# Parallel Main-Memory Indexing for Moving-Object Query and Update Workloads


**Darius Šidlauskas**
Aalborg University
darius@cs.aau.dk

**Simonas Šaltenis**
Aalborg University
simas@cs.aau.dk

**Christian S. Jensen**
Aarhus University
csj@cs.au.dk



## Abstract

This paper presents PGrid, a main-memory spatial index that targets extreme location-related query/update workloads generated by very large populations of moving objects. PGrid implements the proposed freshness semantics that deliver up-to-date query results and enable exploiting the thread-level parallelism offered by multi-core processors.


## Motivation

Today, increased on-chip parallelism is a key mean of improving processor performance. This development calls for software techniques that are capable of scaling linearly with the available hardware threads. Moving-object workloads with queries and massive numbers of updates render it particularly challenging to avoid inter-thread interference and thus achieve scalability.

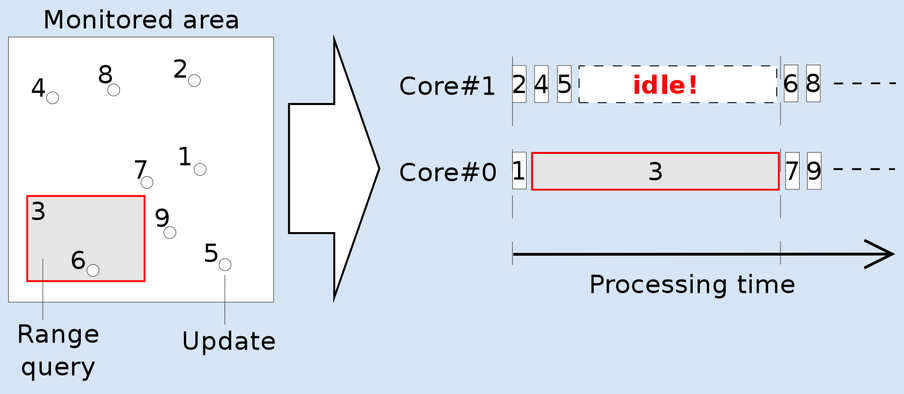
Existing spatial indexing techniques efficiently remove conflicts between concurrent queries and updates by maintaining two separate index structures—one for queries and one for updates [1, 3]. However, such snapshot-based techniques suffer from the following disadvantages:

- Stale query results
- CPU waste on full snapshots
- Stop-the-world problem
- The snapshotting frequency is difficult to set

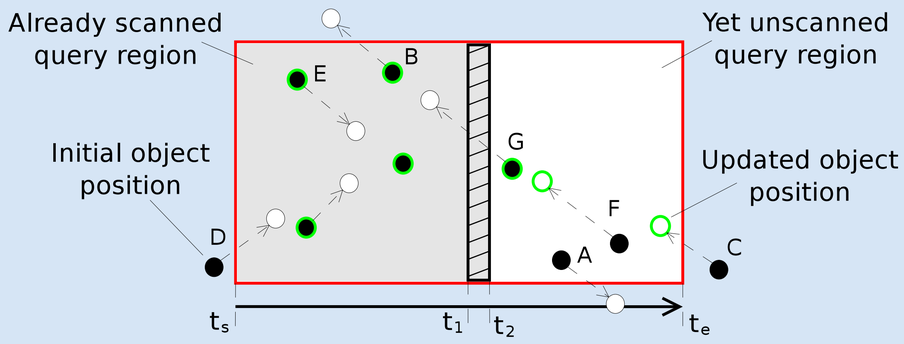
We explore the specifics of application domain and propose a highly parallel index that avoids the mentioned drawbacks.

## Freshness Semantics

Traditional database serializability requires extensive locking and implements timeslice semantics, meaning that a query reports precisely the objects within its range at a specific time instance, usually just after the query processing starts (and the necessary locks are acquired). Consequently, many items are locked for the duration of the query, dramatically slowing down updates.



