Parallel Computation for the All-Pairs Suffix-Prefix Problem

Felipe A. Louza[1] Simon Gog[2] Leandro Zanotto[1]
Guido Araujo[1] Guilherme P. Telles[1]

[1]Institute of Computing, UNICAMP, Brazil
[2]Institute of Theoretical Informatics, KIT, Germany

SPIRE'16
Beppu, Japan

Outline

1. Introduction
2. Preliminaries
3. Related work
4. Parallel Algorithm
5. Experiments
6. Conclusion

Introduction

All-pairs suffix-prefix matching problem (APSP):

- Given a collection of strings $\mathcal{S} = S^1, S^2, \ldots, S^m$, and a threshold τ.
- APSP is to find, for all pairs S^i and S^j, the longest suffix of S^i that overlaps S^j that is larger than τ.
- Suffix-prefix match (overlap):

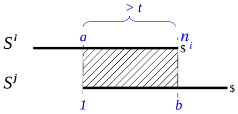


Figure: S^i overlaps S^j, the suffix $S^i[a, n_i - 1]$ is equal to the prefix $S^j[1, b]$

Motivation:

- DNA assembly (bottleneck stage).
- EST clustering.
- Approximating the shortest common superstring.

Introduction

Practical Algorithms:

- DNA assemblers provide fast solutions with non-optimal algorithms.
- SGA [Simpson and Durbin, 2010] and Readjoiner [Gonnella and Kurtz, 2012] have isolated overlap detection stage.
- [Rachid and Malluhi, 2015] presented SOF to solve the APSP for DNA sequences. SOF has a good performance with multiple threads[1].

Optimal Algorithms:

- [Gusfield et al., 1992] solved the APSP in optimal time $\rightarrow$ suffix trees and stacks.
- [Ohlebusch and Gog, 2010] improved memory usage and running time $\rightarrow$ enhanced suffix arrays and stacks.
- [Tustumi et al., 2016] proposed a different traversal of the enhanced suffix array and replaced stacks by linked lists to achieve a better practical running time.

[1] SGA, Readjoiner and SOF may be executed in multithreading environments.

Introduction

Our proposal:

- We showed how to parallelize the optimal algorithm by [Tustumi et al., 2016] to solve the APSP.
- Our parallel algorithm achieves a consistent speedup with a small memory footprint when compared with [Tustumi et al., 2016].
- Also, it is competitive with SOF when the threshold τ is small.

Workflow:

- Our algorithm is composed by three phases.
- We separated the computation of the local solution followed by the global solution and by the identical suffixes searching into independent phases.

Outline

1. Introduction

2. Preliminaries

3. Related work

4. Parallel Algorithm

5. Experiments

6. Conclusion

Preliminaries:

Let $S[1,n]$ be a string of length $|S| = n$ over an ordered alphabet Σ.

- A prefix is a substring of the form $S[1,i]$.
- A suffix is a substring $S[i,n]$ that will be denoted by S_i.

The suffix array of $S[1,n]$, SA, is an array of integers in $[1,n]$ that gives the lexicographic order of all suffixes:

- $S_{\mathrm{SA}[1]} < S_{\mathrm{SA}[2]} < \ldots < S_{\mathrm{SA}[n]}$;
- We denote the position of S_i in SA as $\mathrm{pos}(S_i)$.

The LCP-array stores the length of the longest common prefix (*lcp*) of two consecutive suffixes in SA:

- $\mathrm{LCP}[i] = lcp(S_{\mathrm{SA}[i]}, S_{\mathrm{SA}[i-1]})$ for $1 < i \leq n$, and
- $\mathrm{LCP}[1] = 0$.

The range minimum query (*rmq*) on LCP gives the smallest *lcp* value in an interval of SA:

- $rmq(i,j) = \min_{i<k\leq j}\{\mathrm{LCP}[k]\}$.
- It is well known that $lcp(S_{\mathrm{SA}[i]}, S_{\mathrm{SA}[j]}) = rmq(i,j)$, with $1 \leq i < j \leq n$.

Preliminaries:

Let $\mathcal{S} = S^1, S^2, \ldots, S^m$ be a collection of strings of lengths $n_i = |S^i|$.

- $S^{cat} = S^1 \cdot \$_1 \cdot S^2 \cdot \$_2 \cdots S^m \cdot \$_m$ is the concatenated string of length $N = m + \Sigma_{i=1}^{m} n_i$.
- Each $\$_i$ is a distinct separator not in Σ that precedes $\forall \alpha \in \Sigma$, and $\$_i < \$_j$ if $i < j$.
- $S_k^{\$}$ denotes the prefix of S_k^{cat} that ends at the first separator $\$_j$.

