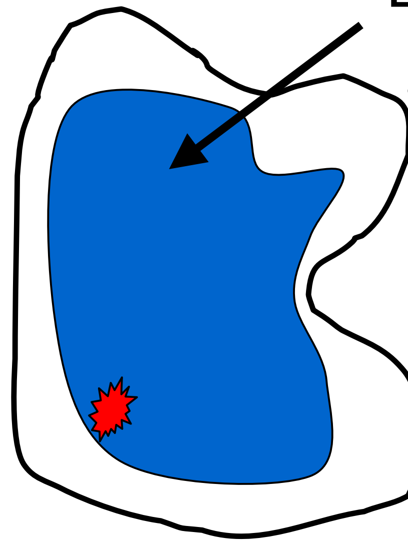
"Ad-hoc Testing"

Incomplete analysis
of an unabstracted system



"Model Checking"

Exhaustive analysis
of a suitably
abstracted system



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 unabstracted 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

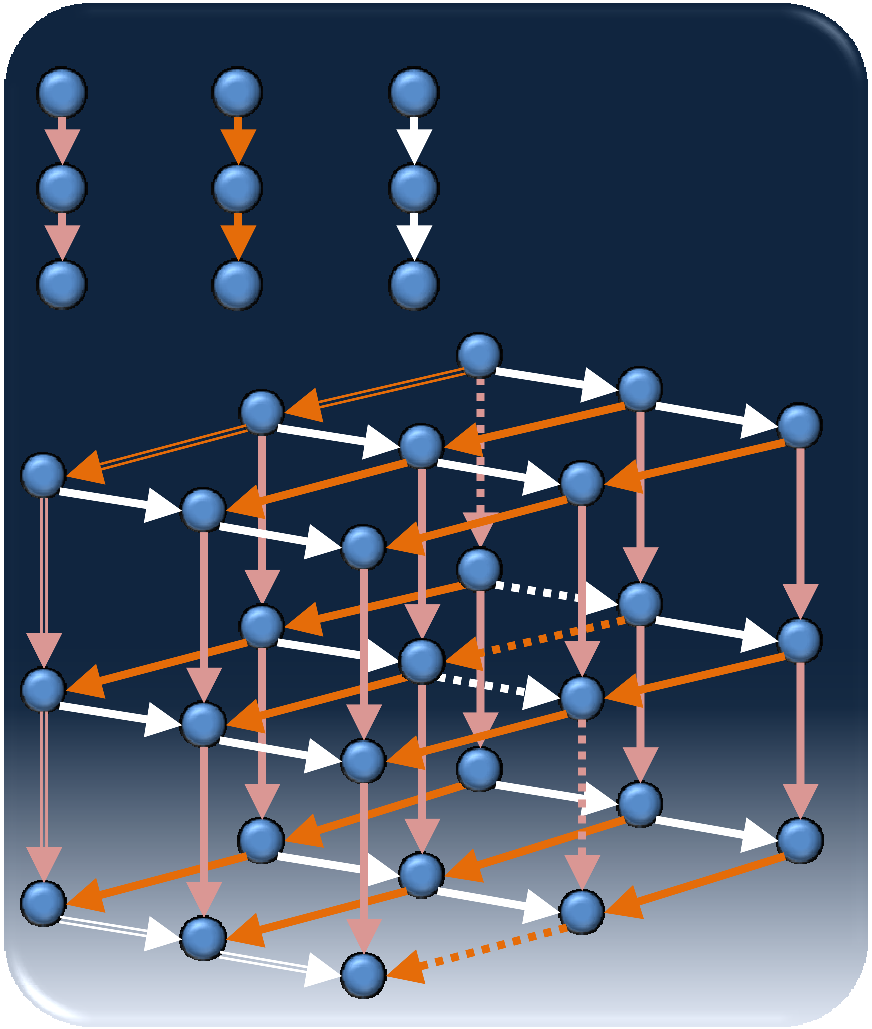
1. **MC Based On "Golden" Semantics of MPI**
   1. ***Limited*** **Subsets of MPI / C Translated to TLA+ (FMICS 2007)**
   2. ***Limited*** **C Front-End with Slicing using Microsoft Phoenix**
2. **Hand Modeling / Automated Verif. in Executable Lower Level Formal Notations**
   1. **Modeling / Verif 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 UNIVERSITY OF UTAH

