Verifying Parallel Programs with MPI-Spin
Part 3: Using MPI-Spin

Stephen F. Siegel

Department of Computer and Information Sciences
University of Delaware

EuroPVM/MPI 2007
Paris, France
30 September 2007
Overview

1. 4 steps
2. Using ms
3. Using mscc
4. Executing pan
5. Interpreting the output of pan
6. Reduction theorems
Using MPI-Spin: 4 steps

1. generate the analyzer
   - `ms foo.prom`
   - generates C source files `pan.*`

2. compile the analyzer
   - `mscc`
   - compiles and links the source files
   - results in executable `pan`

3. execute the analyzer
   - `./pan`
   - sends output to stdout

4. if counterexample found, play back the trail
   - `./pan -r foo.prom`
   - sends output to stdout
Using MPI-Spin: 4 steps

1. generate the analyzer
   - `ms foo.prom`
   - generates C source files `pan.*`

2. compile the analyzer
   - `mscc`
   - compiles and links the source files
   - results in executable `pan`
Using MPI-Spin: 4 steps

1. generate the analyzer
   - `ms foo.prom`
   - generates C source files `pan.*`

2. compile the analyzer
   - `mscc`
   - compiles and links the source files
   - results in executable `pan`

3. execute the analyzer
   - `./pan`
   - sends output to `stdout`

4. if counterexample found, play back the trail
   - `./pan -r foo.prom`
   - sends output to `stdout`
Using MPI-Spin: 4 steps

1. generate the analyzer
   - `ms foo.prom`
   - generates C source files `pan.*`

2. compile the analyzer
   - `mscc`
   - compiles and links the source files
   - results in executable `pan`

3. execute the analyzer
   - `./pan`
   - sends output to `stdout`

4. if counterexample found, play back the trail
   - `./pan -r foo.prom`
   - sends output to `stdout`
ms syntax

- **ms** (with no arguments)
  - see all command-line options
- **ms [options] foo.prom**
  - generate analyzer source code from foo.prom
- **-v**
  - verbose mode
- **-np=INT**
  - number of MPI processes (required)
- **-DMYMACRO**
  - equivalent to adding `#define MYMACRO` to beginning of model file
- **-DMYTHING=VAL**
  - equivalent to adding `#define MYMACRO VAL` to beginning of model file
**ms options: nonblocking mode**

- nonblocking mode is the default mode
  - can be used for nonblocking and blocking communication
  - explicit list of request objects are modeled as part of the state
ms options: nonblocking mode

- nonblocking mode is the default mode
  - can be used for nonblocking and blocking communication
  - explicit list of request objects are modeled as part of the state
- `-buf=<INT>`
  - upper bound on the number of messages that can be buffered at any one time
  - used only in nonblocking mode (required)
  - when upper bound is reached, sends block until number of buffered messages falls below bound
  - hence some executions may not be modeled
    - but small scope hypothesis...
ms options: nonblocking mode

- nonblocking mode is the default mode
  - can be used for nonblocking and blocking communication
  - explicit list of request objects are modeled as part of the state
- `-buf=<INT>`
  - upper bound on the number of messages that can be buffered at any one time
  - used only in nonblocking mode (required)
  - when upper bound is reached, sends block until number of buffered messages falls below bound
  - hence some executions may not be modeled
    - but small scope hypothesis...
- `-req=<INT>`
  - upper bound on the number of request objects that can be allocated at any one time
  - used only in nonblocking mode (required)
  - when upper bound is reached, attempt to post a send or receive request results in error
ms options: blocking mode

• -block
  • use channel-based mode
  • optimization for models with only blocking communication

Interpreting the output Reduction theorems
ms options: blocking mode

- **-block**
  - use channel-based mode
  - optimization for models with only blocking communication

- **-chansize=<INT>**
  - channel size
  - used only in blocking mode (required)
  - for any processes \( p \) and \( q \), this is an upper bound on number of messages send from \( p \) to \( q \) but not yet received
  - when upper bound is reached, sends from \( p \) to \( q \) block until number of such messages falls below bound
  - hence some executions may not be modeled
    - but small scope hypothesis...
ms options: optimizations

- these options reduce the number of states explored for certain models
- `--noanysource`
  - optimization for models that do not use `MPI_ANY_SOURCE`
ms options: optimizations

• these options reduce the number of states explored for certain models
  • -noanysource
    • optimization for models that do not use MPI_ANY_SOURCE
  • -notest
    • optimization for models that do not use MPI_Test*, MPI_Waitany, MPI_Waitsome
ms options: optimizations

- these options reduce the number of states explored for certain models
  - `--noanysource`
    - optimization for models that do not use `MPI_ANY_SOURCE`
  - `--notest`
    - optimization for models that do not use `MPI_Test*`, `MPI_Waitany`, `MPI_Waitsome`
  - `--noprobe`
    - optimization for models that do not use `MPI_Probe`, `MPI_Iprobe`
ms options: optimizations

