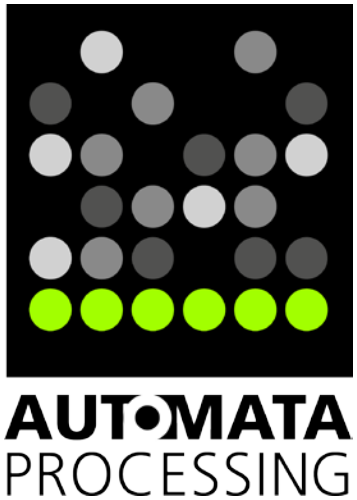
Automata Computing



***Mircea R. Stan (mircea@virginia.edu)*, UVA-ECE**
Center for Automata Processing (CAP) Co-Director
Kevin Skadron, UVA-CS, CAP Director

Questions to answer

- What is the Automata Processor (AP)?
- What can it do, and how does it do it?
- What is it, or is it not?
- What are some of the challenges?
- What are some of the opportunities?
- How can you get involved? – UVA Center for Automata Processing (CAP)

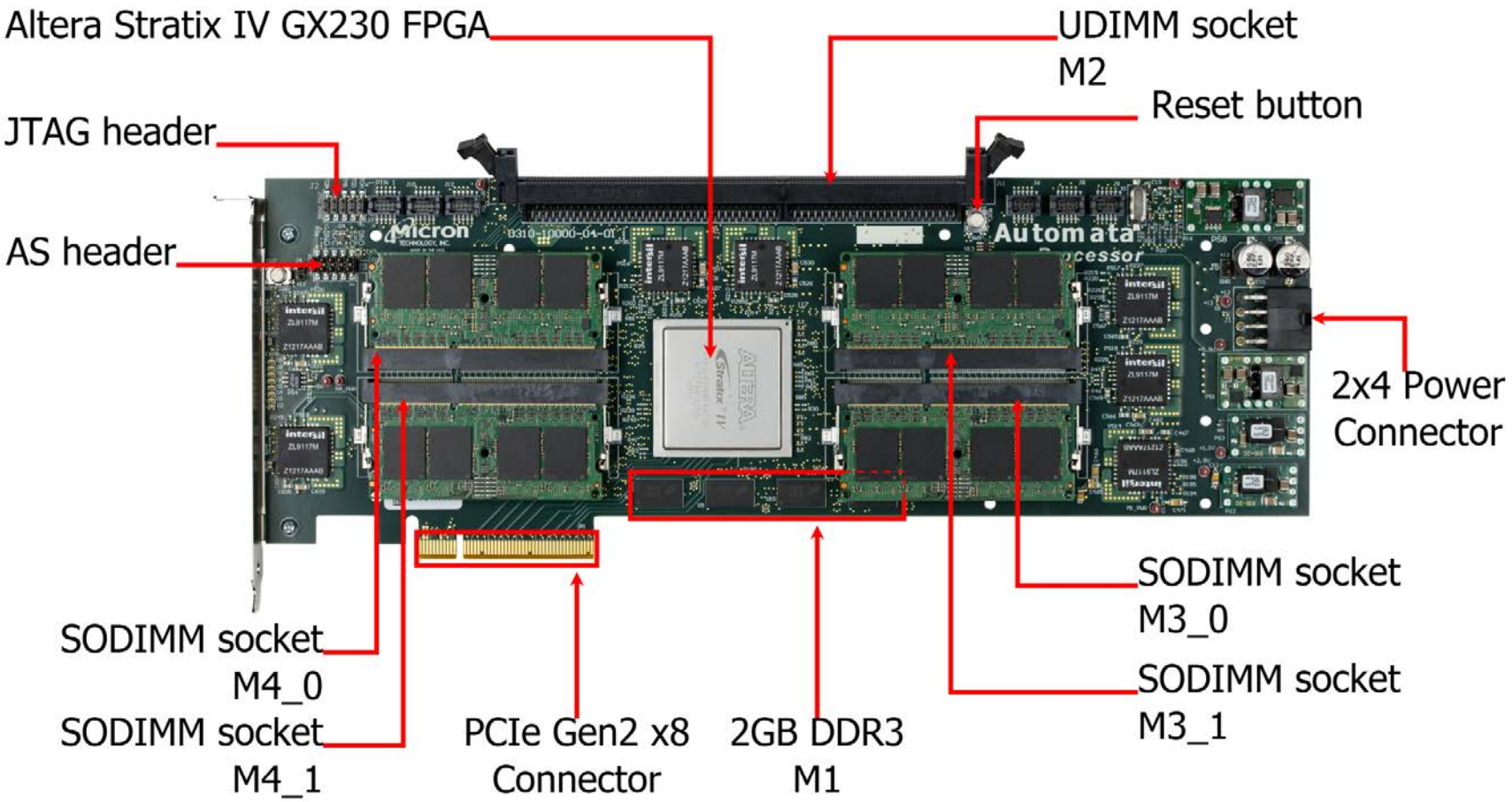
What it the Automata Processor (AP)?

The Automata Processor (AP) is a programmable silicon device developed by *Micron Semiconductor* capable of performing very high-speed, symbolic pattern matching, allowing comprehensive search and analysis of complex, unstructured data streams.

