Using TM for high-performance Discrete-Event Simulation on multi-core architectures

EuroTM 2013 Workshop, Prague, April 2013

Olivier Dalle

Université Nice Sophia Antipolis & INRIA Sophia Antipolis

April 14, 2013
Outline

1. Discrete Event Simulation
   - Types of Simulation Models
   - Programming styles

2. Parallel Discrete-Event Simulation
   - Parallelization approaches
   - Rationale of existing approaches

3. Using TM for PDES
   - Rationale for using TM
   - A tentative algorithm
   - Expected Benefits

4. Conclusions & Future Works
Model Charactérisation

Discrete Event Simulation (D.E.S.)
Model Charactérisation

Discrete Event Simulation (D.E.S.)
- Stochastic Models (vs. deterministic)
Discrete Event Simulation (D.E.S.)
- Stochastic Models (vs. deterministic)
- Dynamic models (vs. static)
Model Charactérisation

Discrete Event Simulation (D.E.S.)
- Stochastic Models (vs. deterministic)
- Dynamic models (vs. static)
- Discrete (Time) Models (vs. continuous)
Model Charactérisation

Discrete Event Simulation (D.E.S.)
- Stochastic Models (vs. deterministic)
- Dynamic models (vs. static)
- Discrete (Time) Models (vs. continuous)
Simulation of the system’s dynamics can be:
Programming Styles

Simulation of the system’s dynamics can be:

- **Event-Based**
  - Schedule events at specific times
    - exogeneous: from external source (starter, trace, ...)
    - endogenous: from previous event processing
  - Events are associated to callbacks
Simulation of the system’s dynamics can be:

- **Event-Based**
  - Schedule events at specific times
    - exogeneous: from external source (starter, trace, …)
    - endogenous: from previous event processing
  - Events are associated to callbacks

- **Process-Based**
  - Model composed of logical entities
  - Each with a separate control flow (logical process)
    - Process state changes when awaken
    - Sleep until next activation time
Example

**Event-based**

```python
def init_callback(evt):
    schedule(takeoff_cb, t=10, {from=london, to=paris})
    schedule(takeoff_cb, t=20, {from=paris, to=newyork})

def takeoff_cb(evt):
    if (evt.from==london && evt.to==paris):
        schedule(landing_cb, t=current+1.1, {evt.from, evt.to})
    ...

def landing_cb(evt):
    ...
```
Example

Process-based

```python
def init_thread()
    wait(10)
    create_thread(flight_thread, {from=london, to=paris})
    wait(10)
    create_thread(flight_thread, {from=paris, to=newyork})
    ...

def flight_thread(params)
    if (params.from == london && param.to==paris)
        wait(1.1)
    wait(1.1)
    ...
```
Parallelization approaches

- Multiple replications in parallel (MRIP)
  - Repeat simulation for statistics/confidence interval
  - No synchronisation
Parallelization approaches

- Multiple replications in parallel (MRIP)
  - Repeat simulation for statistics/confidence interval
  - No synchronisation

- Parallel execution of one simulation

  Usual assumptions:
  - distributed memory
  - message passing
  - logical processes
Parallelization of a single D.E.S.:

- **Conservative (Chandi & Misra, 1981)**
  - Only process "safe" events
  - Safe iff no event with earlier time
  - Synchronization: computes LBTS, avoid deadlocks...

- **Optimistic (Jefferson, 1982)**
  - Allow unsafe events
  - Snapshot/Rollback + Anti-message
  - Synchronization: computes GVT (in background)
Rationale of existing (past) approaches

- Local Causality Constraint (Fujimoto, 2000)
Rationale of existing (past) approaches

- Local Causality Constraint (Fujimoto, 2000)
  - process events in non decreasing order. Fujimoto:
Local Causality Constraint (Fujimoto, 2000)
- process events in non decreasing order. Fujimoto:
  
  "It is clear that if one violates the LCC, causality errors MAY occur"
Local Causality Constraint (Fujimoto, 2000)

- process events in non-decreasing order. Fujimoto:
  
  "It is clear that if one violates the LCC, causality errors MAY occur"

  "One must decide whether or not $E_{10}$ can be processed concurrently with $E_{20}$. How does the simulator decide without simulating $E_{10}$?"

- No shared memory
- Not supported in most architectures
Local Causality Constraint (Fujimoto, 2000)

- process events in non decreasing order. Fujimoto:
- "It is clear that if one violates the LCC, causality errors MAY occur"
- "One must decide whether or not $E_{10}$ can be processed concurrently with $E_{20}$. How does the simulator decide without simulating $E_{10}$?"
- Answer is (was): do NOT process $E_{20}$ ...
Rationale of existing (past) approaches

- Local Causality Constraint (Fujimoto, 2000)
  - process events in non decreasing order. Fujimoto:
  - "It is clear that if one violates the LCC, causality errors MAY occur"
  - "One must decide whether or not \( E_{10} \) can be processed concurrently with \( E_{20} \). How does the simulator decide without simulating \( E_{10} \)?"
  - Answer is (was): do NOT process \( E_{20} \) . . .