The generalized suffix array of $\mathcal{S}$, GSA, is the SA of the concatenated string S^{cat}.

For a clearer notation, we introduce:

- STR indicates which string in $\mathcal{S}$ a suffix came from, $\mathrm{STR}[i] = j$ if $S_{\mathrm{SA}[i]}^{\$}$ ends with $\$_j$.
- SA′ holds the position of a suffix with respect to the string it came from (up to the separator), defined as $\mathrm{SA}'[i] = k$ if $S_{\mathrm{SA}[i]}^{\$} = S_k^j \$_j$.

GESA denotes the GSA enhanced with the arrays STR, SA′ and LCP.

Preliminaries:

Let S^k be the j-th (lexicographically) smallest string in $\mathcal{S}$.

- P is an array of $m+1$ integers that stores in $\mathrm{P}[j] = \mathrm{pos}(S^k[1, n_k])$.
- We define $\mathrm{P}[0] = m + 1$.

Let $B^j = (\mathrm{P}[j-1], \mathrm{P}[j]]$[2] be a block of GESA corresponding to S^k.

- GESA can be partitioned into m blocks $B^1, B^2, \ldots, B^m$, one for each string S^k in $\mathcal{S}$.

	i	SA	LCP	STR	SA′	$S^\$_{\mathrm{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

Figure: GESA of $\mathcal{S} = \{\mathrm{aac}, \mathrm{aca}, \mathrm{aa}, \mathrm{caa}\}$. Suffixes in block 1 are highlighted.

[2] open interval: $\mathrm{P}[j-1]+1$ stores the $lcp(S_{\mathrm{SA}[\mathrm{P}[j-1]+1]}, S_{\mathrm{SA}[\mathrm{P}[j-1]]})$

Outline

1. Introduction

2. Preliminaries

3. Related work

4. Parallel Algorithm

5. Experiments

6. Conclusion

Related work:

The algorithm by [Tustumi et al., 2016] solves the APSP in optimal $O(N + m^2)$ time, based on the following remarks:

- All suffixes that overlap S^k are in positions prior to pos(S^k) or are identical to S^k and directly succeed pos(S^k).
- If two different suffixes of S^r are a prefix of S^k, the longest is closer to pos(S^k).
- Given two prefixes $S^t < S^k$, if a suffix of S^r, of length ℓ, overlap S^t and $\ell > lcp(S^t, S^k)$, then such suffix does not overlap S^k.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathrm{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

Figure: GESA of $\mathcal{S} = \{\mathrm{aac}, \mathrm{aca}, \mathrm{aa}, \mathrm{caa}\}$.

Related work:

The algorithm by [Tustumi et al., 2016] solves the APSP in optimal $O(N + m^2)$ time, based on the following remarks:

- All suffixes that overlap S^k are in positions prior to pos(S^k) or are identical to S^k and directly succeed pos(S^k).
- If two different suffixes of S^r are a prefix of S^k, the longest is closer to pos(S^k).
- Given two prefixes $S^t < S^k$, if a suffix of S^r, of length ℓ, overlap S^t and $\ell > lcp(S^t, S^k)$, then such suffix does not overlap S^k.

	i	SA	LCP	STR	SA′	$S^{\$}_{SA[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

Figure: GESA of $\mathcal{S} = \{\text{aac}, \text{aca}, \text{aa}, \text{caa}\}$.

Related work:

The algorithm by [Tustumi et al., 2016] solves the APSP in optimal $O(N + m^2)$ time, based on the following remarks:

- All suffixes that overlap S^k are in positions prior to pos(S^k) or are identical to S^k and directly succeed pos(S^k).
- If two different suffixes of S^r are a prefix of S^k, the longest is closer to pos(S^k).
- Given two prefixes $S^t < S^k$, if a suffix of S^r, of length ℓ, overlap S^t and $\ell > lcp(S^t, S^k)$, then such suffix does not overlap S^k.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathrm{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

Figure: GESA of $\mathcal{S} = \{\mathrm{aac}, \mathrm{aca}, \mathrm{aa}, \mathrm{caa}\}$.

