On the Benefit of Merging Suffix Array Intervals for Parallel Pattern Matching

Johannes Fischer and Dominik Köppl and *Florian Kurpicz*

February 16, 2016

71. Workshop über Algorithmen und Komplexität

Notations

- Σ is the alphabet with $|\Sigma| = \sigma$
- $\$ \notin \Sigma$ and $\forall \alpha \in \Sigma : \$ <_{\text{lex}} \alpha$
- $T \in \Sigma^\star \cup \{\$\}$ and $P \in \Sigma^\star$
- $|T| = n$ and $|P| = m$
- p is the number of processors

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

$T = \texttt{banana\$}$

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

$T = \texttt{banana\$}$

$P_1 = \texttt{b}$

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

$T = \texttt{banana\$}$

$P_1 = \texttt{b}$

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

$T = \texttt{banana\$}$

$P_1 = \texttt{b}$ and $P_2 = \texttt{a}$

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

$T = \texttt{banana\$}$

$P_1 = \texttt{b}$ and $P_2 = \texttt{a}$

Pattern Matching

Given a text T of length n and a pattern P of length m, find all occurrences of P in T.

$T = \texttt{banana\$}$

$P_1 = \texttt{b}$ and $P_2 = \texttt{a}$

Sequential Times

Type	Query Time	Idea
exact	$\mathcal{O}(m)$	Suffix Tree
k-errors	$\mathcal{O}\left(m^k \sigma^k \max(k, \lg\lg n) + occ\right)$	[Lam et al., 2007]

Notations

Prefix and Suffix

$$P_i = T[1..i] \text{ is the } i\text{-th prefix of } T \text{ for all } i \in [1, n]$$
$$S_i = T[i..n] \text{ is the } i\text{-th suffix of } T \text{ for all } i \in [1, n]$$

Notations

Prefix and Suffix

$$P_i = T[1..i] \text{ is the } i\text{-th prefix of } T \text{ for all } i \in [1, n]$$
$$S_i = T[i..n] \text{ is the } i\text{-th suffix of } T \text{ for all } i \in [1, n]$$

$T = \texttt{banana\$}$

i	1	2	3	4	5	6	7
S_i	banana$	anana$	nana$	ana$	na$	a$	$

The Suffix Array

Suffix Array of T

The SA is a permutation of $[1, n]$ such that for all $i \in [1, n-1]$:

$$T[SA[i]..n] <_{\text{lex}} T[SA[i+1]..n]$$

The Suffix Array

Suffix Array of T

The SA is a permutation of $[1, n]$ such that for all $i \in [1, n-1]$:

$$T\left[SA\left[i\right]..n\right] <_{\text{lex}} T\left[SA\left[i+1\right]..n\right]$$

$T =$ `banana$`

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

The Suffix Array

Suffix Array of T

The SA is a permutation of $[1, n]$ such that for all $i \in [1, n-1]$:

$$T[SA[i]..n] <_{\text{lex}} T[SA[i+1]..n]$$

$T =$ `banana$`

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

Suffix Array Interval (SAI) of P

$$i \in \mathcal{I}(P) \iff T[SA[i]..SA[i] + |P| - 1] = P$$

The Suffix Array

Suffix Array of T

The SA is a permutation of $[1, n]$ such that for all $i \in [1, n-1]$:

$$T\left[SA\left[i\right]..n\right] <_{\text{lex}} T\left[SA\left[i+1\right]..n\right]$$

$T =$ `banana$`

$\mathcal{I}(\texttt{a}) = [2, 4]$

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

Suffix Array Interval (SAI) of P

$$i \in \mathcal{I}(P) \iff T\left[SA\left[i\right]..SA\left[i\right] + |P| - 1\right] = P$$

The Suffix Array

Suffix Array of T

The SA is a permutation of $[1, n]$ such that for all $i \in [1, n-1]$:

$$T[SA[i]..n] <_{\text{lex}} T[SA[i+1]..n]$$

$T = \texttt{banana\$}$

$\mathcal{I}(\texttt{a}) = [2, 4]$

$\mathcal{I}(\texttt{n}) = [6, 7]$

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
	\$	a	a	a	b	n	n
		\$	n	n	a	a	a
			a	a	n	\$	n
			\$	n	a		a
				a	n		\$
				\$	a		
					\$		

