ERA revisited: Theoretical and Experimental evaluation

Matevž Jekovec, Andrej Brodnik

University of Ljubljana
Faculty of Computer and Information Science

KAUST, March 1-5, 2014
Text indexing problem

Problem statement
Given unstructured input string $S$ consisting of $N$ characters from alphabet $\Sigma$ of size $\sigma$ build an index such that for the pattern $P$ we:
- determine whether $P$ occurs in $S$ in time $O(P)$,
- find all occurrences of $P$ in $S$ in time $O(P + \text{occ})$,
- find the longest common prefix (LCP) of $P$ and any suffix of $S$ in time $O(\text{LCP}(P, S))$.

Solution
Suffix tree and suffix array with LCP information are fundamental data structures for indexing unstructured text.
Problem statement

Given unstructured input string $S$ consisting of $N$ characters from alphabet $\Sigma$ of size $\sigma$ build an index such that for the pattern $P$ we:

- determine whether $P$ occurs in $S$ in time $O(P)$,
- find all occurrences of $P$ in $S$ in time $O(P + \text{occ})$,
- find the longest common prefix (LCP) of $P$ and any suffix of $S$ in time $O(LCP(P, S))$. 

Solution

Suffix tree and suffix array (SA) with LCP information are fundamental data structures for indexing unstructured text.
Problem statement

Given unstructured input string $S$ consisting of $N$ characters from alphabet $\Sigma$ of size $\sigma$ build an index such that for the pattern $P$ we:

- determine whether $P$ occurs in $S$ in time $O(P)$,
- find all occurrences of $P$ in $S$ in time $O(P + \text{occ})$,
- find the longest common prefix (LCP) of $P$ and any suffix of $S$ in time $O(LCP(P, S))$.

Solution

*Suffix tree* and *suffix array* (SA) with LCP information are fundamental data structures for indexing unstructured text.
Suffix tree — Example

T = ABRAKADABRA$
## Suffix tree construction algorithms

### Theoretical:

<table>
<thead>
<tr>
<th>Work w.c.</th>
<th>W ('73), McC ('78)</th>
<th>U ('95)</th>
<th>F-C et al. ('00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N lg N)</td>
</tr>
<tr>
<td>I/O efficiency</td>
<td>String</td>
<td>String</td>
<td>Result+String</td>
</tr>
<tr>
<td>Unbounded Σ</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel</td>
<td>No</td>
<td>No</td>
<td>PDAM</td>
</tr>
</tbody>
</table>

1 Bedathur and Haritsa (2004)
### Suffix tree construction algorithms

#### Theoretical:

<table>
<thead>
<tr>
<th>Work w.c.</th>
<th>W ('73), McC ('78)</th>
<th>U ('95)</th>
<th>F-C et al. ('00)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( O(N) )</td>
<td>( O(N) )</td>
<td>( O(N \lg N) )</td>
</tr>
<tr>
<td>Online</td>
<td>No</td>
<td>Yes</td>
<td>Yes&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>I/O efficiency</td>
<td>String</td>
<td>String</td>
<td>Result+String</td>
</tr>
<tr>
<td>Unbounded ( \Sigma )</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel</td>
<td>No</td>
<td>No</td>
<td>PDAM</td>
</tr>
</tbody>
</table>

#### Practical:

<table>
<thead>
<tr>
<th>Work w.c.</th>
<th>Semi-disk-based</th>
<th>Out-of-core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TDD ('04)</td>
<td>WF ('09)</td>
</tr>
<tr>
<td></td>
<td>TRLS. ('07)</td>
<td>ERA ('11)</td>
</tr>
<tr>
<td></td>
<td>( B^2 \text{ST} ('09) )</td>
<td>PCF ('13)</td>
</tr>
<tr>
<td>I/O eff.</td>
<td>( O(N^2) )</td>
<td>( O(N^2) )</td>
</tr>
<tr>
<td></td>
<td>R.</td>
<td>R. + S.</td>
</tr>
<tr>
<td>Unbnd. ( \Sigma )</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Parallel</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<sup>1</sup>Bedathur and Haritsa (2004)
Suffix tree construction algorithms

- **Theoretical:**
  - W ('73), McC ('78)
  - U ('95)
  - F-C et al. ('00)

<table>
<thead>
<tr>
<th></th>
<th>W ('73), McC ('78)</th>
<th>U ('95)</th>
<th>F-C et al. ('00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work w.c.</td>
<td>(O(N))</td>
<td>(O(N))</td>
<td>(O(N \lg N))</td>
</tr>
<tr>
<td>Online</td>
<td>No</td>
<td>Yes</td>
<td>Yes(^1)</td>
</tr>
<tr>
<td>I/O efficiency</td>
<td>String</td>
<td>String</td>
<td>Result+String</td>
</tr>
<tr>
<td>Unbounded (\Sigma)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
<td>PDAM</td>
</tr>
</tbody>
</table>

- **Practical:**

<table>
<thead>
<tr>
<th></th>
<th>Semi-disk-based</th>
<th>Out-of-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work w.c.</td>
<td>TDD ('04) (O(N^2))</td>
<td>WF ('09) (O(N^2))</td>
</tr>
<tr>
<td>I/O eff.</td>
<td>TRLS. ('07) (O(N^2))</td>
<td>ERA ('11) (O(N^2))</td>
</tr>
<tr>
<td>Unbnd. (\Sigma)</td>
<td>B(^2)ST ('09) (O(N^2))</td>
<td>PCF ('13) (O(\sqrt{pN}))</td>
</tr>
<tr>
<td>Parallel</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\(^1\) Bedathur and Haritsa (2004)

Huge gap between the theoretical and practical results!
### Suffix tree construction lower bounds

**Sequential:**

<table>
<thead>
<tr>
<th></th>
<th>bounded $\Sigma$</th>
<th>unbounded $\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>$\Omega(\text{Sort}(N))$</td>
<td>$\Omega(\text{Sort}(N))$</td>
</tr>
<tr>
<td>**I/Os}^2</td>
<td>$\Omega(\text{Sort}(N))$</td>
<td>$\Omega(\text{Sort}(N))$</td>
</tr>
<tr>
<td>**Space}^3</td>
<td>$\Omega(N \lg \sigma)$ bits</td>
<td>$\Omega(N \lg \sigma)$ bits</td>
</tr>
</tbody>
</table>

2 EM model  
3 Uncompressed index in word RAM  
4 PEM model
Suffix tree construction lower bounds

Sequential:

