THE CENTER FOR AUTOMATA PROCESSING

UNIVERSITY of VIRGINIA

STRING KERNEL TESTING ACCELERATION USING MICRON'S AUTOMATA PROCESSOR

Chunkun Bo[1,2], Ke Wang[1,2], Yanjun (Jane) Qi[1], Kevin Skadron[1,2]

[1]Department of Computer Science

[2]Center for Automata Processing

University of Virginia

June 14th, 2015

Outline

- Motivation & Contribution
- Introduction to Micron's Automata Processor
- Introduction to String Kernel (SK)
- Automata design for various SK mapping functions
- Initial Performance Evaluation
- Summary & Future Work

Motivation

- String Kernel (SK) is a widely used kernel in machine learning and text mining
- Fast processing is required, especially for the testing phase
- Feature vector mapping is the current performance bottleneck, which involves a lot of pattern matching
- Micron's Automata Processor (AP) can implement nondeterministic finite automata (NFA) directly in hardware, and match complex regular expressions in massive parallelism

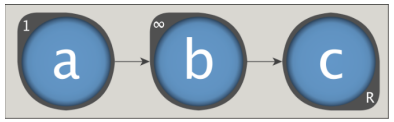
Using the AP to accelerate String Kernel Testing

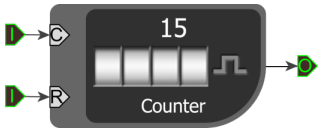
Contributions

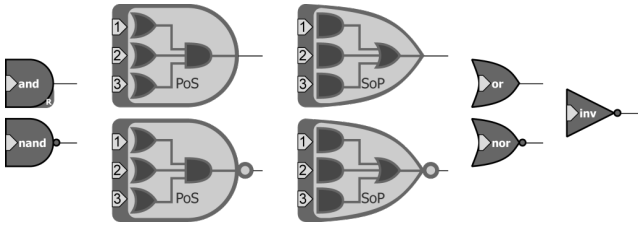
- Propose a novel AP-accelerated framework for String Kernel
- Present various automata designs that can process different mapping functions
  - E.g. mismatch kernel, gappy kernel, spacial kernel, etc.
- Compare the proposed method with state-of-the-art CPU methods
  - Performance results show great speedup

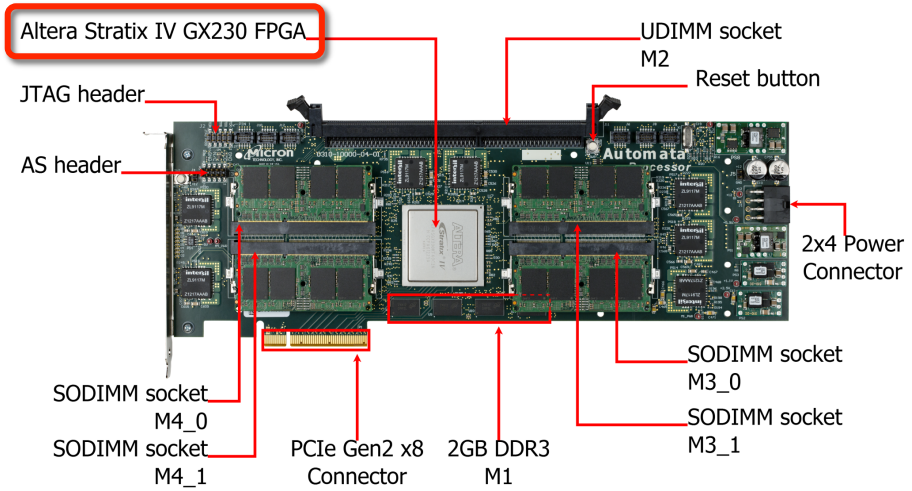
AP Architecture Overview

- An efficient and scalable semiconductor architecture for parallel automata processing
- Functional Elements
  - State Transition Elements (STE)
    - consist of current state memory and next state decoder
    - start, all-input, reporting
  - Counter Elements (12-bit)
  - Boolean Elements
    - *OR, AND, NAND, NOR, sum of products, etc.*
- Hardware resources of a 32-chip AP board:
  - STEs: 1,572,864
    Reporting STEs: 196,608
  - Counter Elements: 24,576
  - Boolean Elements: 73,728
  - 133 MHz
  - FPGA



