Scalable String and Suffix Sorting: Algorithms, Techniques, and Tools

Timo Bingmann · Dissertation Defense · July 3rd, 2018
Overview

Multi-Core Scalable String Sorting

<table>
<thead>
<tr>
<th>1</th>
<th>alpha</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>arcade</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>array</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>kayak</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>kernel</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>kit</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>kitchen</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>kitten</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>krypton</td>
<td>0</td>
</tr>
</tbody>
</table>

External and Distributed Scalable Suffix Sorting

<table>
<thead>
<tr>
<th>0</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$</td>
</tr>
<tr>
<td>abcba</td>
<td>a</td>
</tr>
<tr>
<td>abcba</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>abcba</td>
<td>a</td>
</tr>
<tr>
<td>abcba</td>
<td>a</td>
</tr>
<tr>
<td>cba</td>
<td>$</td>
</tr>
<tr>
<td>abcba</td>
<td>a</td>
</tr>
<tr>
<td>abcba</td>
<td>a</td>
</tr>
<tr>
<td>cba</td>
<td>abcba</td>
</tr>
<tr>
<td>cba</td>
<td>abcba</td>
</tr>
<tr>
<td>cba</td>
<td>abcba</td>
</tr>
<tr>
<td>cba</td>
<td>abcba</td>
</tr>
<tr>
<td>cba</td>
<td>abcba</td>
</tr>
</tbody>
</table>
Input: $n$ strings containing $N$ characters in total.
String Sorting Algorithms

Theoretical Parallel Algorithms

- “Optimal Parallel String Algorithms: . . .” [Hagerup ’94]
  \[ O(\log N / \log \log N) \text{ time and } O(N \log \log N) \text{ work on CRCW PRAM} \]

Existing Basic Sequential Algorithms

- Radix Sort \[ O(D + n \log \sigma) \] [McIlroy et al. ’95]
- Multikey Quicksort \[ O(D + n \log n) \text{ exp.} \] [Bentley, Sedgewick ’97]
- Burstsort \[ O(D + n \log \sigma) \text{ exp.} \] [Sinha, Zobel ’04]
- Binary LCP-Mergesort \[ O(D + n \log n) \] [Ng, Kakehi ’08]

Existing Algorithm Library

- in C/C++ by Rantala (for Engineering Radix Sort [Kärkkäinen, Rantala ’09])

Our Contributions: New Basic and Practical Parallel Algorithms

- Parallel Super Scalar String Sample Sort (pS$^5$) [B, Sanders, ESA’13]
- Parallel $K$-way LCP-aware Mergesort (and Merge) [B, et al. Algorithmica’17]
Super Scalar String Sample Sort (S$^5$)

Array 0

Kit 0

Arrange 0

Kayak 0

Kernel 0

Kitchen 0

Kitten 0

Arcade 0

Kite 0

Abacus 0

Krypton 0

Alpha 0

Arcane 0

Based on Super Scalar Sample Sort

[Sanders, Winkel ’04]
Super Scalar String Sample Sort ($S^5$)

```
array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
arcane
```

based on Super Scalar Sample Sort
[Sanders, Winkel ’04]
Super Scalar String Sample Sort ($S^5$)

- partition by $w$ chars
- store in level-order and use predicated instructions

\[ i := 2i + 0/1 \]
Super Scalar String Sample Sort ($S^5$)

- **array**: 0
- **kit**: 0
- **arrange**: 0
- **kayak**: 0
- **kernel**: 0
- **kitchen**: 0
- **kitten**: 0
- **arcade**: 0
- **kite**: 0
- **abacus**: 0
- **krypton**: 0
- **alpha**: 0
- **arcane**: 0

- **equality checking:**
  1. at each splitter
  2. after full descent

- **interleave** tree descents:
  classify four strings at once
  ⇒ super scalar parallelism
Super Scalar String Sample Sort (S^5)

- easy parallelization
- classification tree in L2 caches of processors
Super Scalar String Sample Sort ($S^5$)