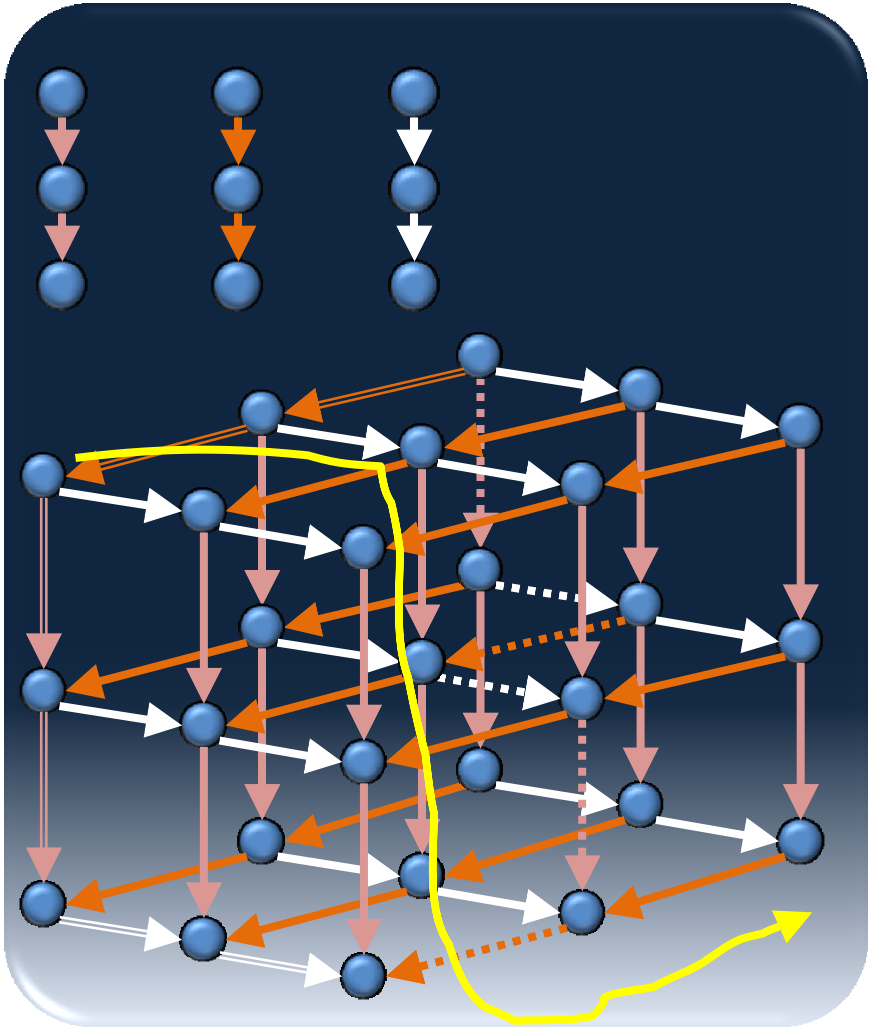
The Importance of Partial Order Reduction During Model Checking

- 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

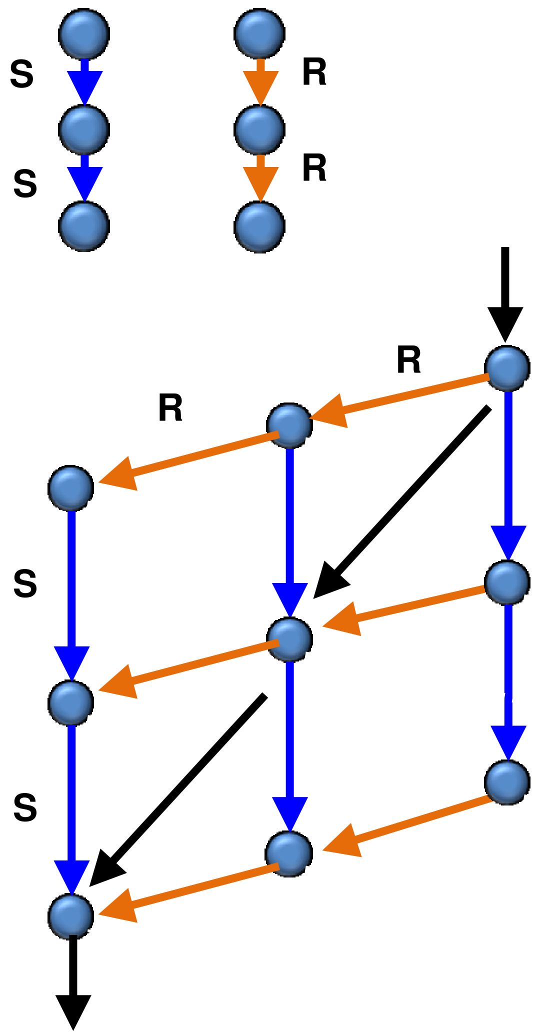


The Importance of Partial Order Reduction for Model Checking

- 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)