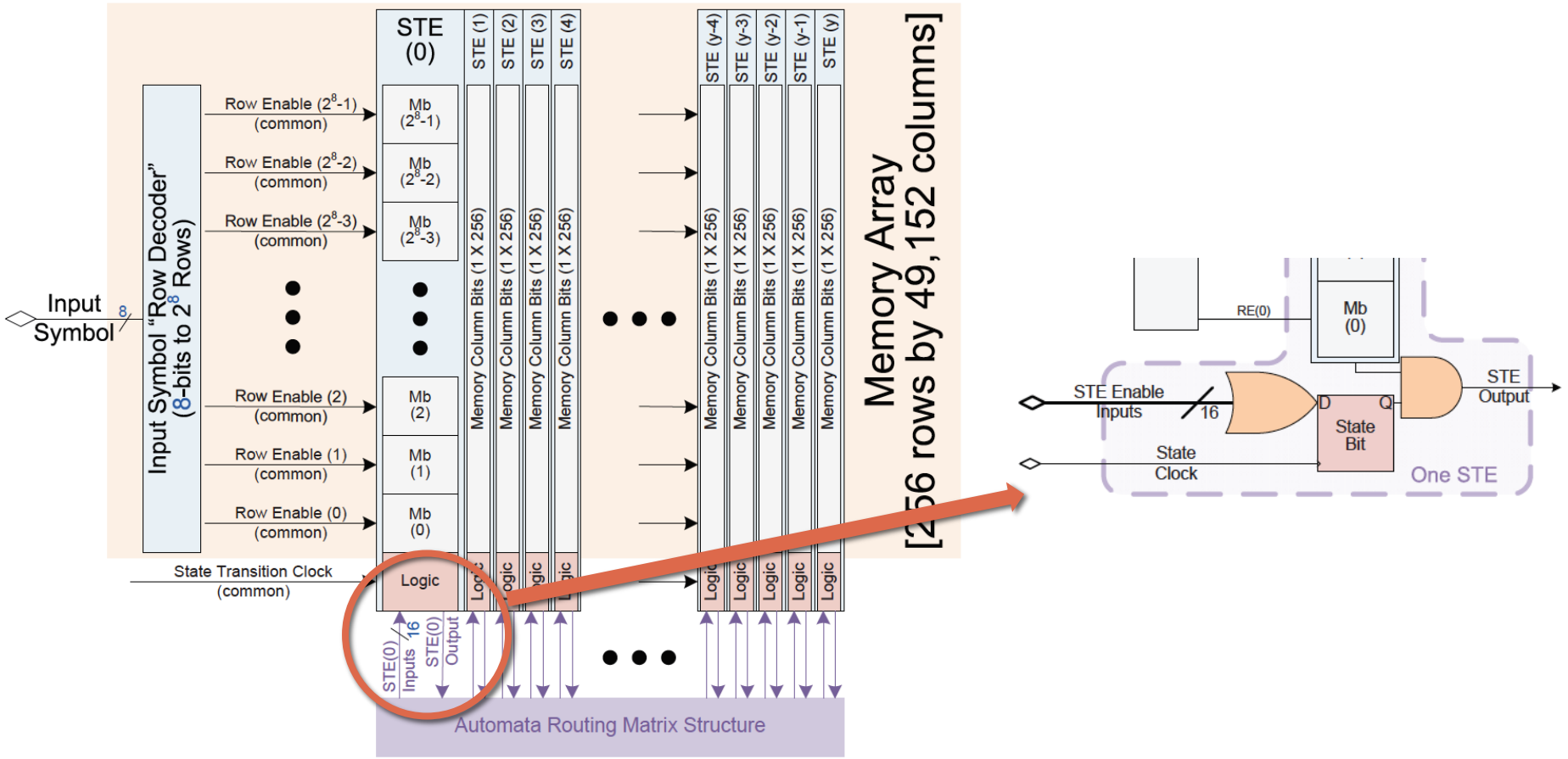






Introduction to AP

- Uses a non-Von-Neumann architecture and directly implements NFA in hardware
- Capable of matching complex regular expressions



Introduction to AP

- Programming
  - Automata Network Markup Language (ANML): describes composition of automata networks
  - Graphical user interface tool (AP Workbench)
  - C and Python interfaces
  - Macro: a container of automata
- Reconfiguration
  - Symbols in an STE can be reconfigured
  - Takes 0.24ms for one block

Introduction to String Kernel

- Definition
  - A function to differentiate strings

- Subsequence:
  - Any ordered sequence of K characters occurring in input sequence (not necessarily contiguously)
  - Also known as K-mers
  - E.g. cart -> car, art, cat, crt

Introduction to String Kernel

- Mapping function $\phi(x)$
  - Project the input sequence to a high-dimensional feature space generated by the K-mers
  - Make it possible to draw the hyper-plane to classify the input sequences
- Kernel function
  - Inner product in the feature space
  - $K(x , y) = < \phi(x), \phi(y) >$
- Kernel Matrix ($N \times N$)
  - Stores all the inner products of input pairs

Introduction to String Kernel

- Why is it important?
  - Extension of previous classification methods that cannot be vectorized
  - Able to process sequence data
  - A critical kernel for many applications

