ERa — A Practical Approach to Parallel Construction of Suffix Trees

Andrej (Andy) Brodnik[1,2], Matevž Jekovec[1]

[1] University of Ljubljana, Faculty of Computer and Information Science
[2] University of Primorska, Department of Information Sciences and Technologies

Dagstuhl, *Data Structures and Advanced Models of Computation on Big Data*, February 24-28, 2014

Text indexing problem

Text indexing problem

Problem statement

Given unstructured input text T consisting of N characters from alphabet Σ of size σ build an index such that for query pattern P we:

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

Text indexing problem

Problem statement

Given unstructured input text T consisting of N characters from alphabet Σ of size σ build an index such that for query pattern P we:

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

Solution

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

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Work w. c.	$O(N)$	$O(N)$	$O(N \lg N)$
Online	No	Yes	Yes[1]
I/O efficiency	String	String	Result+String
Unbounded Σ	No	No	Yes
Parallel	No	No	PDAM

[1] Bedathur and Haritsa (2004)

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Work w. c.	$O(N)$	$O(N)$	$O(N \lg N)$
Online	No	Yes	Yes[1]
I/O efficiency	String	String	Result+String
Unbounded Σ	No	No	Yes
Parallel	No	No	PDAM

- Practical:

	Semi-disk-based			Out-of-core		
	TDD ('04)	TRLS. ('07)	B^2ST ('09)	WF ('09)	ERa ('11)	PCF ('13)
Work w. c.	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(pN \lg N$
I/O eff.	R.	R.	R.+S.	R.+S.	R.+S.	R.+S.
Unbnd. Σ	No	No	No	No	No	No
Parallel	No	No	No	Yes	Yes	Yes

[1] Bedathur and Haritsa (2004)

Suffix tree construction algorithms

- Theoretical:

	W ('73), McC ('78)	U ('95)	F-C et al. ('00)
Work w. c.	$O(N)$	$O(N)$	$O(N \lg N)$
Online	No	Yes	Yes[1]
I/O efficiency	String	String	Result+String
Unbounded Σ	No	No	Yes
Parallel	No	No	PDAM

- Practical:

	Semi-disk-based			Out-of-core		
	TDD ('04)	TRLS. ('07)	B^2ST ('09)	WF ('09)	ERa ('11)	PCF ('13)
Work w. c.	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(pN \lg N$
I/O eff.	R.	R.	R.+S.	R.+S.	R.+S.	R.+S.
Unbnd. Σ	No	No	No	No	No	No
Parallel	No	No	No	Yes	Yes	Yes

[1] Bedathur and Haritsa (2004)

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(Sort(N))$	$\Omega(Sort(N))$
I/Os[2]	$\Omega(Sort(N))$	$\Omega(Sort(N))$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

[2] EM model
[3] Uncompressed index in word RAM
[4] PEM model

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(Sort(N))$	$\Omega(Sort(N))$
I/Os[2]	$\Omega(Sort(N))$	$\Omega(Sort(N))$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

- Parallel on p processing units:

	bounded Σ	unbounded Σ
Parallel time	$\Omega(Sort_p(N))$	$\Omega(Sort_p(N))$
Parallel I/Os[4]	$\Omega(Sort_p(N))$	$\Omega(Sort_p(N))$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

[2] EM model
[3] Uncompressed index in word RAM
[4] PEM model

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(N)$	$\Omega(N \log N)$
I/Os[2]	$\Omega\left(\frac{N}{B}\right)$	$\Omega\left(\frac{N}{B} log_{\frac{M}{B}} \frac{N}{B}\right)$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

- Parallel on p processing units:

	bounded Σ	unbounded Σ
Parallel time	$\Omega\left(\frac{N}{p}\right)$	$\Omega\left(\frac{N}{p} \log N\right)$
Parallel I/Os[4]	$\Omega\left(\frac{N}{pB}\right)$	$\Omega\left(\frac{N}{pB} log_{\frac{M}{B}} \frac{N}{B}\right)$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

