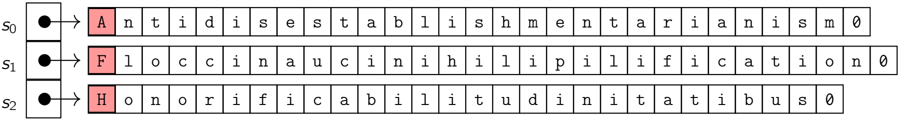
Karlsruhe Institute of Technology

Communication-Efficient String Sorting

Timo Bingmann, Peter Sanders, Matthias Schimek · 2020-05-18 @ IPDPS'20

Institute of Theoretical Informatics – Algorithmics



Abstract

There has been surprisingly little work on algorithms for sorting strings on distributed-memory parallel machines. We develop efficient algorithms for this problem based on the multi-way merging principle. These algorithms inspect only characters that are needed to determine the sorting order. Moreover, communication volume is reduced by also communicating (roughly) only those characters and by communicating repetitions of the same prefixes only once. Experiments on up to 1280 cores reveal that these algorithm are often more than five times faster than previous algorithms.

Why String Sorting?

- string: array of characters over alphabet Σ

s	t	r	i	n	g	0

- sorted string set: sorted lexicographically
  $\Rightarrow$ like in a dictionary
- characteristics of string sets
  - #strings n, #characters N
  - sum distinguishing prefix lengths D

  $\Rightarrow$ multidimensional data

s_0	a	l	g	o	r	i	t	h	m	0	
s_1	c	o	m	p	a	r	e	0			
s_2	c	o	m	p	a	r	i	s	o	n	0
s_3	p	r	e	f	i	x	0				

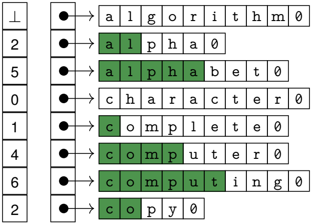
- only published distributed string sorting algorithm: one paragraph in [Fischer and Kurpicz, ALENEX'19]

String Sorting Toolbox

- **Sequential Sorting:** String Radix Sort, Multikey Quicksort, ...

  [Kärkkäinen et al., SPIRE'08], [Bentley and Sedgewick, SODA'97]

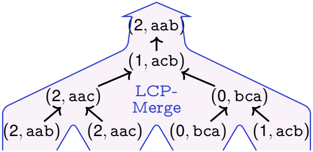
  - evaluation of many sequential algorithms in [Bingmann '18]
  - needed: string sorting + Longest Common Prefix (LCP) array computation



- **Multiway Merging: LCP Losertree**

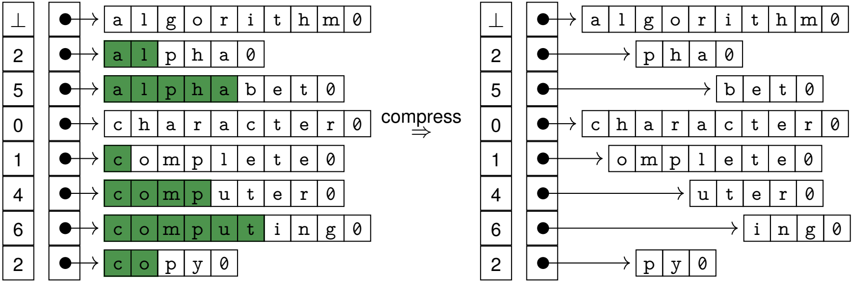
  [Bingmann et. al, Algorithmica'17]

  - exploit LCP values to save character-comparisons



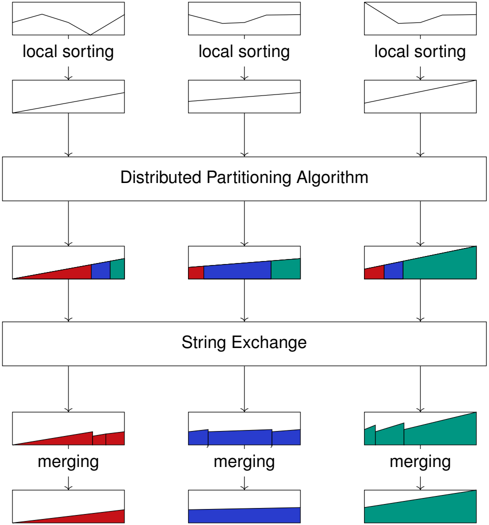
String Sorting Toolbox

LCP Compression



- each longest common prefix is sent only once
- compression: iterate over strings + LCP array
- decompression: iterate over compressed strings + LCP array