<table>
<thead>
<tr>
<th></th>
<th>bounded $\Sigma$</th>
<th>unbounded $\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>$\Omega(\text{Sort}(N))$</td>
<td>$\Omega(\text{Sort}(N))$</td>
</tr>
<tr>
<td>**I/Os}^2</td>
<td>$\Omega(\text{Sort}(N))$</td>
<td>$\Omega(\text{Sort}(N))$</td>
</tr>
<tr>
<td>**Space}^3</td>
<td>$\Omega(N \lg \sigma)$ bits</td>
<td>$\Omega(N \lg \sigma)$ bits</td>
</tr>
</tbody>
</table>

Parallel on $p$ processing units:

<table>
<thead>
<tr>
<th></th>
<th>bounded $\Sigma$</th>
<th>unbounded $\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel time</strong></td>
<td>$\Omega(\text{Sort}_p(N))$</td>
<td>$\Omega(\text{Sort}_p(N))$</td>
</tr>
<tr>
<td>**Parallel I/Os}^4</td>
<td>$\Omega(\text{Sort}_p(N))$</td>
<td>$\Omega(\text{Sort}_p(N))$</td>
</tr>
<tr>
<td>**Space}^3</td>
<td>$\Omega(N \lg \sigma)$ bits</td>
<td>$\Omega(N \lg \sigma)$ bits</td>
</tr>
</tbody>
</table>

---

$^2$EM model  
$^3$Uncompressed index in word RAM  
$^4$PEM model
Suffix tree construction lower bounds

- **Sequential:**
  - **Time**
    - bounded $\Sigma$: $\Omega(N)$
    - unbounded $\Sigma$: $\Omega(N \log N)$
  - **I/Os**
    - bounded $\Sigma$: $\Omega\left(\frac{N}{B}\right)$
    - unbounded $\Sigma$: $\Omega\left(\frac{N}{B} \log\frac{M}{B} \frac{N}{B}\right)$
  - **Space**
    - bounded $\Sigma$: $\Omega(N \lg \sigma)$ bits
    - unbounded $\Sigma$: $\Omega(N \lg \sigma)$ bits

- **Parallel on $p$ processing units:**
  - **Parallel time**
    - bounded $\Sigma$: $\Omega\left(\frac{N}{p}\right)$
    - unbounded $\Sigma$: $\Omega\left(\frac{N}{p} \log N\right)$
  - **Parallel I/Os**
    - bounded $\Sigma$: $\Omega\left(\frac{N}{pB}\right)$
    - unbounded $\Sigma$: $\Omega\left(\frac{N}{pB} \log\frac{M}{B} \frac{N}{B}\right)$
  - **Space**
    - bounded $\Sigma$: $\Omega(N \lg \sigma)$ bits
    - unbounded $\Sigma$: $\Omega(N \lg \sigma)$ bits

---

2 EM model
3 Uncompressed index in word RAM
4 PEM model
Challenges

Theoretical algorithms: Lack of locality of reference.
Goal: Use scans only both for input text and the resulting suffix tree!
Counterintuitive: Input text is arbitrary, suffix tree is lexicographically ordered.
Challenges

Theoretical algorithms: Lack of locality of reference.
Challenges

Theoretical algorithms: Lack of locality of reference.
Goal: Use scans only both for input text and the resulting suffix tree!
Challenges

Theoretical algorithms: Lack of locality of reference.
Goal: Use scans only both for input text and the resulting suffix tree!
Counterintuitive: Input text is arbitrary, suffix tree is lexicographically ordered.
I/O efficient solution (eg. WF-ERA, $B^2$ST-PCF):

1. Scan part of the input text from the disk,
Challenges contd.

I/O efficient solution (eg. WF-ERA, B²ST-PCF):
1. Scan part of the input text from the disk,
2. do in-memory sorting (=random accesses in fast memory only!),

Challenges contd.

I/O efficient solution (eg. WF-ERA, B²ST-PCF):

1. Scan part of the input text from the disk,
2. do in-memory sorting (=random accesses in fast memory only!),
3. construct the corresponding part of the suffix tree,
I/O efficient solution (eg. WF-ERA, B²ST-PCF):

1. Scan part of the input text from the disk,
2. do in-memory sorting (random accesses in fast memory only!),
3. construct the corresponding part of the suffix tree,
4. glue parts together,
I/O efficient solution (eg. WF-ERA, B²ST-PCF):

1. Scan part of the input text from the disk,
2. do in-memory sorting (=random accesses in fast memory only!),
3. construct the corresponding part of the suffix tree,
4. glue parts together,
5. and contiguously write it to disk.
**ERA — Elastic Range**

The fastest practical, parallel suffix tree construction algorithm to date. Time complexity: $O(N^2)$ w.c. — for extremely skewed text! Yet, it's fast in practice: Constructs and stores the human genome's suffix tree in 20 minutes on 16-core desktop PC with HDD or 13 minutes with SSD!

---

5 Mansour, Allam, Skiadopoulos, Kalnis (2011)
ERA — Elastic Range\(^5\)

- The fastest practical, parallel suffix tree construction algorithm to date.

\(^5\)Mansour, Allam, Skiadopoulos, Kalnis (2011)
ERA — Elastic Range\textsuperscript{5}

- The fastest practical, parallel suffix tree construction algorithm to date.
- Time complexity: $O(N^2)$ w.c. — for extremely skewed text!

\textsuperscript{5}Mansour, Allam, Skiadopoulos, Kalnis (2011)
The fastest practical, parallel suffix tree construction algorithm to date.

Time complexity: $O(N^2)$ w.c. — for extremely skewed text!

Yet, it’s **fast** in practice: Constructs and stores the human genome’s suffix tree in 20 minutes on 16-core desktop PC with HDD or 13 minutes with SSD!

---

5Mansour, Allam, Skiadopoulos, Kalnis (2011)
ERA contd.

ERA constructs the suffix tree in two steps:

1. The vertical partitioning step determines 1) the suffix subtrees just fitting into the main memory $M$ and 2) constructs the suffix tree top.
2. The horizontal partitioning step builds the actual suffix subtrees.
ERA constructs the suffix tree in two steps:

1. The **vertical partitioning** step determines 1) the suffix subtrees just fitting into the main memory $M$ and 2) constructs the suffix tree top.
ERA constructs the suffix tree in two steps:

1. The **vertical partitioning** step determines 1) the suffix subtrees just fitting into the main memory $M$ and 2) constructs the suffix tree top.

