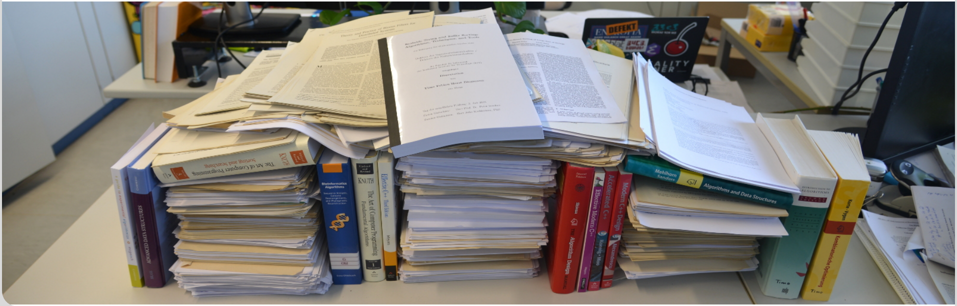
Scalable String and Suffix Sorting: Algorithms, Techniques, and Tools

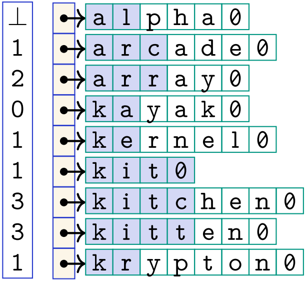
Timo Bingmann · Dissertation Defense · July 3rd, 2018

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS

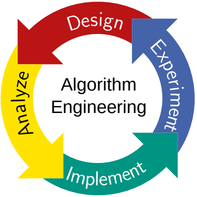


Overview

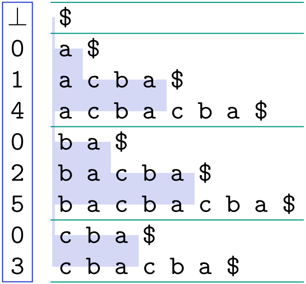
Multi-Core Scalable String Sorting



⊥	a l p h a 0
1	a r c a d e 0
2	a r r a y 0
0	k a y a k 0
1	k e r n e l 0
1	k i t 0
3	k i t c h e n 0
3	k i t t e n 0
1	k r y p t o n 0

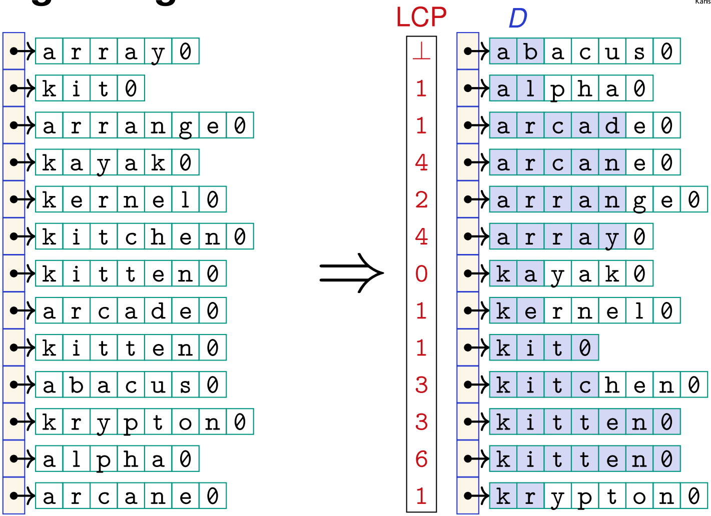


External and Distributed Scalable Suffix Sorting



⊥	$
0	a $
1	a c b a $
4	a c b a c b a $
0	b a $
2	b a c b a $
5	b a c b a c b a $
0	c b a $
3	c b a c b a $

Sorting Strings



array0
kit0
arrange0
kayak0
kernel0
kitchen0
kitten0
arcade0
kitten0
abacus0
krypton0
alpha0
arcane0

⇒

LCP	*D*
⊥	abacus0
1	alpha0
1	arcade0
4	arcane0
2	arrange0
4	array0
0	kayak0
1	kernel0
1	kit0
3	kitchen0
3	kitten0
6	kitten0
1	krypton0

Input: *n* strings containing *N* characters in total.

