Introduction

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 requires 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 than ever before.
- High-throughput sequencing technologies are capable of generating more data than can be efficiently processed in a single node.
- Memory requirements are huge.
- TBs of DNA reads are common.
- 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
  - Annotation and analysis of genome sequences, e.g. RNA genes, protein coding regions
- Two major approaches:
  1. Index target sequence by fixed size substrings: k-mer index
  2. Index all suffixes: Suffix Arrays, Suffix Trees, FM index

Abstract

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

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 leaves 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

- 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/pac

Motegametic 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:
  - 113 billion edges
  - Too large to assemble directly
  - High species level heterogeneity leads to large number of disjoint connected components (Kowke et al. ’14)

Part II: Suffix Tree Construction

- ST Construction Algorithm

Hybrid Algorithm & Optimizations

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

Annexes

- Internal nodes for each unique

Complexity

- O( n/p ) time

Best Student Paper

- 2015 SC paper: "A parallel connectivity algorithm for de novo assembly of metagenomic datasets" by Patrick Flick, Srinivas Aluru, Patrick Flick, Srinivas Aluru

Algorithm Key Ideas

- Label Propagation
  - Each node (read) starts with its 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

- Adding label steps for achieving logarithmic convergence

Adaptive algorithm for general graphs

- BFS is better for some graphs
- Adaptively decide when to run BFS vs our algo

- Beats state-of-the-art (Multistep [Stota SC’16]) on most graphs

Results

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

References


Acknowledgments

- Advisor: Srinivas Aluru
- Co-Authors: Chirag Jain, Tony Pan
- Contact: Patrick Flick
- Funding: CNS-1545820, CNS-1223981

Experimental System:

- 100 nodes, 2x 8 core Intel E5-2690, 128 GB RAM per node, QDR Infiniband

Results SA + LCP Construction

- > 110x Speedup for Human Genome

- Speedup over distributed

Method

- Random RCV reads

- Length

- Depth

- LCP

- Speedup

Algorithm

- Distributed Manber & Myers Communication avoiding prefix doubling

Hybrid

- Interspersively switch between algorithms based on number of non-singleton buckets

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. Aligner process minimas mi
  3. Determine sections to exchange based on (mi, mi+1, …, m|S|)
  4. Send / Receive sections
  5. So that each processor sends/receives at most n/p elements
  6. Send solutions back to origin

Complexity

- O( n/p ) time

- L[comm] = log2 L

Results SA + LCP + ST Construction

- Construction Time for Human Genome

Method

- Experimental System

- 100 nodes, 2x 8 core Intel E5-2690, 128 GB RAM per node, QDR Infiniband

Algorithm

- Distributed Manber & Myers Communication avoiding prefix doubling