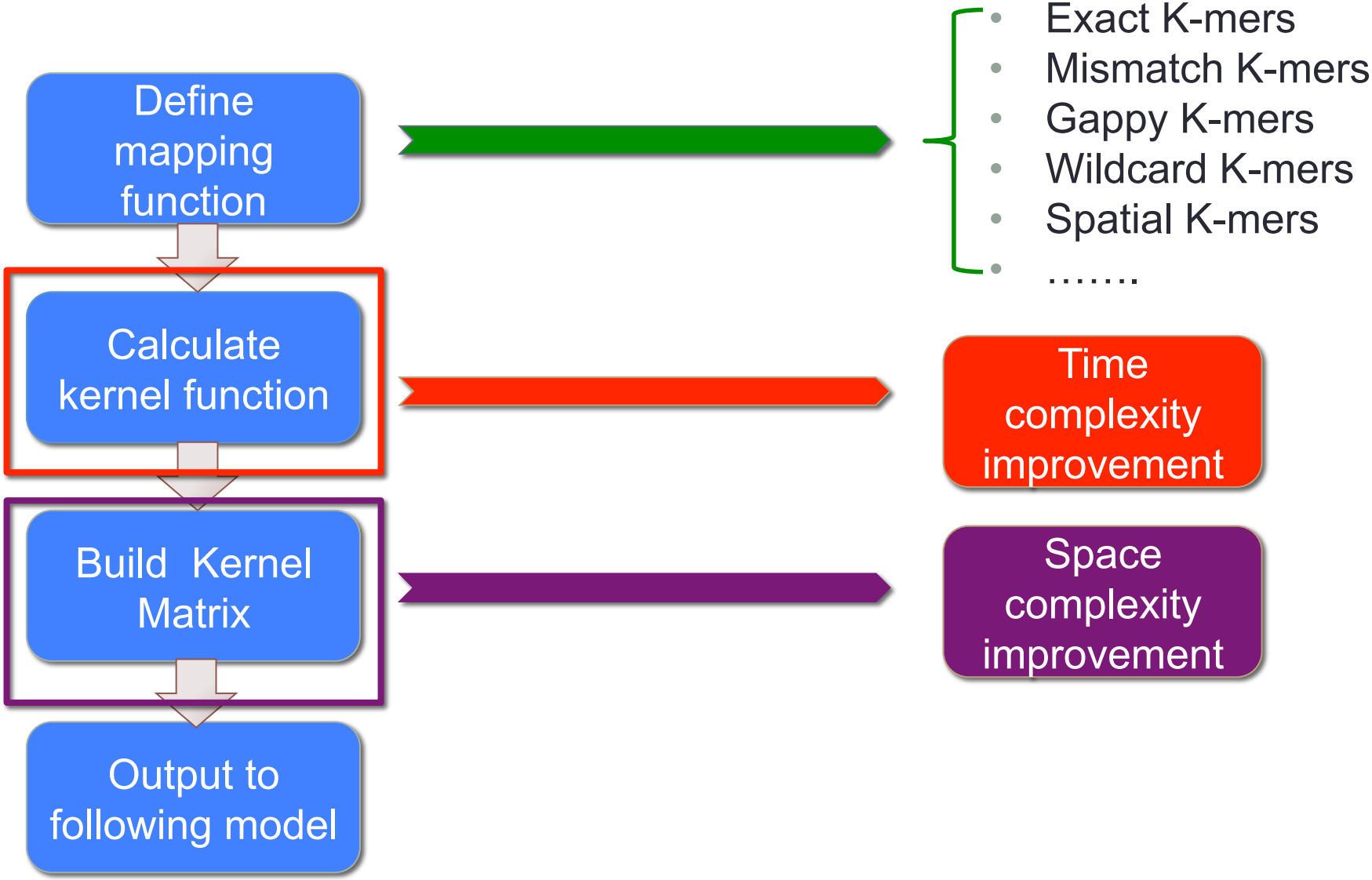
    E.g. bio-sequence analysis (DNA/RNA/Protein classification)

    text/document classification

    action categorization

    …
- Challenges
  - Computationally expensive for large data sets
  - Fast computation of feature vector is required, especially for testing