2. The **horizontal partitioning** step builds the actual suffix subtrees.
Algorithm 1: ERA

**Input**: String $S$, Alphabet $\Sigma$, Processors $P$, Private cache size $M$

**Output**: Suffix tree $\mathcal{T}$

1. $\mathcal{T}_{top}, G \leftarrow \text{VerticalPartitioning}(S, \Sigma, M)$
2. $\mathcal{T} \leftarrow \mathcal{T}_{top}$
3. while $|G| > 0$ do
   4. for $p \in P$ do in parallel
   5. if $|G| > 0$ then
      6. $\pi \leftarrow G.pop()$
      7. $\mathcal{T}_{\pi} \leftarrow \text{HorizontalPartitioning}(S, \Sigma, \pi)$
      8. $\text{Link}(\mathcal{T}, \mathcal{T}_{\pi})$
   9. return $\mathcal{T}$
Define **S-prefix** $\pi$ as the prefix of the suffixes in the text.
Vertical partitioning

Define **S-prefix** $\pi$ as the prefix of the suffixes in the text. Idea: The S-prefix frequency $f_\pi$ equals $\#$ of leaves in the suffix subtree corresponding to $\pi$. Assume $2f_\pi$ is the w. c. subtree size.

---

$^6$Yue (1991)
Vertical partitioning

Define **S-prefix** \( \pi \) as the prefix of the suffixes in the text. 
Idea: The S-prefix frequency \( f_\pi \) equals \# of leaves in the suffix subtree corresponding to \( \pi \). Assume \( 2f_\pi \) is the w. c. subtree size. 
Vertical partitioning steps:

1. Scan the text and obtain the characters frequency \( f_\pi : \pi \in \Sigma \).
   - We counted all S-prefixes of length 1.

---

\(^6\) Yue (1991)
Vertical partitioning

Define **S-prefix** $\pi$ as the prefix of the suffixes in the text. Idea: The S-prefix frequency $f_\pi$ equals the number of leaves in the suffix subtree corresponding to $\pi$. Assume $2f_\pi$ is the worst-case subtree size. Vertical partitioning steps:

1. **Scan the text and obtain the characters frequency** $f_\pi : \pi \in \Sigma$.
   - We counted all S-prefixes of length 1.

2. **For each** $\pi : f_\pi > M$, expand S-prefix with the right character and count the frequency of obtained S-prefixes (now length 2).

---

Yue (1991)
Define **S-prefix** $\pi$ as the prefix of the suffixes in the text. Idea: The S-prefix frequency $f_\pi$ equals $\#$ of leaves in the suffix subtree corresponding to $\pi$. Assume $2f_\pi$ is the w. c. subtree size.

Vertical partitioning steps:

1. Scan the text and obtain the characters frequency $f_\pi : \pi \in \Sigma$.
   - We counted all S-prefixes of length 1.

2. For each $\pi : f_\pi > M$, expand S-prefix with the right character and count the frequency of obtained S-prefixes (now length 2).

3. Repeat step two for S-prefixes of length 3, 4..., until all $T_\pi$ just fit into the memory $M$.

---

6 Yue (1991)
Vertical partitioning

Define **S-prefix** $\pi$ as the prefix of the suffixes in the text. Idea: The S-prefix frequency $f_{\pi}$ equals $\#$ of leaves in the suffix subtree corresponding to $\pi$. Assume $2f_{\pi}$ is the w. c. subtree size.

Vertical partitioning steps:

1. Scan the text and obtain the characters frequency $f_{\pi} : \pi \in \Sigma$.
   - We counted all S-prefixes of length 1.

2. For each $\pi : f_{\pi} > M$, expand S-prefix with the right character and count the frequency of obtained S-prefixes (now length 2).

3. Repeat step two for S-prefixes of length 3, 4..., until all $T_{\pi}$ just fit into the memory $M$.

4. Extra: To optimally fill the main memory, combine the S-prefixes into *virtual groups* $G$, fitting into the main memory as tight as possible.
   - Use First-Fit Decreasing heuristic for bin packing problem\(^6\).

---

\(^6\) Yue (1991)
Vertical partitioning — Example

\[ \pi = ACC \]

Frequency \( f_{ACC} = 12 \)

TAACCCTA
ACCCTAAC
CCTAACCC
TAACCCTA
ACCCTAAC
CCTAACCC
TAACCCTA
ACCCTAAC
CCTAACCC
TAAC

\[ \begin{array}{c}
101:0 \\
2:1 \\
3:2 \\
4:1 \\
5:6 \\
101:0 \\
5:6 \\
101:0 \\
4:1 \\
11:6 \\
7:4 \\
5:1 \\
6:5
\end{array} \]

\[ \begin{array}{c}
TAAC \\
CCTAAC \\
TAAC \\
CCTAAC \\
TAAC \\
CCTAAC \\
TAAC \\
CCTAAC
\end{array} \]
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:

1. Locate and store all positions of S-prefixes for the virtual group. Each of located S-prefixes is to become a leaf in the working suffix subtree.
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:

1. Locate and store all positions of S-prefixes for the virtual group. Each of located S-prefixes is to become a leaf in the working suffix subtree.

2. Calculate the optimal buffer length range
   - Note: The name Elastic Range.
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:

1. Locate and store all positions of S-prefixes for the virtual group. Each of located S-prefixes is to become a leaf in the working suffix subtree.

2. Calculate the optimal buffer length `range`
   - Note: The name *Elastic Range*.

3. Read the next `range` characters for each S-prefix occurrence.
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:

1. Locate and store all positions of S-prefixes for the virtual group. Each of located S-prefixes is to become a leaf in the working suffix subtree.

2. Calculate the optimal buffer length \textit{range}.
   - Note: The name \textit{Elastic Range}.

3. Read the next \textit{range} characters for each S-prefix occurrence.

4. Do in-memory sorting of read text, remember branching information (=LCP) and the original position (=SA).
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:

1. Locate and store all positions of S-prefixes for the virtual group. Each of located S-prefixes is to become a leaf in the working suffix subtree.
2. Calculate the optimal buffer length \textit{range}
   - Note: The name \textit{Elastic Range}.
3. Read the next \textit{range} characters for each S-prefix occurrence.
4. Do in-memory sorting of read text, remember branching information (=LCP) and the original position (=SA).
5. Until all the read buffers are unique, goto step 2.
   - In the next step: While less leaves are orphans, \textit{range} increases, frequency drops.
Horizontal partitioning

For each virtual group, construct the corresponding suffix subtrees in parallel:

1. Locate and store all positions of S-prefixes for the virtual group. Each of located S-prefixes is to become a leaf in the working suffix subtree.
2. Calculate the optimal buffer length \textit{range}.
   - Note: The name \textit{Elastic Range}.
3. Read the next \textit{range} characters for each S-prefix occurrence.
4. Do in-memory sorting of read text, remember branching information (\(=\text{LCP}\)) and the original position (\(=\text{SA}\)).
5. Until all the read buffers are unique, goto step 2.
   - In the next step: While less leaves are orphans, \textit{range} increases, frequency drops.
6. Construct suffix subtree in D-F manner using SA and LCP.
Model of computation

Arge, Goodrich, Nelson, Sitchinava 2008
Parallel External Memory model (PEM):\(^7\)

- Shared memory model,
- 2-level memory hierarchy:
  - \(p\) processors, each with private cache of size \(M\) bytes.
  - parallel memory transfers in blocks of size \(B\) bytes.
- Performance metrics:
  - parallel time,
  - parallel block transfers (cache complexity).
- Concurrent reads assumed.

