Managing Pending Events In Sequential and Optimistic Parallel Discrete Event Simulations

Julius D. Higiro

Miami University

October 8, 2017

Overview

Introduction

- **Discrete Event Simulation (DES)** is used to simulate systems of interacting objects.
- The objects in the simulation are referred to as logical processes (LPs) and model objects in the real world.
- LPs interact by exchanging time stamped events and process events in priority order with event priorities determined by their time stamp.
- Events that have yet to be processed are called "**pending events**".
- Data structures for handling pending events follow a priority queue based implementation.

Research Motivation

- Data structures for managing and prioritizing pending events play a critical role in ensuring efficient sequential and parallel simulations.
- The synchronization strategy in PDES can also impact the effectiveness of data structure because of the additional processing required during rollback recovery operations.
- A data structure that can effectively handle large number of concurrent events (**i.e. an average of one million events**).

Related Work

- Dickman et al., compared event list data structures that consisted of Splay Tree, STL Multiset and Ladder Queue. However, the focus of their paper was in developing a framework for handling pending event set data structure in shared memory PDES. A central component of their study was the identification of an appropriate data structure and design for the shared pending event set.
- Gupta et al., extended their implementation of Ladder Queue for shared memory Time Warp based simulation environment, so that it supports lock-free access to events in the shared pending event set. The modification involved the use of an unsorted lock-free queue in the underlying Ladder Queue structure.
- Marotta et al., have contributed to the study of pending event set data structures in threaded PDES through the design of the Non-Blocking Priority Queue (NBPQ) data structure. A pending event set data structure that is closely related to Calendar Queues with constant time performance.

Thesis

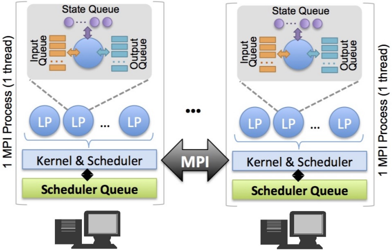
- **Thesis statement**: Multi-tier data structures (**2tLadderQ** and **3tHeap**) outperform all other data structures in sequential and optimistic parallel simulations.

- **Contribution:**

1. **3-Tier Heap.**
2. **2-Tier Ladder Queue (extension of Ladder Queue).**
3. **Identification of influential model characteristics for choice of queue.**

Overview

Parallel Simulator Overview



- Assessment of the data structures was conducted on MUSE.
- MUSE performs sequential and optimistically parallel simulations.
- MUSE uses Message Passing Interface (MPI) library for parallel processing.
- The kernel handles LP registration, event processing, state saving, synchronization and garbage collection.

Parallel Simulator Overview

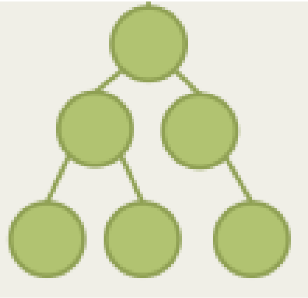
A scheduler queue is required to implement four key operations to manage pending events.

1. **Enqueue one or more future events**.
2. **Peek next event in priority order**.
3. **Dequeue events with the same time stamp for next LP**.
4. **Cancel pending events after a given time**.

Scheduler Queues

- MUSE contains 6 scheduling queues for managing pending events.
- The queues are classified into two categories: single-tier and multi-tier queues.
- Single-tier queues use only a single data structure to implement the 4 key operations.
- Multi-tier queues organizes events into tiers.
- Each tier is implemented using different data structures.

Binary Heap (**heap**)

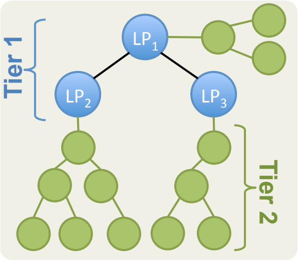


Time Complexity

Enqueue: $\log(e \cdot l)$
Dequeue: $\log(e \cdot l)$
Cancel: $z \cdot \log(e \cdot l)$
Legend:
l: #LPs
e: #events / LP
z: #canceled events

- It is a single-tier data structure that it is implemented as an array object.
- A std::vector is used as the backing container and C++11 algorithms (std::push_heap, st::pop_heap) are used to maintain the heap.
- The heap is prioritized based on time stamp with the lowest time stamp at the root of the heap.

2-tier Heap (**2tHeap**)

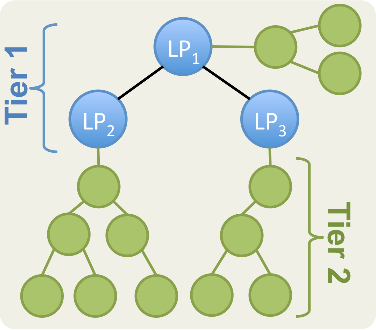


Time Complexity

Enqueue: $\log(e)+\log(l)$
Dequeue: $\log(e)+\log(l)$
Cancel: $z \cdot \log(e)+\log(l)$
Legend:
l: #LPs
e: #events / LP
z: #canceled events

- 2tHeap was designed to reduced the time complexity of cancel operations by subdividing events into two distinct tiers.
- The first tier has containers for each local LP on an MPI-process.
- Each of the the tier-1 containers has a heap of events to be processed by a given LP.
- A std::vector is used as the backing container for both tiers and standard algorithms are used to maintain the min-heap property for both tiers after each operation.