Distributed Merge String Sort (MS)



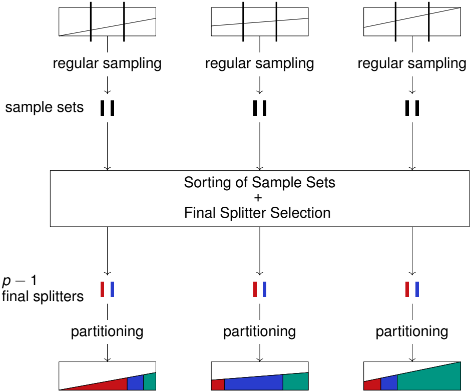
- **Local Sorting**
  - String Radix Sort

  new: String Radix Sort + LCP array
- **String Exchange**
  - no compression

  new: LCP compression
- **Merging**
  - plain losertree

  new: LCP losertree

Distributed Merge String Sort (MS)

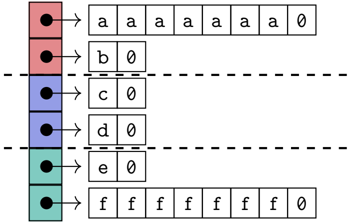


- **Partitioning**
  - equidistant sampling
  - gather + seq. sort

  new: hypercube quicksort
  [Axtmann and Sanders, ALENEX'17]
  - broadcast final splitters
  - partitioning

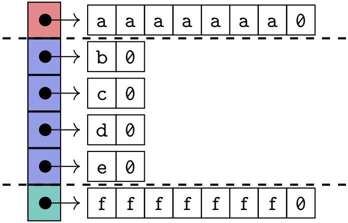
Partitioning – Sampling Approaches

string-based sampling



- **Goal:** equal number of strings per bucket
- sampling of string array
- provable upper bounds

character-based sampling



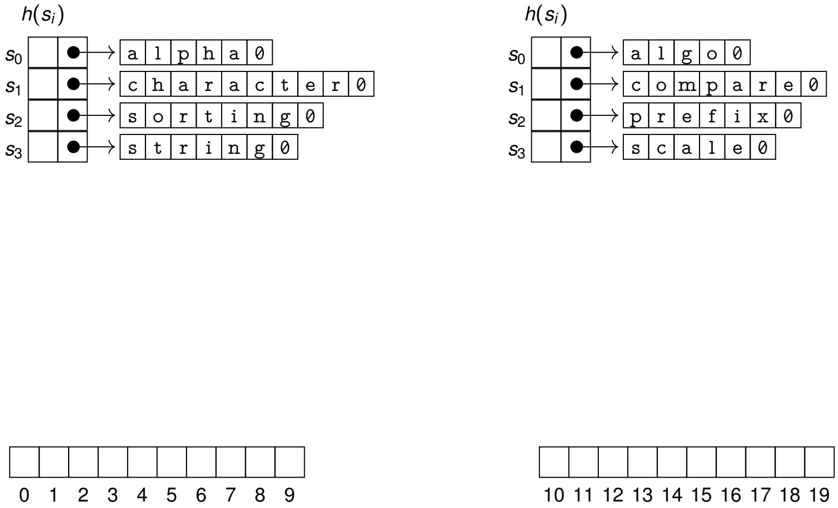
- **Goal:** equal number of characters per bucket
- sampling of character array
- provable upper bounds

Prefix Doubling String Merge Sort (PDMS)

PE1:	A	n	t	i	d	i	s	e	s	t	a	b	l	i	s	h	m	e	n	t	a	r	i	a	n	i	s	m	0	
PE2:	F	l	o	c	c	i	n	a	u	c	i	n	i	h	i	l	i	p	i	l	i	f	i	c	a	t	i	o	n	0
PE3:	H	o	n	o	r	i	f	i	c	a	b	i	l	i	t	u	d	i	n	i	t	a	t	i	b	u	s	0		

- same main structure as before
- use distributed Single-Shot Bloom Filter (dSBF) to approximate distinguishing prefixes with distributed duplicate detection [Sanders et al., IEEE BigData'13]
- only operate on those characters
- calculate only the permutation for sorting (exchanging further characters is optional).

