# A Comparison of the Use of Virtual Versus Physical Snapshots for Supporting Update-Intensive Workloads


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

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

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



## Abstract

This paper studies two fundamentally different approaches to data snapshotting in main memory. Physical, `memcpy`-based and virtual, `fork`-based snapshotting techniques are thoroughly compared in a series of micro-benchmarks. The use of physical snapshots is surprisingly efficient in many cases.

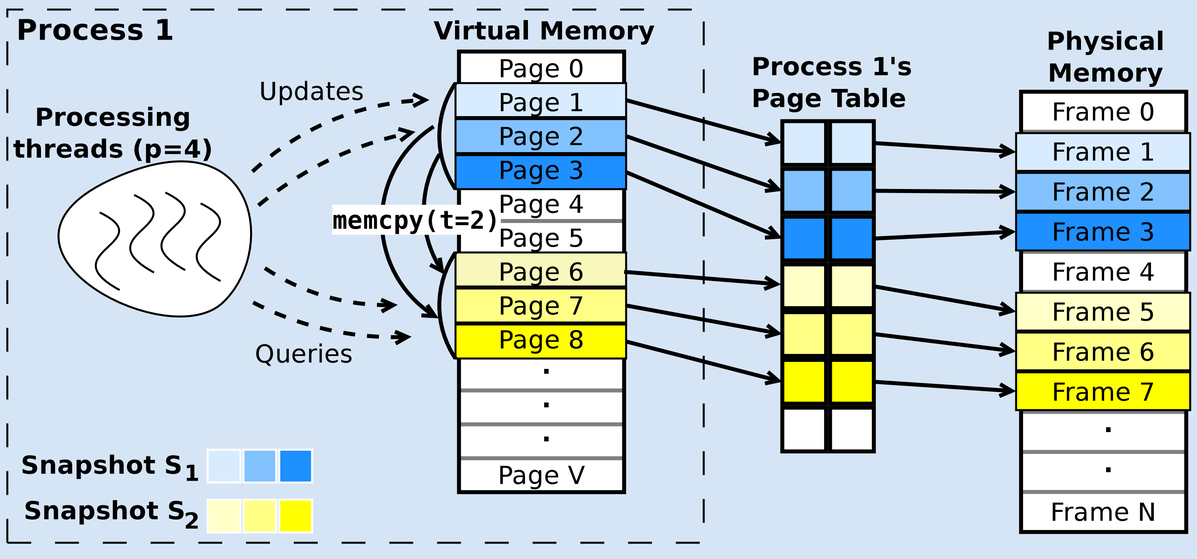

## Motivation

Current processors offer substantial parallel processing capabilities. However, the parallel processing of workloads that intermix queries with frequent updates is non-trivial because programmers face complications of concurrency control, which often causes serialization bottlenecks. Snapshot-based parallel processing is attractive for several reasons:

- Isolates otherwise conflicting operations
- Enables atomic (full) data scans
- Simplifies concurrency control (and its verification)
- Current technologies permit very frequent snapshotting