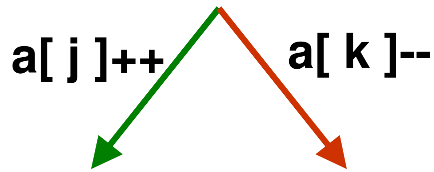


POR in the presence of FIFO Channels…



- 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**

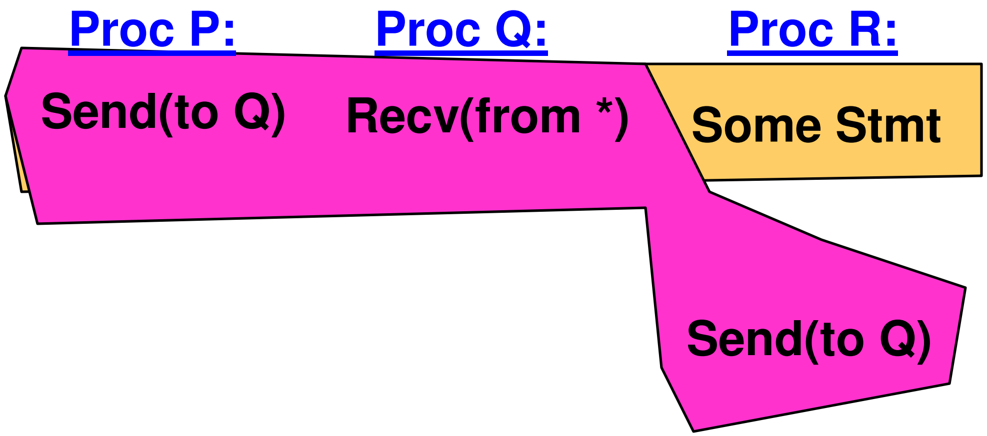
THE UNIVERSITY OF UTAH

Similar Situation Arises with Wildcards…

Proc P:	Proc Q:	Proc R:
Send(to Q)	Recv(from *)	Some Stmt
		Send(to Q)

- 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

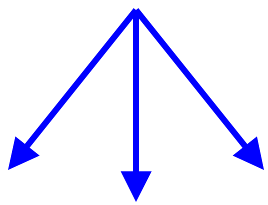
POR in the presence of Wildcards…



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

DPOR Exploits Knowledge of "Future" to Compute Dependencies More Accurately

Ample determined using "local" criteria



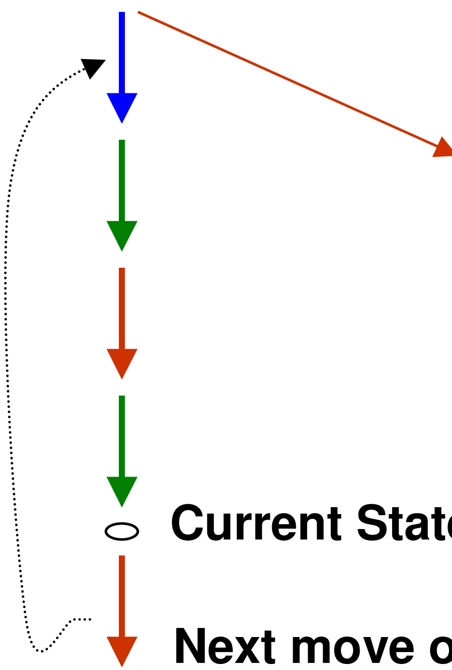
{ BT }, { Done }

Add Red Process to "Backtrack Set"