String Sorting Algorithms

Karlsruhe Institute of Technology

Theoretical Parallel Algorithms

- "Optimal Parallel String Algorithms: . . ." [Hagerup '94]
  $\mathcal{O}(\log N / \log\log N)$ time and $\mathcal{O}(N \log\log N)$ work on CRCW PRAM

Existing Basic Sequential Algorithms

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

Existing Algorithm Library

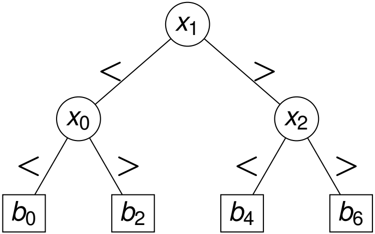
- in C/C++ by Rantala (for Engineering Radix Sort [Kärkkäinen, Rantala '09])

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)

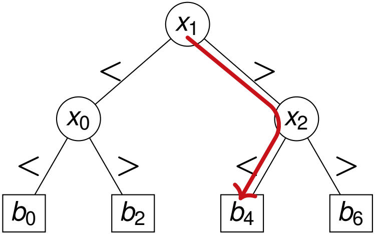
```
array0
kit0
arrange0
kayak0
kernel0
kitchen0
kitten0
arcade0
kite0
abacus0
krypton0
alpha0
arcane0
```



based on Super Scalar Sample Sort
[Sanders, Winkel '04]

Super Scalar String Sample Sort (S^5)

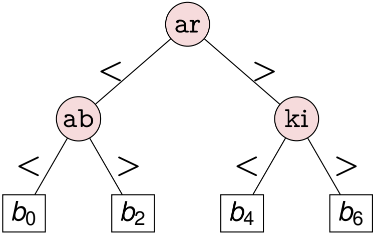
array0
kit0
arrange0
kayak0
kernel0
kitchen0
kitten0
arcade0
kite0
abacus0
krypton0
alpha0
arcane0



based on Super Scalar Sample Sort
[Sanders, Winkel ’04]

Super Scalar String Sample Sort (S^5)

```
array0
kit0
arrange0
kayak0
kernel0
kitchen0
kitten0
arcade0
kite0
abacus0
krypton0
alpha0
arcane0
```



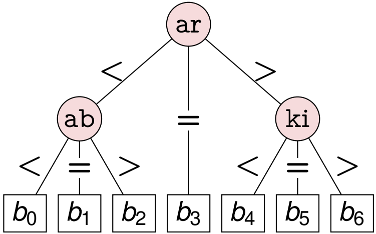
- partition by *w* chars
- store in level-order and use predicated instructions

1	2	3
ar	ab	ki

$i := 2i + 0/1$

Super Scalar String Sample Sort (S^5)

array0
kit0
arrange0
kayak0
kernel0
kitchen0
kitten0
arcade0
kite0
abacus0
krypton0
alpha0
arcane0



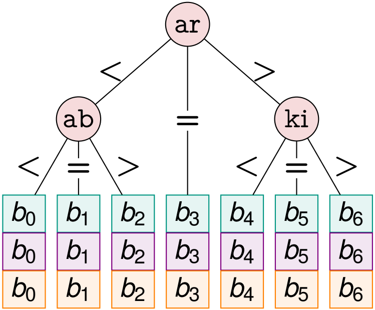
- equality checking:
  1. at each splitter
  2. after full descent
- interleave tree descents:
  classify four strings at once
  ⇒ super scalar parallelism

