Abstract

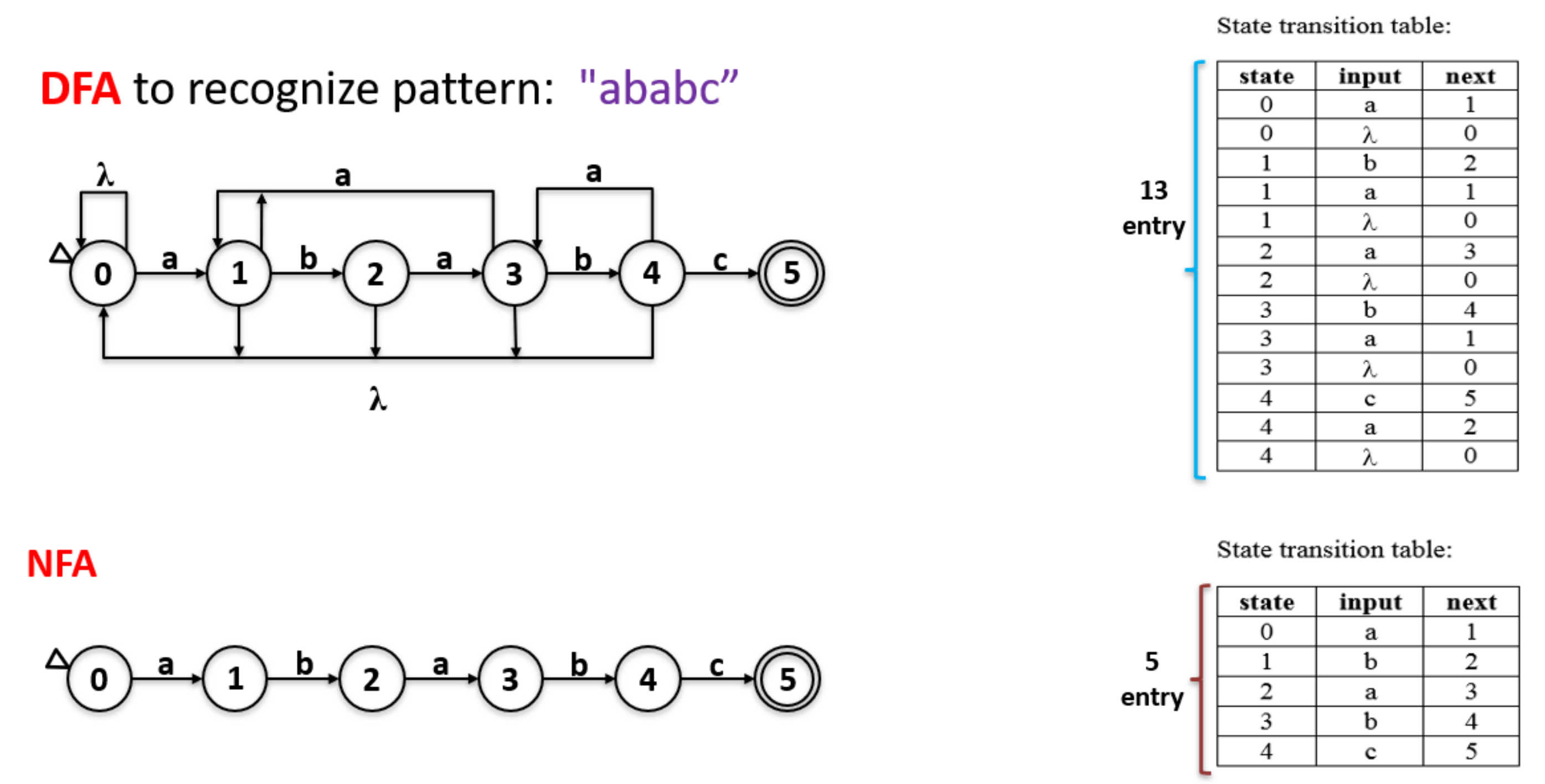
Deterministic and Non-deterministic Finite Automata (DFA and NFA) comprise the fundamental unit of work for many emerging big data applications, motivating recent efforts to develop Domain-Specific Architectures (DSAs) to exploit fine-grain parallelism available in automata workloads.
Our research involves conceptual and applied components. The applied part involves design a reusable customized automata processor overlay on a Field-Programmable Gate Array (FPGA)-based platform.
This design can maximally exploit on-chip memory parallelism for NFA evaluation. The conceptual part is the application of novel hardware allocation algorithms to a complex space of design trade-offs in both the computational and communication aspects of the design.