2-tier Fibonnaci Heap (**fibHeap**)

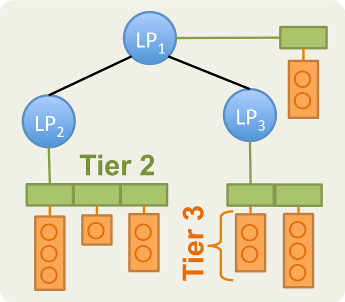


Time Complexity

Enqueue: $\log(e) + 1^*$
Dequeue: $\log(e) + 1^*$
Cancel: $z \cdot \log(e) + 1^*$
Legend:
l: #LPs
e: #events / LP
z: #canceled events
1^*: amortized constant

- The fibHeap is an extension of the 2tHeap data structure. It uses a Fibonnaci heap for scheduling LPs.
- The second tier is a binary heap data structure.

3-tier Heap (**3tHeap**)

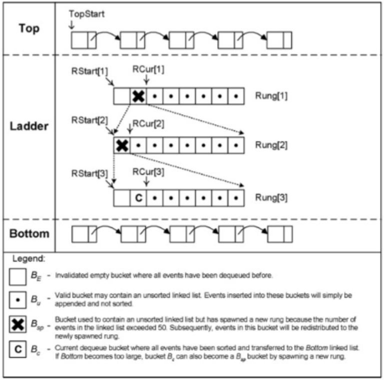


Time Complexity

Enqueue: $\log(\frac{e}{c}) + \log(l)$
Dequeue: $\log(l)$
Cancel: $e + \log(l)$
Legend:
l: #LPs
e: #events / LP
c: #concurrent events
z: #canceled events

- The 3tHeap builds upon 2tHeap by further subdividing the second tier into two tiers.
- The binary heap implementation for the first tier that manages LPs for scheduling has been retained from 2tHeap.
- Assuming each LP has c concurrent events on an average, there are $\frac{e}{c}$ tier-2 entries with each one having c pending events.

Ladder Queue (**ladderQ**)

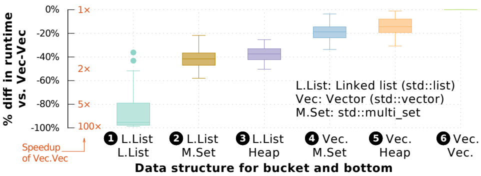


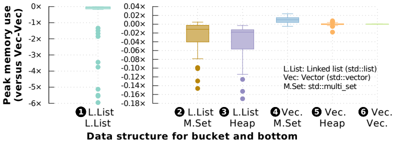
Time Complexity

Enqueue: 1^*
Dequeue: 1^*
Cancel: $e \cdot l$
Legend:
l: #LPs
e: #events / LP
1^*: amortized constant

- Ladder Queue is a priority queue implementation proposed by Tang et al. with amortized constant time complexity.
- There are two key ideas underlying the Ladder Queue, namely: minimize the number of events to be sorted and delay sorting of events as much as possible.

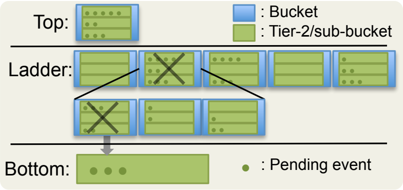
Ladder Queue (**ladderQ - Fine Tuned**)





- Comparison of execution time and peak memory using 6 different ladderQ configurations.
- L.List-L.List configuration was slowest and performed 85x slower than the Vec-Vec configuration.
- The increased performance of Vec-Vec comes at about a 6x increase in peak memory footprint when compared to L.List-L.List.

2-tier Ladder Queue (**2tLadderQ**)



Time Complexity

Enqueue: 1^*

Dequeue: 1^*

Cancel: $e \cdot l \div {}_{t2}k$

Legend:

l: #LPs

e: #events / LP

1^*: amortized constant

${}_{t2}k$: parameter

- 2-tier Ladder Queue is the proposed alternative to Ladder Queue because the cost of event cancellation during rollbacks is reduced.
- 2tLadderQ retains the amortized constant time complexity of ladderQ with performance gains during event cancellation.

Table: Comparison of algorithmic time complexities of different data structures

Legend – l: #LPs, e: #events / LP, c: #concurrent events, z: # canceled events, ${}_{t2}k$: parameter, 1^*: amortized constant

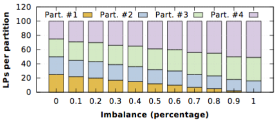
Name	Enqueue	Dequeue	Cancel
heap	$\log(e \cdot l)$	$\log(e \cdot l)$	$z \cdot \log(e \cdot l)$
2tHeap	$\log(e)+$ $\log(l)$	$\log(e)+$ $\log(l)$	$z \cdot \log(e)+$ $\log(l)$
fibHeap	$\log(e) + 1^*$	$\log(e) + 1^*$	$z \cdot \log(e) + 1^*$
3tHeap	$\log(\frac{e}{c}) + \log(l)$	$\log(l)$	$e + \log(l)$
ladderQ	1^*	1^*	$e \cdot l$
2tLadderQ	1^*	1^*	$e \cdot l \div {}_{t2}k$

Simulation Model - PHOLD

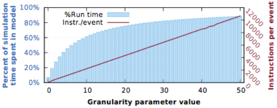
Table: **Parameters in PHOLD benchmark**

Parameter	Description
rows	Total number of rows in model.
cols	Total number of columns in model. #LPs = **rows** x **cols**
eventsPerLP	Initial number of events per LP.
delay or λ	Value used with distribution – Lambda (λ) value for exponential distribution *i.e.*, $P(x\|\lambda) = \lambda e^{-\lambda x}$.
%selfEvents	Fraction of events LPs send to self
granularity	Additional compute load per event.
imbalance	Fractional imbalance in partition to have more LPs on a MPI-process.
simEndTime	GVT when simulation logically ends.