- Hardware implementation of *non-deterministic finite automata (NFA),* plus some extra features
- A massively parallel, scalable, reconfigurable, two dimensional fabric comprised of ~49,000 simple pattern-matching elements per chip. First-generation boards have 32 chips, giving ~1.5M processing elements/board
- Exploits the very high and natural level of low-level parallelism found in *DRAM technologies*
- On-board FPGA will allow sophisticated processing pipelines

Automata Processor Development Board

PCIe, 4 Ranks, 32 chips, 1.5M STEs

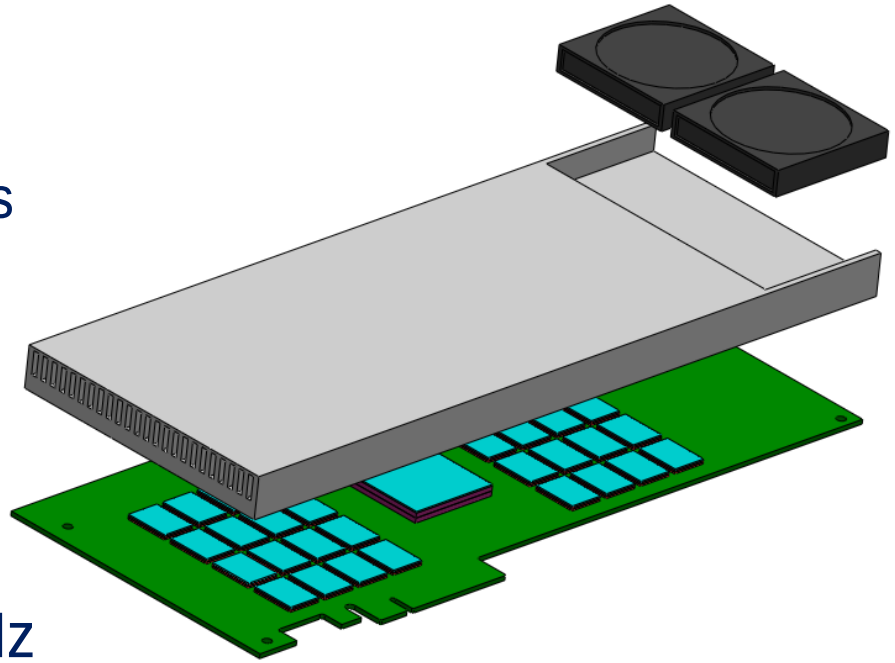


The FPGA provides substantial potential flexibility to augment the NFAs with other types of computation (NDP)

hplp
High-Performance Low-Power

AP Future Production Board

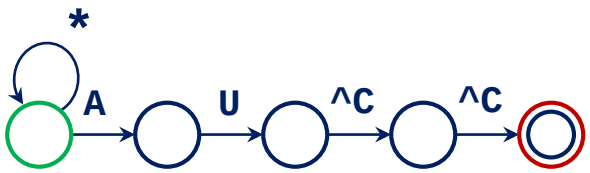
- PCIe Gen3 x8
  - 7.877 GBps is theoretical max
- Arria 10 GX270 FPGA
  - 270,000 LE's / 101,620 ALMs
  - 17,451 Kb Total Memory
- Four Ranks of AP
  - Soldered down
- 4GB DDR3 – 32bit @ 533 MHz



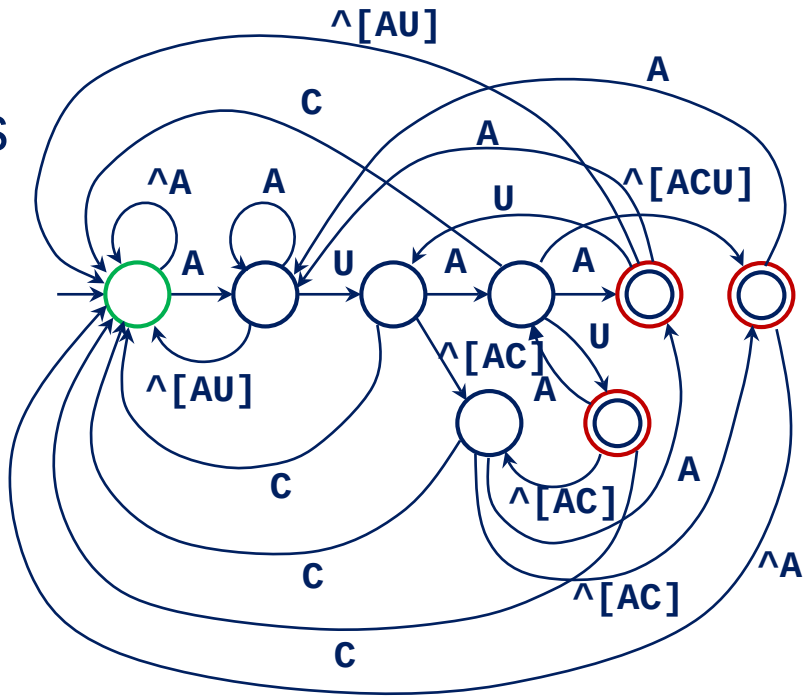
Figures courtesy of Micron

What can it do?

- Non-Deterministic Finite Automata (NFA)
- Any nondeterministic machine can be modeled as deterministic – at the expense of exponential growth in states
- SNORT example: 100 NFA nodes replace 10,000 DFA nodes



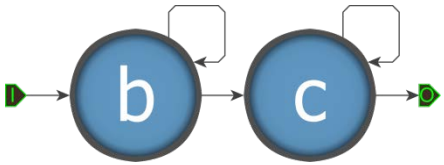
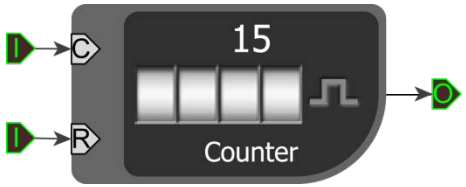
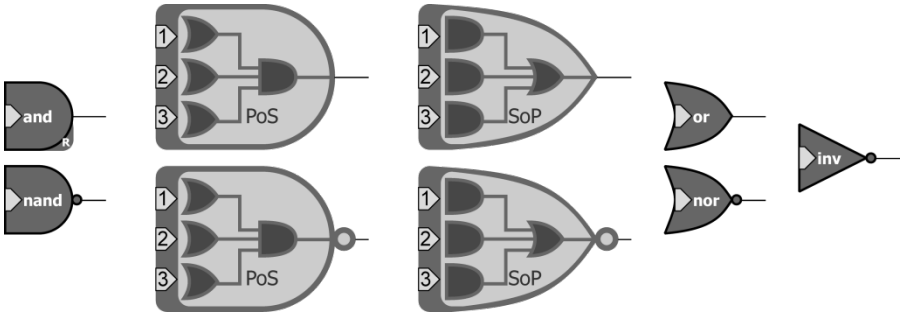
Nondeterministic Finite Automaton (NFA)