Our research continues to adapt the practical components of Automata Processor design to operate in the cloud, in addition to develop techniques to improve the programmable routing of the design and reduce the output latency.

Introduction

Automata (NFA, DFA) is a directed graph of states and edges. NFA demands high memory bandwidth as a result of multiple memory access at a time. DFA requires expensive memory as a result of large transition table as shown in Fig.1.

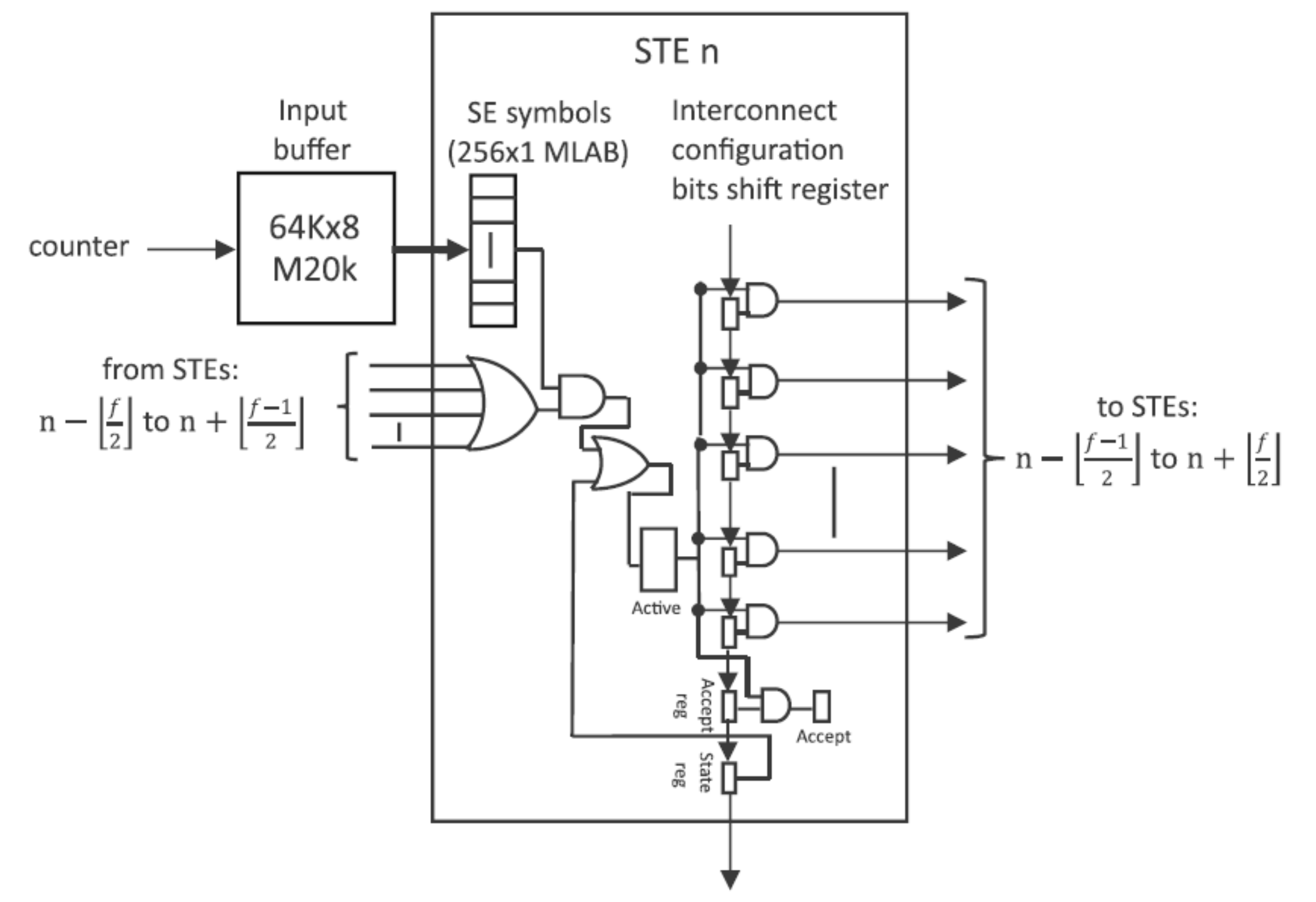
Fig. 1 NFA and DFA implementation



NAPOLY (Non-deterministic Automata OverLaY) is an automata architecture designed on top of FPGA (a DSA architecture).

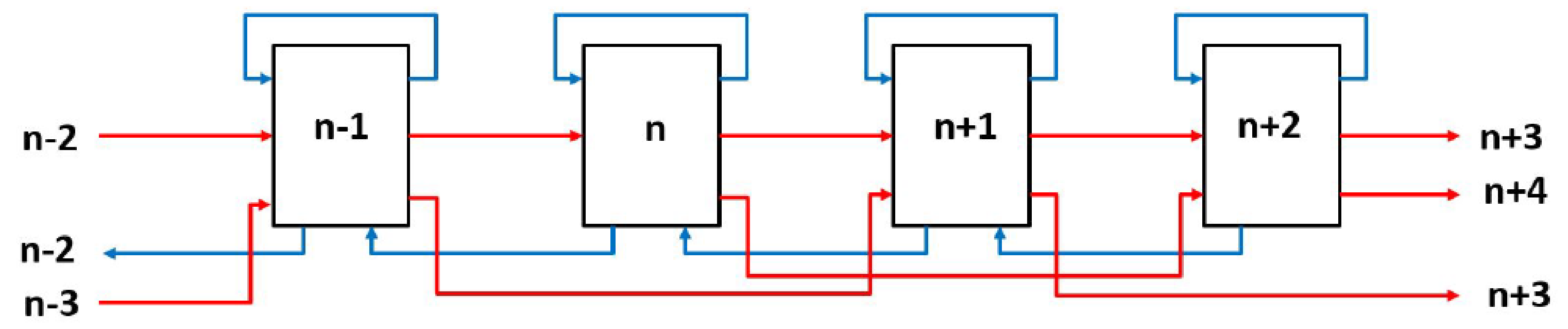
The transition tables are stored in SEs, which comprises of a static memory, gate and flip-flop as shown in Fig.

Fig. 2 NAPOLY State Element



STEs connected to each other using point-point routing as shown in Fig.

