Parallel Distributed Memory String Indexes

Efficient Construction and Querying

Patrick Flick & Srinivas Aluru
Computational Science and Engineering
Georgia Institute of Technology
In this talk...

Overview

1. Very Short Intro to Parallel and Distributed Computing

2. Distributed Construction of Suffix Arrays and LCP Arrays

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

   [IPDPS '17]

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 \( \tau \)
  • 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)`

<table>
<thead>
<tr>
<th></th>
<th>P₀</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S:</strong></td>
<td>m i s s i s s i</td>
<td>s i p p i</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td><strong>SA:</strong></td>
<td>11 10 7 4 1 0</td>
<td>9 8 6 3 5 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ISA:</strong></td>
<td>5 4 11 9 3 10</td>
<td>8 2 7 6 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LCP:</strong></td>
<td>0 1 1 4 0</td>
<td>0 1 0 2 1 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n/p</td>
<td>n/p</td>
<td>n/p</td>
<td>n/p</td>
</tr>
</tbody>
</table>
Outline

1. Very Short Intro to Parallel and Distributed Computing

2. Distributed Construction of Suffix Arrays and LCP Arrays
   [SC ‘15]

3. All-Nearest-Smaller-Values and Distributed Construction of Suffix Trees
   [IPDPS ‘17]

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**
  - ctgccagtgagattatcggctatatgcacactttgactaggaactaaat

- **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 Tree and Array Diagram](image)
Contributions:

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

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

- 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**

---

1. **k-mer sorting**

2. **Hybrid**

---

**Algorithm 1:** Distributed Manber & Myers

**Algorithm 2:** Communication avoiding prefix-doubling

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

---

**Runtime per Iteration for Human Genome (k=1):**
- Number Non-singleton elements
- Runtime Algorithm 1
- Runtime Algorithm 2

**Runtime per Iteration for Human Genome (k=21):**
- Number Non-singleton elements
- Runtime Algorithm 1+2

---

**K-mer sorting**
### Experiments and Results

## Results

### SA Construction

**Runtime (seconds) of different methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>H 2G</th>
<th>H 3G</th>
<th>P 12G</th>
</tr>
</thead>
<tbody>
<tr>
<td>divsufsort</td>
<td>424.5</td>
<td>586.4</td>
<td>X</td>
</tr>
<tr>
<td>mkESA (1)</td>
<td>586.6</td>
<td>1,123</td>
<td>X</td>
</tr>
<tr>
<td>mkESA (4)</td>
<td>462.6</td>
<td>759</td>
<td>X</td>
</tr>
<tr>
<td>cloudSACA (128)</td>
<td>40.6</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Our method (128)</td>
<td>16.3</td>
<td>22.1</td>
<td>142.6</td>
</tr>
<tr>
<td>Our method (1600)</td>
<td>3.5</td>
<td>4.8</td>
<td>14.8</td>
</tr>
</tbody>
</table>

**Experimental System:**
- 100 nodes: 2x 8 core Intel E5-2650
- 128 GB RAM per node
- QDR Infiniband
1. Very Short Intro to Parallel and Distributed Computing

2. Distributed Construction of Suffix Arrays and LCP Arrays
   - [SC '15]

3. All-Nearest-Smaller-Values and Distributed Construction of Suffix Trees
   - [IPDPS '17]

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 \( \geq t \)
- 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 \( r > l \)
  - \( \max(LCP[j], LCP[h]) \)

\( \Rightarrow \) All Nearest Smaller Values (ANSV)
Parallel Suffix Tree Construction

All Nearest Smaller Values

- For each element in an array $A$
  - find nearest smaller: $l_{NS}(a_i) = \max_{j<i} \{j | a_j < a_i\}$

- Well studied problem with parallel solutions
  - however, with too restrictive assumptions

- Problem: which is the unique parent node? $\Rightarrow$ Leftmost of equal value

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

[Diagram showing suffix tree construction with SA and LCP arrays]
All Nearest Smaller Values Parallel Algorithm

