Parallel Distributed Memory String Indexes

Efficient Construction and Querying

Patrick Flick & Srinivas Aluru
Computational Science and Engineering
Georgia Institute of Technology

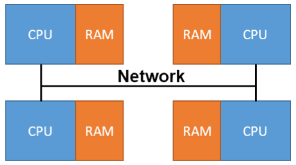
In this talk…

Georgia Institute of Technology

Overview

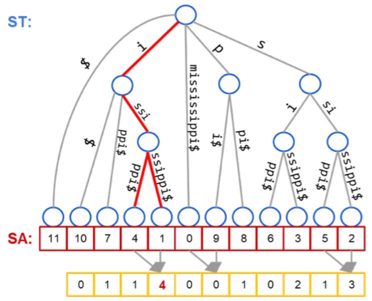
1

Very Short Intro to **Parallel** and **Distributed** Computing



3

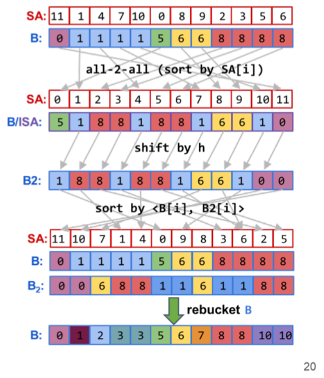
All-Nearest-Smaller-Values and Distributed Construction of **Suffix Trees**



[IPDPS '17]

2

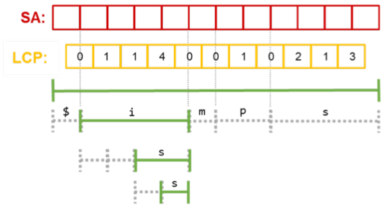
Distributed Construction of **Suffix Arrays** and **LCP Arrays**



[SC '15]

4

Distributed Enhanced Suffix Arrays

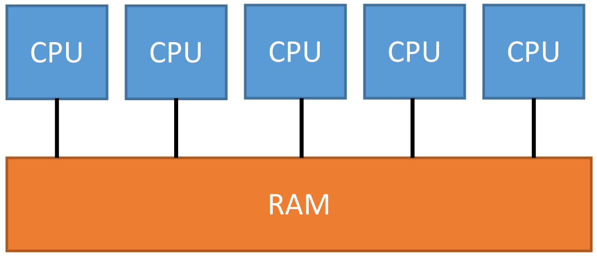


[Under Review]

Introduction: Parallel Models

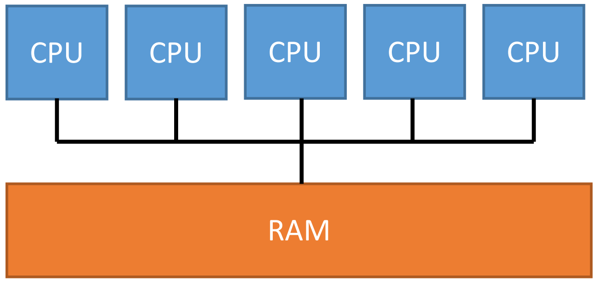
PRAM (Parallel Random Access Machine)

- Algorithms formulated using n processors for n input items, analyzed w.r.t.:
  - ***Work*** (total #ops by all processors)
  - ***Depth*** ($=$ time) (time steps till completion)
- Variants:
  - Exclusive Read Exclusive Write (EREW)
  - Concurrent Read Exclusive Write (CREW)
  - Concurrent Read Concurrent Write (CRCW)
- ***Not realistic***



Shared-Memory Parallel

- Parallel processors
- Sequential memory
- Programming via threads/processes
- Limited number of processors and RAM possible

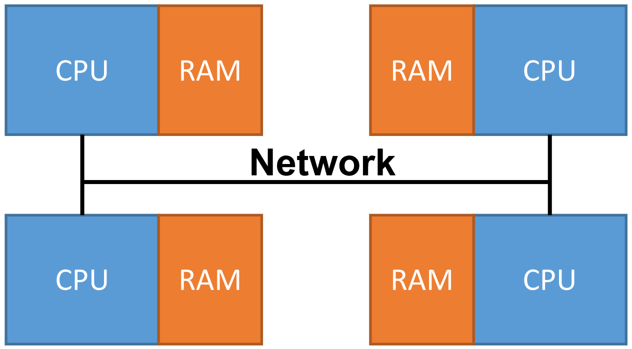


Introduction: Parallel Models

Distributed Parallel Model

- Distributed Memory
  - Can't directly access memory at remote processors
- Explicit communication via messages *(Message Passing Interface MPI)*
  - `send / receive` $O(\tau + \mu m)$
  - `all-to-all`
  - `(all-) reduce`
  - `prefix-sum` $O(\log(p)\,(\tau + \mu m))$
- Analysis with respect to:

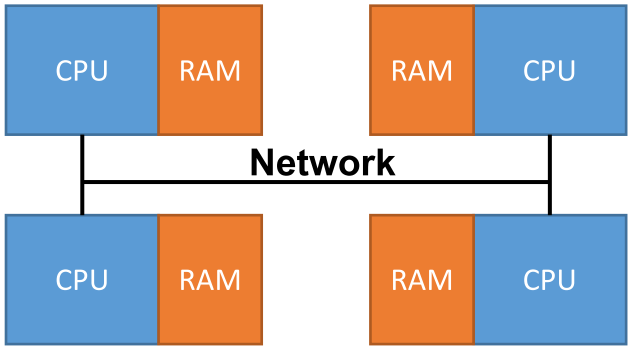
Input size	n
Number of processors	p
Latency	τ
Bandwidth	$1/\mu$

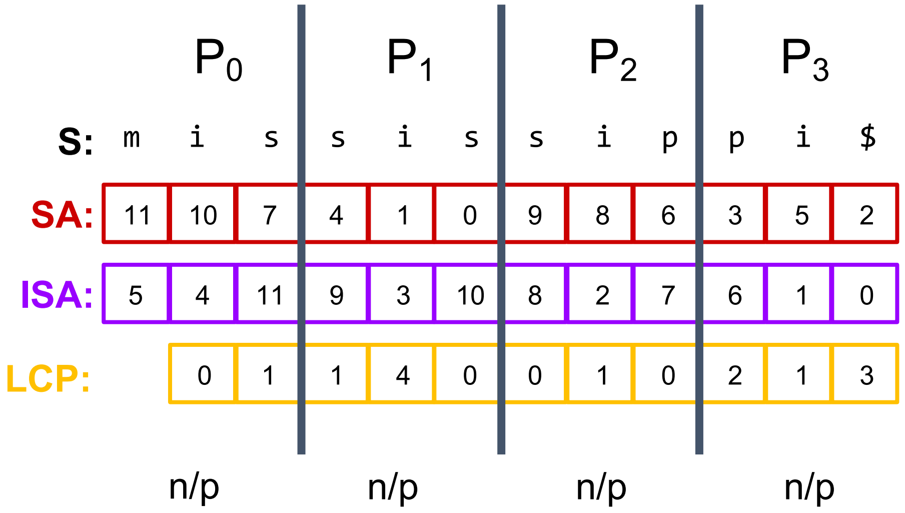


Introduction: Parallel Models

Distributed Parallel Model

- Usage of large compute clusters and distributed memory required when the problem
  a) Needs a large number of processors
  b) Needs a large cumulative size of memory
- Memory Scalability:
  - A problem of size n requires $O(n/p)$ memory per processor
- Arrays and data are *(equally) block distributed* across p processors
  - n/p elements per processor
  - constant time lookups:
    `index_range(rank)`
    `rank_of(global_index)`