Super Scalar String Sample Sort (S^5)

array0
kit0
arrange0
kayak0
kernel0

kitchen0
kitten0
arcade0
kite0
abacus0

krypton0
alpha0
arcane0

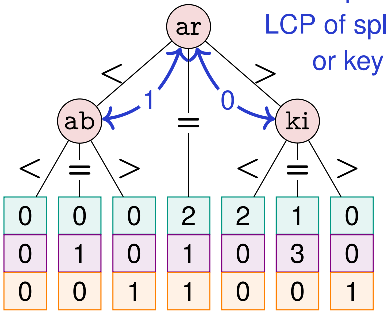


ar
<
=
>
ab
ki
<
=
>
<
=
>
b_0 b_1 b_2 b_3 b_4 b_5 b_6
b_0 b_1 b_2 b_3 b_4 b_5 b_6
b_0 b_1 b_2 b_3 b_4 b_5 b_6

- easy parallelization
- classification tree in L2 caches of processors

Super Scalar String Sample Sort (S^5)

	prefix
abacus0	2
alpha0	1
array0 arrange0 arcade0 arcane0	2
kayak0 kernel0	0
kit0 kitchen0 kitten0 kite0	2
krypton0	0

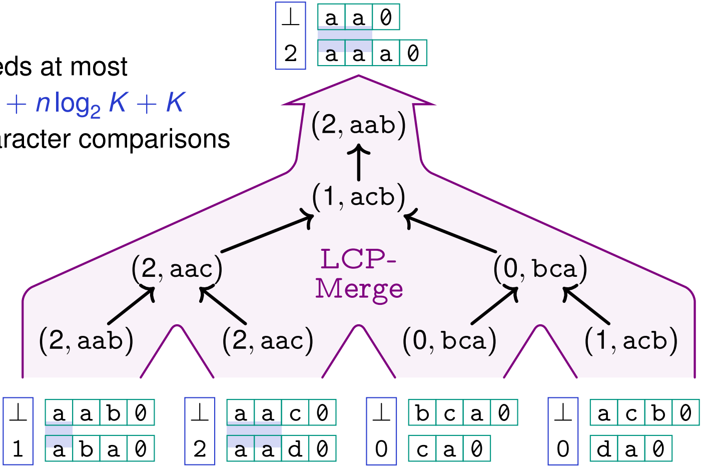


increase prefix by LCP of splitters or key size.

- reorder out-of-place, in-place, and/or in parallel
- top-level algorithm in parallel S^5