[2] EM model
[3] Uncompressed index in word RAM
[4] PEM model

Suffix tree construction lower bounds

- Sequential:

	bounded Σ	unbounded Σ
Time	$\Omega(N)$	$\Omega(N \log N)$
I/Os[2]	$\Omega\left(\frac{N}{B}\right)$	$\Omega\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

- Parallel on p processing units:

	bounded Σ	unbounded Σ
Parallel time	$\Omega\left(\frac{N}{p}\right)$	$\Omega\left(\frac{N}{p} \log N\right)$
Parallel I/Os[4]	$\Omega\left(\frac{N}{pB}\right)$	$\Omega\left(\frac{N}{pB} \log_{\frac{M}{B}} \frac{N}{B}\right)$
Space[3]	$\Omega(N \lg \sigma)$ bits	$\Omega(N \lg \sigma)$ bits

[2] EM model

[3] Uncompressed index in word RAM

[4] PEM model

Theory and Practice

- Substantial gap between the theoretical and practical results.

Theory and Practice

- Substantial gap between the theoretical and practical results.
- Practitioners (often) do not use theoretically the best results.

Theory and Practice

- Substantial gap between the theoretical and practical results.
- Practitioners (often) do not use theoretically the best results.
- Perhaps we should look at practical solutions more carefully.

ERa — Elastic Range (Mansour et al. (2011))

- Currently the fastest practical, parallel suffix tree construction algorithm.

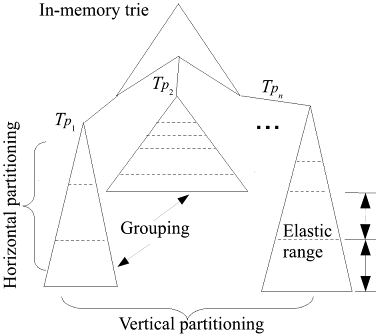
ERa — Elastic Range (Mansour et al. (2011))

- Currently the fastest practical, parallel suffix tree construction algorithm.
- Time complexity: $O(N^2)$ w.c. – for (extremely) skewed text!

ERa — Elastic Range (Mansour et al. (2011))

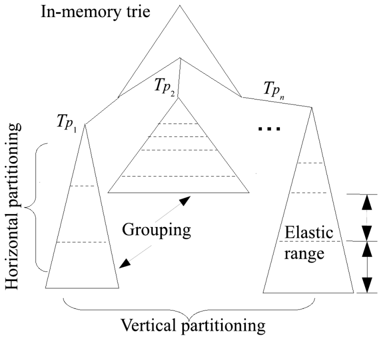
- Currently the fastest practical, parallel suffix tree construction algorithm.
- Time complexity: $O(N^2)$ w.c. – for (extremely) skewed text!
- Yet, it's **fast** in practice: Constructs and stores the human genome's suffix tree in 20 minutes on 16-core desktop PC with HDD or 13 minutes with SSD!

ERa



ERa constructs the suffix tree in two steps:

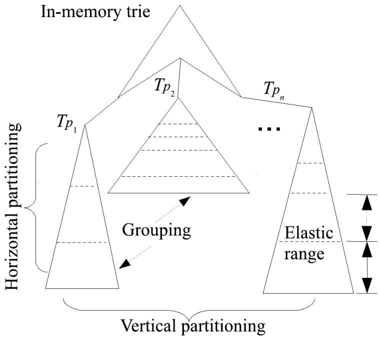
ERa



ERa constructs the suffix tree in two steps:

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

ERa



ERa constructs the suffix tree in two steps:

1. The **vertical partitioning** step determines 1) the suffix subtrees just fitting into M and 2) constructs the suffix tree top.
2. The **horizontal partitioning** step builds the actual suffix subtrees.