String Kernel Method Procedure



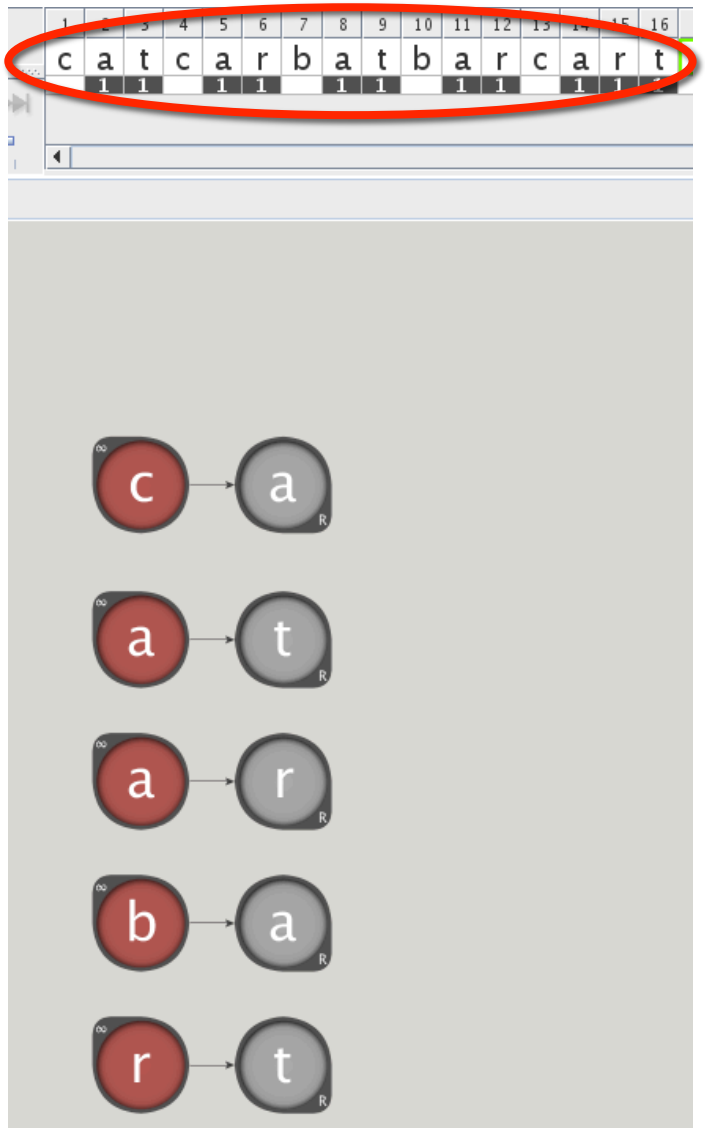
Design in AP

- Exact Match Kernel
  - K = 2
  - Input: cat, car, bat, bar, cart
  - Kernel Function Results

    k(bat, car) = 0

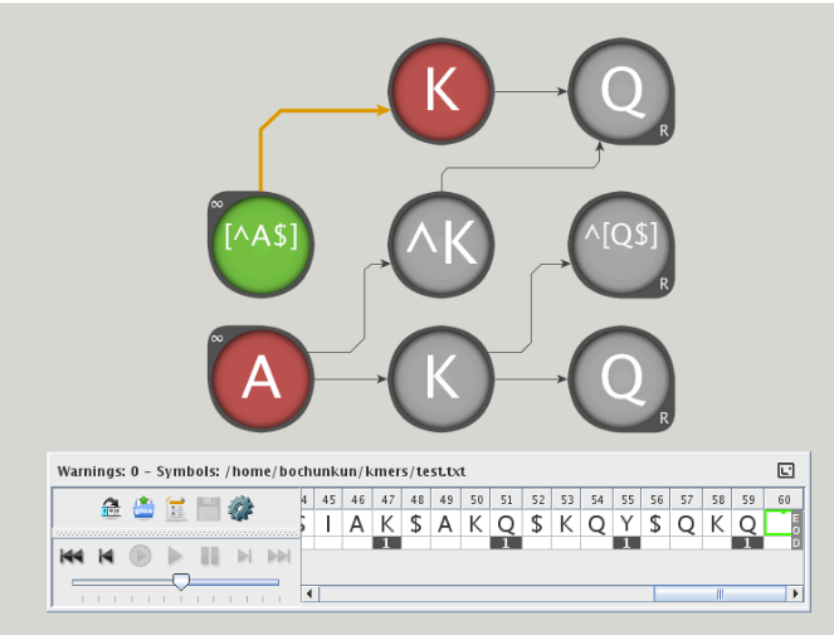
    k(cat, car) = 1

	ca	at	ar	ba	rt
cat	1	1	0	0	0
car	1	0	1	0	0
bat	0	1	0	1	0
bar	0	0	1	1	0
cart	1	0	1	0	1