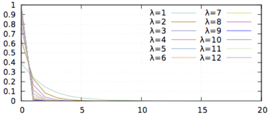
PHOLD - Key Parameters



(a) impact of imbalance



(b) impact of granularity



(c) impact of λ (delay)

- Imbalance parameter influences the partition.
- Granularity impacts the processing time of events.
- λ impacts the distribution of the time stamp values.

Simulation Model - PCS

Table: **Parameters in PCS simulation model**

Parameter	Description
rows	Total number of rows in model.
cols	Total number of columns in model.
	#Cells/LPs = **rows** x **cols**
portables	The portables represents a mobile phone unit that resides at the Cell for a period of time and then moves to one of four neighboring cells.

Table: **Parameters in PCS simulation model**

Parameter	Description
moveIntervalMean	Mean value of an exponential random distribution used to generate the time when a portable will move to an adjacent cell.
callIntervalMean	Mean value of an exponential random distribution from where the next call timestamp value is determined.
callDurationMean	Mean value of a poisson distribution used to generate the length/-duration of a call to a portable.

Simulation Model - PCS

Table: **Parameters in PCS simulation model**

Parameter	Description
#channels	The maximum number of channels assigned to each PCS cell.
imbalance	Fractional imbalance in partition to have more LPs on a MPI-process.
simEndTime	GVT when simulation logically ends.

Overview

1. Introduction/Motivation
2. Background
3. Methodology
4. GSA results
5. Sequential simulation results
6. Parallel simulation assessments
7. Conclusion and Future Work

Methodology

- Run different configurations of the PHOLD benchmark and PCS simulation in sequential and optimistically parallel simulations.
- Identify the most influential parameters that impact performance of the scheduler queues using **Generalized Sensitivity Analysis**.
- The data to be collected and assessed consists of the following:
  1. **simulation run time.**
  2. **peak memory usage.**
  3. **# of rollbacks.**
  4. **characteristics of key queue operations.**
  5. **# of network messages exchanged.**

Overview

Summary of GSA sequential results

- **Events per LP** - parameter with most influence using **PHOLD** benchmark.
- **lambda** - parameter with marginal influence using **PHOLD** benchmark.
- **Portables per Cell** - parameter with most influence using **PCS** simulation.
- Model size - marginal influence using **PCS** simulation.
- **2tLadderQ** performed better or the same when compared to **ladderQ**, **2tHeap**, **fibHeap**, and **heap**.
- **ladderQ** and **2tLadderQ** performance was almost indistinguishable with ${}_{t2}k = 1$.
- **3tHeap** outperformed **2tLadderQ** in certain configurations.

GSA sequential results - PHOLD

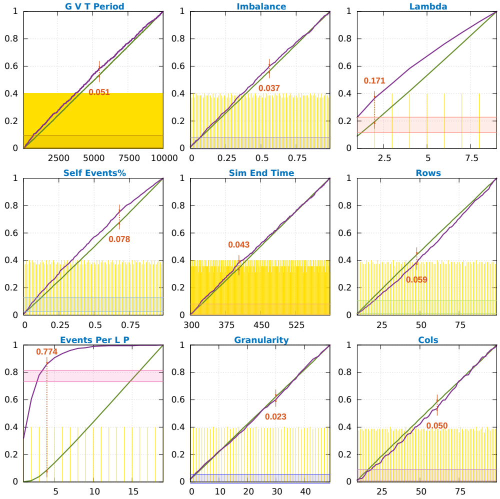


Figure: **Results from GSA comparing 2tLadderQ and 3tHeap using the PHOLD benchmark.**

GSA sequential results - PHOLD

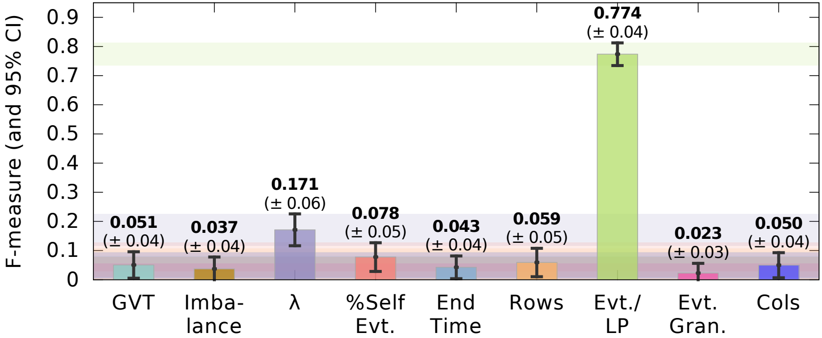


Figure: **GSA data from sequential simulations (1 MPI-process) showing influential PHOLD parameters (2tLadderQ vs. 3tHeap).**

