Hybrid Skiplist: Combining the Best of Near-Data-Processing and Lock-Free Algorithms

Jiwon Choe
CS Dept., Brown University
jiwon_choe@brown.edu

Tali Moreshet
ECE Dept., Boston University
talim@bu.edu

Maurice Herlihy
CS Dept., Brown University
mph@cs.brown.edu

R. Iris Bahar
CS Dept., Brown University
iris_bahar@brown.edu

ABSTRACT

While there have been efforts to build high-performance data structures with near-data-processing (NDP), prior designs have mostly failed to consider the data access patterns and impacts of cache. We propose the *hybrid skiplist*, a concurrent skiplist algorithm that takes advantage of both the cache-friendly nature of lock-free skiplists and low-latency memory access of NDP. We also propose the *hybrid biased skiplist*, where frequently-accessed nodes are dynamically promoted to higher levels of the skiplist for better performance.

1 BACKGROUND & MOTIVATION

Near-data-processing (NDP) empowered by 3D die-stacking technology has recently re-emerged as a promising way around the memory wall problem. As shown in Figure 1, in generic NDP architectures, the memory is divided into vertical sections, called *NDP vaults*, and each NDP vault is tightly coupled with a compute unit, called the *NDP core*. Data-intensive parts of computation can be offloaded to these NDP cores, where the physical proximity to coupled NDP vaults allows for low-latency memory access.

Among the prior work on improving general-purpose concurrent data structures with NDP [3, 7–9, 11], Liu *et al.* [9] particularly proposed NDP-based data structures that apply *flat-combining* techniques [4] in the NDP cores to take advantage of low-latency memory access without sacrificing concurrency.

We focus on one useful concurrent data structure, the skiplist, which is a popular data structure for key-value stores or database indexes. The skiplist [10] is an ordered, pointer-chasing data structure with multiple levels of pointers at each node. Each node's height is randomly assigned according to a particular distribution, and the skiplist probabilistically achieves logarithmic time operation execution.

The NDP-based implementation [9] partitions the skiplist across multiple NDP vaults (Figure 2). Each NDP core *flat-combines* the operations made on its coupled partition to handle concurrent operations from the host threads. Because of the skiplist's pointer-chasing nature, it was expected to benefit greatly from the NDP-based design.

However, our empirical evaluations of the NDP-based data structures on a cycle-accurate full-system architecture simulator [3] revealed that the skiplist must be significantly larger than cache for the NDP-based implementation to outperform the non-NDP state-of-the-art lock-free skiplist [5]. Because of the skiplist's hierarchical structure, every operation begins by traversing through the few

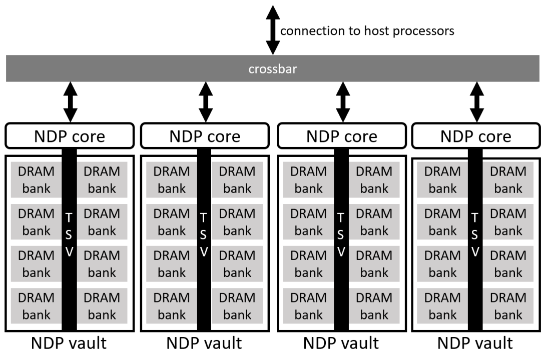


Figure 1: Generic near-data-processing architecture.

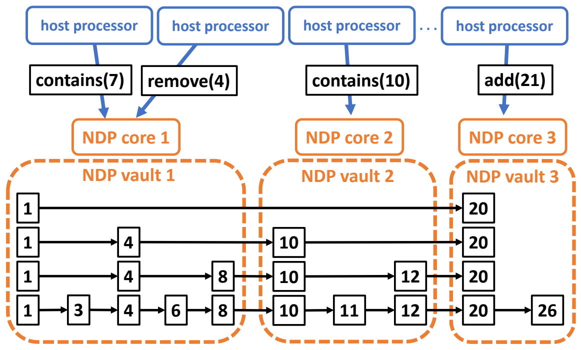


Figure 2: NDP-based skiplist [9].

higher-level nodes, which are then accessed repeatedly over the course of many operations. In the non-NDP context, these nodes are likely to remain in cache, causing only a few node accesses to actually go out to memory.

2 HYBRID SKIPLIST

For a data structure to be useful, its implementation must take into account the interplay of data access patterns and effects of the underlying architecture, including cache. We are the first to propose such an algorithm – the *hybrid skiplist*, which combines the best of NDP-based and lock-free skiplists. Figure 3 depicts the hybrid skiplist design. The frequently-traversed higher levels of the skiplist are maintained as a lock-free skiplist, taking advantage of the host cache. The remaining lower levels are implemented as an NDP-based skiplist to exploit the benefits of near-data-processing. The hybrid skiplist is *linearizable* [6], with the linearization point of add and remove operations always in the NDP-based portion.

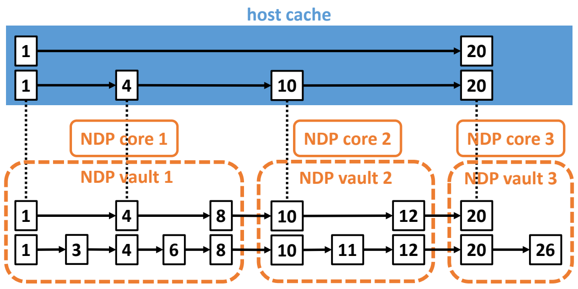


Figure 3: Hybrid skiplist.

Table 1: Evaluation framework details.