\(^7\)Arge, Goodrich, Nelson, Sitchinava 2008
If worst-case input text for ERA is skewed:

\[ T = AAA... \]
Assumption

If worst-case input text for ERA is skewed:

\[ T = \text{AAA...} \]

then

- vertical partitioning requires \( N \) scans = \( O(N^2) \) comparisons,
- cache complexity \( O\left(\frac{N^2}{B}\right)\).
If worst-case input text for ERA is skewed:

\[ T = \text{AAA...} \]

then

- vertical partitioning requires \( N \) scans = \( O(N^2) \) comparisons,
- cache complexity \( O\left(\frac{N^2}{B}\right) \).

Our assumption:

- Input text is random (viable for a single human genome, proteins).
- At any place the probability of each character to occur is \( \frac{1}{\sigma} \).
- Goal: Calculate expected time and cache complexity.
Algorithm 2: VerticalPartitioning

Input: Input string S, alphabet Σ, 1st level memory size M
Output: Set of VirtualTrees

1. VirtualTrees ← ∅
2. P ← ∅
3. P′ ← {∀ symbol s ∈ Σ generate a S-prefix π_i ∈ P′}
4. repeat
   5. scan input string S
   6. count in S the frequency f_{π_i} of every S-prefix π_i ∈ P′
   7. forall the π_i ∈ P′ do
      8. if 0 < f_{π_i} ≤ M then add π_i to P
      9. else forall the symbol s ∈ Σ do add π_i s to P′
     10. remove π_i from P′
   11. until P′ = ∅
12. sort P in descending f_{π_i} order
13. repeat
   14. G ← ∅
   15. add P.head to G and remove the item from P
   16. curr ← next item in P
   17. while NOT end of P do
      18. if f_{curr} + \sum_{γ_i \in G} (f_{γ_i}) ≤ M then
         19. add curr to G and remove the item from P
      20. curr ← next item in P
   21. add G to VirtualTrees
   22. until P = ∅
23. return VirtualTrees
### Analysis: Vertical partitioning

#### Expected behavior:

1. **Extension of S-prefixes:**
   
   Initially, \( \sigma \) S-prefixes of frequency \( f \) \( \pi \) each. \( f \pi \) divided by \( \sigma \) each iteration until \( f \pi < M \).

   Total log \( \sigma N - \log \sigma M = \log \sigma N M \) iterations.

   Finally, \( N M \) unique S-prefixes with frequency \( M \sigma < f \pi \leq M \).

2. **Atomic sorting the frequencies using one of the comparison-based sorting algorithms.**

3. **Virtual trees construction (bin packing problem):** At least 1 and at most \( \sigma \) S-prefixes are packed each iteration. External loop iterated between \( N \sigma M \) and \( N M \) times.
Analysis: Vertical partitioning

Expected behavior:

1. **Extension of S-prefixes:**
   - Initially $\sigma$ S-prefixes of frequency $f_\pi = \frac{N}{\sigma}$ each.
   - $f_\pi$ divided by $\sigma$ each iteration until $f_\pi < M$.
   - Total $\log_\sigma N - \log_\sigma M = \log_\sigma \frac{N}{M}$ iterations.
   - Finally $\frac{N}{M}$ unique S-prefixes with frequency $\frac{M}{\sigma} < f_\pi \leq M$. 

2. Atomic sorting the frequencies using one of the comparison-based sorting algorithms.

3. Virtual trees construction (bin packing problem): At least 1 and at most $\sigma$ S-prefixes are packed each iteration. External loop iterated between $N \sigma M$ and $N M$ times.
Expected behavior:

1. **Extension of S-prefixes:**
   - Initially $\sigma$ S-prefixes of frequency $f_\pi = \frac{N}{\sigma}$ each.
   - $f_\pi$ divided by $\sigma$ each iteration until $f_\pi < M$.
   - Total $\log_\sigma N - \log_\sigma M = \log_\sigma \frac{N}{M}$ iterations.
   - Finally $\frac{N}{M}$ unique S-prefixes with frequency $\frac{M}{\sigma} < f_\pi \leq M$.

2. **Atomic sorting the frequencies using one of the comparison-based sorting algorithms.**
Expected behavior:

1. **Extension of S-prefixes:**
   - Initially $\sigma$ S-prefixes of frequency $f_\pi = \frac{N}{\sigma}$ each.
   - $f_\pi$ divided by $\sigma$ each iteration until $f_\pi < M$.
   - Total $\log_\sigma N - \log_\sigma M = \log_\sigma \frac{N}{M}$ iterations.
   - Finally $\frac{N}{M}$ unique S-prefixes with frequency $\frac{M}{\sigma} < f_\pi \leq M$.

2. **Atomic sorting the frequencies using one of the comparison-based sorting algorithms.**

3. **Virtual trees construction (bin packing problem):**
   - At least 1 and at most $\sigma$ S-prefixes are packed each iteration.
   - External loop iterated between $\frac{N}{\sigma M}$ and $\frac{N}{M}$ times.
Analysis: Vertical partitioning — Time

Analysis of the vertical partitioning approach:

- **Extension of S-prefixes**
  \[ \log_\sigma N M \sum_{i=1}^{\text{scan}(n)} (\sigma_{i+1}) = \log_\sigma N M \cdot \text{scan}(n) + \sigma^2 (N - M) M \cdot \sigma - M = O(N \log_\sigma N M) \]

- **Sorting**
  \[ O(N M \log_\sigma N M) \]

- **Virtual trees construction**
  \[ O((N M)^2) \]

**Overall:**

- If \( \sigma < M \):
  \[ O(N \log_\sigma N M + (N M)^2) \]

- If \( \sigma \geq M \):
  \[ O(N \log_\sigma N M + \sigma N M + N M \log_\sigma N M + (N M)^2) \]
Analysis: Vertical partitioning — Time