Deterministic Finite Automaton (DFA)

Figures courtesy of Micron

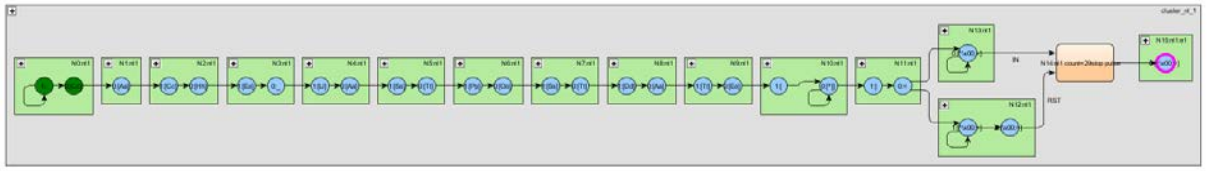
Automata Processor Hardware Building Blocks

	per chip
State Transition Element (STE) 	49,152
Counter Element 	768
Boolean Logic Element Nine Programmable Functions 	2,304
Parallel event output “virtual pins”	6,144

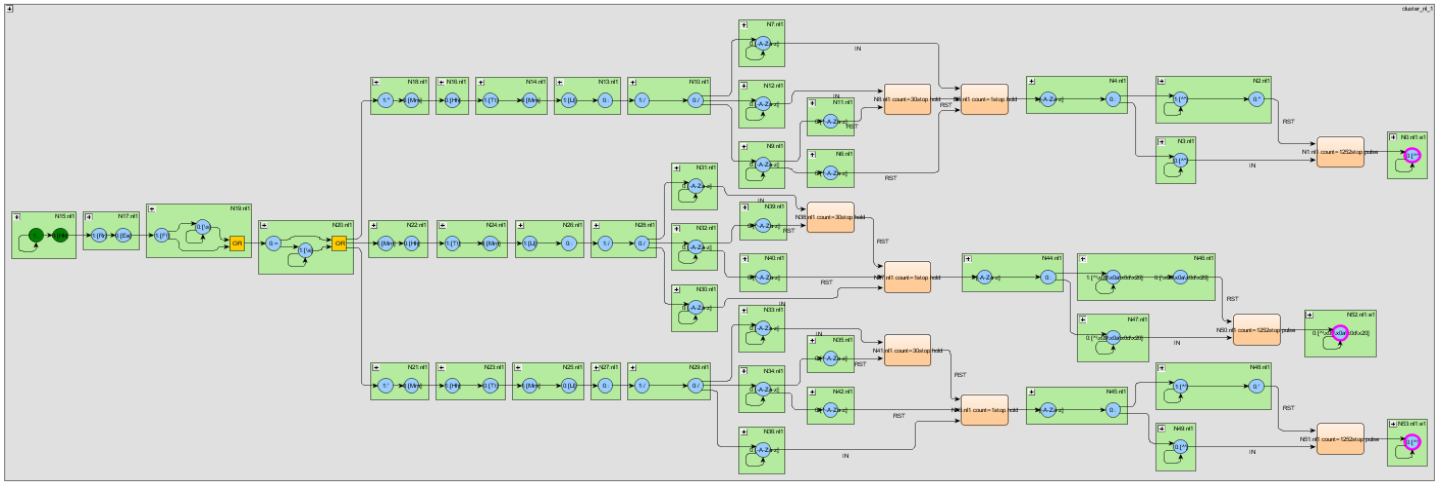
Figures courtesy of Micron

Parallel Automata

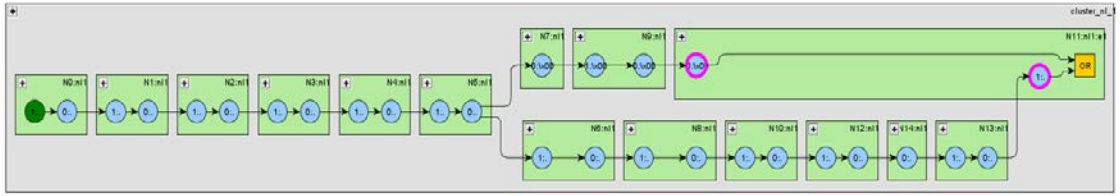
Pattern #1 →