Distinguishing Prefix Computation



Distinguishing Prefix Computation

Karlsruhe Institute of Technology

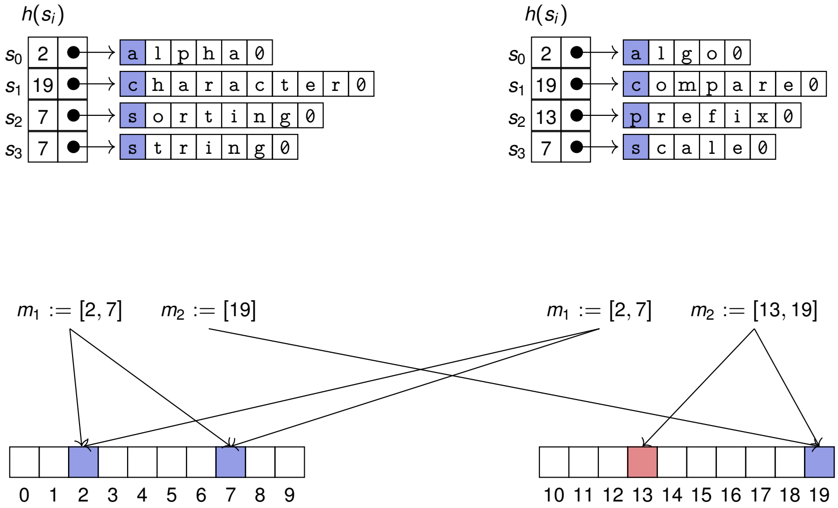
	$h(s_i)$											
s_0	2	• →	a	l	p	h	a	0				
s_1	19	• →	c	h	a	r	a	c	t	e	r	0
s_2	7	• →	s	o	r	t	i	n	g	0		
s_3	7	• →	s	t	r	i	n	g	0			

	$h(s_i)$										
s_0	2	• →	a	l	g	o	0				
s_1	19	• →	c	o	m	p	a	r	e	0	
s_2	13	• →	p	r	e	f	i	x	0		
s_3	7	• →	s	c	a	l	e	0			