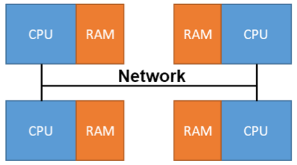


	P_0			P_1			P_2			P_3		
S:	m	i	s	s	i	s	s	i	p	p	i	$
SA:	11	10	7	4	1	0	9	8	6	3	5	2
ISA:	5	4	11	9	3	10	8	2	7	6	1	0
LCP:		0	1	1	4	0	0	1	0	2	1	3
	n/p			n/p			n/p			n/p		

Outline

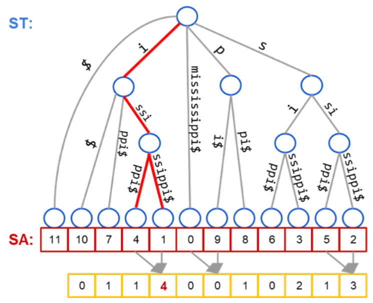
1

Very Short Intro to **Parallel** and **Distributed** Computing



3

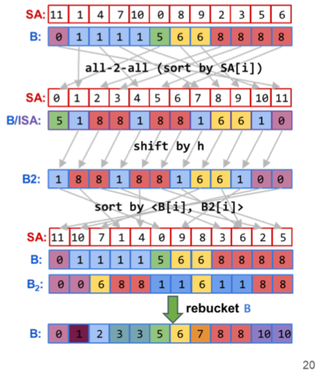
All-Nearest-Smaller-Values and Distributed Construction of **Suffix Trees**



[IPDPS '17]

2

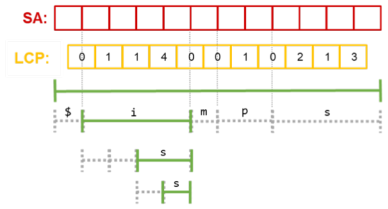
Distributed Construction of **Suffix Arrays** and **LCP Arrays**



[SC '15]

4

Distributed Enhanced Suffix Arrays



[Under Review]

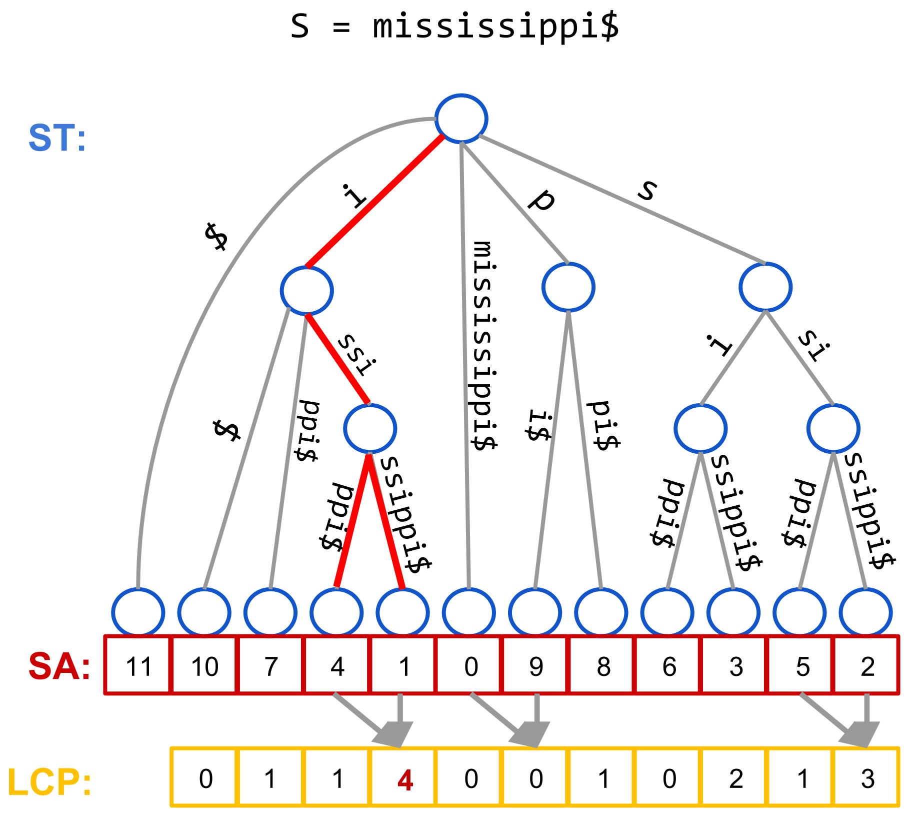
Motivation: String Indexing

- **Indexing** is required for fast **pattern searching & matching**
- **Structured texts** are "easy" to index
  - e.g. natural language, websites, documents, etc
- **Genomic sequences: unstructured texts**

  ctgccagtgagattatcggcctatatgcacactttggactaggaactaaat
- **Two major approaches:**
  1. Index target sequence by fixed size substrings**: k-mer index**
  2. Index all suffixes: **suffix arrays, suffix trees, FM index**

Introduction

- **Suffix Tree (ST)**
  - trie of all suffixes of a string
  - fundamental and powerful indexing structure
- **Suffix Array (SA)**
  - array of sorted suffixes
  - represents leafs of **ST**
- **Longest Common Prefix (LCP)**
  - length of prefix match between consecutive suffixes in **SA**
- **Important Applications:**
  - Approximate pattern matching, finding of longest common substrings, all-pair maximal overlaps, data compression



Suffix Arrays and Trees

Contributions:

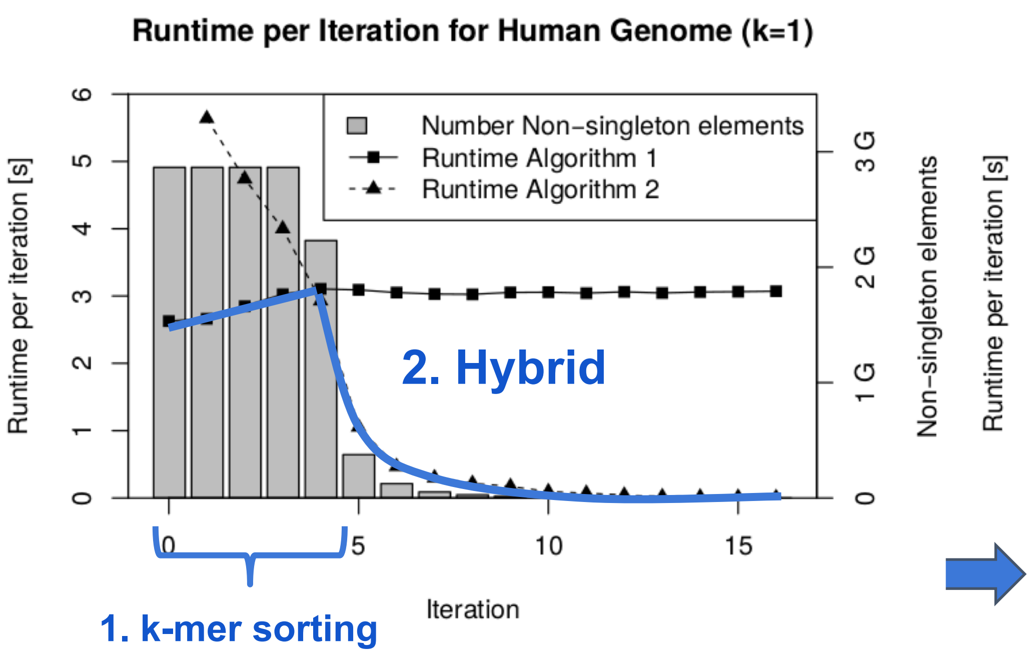
- Parallel Distributed Memory Construction of **Suffix Array,** **LCP Array**, and **Suffix Tree**
  - Indexing of Human Genome on 1024 Xeon cores in < 9.5s