Suffix Array Interval (SAI) of P

$$i \in \mathcal{I}(P) \iff T[SA[i]..SA[i] + |P| - 1] = P$$

The Suffix Array

Suffix Array of T

The SA is a permutation of $[1, n]$ such that for all $i \in [1, n-1]$:

$$T[SA[i]..n] <_{\text{lex}} T[SA[i+1]..n]$$

$T = \texttt{banana\$}$

$$\mathcal{I}(\texttt{a}) = [2, 4]$$
$$\mathcal{I}(\texttt{n}) = [6, 7]$$
$$\mathcal{I}(\texttt{an}) = [3, 4]$$

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

Suffix Array Interval (SAI) of P

$$i \in \mathcal{I}(P) \iff T[SA[i]..SA[i] + |P| - 1] = P$$

The Inverse Suffix Array

Inverse Suffix Array of T

The SA^{-1} is a permutation of $[1, n]$ such that for all $i \in [1, n]$:

$$SA^{-1}[SA[i]] = i$$

The Inverse Suffix Array

Inverse Suffix Array of T

The SA^{-1} is a permutation of $[1, n]$ such that for all $i \in [1, n]$:

$$SA^{-1}[SA[i]] = i$$

$T = \texttt{banana\$}$

$\mathcal{I}(\texttt{a}) = [2, 4]$

$\mathcal{I}(\texttt{n}) = [6, 7]$

$\mathcal{I}(\texttt{an}) = [3, 4]$

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
$SA^{-1}[i]$	5	4	7	3	6	2	1
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

The Inverse Suffix Array

Inverse Suffix Array of T

The SA^{-1} is a permutation of $[1, n]$ such that for all $i \in [1, n]$:

$$SA^{-1}[SA[i]] = i$$

$T = \texttt{banana\$}$

$\mathcal{I}(\texttt{a}) = [2, 4]$

$\mathcal{I}(\texttt{n}) = [6, 7]$

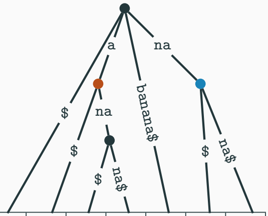
$\mathcal{I}(\texttt{an}) = [3, 4]$

	1	2	3	4	5	6	7
$SA[i]$	7	6	4	2	1	5	3
$\Psi^1[i]$	-	1	6	7	4	2	3
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

Find the rest of the suffix

$$\Psi^k[i] = SA^{-1}[SA[i] + k]$$

The Suffix Tree



$SA[i]$	7	6	4	2	1	5	3
	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		

Tree above the Suffix Array

- Nodes cover *relevant SAIs*

$$\mathcal{I}(\texttt{a}) = [2,4]$$
$$\mathcal{I}(\texttt{n}) = [6,7]$$

The Idea

Find occurrences of subpatterns and merge suffix array intervals

Suffix Array Interval Merging

The Idea

Find occurrences of subpatterns and merge suffix array intervals

The Problem

How to find the interval gained by merging two suffix array intervals?

Suffix Array Interval Merging

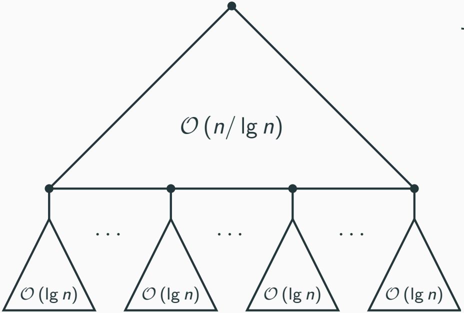
The Idea

Find occurrences of subpatterns and merge suffix array intervals

The Problem

How to find the interval gained by merging two suffix array intervals?

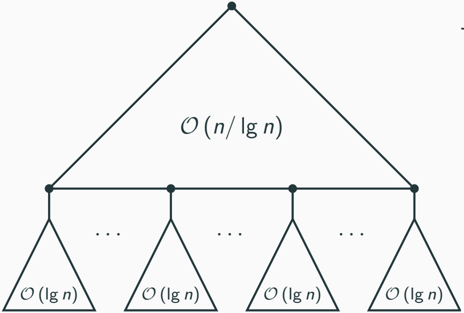
Paper	Running Time	Idea
[Huynh et al., 2006]	$\mathcal{O}(\lg n)$	Binary Search
[This talk]	$\mathcal{O}(\lg \lg n)$	Extending [Lam et al., 2007], Sampling Ψ in y-fast trie
	$\mathcal{O}\left(\lg_p \lg n\right)$	Parallel Binary Search



y-**Fast Trie** **[Willard, 1983]**

- Each leaf stores $\mathcal{O}(\lg n)$ elements in a binary search tree
- *x*-fast trie for $\mathcal{O}(n/\lg n)$ elements
- Prefixes of elements in $\mathcal{O}(\lg n)$ hash tables