- No shared memory
Rationale of existing (past) approaches

- **Local Causality Constraint (Fujimoto, 2000)**
  - process events in non decreasing order. Fujimoto:
  - "It is clear that if one violates the LCC, causality errors MAY occur"
  - "One must decide whether or not $E_{10}$ can be processed concurrently with $E_{20}$. How does the simulator decide without simulating $E_{10}$?"
  - Answer is (was): do NOT process $E_{20}$ . . .

- No shared memory
  - Not supported in most architectures
Breaking limitation using TM?

- Independent parallel sequence need no synchronization
  - As long as no causal relation is found
  - Let simulator discover causality and rollback transaction
- Shared memory IS now available in most architecture
Tentative simulation algorithm
Data Structure

Data structure assumptions:

- First events sorted in Time Stamp order
- Events have state $\in \text{pending, processing, commit, complete}$
- Test-and-set-like atomic instruction to change state (and assign event)
Tentative simulation algorithm
Algorithm (1/2)

// Processing phase
Transaction.Start()

NP = select next pending event and change state to processing

Process event:

   Prepare new state
   Produce new events:
      Foreach Evt
      Insert({Evt,pending,Time})
      conflicts = causal_conflict(Evt, non_pending_after(Evt))
      if #conflicts > 0:
         invalidate_transaction(conflicts)
Tentative simulation algorithm

Algorithm (2/2)

// Commit phase
state = commit
if (prev_causal.state != complete):
    wait_on_queue(prev_causal.dependents)
if (invalidated)
    invalidate_and_wakeup(dependents)
    Transaction.Abort()
else
    state = complete
    wakeup(dependents)
    Transaction.Commit()
Relaxed Causality (compared to strict LCC)

Where is it involved:
Relaxed Causality (compared to strict LCC)

Where is it involved:

- used to decide causal conflict
Relaxed Causality (compared to strict LCC)

Where is it involved:

- used to decide causal conflict
- to compute causal dependency
Relaxed Causality (compared to strict LCC)

Where is it involved:

- used to decide causal conflict
- to compute causal dependency

Examples:
Relaxed Causality (compared to strict LCC)

Where is it involved:
- used to decide causal conflict
- to compute causal dependency

Examples:
- Includes TS order (conflict with next, dependent = previous)
Relaxed Causality (compared to strict LCC)

Where is it involved:

- used to decide causal conflict
- to compute causal dependency

Examples:

- Includes TS order (conflict with next, dependent = previous)
- Identify causal chains of events
  - Protocols: peers on same router, sender/receiver relation
Relaxed Causality (compared to strict LCC)

Where is it involved:
- used to decide causal conflict
- to compute causal dependency

Examples:
- Includes TS order (conflict with next, dependent = previous)
- Identify causal chains of events
  - Protocols: peers on same router, sender/receiver relation
- Identify chains of unrelated events
  - Lookahead information: peers on separate networks have no causal relation happening in less than latency time
STM requirements

DracoSTM/Boost.STM like:
STM requirements

DracoSTM/Boost.STM like:

- invalidate-on-commit
STM requirements

DracoSTM/Boost.STM like:
- invalidate-on-commit
- composition
Expected Benefits

Observations & Speculations:

- Conflict occur at insertion time
Expected Benefits

Observations & Speculations:
- Conflict occur at insertion time
- Rollback likely if insertion in "close" future
Expected Benefits

Observations & Speculations:

- Conflict occur at insertion time
- Rollback likely if insertion in "close" future
- Great potential if insertions in mid-range to far future
Expected Benefits

Observations & Speculations:

- Conflict occur at insertion time
- Rollback likely if insertion in "close" future
- Great potential if insertions in mid-range to far future
- Only rollback dependent transactions
  - vs Optimistic PDES: rollback everything up to latest consistent cut (GVT)
  - May save lots of memory (very large systems with infrequent rollbacks)
Work status: Early Work-In-Progress
Work status & Future works

Work status: Early Work-In-Progress

- C++ dev just started: new C++11/Boost-based PDES
Work status & Future works

Work status: Early Work-In-Progress

- C++ dev just started: new C++11/Boost-based PDES
- Perf study of relevant data structures
Work status & Future works

Work status: Early Work-In-Progress

- C++ dev just started: new C++11/Boost-based PDES
- Perf study of relevant data structures
- Perf. comparisons lock-based vs. STM (?)
Work status: Early Work-In-Progress

- C++ dev just started: new C++11/Boost-based PDES
- Perf study of relevant data structures
- Perf. comparisons lock-based vs. STM (?)
- Perf. comparisons of STM libs