1. Extension of S-prefixes

\[
\sum_{i=1}^{N} \left( \text{scan}(n) + \sigma^{i+1} \right) = \log_{\sigma} \frac{N}{M} \cdot \text{scan}(n) + \frac{\sigma^2(N-M)}{M \cdot (\sigma-M)} = O \left( N \log_{\sigma} \frac{N}{M} + \frac{\sigma N}{M} \right)
\]
Analysis: Vertical partitioning — Time

1. Extension of S-prefixes

\[ \log_{\sigma} \frac{N}{M} \sum_{i=1}^{\sigma+1} (\text{scan}(n) + \sigma^{i+1}) = \log_{\sigma} \frac{N}{M} \cdot \text{scan}(n) + \frac{\sigma^2(N-M)}{M \cdot \sigma - M} = O\left( N \log_{\sigma} \frac{N}{M} + \frac{\sigma N}{M} \right) \]

2. Sorting

\[ O\left( \frac{N}{M} \log \frac{N}{M} \right) \]
Analysis: Vertical partitioning — Time

1. Extension of S-prefixes
\[
\log_\sigma \frac{N}{M} \sum_{i=1}^{\sigma} (\text{scan}(n) + \sigma^{i+1}) = \log_\sigma \frac{N}{M} \cdot \text{scan}(n) + \frac{\sigma^2(N-M)}{M \cdot (\sigma-M)} = O\left(N \log_\sigma \frac{N}{M} + \frac{\sigma N}{M}\right)
\]

2. Sorting
\[
O\left(\frac{N}{M} \lg \frac{N}{M}\right)
\]

3. Virtual trees construction
\[
O\left(\left(\frac{N}{M}\right)^2\right)
\]
Analysis: Vertical partitioning — Time

1. Extension of S-prefixes
   \[ \log_{\sigma} \frac{N}{M} \sum_{i=1}^{\sigma} (\text{scan}(n) + \sigma^{i+1}) = \log_{\sigma} \frac{N}{M} \cdot \text{scan}(n) + \sigma^2 \frac{(N-M)}{M \cdot (\sigma-M)} = \]
   \[ O \left( N \log_{\sigma} \frac{N}{M} + \frac{\sigma N}{M} \right) \]

2. Sorting
   \[ O \left( \frac{N}{M} \lg \frac{N}{M} \right) \]

3. Virtual trees construction
   \[ O \left( \left( \frac{N}{M} \right)^2 \right) \]

Overall:
- If \( \sigma < M \): \( O \left( N \log_{\sigma} \frac{N}{M} + \left( \frac{N}{M} \right)^2 \right) \)
- If \( \sigma \geq M \): \( O \left( N \log_{\sigma} \frac{N}{M} + \frac{\sigma N}{M} + \frac{N}{M} \lg \frac{N}{M} + \left( \frac{N}{M} \right)^2 \right) \)
Analysis: Vertical partitioning — I/O

1. Extension of S-prefixes
   - Line 6: $\text{scan}(N)$ for reading
Analysis: Vertical partitioning — I/O

1. Extension of S-prefixes
   - Line 6: \( \text{scan}(N) \) for reading
   - \( |P'| = O \left( \frac{N}{M} \right) \)
   - If \( |P'| \leq M \): no I/Os for writing \( f_\pi \)
   - If \( |P'| > M \): \( \frac{M}{|P'|} = \frac{M^2}{N} \) I/Os for storing \( f_\pi \)
Analysis: Vertical partitioning — I/O

1 Extension of S-prefixes

- Line 6: $\text{scan}(N)$ for reading
  \[ |P'| = O \left( \frac{N}{M} \right) \]
  - If $|P'| \leq M$: no I/Os for writing $f_\pi$
  - If $|P'| > M$: $\frac{M}{|P'|} = \frac{M^2}{N}$ I/Os for storing $f_\pi$
- Lines 7-10:
  - If $|P'| \leq M$: no I/Os
  - If $|P'| > M$: $\frac{P'}{B} = \frac{N}{M \cdot B}$ I/Os
Extension of S-prefixes

- Line 6: \( \text{scan}(N) \) for reading
  \[ |P'| = O\left(\frac{N}{M}\right) \]
  If \( |P'| \leq M \): no I/Os for writing \( f_\pi \)
  If \( |P'| > M \): \( \frac{M}{|P'|} = \frac{M^2}{N} \) I/Os for storing \( f_\pi \)

- Lines 7-10:
  If \( |P'| \leq M \): no I/Os
  If \( |P'| > M \): \( \frac{P'}{B} = \frac{N}{M \cdot B} \) I/Os

Sorting \( |P| = \frac{N}{M} \) elements:
- If \( M \geq \sqrt{N} \): no I/Os
- If \( M < \sqrt{N} \): \( O\left(\frac{N}{M \cdot B} \log \frac{M}{B} \frac{N}{M \cdot B}\right) \) I/Os
Analysis: Vertical partitioning — I/O

1. Extension of S-prefixes
   - Line 6: \( \text{scan}(N) \) for reading
     \[ |P'| = O \left( \frac{N}{M} \right) \]
     If \( |P'| \leq M \): no I/Os for writing \( f_\pi \)
     If \( |P'| > M \): \( \frac{M}{|P'|} = \frac{M^2}{N} \) I/Os for storing \( f_\pi \)
   - Lines 7-10:
     If \( |P'| \leq M \): no I/Os
     If \( |P'| > M \): \( \frac{P'}{B} = \frac{N}{M \cdot B} \) I/Os

2. Sorting \( |P| = \frac{N}{M} \) elements:
   - If \( M \geq \sqrt{N} \): no I/Os
   - If \( M < \sqrt{N} \): \( O \left( \frac{N}{M \cdot B} \log \frac{M}{B} \cdot \frac{N}{M \cdot B} \right) \) I/Os

3. Virtual tree \( G \leq M \):
   - \( M \geq \sqrt{N} \): no I/Os
   - \( M < \sqrt{N} \): \( \frac{|P|}{B} = \frac{N}{M \cdot B} \) I/Os
Analysis: Vertical partitioning — I/O contd.