Related work:

Algorithm [Tustumi et al., 2016]:

- The blocks are processed in order, $B^1, B^2, \ldots, B^m$.
- For each B^j: suppose that $\text{pos}(S^k) = \text{P}[j]$ and $\text{pos}(S^t) = \text{P}[j-1]$.
  1. Local solution is found scanning B^j backwards;
  2. Global solution is obtained reusing the local solutions of the previous blocks.
  3. Identical suffixes are found scanning GESA forward from $\text{P}[j]+1$.
- The algorithm uses m local lists and m global lists to track all overlaps seen so far.

	i	SA	LCP	STR	SA′	$S^{\$}_{\text{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] $\rightarrow$	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] $\rightarrow$	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[$j-1$] $\rightarrow$	10	1	2	1	1	aac$\$_1$ $\leftarrow S^t$
	11	2	1	1	2	ac$\$_1$
P[j] $\rightarrow$	12	5	2	2	1	aca$\$_2$ $\leftarrow S^k$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] $\rightarrow$	15	12	2	4	1	caa$\$_4$

Figure: GESA of $\mathcal{S} = \{\text{aac}, \text{aca}, \text{aa}, \text{caa}\}$.

Related work:

Algorithm [Tustumi et al., 2016]:

- Local solution:
  - For block B^j:
  - GESA is scanned backwards, from $\mathsf{P}[j]$ to $\mathsf{P}[j-1]+1$.
  - $\ell = rmq(i, \mathsf{P}[j])$ is computed in $O(1)$ time during the scanning of B^j.
    - if $|S^{\mathsf{STR}[i]}_{\mathsf{SA}'[i]}| = \ell$ then $S^{\mathsf{STR}[i]}$ overlaps S^k in ℓ symbols, and $insert\_at\_end(L_{local}[\mathsf{STR}[i]], \ell)$.
    - At the end, the longest overlaps in B^j are at the front of each local list.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathsf{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

i	$L_{local}[1]$	$L_{local}[2]$	$L_{local}[3]$	$L_{local}[4]$
8	[ ]	[ ]	[ ]	[ ]
7	[ ]	[ ]	[ ]	[1]
6	[ ]	[ ]	[1]	[1]
5	[ ]	[1]	[1]	[1]

Figure: GESA of $\mathcal{S} = \{\mathrm{aac}, \mathrm{aca}, \mathrm{aa}, \mathrm{caa}\}$.

Related work:

Algorithm [Tustumi et al., 2016]:

- Global solution:
  - The longest suffix of S^r that overlaps S^k may be positioned in a previous block.
  - The global lists store these overlaps.
  - Each $L_{global}[r]$ is updated as each block is processed.
    - First, we remove suffixes larger than $lcp(S^t, S^k)$ from local lists.
    - $L_{local}[r]$ is prepended in the global list $L_{global}[r]$.
    - The first element of each $L_{global}[r]$ is the longest overlap of S^r to S^k, that is Ov$[k, r]$.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathrm{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

i	$L_{local}[1]$	$L_{local}[2]$	$L_{local}[3]$	$L_{local}[4]$
12	[ ]	[ ]	[ ]	[ ]
11	[2]	[ ]	[ ]	[ ]
10	[2]	[ ]	[ ]	[ ]

$P[j-1]$	$L_{global}[1]$	$L_{global}[2]$	$L_{global}[3]$	$L_{global}[4]$
5	[ ]	[1]	[1]	[1]
8	[ ]	[1]	[2,1]	[2,1]
10	[2]	[1]	[1]	[1]

Figure: GESA of $\mathcal{S} = \{\mathrm{aac}, \mathrm{aca}, \mathrm{aa}, \mathrm{caa}\}$.

Related work:

Algorithm [Tustumi et al., 2016]:

- Identical suffixes:
  - We scan GESA forward from $i = \mathrm{P}[j] + 1$ to q, while $\mathrm{LCP}[q] = n_k$.
  - The length of these suffixes are inserted in Ov, possibly overwriting the results in $L_{global}[r]$, which is correct as such overlaps are larger.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathrm{SA}[i]}$
	1	4	0	1	4	$1
	2	8	0	2	4	$2
	3	11	0	3	3	$3
	4	15	0	4	4	$4
P[0] →	5	7	0	2	3	a$2
	6	10	1	3	2	a$3
	7	14	1	4	3	a$4
P[1] →	8	9	1	3	1	aa$3
	9	13	2	4	2	aa$4
P[2] →	10	1	2	1	1	aac$1
	11	2	1	1	2	ac$1
P[3] →	12	5	2	2	1	aca$2
	13	3	0	1	3	c$1
	14	6	1	2	2	ca$2
P[4] →	15	12	2	4	1	caa$4

$$Ov[m, m]^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & & 1 & 2 & 2 \\ 2 & 2 & & 1 & 1 \\ 3 & & 1 & & 2 \\ 4 & 1 & 2 & & \end{array}$$

Figure: GESA of $\mathcal{S} = \{\mathrm{aac}, \mathrm{aca}, \mathrm{aa}, \mathrm{caa}\}$.