Fig. 3 NAPOLY Interconnections



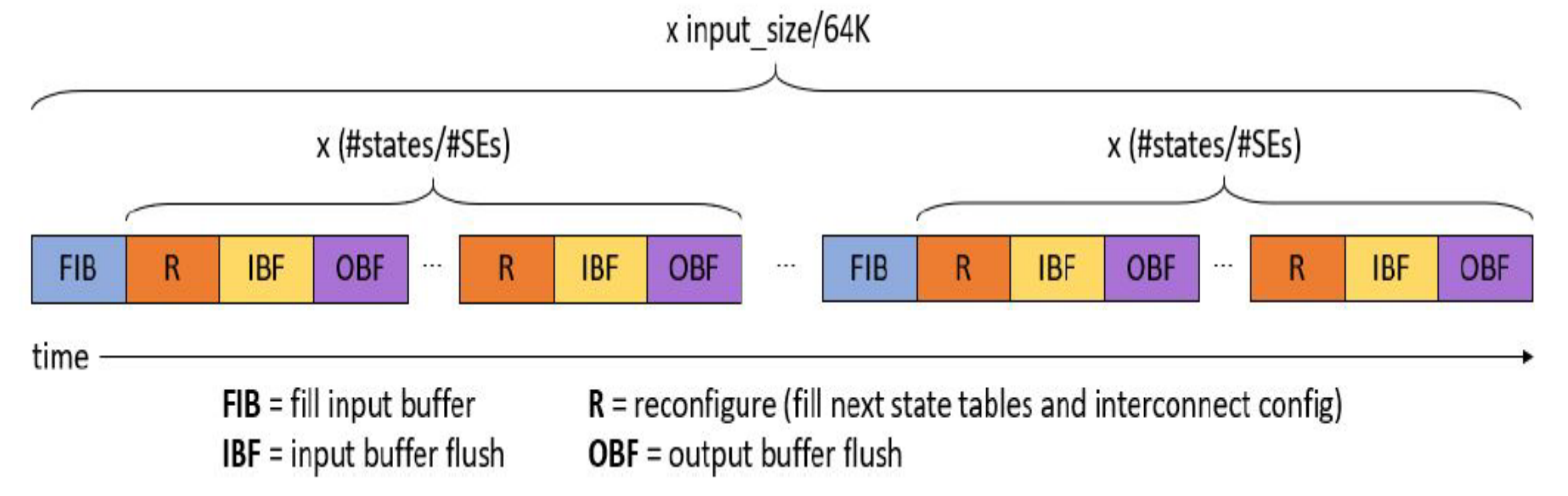
NAPOLY is connected to 64KB DRAM to read input symbols and 64KB DRAM to report output. Fig. shows NAPOLY time diagram

Comparing NAPOLY with existing CPU and GPU implementations, NAPOLY outperforms among 10 out of 12 of Benchmarks as shown in Table 1.

Table 1. NAPOLY Comparing with GPU and CPU Implementations

Benchmark	NAPOLY Throughput	Average Active States (AS)	GPU Throughput (MB/s)	CPU Throughput (MB/s)	Speedup vs Max (GPU,CPU)
Brill	36	14	7	1	9
ClamAv	16	4	4	14	1.14
DotStar	12	3	40	10	0.3
Entity Resolution	7	10	4	1	1.75
Fermi	31	3854	2	1	15.5
Hamming	63	240	18	10	3.5
Levenshtein	63	88	38	1	1.65
Power En	31	31	53	10	0.58
Protomata	24	19	5	1	4.8
Random Forest	16	968	2	0.5	8
Snort	15	98	14	0.4	1.07
SPM	14	6631	0.5	0.1	28