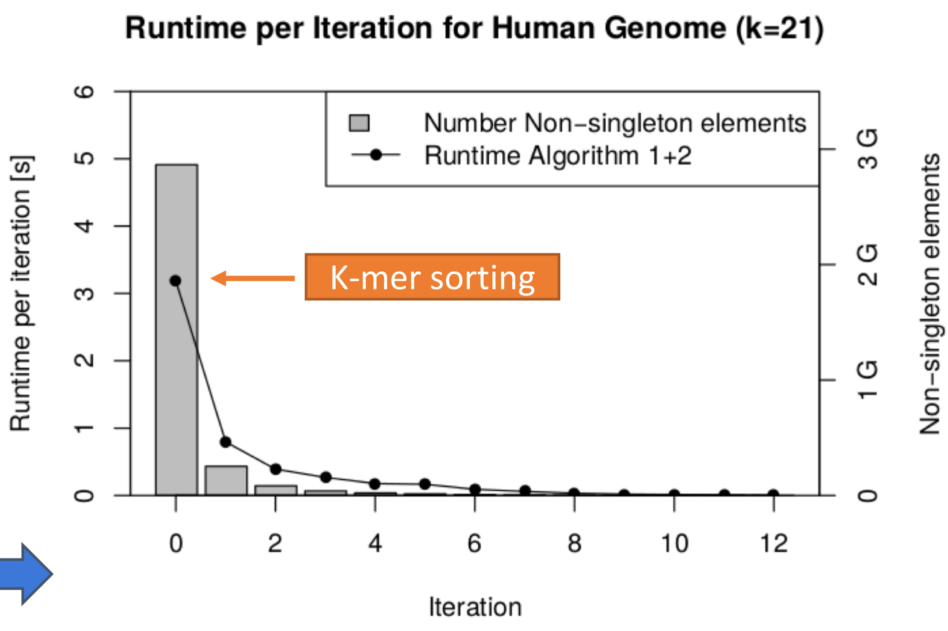
$$S \xrightarrow[\textit{7.5s}]{\textbf{[SC'15]}} \textbf{SA} + \textbf{LCP} \xrightarrow[\textit{1.7s}]{\textbf{[IPDPS'17]}} \textbf{ST}$$

  - Scalable to **large strings:** $O(n/p)$ memory per node
  - Better theoretical complexity than prior distributed memory algorithms or available shared memory implementations
  - Outperforms state-of-the-art in shared and distributed memory

Experiments and Results

Results SA Construction: Hybrid Algo1 + Algo2





Algorithm 1: Distributed Manber & Myers
Algorithm 2: Communication avoiding prefix-doubling

Hybrid: Introspectively switch between algorithms based on number of non-singleton buckets

Experiments and Results

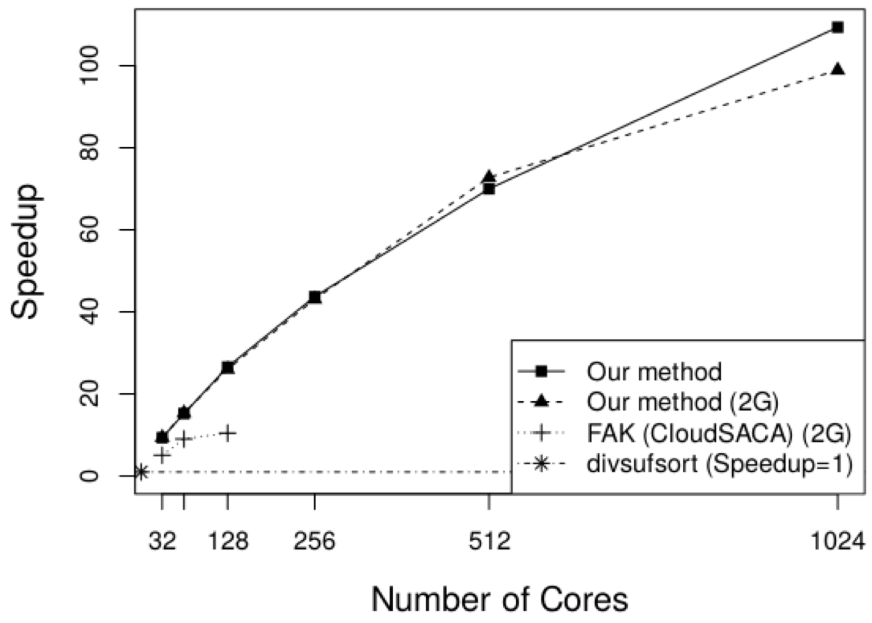
Georgia Institute of Technology

Results SA Construction

Runtime (seconds) of different methods

Method	H 2G	H 3G	P 12G
divsufsort	424.5	586.4	X
mkESA (1)	586.6	1,123	X
mkESA (4)	462.6	759	X
cloudSACA (128)	40.6	X	X
Our method (128)	16.3	22.1	142.6
Our method (1600)	**3.5**	**4.8**	**14.8**

Speedup over divsufsort



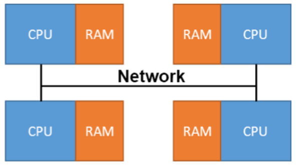
Experimental System:

- 100 nodes: 2x 8 core **Intel E5-2650**
- 128 GB RAM per node
- QDR Infiniband

Outline

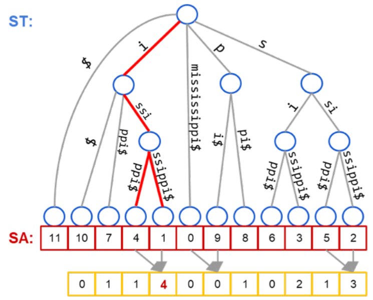
1

Very Short Intro to **Parallel** and **Distributed** Computing



3

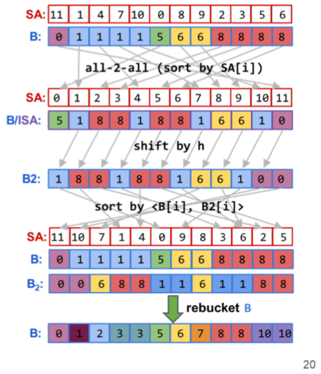
All-Nearest-Smaller-Values and Distributed Construction of **Suffix Trees**



[IPDPS '17]

2

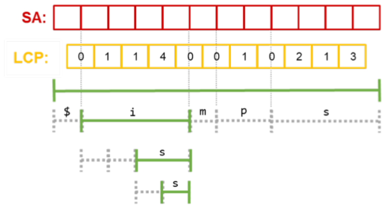
Distributed Construction of **Suffix Arrays** and **LCP Arrays**