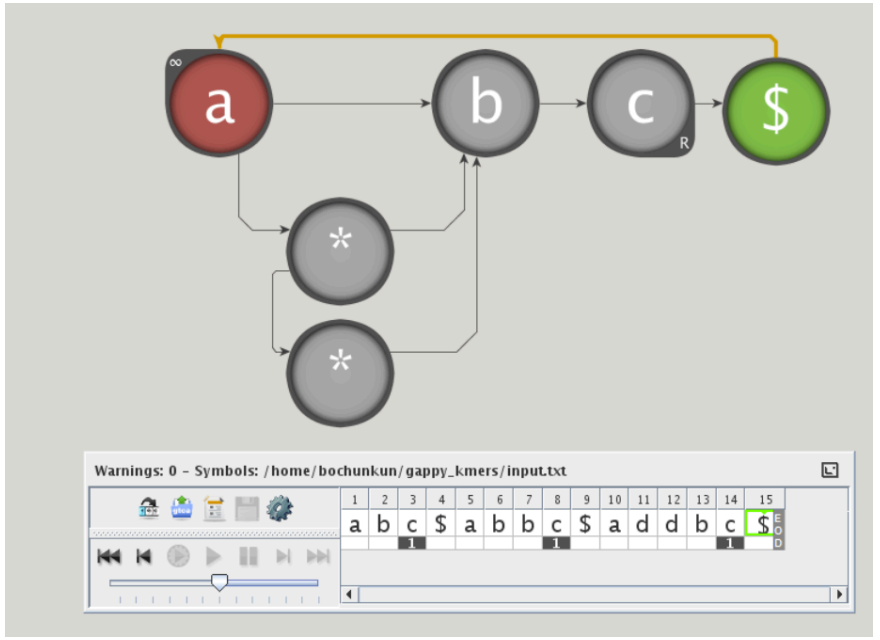


Design in AP

- Mismatch kernel
  - K = 3
  - m = 0, 1



- Gappy kernel
  - K = 3
  - g <= 2



Design in AP

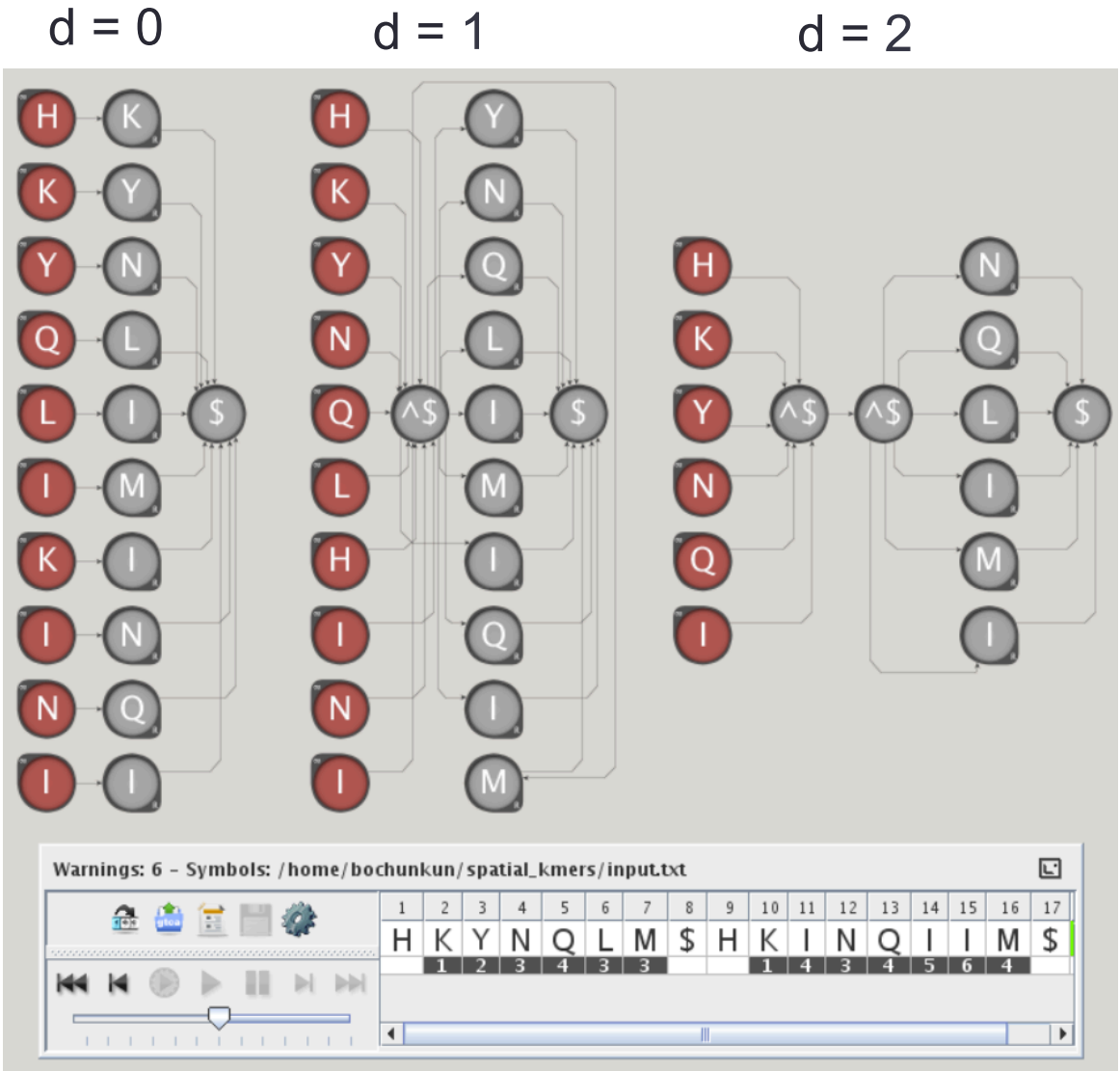
- Spatial Kernel
  - t = 2, k = 1, d < 5

  Input1 = HKYNQLM

  Input2 = HKINQIIM

HK	H_Y	H__N	H___Q	H____L
KY	K_N	K__Q	K___L	K____I
YN	Y_Q	Y__L	Y___I	Y____M
NQ	N_L	N__I	N___M	
QL	Q_I	Q__M		
LI	L_M			
IM				

HK	H_I	H__N	H___Q	H____I
KI	K_N	K__Q	K___I	K____I
IN	I_Q	I__I	I___I	I____M
NQ	N_I	N__I	N___M	
QI	Q_I	Q__M		
II	I_M			
IM				



Time Complexity Improvement

- CPU algorithm time complexity

Method	Complexity
Gappy[1]	$O(g^{g-k}nN)$
Mismatch[1]	$O(K^{m+1}\|\Sigma\|mnN)$
Wildcard[1]	$O(K^{m+1}nN)$
Spatial(double) (Triple)[2]	$O(dnN)$ $O(d^2nN)$

N: number of input sequences
n: sequence length
K: subsequent length
g: gaps allowed
m: mismatch allowed
$|\Sigma|$: dictionary size
d: distance between subsequence

[1]Leslie, Christina, and Rui Kuang. "Fast string kernels using inexact matching for protein sequences." *The Journal of Machine Learning Research*, 2004

[2]Kuksa, Pavel. "Scalable kernel methods and algorithms for general sequence analysis." PhD diss., Rutgers University-Graduate School-New Brunswick, 2011.