GSA sequential results - PCS

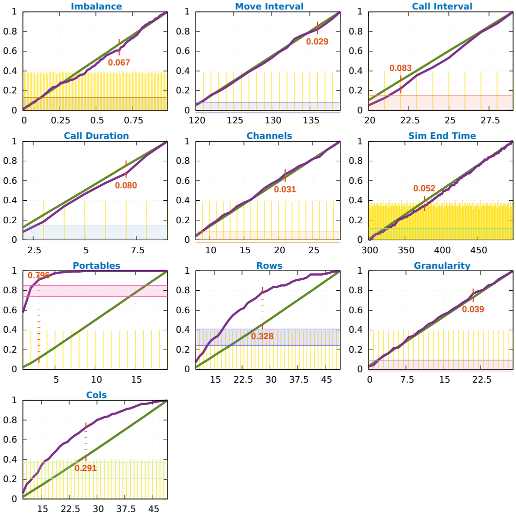


Figure: **Results from GSA comparing 2tLadderQ and 3tHeap for sequential simulation using PCS simulation.**

GSA sequential results - PCS

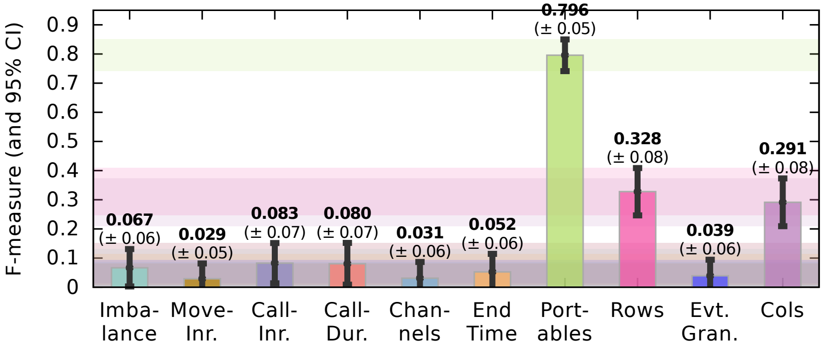


Figure: **GSA data from sequential simulations (1 MPI-process) showing influential PCS parameters (2tLadderQ vs. 3tHeap).**

Summary of GSA parallel results

-GSA parallel simulations focused on **ladderQ**, **2tLadderQ** and **3tHeap**.
-Parallel results are nearly consistent with sequential results.

- **Events per LP** - parameter with most influence using **PHOLD** benchmark.
- **%Self-Events** - parameter has more pronounced influence in comparison with **lambda** using **PHOLD** benchmark.
- **Portables per Cell** - parameter with most influence using **PCS** simulation.
- Model size - marginal influence using **PCS** simulation.

GSA parallel results - PHOLD

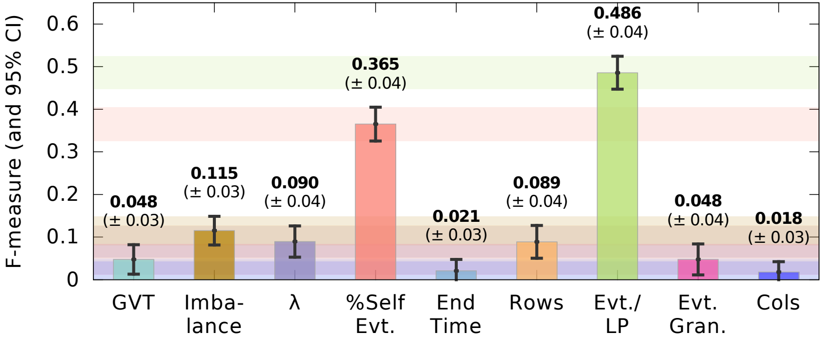


Figure: **GSA data from parallel simulations (4 MPI-processes) showing influential PHOLD parameters (2tLadderQ vs. 3tHeap).**

GSA parallel results - PCS

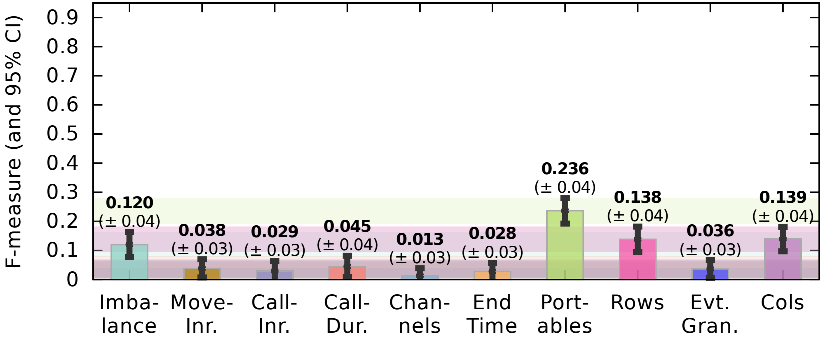


Figure: **GSA data from parallel simulations (4 MPI-processes) showing influential PCS parameters (2tLadderQ vs. 3tHeap).**