[SC '15]

4

Distributed Enhanced Suffix Arrays



[Under Review]

Parallel Suffix Tree Construction

PRAM (n processor) ST Construction

- [Apostolico ‘88] CRCW O(log n) time O(n^2) space
- [Hariharan ‘94] CRCW work optimal for constant alphabets
- → theoretical importance, (probably) not practical

Distributed Memory

- [Ghoting ‘09] *Wavefront*
- [Mansour ‘11] *Elastic Range (ERA)*
- [Comin ‘13] *Parallel Continuous Flow (PCF)*
- → quadratic worst-case complexity, $O(n)$ memory per processor

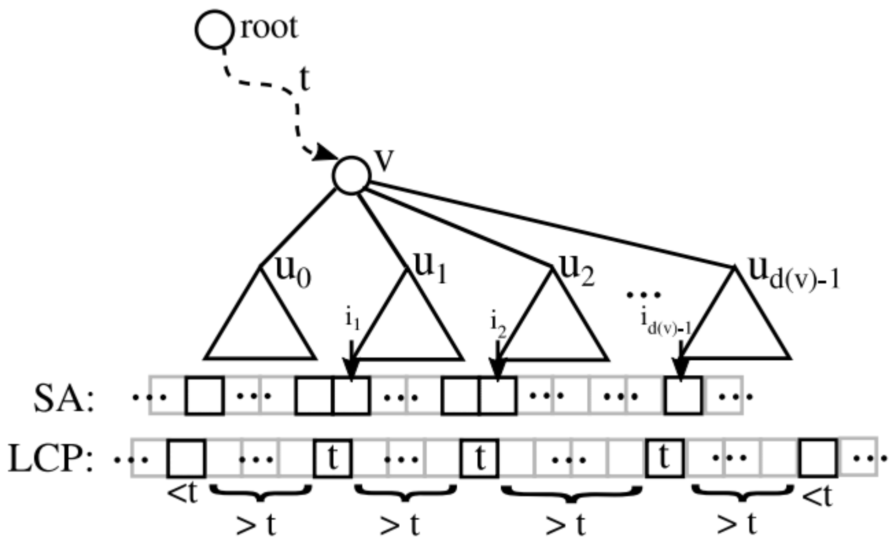
PRAM from SA+LCP

- [Iliopoulos ’04] CREW O(log n) time O(n log n) work
- [Shun ’14] EREW optimal O(n) work O($\log^2 n$) time,
  but O(n log n) work implementation

Parallel Suffix Tree Construction

ST Construction from SA and LCP

- **SA** = leafs of **ST**
- **LCP** = internal nodes of **ST**
- Any internal node v of depth t
  - defines a subtree of suffixes which share a prefix of length at least ***t***
  - **LCP** values for subtree range **>= *t***
  - if ***v*** has d children, ***d-1*** values are **= *t***
  - **LCP** values at borders **< *t***
  - **Parent** of ***v*** is one of the two border items

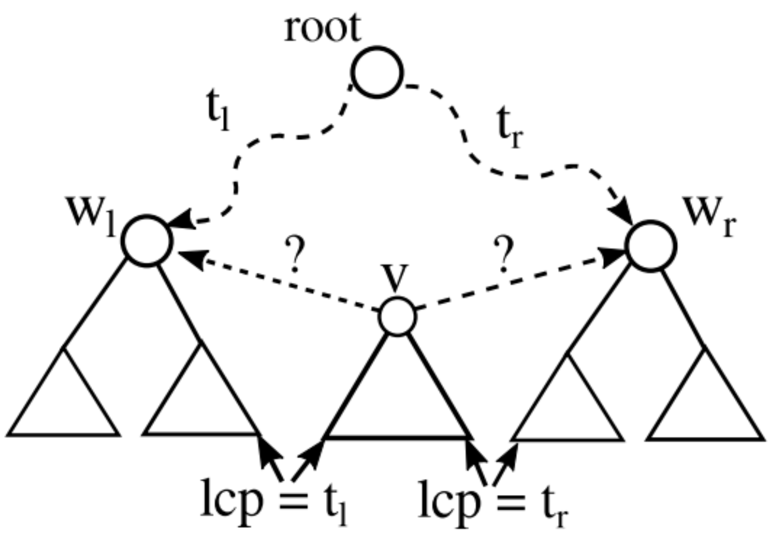


- Determine parent for each node

  for each `SA[i]`:
  - `max(LCP[i], LCP[i+1])`

  for each `LCP[i]`:
  - Find nearest smaller LCP[l] < LCP[i] to left l < i
  - Find nearest smaller LCP[r] < LCP[i] to right r > l
  - `max(LCP[j], LCP[h])`

=> All Nearest Smaller Values (ANSV)



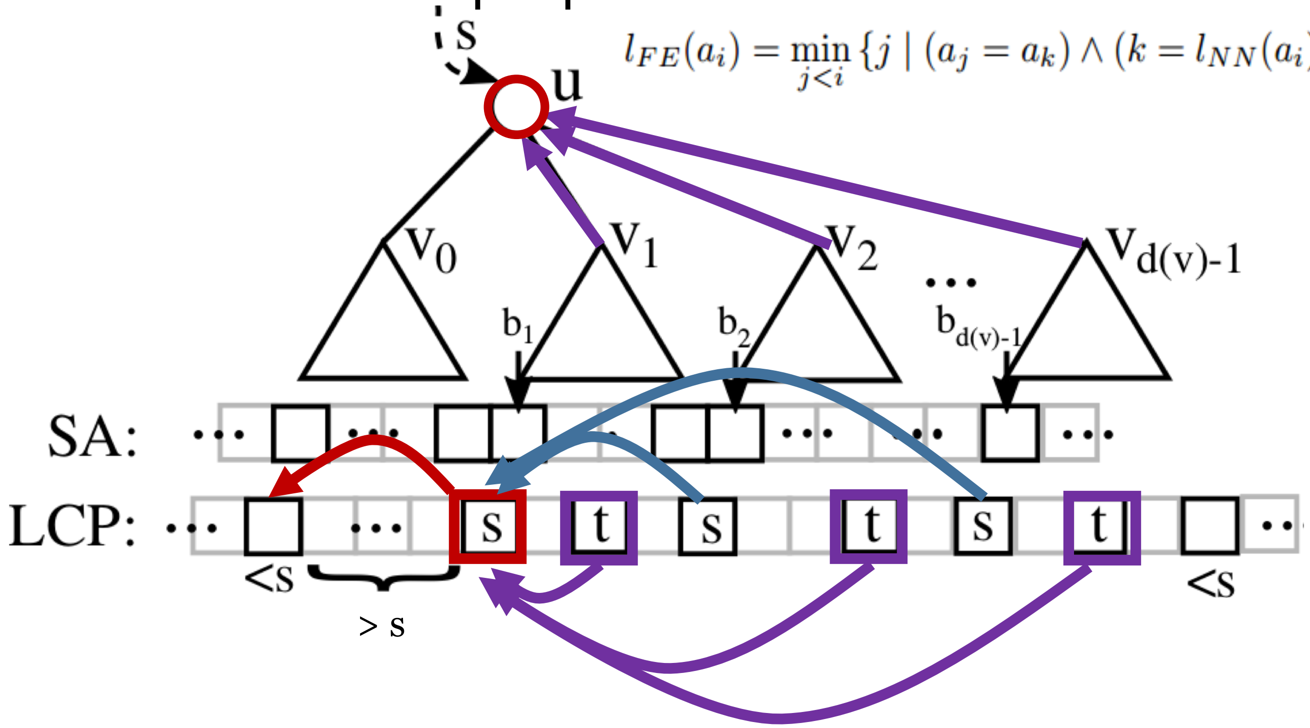
Parallel Suffix Tree Construction

Georgia Institute of Technology

All Nearest Smaller Values

- For each element in an array A
  - find nearest smaller: $l_{NS}(a_i) = \max_{j<i}\{j \mid a_j < a_i\}$
- Well studied problem with parallel solutions
  - however, with too restrictive assumptions
- Problem: which is the unique parent node? → **Leftmost of equal value**