This builds the "Ample set" incrementally based on observed dependencies

Blue is in "Done" set

Nearest Dependent Transition Looking Back



Current State

Next move of Red process

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

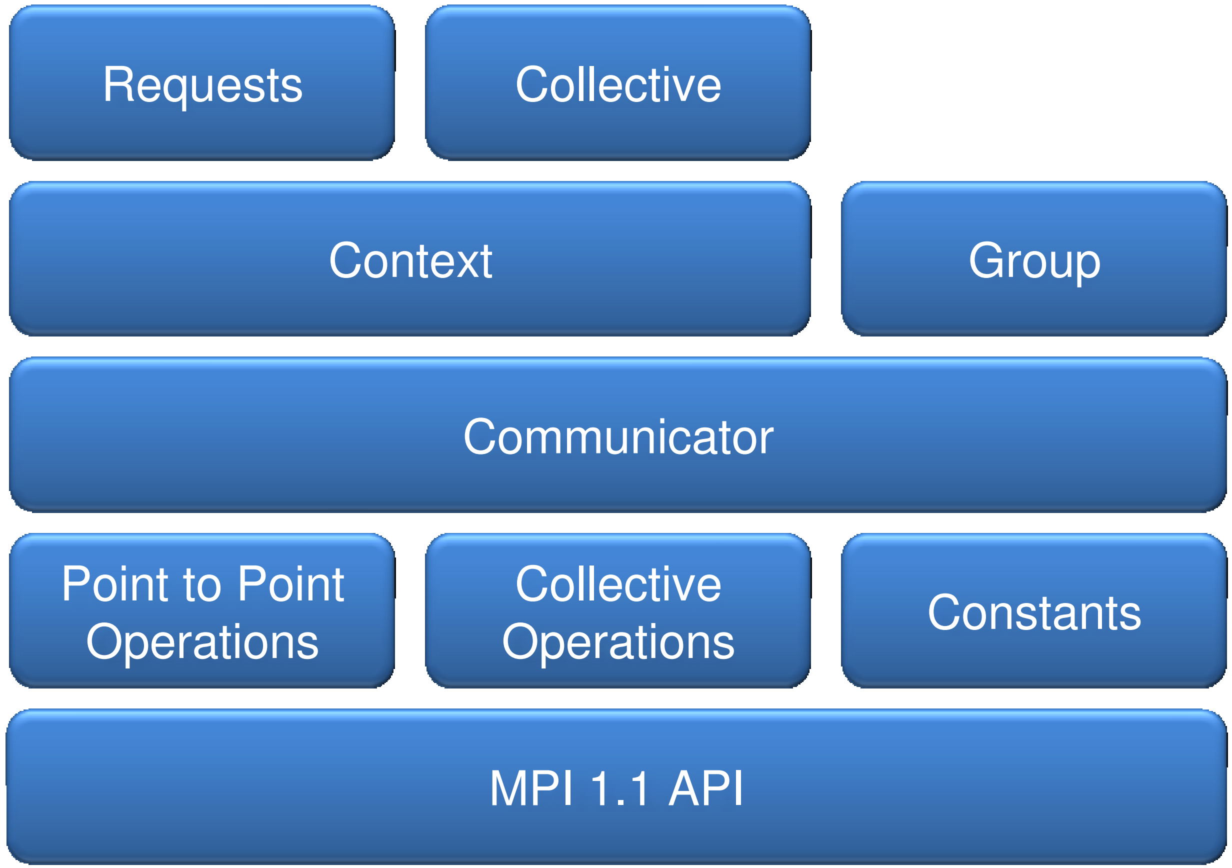
Spec of MPI_Wait (Slide 1/2) – FMICS07

```
MPI_Wait(request, status, return, proc) ==
  LET r == requests[proc][Memory[proc][request]] IN
  /\ Assert(initialized[proc] = "initialized", \* 200.10-200.12
            "Error: MPI_Wait called with proc not in initialized state.")
        \* 41.32-41.39 The request handle is not the null handle.
  /\ \/ /\ Memory[proc][request] /= MPI_REQUEST_NULL
        /\ r.localactive                          \* The request is active locally.
        /\ \/ /\ r.message.src /= MPI_PROC_NULL   \* The message src is not null
              /\ r.message.dest /= MPI_PROC_NULL  \* The message dest is not null
                                  \* 41.32 - Blocks until complete
              /\ \/ r.transmitted \* The message was transmitted or
                 \/ r.canceled   \* canceled by the user program or
                 \/ r.buffered    \* buffered by the system
              /\ Memory' =
                   [Memory EXCEPT ![proc] =   \* 41.36
                      [@ EXCEPT ![Status_Canceled(status)] =
                                     /\ r.canceled
                                     /\ \lnot r.transmitted, \* 54.46
                            ![Status_Count(status)] = r.message.numelements,
                            ![Status_Source(status)] = r.message.src,
                            ![Status_Tag(status)] = r.message.msgtag,
                            ![Status_Err(status)] = r.error,
                            ![request] = \* 41.32-41.35, 58.34-58.35
                               IF r.persist
                               THEN @
                               ELSE MPI_REQUEST_NULL]]
           \/ /\ \/ r.message.src = MPI_PROC_NULL
                 \/ r.message.dest = MPI_PROC_NULL
              /\ Memory' = [Memory EXCEPT ![proc] =   \* 41.36
                  [@ EXCEPT ![Status_Canceled(status)] = r.canceled,
                            ![Status_Count(status)] = 0,
                            ![Status_Source(status)] = MPI_PROC_NULL,
                            ![Status_Tag(status)] = MPI_ANY_TAG,
                            ![Status_Err(status)] = 0,
                            ![request] = \* 41.32-41.35, 58.34-58.35
                               IF r.persist
                               THEN @
                               ELSE MPI_REQUEST_NULL]]
```

Spec of MPI_Wait (Slide 2/2)

```
          /\ 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.localactive          \* null or the request is not active
          /\ Memory' = [Memory EXCEPT ![proc] = \* 41.36
                         [@ EXCEPT ![Status_Canceled(status)] = FALSE,
                                   ![Status_Count(status)] = 0,
                                   ![Status_Source(status)] = MPI_ANY_SOURCE,
                                   ![Status_Tag(status)] = MPI_ANY_TAG,
                                   ![Status_Err(status)] = 0]]
          /\ UNCHANGED << requests >>
    /\ UNCHANGED << group, communicator, bufsize, message_buffer,
                    initialized, collective >>
```