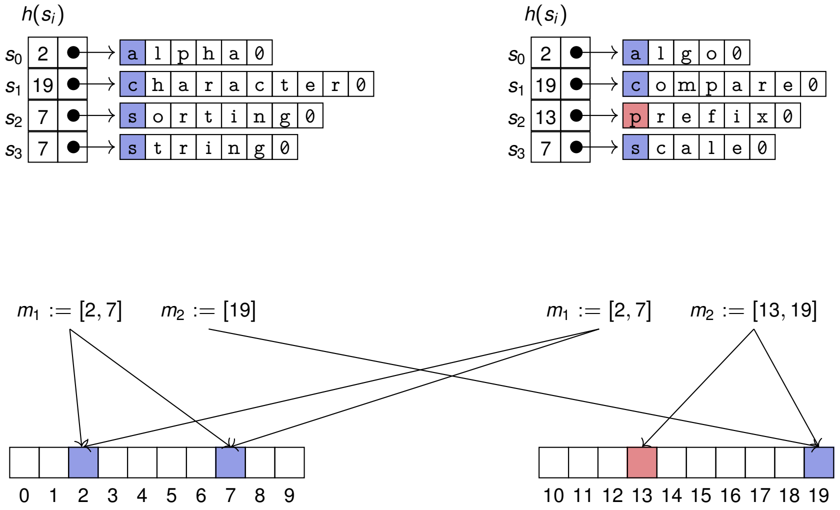
0	1	2	3	4	5	6	7	8	9

10	11	12	13	14	15	16	17	18	19

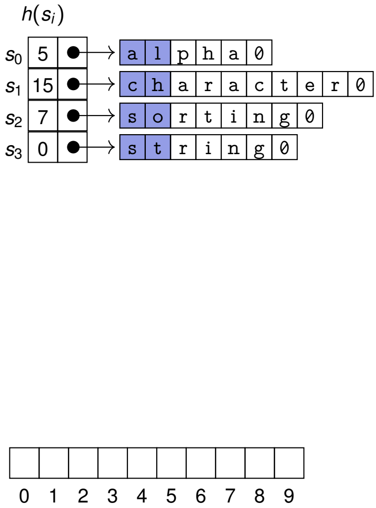
Distinguishing Prefix Computation

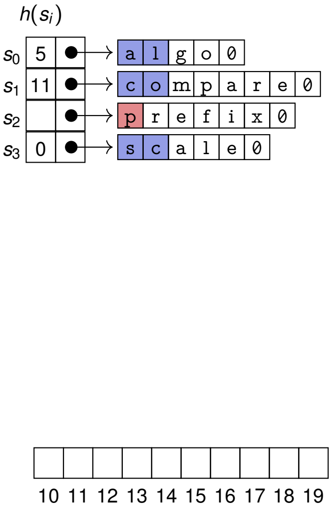


Distinguishing Prefix Computation

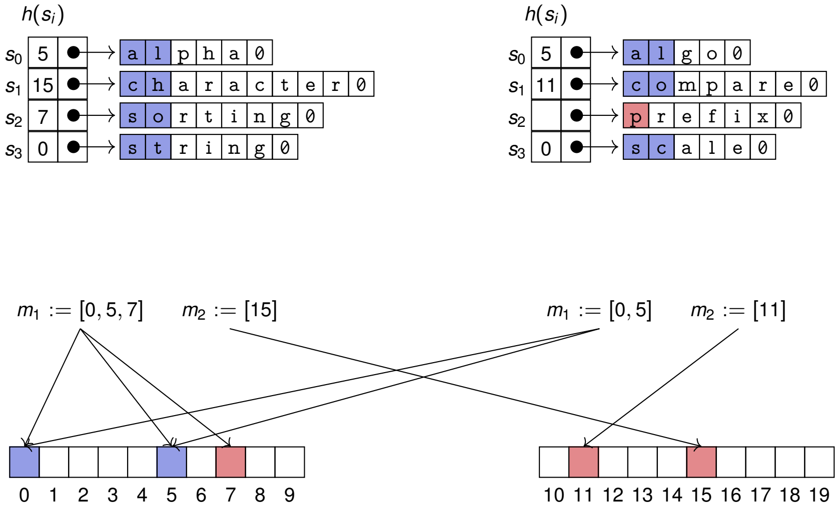


Distinguishing Prefix Computation

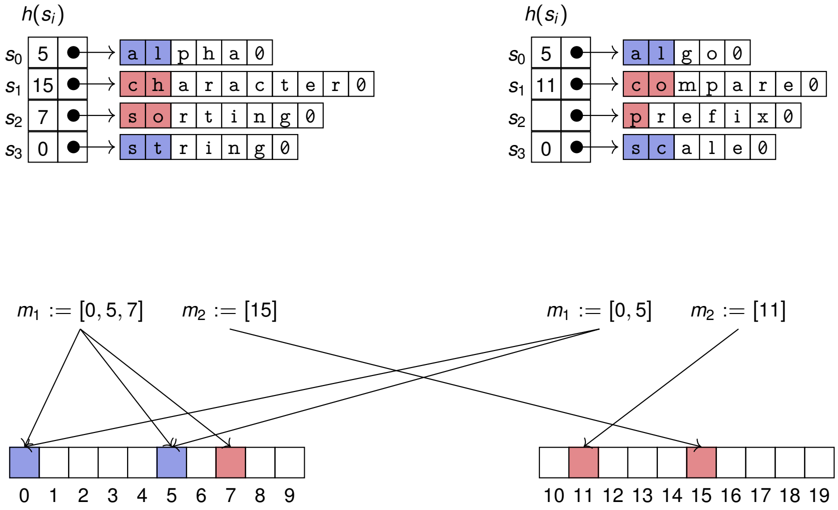




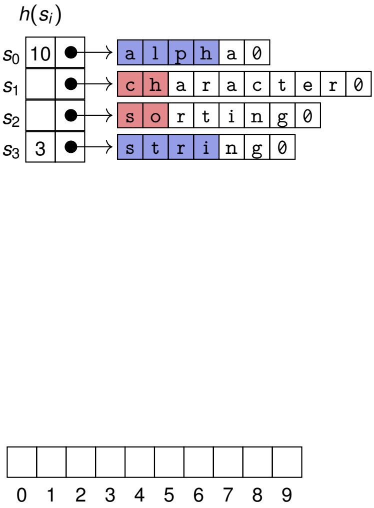
Distinguishing Prefix Computation

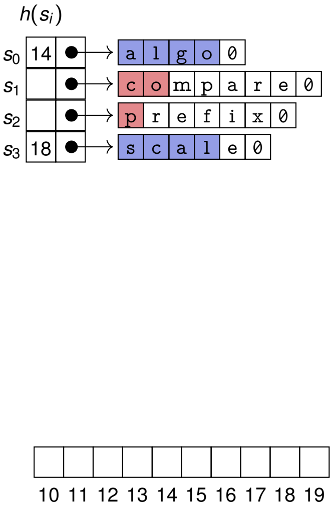


Distinguishing Prefix Computation

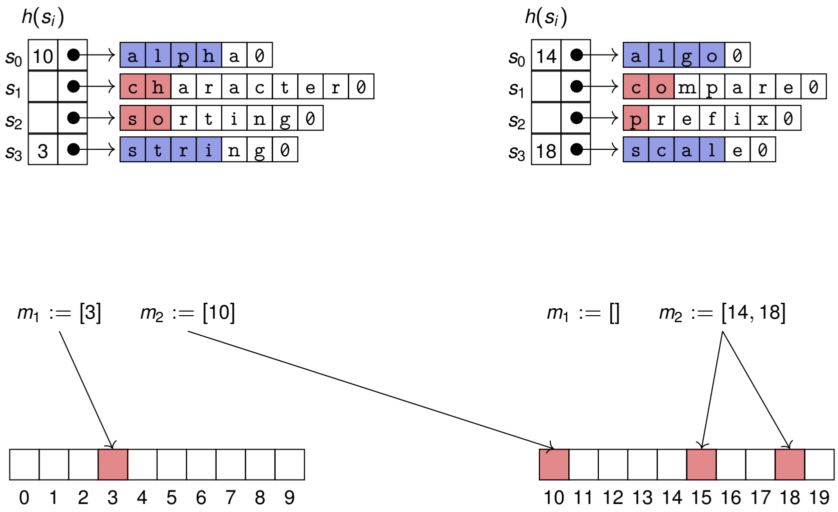


Distinguishing Prefix Computation

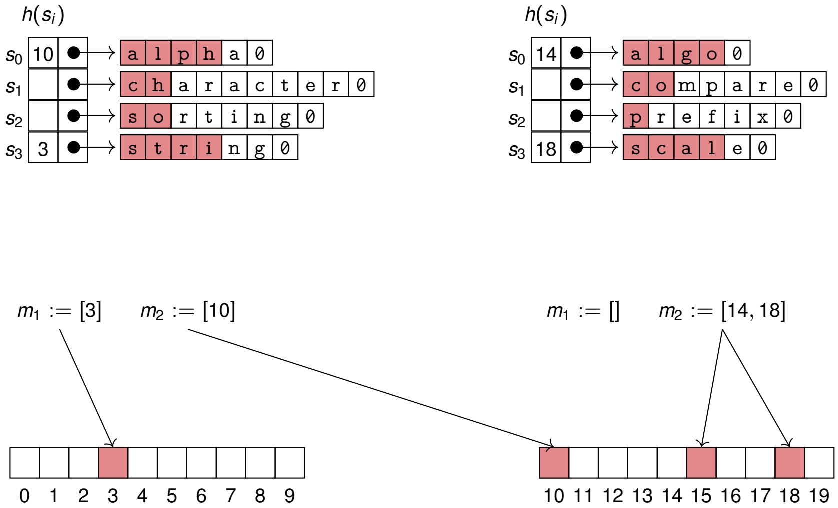




