Formal methods for HPC

Hac Specis

SIMGRID

November 9, 2020

Formal methods in Hac Specis

Problem statement

- SimGrid/SMPI efficiently emulates one trajectory of an MPI application
- Mc SimGrid was an experimental/inefficient model-checker for MPI code

#1 Efficient formal verification: correctness

- Exhaustive exploration of relevant system behaviors (time abstracted away)
- The Anh Pham (Phd, Rennes): Modeling; Efficient exploration

#2 Statistical model-checking: probabilistic evaluation of performance

- Controlled Monte-carlo simulations $\rightsquigarrow$ events probability and expected values
- Yann Duplouy (postdoc, Nancy): Prototype of SMC over SimGrid simulations

#3 Bridging the gap: from real execution to simulation to verification

- Checkpoint an application on OpenMPI, restart it within SimGrid
- Millian Poquet (postdoc, Rennes): OpenMPI driver using SimGrid, use DMTCP

Efficient exhaustive exploration of MPI apps

Problem: State Space Explosion

- Asynchronously send one task to each worker; wait for the answers
- Millions of execution orders (N=3), one single outcome (if independent workers)

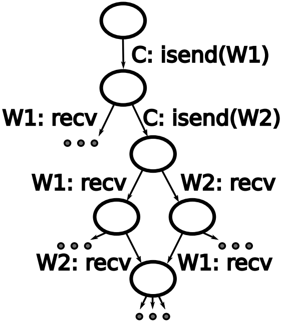
```
actor coordinator:
    for i in [1; N] do
        req[i] = Isend to worker[i]
    for i in [1; N] do
        wait request req[i]
    for i in [1; N] do
        receive from worker[i]
actor worker (N instances):
    receive from coordinator
    compute()
    send to coordinator
```

Efficient exhaustive exploration of MPI apps

Problem: State Space Explosion

- Asynchronously send one task to each worker; wait for the answers
- Millions of execution orders (N=3), one single outcome (if independent workers)

```
actor coordinator:
    for i in [1; N] do
        req[i] = Isend to worker[i]
    for i in [1; N] do
        wait request req[i]
    for i in [1; N] do
        receive from worker[i]
actor worker (N instances):
    receive from coordinator
    compute()
    send to coordinator
```



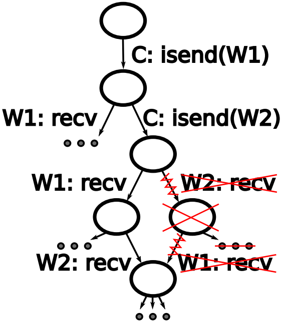
Efficient exhaustive exploration of MPI apps

Problem: State Space Explosion

- Asynchronously send one task to each worker; wait for the answers
- Millions of execution orders (N=3), one single outcome (if independent workers)

Dynamic Partial Ordering Reduction (DPOR)

- Explore once permutations of independent actions
- Dynamic: compute independence at runtime
- Many techniques to (quickly) approximate DPOR
  - Optimal exploration is slow to compute



Unfoldings

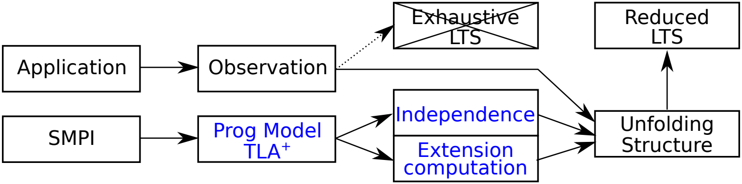
- Event structure: concurrency, conflict, causality
- NP-difficult to build in general case

Unfolding-based DPOR [Rodriguez et Al. 2015]

- Guide DPOR with Unfolding structure, for shared memory with mutexes

Efficient UDPOR for asynchronous distributed apps

Contributions of The Anh Pham PhD work



- Abstract model of asynchronous communications and locks
  - AsyncSend, AsyncRecv, AsyncLock, Unlock, WaitAny, TestAny, LocalCompute
- Formal specification of the programming model in TLA+
- Efficient computation of UDPOR extension in this model

TLA+ specification of MPI programming model

Actions' specification in TLA+

```
AsyncReceive(aId, mbId, data_addr, comm_addr) ==
  /\ aId \in ActorsIds
  /\ mbId\in MailboxesIds
  /\ data_addr \in Addresses
  /\ comm_addr \in Addresses
  /\ pc[aId] \in ReceiveIns
  /\ \/ /\ \/ Len(Mailboxes[mbId]) = 0
           \/ /\ Len(Mailboxes[mbId]) > 0
              /\ Head(Mailboxes[mbId]).status = "receive"
        /\ LET comm ==
                 [id |-> commId,
                  status |-> "receive",
                  src |-> NoActor,
                  dst |-> aId,
                  data_src |-> NoAddr,
                  data_dst |-> data_addr]
           IN
             /\ Mailboxes' = [Mailboxes EXCEPT ![mbId] = Append(Mailboxes[mbId],
                                                                comm)]
             /\ Memory' = [Memory EXCEPT ![aId][comm_addr] = comm.id]
             /\ UNCHANGED <<Communications>>
             /\ commId' = commId+1
```

Mailbox

- FIFO of comms to be matched
- Cannot mix send and recv comms

Mutex

- FIFO of requesting actors
- Owner is the FIFO head

Independence theorems

- Expressed in TLA+, proved manually
- Example: *AsyncSend* and *AsyncRecv* are independent.