Fig. 4 NAPOLY Timing Diagram



Objectives

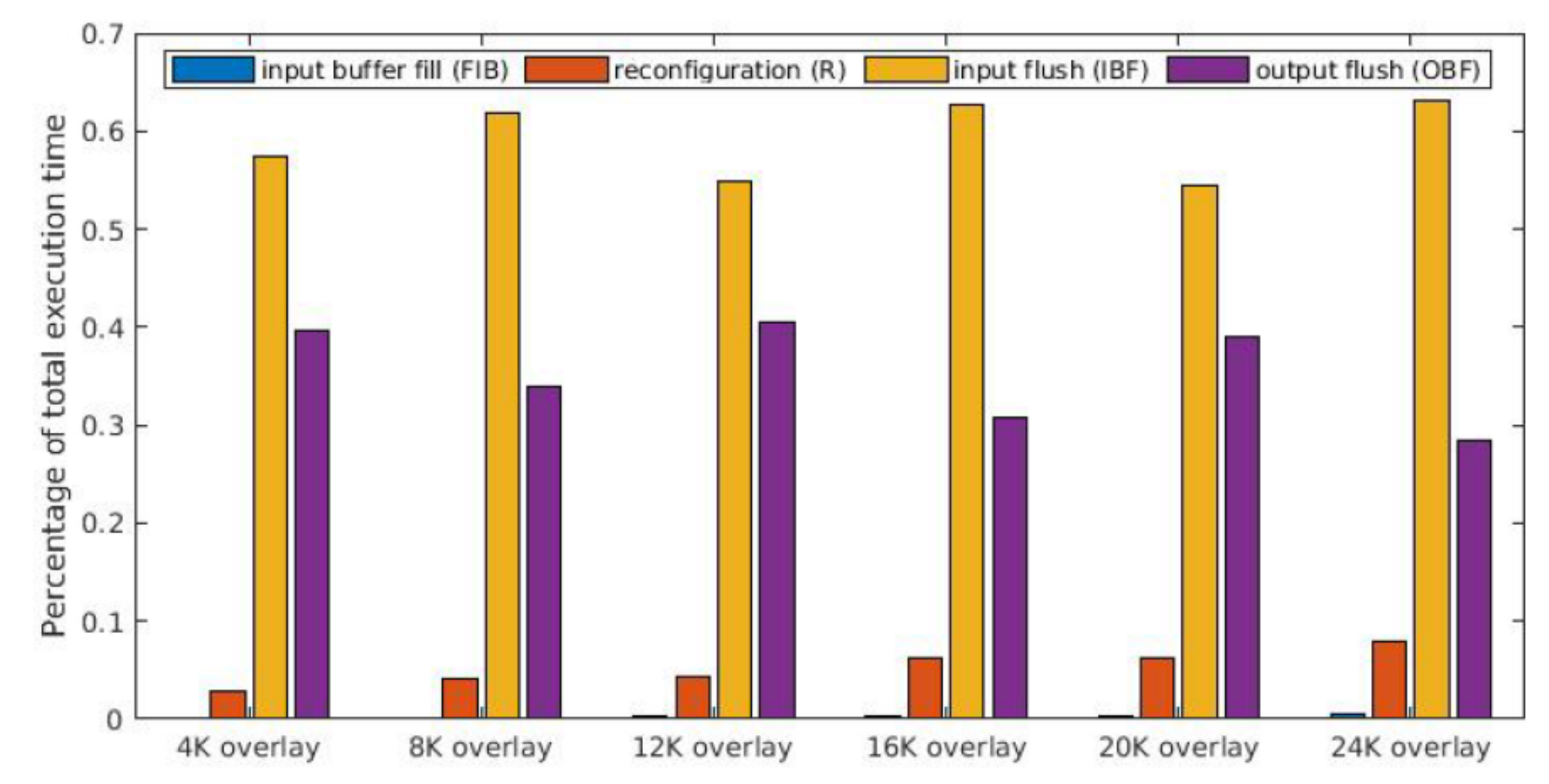
The major limitation in NAPOLY implementation is the hardware connections as shown in Fig. 3

Table 2. NAPOLY Wire Utilization

Benchmark	Max Logical Fan-in/ Fan-out	Min Hardware Fan-in/ Fan-out	Average Fan-in degree	Average Fan-out degree	Fan-in Wire Utilization	Fan-out Wire Utilization
Brill	4/4	8/8	1.11	0.72	13.8%	9%
ClamAV	11/2	18/18	1.01	1.003	5.6%	5.6%
DotStar	2/2	4/4	1.00	0.48	25%	12%
Entity Resolution	28/5	41/41	1.89	1.15	4.6%	2.8%
Fermi	2/2	5/5	1.33	1.41	26.6%	28.2%
Hamming	4/2	14/14	1.69	1.69	12%	12%
Levenshtein	8/5	16/16	2.89	1.63	18%	10.2%
PowerEn	4/3	6/6	1.08	0.51	18%	8.5%
Protomata	3/106	9/9	1.02	0.49	11.3%	5.4%
Random Forest	2/2	6/6	1.05	0.5	17.5%	8.3%
Snort	19/19	9/9	1.22	0.6	13.5%	6.6%
SPM	3/2	6/6	2.1	1.05	35%	17.5%