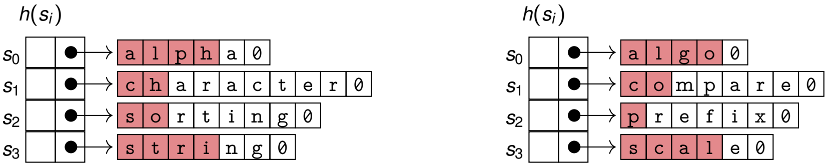
Distinguishing Prefix Computation



Distinguishing Prefix Computation



Distinguishing Prefix Computation



Implementation Remarks on dSBF

- Golomb encoding for hash values
- no need to materialize Bloom filter
  $\Rightarrow$ merge received sequences

Experimental Evaluation – Setup

Input Data

- weak scaling with *D*/*N*-Generator
- strong scaling with CommonCrawl and DnaReads

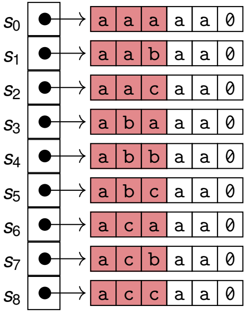
Hardware (ForHLR I at KIT)

- 2 Deca-core Intel Xeon E5-2670 v2 (2.5 GHz) and
- 64 GB RAM per compute node
- InfiniBand 4X FDR interconnect