ERa

Algorithm 1: ERa

Input: String S, Alphabet Σ, Processors P, Private cache size M

Output: Suffix tree $\mathcal{T}$

$\mathcal{T}_{top}, G \leftarrow \mathit{VerticalPartitioning}(S, \Sigma, M)$
$\mathcal{T} \leftarrow \mathcal{T}_{top}$
while $|G| > 0$ **do**
    for $p \in P$ **do in parallel**
        if $|G| > 0$ **then**
            $\pi \leftarrow G.pop()$
            $\mathcal{T}_\pi \leftarrow \mathit{HorizontalPartitioning}(S, \Sigma, \pi)$
            $\mathit{Link}(\mathcal{T}, \mathcal{T}_\pi)$
        end
    end
end
return $\mathcal{T}$

Vertical partitioning

Define **S-prefix** π as the prefix of the suffix in the text; $f_\pi = \#$of suffixes starting with π; and $\mathcal{T}_\pi$ is a subtree with a root corresponding π.

Vertical partitioning

Define **S-prefix** π as the prefix of the suffix in the text; $f_\pi = \#$of suffixes starting with π; and $\mathcal{T}_\pi$ is a subtree with a root corresponding π.

Idea: Approximate the size of $\mathcal{T}_\pi$ as cf_π for some constant c and expand the π so much, that $cf_\pi \leq M$.

Vertical partitioning

Define **S-prefix** π as the prefix of the suffix in the text; $f_\pi = \#$of suffixes starting with π; and $\mathcal{T}_\pi$ is a subtree with a root corresponding π.

Idea: Approximate the size of $\mathcal{T}_\pi$ as cf_π for some constant c and expand the π so much, that $cf_\pi \leq M$.

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

Vertical partitioning

Define **S-prefix** π as the prefix of the suffix in the text; $f_\pi = \#$of suffixes starting with π; and $\mathcal{T}_\pi$ is a subtree with a root corresponding π.

Idea: Approximate the size of $\mathcal{T}_\pi$ as cf_π for some constant c and expand the π so much, that $cf_\pi \leq M$.

1. Scan the text and obtain the characters frequency $f_\pi : \pi \in \Sigma$ (counted all S-prefixes of length 1)
2. For each $\pi : cf_\pi > M$, $\pi' = \pi\Sigma$ and count $f_{\pi'}$.

Vertical partitioning

Define **S-prefix** π as the prefix of the suffix in the text; $f_\pi = \#$of suffixes starting with π; and $\mathcal{T}_\pi$ is a subtree with a root corresponding π.

Idea: Approximate the size of $\mathcal{T}_\pi$ as cf_π for some constant c and expand the π so much, that $cf_\pi \leq M$.

1. Scan the text and obtain the characters frequency $f_\pi : \pi \in \Sigma$ (counted all S-prefixes of length 1)
2. For each $\pi : cf_\pi > M$, $\pi' = \pi\Sigma$ and count $f_{\pi'}$.
3. Repeat step two for S-prefixes of length $3, 4, \ldots$, until all $\mathcal{T}_\pi$ just fits into the memory M.

Vertical partitioning

Define **S-prefix** π as the prefix of the suffix in the text; $f_\pi = \#$of suffixes starting with π; and $\mathcal{T}_\pi$ is a subtree with a root corresponding π.

Idea: Approximate the size of $\mathcal{T}_\pi$ as cf_π for some constant c and expand the π so much, that $cf_\pi \leq M$.

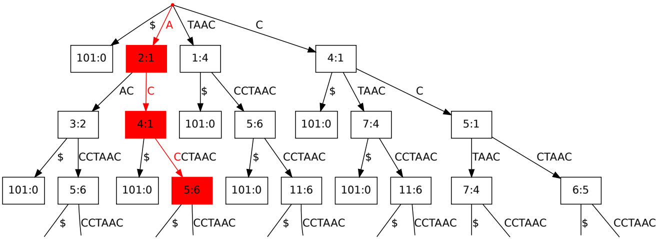
1. Scan the text and obtain the characters frequency $f_\pi : \pi \in \Sigma$ (counted all S-prefixes of length 1)
2. For each $\pi : cf_\pi > M$, $\pi' = \pi\Sigma$ and count $f_{\pi'}$.
3. Repeat step two for S-prefixes of length $3, 4, \ldots$, until all $\mathcal{T}_\pi$ just fits into the memory M.