MPI Formal Specification Organization



Example: Challenge posed by a 5-line MPI program…

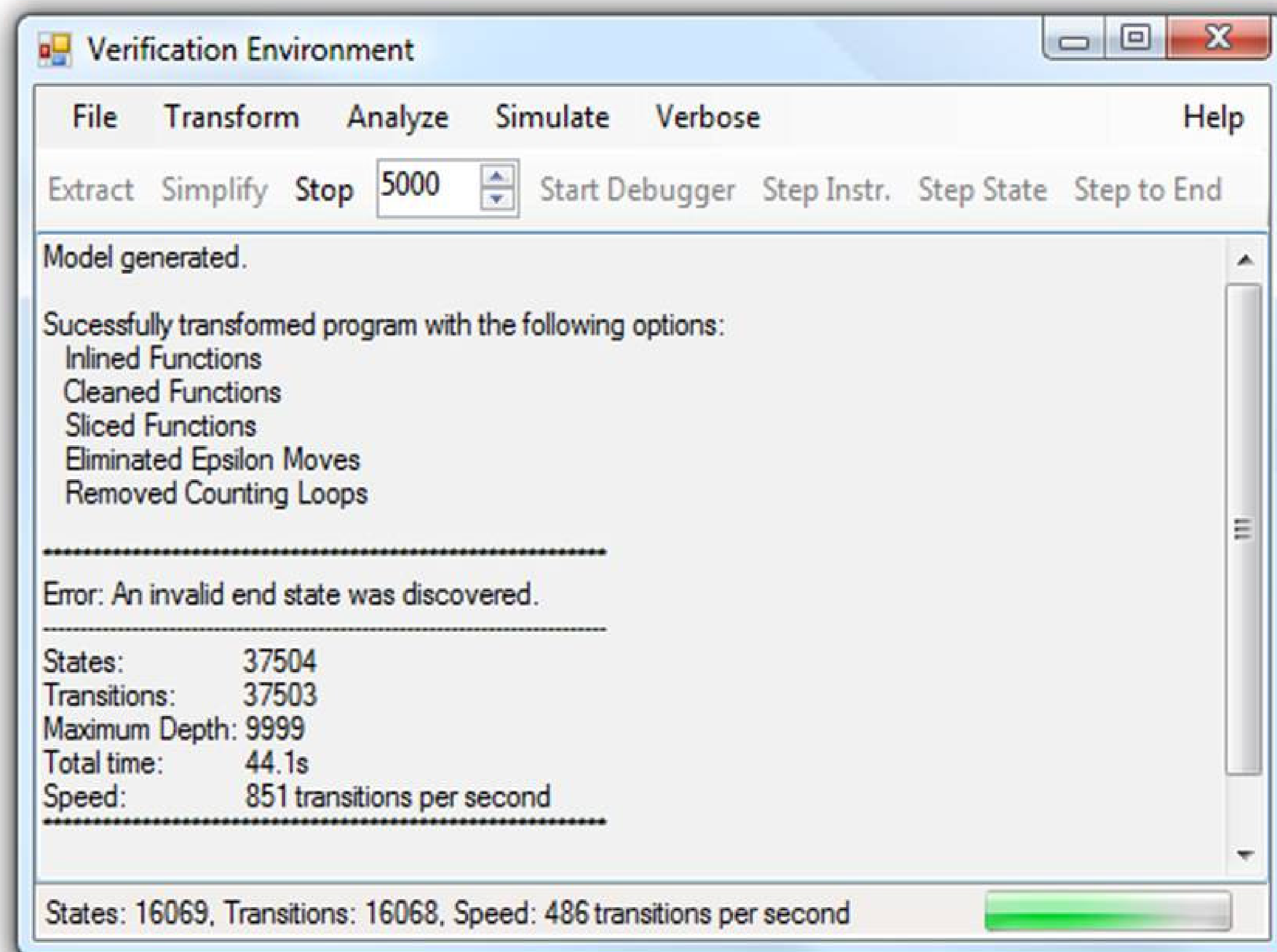
```
p0: { Irecv(rcvbuf1,  from p1);
