Charades

An Adaptive Parallel Discrete Event Simulation Framework on Charm++

Eric Mikida
Charm++ Adaptive Discrete Event Simulator
Brief PDES Description

• Simulation made up of Logical Processes (LPs)
• LPs process events in timestamp order
• Synchronization is conservative or optimistic
• Periodically compute global virtual time (GVT)
Performance Metrics

Event Rate = $\frac{E_{\text{committed}}}{s}$

Event Efficiency = $\frac{E_{\text{committed}}}{E_{\text{total}}}$
Performance Tuning Tradeoffs

• Shown good performance on benchmarks
  – We can execute/send large numbers of fine-grained events effectively

• How do we adapt to improve performance in the less ideal cases?
  – What events are we actually executing/sending
GVT Computation

- Global computation, required frequently
- Common solution blocks entirely during computation
  - Side effect of blocking is bounded-optimism
  - Leads to higher event efficiency
GVT Tradeoff

- One coupled tuning knob
- Common case is high synchronization
GVT Tradeoff

• One coupled tuning knob
• Common case is high synchronization
• As we lower synchronization, we lose efficiency
GVT Tradeoff

• One coupled tuning knob
• Common case is high synchronization
• As we lower synchronization, we lose efficiency
Reducing Synchronization Costs

Asynchronous Reduction

Event Execution → Event Execution
QD Red FC

Continuous Execution

Event Exec. Event Execution Event Execution
CD Red FC CD Red FC
Performance on Blue Waters

Algorithm Speedups (Leash Trigger)

<table>
<thead>
<tr>
<th></th>
<th>Sync</th>
<th>Cont</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHOLD Base</td>
<td>98%</td>
<td>95%</td>
</tr>
<tr>
<td>PHOLD Work</td>
<td>76%</td>
<td>52%</td>
</tr>
<tr>
<td>PHOLD Event</td>
<td>84%</td>
<td>60%</td>
</tr>
<tr>
<td>PHOLD Combo</td>
<td>93%</td>
<td>31%</td>
</tr>
<tr>
<td>DFly Uniform</td>
<td>62%</td>
<td>36%</td>
</tr>
<tr>
<td>DFly Worst</td>
<td>91%</td>
<td>2%</td>
</tr>
<tr>
<td>DFly Trans</td>
<td>85%</td>
<td>27%</td>
</tr>
<tr>
<td>DFly NN</td>
<td>93%</td>
<td>67%</td>
</tr>
<tr>
<td>Traffic Base</td>
<td>96%</td>
<td>55%</td>
</tr>
<tr>
<td>Traffic Src</td>
<td>97%</td>
<td>16%</td>
</tr>
<tr>
<td>Traffic Dest</td>
<td>96%</td>
<td>52%</td>
</tr>
<tr>
<td>Traffic Route</td>
<td>97%</td>
<td>15%</td>
</tr>
</tbody>
</table>
Load Balancing on Blue Waters

Load Balancing Speedup (Count Trigger)

- CPU Time
- Commit Count

Speedup

- PHOLD Work
- PHOLD Event
- PHOLD Combo
- Traffic Src
- Traffic Dest
- Traffic Route
Load Balancing on Blue Waters

Load Balancing Efficiency (Count Trigger)
Decoupling the Tuning Knob

- GVT can tune for synchronization
- LB can tune for efficiency
Decoupling the Tuning Knob

- GVT can tune for synchronization
- LB can tune for efficiency
Continuous GVT w/ Load Balancing
Bucketed GVT Scheme
Bucketed GVT Scheme

1. Divide entire simulation timeline into buckets
Bucketed GVT Scheme

1. Divide entire simulation timeline into buckets
2. Monitor incoming/outgoing events
Bucketed GVT Scheme

1. Divide entire simulation timeline into buckets
2. Monitor incoming/outgoing events
3. As buckets get passed, advance GVT

<table>
<thead>
<tr>
<th>Sent</th>
<th>Recvd</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Virtual Time
Bucketed GVT Scheme

1. Divide entire simulation timeline into buckets
2. Monitor incoming/outgoing events
3. As buckets get passed, advance GVT

How do we know when a bucket is passed?
Bucketed GVT Scheme

Each PE contributes to a min reduction when it passes a bucket boundary.
Each PE contributes to a min reduction when it passes a bucket boundary.

Contributes 1, for 1 bucket passed.
Bucketed GVT Scheme

Each PE contributes to a min reduction when it passes a bucket boundary.

Contributes 1, for 1 bucket passed

Contributes 2, for 2 buckets passed
Bucketed GVT Scheme

- Once the reduction completes, we know everyone has passed at least some buckets
- Start a series of “tuple” reductions of bucket counts, and buckets passed
- Similar to completion detection with extra information
Bucketed GVT Scheme

• Result of count reductions for n buckets
  – Sent counts: \([s_1, s_2, \ldots, s_n]\)
  – Received counts: \([r_1, r_2, \ldots, r_n]\)
  – New min bucket passed: \(k\)

• Find \(x\) such that \(s_i = r_i\) for all \(i \leq x\) and \(x \leq k\)

• Advance GVT \(x\) buckets, \(k-x\) keep reducing
Bucketed GVT Scheme

First 2 buckets counts match, and all PEs have passed at least 3 buckets
Bucketed GVT Scheme

First 2 buckets counts match, and all PEs have passed at least 3 buckets

Advance GVT 2 buckets, continue waiting on the 3rd, and possibly pull in more
Early Results

• Comparable, or better, to Continuous
• More manageable/robust
• Easier to tune/understand
• Opportunity for adaptive event control
  – Adaptively hold back high-risk events
  – Reduce overall communication load
Adaptive Event Delay

GVT

Offset From GVT

Reg: 101
Anti: 1

Reg: 90
Anti: 3

Reg: 40
Anti: 8

Reg: 38
Anti: 30

Reg: 27
Anti: 26

Reg: 5
Anti: 5
Adaptive Event Delay

High-Risk Events

GVT

Offset From GVT

Reg: 101
Anti: 1

Reg: 90
Anti: 3

Reg: 40
Anti: 8

Reg: 38
Anti: 30

Reg: 27
Anti: 26

Reg: 5
Anti: 5
Questions?