Configuration for further analysis

Table: **Configurations of PHOLD and PCS**

Name	#LPs (Rows x Cols)	Sim. End Time	
		Seq	Parallel
ph3	1,000 (100 x 10)	5000	20000
ph4	10,000 (100 x 100)	500	5000
ph5	100,000 (1000 x 100)	100	1000
pcs6	100 (10 x 10)	5000	50000
pcs7	1,000 (100 x 10)	1000	4500
pcs8	10,000 (100 x 100)	100	200

Overview

1. Introduction/Motivation
2. Background
3. Methodology
4. GSA results
5. Sequential simulation results
6. Parallel simulation assessments
7. Conclusion and Future Work

Sequential simulation results - PHOLD

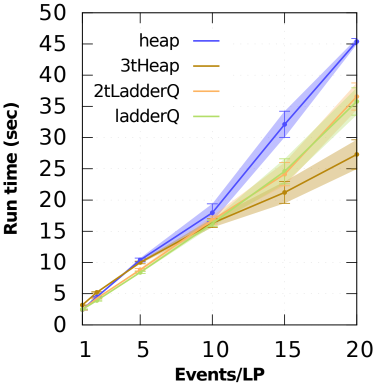


Figure: **Sequential simulation runtimes and peak memory usage with ph3.**

Sequential simulation results - PHOLD

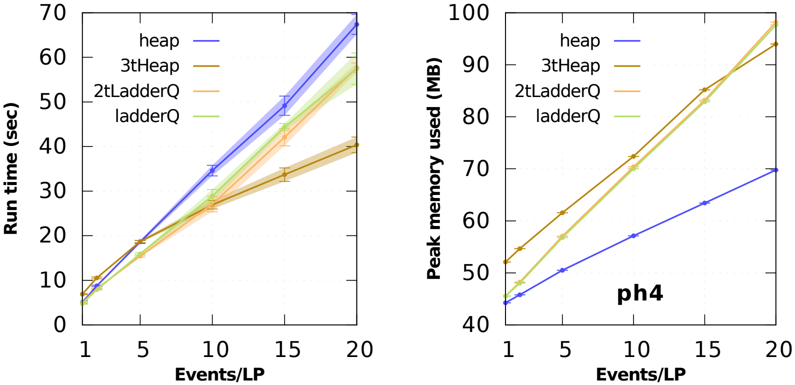


Figure: **Sequential simulation runtimes and peak memory usage with ph4.**

Sequential simulation results - PHOLD

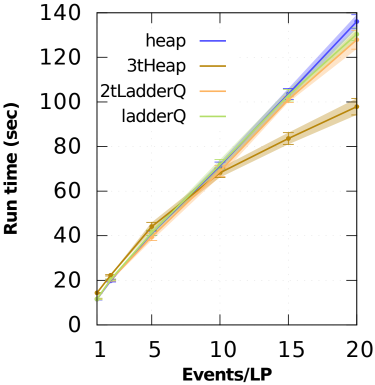


Figure: **Sequential simulation runtimes and peak memory usage with ph5.**

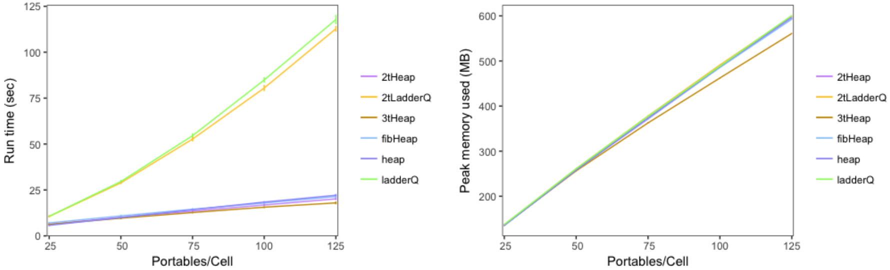


Figure: **Sequential simulation runtimes and peak memory usage with pcs6.**

Sequential simulation results - PCS

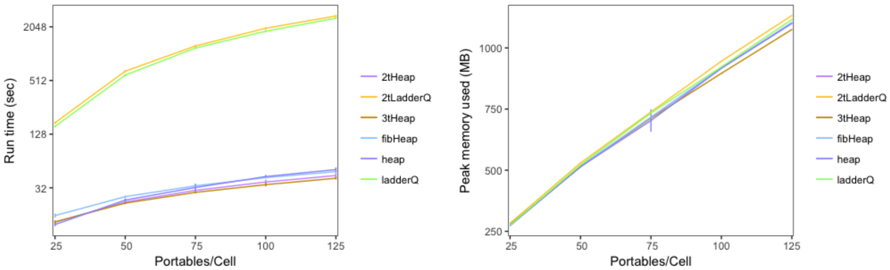


Figure: **Sequential simulation runtimes and peak memory usage with pcs7.**

Sequential simulation results - PCS

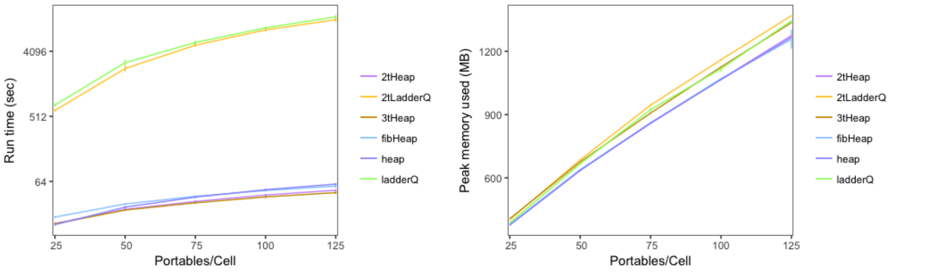


Figure: **Sequential simulation runtimes and peak memory usage with pcs8.**

Overview

1. Introduction/Motivation
2. Background
3. Methodology
4. GSA results
5. Sequential simulation results
6. Parallel simulation assessments
7. Conclusion and Future Work