Integer Dictionaries



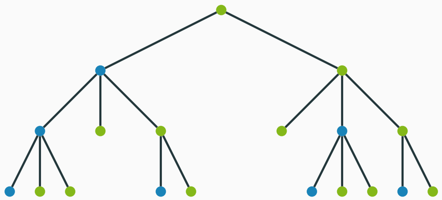
y-**Fast Trie** **[Willard, 1983]**

- Each leaf stores $\mathcal{O}(\lg n)$ elements in a binary search tree
- x-fast trie for $\mathcal{O}(n/\lg n)$ elements
- Prefixes of elements in $\mathcal{O}(\lg n)$ hash tables

FIND, PREDECESSOR and SUCCESSOR in $\mathcal{O}(\lg \lg n)\ldots$

- ... expected time **or**
- ... deterministic time with $\mathcal{O}(n \lg \lg n)$ construction time.

Heavy Path Decomposition

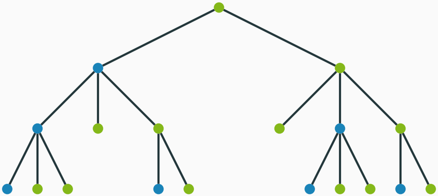


Nodes are

Heavy if they are in the largest subtree

Light otherwise (or if they are the root)

Heavy Path Decomposition



Nodes are

Heavy if they are in the largest subtree

Light otherwise (or if they are the root)

Sample Ψ for each light node

Sampling Ψ – Light nodes

Given two *SAIs* $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta)$

- Find all $i \in \mathcal{I}(\alpha) : \Psi^{|\alpha|}[i] \in \mathcal{I}(\beta)$
- $\Psi^{|\alpha|}[i]$ is monotonically increasing for all $i \in \mathcal{I}(\alpha)$

Given two *SAIs* $\mathcal{I}(\alpha)$ **and** $\mathcal{I}(\beta)$

- Find all $i \in \mathcal{I}(\alpha) : \Psi^{|\alpha|}[i] \in \mathcal{I}(\beta)$
- $\Psi^{|\alpha|}[i]$ is monotonically increasing for all $i \in \mathcal{I}(\alpha)$

Sampling for Light Nodes v **of** $\mathcal{I}(\alpha)$ **in** y**-fast trie**

$$\Gamma(v) := \left\{ \left(\Psi^{|\alpha|}[i], i\right) : i \equiv 1 \pmod{\lg^2 n} \wedge i \in \mathcal{I}(\alpha) \right\}$$

$\Psi^{\lvert\alpha\rvert}$			$\dots$				$\dots$				$\dots$			

Sampling Ψ – Light nodes

Given two *SAIs* $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta)$

- Find all $i \in \mathcal{I}(\alpha) : \Psi^{|\alpha|}[i] \in \mathcal{I}(\beta)$
- $\Psi^{|\alpha|}[i]$ is monotonically increasing for all $i \in \mathcal{I}(\alpha)$

Sampling for Light Nodes v of $\mathcal{I}(\alpha)$ in y-fast trie

$$\Gamma(v) := \left\{ \left(\Psi^{|\alpha|}[i], i\right) : i \equiv 1 \pmod{\lg^2 n} \wedge i \in \mathcal{I}(\alpha) \right\}$$

$\Psi^{|\alpha|}$

Sampling Ψ – Light nodes

Given two *SAIs* $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta)$

- Find all $i \in \mathcal{I}(\alpha) : \Psi^{|\alpha|}[i] \in \mathcal{I}(\beta)$
- $\Psi^{|\alpha|}[i]$ is monotonically increasing for all $i \in \mathcal{I}(\alpha)$

Sampling for Light Nodes v of $\mathcal{I}(\alpha)$ in y-fast trie

$$\Gamma(v) := \left\{ \left(\Psi^{|\alpha|}[i], i\right) : i \equiv 1 \pmod{\lg^2 n} \wedge i \in \mathcal{I}(\alpha) \right\}$$