Prefix

<table>
<thead>
<tr>
<th>abacus</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>1</td>
</tr>
<tr>
<td>array</td>
<td>2</td>
</tr>
<tr>
<td>arrange</td>
<td>2</td>
</tr>
<tr>
<td>arcade</td>
<td>2</td>
</tr>
<tr>
<td>arcane</td>
<td>0</td>
</tr>
<tr>
<td>kayak</td>
<td>0</td>
</tr>
<tr>
<td>kernel</td>
<td>0</td>
</tr>
<tr>
<td>kit</td>
<td>0</td>
</tr>
<tr>
<td>kitchen</td>
<td>2</td>
</tr>
<tr>
<td>kitten</td>
<td>2</td>
</tr>
<tr>
<td>kite</td>
<td>0</td>
</tr>
<tr>
<td>krypton</td>
<td>0</td>
</tr>
</tbody>
</table>

Increase prefix by LCP of splitters or key size.

- reorder out-of-place, in-place, and/or in parallel
- top-level algorithm in parallel $S^5$
LCP Loser Tree – $K$-way LCP-Merge

needs at most
$\Delta L + n \log_2 K + K$
character comparisons

(2, aab)  LCP-Merge  (0, bca)
(2, aac)  (1, acb)  (1, acb)
(2, aab)  (2, aac)  (0, bca)

Timo Bingmann – Scalable String and Suffix Sorting: Algorithms, Techniques, and Tools
Institute of Theoretical Informatics – Algorithmics
July 3rd, 2018  11 / 29
LCP Loser Tree – $K$-way LCP-Merge

needs at most
\[ \Delta L + n \log_2 K + K \]
character comparisons

\[ (2, aab) \]
\[ (1, acb) \]

\[ (2, aac) \]
\[ (2, aab) \]
\[ (2, aac) \]
\[ (0, bca) \]
\[ (1, acb) \]

\[ (0, bca) \]
\[ (1, acb) \]

\[ (0, bca) \]
\[ (2, aac) \]
\[ (2, aab) \]

\[ (0, bca) \]
\[ (1, acb) \]

\[ (0, bca) \]
\[ (1, acb) \]
Contributed String Sorting Algorithms

- Parallel Super Scalar String Sample Sort (pS\(^5\))
  - fully parallel S\(^5\), sequential S\(^5\), and fast base case sorters
  - sequential running time of S\(^5\):
    \(O\left(\frac{D}{w} + n \log n\right)\) expected time with equality checks, and
    \(O\left(\left(\frac{D}{w} + n\right) \log v + n \log n\right)\) expected time with unrolled descents.
  - parallel running time of a single step of fully parallel S\(^5\):
    \(O\left(\frac{n}{p} \log v + \log p\right)\) time and \(O\left(n \log v + pv\right)\) work.

- Hybrid NUMA-aware pS\(^5\) + K-way LCP-Merge
- Parallel Multikey Quicksort
- Parallel Radix Sort (Adaptive 16-bit and 8-bit)

Additional Algorithms:

- (Parallel) Multiway LCP-aware Mergesort \(O(D + n \log n + \frac{n}{K})\)
- Sequential LCP-aware Insertion Sort \(O(D + n^2)\)
128 GiB GOV2 – Speedup on 32-Core Intel

Input characteristics: \( n = 3.1 \text{ G}, N = 128 \text{ Gi}, \frac{D}{N} = 82.7\% \).
Overview