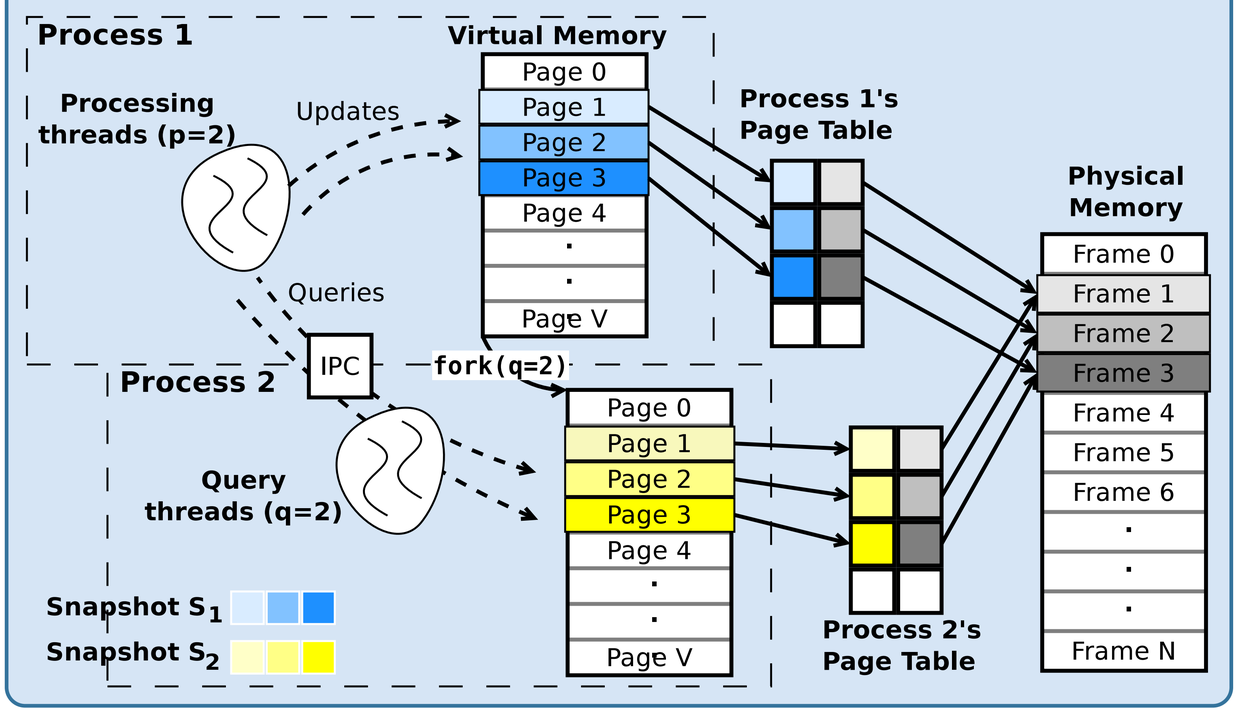
## Physical Snapshotting

- Uses the standard C library `memcpy` function
- Supported by hardware (prefetching, cache bypassing instructions)
- Represents eager copying (a brute-force approach)



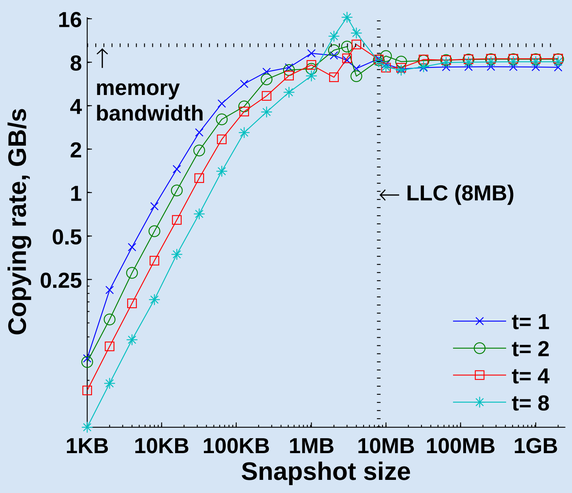

## Virtual Snapshotting

- Uses the `fork` system call
- Supported by HW (MMU, TLB)
- Represents incremental/lazy copying (a copy-on-write approach)

