Parallel regular expression matching in Julia

Zachary Deng

18.337

May 10, 2023

Overview

- Regular expressions are a convenient way to specify pattern-matching tasks:
  - Lexical analysis of programming languages
  - Monitoring log files for intrusions
  - Validating user input

Overview

- Regular expressions are a convenient way to specify pattern-matching tasks:
  - Lexical analysis of programming languages
  - Monitoring log files for intrusions
  - Validating user input
- Most regex implementations are single-threaded, including Julia's (a wrapper around PCRE library)

Overview

- Regular expressions are a convenient way to specify pattern-matching tasks:
  - Lexical analysis of programming languages
  - Monitoring log files for intrusions
  - Validating user input
- Most regex implementations are single-threaded, including Julia's (a wrapper around PCRE library)

Objective 1

Implement efficient multi-threaded regex engine

Overview

- Regular expressions are a convenient way to specify pattern-matching tasks:
  - Lexical analysis of programming languages
  - Monitoring log files for intrusions
  - Validating user input
- Most regex implementations are single-threaded, including Julia's (a wrapper around PCRE library)

Objective 1

Implement efficient multi-threaded regex engine

Objective 2

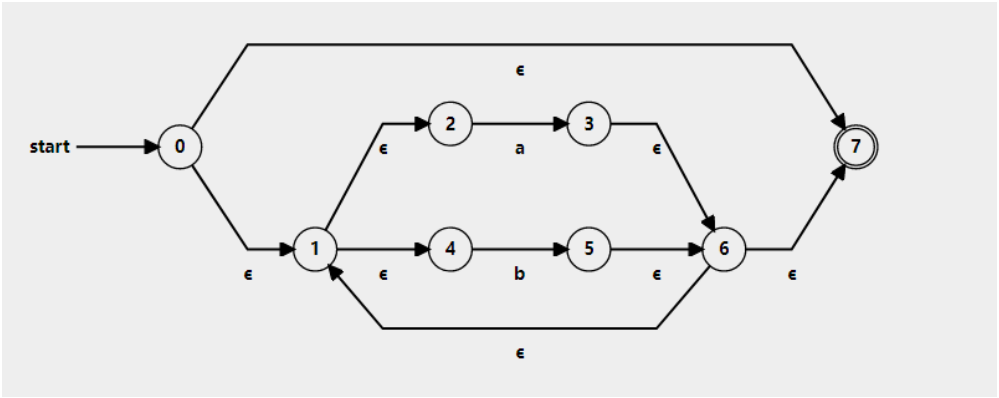
Characterize performance of various algorithms

Regular expressions

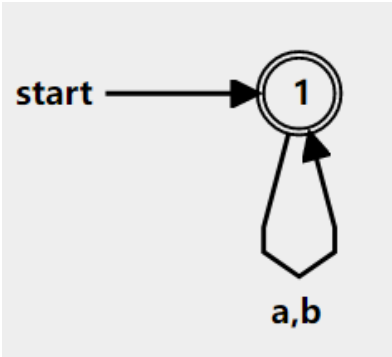
- Simplest class of regular expressions defined by following recursion:
  - $S ::=$ '(' S ')' (match S)
  - $S ::= S_1\ S_2$ (match S_1 then S_2)
  - $S ::= S_1$ '|' S_2 (match S_1 or S_2)
  - $S ::= S$ '*' (match S any number of times)
  - $S ::=$ 'a' (match the character a)
  - ...

Finite automata

- Regular expression
  (a | b) *
- To nondeterministic finite automata (NFA):