$\Psi^{|\alpha|}$ $\square$ $\cdots$ $\square$ $\cdots$ $\square$ $\cdots$ $\square$

Merging *SAI*s – Light nodes

Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements

$\Psi^{|\alpha|}$

Merging *SA*s – Light nodes

Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$

$\Psi^{|\alpha|}$

Merging *SAI*s – Light nodes

Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$
- Extend j_l, j_r using binary search on $< \lg^2 n$ elements

$\Psi^{|\alpha|}$

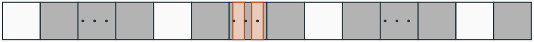
Merging *SAI*s – Light nodes

Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$
- Extend j_l, j_r using binary search on $< \lg^2 n$ elements
- If either j_l or j_r does not exist there is no j such that

$$b_\beta \leq \Psi^{|\alpha|}[j] \leq e_\beta$$

$\Psi^{|\alpha|}$



Merging *SAI*s – Light nodes

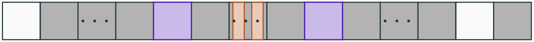
Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$
- Extend j_l, j_r using binary search on $< \lg^2 n$ elements
- If either j_l or j_r does not exist there is no j such that

$$b_\beta \leq \Psi^{|\alpha|}[j] \leq e_\beta$$

Find k_l, k_r : $\Psi^{|\alpha|}[k_l] \leq b_\beta$ **and** $e_\beta \leq \Psi^{|\alpha|}[k_r]$

$\Psi^{|\alpha|}$



Merging *SA*Is – Light nodes

Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

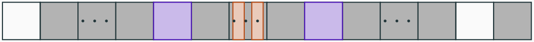
- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$
- Extend j_l, j_r using binary search on $< \lg^2 n$ elements
- If either j_l or j_r does not exist there is no j such that

$$b_\beta \leq \Psi^{|\alpha|}[j] \leq e_\beta$$

  Find k_l, k_r : $\Psi^{|\alpha|}[k_l] \leq b_\beta$ **and** $e_\beta \leq \Psi^{|\alpha|}[k_r]$

- Shrink k_l, k_r using binary search on $< \lg^2 n$ elements

$\Psi^{|\alpha|}$



Merging SAIs – Light nodes

Let v **be the light node of** $\mathcal{I}(\alpha)$ **and** $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$
- Extend j_l, j_r using binary search on $< \lg^2 n$ elements
- If either j_l or j_r does not exist there is no j such that

$$b_\beta \leq \Psi^{|\alpha|}[j] \leq e_\beta$$

- Shrink k_l, k_r using binary search on $< \lg^2 n$ elements

Similar idea for heavy nodes

Merging SAIs – Light nodes

Let v be the light node of $\mathcal{I}(\alpha)$ and $\mathcal{I}(\beta) = [b_\beta, e_\beta]$

- If $\Gamma(v) = \emptyset \rightarrow$ Binary search on $< \lg^2 n$ elements
- Find j_l, j_r : $b_\beta \leq \Psi^{|\alpha|}[j_l]$ **and** $\Psi^{|\alpha|}[j_r] \leq e_\beta$
- Extend j_l, j_r using binary search on $< \lg^2 n$ elements
- If either j_l or j_r does not exist there is no j such that

$$b_\beta \leq \Psi^{|\alpha|}[j] \leq e_\beta$$

- Shrink k_l, k_r using binary search on $< \lg^2 n$ elements

Similar idea for heavy nodes

Lemma

We can merge two SAIs in $\mathcal{O}(\lg \lg n)$ time.

Parallelize the Merging

What are we doing to merge two *SAI*s

Query y-fast tries **and** binary search

Parallelize the Merging

What are we doing to merge two *SAI*s

Query y-fast tries **and** binary search

Parallelize these queries

- Binary search requires $\mathcal{O}\left(\lg_p n\right)$ parallel time [Snir 1985]
- Binary search in the x-fast trie
- Static y-fast trie $\rightarrow$ arrays instead of binary search trees

Parallelize the Merging

What are we doing to merge two *SAI*s

Query y-fast tries **and** binary search

Parallelize these queries

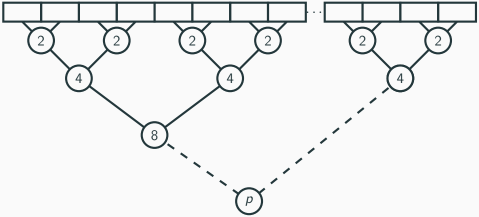
- Binary search requires $\mathcal{O}\left(\lg_p n\right)$ parallel time [Snir 1985]
- Binary search in the x-fast trie
- Static y-fast trie $\rightarrow$ arrays instead of binary search trees

