# Parallel and Scalable Combinatorial String and Graph Algorithms on Distributed Memory Systems

Patrick Flick, Srinivas Aluru

*School of Computational Science and Engineering, Georgia Institute of Technology*

## Introduction

### Abstract

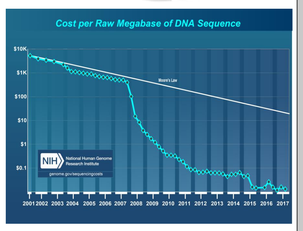
Methods for processing and analyzing DNA and genomic data are built upon combinatorial graph and string algorithms. To process and analyze state-of-the-art genomic data sets require the design of scalable and efficient parallel algorithms and the use of large computing clusters.

We present **distributed-memory parallel** algorithms for:
- Indexing large genomic datasets
  - Construction of Suffix Arrays,
  - LCP arrays,
  - Suffix Trees.
- All-Nearest-Smaller-Values (ANSV) problem
- Generalizations to multiple input strings (millions of short reads)
- clustering de-Bruijn graphs and its application to solving a Grand Challenge Metagenomic dataset.

### Motivation




- Sequencing technology is getting better, faster, and cheaper at a much faster rate than computer technology




- Sequence and genomic datasets are large
  - Human Genome: 3.2 GB
  - Matching Read Set: 150 GB
  - Metagenomics Data: 400 GB

- Slow sequential processing times
  - Assembly: 30-50 hours

- Memory requirements are huge
  - TBs of main memory required

➡ High Performance, Parallel, and Distributed Computing

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

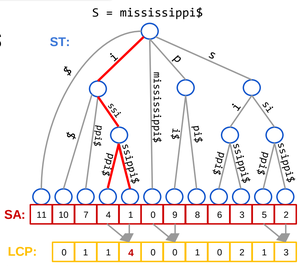
## Distributed Memory String Indexing

### Suffix Arrays and Trees

**Suffix Tree (ST)**
- trie of all suffixes of a string S
- 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

### Key Contributions

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

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

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

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

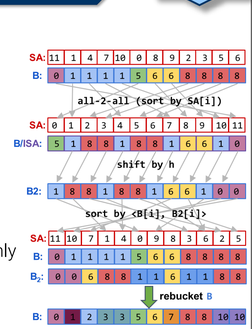
## Part I: Suffix Array Construction

SC'15 Best Student Paper

### Algorithms

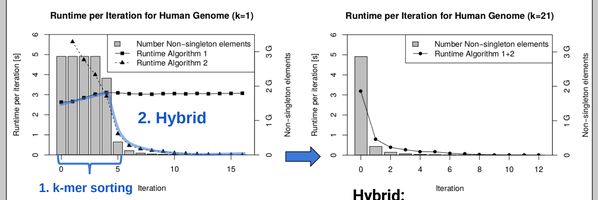
***Algorithm 1:*** Distributed parallel prefix-doubling
- sort suffixes based on bucket-rank
- Double sorted length each iteration by sorting `(rank[i], rank[i+2h])`

***Algorithm 2***: Communication avoiding prefix-doubling
- keep track of unfinished buckets
- Communicate doubling information only for unfinished buckets




**LCP-Construction:**
- Via distributed bulk-parallel RMQ

### Hybrid Algorithm & Optimizations




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

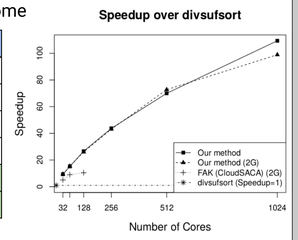
**Hybrid:** Introspectively switch between algorithms based on number of non-singleton buckets

### Results

**Results SA + LCP Construction**

> 110x Speedup for Human Genome

| 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** |




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

## Part II: Suffix Tree Construction

IPDPS'17

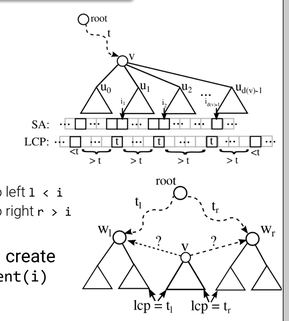
### ST Construction Algorithm

**SA** = leafs of **ST**

**LCP** = internal nodes of **ST**
- Construct **ST** "bottom-up"
- Determine parent for each node

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

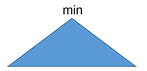
  `for each LCP[i]:`
  - Find nearest smaller `LCP[l] < LCP[i]` to left `l < i`
  - Find nearest smaller `LCP[r] < LCP[i]` to right `r > i`
  - `max(LCP[l], LCP[r])`
- Inverse edges `(i, parent(i))` and create internal nodes for each unique `parent(i)`
- Complexity $O(\frac{n}{p} + p)$ time