- these options reduce the number of states explored for certain models

- `-noanysource`
  - optimization for models that do not use `MPI_ANY_SOURCE`

- `-notest`
  - optimization for models that do not use `MPI_Test*`, `MPI_Waitany`, `MPI_Waitsome`

- `-noprobe`
  - optimization for models that do not use `MPI_Probe`, `MPI_Iprobe`

- `-nocancel`
  - optimization for models that do not use `MPI_Cancel`
ms options: deadlock

- `-dl`
  - perform a full deadlock search
  - for each standard-mode send, explore both possibilities:
    1. message is buffered if sufficient buffering space is available
    2. send is forced to synchronize

- without this option, only possibility 1 is explored
**ms options: deadlock**

- **-dl**
  - perform a full deadlock search
  - for each standard-mode send, explore both possibilities:
    1. message is buffered if sufficient buffering space is available
    2. send is forced to synchronize

- without this option, only possibility 1 is explored
- the option has no effect if used with
  - **-req=0**, or
  - **-block -chansize=0**
Using msc
c

- msc [options]
  - compile and link pan.*
Using msc
c
- msc [options]
  - compile and link pan.*
- -DSAFETY
  - Spin optimization allowed if not checking certain temporal properties
Using mscc

- **mscc [options]**
  - compile and link **pan.**
- **-DSAFETY**
  - **Spin** optimization allowed if not checking certain **temporal** properties
- **-DCOLLAPSE**
  - effective **Spin** **compression** algorithm for stored states
  - time-memory tradeoff
    - runs slightly slower but uses less memory
Using \texttt{mscc}

- \texttt{mscc} [options]
  - compile and link \texttt{pan.*}
- \texttt{-DSAFETY}
  - \texttt{Spin} optimization allowed if not checking certain temporal properties
- \texttt{-DCOLLAPSE}
  - effective \texttt{Spin} compression algorithm for stored states
    - time-memory tradeoff
      - runs slightly slower but uses less memory
- \texttt{-DMYMACRO}
  - equivalent to adding \texttt{#define MYMACRO} to beginning of analyzer C source file
- \texttt{-DMYTHING=VAL}
  - equivalent to adding \texttt{#define MYMACRO VAL} to beginning of analyzer C source file
4 steps

Using ms

Using msc

Executing pan

Interpreting the output

Reduction theorems

Common options for **pan**

- `-n`
  - suppress printing of non-reachable states
Common options for `pan`

- `-n`
  - suppress printing of non-reachable states
- `-m<INT>`
  - bound search depth to specified number
  - bounds length (number of steps) of execution
Common options for pan

- **-n**
  - suppress printing of non-reachable states
- **-m<INT>**
  - bound search depth to specified number
  - bounds length (number of steps) of execution
- **-i**
  - find a counterexample with the minimal number of steps
  - require `mscc` option `-DREACH`
Common options for pan

- **-n**
  - suppress printing of non-reachable states
- **-m<INT>**
  - bound search depth to specified number
  - bounds length (number of steps) of execution
- **-i**
  - find a counterexample with the minimal number of steps
  - require mscc option -DREACH
- **-r**
  - play back trail after error is found
  - often used in conjunction with **-n** to reduce detail
  - **printfs** are executed
  - values of variables at each step
Interpreting the output of \texttt{pan}

- the most important thing
  - \texttt{errors: 0}
    - no errors found: your property holds
  - \texttt{errors: 1}
    - an error was found: your property does not hold
Interpreting the output of \texttt{pan}

- the most important thing
  - \texttt{errors}: 0
    - no errors found: your property holds
  - \texttt{errors}: 1
    - an error was found: your property does not hold

- the most important stats
  - number of states explored
    - 74 states, stored
  - amount of memory used: sum of
    1. 2.622 memory usage (Mbyte)
    2. MPI-Spin memory usage (bytes): 125633
Additional information using `ms -v`

- the maximum number of messages buffered at one time
  - `Max num buffered messages achieved` ...... 0
- the maximum number of simultaneously allocated request objects
  - `Max num outstanding requests achieved` .... 0
Reduction theorems

• it is not always necessary to explore all possible interleavings
Reduction theorems

- it is not always necessary to explore all possible interleavings

**Modeling Wildcard-Free MPI Programs for Verification**
- S. Siegel, G. Avrunin (PPoPP 2005)
- if you restrict to a subset of blocking operations...
  - `MPI_Send`, `MPI_Recv`, `MPI_Sendrecv`, `MPI_Sendrecv_replace`, `MPI_ANY_TAG`, `MPI_Bcast`, `MPI_Barrier`, ...
  - **no `MPI_ANY_SOURCE`**
Reduction theorems