Pattern #2 →



Pattern #3 →

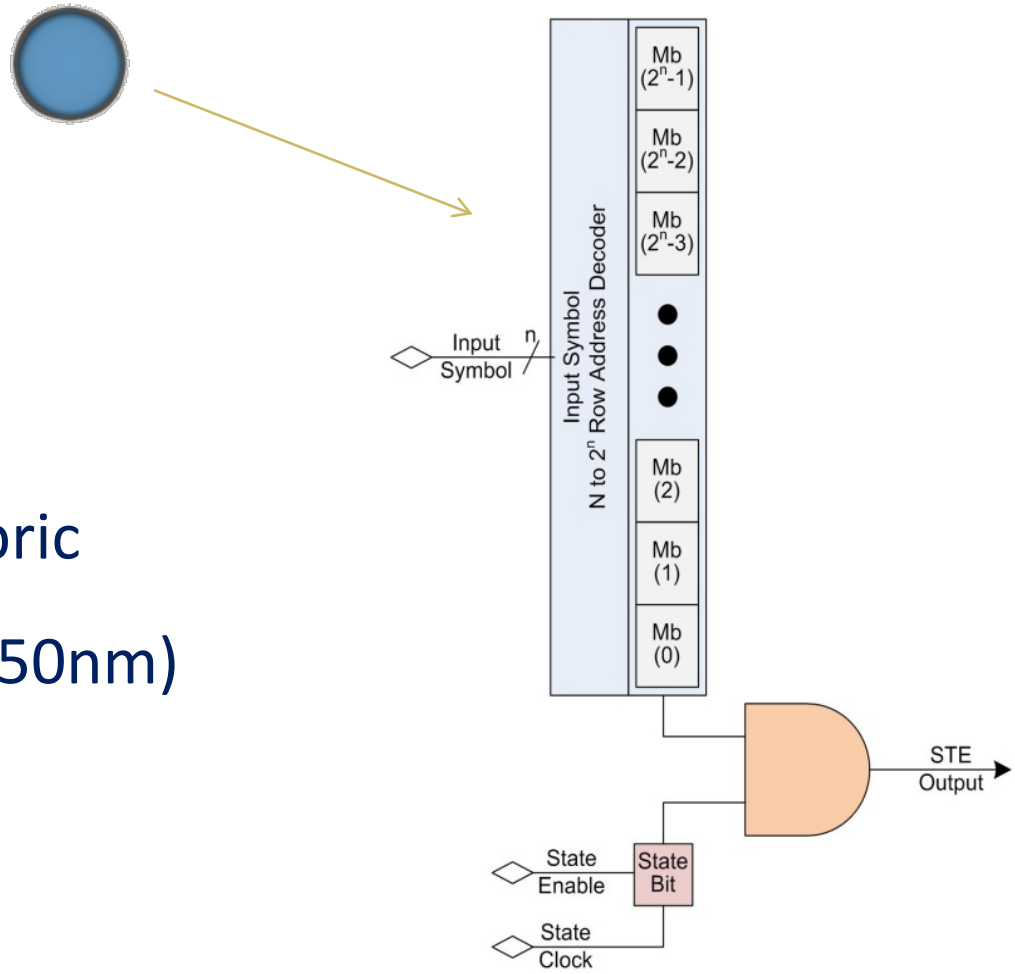


Parallelization of automata requires no special consideration by the user. Each automaton operates independently upon the input data stream.

Figures courtesy of Micron

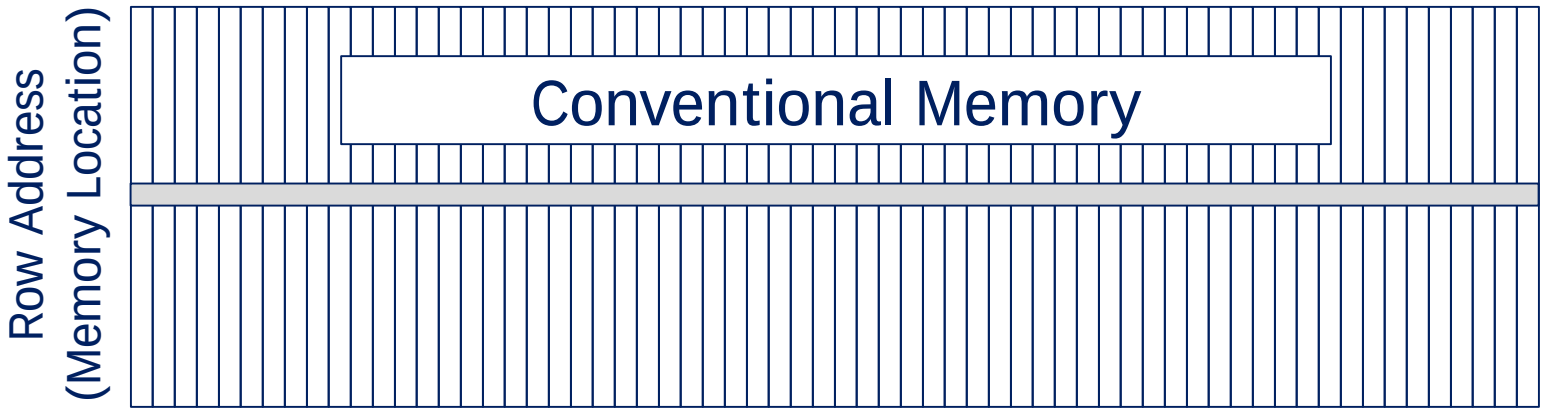
Automata Processor: Basic Operation

- STE "fires" when
  - Symbol match
  - AND the STE is active
- Why only ~49K STEs?
  - Substantial routing fabric
  - N-4 technology node (50nm)

