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

SC'18 Doctoral Showcase Supplementary File

Patrick Flick
Georgia Institute of Technology
patrick.flick@gatech.edu

Srinivas Aluru
Georgia Institute of Technology
aluru@cc.gatech.edu

ABSTRACT

Methods for processing and analyzing DNA and genomic data are built upon combinatorial graph and string algorithms. The advent of high-throughput DNA sequencing is enabling the generation of billions of reads per experiment. Classical and sequential algorithms can no longer deal with these growing data sizes - which for the last 10 years have greatly out-paced advances in processor speeds. 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.

Here, we present our distributed-memory parallel algorithms for indexing large genomic datasets, including algorithms for construction of suffix arrays and LCP arrays, solving the All-Nearest-Smaller-Values problem and its application to the construction of suffix trees. Our parallel algorithms exhibit superior runtime complexity as well as practical performance compared to the state-of-the-art. Furthermore, we present our work on distributed-memory algorithms for clustering de-bruijn graphs and its application to solving a grand challenge metagenomic dataset.

1 DISTRIBUTED STRING INDEXING

Basics

The *Suffix Tree (ST)* of a string S is a (compacted) trie of all the suffixes of S. For a given string S of length n, its *Suffix Tree* can be constructed in $O(n)$ time [20]. Once constructed, the Suffix tree allows identification of all occurrences of a pattern P in S in $O(|P|)$ time and independent of n, given that the alphabet size is constant. Since their inception in the early 1970's, suffix trees have become the most widely used string indexing structure, with various applications such as approximate pattern matching, identification of longest common substrings, finding maximal suffix-prefix overlaps, data compression, and many more [6].

A *Suffix Array (SA)* is an array containing the lexicographically sorted order of all suffixes of a string, and as such, compactly represents the leafs of a *Suffix Tree*. Manber and Myers first introduced suffix arrays as a space-efficient alternative to suffix trees [13], and showed how to use suffix arrays for exact pattern matching in $O(|P| \log n)$ time, and when used in conjunction with the *Longest Common Prefix (LCP)* array in $O(|P| + \log n)$ time, i.e., at the additional cost of $O(\log n)$ compared to suffix trees. Sequential algorithms for constructing suffix arrays are abundant. Puglisi *et al.* give a good survey of the various approaches [17]. Suffix arrays can also be constructed in linear time. Interestingly, direct linear time algorithms that avoid suffix trees as an intermediary step were invented only in 2003 [10, 12].

Suffix arrays are often used in conjunction with *Longest Common Prefix (LCP)* arrays. The LCP array simply records the length of the longest common prefix between every pair of consecutive suffixes in the suffix array. Abouelhoda *et al.* showed that suffix arrays combined with *LCP* arrays can support a majority of operations supported by suffix trees [1]. The *LCP* array can be constructed either during the construction of the suffix array [10, 13], or from a given suffix array in linear $O(n)$ time [11].

Motivation

Much recent work on suffix arrays and trees is motivated by their ubiquitous presence in computational biology applications. The advent of high-throughput DNA sequencing is generating billions of short reads per experiment, necessitating the design of parallel algorithms.

It is particularly challenging to parallelize linear time sequential algorithms in distributed memory, owing to the difficulty in designing communication-efficient algorithms that can mask communication time effectively with such low computational cost. Nevertheless, distributed-memory construction of suffix arrays and trees is an important problem because a) memory issues preclude sequential construction for large data sets, particularly given the size of a tree is significantly larger than the original string by an order of magnitude or more, and b) it is a necessary first step to parallelize the myriad applications that use suffix arrays or trees.

Contributions

In this work, we present provably and practically efficient distributed memory algorithms for constructing Suffix Arrays, LCP arrays, and Suffix Trees. In contrast to most previous work, in our approach the $O(n)$ input, output, and all working data is fully distributed and split onto p processors, such that every processor requires at most $O(\frac{n}{p})$ memory. This is particularly important as most works have focused on parallelizing compute time while assuming each processor has a copy of everything, severely limiting their scalability when used in practice.

Distributed Suffix Array Construction. We present parallel algorithms for distributed memory construction of Suffix Arrays and Longest Common Prefix (LCP) arrays that simultaneously achieve good worst-case run-time bounds and superior practical performance. We published this work at Supercomputing 2015 [3].

Our algorithm provides a worst case run-time guarantee of $O(T_{sort}(n,p) \cdot \log(n))$ where $T_{sort}(n,p)$ is the run-time of parallel sorting. Additionally, our approach constructs the LCP array alongside the suffix array, also in a fully distributed fashion.

We introduced several algorithm engineering techniques that improve performance in practice. We provide an efficient, scalable implementation of our algorithm, which constructs the suffix and LCP arrays of the human genome in 7.3 seconds on 1024 Intel Xeon cores (64 nodes: 2x Xeon E5-2650, 128 GB RAM, QDR IB). We reach speedups of over 110× compared to *divsufsort* [14], the fastest sequential suffix array construction implementation, commonly used as comparison [16] [18].

Suffix Tree Construction. Sequentially, the construction of suffix trees takes linear time, and optimal parallel algorithms exist only for the PRAM model. Recent works mostly target low core-count shared-memory implementations but achieve suboptimal complexity, and prior distributed-memory parallel algorithms have quadratic worst-case complexity. We present a novel, efficient distributed memory algorithm for constructing the suffix tree for a string given its suffix array and LCP array in $O(\frac{n}{p} + p)$ time.