Lemma

We can merge two SAIs in $\mathcal{O}\left(\lg_p \lg n\right)$ *parallel time.*

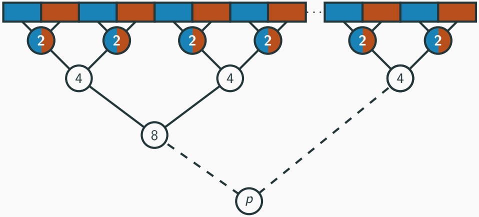
Parallel Exact Pattern Matching

- $P = P_1 P_2 \ldots P_p$ with $|P_i| = m/p$
- Compute $\mathcal{I}(P_i)$ in $\mathcal{O}(m/p)$ time
- Merge *SAI*s in $\mathcal{O}\left(\lg_p \lg n\right)$ time



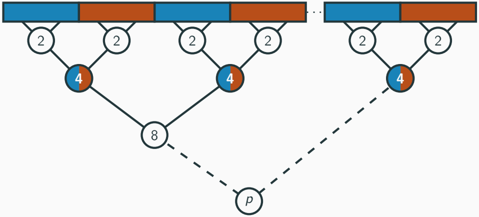
Parallel Exact Pattern Matching

- $P = P_1 P_2 \dots P_p$ with $|P_i| = m/p$
- Compute $\mathcal{I}(P_i)$ in $\mathcal{O}(m/p)$ time
- Merge SAIs in $\mathcal{O}\left(\lg_p \lg n\right)$ time



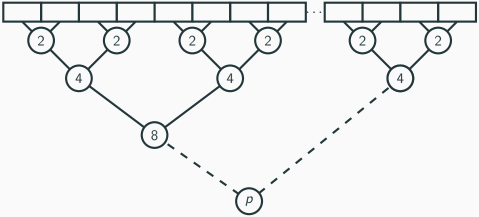
Parallel Exact Pattern Matching

- $P = P_1 P_2 \ldots P_p$ with $|P_i| = m/p$
- Compute $\mathcal{I}(P_i)$ in $\mathcal{O}(m/p)$ time
- Merge *SAI*s in $\mathcal{O}\left(\lg_p \lg n\right)$ time



Parallel Exact Pattern Matching

- $P = P_1 P_2 \ldots P_p$ with $|P_i| = m/p$
- Compute $\mathcal{I}(P_i)$ in $\mathcal{O}(m/p)$ time
- Merge *SAI*s in $\mathcal{O}\left(\lg_p \lg n\right)$ time



In the k-th Step

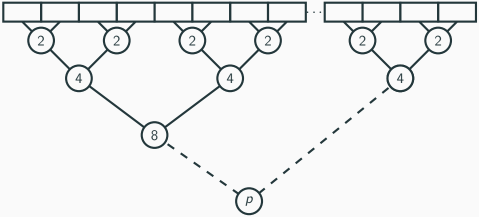
$p/2^k$ *SAI*s $\rightarrow 2^k$ processors

Number of Steps

There are $\lg p$ merge steps

Parallel Exact Pattern Matching

- $P = P_1 P_2 \dots P_p$ with $|P_i| = m/p$
- Compute $\mathcal{I}(P_i)$ in $\mathcal{O}(m/p)$ time
- Merge *SAI*s in $\mathcal{O}\left(\lg_p \lg n\right)$ time



In the k-th Step

$p/2^k$ *SAI*s $\to 2^k$ processors

Number of Steps

There are $\lg p$ merge steps

Theorem

Parallel exact pattern matching requires $\mathcal{O}\left(m/p + \lg\lg p \lg\lg n\right)$ *time.*

The k-Difference and k-Mismatch Problem

Given a text T of length n and a pattern P of length m ...

k-Difference Problem

... find all occurrences of P' in T such that P can be transformed to P' using $\leq k$ Insert, Change and Delete operations.

The k-Difference and k-Mismatch Problem

Given a text T of length n and a pattern P of length m . . .

k-Difference Problem

. . . find all occurrences of P' in T such that P can be transformed to P' using $\leq k$ INSERT, CHANGE and DELETE operations.