$$l_{FE}(a_i) = \min_{j<i}\{j \mid (a_j = a_k) \wedge (k = l_{NN}(a_i)) \wedge (j > l_{NS}(a_k))\}$$

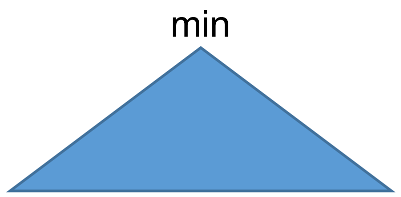


All-Nearest-Smaller-Values

All Nearest Smaller Values Parallel Algorithm

- Distributed memory: n/p elements per processor

1. Sequentially find matches locally:
   - Keep unmatched elements: bitonic sequence

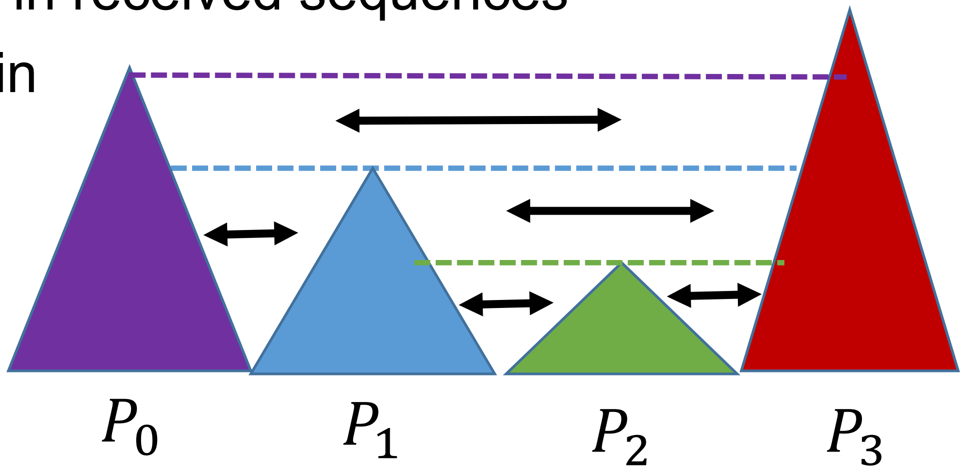


2. Allgather processor minimas m_i
3. Determine sections to exchange based on $(m_0, m_1, \dots, m_{p-1})$
4. Send / Receive sections
   - So that each processor sends/receives at most n/p elements
5. Solve unmatched elements in received sequences
6. Send solutions back to origin

Complexity

$O(\frac{n}{p} + p)$ time

$\mathbb{E}[comm] = \log(\frac{n}{p})$



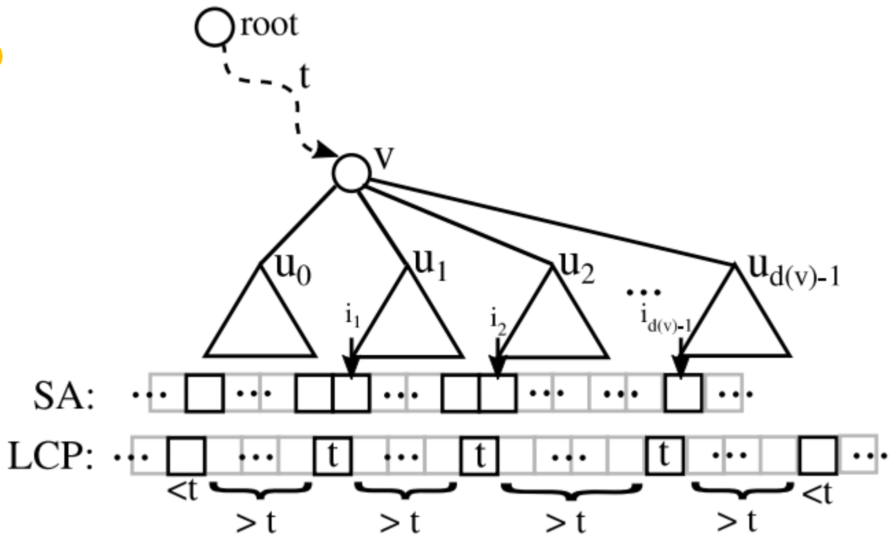
Parallel Suffix Tree Construction

ST Construction from SA and LCP

- **SA** = leafs of **ST**
- **LCP** = internal nodes of **ST**

Algorithm steps

- Construct **ST** "bottom-up"
- Each node determines its parent
  - Solve the ***gANSV*** problem on the **LCP**
  - Combine results from left and right matches
- Inverse edges *(i, parent(i))* and create internal nodes for each unique *parent(i)*
- Label each edge with its first character

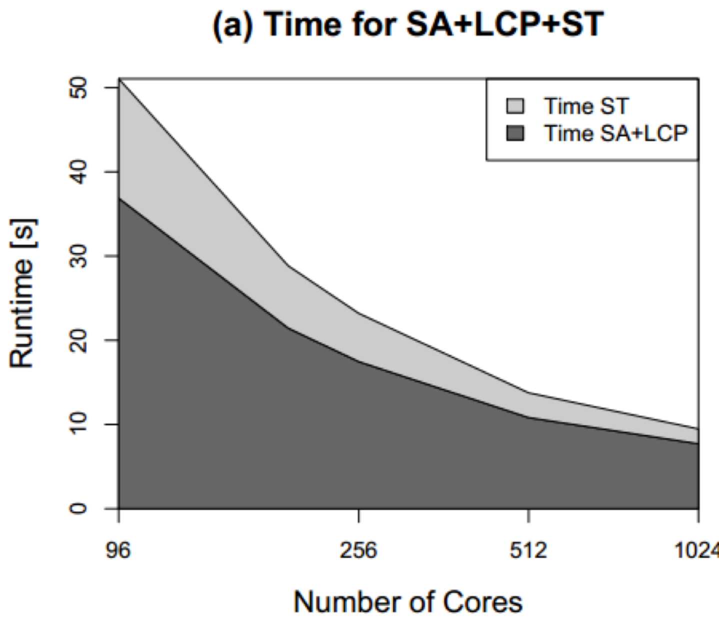


Experiments and Results

Results SA + LCP + ST Construction

Construction Time for Human Genome

Method	System	Cores	Time
WaveFront	IBM BG/L	1024	15 min
ERA	16x Intel 2-core nodes	32	13.7 min
PCF	MareNostrum	172	7 min
Shun	4x 10 core **Intel E7-8870**	40	168 s
Shun	4x 18 core **Intel E7-8870**	72	146 s
Our method	4x 18 core **Intel E7-8870**	72	**63** s
Our method	64 nodes: 2x 8 core **Intel E5-2650**	1024	**9.5 s**