To do so, we introduced a novel generalization of the *All-Nearest-Smaller-Values (ANSV)* problem and give an optimal algorithm to solve this problem in distributed memory, minimizing overall communication volume. Combining this with our work on constructing suffix arrays [3] results in a parallel algorithm for constructing the suffix tree from a given input string. Compared to previous distributed memory algorithms, this yields superior theoretical complexity as well as practical performance. We demonstrate the construction of the suffix tree for the human genome given its suffix and LCP arrays in under 2 seconds on 1024 Intel Xeon cores. Furthermore, we demonstrate that our MPI based implementation performs better in shared memory than state-of-the-art shared memory algorithms, and can scale to a large number of cores in distributed memory. We published this work at IPDPS 2017 [4].

All-Nearest-Smaller-Values. Given a sequence of values, solving the *All-Nearest-Smaller-Values (ANSV)* problem requires finding for each element the first smaller element to the left (or right). A number of problems can be reduced to the *ANSV* problem, including merging of two sorted sequences, monotone polygon triangulation, Cartesian tree construction, and parenthesis matching [2, 7]. Thus, while we present our parallel algorithm for the ANSV problem to facilitate construction of suffix trees, it is a problem worth studying in its own right and our algorithm can be used in these and many other applications.

2 DISTRIBUTED CONNECTED COMPONENTS

The Grand Challenge Iowa corn soil metagenomic data set sequenced at the Joint Genome Institute contains 1.8 billion sequencing reads [15]. The corresponding de-Bruijn graph consists of approximately 135 billion vertices and edges, too large for any assembler to assembly directly. Howe *et al.* [8] discovered that the high species level heterogeneity in metagenomic data sets leads to a large number of disjoint connected components in the de Bruijn graph. This property can be exploited to partition the reads into disjoint sets and assemble each set independently.

Motivated by this application, we developed a distributed memory parallel connected components algorithm, making use of iterative sorting of an edge list graph format and merging of neighboring or overlapping components, as well as a neighbor doubling approach similar to the pointer jumping method used in list ranking. We demonstrated the scalability of this algorithm by partitioning the grand challenge Iowa corn soil Metagenomic data set with 1.8 billion reads, a graph with ≈ 135 billion edges and ≈ 390 million components, in 22 minutes (previously 120h [8]) using 80 nodes (2 socket Xeon E5-2650) totaling 1280 cores [5].

Our algorithm showed promising results also for other types for graphs. We continued this work by creating a hybrid approach between our Connected Components (CC) algorithm and distributed memory Breadth-First-Search (BFS) [9]. We showed that using runtime algorithm selection between BFS and our CC algorithm, the hybrid method generalizes to diverse graph topologies and achieves superior performance compared to the state-of-the-art *MultiStep* method[19].

REFERENCES

[1] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. 2004. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms* 2, 1 (2004), 53–86.

[2] Omer Berkman, Baruch Schieber, and Uzi Vishkin. 1993. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms* 14, 3 (1993), 344–370.

[3] Patrick Flick and Srinivas Aluru. 2015. 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*. ACM, 16.

[4] Patrick Flick and Srinivas Aluru. 2017. Parallel Construction of Suffix Trees and the All-Nearest-Smaller-Values Problem. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 12–21.

[5] Patrick Flick, Chirag Jain, Tony Pan, and Srinivas Aluru. 2015. 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*. ACM, 15.

[6] Dan Gusfield. 1997. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press.

[7] Xin He and Chun-Hsi Huang. 2001. Communication efficient BSP algorithm for all nearest smaller values problem. *J. Parallel and Distrib. Comput.* 61, 10 (2001), 1425–1438.

[8] Adina Chuang Howe, Janet K Jansson, Stephanie A Malfatti, Susannah G Tringe, James M Tiedje, and C Titus Brown. 2014. Tackling Soil Diversity with the Assembly of Large, Complex Metagenomes. *Proceedings of the National Academy of Sciences* 111, 13 (2014), 4904–4909.

[9] Chirag Jain, Patrick Flick, Tony Pan, Oded Green, and Srinivas Aluru. 2017. An Adaptive Parallel Algorithm for Computing Connected Components. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2428–2439.

[10] Juha Kärkkäinen and Peter Sanders. 2003. Simple linear work suffix array construction. In *Automata, Languages and Programming*. Springer, 943–955.

[11] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. 2001. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial pattern matching*. Springer, 181–192.

[12] Pang Ko and Srinivas Aluru. 2003. Space efficient linear time construction of suffix arrays. In *Combinatorial Pattern Matching*. Springer, 200–210.

[13] Udi Manber and Gene Myers. 1993. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing* 22, 5 (1993), 935–948.

[14] Yuta Mori. [n. d.]. libdivsufsort. https://github.com/y-256/libdivsufsort.

[15] Henrik Nordberg, Michael Cantor, Serge Dusheyko, Susan Hua, Alexander Poliakov, Igor Shabalov, Tatyana Smirnova, Igor V Grigoriev, and Inna Dubchak. 2014. The Genome Portal of the Department of Energy Joint Genome Institute: 2014 Updates. *Nucleic Acids Research* 42, D1 (2014), D26–D31.

[16] Vitaly Osipov. 2012. Parallel suffix array construction for shared memory architectures. In *String Processing and Information Retrieval*. Springer, 379–384.

[17] Simon J Puglisi, William F Smyth, and Andrew H Turpin. 2007. A taxonomy of suffix array construction algorithms. *ACM Computing Surveys (CSUR)* 39, 2 (2007), 4.

[18] Julian Shun. 2014. Fast parallel computation of longest common prefixes. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 387–398.

[19] George M Slota, Sivasankaran Rajamanickam, and Kamesh Madduri. 2016. A Case Study of Complex Graph Analysis in Distributed Memory: Implementation and Optimization. In *Parallel and Distributed Processing Symposium, 2016 IEEE 30th International*. IEEE.

[20] Esko Ukkonen. 1995. On-line construction of suffix trees. *Algorithmica* 14, 3 (1995), 249–260.