*ng*AP: Non-blocking Large-scale Automata Processing on GPUs

Tianao Ge[1], Tong Zhang[2], Hongyuan Liu[1]

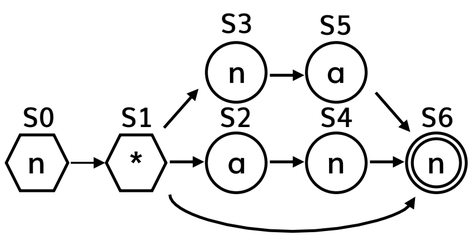
[1]Hong Kong University of Science and Technology (Guangzhou),
[2]Samsung Electronics

ASPLOS 2024

Background

Finite automata serve as compute kernels for various applications that require high throughput.

The structure of automata

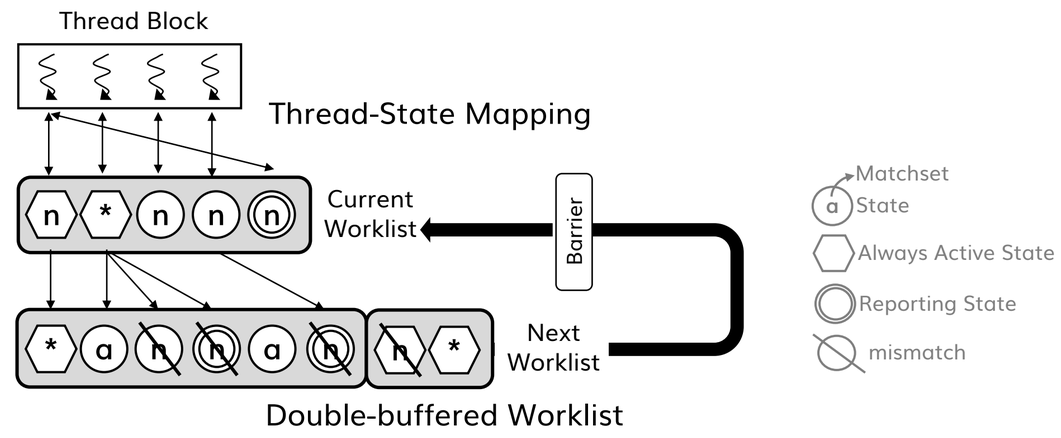


An example of matching process

Iter	Sym	Active states	Matched States	Report
0	b	S0, S1	S1	
1	a	S0, S1, S2, S3, S6	S1, S2	
2	n	S0, S1, S2, S3, S4, S6	S0, S1, S3, S4, S6	S6
3	a	S0, S1, S2, S3, S5, S6	S1, S2, S5	
4	n	S0, S1, S2, S3, S4, S6	S0, S1, S3, S4, S6	S6
5	a	S0, S1, S2, S3, S5, S6	S1, S2, S5	

GPU provides high throughput, making it an attractive option for processing large-scale automata applications.

Blocking Automata Processing on GPUs



Previous GPU-based approaches utilized a data-driven approach to enhance GPU utilization, incorporating:

1. A **double-buffered worklist** to store active states.
2. **Dynamic mapping** between threads and states.

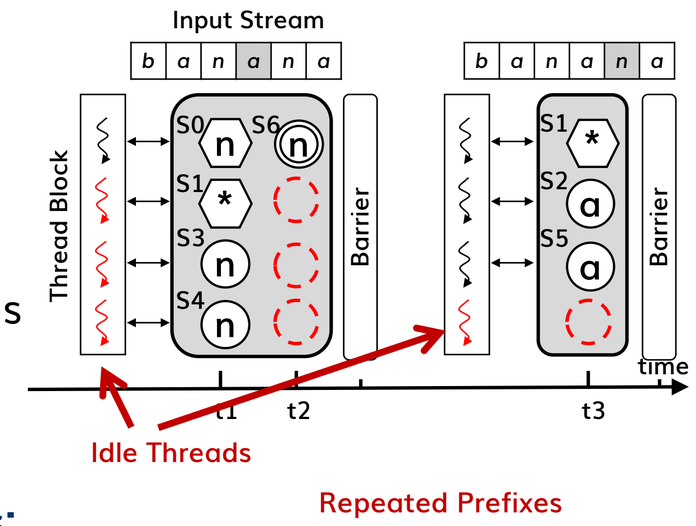
However, each state had to wait at a barrier until all states in the worklist were processed, resulting in a "**blocking**" **process.**