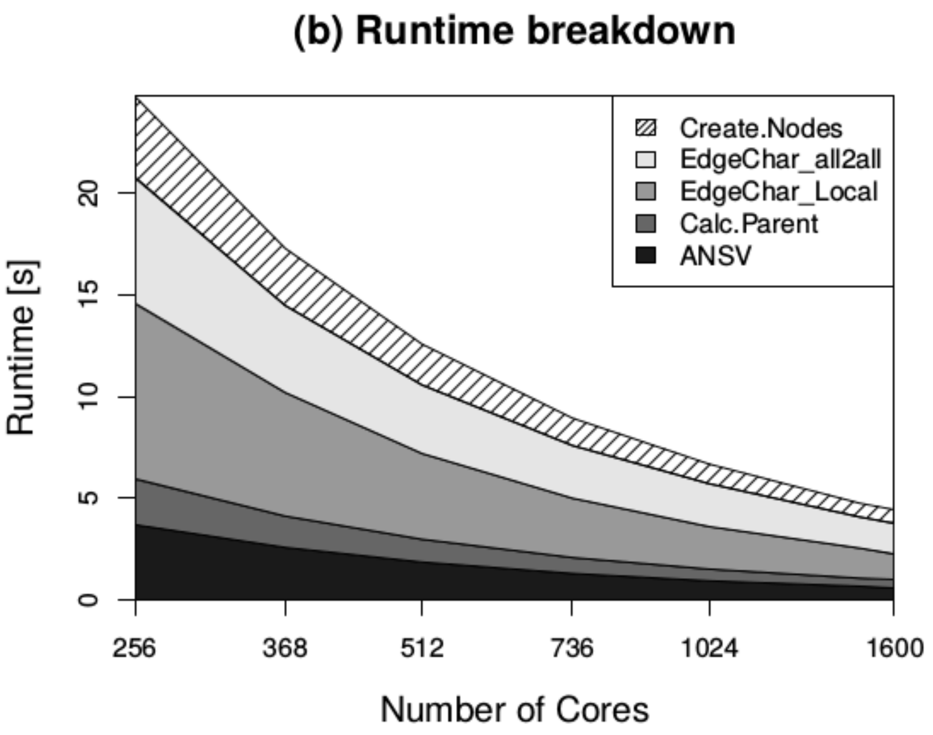
Experimental System:

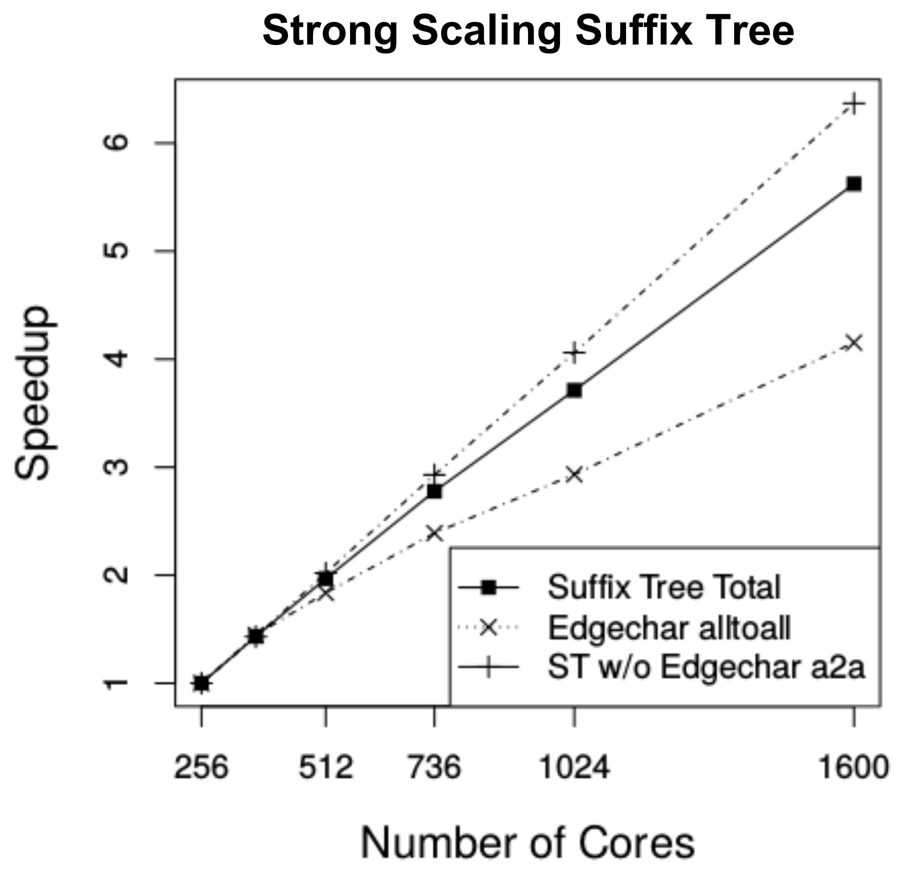
- 100 nodes: 2x 8 core **Intel E5-2650**
- 128 GB RAM per node
- QDR Infiniband

Experiments and Results

Results ST Construction

Construction for Pine Genome (12 GB)





Summary

Parallel Distributed Memory **Suffix Array, LCP Array**, and **Suffix Tree** Construction

- Indexing of Human Genome on 1024 Xeon cores in < 9.5s

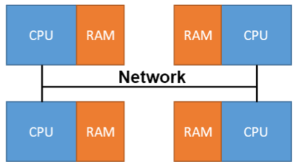
$$S \xrightarrow[7.5s]{[SC'15]} SA + LCP \xrightarrow[1.7s]{[IPDPS'17]} ST$$

- Scalable to **large strings:** O(n/p) memory per node
- Better theoretical complexity than prior distributed memory algorithms or available shared memory implementations
- Outperforms state-of-the-art in shared and distributed memory
- **Open Source** C++ implementation: **github.com/patflick/psac**

Outline

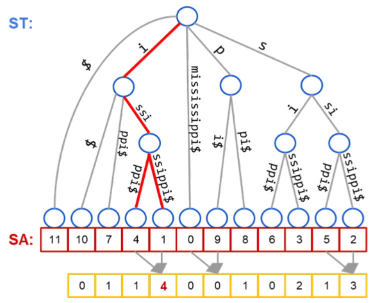
1

Very Short Intro to **Parallel** and **Distributed** Computing



3

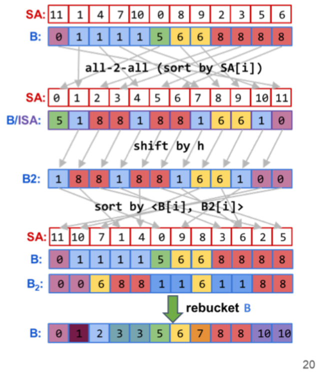
All-Nearest-Smaller-Values and Distributed Construction of **Suffix Trees**



[IPDPS '17]

2

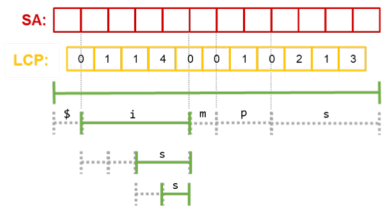
Distributed Construction of **Suffix Arrays** and **LCP Arrays**



[SC '15]

4

Distributed Enhanced Suffix Arrays



[Under Review]

DESA: Background

Enhanced Suffix Arrays (ESA) [Abouelhoda ’04] [Fischer ’07]

- Space efficient virtual representation of **Suffix Tree**
- Consists of:
  - **Suffix Array**
  - **LCP Array** (+ virtual LCP interval tree)
  - **Child Table** [Abouelhoda ’04] **/ RMQ** over LCP [Fischer ’07]
- Forward-search query algorithms require random accesses into the string S at every step
  - Possibly anywhere in $[0, n]$
  - In distributed memory: prohibitively expensive
- Backward-search algorithms (e.g., FM-Index) also require many random accesses into the size n data structures.