Multi-Core Scalable String Sorting

External and Distributed Scalable Suffix Sorting

---

<table>
<thead>
<tr>
<th>1</th>
<th>alpha 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>arcade 0</td>
</tr>
<tr>
<td>0</td>
<td>kayak 0</td>
</tr>
<tr>
<td>1</td>
<td>kernel 0</td>
</tr>
<tr>
<td>1</td>
<td>kit 0</td>
</tr>
<tr>
<td>3</td>
<td>kitchen 0</td>
</tr>
<tr>
<td>3</td>
<td>kitten 0</td>
</tr>
<tr>
<td>1</td>
<td>krypton 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>⊥</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a $</td>
</tr>
<tr>
<td>1</td>
<td>abcba $</td>
</tr>
<tr>
<td>4</td>
<td>abcbaabcba $</td>
</tr>
<tr>
<td>0</td>
<td>b a $</td>
</tr>
<tr>
<td>2</td>
<td>bcba $</td>
</tr>
<tr>
<td>5</td>
<td>bcbacbacba $</td>
</tr>
<tr>
<td>0</td>
<td>cba $</td>
</tr>
<tr>
<td>3</td>
<td>cbacbacba $</td>
</tr>
</tbody>
</table>
Example $T = \[tobernottobes\]$  

<table>
<thead>
<tr>
<th>$i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>tobeornottobes $</td>
</tr>
<tr>
<td>1</td>
<td>obeornottobes $</td>
</tr>
<tr>
<td>2</td>
<td>beornottobes $</td>
</tr>
<tr>
<td>3</td>
<td>eornottobes $</td>
</tr>
<tr>
<td>4</td>
<td>ornottobes $</td>
</tr>
<tr>
<td>5</td>
<td>rnotobes $</td>
</tr>
<tr>
<td>6</td>
<td>notobes $</td>
</tr>
<tr>
<td>7</td>
<td>notobes $</td>
</tr>
<tr>
<td>8</td>
<td>tobes $</td>
</tr>
<tr>
<td>9</td>
<td>tobes $</td>
</tr>
<tr>
<td>10</td>
<td>othes $</td>
</tr>
<tr>
<td>11</td>
<td>be $</td>
</tr>
<tr>
<td>12</td>
<td>e $</td>
</tr>
<tr>
<td>13</td>
<td>$</td>
</tr>
</tbody>
</table>
Example $T = [tobeornottobe\$$]

<table>
<thead>
<tr>
<th>SA_i</th>
<th>LCP_i</th>
<th>$T_{SA_i...n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>-</td>
<td>$$$</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>be $$$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>beornottobe $$$</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>e $$$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>eornottobe $$$</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>nottobe $$$</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>obe $$$</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>obeornottobe $$$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>ornottobe $$$</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>ottoobe $$$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>rnottoobe $$$</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>tobe $$$</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>tobeornottobe $$$</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>tobe $$$</td>
</tr>
</tbody>
</table>
External Memory Algorithms

Prefix Doubling – [CF02]
\( O(sort(n) \cdot \log(maxlcp)) \) I/Os

Prefix Doubling – [DKMS05]
\( sort(5n) \cdot \log_2(maxlcp) + O(sort(n)) \) I/Os

DC7 – [DKMS05]
\( sort(24.75n) + \text{scan}(6n) \) I/Os

SA-IS

eSAIS – [B, Fischer, Osipov, ALENEX’13, JEA’16]
\( sort(17n) + \text{scan}(9n) \) I/Os

fSAIS – [KKPZ17]
\( O(\text{scan}(n) \cdot \log_2^{M/B}(n/B)) \) I/Os
Big Data Batch Processing

Our Requirements:
- compound primitives into complex algorithms
- efficient simple data types,
- overlap computation and communication,
- automatic disk usage,
- C++, and much more...

New Framework:
- Thrill

Lower Layers
- Apache Spark
- Apache Flink
- MapReduce

High Level Simple

Low Level Difficult
Distributed Immutable Array (DIA)

User Programmer’s View:

- DIA<T> = distributed array of items T on the cluster
- Cannot access items directly, instead use small set of scalable primitives, for example: Map, Sort, ReduceByKey, Zip, Window, etc.

Framework Designer’s View:

Goals: distribute work, optimize execution on cluster, add redundancy

⇒ build data-flow graph.

DIA<T> = pipelined chain of computations
Distributed Immutable Array (DIA)

User Programmer’s View:

- **DIA\(<T>** = **distributed array** of items T on the cluster
- Cannot access items directly, instead use small set of **scalable primitives**, for example: Map, Sort, ReduceBy...