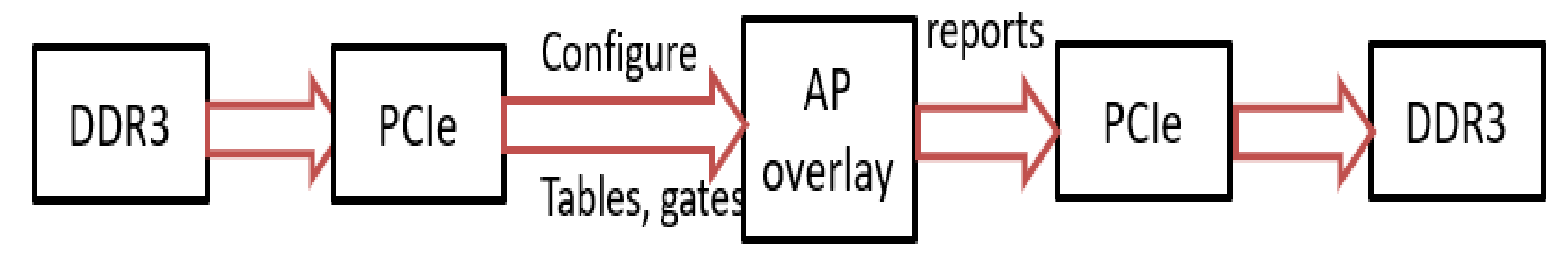
Major limitation in NAPOLY performance is the high latency for flushing the input and output buffers as shown in Fig. 2.

Fig. 5 Factors of NAPOLY Execution Time



Improve the rate of configuring NAPOLY Overlay using a combination of PCIe- DDR3 DRAM interface. To do this, we use AMAZON F2 Cloud.