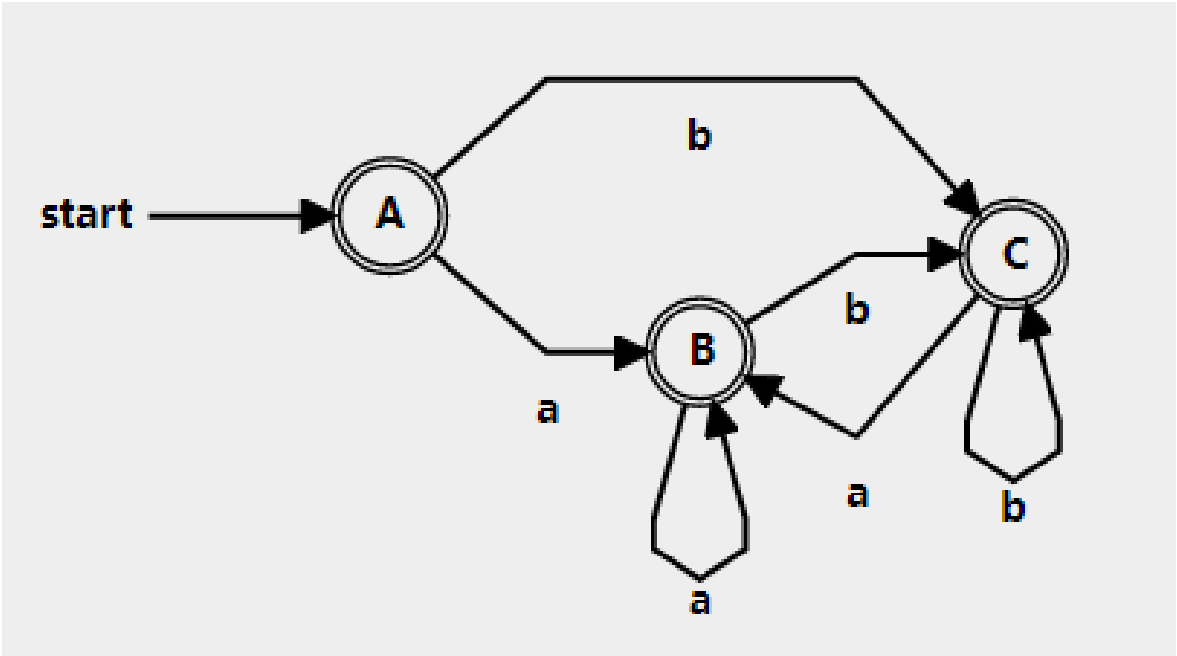


- To deterministic finite automata (DFA):



Serial matching on a DFA

- For each character in a string, take the corresponding edge of the DFA



- "aabbba" will end at state B so it matches the regex

How to parallelize?

- Speculative execution [1]
  - Split string into p chunks, one for each processor
  - For each processor, execute its chunk on the DFA starting from every possible state
  - After all processors done, combine the results for each chunk sequentially
  - Not work efficient (with p processors, string of length n, and DFA with m states, requires $O(\frac{nm}{p} + p)$ time)
- Improved version (PaREM) [2]
  - Similar idea as above
  - For each processor, eliminate some possible starting states if they can't be reached at the start of the chunk

How to parallelize?