[3] $\mathrm{Ov}[k, r]$ the length of the longest suffix of S^r that overlaps S^k.

Outline

1. Introduction

2. Preliminaries

3. Related work

4. Parallel Algorithm

5. Experiments

6. Conclusion

Parallel Algorithm:

We split the computation of all overlaps to solve the APSP in parallel:

At a glance, our algorithm is composed by:

1. Compute all local solutions scanning the blocks B^j concurrently.
2. Compute all global solutions accessing the local lists in parallel.
3. Compute all identical suffixes of all strings scanning the GESA in parallel.

Algorithm: Suppose that for each block B^j, $pos(S^k) = \mathrm{P}[j]$ and $\mathrm{pos}(S^t) = \mathrm{P}[j-1]$.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathrm{SA}[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[$j-1$] →	10	1	2	1	1	aac$\$_1$ ← S^t
	11	2	1	1	2	ac$\$_1$
P[j] →	12	5	2	2	1	aca$\$_2$ ← S^k
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

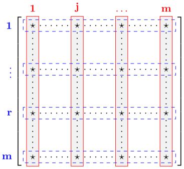
Figure: GESA of $\mathcal{S} = \{\mathtt{aac}, \mathtt{aca}, \mathtt{aa}, \mathtt{caa}\}$.

Parallel Algorithm:

1. Local solutions:
    - Scan all the m blocks in parallel.
        - Each block B^j is scanned backwards.
        - Whenever $|S^{r=\mathrm{STR}[i]}_{\mathrm{SA}'[i]}| = rmq(\mathrm{P}[j], i)$[4] then a suffix of S^r overlaps S^k.
    - We store the overlaps at a squared matrix of local lists:
        - $L_{local}[r][j]$ stores the length of a suffix of S^r that overlaps S^k.
        - The overlaps are ordered decreasingly by their lengths due to the backward scan.
    - We compute $\mathrm{Min}[j] = rmq(\mathrm{P}[j-1], \mathrm{P}[j]) = lcp(S^k, S^t)$
    - At the end, all local solutions have been computed and are stored into the local lists.

	i	SA	LCP	STR	SA′	$S^{\$}_{\mathrm{SA}[i]}$	
	1	4	0	1	4	$\$_1$	
	2	8	0	2	4	$\$_2$	
	3	11	0	3	3	$\$_3$	
	4	15	0	4	4	$\$_4$	
P[0] →	5	7	0	2	3	a$\$_2$	
	6	10	1	3	2	a$\$_3$	
	7	14	1	4	3	a$\$_4$	
P[1] →	8	9	1	3	1	aa$\$_3$	
	9	13	2	4	2	aa$\$_4$	
$\mathrm{P}[j-1]$ →	10	1	2	1	1	aac$\$_1$	$\leftarrow S^t$
	11	2	1	1	2	ac$\$_1$	
$\mathrm{P}[j]$ →	12	5	2	2	1	aca$\$_2$	$\leftarrow S^k$
	13	3	0	1	3	c$\$_1$	
	14	6	1	2	2	ca$\$_2$	
P[4] →	15	12	2	4	1	caa$\$_4$	

	1	2	3	4
Min =	0	2	1	0



Local lists, columns are being accessed in parallel

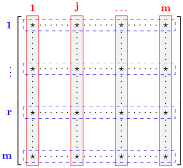
[4] The *rmqs* are solved as LCP is scanned, ℓ starts with ∞ and $\ell = min(\ell, \mathrm{LCP}[i+1])$;.