Motivation

Our detailed characterization reveals that **three challenges** have not been systematically addressed:

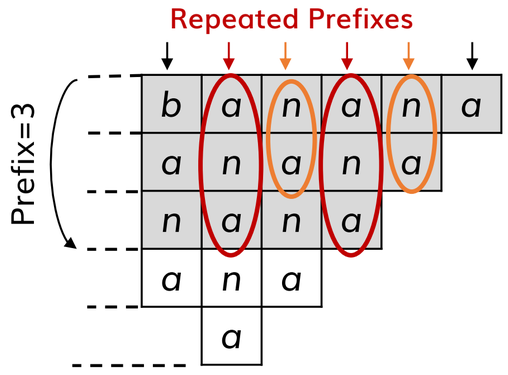
Threads Underutilization:

- Threads only work on states matched **in one iteration.**
- Utilization only achieves average **32.6%.**



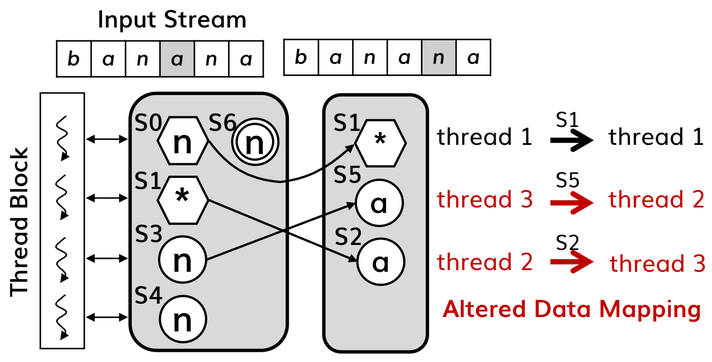
Redundant Computations:

- The always-active states match with some symbols repeatedly in every iteration.
- **93%** work can be reduced if eliminating the redundant matches.



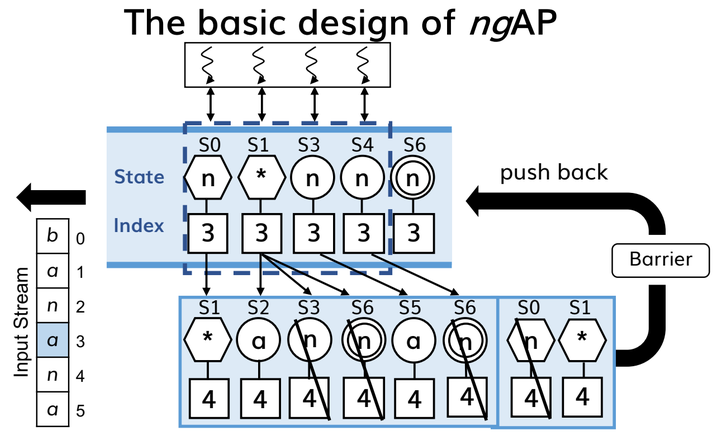
Poor Data Locality:

- Threads and states mapping **switches** frequently.
- L1 cache hit rate only achieves **23%.**



*ng*AP

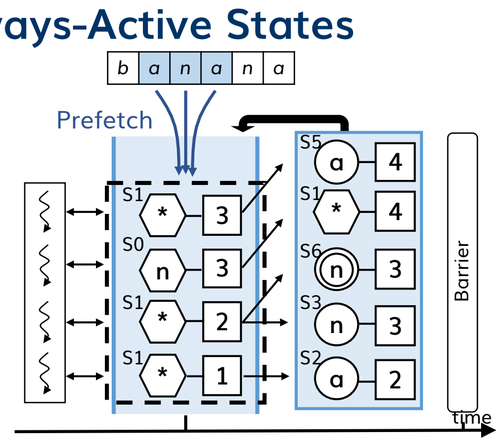
The basic design of *ng*AP



*ng*AP uses (1) **State-Index Pairs** (2) **Sliding Window,** to allow parallel processing of different symbols in the input stream and enable further **3 optimizations.**

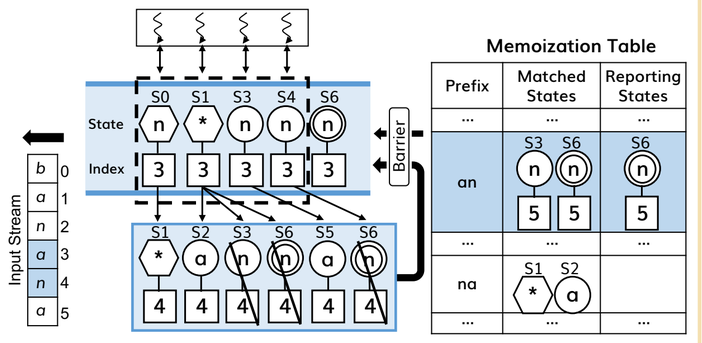
Optimization #1: Prefetching Always-Active States

✓ By prefetching the matches between always-active states and symbols to the worklist, the number of indices coexisting in the worklist increases.