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)

Vertical partitioning — Example

$\pi = ACC,\ f_{ACC} = 12$

TAACCCTA
ACCCTAAC
CCTAACCC
TAACCCTA
ACCCTAAC
CCTAACCC
TAACCCTA
ACCCTAAC
CCTAACCC
TAACCCTA
ACCCTAAC
CCTAACCC
TAAC



Horizontal partitioning

For a prefix π (virtual group G) construct the suffix subtree:

Horizontal partitioning

For a prefix π (virtual group G) construct the suffix subtree:

1. Locate and store all $n = f_\pi$ positions of π – we will sort n strings in-memory; i.e. all strings are ambigous.

Horizontal partitioning

For a prefix π (virtual group G) construct the suffix subtree:

1. Locate and store all $n = f_\pi$ positions of π – we will sort n strings in-memory; i.e. all strings are ambigous.
2. Optimal prefix length range of chars following π $r = \rho\frac{M}{n}$ – *Elastic Range*.

Horizontal partitioning

For a prefix π (virtual group G) construct the suffix subtree:

1. Locate and store all $n = f_\pi$ positions of π – we will sort n strings in-memory; i.e. all strings are ambigous.
2. Optimal prefix length range of chars following π $r = \rho \frac{M}{n}$ – *Elastic Range*.
3. Read the next r characters for each string.

Horizontal partitioning

For a prefix π (virtual group G) construct the suffix subtree:

1. Locate and store all $n = f_\pi$ positions of π – we will sort n strings in-memory; i.e. all strings are ambigous.
2. Optimal prefix length range of chars following π $r = \rho \frac{M}{n}$ – *Elastic Range*.
3. Read the next r characters for each string.
4. Sort strings in-memory and record branching information (=LCP) and the original position (=SA).

Horizontal partitioning

For a prefix π (virtual group G) construct the suffix subtree:

1. Locate and store all $n = f_\pi$ positions of π – we will sort n strings in-memory; i.e. all strings are ambigous.
2. Optimal prefix length range of chars following π $r = \rho\frac{M}{n}$ – *Elastic Range*.
3. Read the next r characters for each string.
4. Sort strings in-memory and record branching information (=LCP) and the original position (=SA).
5. Let n be the number of strings that are still ambigous; go to step 2 until $n = 0$ (as r increases n decreases).

Horizontal partitioning

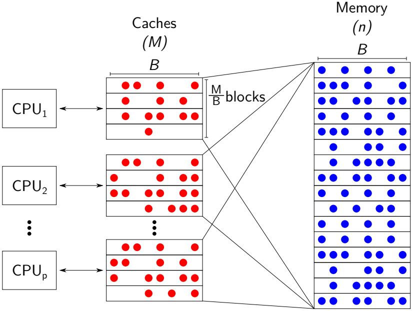
For a prefix π (virtual group G) construct the suffix subtree:

1. Locate and store all $n = f_\pi$ positions of π – we will sort n strings in-memory; i.e. all strings are ambigous.
2. Optimal prefix length range of chars following π $r = \rho\frac{M}{n}$ – *Elastic Range*.
3. Read the next r characters for each string.
4. Sort strings in-memory and record branching information (=LCP) and the original position (=SA).
5. Let n be the number of strings that are still ambigous; go to step 2 until $n = 0$ (as r increases n decreases).
6. Construct suffix subtree in DF manner using SA and LCP.

Model of computation

Parallel External Memory model (PEM):[5]

- 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.



[5] Arge, Goodrich, Nelson, Sitchinava 2008

Assumptions and Definitions

- Alphabet Σ of size σ; the number of processors p; the size of block B; the size of memory M; and the length of input string N.
- The worst-case the input text is skewed: $T = AAA...$

Assumptions and Definitions

- Alphabet Σ of size σ; the number of processors p; the size of block B; the size of memory M; and the length of input string N.
- The worst-case the input text is skewed: $T = AAA...$
- Vertical partitioning expands S-prefixes by one character at the time.
- Requires N scans, that equals $\frac{N^2}{2}$ comparisons.
- Cache complexity $O(\frac{N^2}{B})$.