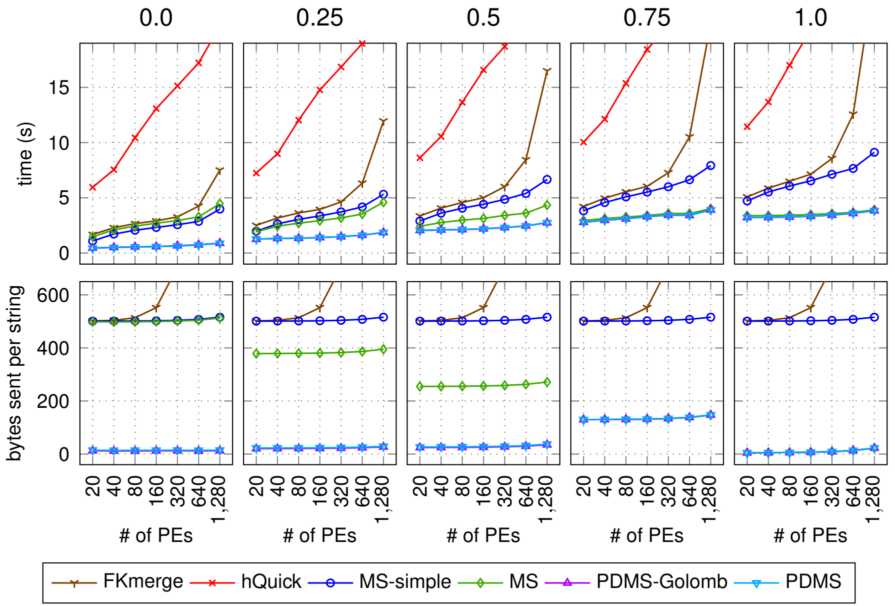
Algorithms

- FKmerge: from Fischer and Kurpicz [ALENEX'19]
- hQuick: distributed quicksort
- our merge sort: MS-simple (no LCP-comp), MS (LCP-comp)
- our prefix doubling merge sort: PDMS-Golomb, PDMS

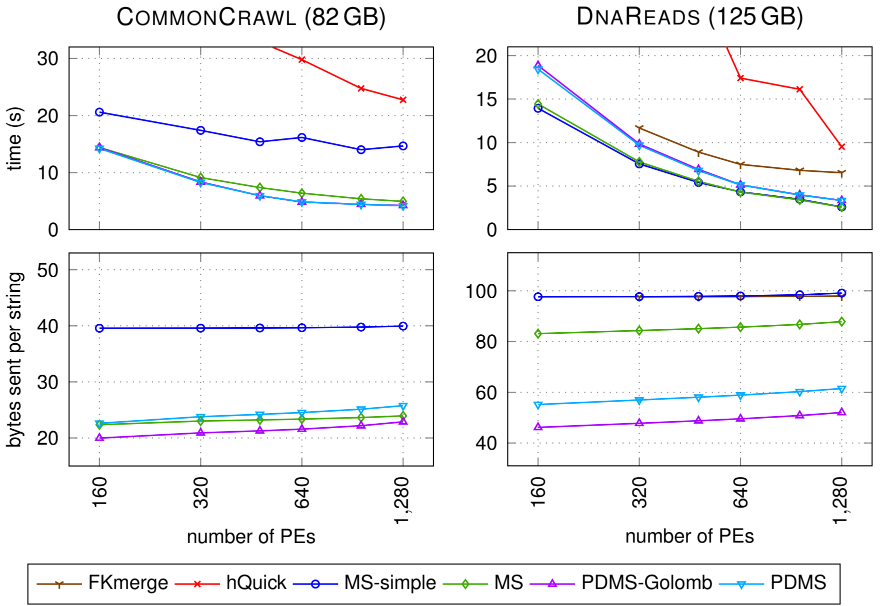
D/N-Generator
(n=9, ℓ=6, D/N=0.5)



D/N-Generator(n=p·500K, ℓ=500, D/N=?)



Strong Scaling with Real-World Inputs



Conclusion

Karlsruhe Institute of Technology

Summary

- two new communication-efficient string sorting algorithms:
  - distributed string merge sort (MS)
  - distributed prefix-doubling string merge sort (PDMS)
- theory and experimental evaluation
- different strategies best for low and high D/N-ratios
- Source code and recording of talk:
  https://panthema.net/2020/0518-distributed-string-sorting

Future Work

- improve balancing by considering strings and characters
- can one show lower bounds?

Questions via email to bingmann@kit.edu