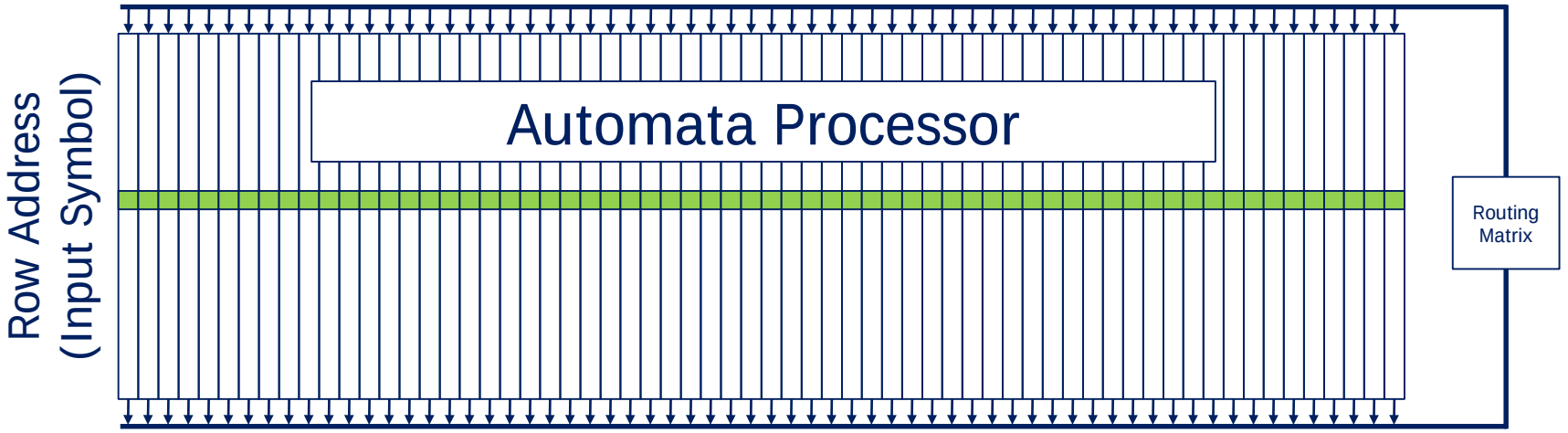


Figures courtesy of Micron

How does it do it?



Row Access results in **one** word being retrieved from memory.

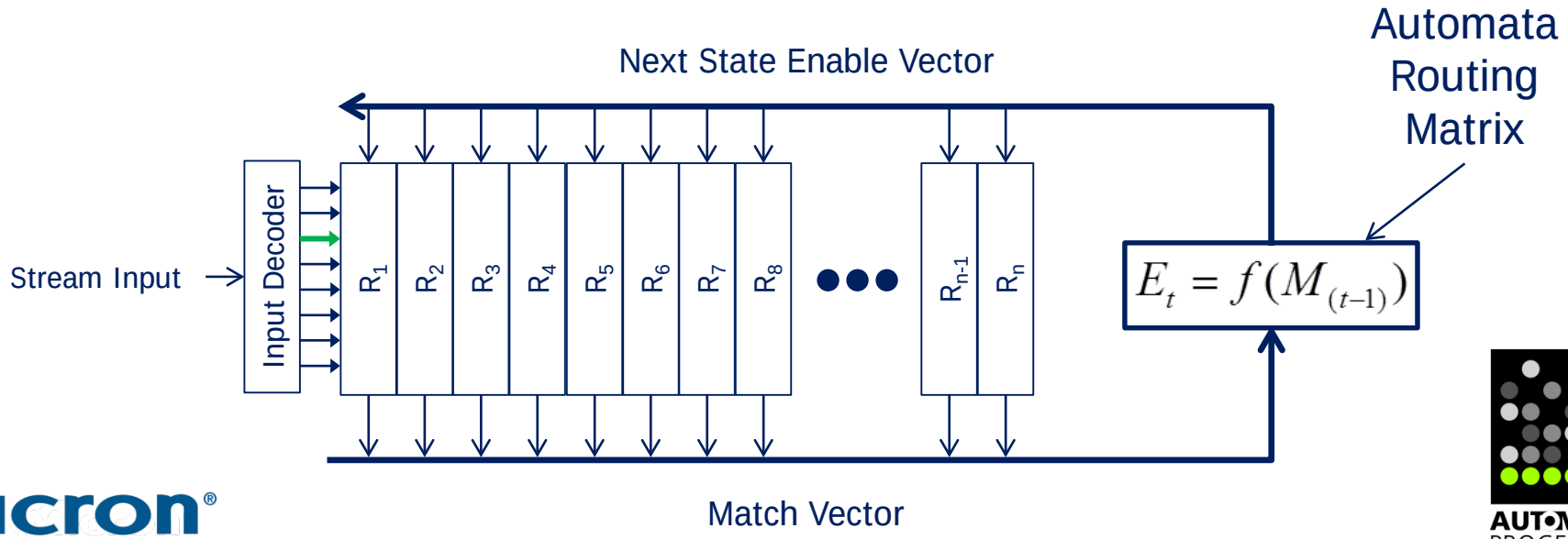


Row Access results in **49,152** match & route operations
(then Boolean AND with “active” bit-vector)

Figures courtesy of Micron

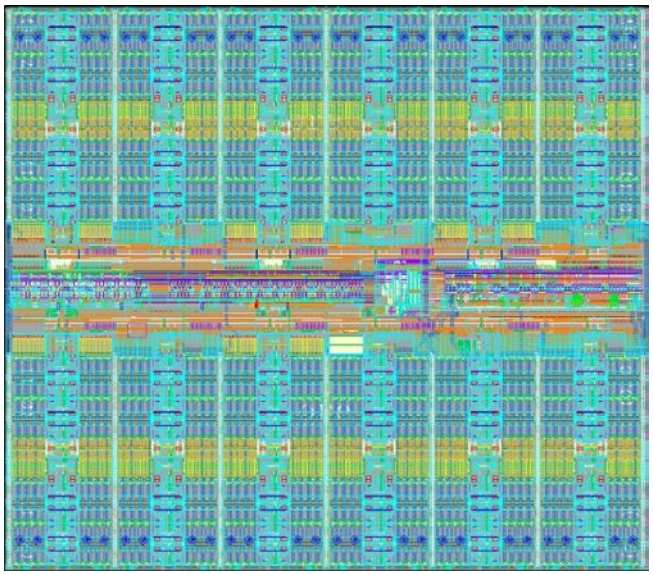
Automata Processor Hardware Concurrency

- Each column represents one STE
- Each row represents the response of every STE to a specific input symbol
- All operations for active elements performed concurrently
- Computes Next State Enable Vector for all state transitions and Match Vector for output on each cycle
- Each output can be routed to activate other STEs
- Has deterministic processing performance irrespective of the complexity of the automata
- Well suited for combinatorial problems