Assumptions and Definitions

- Alphabet Σ of size σ; the number of processors p; the size of block B; the size of memory M; and the length of input string N.
- The worst-case the input text is skewed: $T = AAA...$
- Vertical partitioning expands S-prefixes by one character at the time.
- Requires N scans, that equals $\frac{N^2}{2}$ comparisons.
- Cache complexity $O(\frac{N^2}{B})$.

Our assumption:

- Input text is random (viable for a single genome, proteins).
- At any place the probability of each character to occur is $\frac{1}{\sigma}$.
- The suffix tree build from a random string is shallowest (Szpankowski 1993).

Analysis: Vertical partitioning

Assume $M < \sqrt{N}$.

- Extending π by one from 1 till $\log_\sigma \frac{N}{M}$ and hence this many scans of the text.
- Sorting $P = N/M$ prefixes.
- Packing prefixes into virtual groups.

Analysis: Vertical partitioning

Assume $M < \sqrt{N}$.

- Extending π by one from 1 till $\log_\sigma \frac{N}{M}$ and hence this many scans of the text.
- Sorting $P = N/M$ prefixes.
- Packing prefixes into virtual groups.
- Consequently the number of I/Os

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

Analysis: Horizontal partitioning

For building of P subtrees by p processors, where each subtree:

- Designed that data structure fits into memory.

Analysis: Horizontal partitioning

For building of P subtrees by p processors, where each subtree:

- Designed that data structure fits into memory.
- Consequently the number of I/Os

$$O\left(\frac{N}{B} \cdot \log_{\sigma} \frac{N}{M}\right)$$

Evaluation: environment

Computer and environment:

- 2× 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
- MPI programming

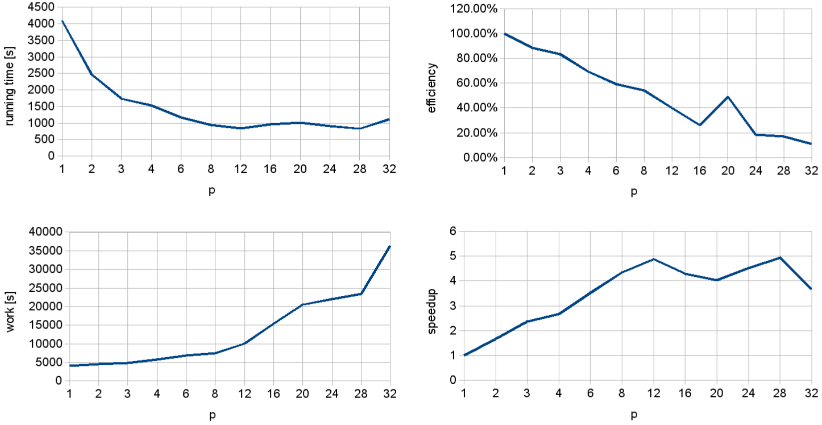
ERa parameters:

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

ERa output:

- Total suffix tree size: 77.3 GB stored in 187 files
- $\mathcal{T}_{top}$ size: 10.2 KB

Results – 1



The time increases as we increase the number of cores.

Results – 2

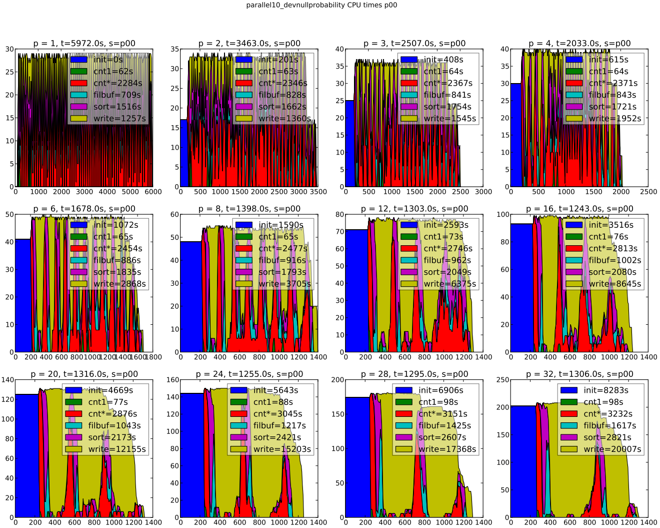
So what is the machine doing?