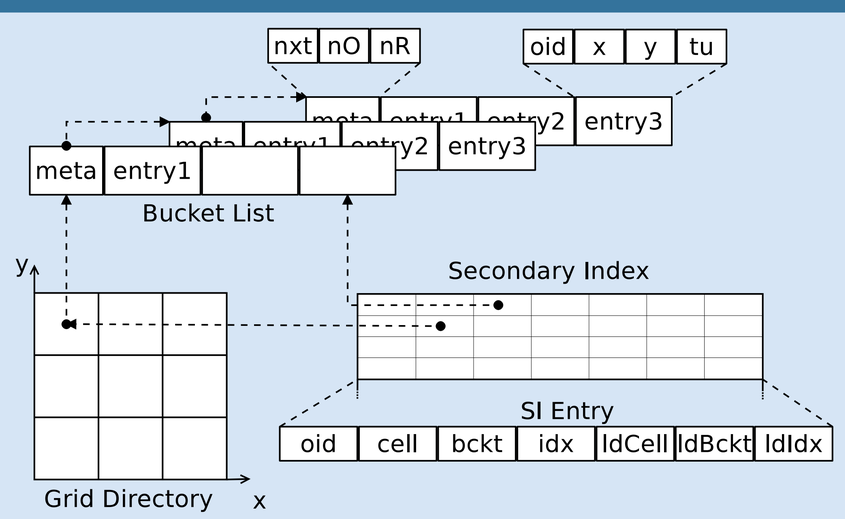

*Freshness* semantics: a query, processed from $t_s$ to $t_e$, returns all objects that have their last reported positions before $t_s$ in the query range, and it reports some (fresher) objects that have their last reported positions after $t_s$ (and before $t_e$) in the query range. In the example, the reported positions are in green (updates occur between $t_1$ and $t_2$).




Key observation: $t_e - t_s < T_o$, where $T_o$ is the time between two consecutive updates of an object.

## PGrid

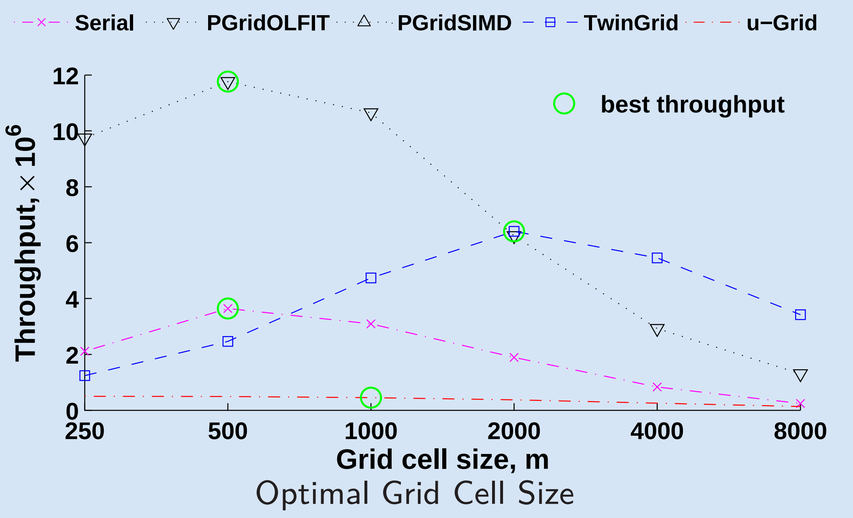
- Implements freshness semantics
- Relies on hardware-assisted atomic operations for concurrency control
- Services both updates and queries using a common data store
- Maintains multiple copies of data at object granularity
- Services queries via a uniform and static grid
- Services updates via a secondary index (bottom-up approach [2])



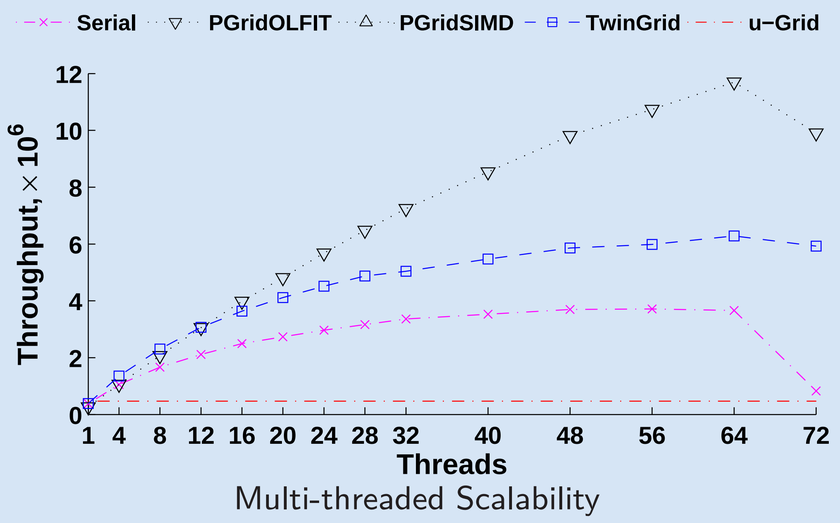

## Empirical Study

- Includes four diverse multi-core platforms
- Studies five grid-based index variants
- Exercises the indexes under massive workloads generated using an open-source moving object trace generator (http://moto.sourceforge.net)

The figures show throughputs obtained on an 8-core SPARC-T2 machine with 64 hardware threads in total. The default workload features 10M moving objects that emit updates every 10 s on average.