Parallel simulation assessment - Efficient case for **ladderQ**

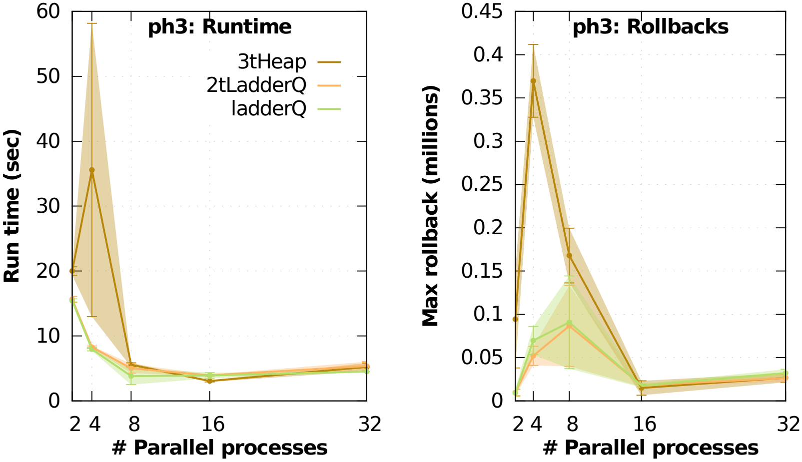


Figure: **Statistics from PH3 configuration of PHOLD parallel simulation with eventsPerLP=2, $\lambda = 1$, %selfEvents=25%**

Parallel simulation assessment - Efficient case for **ladderQ**

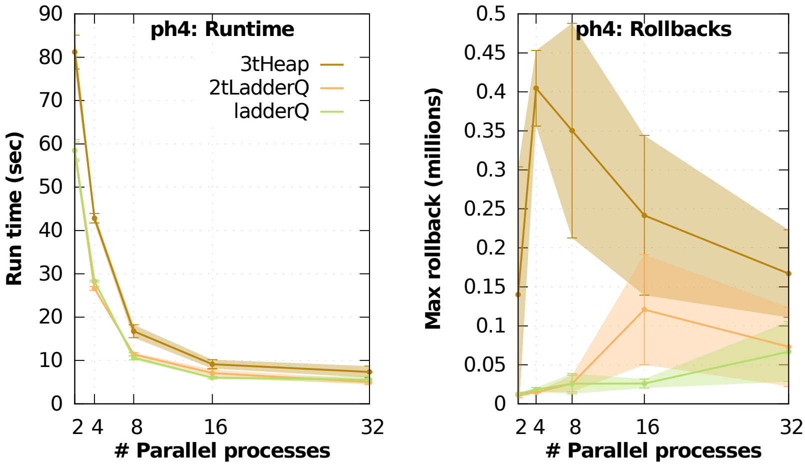


Figure: **Statistics from PH4 configuration of PHOLD parallel simulation with eventsPerLP=2, $\lambda = 1$, %selfEvents=25%**

Parallel simulation assessment - Efficient case for **ladderQ**

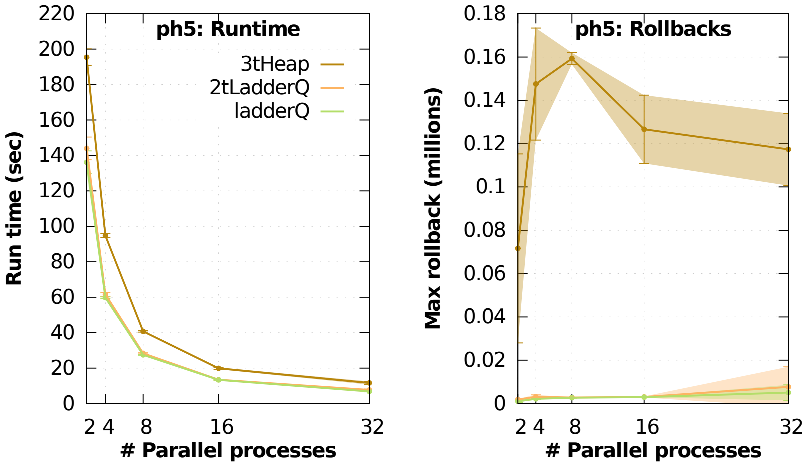


Figure: **Statistics from PH5 configuration of PHOLD parallel simulation with eventsPerLP=2, $\lambda = 1$, %selfEvents=25%**