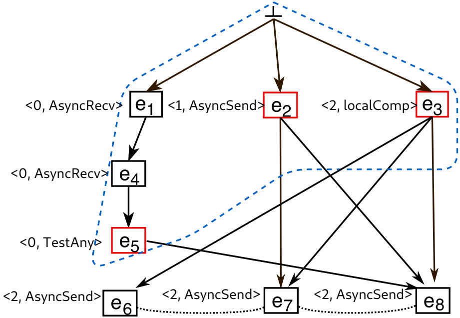
Modeling MPI

- 160 MPI functions are modeled with these 7 actions: P2P and RMA

Computing extensions of a configuration

- A configuration represents a class of equivalent executions
- The extensions are the possible *next steps*

Actor0	req0a = AsyncRecv(m, _) req0b = AsyncRecv(m, _) TestAny(req0a, req0b)
Actor1	req1 = AsyncSend(m, _)
Actor2	localComputation req2 = AsyncSend(m, _)



- Example: Compute the possible extensions using action req2 = AsyncSend
- Naive algorithm: consider all subsets of the configuration
- In this model: at most 3 events to consider
  - Extending with AsyncSend? consider { prevEvt(a,C), AsyncSend, Test}.
  - Polynomial complexity, Computed incrementally since actions are persistent

Conclusion on formal verification

Current state: algorithm *on par* with SotA

- Efficiently reduces small examples such as coordinator/workers:
  - 3 workers: 1,356,444 traces w/o reduction (17mn); 2 traces (0.2s) with UDPOR
  - 4 workers: timeout without reduction; 6 traces (2.5s) with UDPOR
- Ongoing: integration in SimGrid (E. Azimi, fixed-term engineer Inria)
  - PhD prototype required a manual encoding of the MPI programs

Future work: reduction algorithms are difficult to compare

- Generic workbench for formal assessment tools of MPI programs
- SimGrid as a framework to author/study new exploration algorithms (E. Azimi)

Future work: Other exploration algorithms

- Bounded UDPOR: sound exploration of a graph subset
- Liveness properties: reduction must preserve cycles

Statistical model-checking

Problem statement

- SimGrid emulates a single trajectory of a MPI application
- Mc SimGrid explores all possible trajectories, but time is abstracted away
- How to get the probable outcome of several interesting trajectories?

Numerical model-checking

- The system must be Markovian and modeled as a probability matrix
- Exact computation of probabilities from the model

Statistical model-checking

- No constraint on the model nor property, but approximated values
- Controlled Monte-Carlo simulations. Evaluate probabilities, estimate values
  - Select paths satisfying interesting properties (rare events, other)
- Run as many experiments as needed to reach the expected confidence interval

Statistical model-checking in Hac Specis (Yann Duplouy, Postdoc, Nancy)

- Build an experimental SMC tool using SimGrid as system model

SimGrid-StatMC

The tool

- Experiment runner forked from the Cosmos tool (Mexico)
- Interact with SimGrid simulations:
  - Controlled random (random seed and laws; random background load)
  - Retrieve the values of traced variables to evaluate the property

Example: evaluation of the BitTorrent protocol

- Average completion time, according to noisy network performance
- Mean download time per node, according to expected time between failures
- Assess download algorithms against several failure models

Future work

- Better integration between Cosmos and SimGrid (long-term support)
- Make Cosmos outcomes available to HPC users (rare events' studies)

Conclusion

Hac Specis bridges the gaps

- Formal methods made easier for practitioners (MPI as an input)
- Complementary approaches available together

Formal verification: correctness

- Contribution: UDPOR reduction for MPI model (state space explosion)
- Ongoing: integration into Mc SimGrid ; production-ready tools
- Future: other exploration algorithms and properties (bounded, liveness)

Statistical model-checking: performance

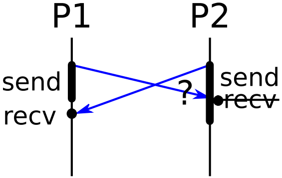
- Contribution: working prototype of statistical model-checker in SimGrid
- Ongoing/Future: leverage better integration between SimGrid and Cosmos

Future

- Verification and performance evaluation of non-MPI programs
- Further combining approaches: verification from a real-life checkpoint

Modeling of Point-to-Point comms

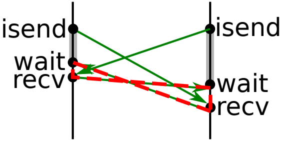
The MPI Standard is not helping :(



Are Blocking Send Blocking?

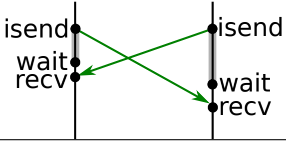
- (both are blocking sends)
- This should block, but **may** not
- Blocking until delivered to recv's buffers

Modeling the Strict Semantic (zero-buffer)



- Split send in `isend+wait`
- `recv` must be posted for sender's wait to finish
- `isend+wait` still atomic ($\text{wait}_2 > \text{recv}_2$)
- Deadlock can be seen in causality loop

Modeling the Relaxed Semantic (infinite-buffer)



- Sender's `wait` not linked to `recv`
- No deadlock anymore