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

2. Allgather processor minimas $m_i$

3. Determine sections to exchange based on $(m_0, m_1, ..., m_{p-1})$

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

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

$\mathbb{E}[comm] = \log\left(\frac{n}{p}\right)$
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 $g\text{ANSV}$ problem on the **LCP**
  - Combine results from left and right matches
- Inverse edges ($i, \text{parent}(i)$) and create internal nodes for each unique $\text{parent}(i)$
- Label each edge with its first character
## Results SA + LCP + ST Construction

### Construction Time for Human Genome

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

### 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)

(b) Runtime breakdown

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

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

  $S \overset{[SC'15]}{\rightarrow} SA + LCP \overset{[IPDPS'17]}{\rightarrow} ST$

- 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](https://github.com/patflick/psac)
Outline

1. Very Short Intro to Parallel and Distributed Computing

2. Distributed Construction of Suffix Arrays and LCP Arrays
   [SC '15]

3. All-Nearest-Smaller-Values and Distributed Construction of Suffix Trees
   [IPDPS '17]

4. Distributed Enhanced Suffix Arrays
   [Under Review]
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
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 precomputed 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
Results DESA Query

Query Time in \( \mu s \) per query

<table>
<thead>
<tr>
<th></th>
<th>dna</th>
<th>proteins</th>
<th>english</th>
<th>sources</th>
<th>dplb</th>
</tr>
</thead>
<tbody>
<tr>
<td>esa_index</td>
<td>7.6</td>
<td>11.2</td>
<td>28.0</td>
<td>30.4</td>
<td>26.6</td>
</tr>
<tr>
<td>desa_index</td>
<td>6.4</td>
<td>9.7</td>
<td>19.1</td>
<td>20.0</td>
<td>15.7</td>
</tr>
<tr>
<td>desa_tl_index</td>
<td>6.0</td>
<td>5.8</td>
<td>14.5</td>
<td>14.7</td>
<td>10.1</td>
</tr>
<tr>
<td>sdsl::csa_wt</td>
<td>6.3</td>
<td>13.7</td>
<td>15.1</td>
<td>19.9</td>
<td>18.8</td>
</tr>
<tr>
<td>sdsl::csa_sada</td>
<td>74.9</td>
<td>72.2</td>
<td>65.9</td>
<td>94.1</td>
<td>97.9</td>
</tr>
</tbody>
</table>

Scaling in Distributed Memory

32M queries on Human Genome: Speedup

System:
Edison (Cray XC-30 Supercomputer)
Summary

Topics covered:

1. Very Short Intro to Parallel and Distributed Computing
2. Distributed Construction of Suffix Arrays and LCP Arrays [SC ’15]
3. All-Nearest-Smaller-Values and Distributed Construction of Suffix Trees [IPDPS ’17]
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 $SA$

1. $[l, r] \leftarrow [0, n]$
2. $c \leftarrow 0$, $found \leftarrow true$
3. while $found \land c < m \land l < r$
4.     $[l, r] \leftarrow \text{getChild}(l, r, P[c])$
5.     if $[l, r] == \emptyset$ then
6.         return 'not found'
7. end
8. $\ell \leftarrow \min(\min_{i \in [l+1, r]} LCP[i, m], m)$
9. $found \leftarrow (S[SA[l] + c \cdots SA[l] + \ell - 1] == P[c \cdots \ell - 1])$
10. $c \leftarrow \ell$
11. end

**Random Accesses into the String $S$**

![Diagram showing ESA top-down traversal with input $S = \text{mississippi}$ and $P = \text{issip}$]
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 $SA$

1. $[l, r] \leftarrow TL[p_0 \cdots p_{q-1}]$
2. $c \leftarrow q$
3. while $c < m \land l < r$ do
4.   $[l, r] \leftarrow \text{getChild}(l, r, P[c])$
5.   $\ell \leftarrow \min(LCP[RMQ(l + 1, r - 1)], m)$
6.   $c \leftarrow \ell$
7. end
8. if $l < r$ then
9.   found $\leftarrow (S[SA[l] \cdots (SA[l] + m - 1)] == P[0 \ldots (m - 1)])$
10. end
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 $L_c$