Basic Specifications

- First silicon fabricated in Micron's 300mm manufacturing facility in Manassas, VA
- First generation device is produced on a trailing edge (50nm) commodity DRAM process

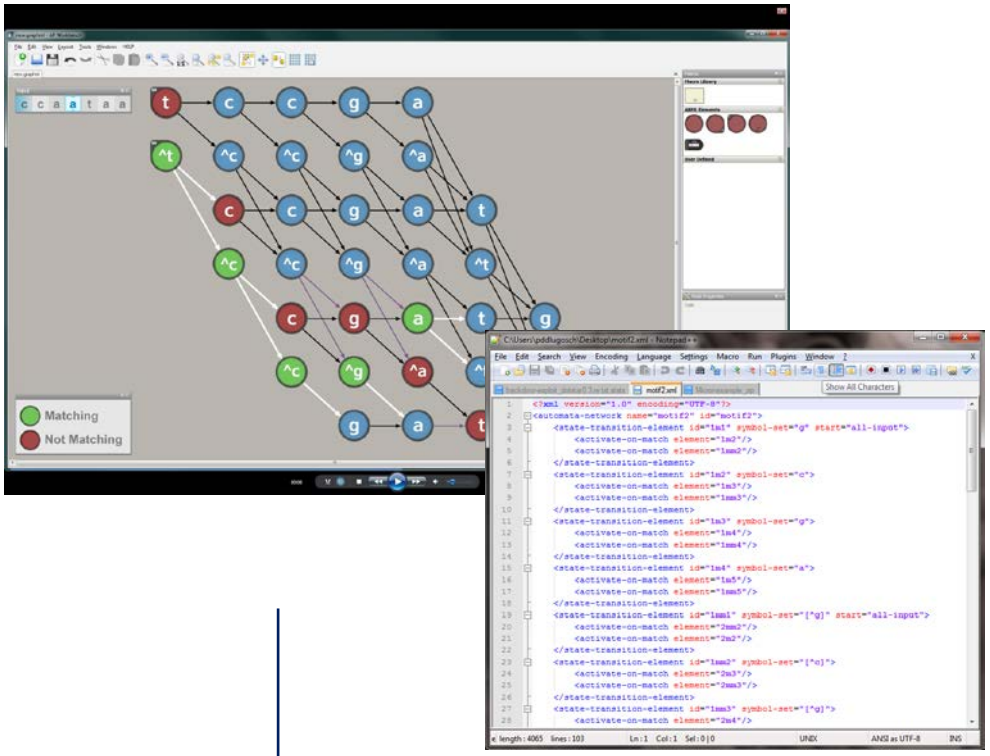


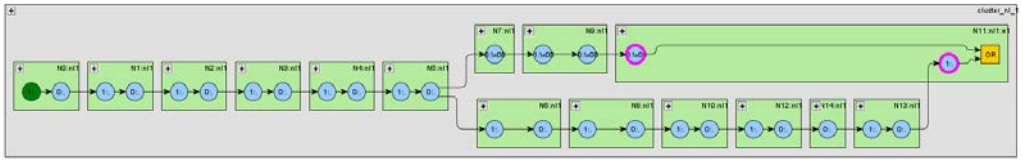
Basic Specs:

- 6.6T path decisions/second
- 4W Max TPD (estimated)
- 512 Entry State Vector Cache
- Inter-chip bus to facilitate scaling
- DDR3 physical interface
- < 150 mm^2 die size

Programming Options

- Input
  - RegEx
  - GUI – Workbench
  - C/Python
  - ANML
- Compiling
  - Input → ANML
  - ANML→ Netlist
  - Netlist → Place & route



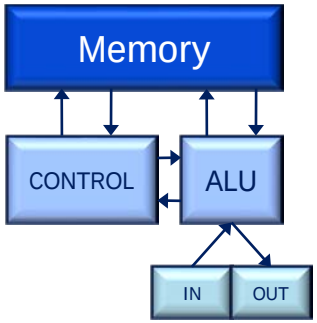


- (For details on programming, see backup slides)

Figures courtesy of Micron

Non von Neumann Parallel Architecture

- AP avoids the von Neumann bottleneck of instruction fetch
  - Instead: hardware reconfiguration



- AP converts time complexity to space complexity
- AP allows massive parallelism
  - Every automaton node can inspect every input symbol
    - (Leverages DRAM row activation)
  - Can process a new input symbol every clock cycle
- Fills the unusual “MISD” role in Flynn’s taxonomy

SISD	SIMD
MISD	MIMD

Many Layers of Parallelism

- Individual STE: test many different symbol matches per cycle, per input symbol
  - Von Neumann (VN) architecture needs multiple instructions
- Across STEs: different matching rules for an input position
  - VN needs multiple instructions
- Multiple activations: branching—activate many potential successor paths
  - Non-determinism very difficult in VN; exp. growth in space complexity
- Multiple automata: independent rules
  - VN requires multiple threads, limited capacity
- Multiple streams
  - VN requires multiple threads

What is it, or is it not?

The Memory is the Processor!

- Processor in Memory: no “processing” in the Von Neumann sense
- FPGA: no general purpose logic
- Systolic Arrays: no arithmetic
- Processor in memory technology: yes
- MISD “processor”: central data, multiple “instructions” (reconfigurable hardwired)

What are some of the challenges?

- No arithmetic, only counting (on-board FPGA can help)
- Changing the “program” requires a reconfiguration step
- DRAM technology (Micron problem ☺)
- New programming model
- Low-level programming (except for RegEx)
- Need to buy another chip/board for your system (compare with GPU)
- Resurgence of FPGA acceleration (Intel/Altera, IBM CAPI)

What are some of the opportunities?