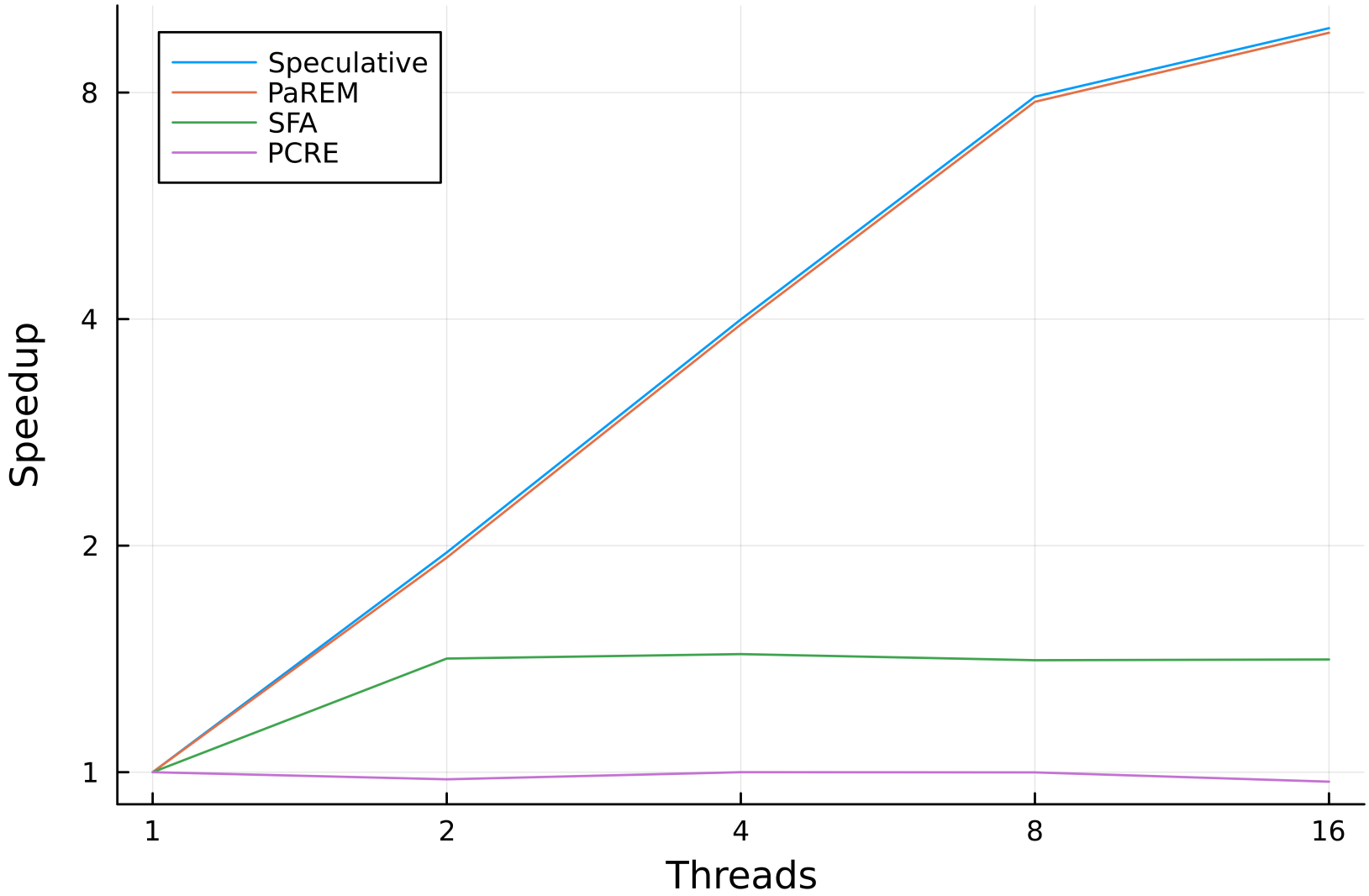
- Simultaneous finite automata (SFA) [3]
  - Still split string into p chunks, one for each processor
  - Previous methods compute a map from starting state to ending state by executing the DFA m times
  - Instead, precompute a new automaton (SFA) where the states correspond to these maps
  - Each SFA state tells us where *every* DFA starting state will end
  - Execute once on each chunk and compose the maps for each processor
  - No more dependence on m: $O(\frac{n}{p} + p)$ time

Implementation

- Simple regex parser using ParserCombinator.jl
- Regex to NFA using McNaughton–Yamada–Thompson algorithm
- NFA to DFA using subset construction
- DFA to SFA using algorithm similar to subset construction
- Automata match binary data and are represented as adjacency lists