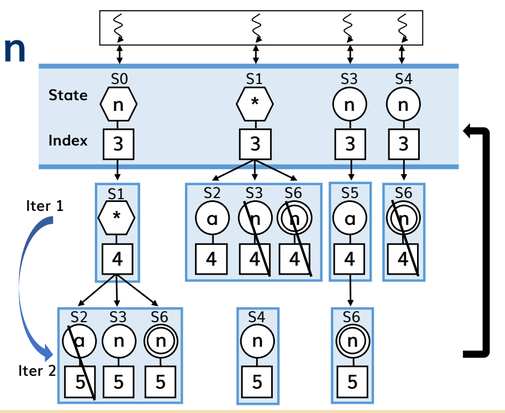
Optimization #2: Prefix Memoization

✓ Prefix Memoization substitutes matches between always-active states and pattern prefixes with table look-ups.



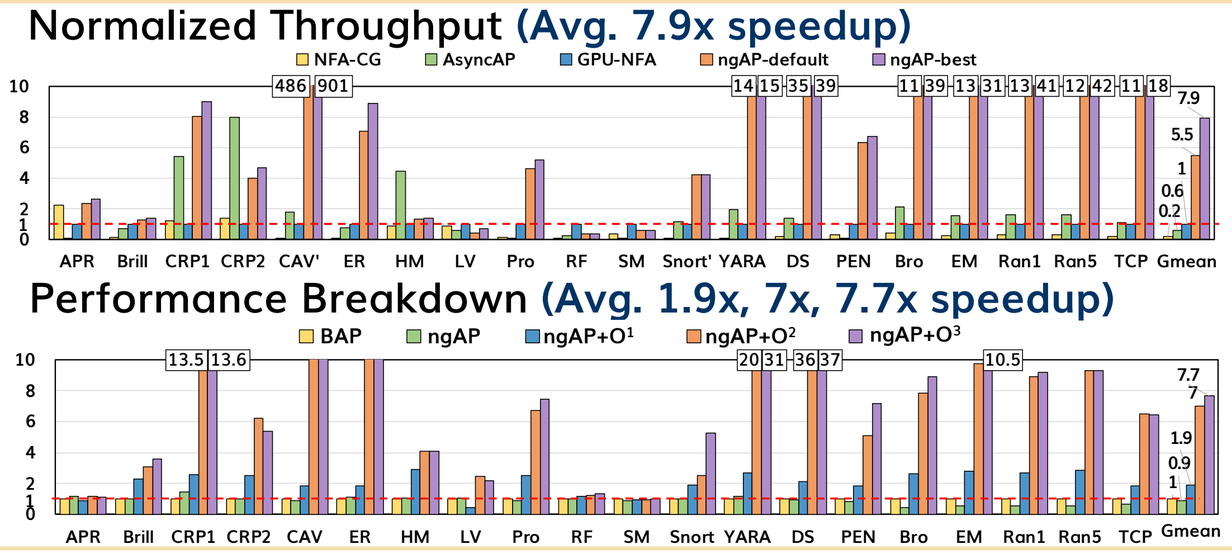
Optimization #3: Work Privatization

✓ Each thread can decide whether it privatizes the extended neighbors without writing them back to the shared worklist.



Results

Normalized Throughput **(Avg. 7.9x speedup)**



Performance Breakdown **(Avg. 1.9x, 7x, 7.7x speedup)**

Conclusion

- We present Non-blocking Automata Processing (***ng*AP**) which allows multiple symbols to be processed on GPUs.
- On top of *ng*AP, we propose **3 optimizations** to address identified challenges.
- Evaluation shows *ng*AP outperforms SOTA by an average of **7.9×** and up to **901×** across 20 applications.

***ng*AP GitHub Repository**
https://github.com/getianao/ngAP

29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, San Diego, USA