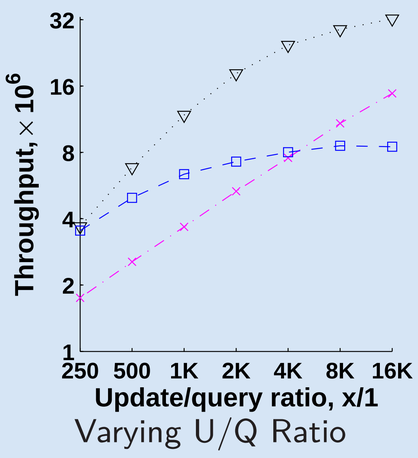

Optimal Grid Cell Size



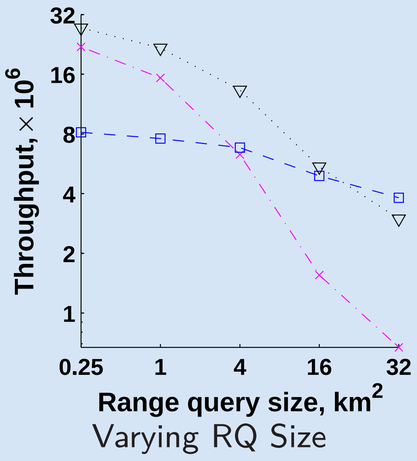

Multi-threaded Scalability

| Method | single-thr'ed | | multi-thr'ed | |
|---|---|---|---|---|
| | read | write | read | write |
| multi-rd/wrt | 6 | 6 | - | - |
| `mutex_t` | 46 | 46 | 358 | 374 |
| `rwlock_t` | 91 | 86 | 716 | 772 |
| `spinlock_t` | 27 | 27 | 98 | 128 |
| 1B latching | 25 | 25 | 108 | 144 |
| OLFIT | 9 | 64 | 251 | 247 |
| SIMD | 4 | 4 | 25 | 21 |

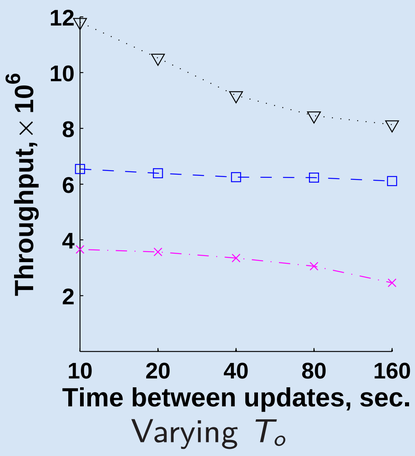
CPU Cycles per 128-bit Read/Write



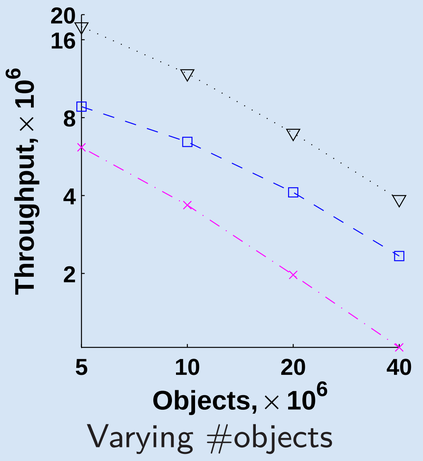

Varying U/Q Ratio




Varying RQ Size




Varying $T_o$




Varying #objects

## Conclusion

PGrid scales near-linearly with #threads and outperforms the alternatives. Key advantages include:

- Fresh query results
- No CPU resources wasted on snapshotting
- No stop-the-world problem
- Treats updates as non-divisible operations
- Eliminates a difficult-to-set tuning parameter—snapshotting frequency


## Acknowledgements

This research was supported by grant 09-064218/FTP from the Danish Council for Independent Research—Technology and Production Sciences. We thank Kenneth A. Ross for his support with the experimental hardware and the anonymous peer reviewers for constructive feedback.


**References:**


[1] J. Dittrich et al. Indexing moving objects using short-lived throwaway indexes. In *SSTD '09*.
[2] M. L. Lee et al. Supporting frequent updates in R-trees: a bottom-up approach. In *VLDB '03*.
[3] D. Šidlauskas et al. Thread-level parallel indexing of update intensive moving-object workloads. In *SSTD '11*.