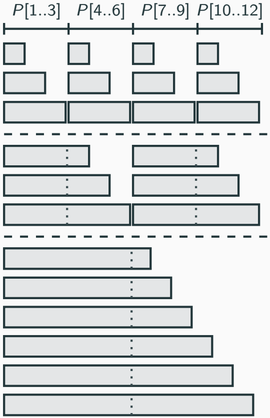
k-Mismatch Problem

. . . find all occurrences of P' in T such that P can be transformed to P' using $\leq k$ CHANGE operations.

Preprocessing

Compute *SAI*s of all prefixes and suffixes of P

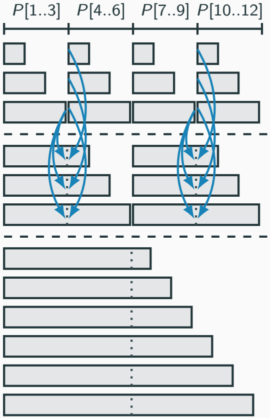
Preprocessing: $|P| = 12, p = 4$



Preprocessing

Compute *SAI*s of all prefixes and suffixes of P

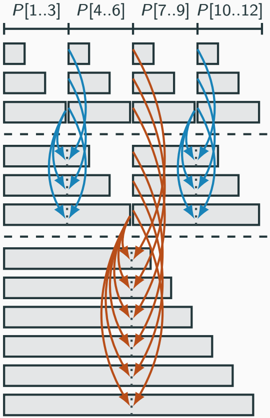
Preprocessing: $|P| = 12, p = 4$



Preprocessing

Compute SAIs of all prefixes and suffixes of P

Preprocessing: $|P| = 12, p = 4$



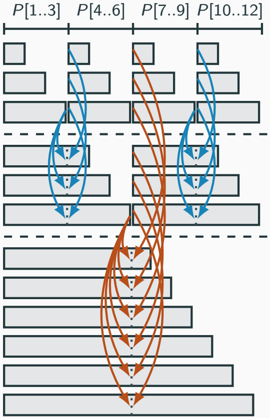
Preprocessing

Compute *SAI*s of all prefixes and suffixes of P

Preprocessing: $|P| = 12, p = 4$

In the k-th Step

- $p/2^k$ left *SAI*s
- $2^k m/p$ right *SAI*s



Preprocessing

Compute *SAI*s of all prefixes and suffixes of P

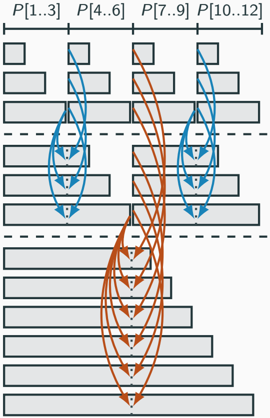
Preprocessing: $|P| = 12, p = 4$

In the k-th Step

- $p/2^k$ left *SAI*s
- $2^k m/p$ right *SAI*s

Cost of Merging

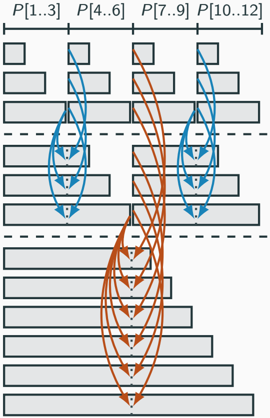
- There are $\lg n$ merge steps
- Merging in $\mathcal{O}\left(\lg_p \lg n\right)$ time



Preprocessing

Compute *SAI*s of all prefixes and suffixes of P

Preprocessing: $|P| = 12, p = 4$



In the k-th Step

- $p/2^k$ left *SAI*s
- $2^k m/p$ right *SAI*s

Cost of Merging

- There are $\lg n$ merge steps
- Merging in $\mathcal{O}\left(\lg_p \lg n\right)$ time

Lemma

The preprocessing requires $\mathcal{O}\left(m/p \lg p \lg \lg n\right)$ time.