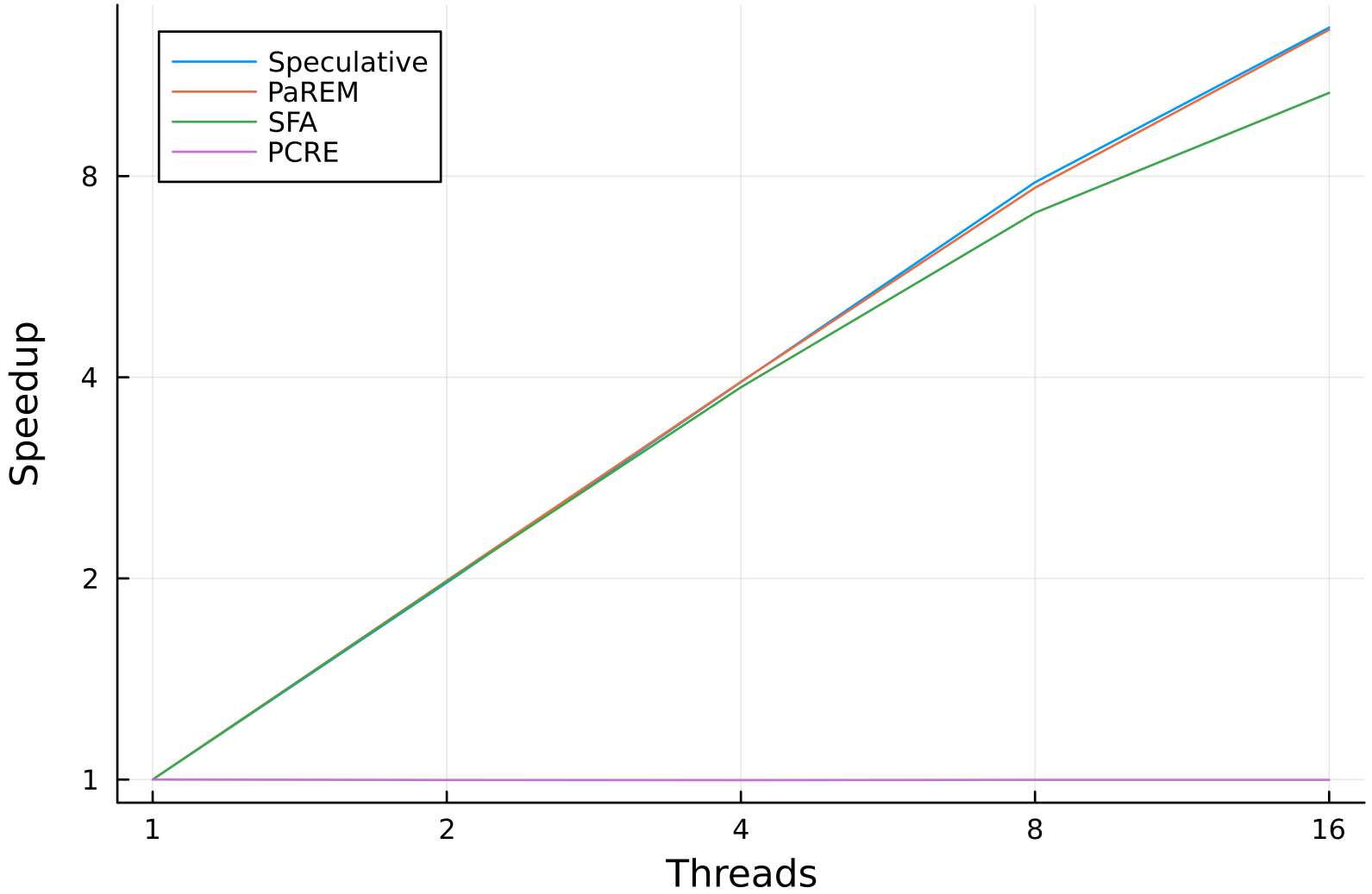
Results

- Benchmark regexes drawn from network intrusion detection system [4]
- Geometric mean speedup on 1MB input string