![Diagram](image)

Framework Designer’s View:

- Goals: distribute work, optimize execution on cluster, add redundancy
  
  \[ \text{DIA}\(<T> = \text{pipelined chain of computations} \]

```
A
A. Map(·) := B
B. Sort(·) := C
```

```
A
B := A. Map()
C := B. Sort()
C
```
Distributed Immutable Array (DIA)

User Programmer’s View:
- $\text{DIA}<T> = \text{distributed array}$ of items $T$ on the cluster
- Cannot access items directly, instead use small set of scalable primitives, for example: $\text{Map}$, $\text{Sort}$, $\text{ReduceByKey}$, $\text{Zip}$, $\text{Window}$, etc.

Framework Designer’s View:
- Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Rightarrow$ build data-flow graph.
- $\text{DIA}<T> = \text{pipelined chain of computations}$
Thrill’s Goal and Current Status

An easy way to program fast distributed algorithms in C++.

Current Status:

- \( \approx 60 \) K lines of C++14 code, 70–80 \% written by B, \( \geq 12 \) contributors
- Published at IEEE Conference on Big Data [B, et al. ’16]

Case Studies:

- Five suffix sorting algorithms [B, Gog, Kurpicz, arXiv’17]
- Louvain graph clustering algorithm [Hamann et al. arXiv’17]
- More examples: stochastic gradient descent, triangle counting, etc.
- Future: fault tolerance, scalability, and more applications.
Data-Flow Graph of DC3 with Recursion

construct lexicographic names

\[ T_3 := T.\text{FlatWindow}_3 \]
\[ S := T_3.\text{Sort} \]
\[ N' := S.\text{FlatWindow}_2 \]
\[ N := N'.\text{PrefixSum} \]
\[ IS := S.\text{Map} \]
\[ TR := \text{Zip}(\{ IS, N \}) \]
\[ NR := \text{ZipWindow}_{[3,2]}(\{ T, IR \}) \]
\[ Z := NR.\text{Window}_2 \]
\[ Z' := Z'.\text{Window}_2 \]
\[ S'_{0} := Z.\text{Map} \]
\[ S_{1} := Z.\text{Map} \]
\[ S'_{2} := Z.\text{Map} \]
\[ S_{0} := S'_{0}.\text{Sort} \]
\[ S_{1} := S'_{1}.\text{Sort} \]
\[ S_{2} := S'_{2}.\text{Sort} \]
\[ Merge(\{ S_{0}, S_{1}, S_{2} \}) \]

recursion and calculate ranks

\[ N := N'.\text{PrefixSum} \]
\[ IS := S.\text{Map} \]
\[ TR := \text{Zip}(\{ IS, N \}) \]
\[ TR' := TR.\text{Sort} \]
\[ SA_R := \text{DC3}(TR'.\text{Map}) \]
\[ IR := SA_R.\text{ZipWithIndex} \]
\[ IR := IR.\text{Sort}.\text{Map} \]
\[ SA_T := \text{Merge}(\{ S_{0}, S_{1}, S_{2} \}) \]

create tuples and merge suffix array
Suffix Sorting Wikipedia with 32 Hosts

Run on 32 × i3.4xlarge AWS EC2 instances containing 16-core Intel Xeon E5-2686 CPUs with 2.30 GHz, 8 GB of RAM, and 2 × 1.9 TB NVMe SSDs.
Overview: Main Contributions

Multi-Core Scalable String Sorting

- Parallel Super Scalar String Sample Sort (pS$^5$) [BS13]
- Parallel Multiway LCP-Merge, Merge Sort, and More [BES17]

External and Distributed Scalable Suffix Sorting

- Induced Sorting in External Memory: eSAIS [BFO13, BFO16]
- New High-Performance Distributed Framework in C++: Thrill [BAJ+16]

Distributed External Suffix Sorting