### All Nearest Smaller Values

For every **LCP[i]**, find nearest smaller element to the left and right

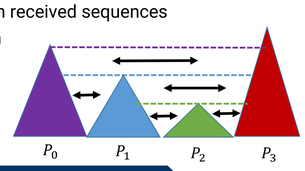
1. Sequentially find matches locally:
   - Keep unmatched elements: bitonic sequence
2. Allgather processor minimas $m_i$
3. Determine sections to exchange based on $(m_0, m_1, \ldots, m_{p-1})$
4. Send / Receive sections
   - So that each processor sends/receives at most n/p elements
5. Solve unmatched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




Complexity

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

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

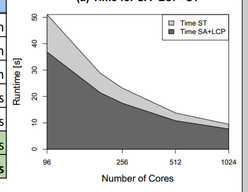



### Results

**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 | 14 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 |
| Ours | 4x 18 core **Intel E7-8870** | 72 | **63** s |
| Ours | 64x 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

## Distributed Connected Components

SC'16 Reproducibility Winner

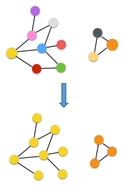
### Metagenomics Motivation

- Soil Metagenomic Datasets (JGI)
  - Iowa Corn Soil: 1.8 billion reads
  - Iowa Prairie Soil: 3.3 billion reads
- De Bruijn graph of Corn Soil reads:
  - 135 billion edges
  - Too large to assemble directly
  - High species level heterogeneity leads to large number of disjoint Connected Components *(Howe et al. '14)*
- Connected Components of de-Bruijn graph
  - "Partition" the read set
  - Each CC can be assembled independently
  - Each read is fully contained within a CC

### Algorithm Key Ideas

**Connected Components for Read Sets**

[Flick, Jain, Pan, Aluru - SC'15]

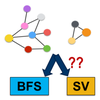


- Label Propagation:
  - Each node (=read) starts with it's own label
  - Iteratively: adopt minimum label in neighborhood
  - Convergence: all nodes/reads in CC have same label
- Graph: distributed edge list

  `< node_id, node_id, label >`
- Neighborhood: Iteratively use parallel sorting:
  1. by node ids , 2. by label ids
- Add doubling step for achieving logarithmic convergence

**Adaptive algorithm for general graphs**

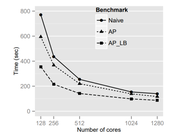
[Jain, Flick, Pan, Green, Aluru - TPDS '17]




- BFS is better for some graphs
- Adaptively decide when to run BFS vs our algo
- Beats state-of-the-art (Multistep [Slota SC'16]) on most graphs

### Results

- Solving Grand Challenge Problem
  - < 22 minutes on 1024 cores
  - < 3 minutes on 32K cores
- Strong scaling
- Load Balanced version performs significantly better




## Acknowledgements

*Advisor*



Srinivas Aluru
aluru@cc.gatech.edu

*Co-Authors*

Chirag Jain
cjain@gatech.edu

Tony Pan
tony.pan@gatech.edu

*Contact*

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

*Funding*

CCF-1360593
IIS-1416259
CNS-1229081

## References

1. **Flick, P.**, & Aluru, S. (2015, November). Parallel distributed memory construction of suffix and longest common prefix arrays. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 16). ACM. **SC' 15**
2. **Flick, P.**, & Aluru, S. (2017, May). Parallel Construction of Suffix Trees and the All-Nearest-Smaller-Values Problem. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International* (pp. 12-21). IEEE. **IPDPS'17**
3. **Flick, P.**, Jain, C., Pan, T., & Aluru, S. (2015, November). A parallel connectivity algorithm for de Bruijn graphs in metagenomic applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 15). ACM. **SC'15**
4. Jain, C., **Flick, P.**, Pan, T., Green, O., & Aluru, S. (2017). An Adaptive Parallel Algorithm for Computing Connected Components. *IEEE Transactions on Parallel and Distributed Systems*, *28*(9), 2428-2439.