Parallel simulation assessment - Knee point for **3tHeap** vs. **ladderQ**

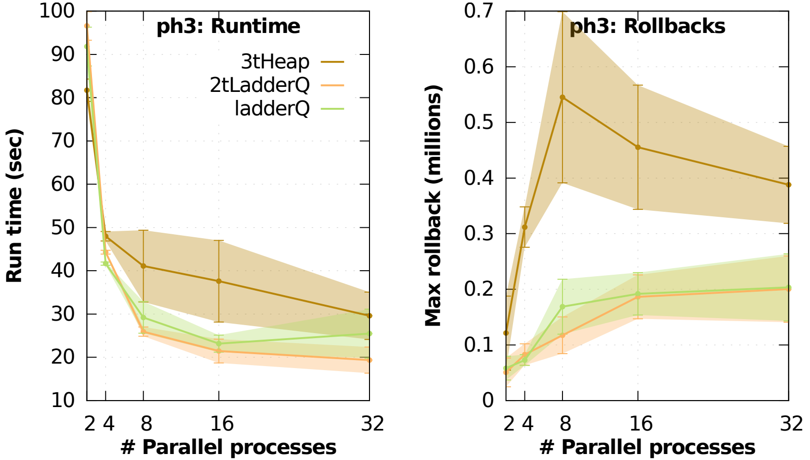


Figure: **Statistics from PH3 configuration of PHOLD parallel simulation with eventsPerLP=10, $\lambda = 10$, %selfEvents=25%**

Parallel simulation assessment - Knee point for **3tHeap** vs. **ladderQ**

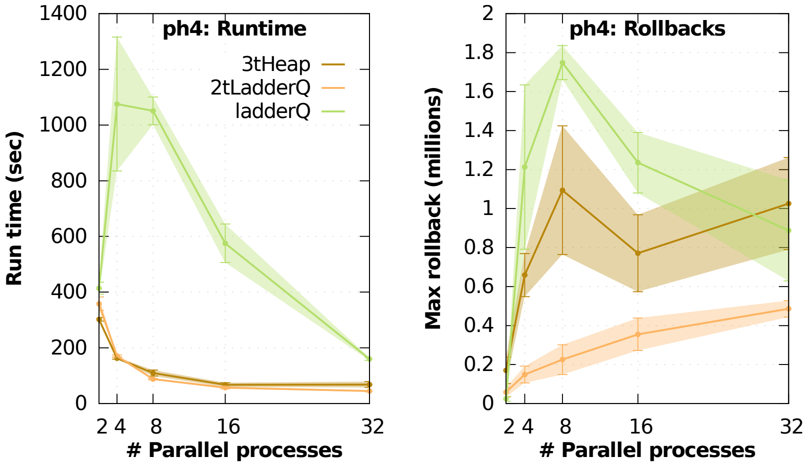


Figure: **Statistics from PH4 configuration of PHOLD parallel simulation with eventsPerLP=10, $\lambda = 10$, %selfEvents=25%**

Parallel simulation assessment - Knee point for **3tHeap** vs. **ladderQ**

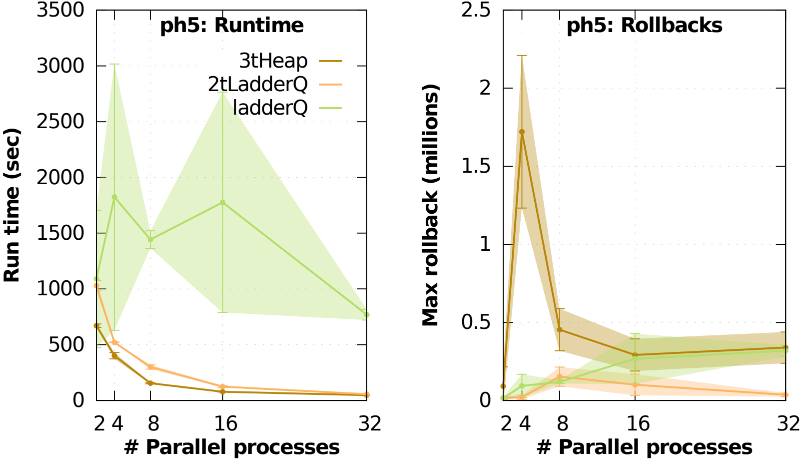


Figure: **Statistics from PH5 configuration of PHOLD parallel simulation with eventsPerLP=10, $\lambda = 10$, %selfEvents=25%**

Parallel simulation assessment - Best case for **3tHeap**

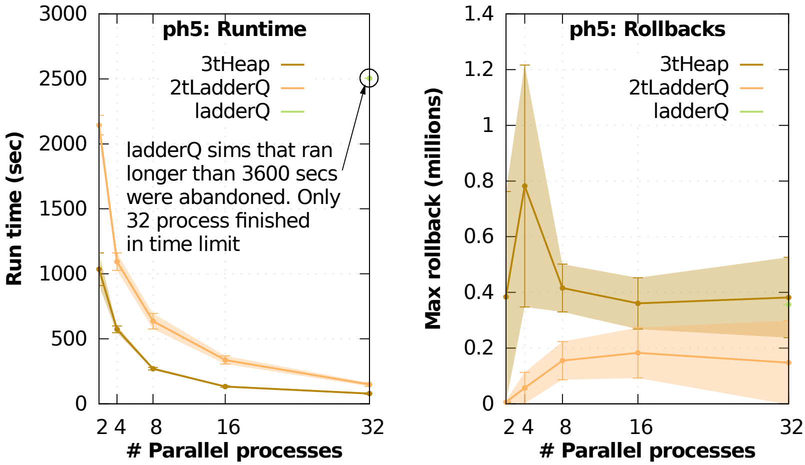


Figure: **Statistics from PH5 configuration of PHOLD parallel simulation with eventsPerLP=20, $\lambda = 10$, %selfEvents=25%**