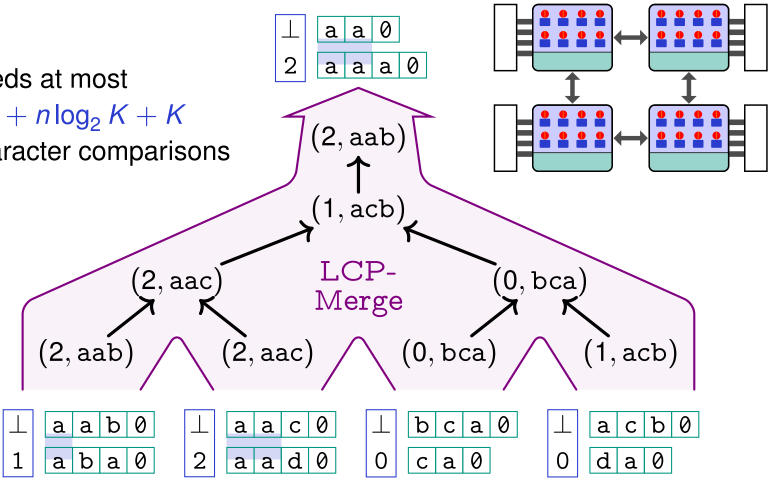
LCP Loser Tree – *K*-way LCP-Merge

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



LCP Loser Tree – *K*-way LCP-Merge

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



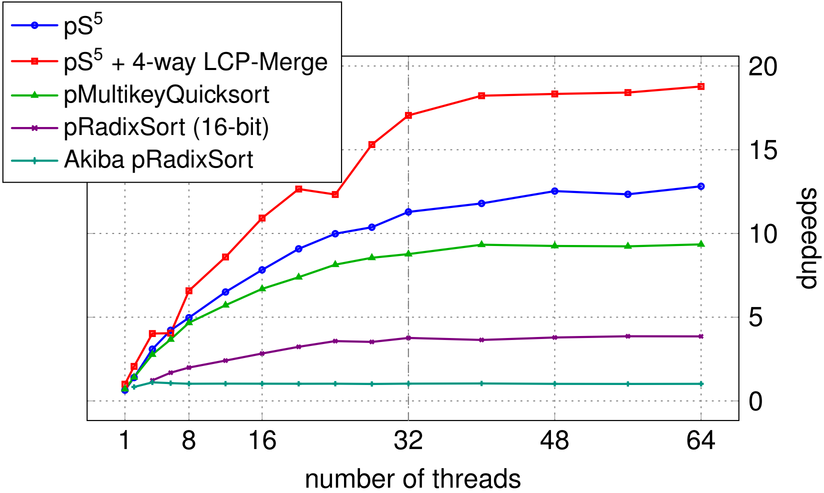
Contributed String Sorting Algorithms

- 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:
    $\mathcal{O}(\frac{D}{w} + n \log n)$ expected time with equality checks, and
    $\mathcal{O}((\frac{D}{w} + n) \log v + n \log n)$ expected time with unrolled descents.
  - parallel running time of a single step of fully parallel S^5:
    $\mathcal{O}(\frac{n}{p} \log v + \log p)$ time and $\mathcal{O}(n \log v + pv)$ 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 $\mathcal{O}(D + n \log n + \frac{n}{K})$
- Sequential LCP-aware Insertion Sort $\mathcal{O}(D + n^2)$

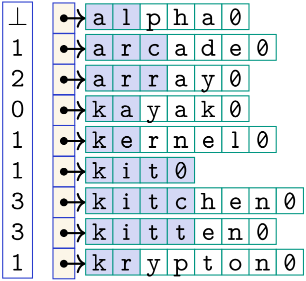
128 GiB GOV2 – Speedup on 32-Core Intel



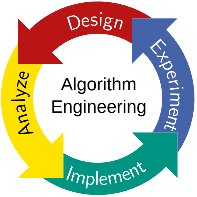
Input characteristics: $n = 3.1$ G, $N = 128$ Gi, $\frac{D}{N} = 82.7\,\%$.

Overview

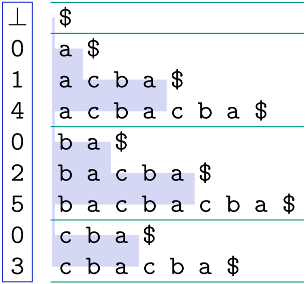
Multi-Core Scalable String Sorting



⊥	alpha0
1	arcade0
2	array0
0	kayak0
1	kernel0
1	kit0
3	kitchen0
3	kitten0
1	krypton0