Host Configuration	
processor	8 in-order processors (ARMv7 Cortex-A15)
L1 cache	32kB icache, 64kB dcache, private, 2-way set-associative 0.8 ns dcache access latency, 256B/block
L2 cache	2MB, shared, 8-way set associative 1.8ns access latency, 256B/block
main memory	1GB

NDP Configuration	
NDP core	1 in-order processor/vault (ARMv7 Cortex-A15)
scratchpad memory	40kB/NDP core, stores instructions and program stack 8 4-byte static registers, 8kB vector register reserved for memory-map
NDP vault	8 NDP vaults, 128MB/vault, open-page row-buffer-management policy

The system's memory is divided into host-accessible main memory and NDP-capable memory (*i.e.*, NDP vaults) that is accessed only by the NDP cores. The NDP cores are simple processors without cache, but each NDP core is equipped with a small scratchpad memory that stores the NDP core's program data and stack. Part of the scratchpad is also memory-mapped into host memory and used for communication between the host processors and NDP core.

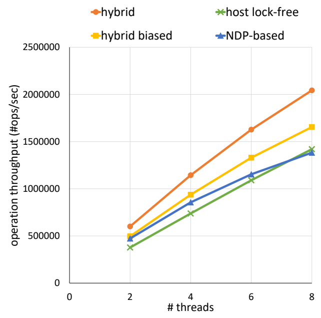
The lock-free portion of the hybrid skiplist is primarily stored in host-accessible main memory. The number of levels in the lock-free portion is determined by a software parameter that is set based on the cache and total skiplist size; this "pins" the high-level nodes in cache. Implementing the lower levels as the NDP-based skiplist – accessed only by NDP cores – prevents the frequently-traversed high-level nodes in cache from being replaced by low-level nodes that are unlikely to be accessed again soon.

We also extend the hybrid skiplist to the *hybrid biased skiplist*[1], which is designed to yield higher performance in more realistic settings, where there is a skew in node accesses. In the hybrid biased skiplist, popular nodes in the NDP-based portion are promoted one level at a time, until reaching the lock-free portion. After a node is promoted, the predecessor node at the promoted level is demoted, so that the skiplist remains balanced. Popularity is determined by a short log of operation timestamps on each node.

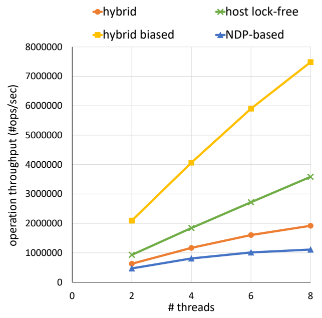
3 EVALUATION & DISCUSSION

We evaluate the proposed hybrid and hybrid biased skiplists on Brown-SMCSim, a gem5 [2] cycle-accurate, full-system NDP architecture simulator used for evaluations in [3]. Table 1 summarizes the simulator details.

Figure 4 compares the operation throughput of the hybrid, hybrid biased, host lock-free, and NDP-based skiplists. The initial skiplist size was set to 0.5GB (2^{22} nodes), and operations were divided as 90% contains, 9% add, and 1% remove. For Figure 4a, the operations were uniformly distributed across all keys; for Figure 4b, the operations had a zipfian distribution with 100 hot keys.



(a) uniform distribution



(b) zipfian distribution (100 hot keys, skew=0.99)

Figure 4: Operation throughput of various skiplist implementations.

In either case, the hybrid biased skiplist outperforms the non-NDP state-of-the-art lock-free skiplist (17% and 2.1x throughput increase with uniform and zipfian distributions, respectively). With a uniform distribution, the hybrid biased skiplist has lower throughput than the hybrid skiplist. This is due to the overhead associated with determining node popularity, but note that the zipfian distribution reflects a more realistic skiplist usage.

[1]named after Bagchi *et al.*'s *biased skiplist* [1]

REFERENCES

[1] Amitabha Bagchi, Adam L Buchsbaum, and Michael T Goodrich. 2005. Biased skip lists. *Algorithmica* 42, 1 (2005), 31–48.

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[3] Jiwon Choe, Amy Huang, Tali Moreshet, Maurice Herlihy, and R Iris Bahar. 2019. Concurrent Data Structures with Near-Data-Processing: an Architecture-Aware Implementation. In *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '19)*. ACM, New York, NY, USA. https://doi.org/10.1145/3323165.3323191

[4] Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir. 2010. Flat Combining and the Synchronization-parallelism Tradeoff. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '10)*. ACM, New York, NY, USA, 355–364. https://doi.org/10.1145/1810479.1810540

[5] Maurice Herlihy, Yossi Lev, Victor Luchangco, and Nir Shavit. 2006. A provably correct scalable concurrent skip list. In *Conference On Principles of Distributed Systems (OPODIS)*. Citeseer.

[6] Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (July 1990), 463–492. https://doi.org/10.1145/78969.78972

[7] Byungchul Hong, Gwangsun Kim, Jung Ho Ahn, Yongkee Kwon, Hongsik Kim, and John Kim. 2016. Accelerating Linked-list Traversal Through Near-Data Processing. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT '16)*. ACM, New York, NY, USA, 113–124. https://doi.org/10.1145/2967938.2967958

[8] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. 2016. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 25–32.

[9] Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu. 2017. Concurrent Data Structures for Near-Memory Computing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '17)*. ACM, New York, NY, USA, 235–245. https://doi.org/10.1145/3087556.3087582

[10] William Pugh. 1990. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Commun. ACM* 33, 6 (June 1990), 668–676. https://doi.org/10.1145/78973.78977

[11] Paulo C Santos, Geraldo F Oliveira, João P Lima, Marco AZ Alves, Luigi Carro, and Antonio CS Beck. 2018. Processing in 3D memories to speed up operations on complex data structures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 897–900.