- AP time complexity: $O(nN)$

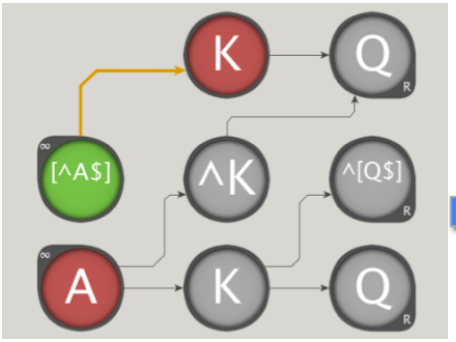
Experiment Setup

- Use PatMaN as the representative CPU method
  - PatMaN: fast tool for searching nucleotide sequence in large databases, allowing for a predefined number of gaps and mismatches

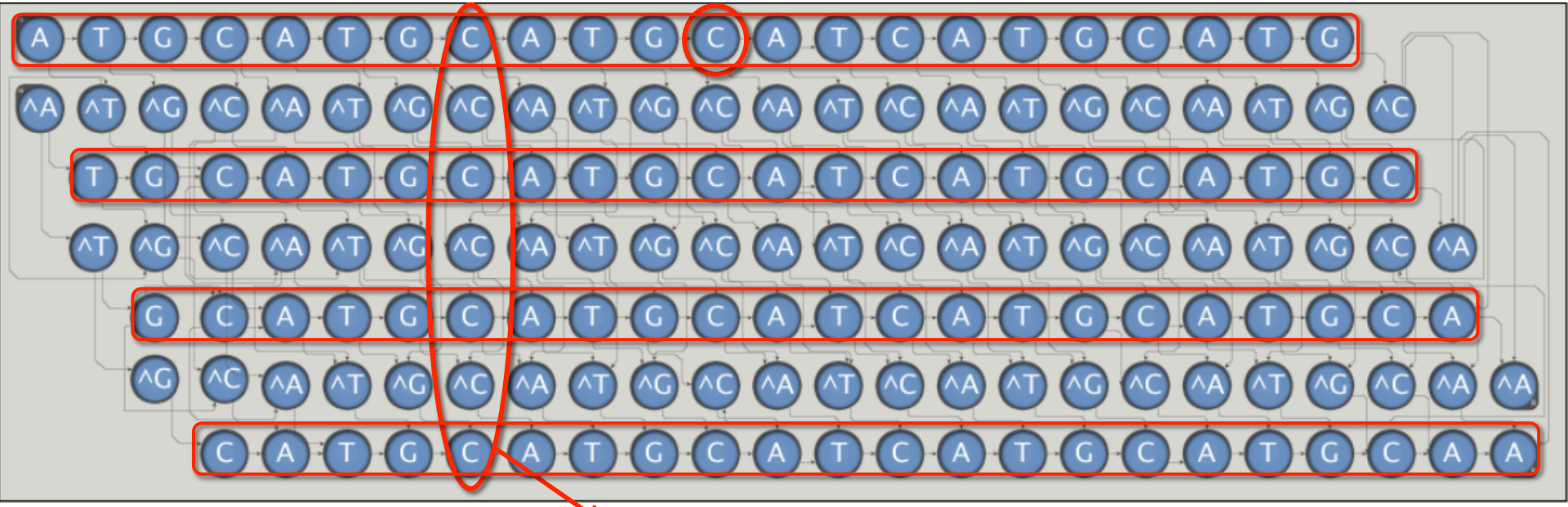
    https://bioinf.eva.mpg.de/patman/
- Different mismatch number: 0, 1, 2, 3, 4, 5
- Different input length: 50million ~ 500million
- Experiment data:
  - Input sequence: DNA sequence
  - Pattern Data: 200,000 25-mers