External and Distributed Scalable Suffix Sorting



⊥	$
0	a$
1	acba$
4	acbacba$
0	ba$
2	bacba$
5	bacbacba$
0	cba$
3	cbacba$

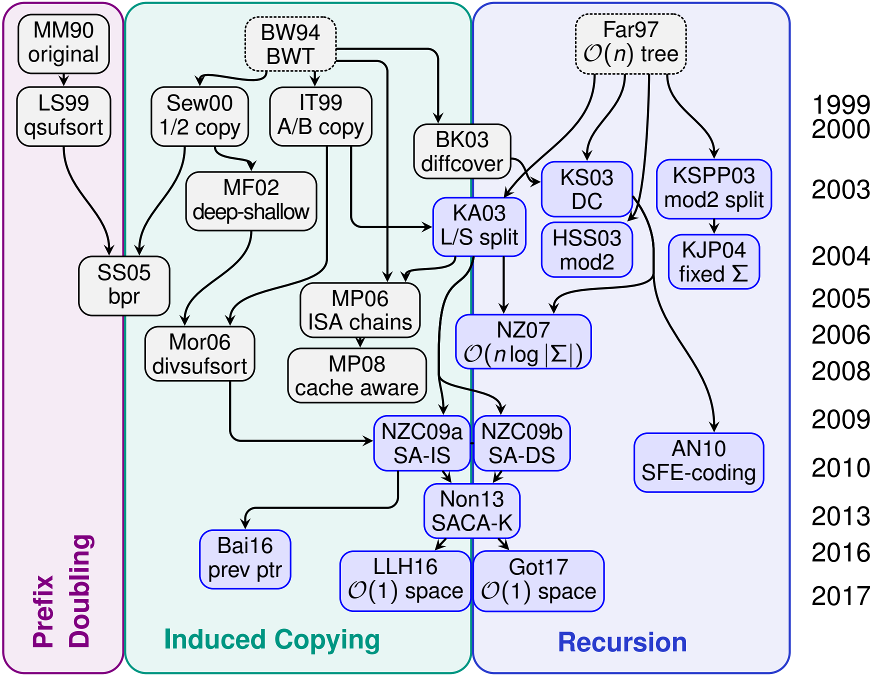
0 1 2 3 4 5 6 7 8 9 10 11 12 13

Example $T = [\texttt{tobeornottobe\$}]$

i	T_i
0	t o b e o r n o t t o b e $
1	o b e o r n o t t o b e $
2	b e o r n o t t o b e $
3	e o r n o t t o b e $
4	o r n o t t o b e $
5	r n o t t o b e $
6	n o t t o b e $
7	o t t o b e $
8	t t o b e $
9	t o b e $
10	o b e $
11	b e $
12	e $
13	$

Example $T = \left[\begin{array}{cccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ \mathtt{t} & \mathtt{o} & \mathtt{b} & \mathtt{e} & \mathtt{o} & \mathtt{r} & \mathtt{n} & \mathtt{o} & \mathtt{t} & \mathtt{t} & \mathtt{o} & \mathtt{b} & \mathtt{e} & \mathtt{\$} \end{array}\right]$

SA_i	LCP_i	$T_{SA_i \ldots n}$
13	-	$
11	0	be$
2	2	beornottobe$
12	0	e$
3	1	eornottobe$
6	0	nottobe$
10	0	obe$
1	3	obeornottobe$
4	1	ornottobe$
7	1	ottobe$
5	0	rnottobe$
9	1	tobe$
0	4	tobeornottobe$
8	1	ttobe$