Results – 2

So what is the machine doing?

- init: Initialization including broadcasting to MPI clients.
- cnt*, cnt1: Vertical partitioning: counting the frequency and locating the S-prefixes in the input text.
- filbuf: Horizontal partitioning: reading the input text.
- sort: Horizontal partitioning: in-memory string sorting.
- write: Horizontal partitioning: writing the final result to disk.

Results – 2 contd.



Hypotheis 1

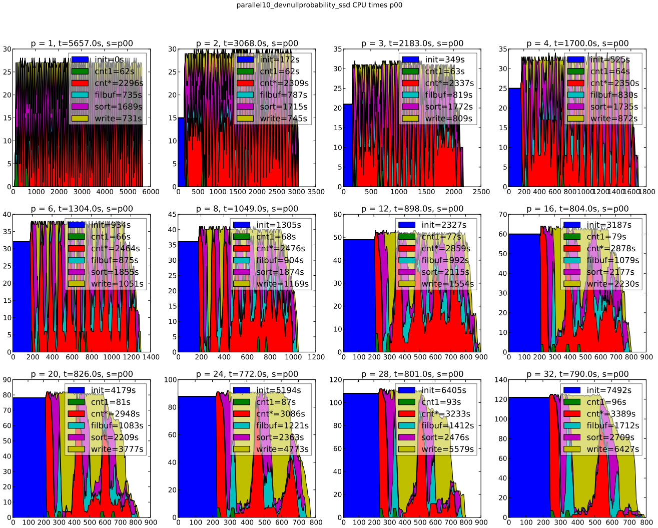
Observation 1: The majority of time is spent writing the final result to the disk.

Hypotheis 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
init=0s
cnt1=62s
cnt*=2296s
filbuf=735s
sort=1689s
write=731s
p = 2, t=3068.0s, s=p00
init=172s
cnt1=62s
cnt*=2309s
filbuf=787s
sort=1715s
write=745s
p = 3, t=2183.0s, s=p00
init=349s
cnt1=63s
cnt*=2337s
filbuf=819s
sort=1772s
write=809s
p = 4, t=1700.0s, s=p00
init=525s
cnt1=64s
cnt*=2350s
filbuf=830s
sort=1735s
write=872s
p = 6, t=1304.0s, s=p00
init=934s
cnt1=66s
cnt*=2464s
filbuf=875s
sort=1855s
write=1051s
p = 8, t=1049.0s, s=p00
init=1305s
cnt1=68s
cnt*=2476s
filbuf=904s
sort=1874s
write=1169s
p = 12, t=898.0s, s=p00
init=2327s
cnt1=77s
cnt*=2859s
filbuf=992s
sort=2115s
write=1554s
p = 16, t=804.0s, s=p00
init=3187s
cnt1=79s
cnt*=2878s
filbuf=1079s
sort=2177s
write=2230s
p = 20, t=826.0s, s=p00
init=4179s
cnt1=81s
cnt*=2948s
filbuf=1083s
sort=2209s
write=3777s
p = 24, t=772.0s, s=p00
init=5194s
cnt1=87s
cnt*=3086s
filbuf=1221s
sort=2363s
write=4773s
p = 28, t=801.0s, s=p00
init=6405s
cnt1=93s
cnt*=3233s
filbuf=1412s
sort=2476s
write=5579s
p = 32, t=790.0s, s=p00
init=7492s
cnt1=96s
cnt*=3389s
filbuf=1712s
sort=2709s
write=6427s