→ For distributed memory, we need different data structures

→ A “subtree” of size n/p should be represented and be efficiently query-able in O(n/p) memory

DESA

Distributed Enhanced Suffix Arrays

- Requirement:
  - Subtree of size n/p should use O(n/p) memory and be efficiently query-able

- Key Ideas:
  - Allow false-positives during traversal
  - Branching characters can be pre-computed and stored
    - Eliminate Random Reads of S
    - For subtree of size n/p, there are $O(n/p)$ branching characters
  - Match only branching characters during top-down pattern matching
  - Single string comparison at the end of the traversal

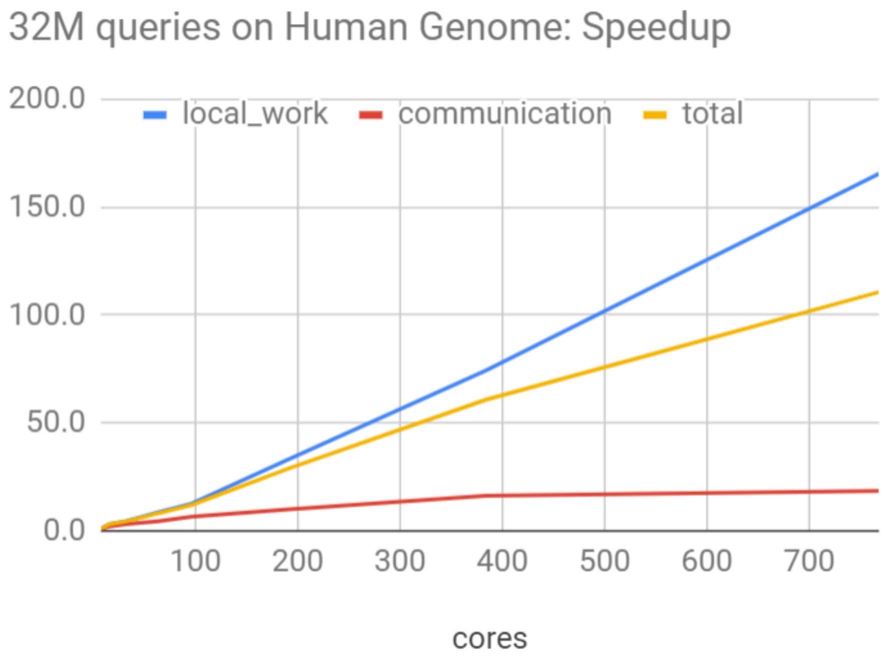
DESA: Results

Results DESA Query

Query Time in μs per query

	dna	proteins	english	sources	dplb
esa_index	7.6	11.2	28.0	30.4	26.6
desa_index	6.4	9.7	19.1	20.0	15.7
desa_tl_index	**6.0**	**5.8**	**14.5**	**14.7**	**10.1**
sdsl::csa_wt	6.3	13.7	15.1	19.9	18.8
sdsl::csa_sada	74.9	72.2	65.9	94.1	97.9

Scaling in Distributed Memory

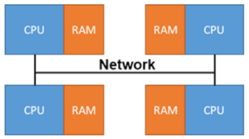


System:
Edison (Cray XC-30 Supercomputer)

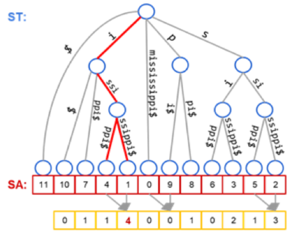
Summary

Topics covered:

1 Very Short Intro to **Parallel** and **Distributed** Computing

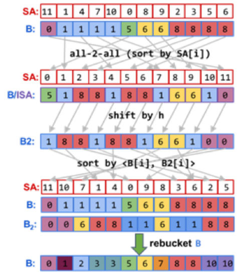


3 All-Nearest-Smaller-Values and Distributed Construction of **Suffix Trees**



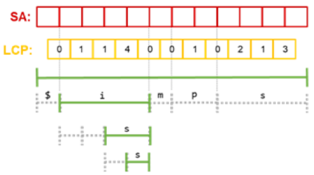
[IPDPS '17]

2 Distributed Construction of **Suffix Arrays** and **LCP Arrays**



[SC '15]

4 **Distributed Enhanced Suffix Arrays**



[Under Review]

Questions?

Software

GitHub
github.com/patflick/psac

Contact

Patrick Flick
patrick.flick@gatech.edu
patflick.github.io

Acknowledgements

Advisor

Srinivas Aluru
aluru@cc.gatech.edu

Funding

NSF

DESA: Background

ESA top-down traversal

ALGORITHM 8: ESA query algorithm [32, 10]