Prefix Doubling

MM90 original

LS99 qsufsort

SS05 bpr

Induced Copying

BW94 BWT

Sew00 1/2 copy

IT99 A/B copy

MF02 deep-shallow

Mor06 divsufsort

MP06 ISA chains

MP08 cache aware

BK03 diffcover

NZC09a SA-IS

Bai16 prev ptr

LLH16 $\mathcal{O}(1)$ space

Recursion

Far97 $\mathcal{O}(n)$ tree

KA03 L/S split

KS03 DC

HSS03 mod2

KSPP03 mod2 split

KJP04 fixed Σ

NZ07 $\mathcal{O}(n \log |\Sigma|)$

NZC09b SA-DS

AN10 SFE-coding

Non13 SACA-K

Got17 $\mathcal{O}(1)$ space

1999

2000

2003

2004

2005

2006

2008

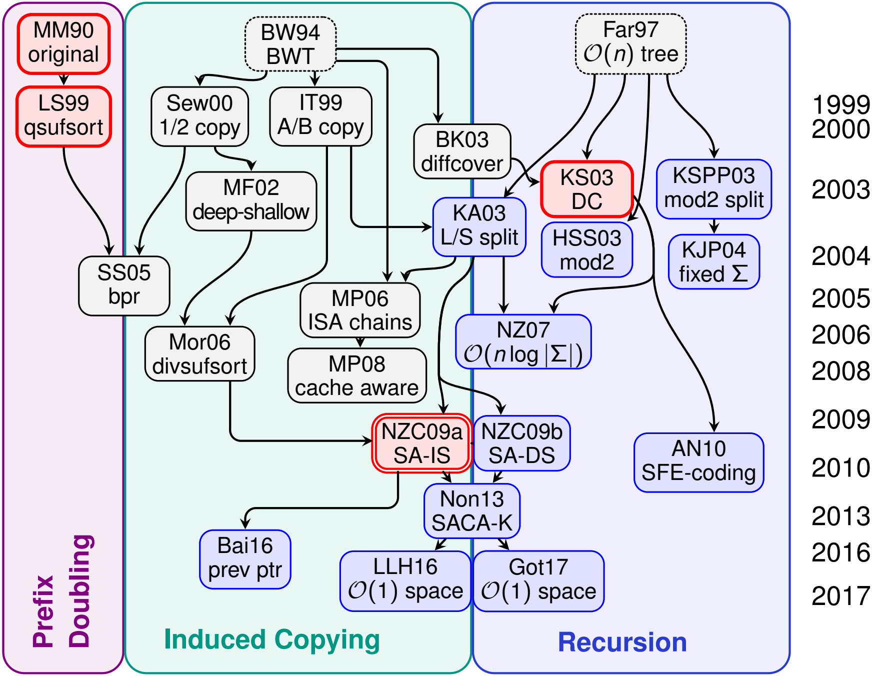
2009

2010

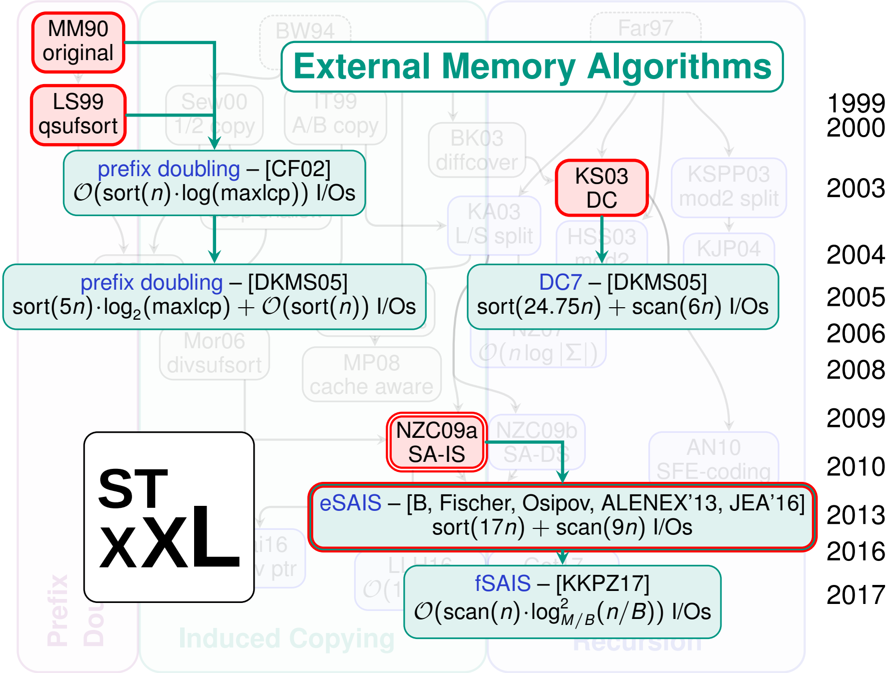
2013

2016

2017


MM90
original
LS99
qsufsort
SS05
bpr
Prefix Doubling
BW94
BWT
Sew00
1/2 copy
IT99
A/B copy
MF02
deep-shallow
BK03
diffcover
MP06
ISA chains
MP08
cache aware
Mor06
divsufsort
NZC09a
SA-IS
Bai16
prev ptr
LLH16
$\mathcal{O}(1)$ space
Induced Copying
Far97
$\mathcal{O}(n)$ tree
KS03
DC
KSPP03
mod2 split
KA03
L/S split
HSS03
mod2
KJP04
fixed Σ
NZ07