Parallel simulation assessment - Best case for **3tHeap**

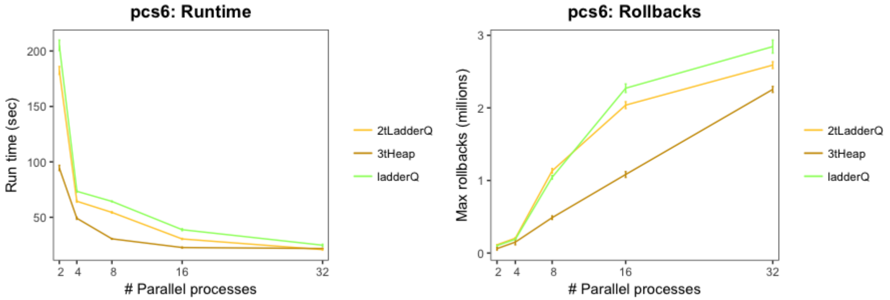


Figure: **Statistics from PCS6 configuration of PCS parallel simulation with portables per cell = 75**

Parallel simulation assessment - Best case for **3tHeap**

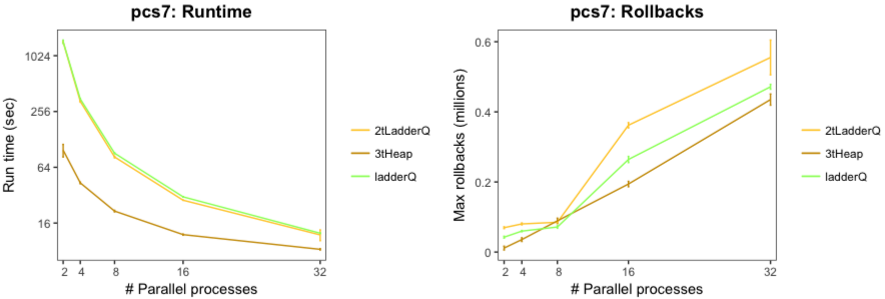


Figure: **Statistics from PCS7 configuration of PCS parallel simulation with portables per cell = 75**

Parallel simulation assessment - Best case for **3tHeap**

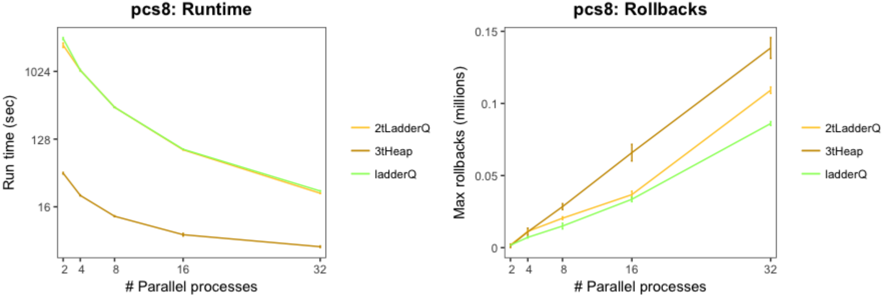


Figure: **Statistics from PCS8 configuration of PCS parallel simulation with portables per cell = 75**

Overview

Conclusions

- Recommend the use of General Sensitivity Analysis (GSA) to reduce the parameter space in simulation models with large parameters.
- **2tLadderQ** performs no worse than **ladderQ** in sequential simulations with ${}_{t2}k=1$.
- Results favor the general use of **2tLadderQ** over the **ladderQ**.
- The advantages of **3tHeap** is realized only when each logical process has 10 or more concurrent events at each time step.
- Multi-tier data structures perform consistently better in optimistic parallel simulations.

Future Work

- Implement our multi-tier data structures in a different language and compare performance.
- Test data structures on other parallel simulation frameworks.
- Assess the performance of our data structures using a wider range of simulation models.
- Assess the effectiveness of multi-tier data structures in multi-threaded simulations.

References

Dickman et al., (2013) Event Pool Structures for PDES on Many-core Beowulf Clusters. *In Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation* ACM, New York, NY, USA, 103-114.

Franceschini et al., (2015) A Comparative Study of Pending Event Set Implementations for PDEVS Simulation. *In Proceedings of the Symposium on Theory of Modeling and Simulation* Society for Computer Simulation International San Diego, CA, USA, 77-84.

Gupta et al., (2014) Lock-free Pending Event Set Management in Time Warp. *In Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation* ACM, New York, NY, USA, 15-26.

Jafer et al., (2013) Synchronization methods in parallel and distributed discrete-event simulation Simulation Modeling Practice and Theory 30(2013), 54-73.

Marotta et al., (2016) A Non-Blocking Priority Queue for the Pending Event Set.In Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques ICST, Brussels, Belgium, 46-55.

Tang et al., (2005) Ladder Queue: An O(1) Priority Queue Structure for Large-scale Discrete Event Simulation.ACM Trans. Model. Comput. Simul. 15, 3(July 2005), 175-204.

The End