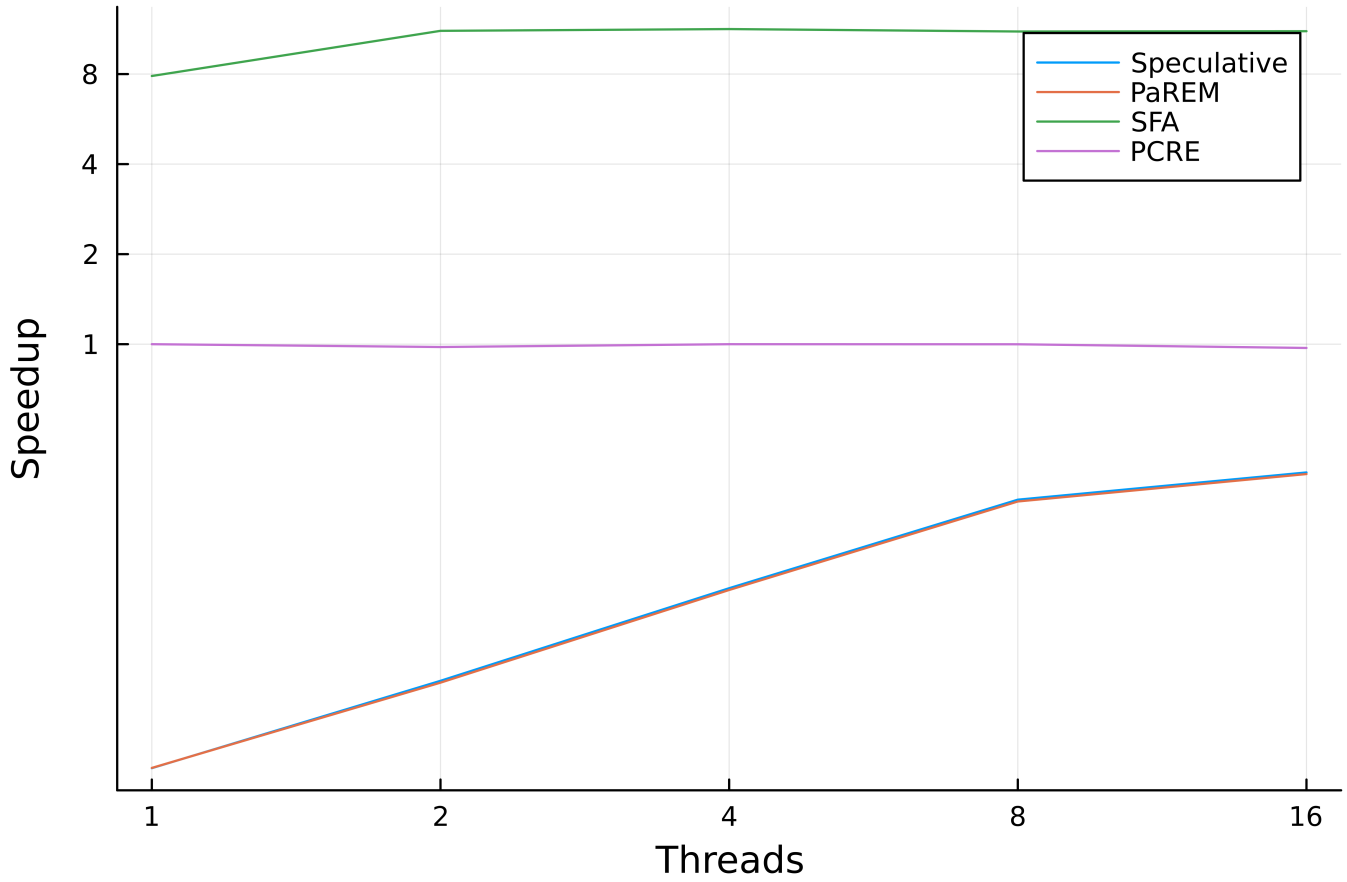
Results

- Benchmark regexes drawn from network intrusion detection system [4]
- Geometric mean speedup on 10MB input string