Hypotheis 2

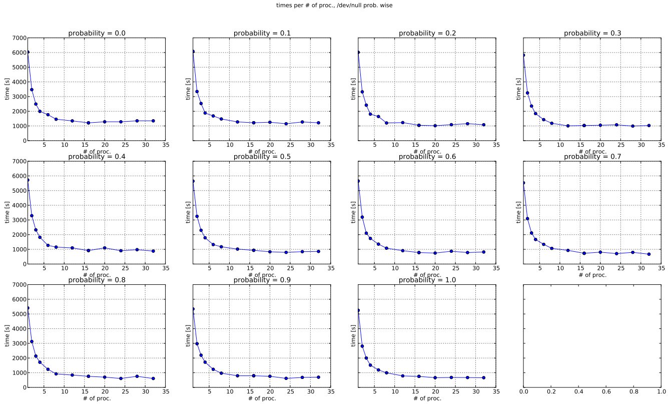
Observation 2: The amount of time for writting decreased, but as the number of cores grows, it is still substantial.

Hypotheis 2

Observation 2: The amount of time for writting 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 writting 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
time [s]
# of proc.

Hypotheis 3

Observation 3: Things are getting better, but there is still an increase in time when the number of cores is increased.

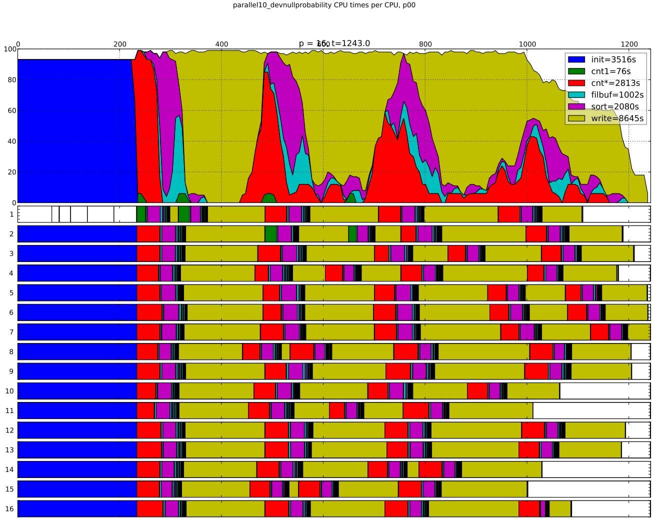
Hypotheis 3

Observation 3: Things are getting better, but there is still an increase in time when the number of cores is increased.

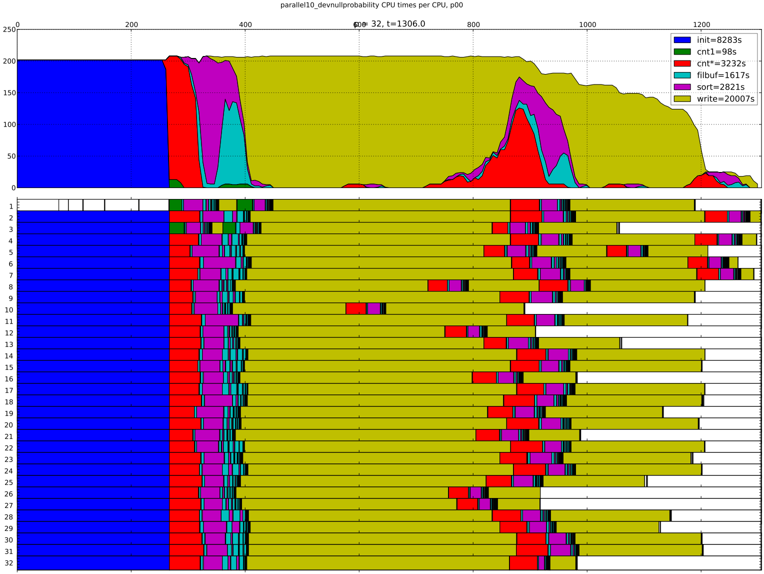
Hypothesis 3: ??

Check in more detail what the processes are doing.

Results – 5 ($p = 16$)



Results – 5 ($p = 32$)



Conclusion

- There is a substantial gap between theoretical results and practically used solutions.
- ERa despite being practically the fastest algorithm is **not theoretically tight** – even for random input strings with uniform substring distribution.

Conclusion

- There is a substantial gap between theoretical results and practically used solutions.
- 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.

Thanx for your attention!