- Significant speed-ups possible on subset of applications
- Complex/fuzzy pattern matching
- Combinatorial search space
- Highly parallel set of analysis steps for each input item
- Unstructured data, unstructured communication
- Another accelerator flavor in a heterogeneous SISD (single “beefy” core)/SIMD (vector)/MISD (AP)/MIMD (many core) system
- Could act as “dumb” memory when not needed for acceleration (DIMM form factor preferable IMHO)

Problems Aligned with the Automata Processor

Applications requiring **deep analysis** of **data streams** containing **spatial** and **temporal** information are often impacted by the **memory wall** and will benefit from the **processing efficiency** and **parallelism** of the Automata Processor

Network Security:

- Millions of patterns
- Real-time results
- Unstructured data

Bioinformatics:

- Large operands
- Complex patterns
- Many combinatorial problems
- Unstructured data

Video Analytics:

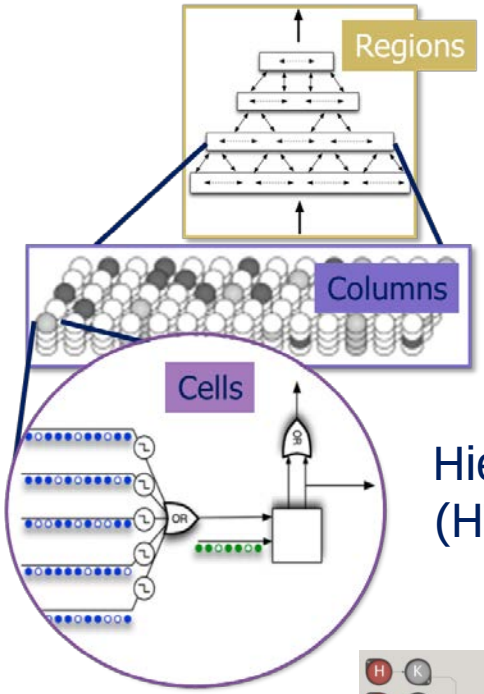
- Highly parallel operation
- Real-time operation
- Unstructured data

Data Analytics:

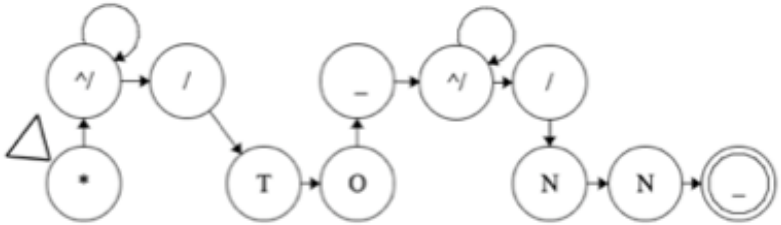
- Highly parallel operation
- Real-time operation
- Complex patterns
- Many combinatorial problems
- Unstructured data

So far: 10-100X+ speedups possible!

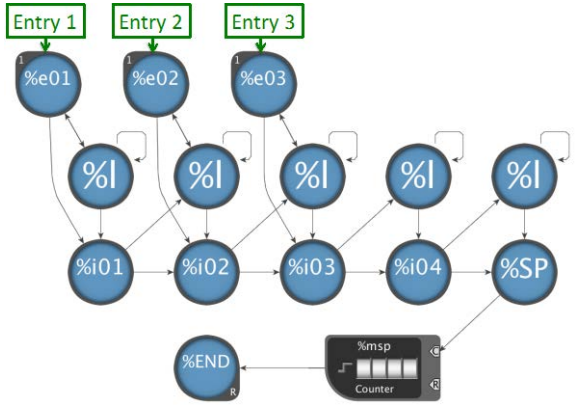
Applications pursued at UVA



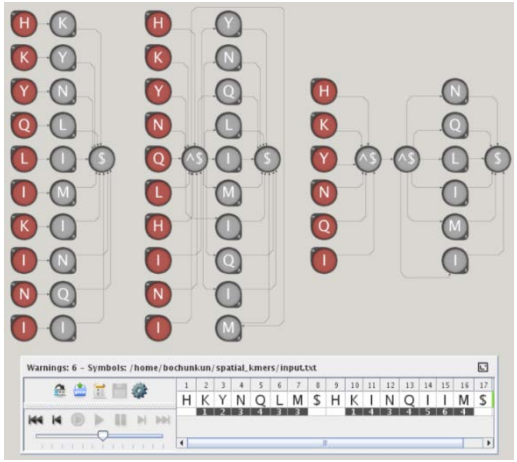
Hierarchical Temporal Memory (HTM)



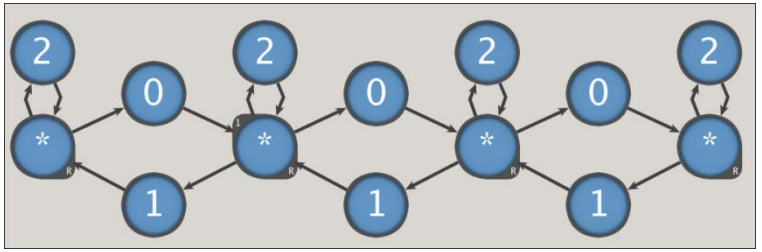
Brill Tagging



Association rule mining



DNA motif search



Markov chain

Brill Part-of-Speech (POS) Tagging

(Keira Zhou, Don Brown, UVA Systems Engineering), ICSC 2015

- A task in Natural Language Processing (NLP)
- Grammatical tagging of words in text (corpus)
  - E.g.