Overall:

- If \( M \geq \sqrt{N} \):
  \[
  O \left( \frac{N}{B} \log_{\sigma} \frac{N}{M} \right)
  \]

- If \( M < \sqrt{N} \):
  \[
  O \left( \log_{\sigma} \frac{N}{M} \cdot \left( \frac{N}{B} + M^2 \right) + \frac{N}{M \cdot B} \log_{\frac{M}{B}} \frac{N}{M \cdot B} + \left( \frac{N}{M \cdot B} \right)^2 \right)
  \]
Algorithm 3: HorizontalPartitioning.SubTreePrepare

**Input:** Input string $S$, S-prefix $\pi$

**Output:** Arrays $SA$ and $LCP$ corresponding suffix sub-tree $T_\pi$

1. $SA$ contains the locations of S-prefix $\pi$ in string $S$
2. $LCP \leftarrow \{\}$
3. $ISA \leftarrow \{0, 1, \ldots, |SA| - 1\}$
4. $A \leftarrow \{0, 0, \ldots, 0\}$
5. $Buf \leftarrow \{\}$
6. $P \leftarrow \{0, 1, \ldots, |L| - 1\}$
7. $start \leftarrow |\pi|$
8. **while** there exists an undefined $Buf[i]$, $1 \leq i \leq |SA| - 1$ **do**
   9. $range \leftarrow \text{GetRangeOfSymbols}$
   10. **for** $i \leftarrow 0$ to $|SA| - 1$ **do**
       11. **if** $ISA[i] \neq \text{done}$ **then**
           12. $Buf[ISA[i]] \leftarrow \text{ReadRange}(S, SA[ISA[i]] + start, range)$
               // ReadRange($S$, $a$, $b$) reads $b$ symbols of $S$ starting at position $a$
       13. **for** every active area $AA$ **do**
           14. Reorder the elements of $Buf$, $P$ and $SA$ in $AA$ so that $Buf$ is lexicographically sorted. In the process maintain the index $ISA$
           15. If two or more elements $\{a_1, \ldots, a_t\} \in AA, 2 \leq t$, exist such that $Buf[a_1] = \ldots = Buf[a_t]$ introduce for them a new active area
       16. **for** all $i$ such that $Buf[i]$ is not defined, $1 \leq i \leq |SA| - 1$ **do**
           17. $cp$ is the common prefix of $Buf[i - 1]$ and $Buf[i]$
           18. **if** $|cp| < range$ **then**
               19. $Buf[i] \leftarrow (Buf[i - 1][|cp|], Buf[i][|cp|], start + |cp|)$
               20. **if** $Buf[i - 1]$ is defined or $i = 1$ **then**
                    21. Mark $ISA[P[i - 1]]$ and $A[i - 1]$ as done
               **if** $Buf[i + 1]$ is defined or $i = |SA| - 1$ **then**
                    22. Mark $ISA[P[i]]$ and $A[i]$ as done // last element of an active area
       23. $start \leftarrow start + range$
24. **return** $(SA, LCP)$
Analysis: Horizontal partitioning

Expected behaviour:
- Define $n$ the number of unfinished branches, then $n \cdot \text{range} = O(M)$.

Intuitively, $n$ decreases and $\text{range}$ increases during execution of lines 8-24.

For length $k$, there can be at most $\sigma_k$ unique strings. For random text and step $1 \leq i \leq k$, strings are non-unique until $k$ is reached.

If $O(M)$ random strings need to be processed, then lines 8-24 is iterated $O(\log \sigma M)$ times. The big-oh constant depends on $\text{range}$. 
Analysis: Horizontal partitioning

Expected behaviour:
Expected behaviour:

- Define $n$ the number of unfinished branches, then $n \cdot \text{range} = O(M)$. 
Analysis: Horizontal partitioning

Expected behaviour:

- Define $n$ the number of unfinished branches, then $n \cdot \text{range} = O(M)$.
- Intuitively $n$ decreases and $\text{range}$ increases during execution of lines 8-24.
Analysis: Horizontal partitioning

Expected behaviour:

- Define $n$ the number of unfinished branches, then $n \cdot range = O(M)$.
- Intuitively $n$ decreases and $range$ increases during execution of lines 8-24.
- For length $k$, there can be at most $\sigma^k$ unique strings. For random text and step $1 \leq i \leq k$, strings are non-unique until $k$ is reached.
Expected behaviour:

- Define $n$ the number of unfinished branches, then $n \cdot \text{range} = O(M)$.
- Intuitively $n$ decreases and range increases during execution of lines 8-24.
- For length $k$, there can be at most $\sigma^k$ unique strings. For random text and step $1 \leq i \leq k$, strings are non-unique until $k$ is reached.
- If $O(M)$ random strings need to be processed, then lines 8-24 is iterated $O(\log_\sigma M)$ times. The big-oh constant depends on range.
Analysis: Horizontal partitioning — Time

1. Lines 10-12 required \( n \) time to fill the buffers (constant time read).

2. String sorting requires \( O(n \cdot \text{range}) \) time since the average distinguishing prefix size equals \( O(\text{range}) \).

3. Lines 16-23 require \( O(n \cdot \text{range}) \) time in the worst case.

Overall: Assuming \( p \) processors equally balanced after processing \( O(N/M) \) virtual groups require \( O(N/p \log \sigma M) = O(N/p \log \sigma M) \).
Analysis: Horizontal partitioning — Time

Each iteration:

1. Lines 10-12 required \( n \) time to fill the buffers (constant time read).
Each iteration:

1. Lines 10-12 required $n$ time to fill the buffers (constant time read).

2. String sorting requires $O(n \cdot \text{range})$ time since the average distinguishing prefix size equals $O(\text{range})$. 

Analysis: Horizontal partitioning — Time
Analysis: Horizontal partitioning — Time

Each iteration:

1. Lines 10-12 required $n$ time to fill the buffers (constant time read).
2. String sorting requires $O(n \cdot \text{range})$ time since the average distinguishing prefix size equals $O(\text{range})$.
3. Lines 16-23 require $O(n \cdot \text{range})$ time in the worst case.
Analysis: Horizontal partitioning — Time

Each iteration:

1. Lines 10-12 required $n$ time to fill the buffers (constant time read).
2. String sorting requires $O(n \cdot \text{range})$ time since the average distinguishing prefix size equals $O(\text{range})$.
3. Lines 16-23 require $O(n \cdot \text{range})$ time in the worst case.

Overall: Assuming $p$ processors equally balanced after processing $O(N/M)$ virtual groups require

$$O \left( \frac{N}{M} \frac{M \log \sigma}{p} \frac{M}{p} \right) = O \left( \frac{N}{p} \log \sigma M \right)$$
Analysis: Horizontal partitioning — I/O

Cache misses occur in lines 10-12 only:

\[
\text{If } n \geq N_B, \text{ then } O(N_B) \text{ I/Os.}
\]

Else:

\[
O(n) \text{ I/Os}
\]

When does the change from \( n \geq N_B \) to \( n < N_B \) occur?