- it is not always necessary to explore all possible interleavings
- **Modeling Wildcard-Free MPI Programs for Verification**
  - S. Siegel, G. Avrunin (PPoPP 2005)
  - if you restrict to a subset of blocking operations...
    - **MPI_Send, MPI_Recv, MPI_Sendrecv,**
      **MPI_Sendrecv_replace, MPI_ANY_TAG, MPI_Bcast,**
      **MPI_BARRIER, ...**
    - no **MPI_ANY_SOURCE**
  - ... then you can conclude
    - program is deadlock-free if and only if it synchronously deadlock-free
    - i.e., you only need to examine traces in which all communication takes place synchronously
    - enormous savings in terms of number of states, memory, time
Reduction theorems

- it is not always necessary to explore all possible interleavings
- **Modeling Wildcard-Free MPI Programs for Verification**
  - S. Siegel, G. Avrunin (PPoPP 2005)
  - if you restrict to a subset of blocking operations...
    - MPI_Send, MPI_Recv, MPI_Sendrecv,
      MPI_Sendrecv_replace, MPI_ANY_TAG, MPI_Bcast,
      MPI_BARRIER, ...
    - no MPI_ANY_SOURCE
  - ... then you can conclude
    - program is deadlock-free if and only if is it synchronously deadlock-free
    - i.e., you only need to examine traces in which all communication takes place synchronously
    - enormous savings in terms of number of states, memory, time
  - even better:
    - any property of the final state of program is independent of interleavings
    - i.e., you can choose any interleaving you want
    - e.g., place an atomic block around the body of each process
Reduction with wildcard receives

- **Efficient verification of halting properties for MPI programs with wildcard receives**
  - S. Siegel (VMCAI'05)
  - a method to deal with `MPI_ANY_SOURCE`
  - powerful, but impossible to implement in SPIN
Reduction with wildcard receives

- **Efficient verification of halting properties for MPI programs with wildcard receives**
  - S. Siegel (VMCAI'05)
  - a method to deal with MPI\_ANY\_SOURCE
  - powerful, but impossible to implement in SPIN

- **Combining Symbolic Execution with Model Checking to Verify Parallel Numerical Programs**
  - S. Siegel, A. Mironova, G. Avrunin, L. Clarke (TOSEM, to appear)
  - if model uses MPI\_ANY\_SOURCE
  - use atomic blocks where you like
  - as long as every MPI\_ANY\_SOURCE receive starts a new atomic block
More reduction theorems

- Verification of Halting Properties for MPI Programs Using Nonblocking Operations
  - S. Siegel, G. Avrunin (EuroPVM/MPI 2007)
  - extends results above to certain nonblocking MPI operations
  - good: MPI_Isend, MPI_Irecv, MPI_Wait
  - bad: MPI_Test, MPI_Waitany, MPI_Testany, MPI_Waitsome, MPI_Testsome, ...
More reduction theorems

• **Verification of Halting Properties for MPI Programs Using Nonblocking Operations**
  - S. Siegel, G. Avrunin (EuroPVM/MPI 2007)
  - extends results above to certain nonblocking MPI operations
  - good: `MPI_Isend`, `MPI_Irecv`, `MPI_Wait`
  - bad: `MPI_Test`, `MPI_Waitany`, `MPI_Testany`, `MPI_Waitsome`, `MPI_Testsome`, ...

• **Semantics Driven Partial-order Reduction of MPI-based Parallel Programs**
  - R. Palmer, G. Gopalakrishnan, R.M. Kirby (PADTAD 2007)
  - dynamic partial order reduction
Exercises

1. Consider the C/MPI program `exchange.c` listed on the following slide. Create an MPI-SPIN model of this program and use it to determine whether the program can deadlock.

2. Modify the model to use nonblocking communication to accomplish the exchange. Use MPI-SPIN to determine whether this model can deadlock.

3. Can you find a way to correct `diffusion_par2.c` without introducing a barrier? Use MPI-SPIN to verify your solution.
#define UP 0
#define DOWN 1

int main(int argc, char *argv[]) {
    int np, rank, i, sbuf[1], rbuf[1];
    MPI_Status s;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    for (i = 0; i < 3; i++) {
        sbuf[0] = UP;
        MPI_Send(sbuf, 1, MPI_INT, (rank+1)%np, 9, MPI_COMM_WORLD);
        MPI_Recv(rbuf, 1, MPI_INT, (rank+np-1)%np, 9, MPI_COMM_WORLD, &s);
        fprintf(stdout, "Proc %d received %d\n", rank, rbuf[0]);
        sbuf[0] = DOWN;
        MPI_Send(sbuf, 1, MPI_INT, (rank+np-1)%np, 9, MPI_COMM_WORLD);
        MPI_Recv(rbuf, 1, MPI_INT, (rank+1)%np, 9, MPI_COMM_WORLD, &s);
        fprintf(stdout, "Proc %d received %d\n", rank, rbuf[0]);
    }
    MPI_Finalize();
}

S.F. Siegel – Verifying Parallel Programs with MPI-Spin, 3: Using MPI-Spin