      Irecv(rcvbuf2,  from p1); … }

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

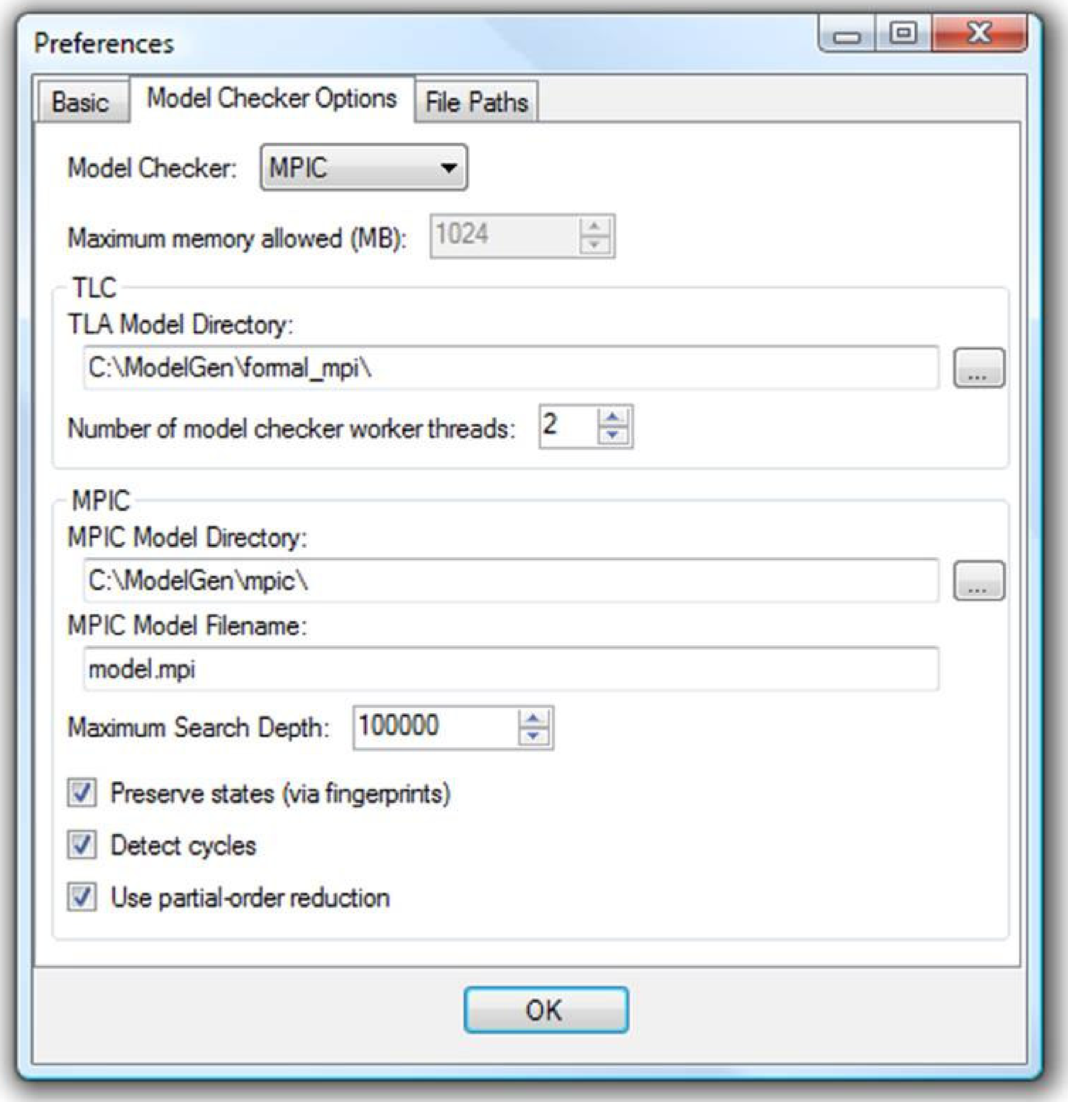
```
#include "mpi.h"