AP design

- Structure for mismatch kernel



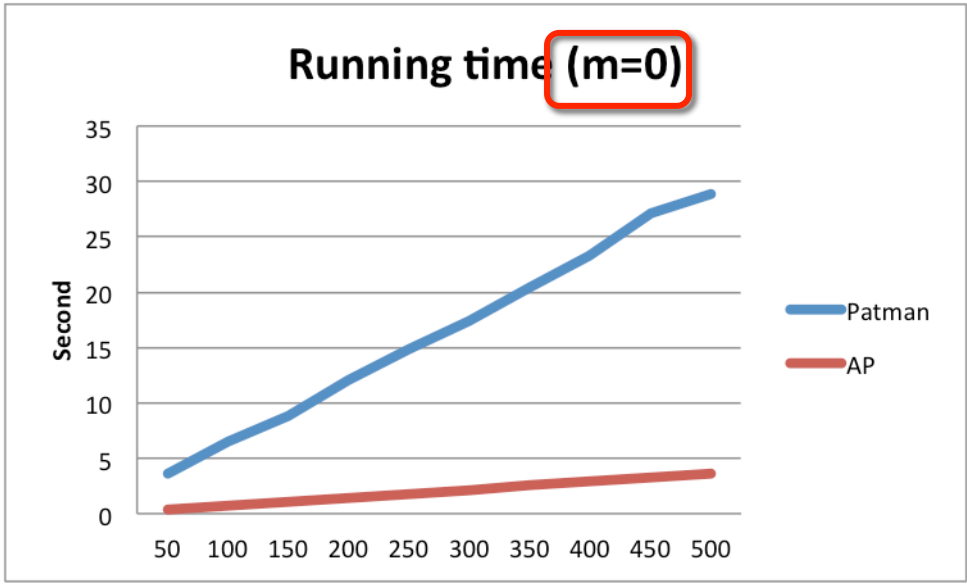
Sequence “AKQ”, K = 3, m = 1

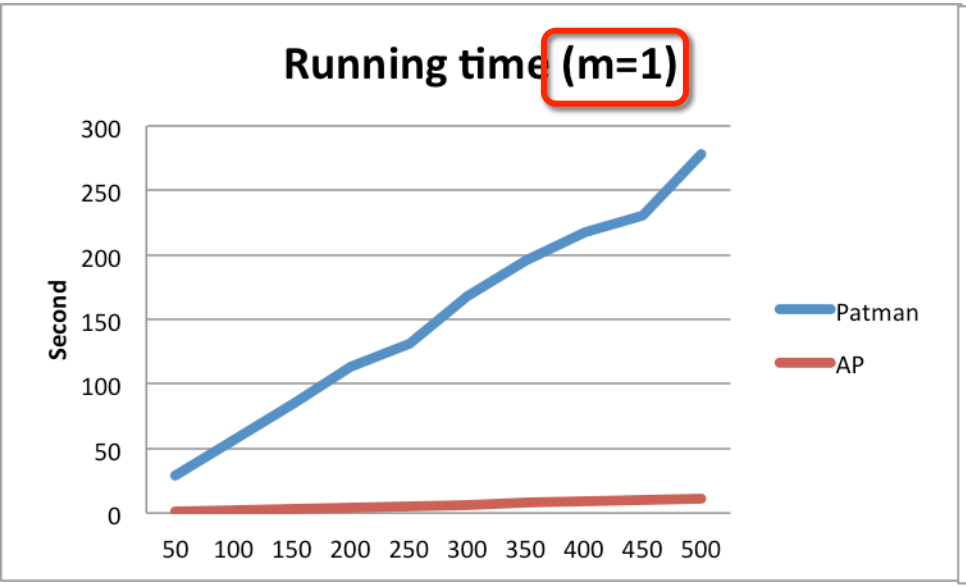


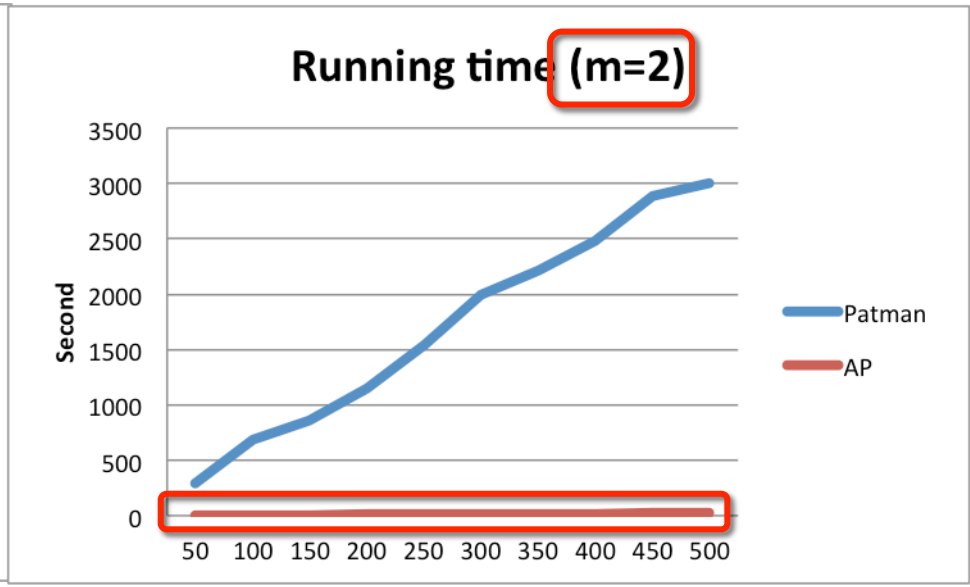
Sequence “ATGCATGCATGCATCATGCATGCAA”, K = 25, m = 3

Performance Evaluation

- Both AP and PatMaN time increase linearly as input size increases
- PatMaN increases much more severely
- Different mismatch distances: similar trends