Input: Pattern $P = p_0 p_1 \cdots p_{m-1}$

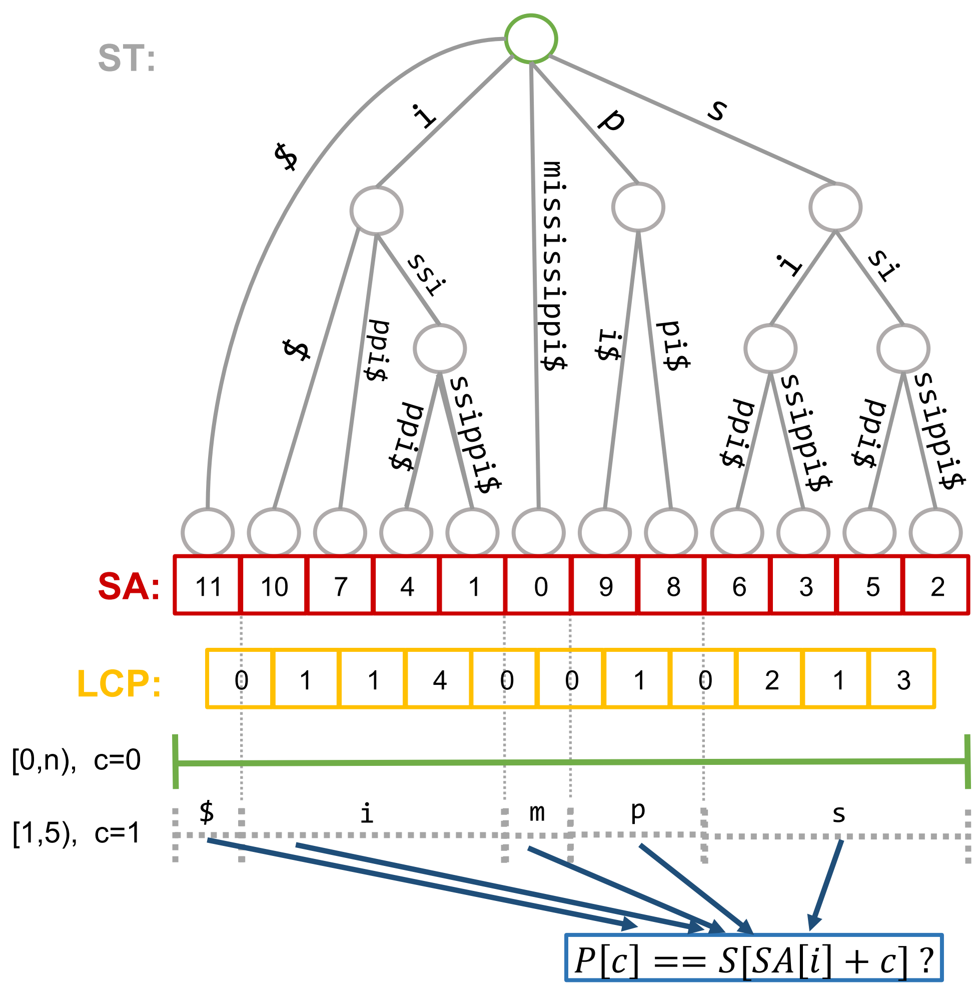
Output: interval of P in SA

```
[l, r] ← [0, n]
c ← 0,   found ← true
while found ∧ c < m ∧ l < r do
    [l, r] ← $\boxed{\texttt{getChild}(l, r, P[c])}$
    if [l, r] == ∅ then
        return'not found'
    end
    $\ell \leftarrow \min(\min_{i \in [l+1, r]} LCP[i], m)$
    found ← $\boxed{(S[SA[l] + c \ldots SA[l] + \ell - 1] == P[c \ldots \ell - 1])}$
    $c \leftarrow \ell$
end
```

Random Accesses into the String S

S = mississippi$

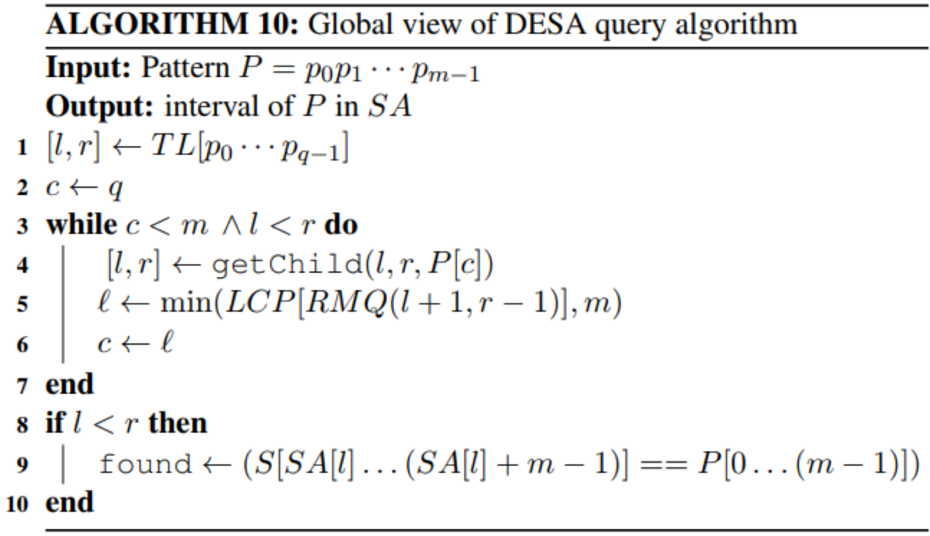
P = issip



DESA

DESA top-down traversal

- Allow false-positives during traversal
  - Only get child intervals by query character
  - Don’t check whole edge label during traversal
- Single comparison at the end of the traversal
- Use Hierarchical Parallel Succinct RMQ [Flick ’15] [Fisher ‘09]
- Top Level Lookup Table (TL)
  - Fixed q-mer offset lookup
  - Skip up to q iterations
  - Narrows the search interval



ALGORITHM 10: Global view of DESA query algorithm

```
Input: Pattern $P = p_0 p_1 \cdots p_{m-1}$
Output: interval of $P$ in $SA$
$[l, r] \leftarrow TL[p_0 \cdots p_{q-1}]$
$c \leftarrow q$
while $c < m \land l < r$ do
    $[l, r] \leftarrow \texttt{getChild}(l, r, P[c])$
    $\ell \leftarrow \min(LCP[RMQ(l + 1, r - 1)], m)$
    $c \leftarrow \ell$
end
if $l < r$ then
    $\texttt{found} \leftarrow (S[SA[l] \ldots (SA[l] + m - 1)] == P[0 \ldots (m - 1)])$
end
```

DESA getChild()

DESA `getChild()`

- Eliminate Random Reads of S
- Introduce Array L_c:

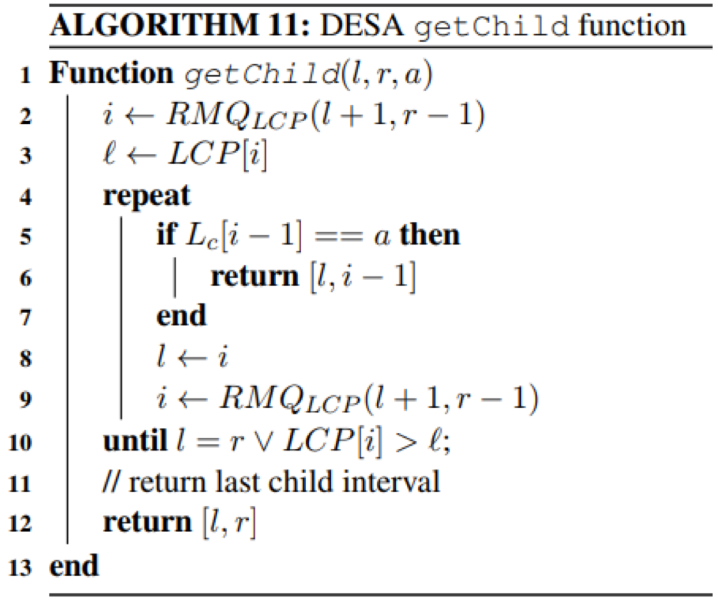
$$L_c[i] = S[SA[i] + LCP[i+1]]$$

- Then string access becomes

$$P[c] == S[SA[i] + l]\ ?$$

$$\Rightarrow P[c] == L_c[i]\ ?$$

- If query interval is local, then all required data is local
    - RMQ, LCP, SA, and L_c



ALGORITHM 11: DESA `getChild` function

```
Function getChild($(l, r, a)$)
    $i \leftarrow RMQ_{LCP}(l+1, r-1)$
    $\ell \leftarrow LCP[i]$
    repeat
        if $L_c[i-1] == a$ then
            return $[l, i-1]$
        end
        $l \leftarrow i$
        $i \leftarrow RMQ_{LCP}(l+1, r-1)$
    until $l = r \vee LCP[i] > \ell$;
    // return last child interval
    return $[l, r]$
end
```