Parallel Algorithm:

2. Global solutions:
   - We process the m lines of matrix L_{local} in parallel.
     - Each local list $L_{local}[r][j]$ corresponds to the suffixes of S^r that overlap S^k in B^j.
   - For each line r (in parallel):
     - We process lists $L_{local}[r][j]$ in order $j = 1, 2, \ldots, m$.
     - We use a global list L_{Global} that is updated according to $\text{Min}[j] = lcp(S^t, S^k)$.
     - L_{global} is initially empty and for each $j = 1, 2, \ldots, m$, all suffixes larger than $\text{Min}[j]$ are removed, these suffixes are no longer overlaps for the next blocks.
     - $L_{local}[r][j]$ is prepended to L_{global}
     - $\text{Ov}[r, k] \leftarrow first(L_{global})$[5].

	i	SA	LCP	STR	SA′	$S^{\$}_{SA[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[$j-1$] →	10	1	2	1	1	aac$\$_1$ $\leftarrow S^t$
	11	2	1	1	2	ac$\$_1$
P[j] →	12	5	2	2	1	aca$\$_2$ $\leftarrow S^k$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

	1	2	3	4
Min =	0	2	1	0



Local lists, lines are being accessed in parallel

[5] The longest suffix of S^r that overlaps S^k will be the first element of L_{global}.

Parallel Algorithm:

3. Identical suffixes:
   - Scan all the end of blocks B^j concurrently.
   - For each block B^j and its corresponding string S^k:
     - All suffixes of S^r identical to S^k appear directly after pos(S^k).
     - We scan from $P[j]+1$ while the next suffixes have the same length of S^k.
     - $Ov[r,k] \leftarrow LCP[i]$[6].

	i	SA	LCP	STR	SA′	$S^{\$}_{SA[i]}$
	1	4	0	1	4	$\$_1$
	2	8	0	2	4	$\$_2$
	3	11	0	3	3	$\$_3$
	4	15	0	4	4	$\$_4$
P[0] →	5	7	0	2	3	a$\$_2$
	6	10	1	3	2	a$\$_3$
	7	14	1	4	3	a$\$_4$
P[1] →	8	9	1	3	1	aa$\$_3$
	9	13	2	4	2	aa$\$_4$
P[2] →	10	1	2	1	1	aac$\$_1$
	11	2	1	1	2	ac$\$_1$
P[3] →	12	5	2	2	1	aca$\$_2$
	13	3	0	1	3	c$\$_1$
	14	6	1	2	2	ca$\$_2$
P[4] →	15	12	2	4	1	caa$\$_4$

[6]Different from [Tustumi et al., 2016] $Ov[r,k]$ the length of the longest suffix of S^r that overlaps S^k.

Parallel Algorithm:

Theoretical costs:

- Time:
  - Our algorithm runs in $O(N + m^2/t)$ time, where t is the number of threads.
    - Local solution: $O(\max_{1 \leq j \leq m} |B^j|)$ to scan all blocks.
    - Global solution: $O(m^2/t)$ to access all local lists.
    - Identical suffixes: $O(N)$, since it reads at most N elements of GESA.
  - The worst case happens only when the string lengths are very unbalanced.
  - In realistic cases ($m \gg t$ and all strings with about the same size) the parallel time is close to $N/t + m^2/t$.
- Space:
  - Our algorithm uses $O(N + m^2)$ of space.
    - which is equal to the sequential algorithm by [Tustumi et al., 2016].
  - The space is given by the $O(N)$ space of the GESA, and by the $O(m^2 + N)$ space of the matrix of local lists, where each list stores at most N overlaps.

Outline

1. Introduction

2. Preliminaries

3. Related work

4. Parallel Algorithm

5. Experiments

6. Conclusion

Experiments:

Implementation:

- C++ using OpenMP.
- sdls-lite v.2 to construct the GESA.
- Source code: `https://github.com/felipelouza/p-apsp`.

Comparison:

- p-apsp: our parallel algorithm.
- apsp: sequential optimal algorithm by [Tustumi et al., 2016].
- SOF: practical (non-optimal) solution by [Rachid and Malluhi, 2015].

Number of threads:

- $t = \{1, 2, 4, 8, 16, 32\}$
- set by `omp_set_num_threads()` for p-apsp and SOF

Configuration:

- 64 bits Debian GNU/Linux 8 (kernel 3.16.0-4)
- Intel Xeon Processor E5-2630 v3 20M Cache 2.40-GH
- 384 GB of RAM.

Experiments:

Dataset:

- We used $m = 300.000$ strings from EST database of *C. elegans*[7].

Minimum overlap length (τ):

- We limited the number of overlaps by $\tau = \{5, 10, 15, 20\}$

Table: Number of overlaps found with 100.000, 200.000 and 300.000 ESTs varying τ.

m/τ	5	10	15	20
100.000	18,853,491	206,154	88,725	82,427
200.000	71,451,170	2,675,759	2,139,431	2,077,125
300.000	162,135,112	7,044,274	5,800,397	5,617,779

Observation:

- The number of overlaps decreases as τ increases.
- For 300.000, when $\tau = 5$ the number of overlaps is 23 times larger than when $\tau = 10$.
- Such variation impacts the performance of all algorithms.