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_Issend(&data1, 1, MPI_INT, 0,
               1, MPI_COMM_WORLD, &req1);
  }
  if(rank == 1){
    MPI_Wait(&req1, &stat);
    MPI_Irsend(&data2, 1, MPI_INT, 0,
               0, MPI_COMM_WORLD, &req2);
    MPI_Irsend(&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)



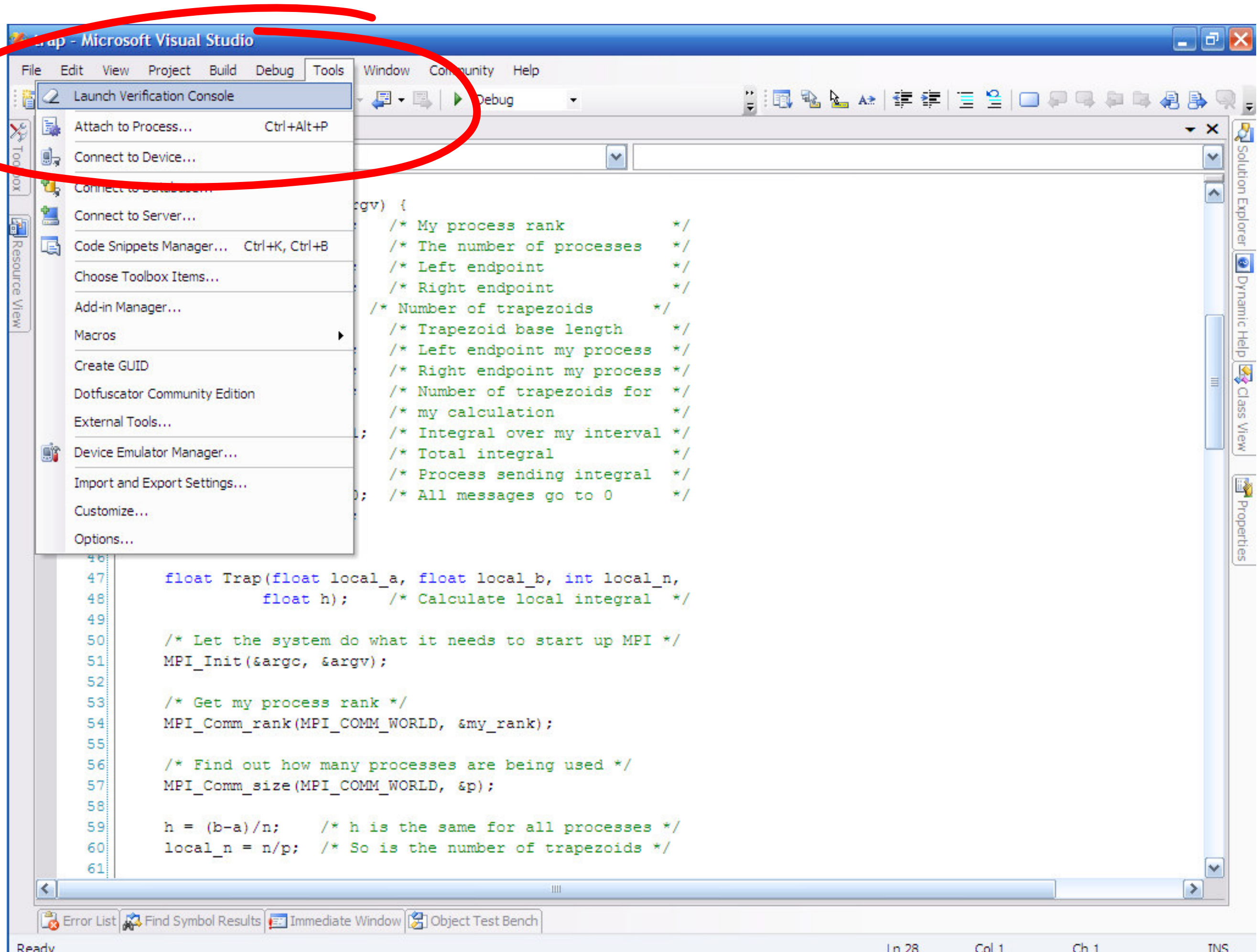
THE UNIVERSITY OF UTAH

The Histrionics of FV for HPC (2)



THE UNIVERSITY OF UTAH

Error-trace Visualization in VisualStudio



THE UNIVERSITY OF UTAH

This paper: Simplified Semantics (e.g. as shown by MPI_Wait)

$$\frac{\begin{array}{l}\Sigma(c,p) \\ \qquad p(i) = (l,g) \\ \qquad \wedge proc(l(vars(\mathtt{pc}))) = \mathtt{'wait\ e'} \\ \qquad \wedge E[\mathtt{e},p_i] = 0 \vee Completed(g(E[\mathtt{e},p_i]))\end{array}}{\begin{array}{l}\Sigma(c,p[\quad i \mapsto (\quad l[vars(\mathtt{pc}) \mapsto next(l(vars(\mathtt{pc}))), \\ \qquad\qquad\qquad vars(\mathtt{e}) \mapsto 0])])\end{array}}$$

$$\frac{\begin{array}{l}\Sigma(c,p) \wedge \\ \qquad p(i) = (l_i,g_i) \ \wedge \ proc(l_i(vars(\mathtt{pc}))) = \mathtt{'wait\ e'} \\ \qquad \wedge\ E[\mathtt{e},p_i] \in Dom(g_i) \wedge \\ \qquad \exists j : \quad p(j) = (l_j,g_j) \wedge \\ \qquad\qquad \exists k : \quad Match(g_j(k), g_i(E[\mathtt{e},p_i]))\wedge \\ \qquad\qquad\qquad \forall m < k : \neg Match(g_j(m), g_i(E[\mathtt{e},p_i]))\end{array}}{\begin{array}{l}\Sigma(c,p[\quad i \mapsto (\quad l_i[vars(\mathtt{pc}) \mapsto next(l_i(vars(\mathtt{pc}))), \\ \qquad\qquad\qquad vars(\mathtt{e}) \mapsto 0] \\ \qquad\qquad\qquad g_i[E[\mathtt{e},p_i] \mapsto g_i(E[\mathtt{e},p_i])[false/true]] \\ \qquad\quad j \mapsto (\quad l_j, g_j[k \mapsto g_j(k)[false/true]]))])\end{array}}$$