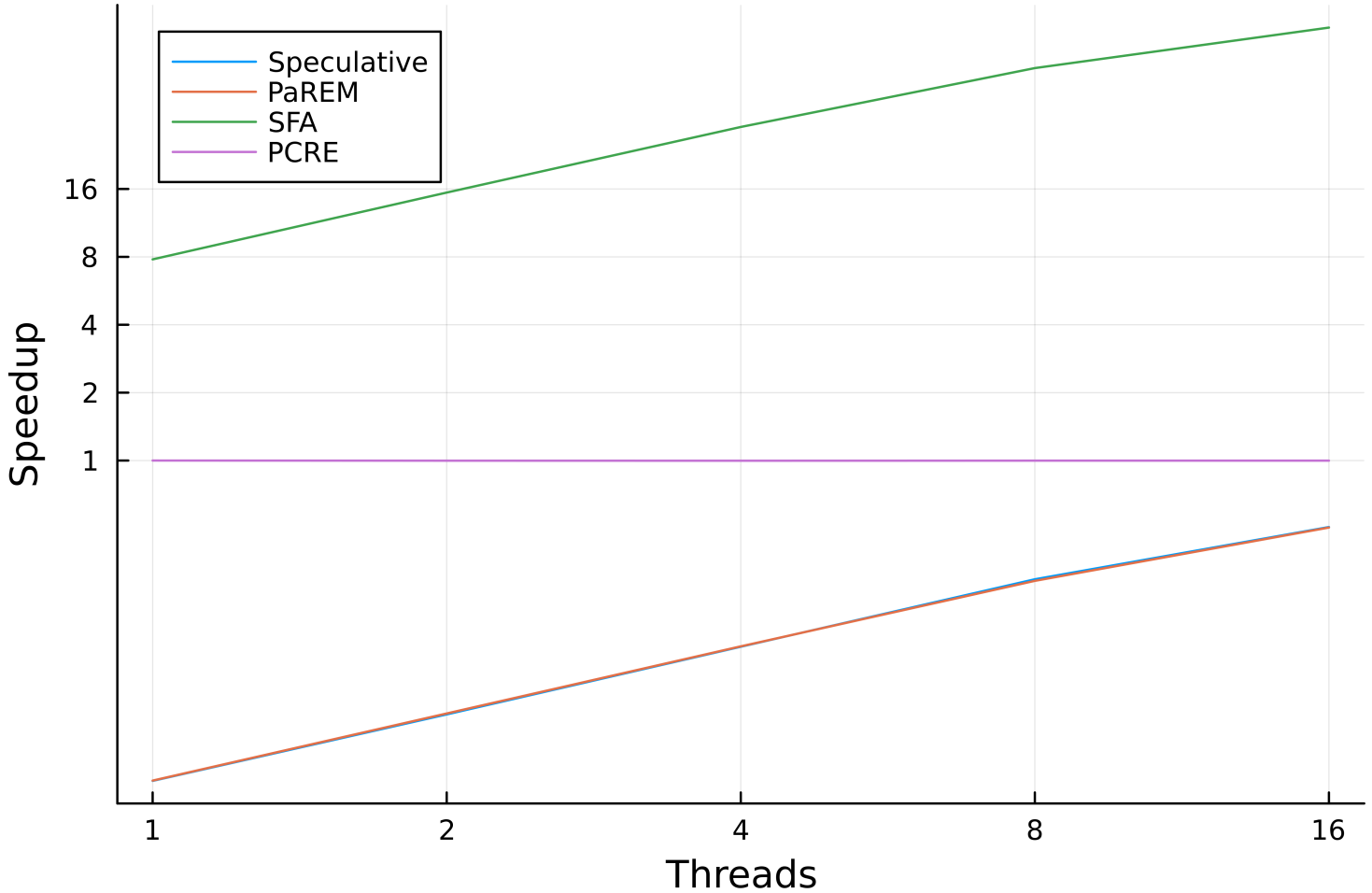
Results

- Benchmark regexes drawn from network intrusion detection system [4]
- Speedup compared to PCRE on 1MB input string



Results

- Benchmark regexes drawn from network intrusion detection system [4]
- Speedup compared to PCRE on 10MB input string



References

[1] Jan Holub and Stanislav Štekr. "On Parallel Implementations of Deterministic Finite Automata". In: *Implementation and Application of Automata*. Ed. by Sebastian Maneth. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 54–64. ISBN: 978-3-642-02979-0. DOI: 10.1007/978-3-642-02979-0_9.

[2] Suejb Memeti and Sabri Pllana. "PaREM: A Novel Approach for Parallel Regular Expression Matching". In: *2014 IEEE 17th International Conference on Computational Science and Engineering*. Dec. 2014, pp. 690–697. DOI: 10.1109/CSE.2014.146. arXiv: 1412.1741 [cs]. URL: http://arxiv.org/abs/1412.1741 (visited on 03/24/2023).

[3] Ryoma Sinya, Kiminori Matsuzaki, and Masataka Sassa. "Simultaneous Finite Automata: An Efficient Data-Parallel Model for Regular Expression Matching". In: *2013 42nd International Conference on Parallel Processing*. 2013 42nd International Conference on Parallel Processing. Oct. 2013, pp. 220–229. DOI: 10.1109/ICPP.2013.31.

[4] Jack Wadden et al. "ANMLzoo: A Benchmark Suite for Exploring Bottlenecks in Automata Processing Engines and Architectures". In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. 2016 IEEE International Symposium on Workload Characterization (IISWC). Sept. 2016, pp. 1–12. DOI: 10.1109/IISWC.2016.7581271.

Q&A

Thank you!