Horizon
Runtime Efficient Event Scheduling in Multi-threaded Network Simulation

Georg Kunz, Mirko Stoffers, James Gross, Klaus Wehrle

http://www.comsys.rwth-aachen.de/
Motivation

• Need for Complex Network Simulation Models
  ▶ Detailed channel and PHY characteristics
  ▶ Large scale P2P and Internet backbone models
  ▶ High processing and runtime demand

• Proliferation of Multi-processor Systems
  ▶ Desktop: 4-8 cores, servers: 24 cores
  ▶ “Desktop Cluster”
  ▶ Cheap, powerful commodity hardware
Motivation

- **Need for Complex Network Simulation Models**
  - Detailed channel and PHY characteristics
  - Large scale P2P and Internet backbone models
  - High processing and runtime demand

- **Proliferation of Multi-processor Systems**
  - Desktop: 4-8 cores, servers: 24 cores
  - “Desktop Cluster”
  - Cheap, powerful commodity hardware

⇒ Utilize Parallelization to Cut Runtimes?
• Parallelization Introduces Overhead
  ▶ Thread synchronization, management of shared data
  ▶ Increased management overhead per event
  ▶ Negative impact on events of low complexity
Motivation: Downside of Parallelization

- **Parallelization Introduces Overhead**
  - Thread synchronization, management of shared data
  - Increased management overhead per event
  - Negative impact on events of low complexity

- **Dilemma / Tradeoff**
Motivation: Downside of Parallelization

- **Parallelization Introduces Overhead**
  - Thread synchronization, management of shared data
  - Increased management overhead per event
  - Negative impact on events of low complexity

- **Dilemma / Tradeoff**

  ![Performance vs. Overhead Diagram]

  ⇒ **Minimize Parallelization Overhead**
Horizon: Approach

- **Horizon**
  - Focus on multi-processor systems
  - Centralized architecture
  - Conservative synchronization
    - Determine independent events

- **Expanded Events**
  - Modeling paradigm
  - Per event lookahead
  - Identify independent events
Horizon: Approach

- **Horizon**
  - Focus on multi-processor systems
  - Centralized architecture
  - Conservative synchronization
    - Determine independent events

- **Expanded Events**
  - Modeling paradigm
  - Per event lookahead
  - Identify independent events
• Expanded Events
  ▶ Model processes that span period of time
  ▶ Augment discrete events with durations
  ⇒ Discrete events span period of *simulated* time
Horizon: Expanded Events

- Expanded Events
  - Model processes that span period of time
  - Augment discrete events with durations
  - Discrete events span period of **simulated** time

![Expanded Event Diagram]
• **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - Discrete events span period of **simulated** time
• **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - Discrete events span period of *simulated* time
Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - Discrete events span period of *simulated* time
Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - Discrete events span period of *simulated* time

- **Independent Events**
  - Events starting between $t_{\text{start}}$ and $t_{\text{end}}$
  - Do not depend on results generated by overlapping event
  - Modeling paradigm
Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - Discrete events span period of *simulated* time

- **Independent Events**
  - Events starting between $t_{start}$ and $t_{end}$
  - Do not depend on results generated by overlapping event
  - Modeling paradigm
Challenges
How to reduce parallelization overhead?
We Address Two Challenges

- Thread Synchronization Overhead
- Event Scheduling Overhead
We Address Two Challenges

- Thread Synchronization Overhead
- Event Scheduling Overhead
Thread Synchronization Overhead: Challenge

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures

- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources
Thread Synchronization Overhead: Challenge

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures

- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources
Thread Synchronization Overhead: Challenge

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures

- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources
Thread Synchronization Overhead: Challenge

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures

- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources
**Thread Synchronization Overhead: Challenge**

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchonization involves future event set
    - Workers wait for incoming jobs
    - Access to shared data structures
  - Increases Threading Overhead
    - Sys-calls into kernel
    - Context switches

- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources
Thread Synchronization Overhead: Approach

- **Challenge**
  - Suspending Threads Increases Overhead

- **Observation**
  - Simulations run on dedicated hardware
  - Freeing-up CPUs is needless
  - Crucial to minimize offloading delay

- **Approach**
  - Use busy waiting for synchronization
  - Master actively pushes jobs to workers
Thread Synchronization Overhead: Solution

- **Push-based Event Offloading**
  - Eliminate shared work queue
  - Introduce local synch. buffer per thread
  - Spinlock for future event set