  Cats love dogs. -> POS Tagger -> Cats/noun love/verb dogs/noun ./.
- Complicated:
  - E.g. I book tickets. -> *book*: Noun? Verb?
- Baseline tagging:
  - Tag each word to its most frequent tag based on training corpus

Brill Tagging

- A two-stage tagging technique
- Stage 1: Baseline tagging
- Stage 2: Update tags based on some rules (AP)
  - Example rule: NN VB PREVTAG TO

*… to/TO conflict/**NN** with/IN … ->* Apply the Rule *-> … to/TO conflict/**VB** with/IN …*

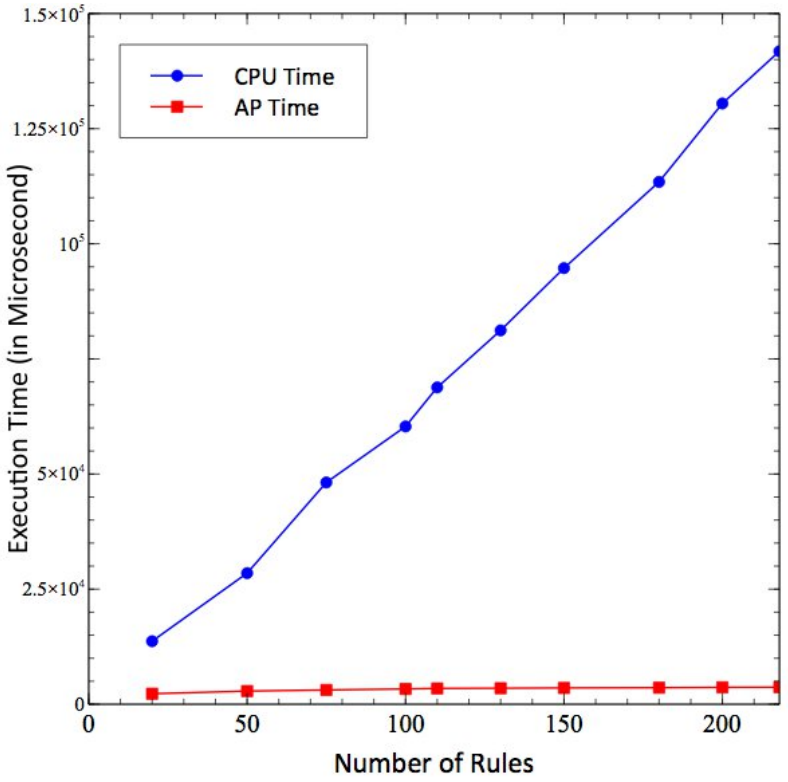
If current tag is NN, previous tag is TO, update current tag to VB

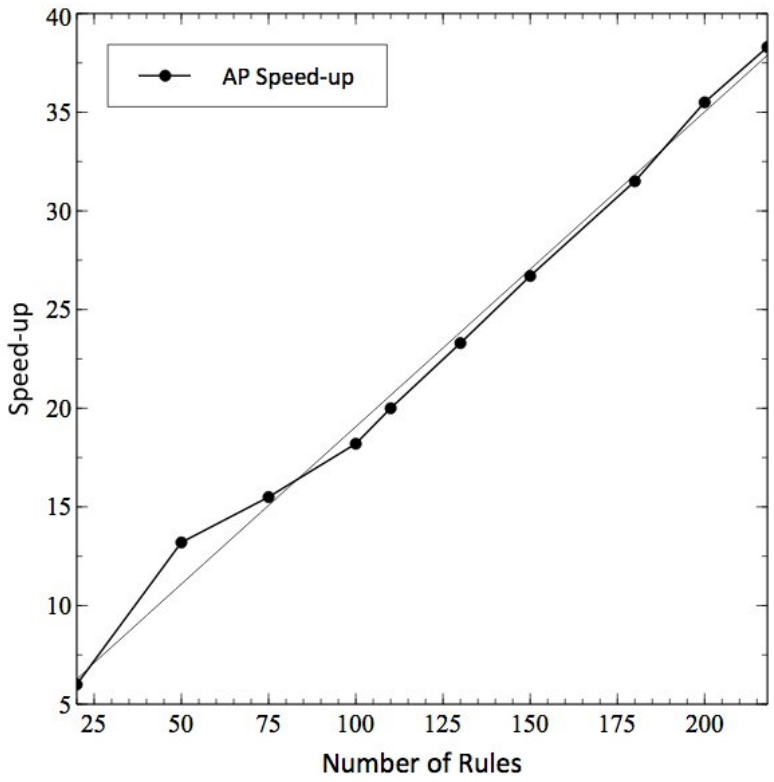
  - 218 context-based rules trained from training corpus publicly available → RegEx's
  - Maximum span: 3 words ahead or 3 words after
  - Slow on CPU but fits the structure of the AP

Brill, Eric. "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging." Computational Linguistics 21.4 (1995): 543-565.

Results

- Performance of the AP as a function of the number of rules





- Maximum number of rules in the literature: 1729 [5]
  - Estimated Speed-up: **276X**
- Processing time is proportional to input length

Brill, Eric. "Unsupervised learning of disambiguation rules for part of speech tagging." *Proceedings of the third workshop on very large corpora. Vol. 30. Association for Computational Linguisstics*, 1995

How can you get involved?
UVA Center for Automata Processing (CAP)

Founded with seed funding from Micron and UVA to:

- Build critical mass of academic/industry collaborations
- Provide early access to cluster of APs
- Train and provide programming support
- Develop foundational, open-source building blocks
- Share IP
- Act as a clearing house for industry-academic funding, proposal teaming

http://cap.virginia.edu/

Summary

- The Automata Processor is a unique example of a MISD architecture built in a DRAM technology that can significantly accelerate specific applications
- Trades-off time for space complexity
- As long as a problem “fits” it transforms combinatorial explosion into linear
- Hardware is being sampled now: CAP

Questions?