[7] `http://www.uni-ulm.de/in/theo/research/seqana.html`

Experiments:

Running Time[8]:

- SOF was the fastest in all experiments.
- p-apsp has shown a good performance for small threshold τ.
- There is an overhead with $\tau = 5$ when comparing p-apsp$_1$ to apsp.

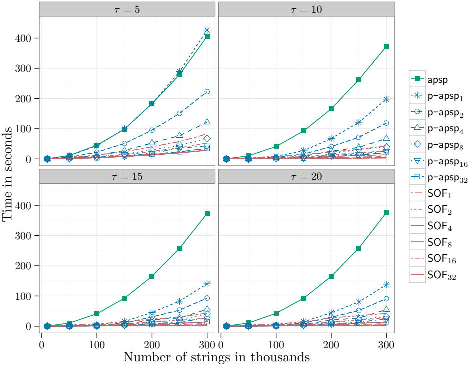


Figure: Running time for varying values of τ (`omp_get_wtime()`).

[8] not accounting for the time to build the GESA for apsp and p-apsp, and compact prefix tree for SOF.

Experiments:

Speedup:

- We evaluated the speedup of p−apsp and SOF over its serial versions.
- p−apsp and SOF improve as the number of threads increases.
- p−apsp achieved a better speedup with the increasing number of threads.

Table: Experiments with 300.000 ESTs with $\tau = 5$. The table shows the running time in seconds.

	apsp	p−apsp		SOF	
n. threads	time	time	speedup	time	speedup
1	397.17	463.33		**82.80**	
2		222.78	**1.91**	**52.49**	1.58
4		121.35	**3.51**	**29.50**	2.81
8		68.11	**6.26**	**28.30**	2.93
16		43.41	**9.82**	**28.64**	2.89
32		34.65	**12.31**	**23.62**	3.50

Experiments:

Peak Memory:

- The memory usage decreases as τ increases (number of overlaps).
- SOF uses less memory in all experiments.
- p–apsp memory usage is very similar to apsp, differing only by a constant factor.

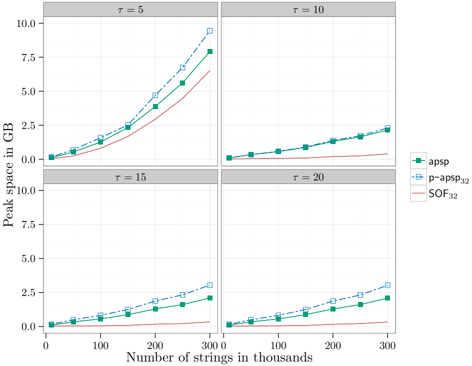


Figure: Running time for varying values of τ (`malloc_count library`).

Outline

1. Introduction

2. Preliminaries

3. Related work

4. Parallel Algorithm

5. Experiments

6. Conclusion

Conclusion:

Future works:

- Semi-external memory: The GESA can be constructed in external memory and its blocks can be accessed as necessary, reducing the peak memory.
- Different architecture model: such as cloud distributed computing, possibly enabling the usage of hundreds of threads.

Thank you!

Gonnella, G. and Kurtz, S. (2012).
Readjoiner: a fast and memory efficient string graph-based sequence assembler.
BMC bioinformatics, 13(1):82.

Gusfield, D., M. Landau, G., and Schieber, B. (1992).
An efficient algorithm for the all pairs suffix-prefix problem.
Information Processing Letters, 41(4):181–185.

Ohlebusch, E. and Gog, S. (2010).
Efficient algorithms for the all-pairs suffix-prefix problem and the all-pairs substring-prefix problem.
Information Processing Letters, 110(3):123–128.

Rachid, M. H. and Malluhi, Q. (2015).
A practical and scalable tool to find overlaps between sequences.
BioMed Research International, 2015:1–12.

Simpson, J. T. and Durbin, R. (2010).
Efficient construction of an assembly string graph using the FM-index.
Bioinformatics, 26(12):i367–i373.

Tustumi, W. H., Gog, S., Telles, G. P., and Louza, F. A. (2016).
An improved algorithm for the all-pairs suffix-prefix problem.
Journal of Discrete Algorithms.