- **Synchronization Buffer**
  - Master assigns jobs to empty buffer
  - Workers spin on empty buffer

- **Additional Benefit**
  - Master can identify busy threads
  - Master handles event instead of worker
  - Make use of scheduler CPU
Thread Synchronization Overhead: Solution

- **Push-based Event Offloading**
  - Eliminate shared work queue
  - Introduce local synch. buffer per thread
  - Spinlock for future event set

- **Synchronization Buffer**
  - Master assigns jobs to empty buffer
  - Workers spin on empty buffer

- **Additional Benefit**
  - Master can identify busy threads
  - Master handles event instead of worker
  - Make use of scheduler CPU
Thread Synchronization Overhead: Solution

- **Push-based Event Offloading**
  - Eliminate shared work queue
  - Introduce local synch. buffer per thread
  - Spinlock for future event set

- **Synchronization Buffer**
  - Master assigns jobs to empty buffer
  - Workers spin on empty buffer

- **Additional Benefit**
  - Master can identify busy threads
  - Master handles event instead of worker
  - Make use of scheduler CPU
Challenges and Solutions

• We Address Two Challenges

- Thread synchronization Overhead
- Event Scheduling Overhead
Event Scheduling Overhead: Challenge

- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished
Event Scheduling Overhead: Challenge

- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished
Event Scheduling Overhead: Challenge

- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished
• **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

• **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished
Event Scheduling Overhead: Challenge

- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished

---

**Doubles Overhead per Event**
- Creation, deletion of events
- Insertion, removal from FES

---

**Georg Kunz, Mirko Stoffers, James Gross, Klaus Wehrle**
Event Scheduling Overhead: Approach

**Observations**
- Push-based synchronization
  - Upper bound for simultaneously offloaded events: \#CPUs
  - Upper bound for simultaneously existing barriers: \#CPUs

**Approach**
- Avoid insertion into locked(!) message queue
- Each thread maintains barrier time of current event
- Pointer to earliest barrier enables fast lookup

![Diagram of OMNeT++ messages and barrier times](image-url)
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

1. Job: \( t_{\text{start}}: 0.0 \text{ s} \), \( t_{\text{end}}: 1.0 \text{ s} \)
2. Job: \( t_{\text{start}}: 0.5 \text{ s} \), \( t_{\text{end}}: 0.8 \text{ s} \)
3. Job: \( t_{\text{start}}: 1.2 \text{ s} \), \( t_{\text{end}}: 1.5 \text{ s} \)
Example Schedule:

Simulator:

Future Event Set

<table>
<thead>
<tr>
<th>Event</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 1</td>
<td>0.5 s</td>
<td>0.8 s</td>
</tr>
<tr>
<td>Event 2</td>
<td>1.2 s</td>
<td>1.5 s</td>
</tr>
</tbody>
</table>

job:

<table>
<thead>
<tr>
<th>Event</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 1</td>
<td>0.0 s</td>
<td>1.0 s</td>
</tr>
</tbody>
</table>

job:

<table>
<thead>
<tr>
<th>Event</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 2</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

job:

<table>
<thead>
<tr>
<th>Event</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

job:

<table>
<thead>
<tr>
<th>Event</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 4</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job:

job:

job:

job:
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job: t_{start}: 1.2 s
t_{end}: 1.5 s

job: t_{start}: 0.0 s
t_{end}: 1.0 s

t_{end}: 1.0 s

t_{end}: 0.8 s

job: t_{start}: 0.5 s
t_{end}: 0.8 s

job: t_{end}: -

job: t_{end}: -
Event Scheduling Overhead: Solution

Example Schedule:
- Simulated time
- Future Event Set
  - Simulator:
    - t_start: 1.2 s
    - t_end: 1.5 s

Simulator:
- Future Event Set
  - t_start: 1.2 s
  - t_end: 1.5 s

<table>
<thead>
<tr>
<th>Job</th>
<th>t_start</th>
<th>t_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0 s</td>
<td>1.0 s</td>
</tr>
<tr>
<td>2</td>
<td>0.5 s</td>
<td>0.8 s</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job: \( t_{\text{start}}: 0.0 \text{ s} \), \( t_{\text{end}}: 1.0 \text{ s} \)

job: \( t_{\text{start}}: 0.5 \text{ s} \), \( t_{\text{end}}: 0.8 \text{ s} \)

job: \( t_{\text{end}}: - \)

job: \( t_{\text{end}}: - \)
Example Schedule:

Simulator:

Future Event Set:
- \( t_{\text{start}}: 0.9 \text{ s} \) \( t_{\text{end}}: 1.1 \text{ s} \)
- \( t_{\text{start}}: 1.2 \text{ s} \) \( t_{\text{end}}: 1.5 \text{ s} \)

job: 
- \( t_{\text{start}}: 0.0 \text{ s} \) \( t_{\text{end}}: 1.0 \text{ s} \)
- \( t_{\text{end}}: - \)
- \( t_{\text{end}}: - \)
- \( t_{\text{end}}: - \)
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job: $t_{\text{start}}$: 0.9 s
$t_{\text{end}}$: 1.1 s

job: $t_{\text{start}}$: 1.2 s
$t_{\text{end}}$: 1.5 s

job: $t_{\text{start}}$: 0.0 s
$t_{\text{end}}$: 1.0 s

job: $t_{\text{end}}$: -
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

<table>
<thead>
<tr>
<th>Job</th>
<th>t_{start}</th>
<th>t_{end}</th>
</tr>
</thead>
<tbody>
<tr>
<td>job1</td>
<td>0.0 s</td>
<td>1.0 s</td>
</tr>
<tr>
<td>job2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>job3</td>
<td>0.9 s</td>
<td>1.1 s</td>
</tr>
<tr>
<td>job4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job: 
\[ t_{\text{start}}: 0.0 \text{ s} \]
\[ t_{\text{end}}: 1.0 \text{ s} \]

\[ t_{\text{end}}: 1.0 \text{ s} \]
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

\[ t_{\text{start}}: 1.2 \text{ s} \]

\[ t_{\text{end}}: 1.5 \text{ s} \]
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job: t_{start}: 1.2 s
t_{end}: 1.5 s

t_{end}: -
Event Scheduling Overhead: Solution

Example Schedule:

Simulator:

Future Event Set

job: 
\( t_{\text{end}}: - \)

job: 
\( t_{\text{end}}: - \)

job: 
\( t_{\text{end}}: - \)

job: 
\( t_{\text{start}}: 1.2 \text{ s} \)
\( t_{\text{end}}: 1.5 \text{ s} \)
\( t_{\text{end}}: 1.5 \text{ s} \)
Evaluation
How does it perform?
• Design Goal
  ▶ Measure event handling overhead

• “Null” Simulation Model
  ▶ 110 independent modules
  ▶ Null module
    ▪ Only re-schedules self-messages
    ▪ No other computations
  ▶ Execute 5.5 Million Events

⇒ Execution Time == Overhead
Evaluation: Thread Synchronization Overhead

~ 9.5x reduction

~ 1000x reduction
Evaluation: Analysis of Context Switches

### Pull-based Thread Synchronization

- **Y-axis**: Number of Context Switches
- **X-axis**: Number of Worker Threads

### Push-based Thread Synchronization

- **Y-axis**: Number of Context Switches
- **X-axis**: Number of Worker Threads
Evaluation: Event Scheduling Overhead

~ 1.5x reduction

~ 1.5x reduction

Georg Kunz, Mirko Stoffers, James Gross, Klaus Wehrle
Conclusions
The take away (barrier) message...
Conclusions

• Parallelization Increases Overhead
  ▶ Thread synchronization
  ▶ Event scheduling

• Two Approaches to Mitigate Overhead
  ▶ **Push-based** thread synchronization minimizes context switches
  ▶ **Local barrier information** replaces barrier messages

• Overhead Reduction
  ▶ Push-based synchronization: ~9.5x reduction
  ▶ Barrier algorithm: ~1.5x reduction
  ▶ Combined: ~14x reduction
Questions?
Backup Slides
Just in case someone asks…
Time Calibration

- **How to Obtain Accurate Timing Information?**
  - Utilize existing techniques

- **Emulation**
  - Accurate profiling on emulated hardware

- **Automatic Simulation Calibration**
  - Applicable to simple hardware platforms only

- **Protocol Specifications**
  - Independent of hardware platform

- **Expert Knowledge**
  - Requires experience and careful judgment
• **Parallel Scheduling**
  
  - Offload independent events to worker CPUs

![Diagram showing parallel scheduling with independent events and dependent events with timestamps](image)

• **Causal Correctness**
  
  - Increasing timestamp order among dependent events

• **Data Integrity**
  
  - Compose model of self-contained functional units
  
  - Functional units correspond to concept of logical processes