PDES

(Don’t worry, this isn’t calculus)
Parallel Discrete Event Simulation
Classification of Computer Simulations

Computer Simulation
Classification of Computer Simulations

Computer Simulation

- Continuous
- Discrete
Classification of Computer Simulations

- Continuous
- Discrete
  - Time stepped
  - Event driven
Classification of Computer Simulations

<table>
<thead>
<tr>
<th>Computer Simulation</th>
<th>Continuous</th>
<th>Discrete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time stepped</td>
<td>Event driven</td>
</tr>
</tbody>
</table>
Discrete Event Simulations

- Logical Processes (LPs) execute events
- Executing an event updates the LP’s state
- Events have a virtual timestamp
- Events must be executed in order
Applications

- Traffic analysis
- Military battles
- Networks
- Circuits
- Economic models
- and many more...
Implementation

- Single event queue
- Sorted by timestamp
- Loop over queue and execute events
- Efficiency depends on queue used
- Very simple
Discrete Event Simulations

LPs

A

B

C

Event List

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
</tr>
<tr>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>17</td>
<td>C</td>
</tr>
</tbody>
</table>

Sim Time

6
Discrete Event Simulations

LPs

A → 6
B → 12
C → 17

Event List

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
</tr>
<tr>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>17</td>
<td>C</td>
</tr>
</tbody>
</table>

Sim Time

6
## Discrete Event Simulations

### LPs

- A
- B
- C

### Event List

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>B</td>
</tr>
<tr>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>15</td>
<td>C</td>
</tr>
<tr>
<td>17</td>
<td>C</td>
</tr>
</tbody>
</table>

### Sim Time

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
Pseudocode

while (running) {
    Event* e = eventList.pop();
    LP* lp = e->destination();
    lp->execute(e);
}

Pretty simple right?
Limitations of Sequential DES

- Millions of LPs
- Billions of events to simulate
- Sequential simulations will take too long
How will parallelization help?

- Events are generally very small
- Simulations can have billions of events

Benefits of parallelism come from executing many events at once
How do we parallelize DES?

- Distribute LPs across processors
- Each processor has its own event list and virtual clock
- How to synchronize event lists and clocks?
(Super) Naive Implementation

```c
while (running) {
    AllReduce(simTime);
    if (events.top()->ts==simTime) {
        Event* e = events.pop();
        LP* lp = e->destination();
        lp->execute(e);
    }
}
```
(Super) Naive Implementation

- Exactly matches sequential semantics
- NO PARALLELISM
- Way too much synchronization

Moral of the Story:
We need to relax our ordering restrictions!
What is the fundamental problem?

Can PE 0 safely execute any events?

<table>
<thead>
<tr>
<th>Event List</th>
<th>Sim Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

PE 0

<table>
<thead>
<tr>
<th>Event List</th>
<th>Sim Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

PE 1
What is the fundamental problem?

<table>
<thead>
<tr>
<th>Event List</th>
<th>Sim Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 A</td>
<td>6</td>
</tr>
<tr>
<td>12 B</td>
<td></td>
</tr>
<tr>
<td>13 B</td>
<td></td>
</tr>
<tr>
<td>17 C</td>
<td></td>
</tr>
</tbody>
</table>

PE 0

What if the first event on PE 1 generates event (5,A)?

<table>
<thead>
<tr>
<th>Event List</th>
<th>Sim Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 D</td>
<td>4</td>
</tr>
<tr>
<td>7 F</td>
<td></td>
</tr>
<tr>
<td>21 E</td>
<td></td>
</tr>
<tr>
<td>25 D</td>
<td></td>
</tr>
</tbody>
</table>

PE 1

Can PE 0 safely execute any events?
What is the fundamental problem?

Can PE 0 safely execute any events?

What if the first event on PE 1 generates event (5,A)?

Causality Error!
What is the fundamental problem?

How can we ever execute events!??
Two Approaches

● Conservative (don’t screw up)
  ○ Only execute events we know won’t be preempted
  ○ High amount of synchronization
  ○ Low amount of parallelism
  ○ Inflexible
  ○ Simple

● Optimistic (if we screw up, we’ll fix it)
  ○ Execute events freely
  ○ If a causality error occurs, rollback the processor
  ○ High amount of parallelism
  ○ Flexible
  ○ Complex
Conservative (Windowed)

- Requires a model-specific lookahead
- Determine the min timestamp in the system
- Execute events in (min + lookahead) window
Conservative (Windowed)

- Requires a model-specific lookahead
- Determine the min timestamp in the system
- Execute events in (min + lookahead) window
(Super) Naive Implementation

while (running) {
    AllReduce(simTime);
    if (events.top()->ts == simTime) {
        Event* e = events.pop();
        LP* lp = e->destination();
        lp->execute(e);
    }
}
Windowed Implementation

while (running) {
    AllReduce(simTime);
    while (events.top()->ts < simTime + lookahead) {
        Event* e = events.pop();
        LP* lp = e->destination();
        lp->execute(e);
    }
}
Windowed Analysis

- Very similar to naive solution
- Performance depends on lookahead
- Low lookahead = low parallelism and high synchronization
- Workload can be unbalanced
Optimistic

- Execute events freely
- When an event is received with a smaller timestamp than your clock, rollback
- How do we rollback efficiently?
- How do rollbacks affect performance?
- How many PEs will a rollback affect?
Rollbacks

- Save previous events (How many?)
- Revert your own state (How?)
- Cancel sent events (How?)
Saving Previous Events

● Events take up memory
  ○ Limits how many events we can save
  ○ Need to reclaim memory periodically

● What can we safely reclaim?

● Find the Global Virtual Time (GVT)
  ○ Minimum clock time of the system
  ○ Everything prior to this can be reclaimed
  ○ Events with observable effects can be committed
GVT

- Global synchronization required
- All events must be accounted for
- Can be synchronous or asynchronous
Reverting Your State

- **State saving**
  - Save the states of LPs after each event
  - Rolling back is equivalent to reverting states
  - High memory consumption
  - Need to reclaim memory more often

- **Reverse computation**
  - During rollback execute events in reverse
  - Better for memory
  - Overhead of executing in reverse
  - Reverse computation is complex
  - Can compilers help?
Cancelling Events

- Events sent erroneously must be cancelled
- First we must find the event
  - If it’s local, that’s easy
  - If it’s remote we need to send an anti-event
- Then we must cancel it
  - If they weren’t executed, just delete them
  - If they have been executed, do a rollback
- Rollbacks can snowball out of control
Pseudocode

while (running) {
    while (executing_events) {
        check_for_rollbacks();
        Event* e = events.pop();
        LP* lp = e->destination();
        lp->execute(e);
    }
    compute_gvt();
}

Summary

Two Main Classes of PDES:

● **Conservative**
  ○ Low parallelism/High synchronization cost
  ○ Model dependent
  ○ Low memory footprint

● **Optimistic**
  ○ High parallelism/Low synchronization cost
  ○ Model independent
  ○ Memory Hungry
  ○ Rollbacks can snowball