Assuming uniformly random text, \( n = c \cdot M \) for some constant \( c \) all the time! (all branches are open until the last iteration)

Suffix subtree construction from SA and LCP requires a single scan \( (N) \text{ I/Os only and is omitted.} \)

I/O complexity for horizontal partitioning is thus \( O(\min(M, N_B) \cdot \log \sigma M) \).
Cache misses occur in lines 10-12 only:
Cache misses occur in lines 10-12 only:

- If $n \geq \frac{N}{B}$, then $O\left(\frac{N}{B}\right)$ I/Os.
Cache misses occur in lines 10-12 only:

- If $n \geq \frac{N}{B}$, then $O\left(\frac{N}{B}\right)$ I/Os.
- Else: $O(n)$ I/Os
Cache misses occur in lines 10-12 only:

- If $n \geq \frac{N}{B}$, then $O\left(\frac{N}{B}\right)$ I/Os.
- Else: $O(n)$ I/Os

When does the change from $n \geq \frac{N}{B}$ to $n < \frac{N}{B}$ occur?
Cache misses occur in lines 10-12 only:

- If $n \geq \frac{N}{B}$, then $O\left(\frac{N}{B}\right)$ I/Os.
- Else: $O(n)$ I/Os

When does the change from $n \geq \frac{N}{B}$ to $n < \frac{N}{B}$ occur?

Assuming uniformly random text, $n = c \cdot M$ for some constant $c$ all the time! (all branches are open until the last iteration)
1 Cache misses occur in lines 10-12 only:
   - If \( n \geq \frac{N}{B} \), then \( O\left(\frac{N}{B}\right) \) I/Os.
   - Else: \( O(n) \) I/Os

2 When does the change from \( n \geq \frac{N}{B} \) to \( n < \frac{N}{B} \) occur?

3 Assuming uniformly random text, \( n = c \cdot M \) for some constant \( c \) all the time! (all branches are open until the last iteration)

4 Suffix subtree construction from SA and LCP requires a single \( \text{scan}(N) \) I/Os only and is omitted.
**Analysis: Horizontal partitioning — I/O**

1. Cache misses occur in lines 10-12 only:
   - If \( n \geq \frac{N}{B} \), then \( O\left(\frac{N}{B}\right) \) I/Os.
   - Else: \( O(n) \) I/Os

2. When does the change from \( n \geq \frac{N}{B} \) to \( n < \frac{N}{B} \) occur?

3. Assuming uniformly random text, \( n = c \cdot M \) for some constant \( c \) **all the time**! (all branches are open until the last iteration)

4. Suffix subtree construction from SA and LCP requires a single \( \text{scan}(N) \) I/Os only and is omitted.

5. I/O complexity for horizontal partitioning is thus

\[
O \left( \min \left( M, \frac{N}{B} \right) \cdot \log_\sigma M \right)
\]
Parallel time complexity of ERA (assuming $\sigma \leq M$):
$$O\left(\frac{N \log \sigma N M^2}{p \log \sigma M}\right)$$

Parallel cache complexity of ERA (assuming $M \geq \sqrt{N}$):
$$O\left(\frac{N B \log \sigma N M}{\min(M, N B)} \cdot \log \sigma M p\right)$$
Parallel time complexity of ERA (assuming $\sigma \leq M$):

$$O\left(N \log_\sigma \frac{N}{M} + \left(\frac{N}{M}\right)^2 + \frac{N}{p} \log_\sigma M\right)$$

Parallel cache complexity of ERA (assuming $M \geq \sqrt{N}$):

$$O\left(\frac{N}{B} \log_\sigma \frac{N}{M} + \frac{\min\left(M, \frac{N}{B}\right) \cdot \log_\sigma M}{p}\right)$$
<table>
<thead>
<tr>
<th>Testing environment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x 16-core AMD Opteron 6272 @2 GHz</td>
</tr>
<tr>
<td>100 MHz</td>
</tr>
<tr>
<td>128 GiB RAM</td>
</tr>
<tr>
<td>Seagate Baracuda 250 GB, 7,200 RPM, 32 MiB cache, SATA</td>
</tr>
<tr>
<td>Ubuntu server 12.04, Linux kernel 3.11.0, ext4 file system, deadline I/O scheduler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERA parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory size per core: 2 GiB</td>
</tr>
<tr>
<td>Input text: Human genome HG18.txt, 2.8 Gbp</td>
</tr>
</tbody>
</table>

| ERA modification:                        |
| Call fsync() after writing each file.    |

| ERA output:                               |
| Total suffix tree size: 77.3 GB stored in 187 files |
| Top size: 10.2 KB                         |
Empirical evaluation

Testing environment:
- 2x 16-core AMD Opteron 6272 @2,100 MHz
- 128 GiB RAM
- Seagate Baracuda 250 GB, 7,200 RPM, 32 MiB cache, SATA
- Ubuntu server 12.04, Linux kernel 3.11.0
- ext4 file system, deadline I/O scheduler

ERA parameters:
- Memory size per core: 2 GiB
- Input text: Human genome HG18.txt, 2.8 Gbp

ERA modification: Call `fsync()` after writing each file.

ERA output:
- Total suffix tree size: 77.3 GB stored in 187 files
- $T_{top}$ size: 10.2 KB
Results – 1

The time increases as we increase the number of cores.
The time **increases** as we increase the number of cores.
So what is the machine doing?
Results – 2

So what is the machine doing?

- **string cpy** Parsing and copying the string.
- **vertpart** Vertical partitioning.
- **cnt1, cnt** Horizontal partitioning: determining locations of S-prefix in virtual trees of size 1 or > 1.
- **filbuf** Horizontal partitioning: reading range characters from S-prefix locations.
- **sort** Horizontal partitioning: string sorting, implicit LCP, SA construction.
- **write** Horizontal partitioning: extraction from LCP and SA to suffix tree, write to disk.
Results – 2 contd.

parallel10_devnullprobability CPU times p00
Results – 2 contd.