Independence Theorems based on Formal Semantics of MPI Subset

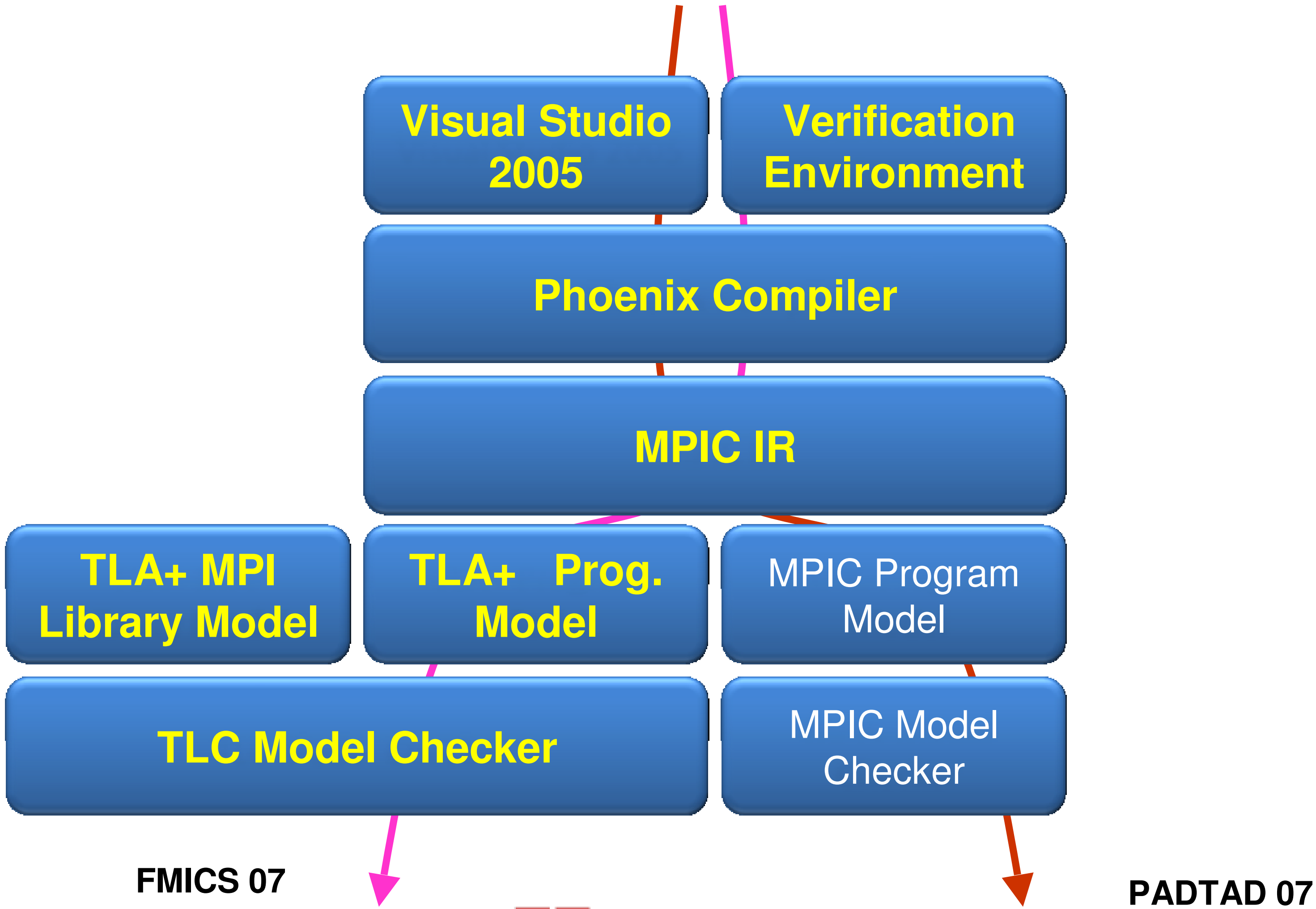
1. **Local actions (Assignment, Goto, Alloc, Assert) are independent of all transitions of other processes.**

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



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);
    }
```

p0:fr 0 p0:fr 1 p0:fr 2

p1:to 0 p2:to 0 p3:to 0

Partial Demo of DPOR Tool for MPIC

THE UNIVERSITY OF UTAH

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!