Fig. 6 An End-to-End Overlay System

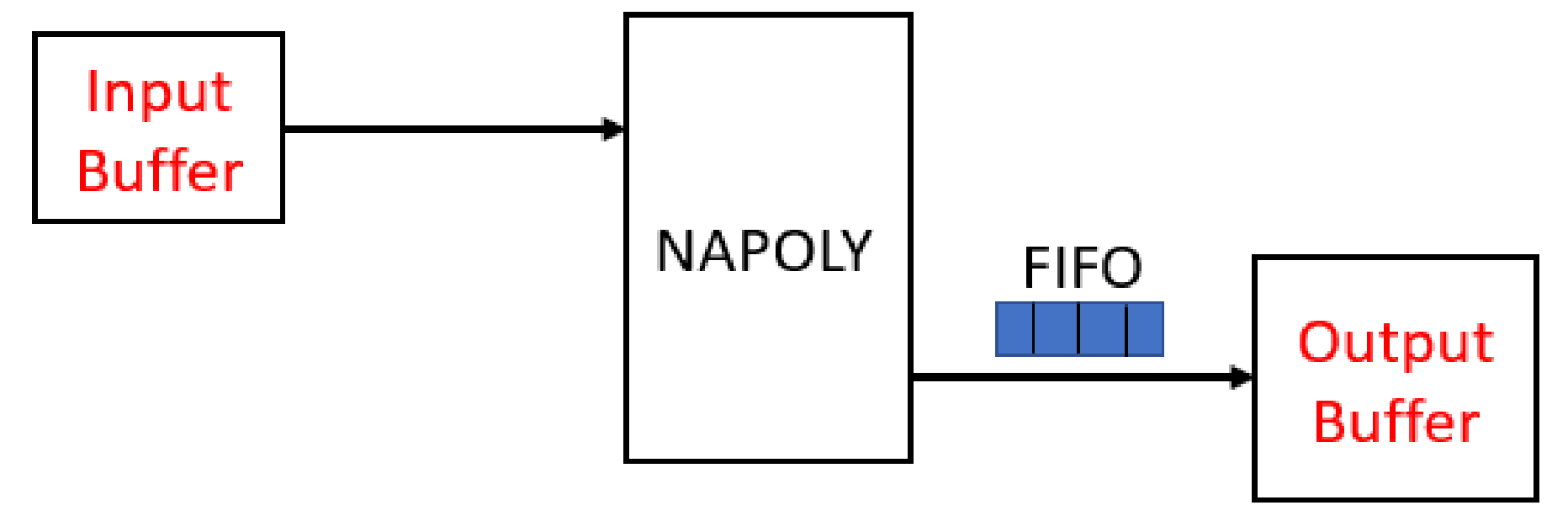


Methods

A. Hide flushing-data latency by:

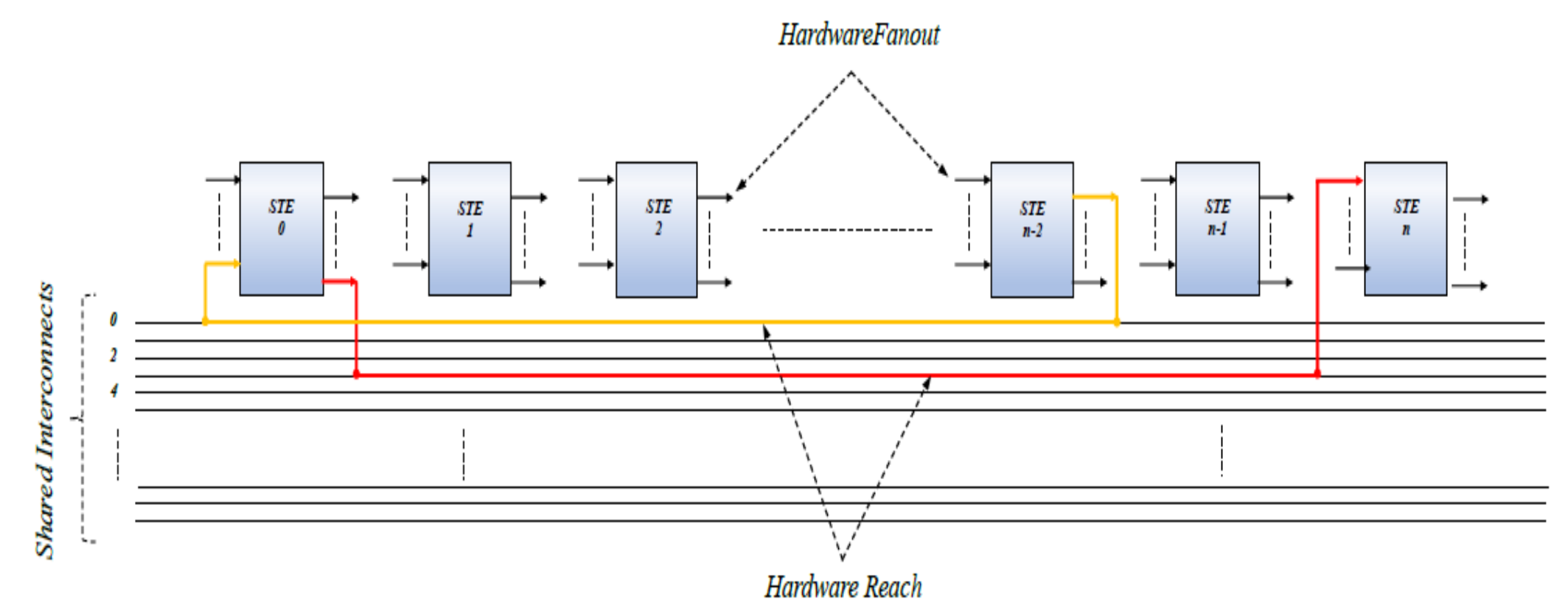
1. Splitting overlay into two halves
2. Overlapping the two operations
3. Adding FIFO to the output buffer

Fig. 6 NAPOLY is split into two halves to flush input and output buffers at same time



B.Solve connection bottleneck by

1. Re-design the overlay unit to support a smaller number of outgoing edges, but higher distance
2. Sharing wires among NAPOLY units



Reference

1. R. Karakchi, L. O. Richards and J. D. Bakos, "A Dynamically Reconfigurable Automata Processor Overlay," 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2017, pp. 1-8, doi: 10.1109/RECONFIG.2017.8279779.

2. R. Karakchi, C. Daniels and J. Bakos, "An Overlay Architecture for Pattern Matching," 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 2019, pp. 165-172, doi: 10.1109/ASAP.2019.000-7.