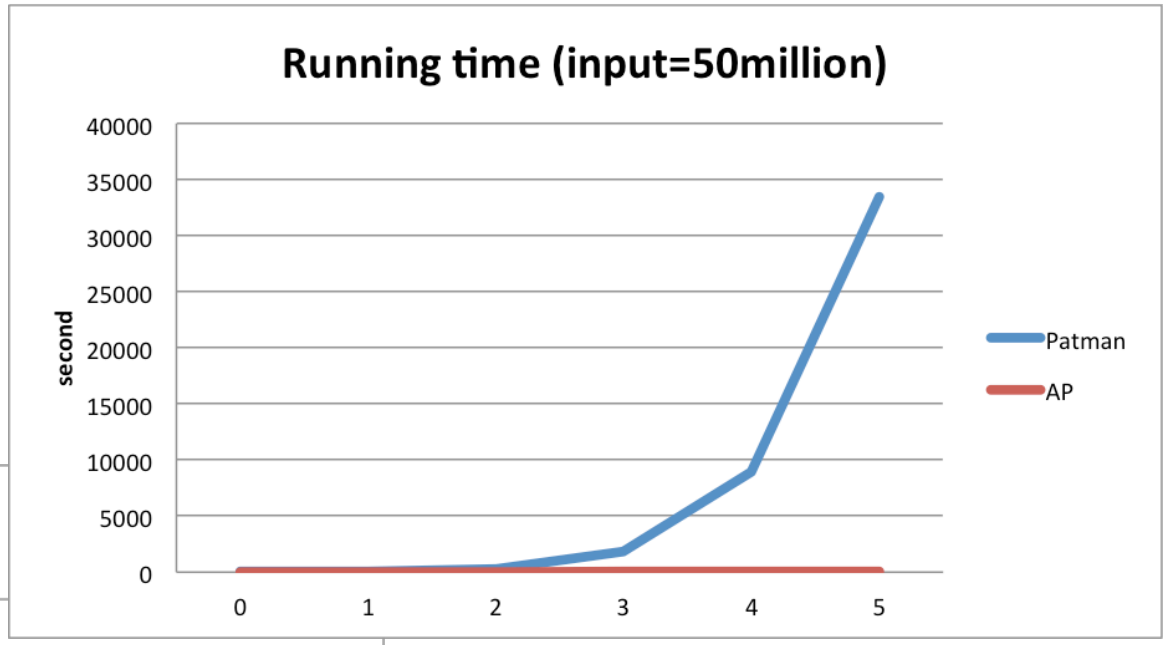


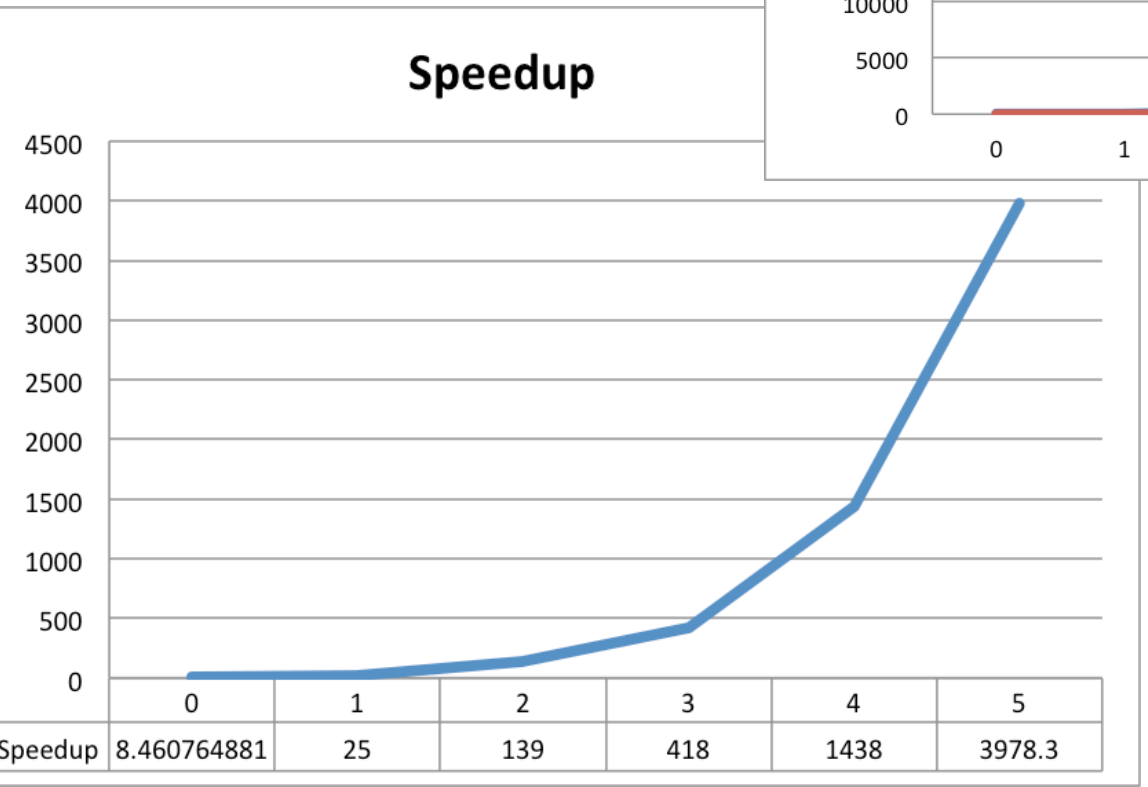




Performance Evaluation

- PatMaN increases exponentially
- AP increases linearly





	0	1	2	3	4	5
Speedup	8.460764881	25	139	418	1438	3978.3

- Speedups increases exponentially

  8.5x ~ 3980x

Summary & Future Work

- Summary
  - Presented an AP-accelerated method for String Kernel
  - Showed various automata designs for mapping functions
  - Achieved 8.5x to 3980x speedup

- Future Work
  - Evaluate accuracy
  - Solve larger data sets
  - Compare with other CPU methods (e.g. GPU, FPGA)

Thanks!

Questions?

http://www.cap.virginia.edu