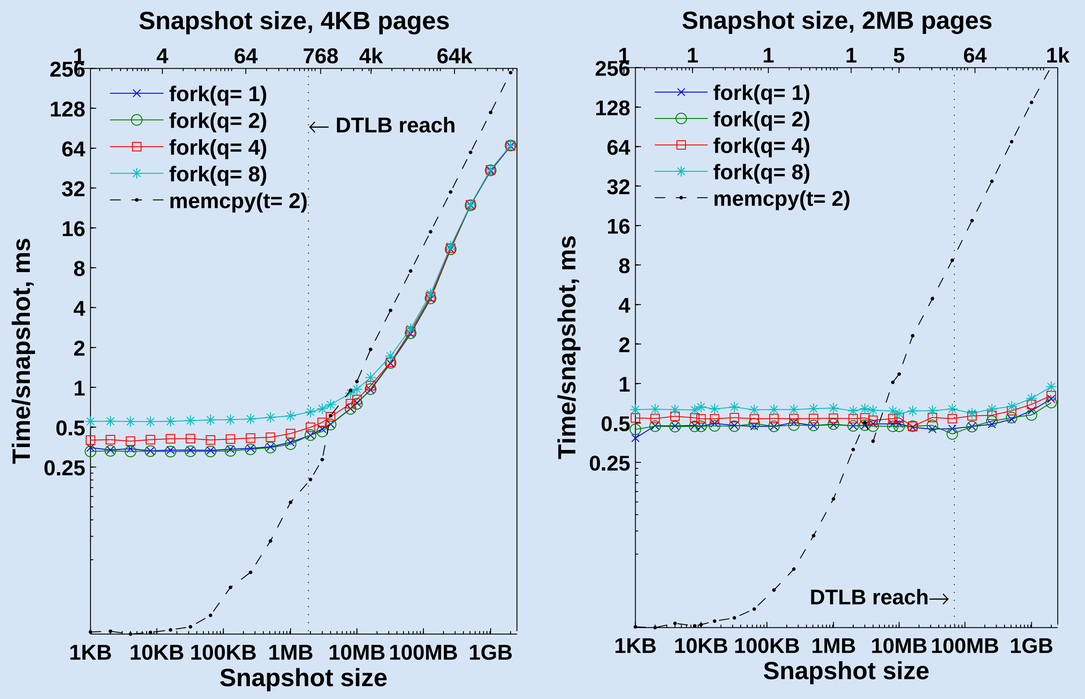



## Empirical Study

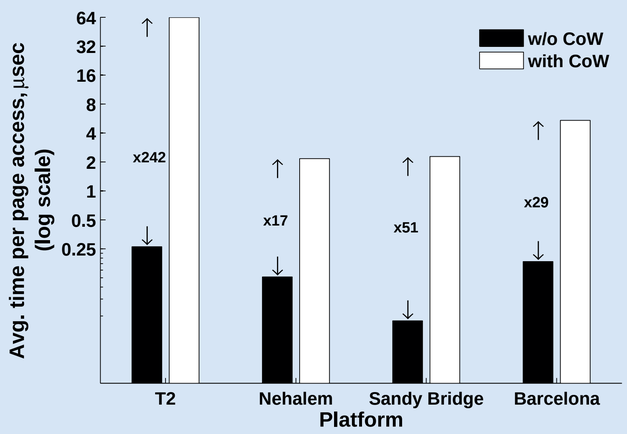
- Includes 4 multi-core platforms:
  - Lean-camp and fat-camp CMP designs
  - Single- and multi-socket configurations
  - Different Unix flavors
- The figures show results obtained on a quad-core Intel Core i7-2600K (Sandy Bridge) with 2 thread contexts per core.
- Findings include:
  - The memory bandwidth utilization in `memcpy`ing is 76–88%.
  - Physical snapshot creation is very competitive (max snapshot size considered is 2GB).
  - In updating, only under very skewed update distributions is virtual snapshotting preferable.
  - The use of huge pages improves virtual snapshotting performance significantly.



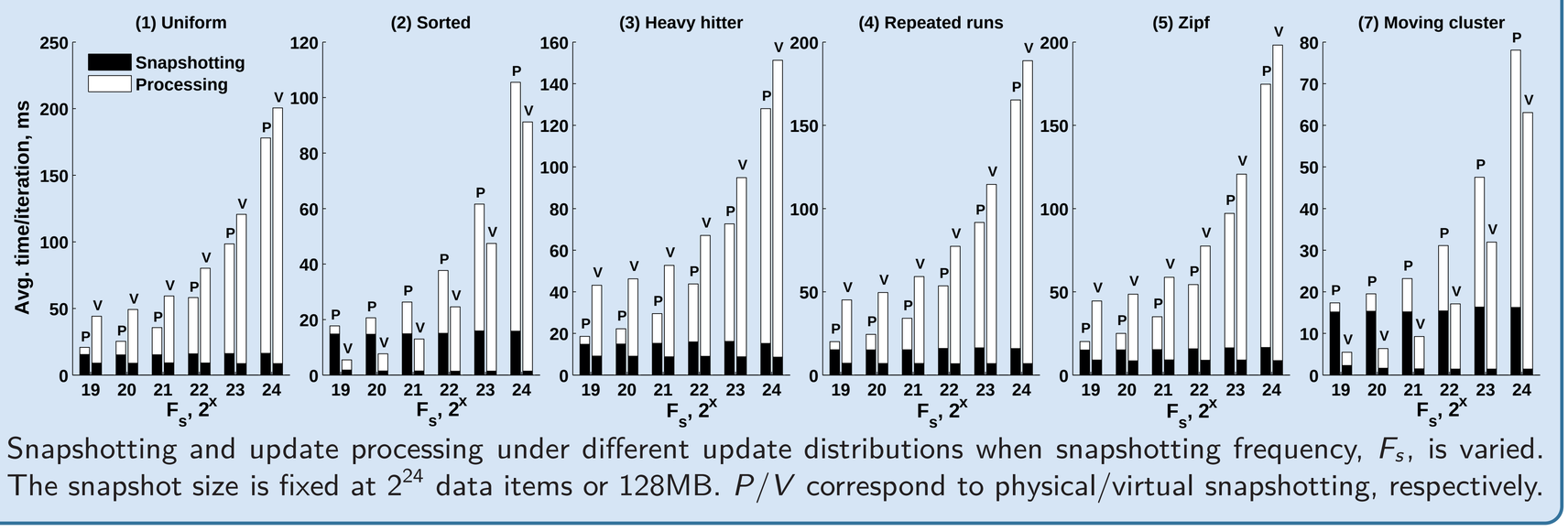

Physical snapshotting with varying number $t$ of threads executing the `memcpy`.




Virtual snapshotting; $q$ is the number of query threads in the `fork`ed process.




Copy-on-write cost on different platforms.




Snapshotting and update processing under different update distributions when snapshotting frequency, $F_s$, is varied. The snapshot size is fixed at $2^{24}$ data items or 128MB. $P/V$ correspond to physical/virtual snapshotting, respectively.

## Conclusion

For most of the considered workloads, the best overall update performance is achieved using physical snapshotting, including the workloads with snapshot sizes an order of magnitude larger than the LLC.

| | Ease of implementation | | Cross-platform | Small snapshots | Huge pages | Update skew (distribution nr.) | | | | | | | Memory footprint |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Linked struct. | queries | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| V | + | − | − | − | + | − | + | − | − | − | − | + | + |
| P | − | + | + | + | − | + | − | + | + | + | + | − | − |


## 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 (supported by NSF grant IIS-1049898) for providing access to the experimental hardware.