p = 1, t=5972.0s
- rMB/s avg: 0.47
- wMB/s avg: 13.09
- await [cs] avg: 17.51
- avg. queue sz avg: 17.94

p = 2, t=3463.0s
- rMB/s avg: 0.74
- wMB/s avg: 22.36
- await [cs] avg: 26.77
- avg. queue sz avg: 30.73

p = 3, t=2507.0s
- rMB/s avg: 1.07
- wMB/s avg: 30.87
- await [cs] avg: 34.04
- avg. queue sz avg: 42.73

p = 4, t=2033.0s
- rMB/s avg: 1.31
- wMB/s avg: 37.95
- await [cs] avg: 40.45
- avg. queue sz avg: 54.7

p = 6, t=1678.0s
- rMB/s avg: 1.62
- wMB/s avg: 46.48
- await [cs] avg: 55.31
- avg. queue sz avg: 69.74

p = 8, t=1398.0s
- rMB/s avg: 2.25
- wMB/s avg: 59.11
- await [cs] avg: 65.77
- avg. queue sz avg: 100.25

p = 12, t=1303.0s
- rMB/s avg: 2.26
- wMB/s avg: 62.23
- await [cs] avg: 65.09
- avg. queue sz avg: 105.2

p = 16, t=1243.0s
- rMB/s avg: 2.32
- wMB/s avg: 61.07
- await [cs] avg: 67.97
- avg. queue sz avg: 106.73

p = 20, t=1316.0s
- rMB/s avg: 2.24
- wMB/s avg: 58.35
- await [cs] avg: 66.98
- avg. queue sz avg: 101.92

p = 24, t=1255.0s
- rMB/s avg: 2.32
- wMB/s avg: 61.07
- await [cs] avg: 67.97
- avg. queue sz avg: 106.73

p = 28, t=1295.0s
- rMB/s avg: 2.26
- wMB/s avg: 59.63
- await [cs] avg: 66.98
- avg. queue sz avg: 101.16

p = 32, t=1306.0s
- rMB/s avg: 2.31
- wMB/s avg: 60.06
- await [cs] avg: 64.75
- avg. queue sz avg: 101.16
Results – 2 contd.
Observation 1: The majority of time is spent writing the final result to the disk.
Hypothesis 1

Observation 1: The majority of time is spent writing the final result to the disk.

Hypothesis 1: Problem is the disk performance, so replace HDD with SSD.
Results – 3

parallel10_devnullprobability_ssd CPU times p00

- p = 1, t=5657.0s, s=p00
- p = 2, t=3068.0s, s=p00
- p = 3, t=2183.0s, s=p00
- p = 4, t=1700.0s, s=p00
- p = 6, t=1394.0s, s=p00
- p = 8, t=1064.0s, s=p00
- p = 12, t=898.0s, s=p00
- p = 16, t=804.0s, s=p00
- p = 20, t=826.0s, s=p00
- p = 24, t=772.0s, s=p00
- p = 28, t=801.0s, s=p00
- p = 32, t=790.0s, s=p00
**Observation 2**: The amount of time for writing decreased, but as the number of cores grows, it is still substantial.
Hypothesis 2

*Observation 2:* The amount of time for writing decreased, but as the number of cores grows, it is still substantial.

*Hypothesis 2:* There it is still a problem with a disk performance and consequently further speed-up disk by writing to `/dev/null`.
Results – 4

Times per # of proc., /dev/null prob. wise

- Probability = 0.0
- Probability = 0.1
- Probability = 0.2
- Probability = 0.3
- Probability = 0.4
- Probability = 0.5
- Probability = 0.6
- Probability = 0.7
- Probability = 0.8
- Probability = 0.9
- Probability = 1.0
Results – 4 contd.
**Observation 3:** Things are getting better, but there is still an increase in time when the number of cores is increased.
Observation 3: Things are getting better, but there is still an increase in time when the number of cores is increased.

Hypothesis 3: ??

Check in more detail what the processes are doing.
Results – 5 ($p = 16$)
Results – 5 ($p = 16$), strace
Results – 5 ($p = 32$)

[Graph showing CPU times per CPU, $p = 32$, with legend indicating different operations and times: string cpy=73s, vertpart=193s, cnt1=98s, cnt*=3232s, filbuf=1617s, sort=2821s, write=20007s.]
Results – 5 ($p = 32$), strace
**Conclusion**

Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.

ERA despite being practically the fastest algorithm is not theoretically tight even for random input strings with uniform substring distribution.

Open challenges:
- Analyse ERA bottlenecks for further improvements (see if they match the critical terms in time and I/O complexities).
- Shall we choose some other basic technique for the implementation of a practical algorithm?
- Design a theoretically tight yet practically competitive parallel algorithm for suffix tree construction.
Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.
**Conclusion**

- Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.
- ERA despite being practically the fastest algorithm is **not theoretically tight** even for random input strings with uniform substring distribution.
Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.

ERA despite being practically the fastest algorithm is not **theoretically tight** even for random input strings with uniform substring distribution.

Open challenges:
Conclusion

- Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.
- ERA despite being practically the fastest algorithm is **not theoretically tight** even for random input strings with uniform substring distribution.

Open challenges:

- Analyse ERA bottlenecks for further improvements (see if they match the critical terms in time and I/O complexities).
Conclusion

- Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.
- ERA despite being practically the fastest algorithm is not theoretically tight even for random input strings with uniform substring distribution.

Open challenges:

- Analyse ERA bottlenecks for further improvements (see if they match the critical terms in time and I/O complexities).
- Shall we choose some other basic technique for the implementation of a practical algorithm?
Huge gap between the theoretical time and I/O asymptotically tight algorithms and the practical ones.

ERA despite being practically the fastest algorithm is not theoretically tight even for random input strings with uniform substring distribution.

Open challenges:

- Analyse ERA bottlenecks for further improvements (see if they match the critical terms in time and I/O complexities).
- Shall we choose some other basic technique for the implementation of a practical algorithm?
- Design a theoretically tight yet practically competitive parallel algorithm for suffix tree construction.
Thank you.

Matevž Jekovec
matevz.jekovec@fri.uni-lj.si

Andrej Brodnik
andrej.brodnik@fri.uni-lj.si

University of Ljubljana
Faculty of Computer and Information Science

Laboratorij za vseprisotne sisteme
Laboratory for Ubiquitous SYstems
http://lusy.fri.uni-lj.si
Shown at the presentation:

- Execution time for different $p$, speed-up, efficiency.
- CPU times, `iostat` and `mpstat` per different phases for $p = \{1, 2, 3, 4, 6, 8, 12, 16, 20, 24, 27, 32\}$.
- `iostat` output for various $p$.
- `mpstat` output for various $p$.
- Work per core for single execution for $p = 32$.
- Using `strace`, fetching `read`, `write`, `lseek` syscalls.
Test scenarios:

- Original code + added `fsync()`, various # of cores, various mem. size per core.
- Various string buffer sizes `BUF_TYPE = \{8, 16, 32, 64\}` bit.
- Integration of Multikey cached quicksort (Rantala-Bentley-Sedgewick) instead of GNU `qsort`.
- Maximum limit of simultaneously opened files for writing $F = \{1, 2, 3, 4, 5, 6, 12, 16\}$.
- Output to `/dev/null` with probability $Pr = [0, 0.1...1]$.
- Separated disk for writing and reading.
- SSD for reading and/or writing.
- Different file system schedulers: `noop`, `default`, `cfq`.
- Different file system max queue length.
- Output to raw device without file system.
- Execution on 12x Raspberry $\pi$ with shared NFS storage.