Solving the 1-Difference and 1-Mismatch Problem

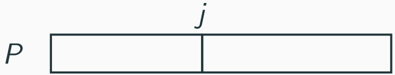
Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P\,[1..i])$ and $\mathcal{I}(P\,[i..n])$ are known
- What is an error at position j

P

Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P[1..i])$ and $\mathcal{I}(P[i..n])$ are known
- What is an error at position j

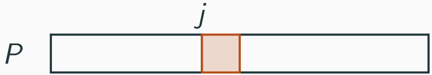


Solving the 1-Difference and 1-Mismatch Problem

Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P[1..i])$ and $\mathcal{I}(P[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j..n])$



Solving the 1-Difference and 1-Mismatch Problem

Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P[1..i])$ and $\mathcal{I}(P[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j..n])$

Solving the 1-Difference and 1-Mismatch Problem

Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P\,[1..i])$ and $\mathcal{I}(P\,[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P\,[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P\,[j..n])$

Change $\mathcal{I}(P\,[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P\,[j+1..n])$

Solving the 1-Difference and 1-Mismatch Problem

Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P[1..i])$ and $\mathcal{I}(P[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j..n])$

Change $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j+1..n])$

Solving the 1-Difference and 1-Mismatch Problem

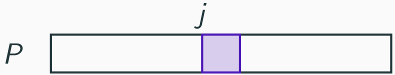
Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P\,[1..i])$ and $\mathcal{I}(P\,[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P\,[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P\,[j..n])$

Change $\mathcal{I}(P\,[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P\,[j+1..n])$

Delete $\mathcal{I}(P\,[1..j-1]) \otimes \mathcal{I}(P\,[j+1..n])$



Solving the 1-Difference and 1-Mismatch Problem

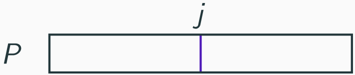
Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P[1..i])$ and $\mathcal{I}(P[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j..n])$

Change $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j+1..n])$

Delete $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(P[j+1..n])$



Solving the 1-Difference and 1-Mismatch Problem

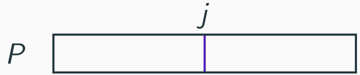
Introducing the Error (Insert, Change or Delete)

- $\mathcal{I}(P[1..i])$ and $\mathcal{I}(P[i..n])$ are known
- What is an error at position j

Insert $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j..n])$

Change $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(\alpha) \otimes \mathcal{I}(P[j+1..n])$

Delete $\mathcal{I}(P[1..j-1]) \otimes \mathcal{I}(P[j+1..n])$



Theorem

Approximate parallel pattern matching with ≤ 1 error can be solved in $\mathcal{O}(\sigma m/p \cdot \lg\lg n + occ)$ time.

Solving the k-Difference and k-Mismatch Problem

Quite similar to $k = 1$

- The same preprocessing
- Introduce $\leq k$ errors by merging SAIs
- Use configurations of positions and parallelize those

Solving the k-Difference and k-Mismatch Problem

Quite similar to $k = 1$

- The same preprocessing
- Introduce $\leq k$ errors by merging SAIs
- Use configurations of positions and parallelize those

Theorem

Approximate parallel pattern matching with $\leq k$ errors can be solved in $\mathcal{O}\left(\sigma^k m^k / p \cdot \lg \lg n + occ\right)$ time.

Problem – Report Occurrence Multiple Times

The Problem: $T = \texttt{aaa\$}$ **and** $P = \texttt{aba}$ **and one error**

Change $P' = \texttt{aaa}$

Delete $P'' = \texttt{aa}$

Both P' and P'' occur at position 1 in T

Problem – Report Occurrence Multiple Times

The Problem: $T = \texttt{aaa\$}$ **and** $P = \texttt{aba}$ **and one error**

Change $P' = \texttt{aaa}$

Delete $P'' = \texttt{aa}$

Both P' and P'' occur at position 1 in T

How do we get $\mathcal{O}(occ)$ reporting time?

Problem – Report Occurrence Multiple Times

The Problem: $T = \texttt{aaa\$}$ **and** $P = \texttt{aba}$ **and one error**

Change $P' = \texttt{aaa}$

Delete $P'' = \texttt{aa}$

Both P' and P'' occur at position 1 in T

How do we get $\mathcal{O}(occ)$ reporting time?

The Solution

- Report only if found with smallest distance [Huynh et al., 2006]
- Can be parallelized

Conclusion

Things we did

- Presented efficient parallel algorithm for merging SAIs
- Parallelized pattern matching (exact and approximative)

Conclusion

Things we did

- Presented efficient parallel algorithm for merging *SAI*s
- Parallelized pattern matching (exact and approximative)

What's still left

- Has this approach practical use
- Work is not good

Conclusion

Things we did

- Presented efficient parallel algorithm for merging SAIs
- Parallelized pattern matching (exact and approximative)

What's still left

- Has this approach practical use
- Work is not good

Thank You