$\mathcal{O}(n \log |\Sigma|)$
NZC09b
SA-DS
AN10
SFE-coding
Non13
SACA-K
Got17
$\mathcal{O}(1)$ space
Recursion
1999
2000
2003
2004
2005
2006
2008
2009
2010
2013
2016
2017

External Memory Algorithms



MM90 original

LS99 qsufsort

prefix doubling – [CF02]
$\mathcal{O}(\text{sort}(n) \cdot \log(\text{maxlcp}))$ I/Os

prefix doubling – [DKMS05]
$\text{sort}(5n) \cdot \log_2(\text{maxlcp}) + \mathcal{O}(\text{sort}(n))$ I/Os

KS03 DC

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

NZC09a SA-IS

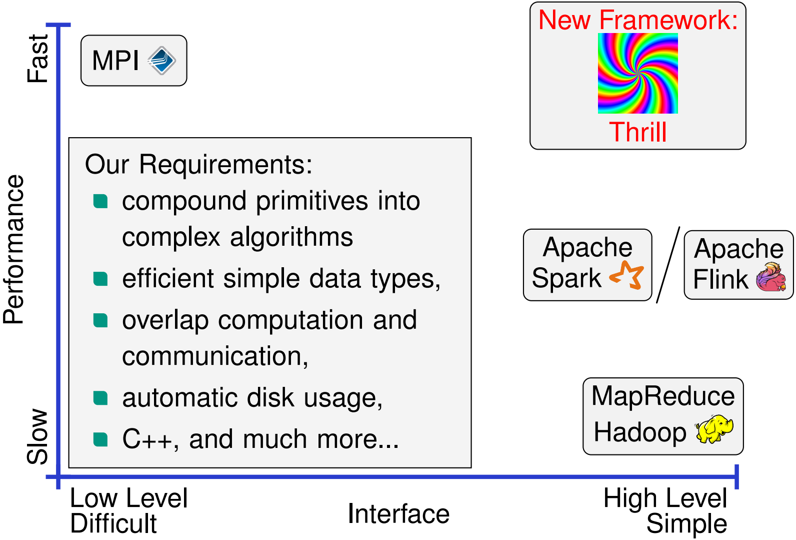
eSAIS – [B, Fischer, Osipov, ALENEX'13, JEA'16]
$\text{sort}(17n) + \text{scan}(9n)$ I/Os

fSAIS – [KKPZ17]
$\mathcal{O}(\text{scan}(n) \cdot \log^2_{M/B}(n/B))$ I/Os

ST xXL

bwUniCluster
© KIT (SCC)

Big Data Batch Processing



Fast

Performance

Slow

MPI

New Framework:

Thrill

Apache Spark / Apache Flink

MapReduce Hadoop

Low Level Difficult

Interface

High Level Simple

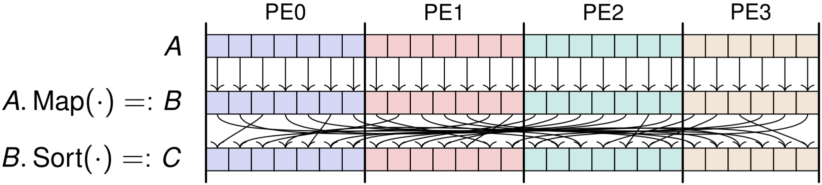
Our Requirements:

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

Distributed Immutable Array (DIA)

User Programmer's View:

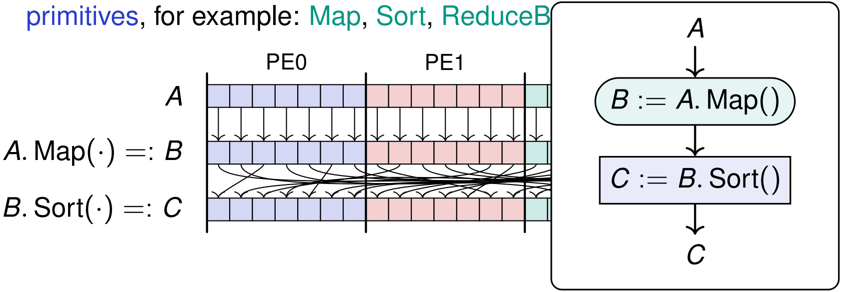
- DIA<T> = distributed array of items T on the cluster
- Cannot access items directly, instead use small set of scalable primitives, for example: Map, Sort, ReduceByKey, Zip, Window, etc.



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, ReduceB



A

$B := A.\mathrm{Map}()$

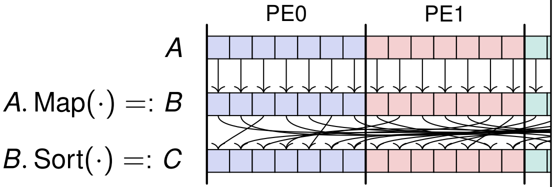
$C := B.\mathrm{Sort}()$

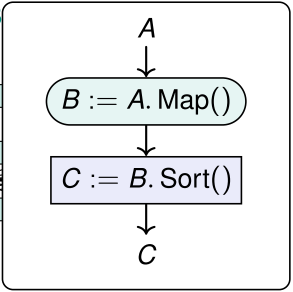
C

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, ReduceB





Framework Designer's View:

- Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Longrightarrow$ build data-flow graph.
- `DIA<T>` = pipelined chain of computations

Thrill's Goal and Current Status

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

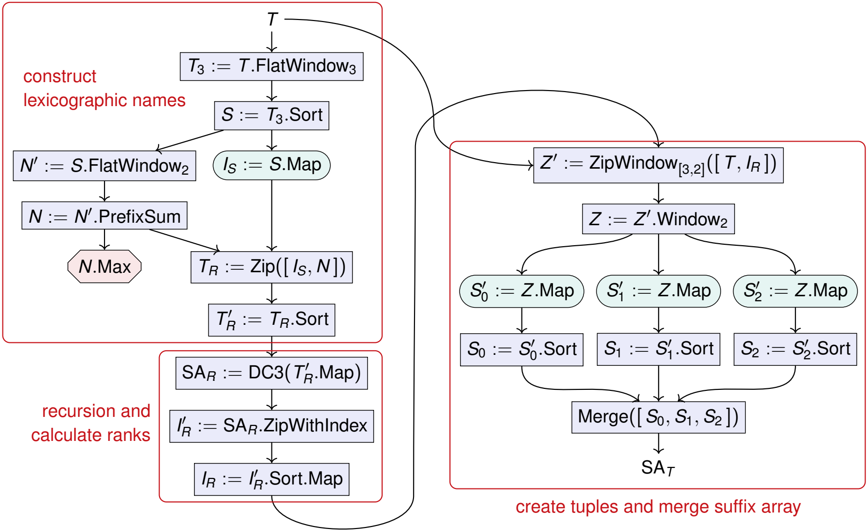
Current Status:

- Open-source prototype at http://github.com/thrill/thrill.
- $\approx$ 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]
- Faster than Apache Spark and Apache Flink on five micro benchmarks: WordCount1000, WordCountCC, PageRank, TeraSort, and K-Means.

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.

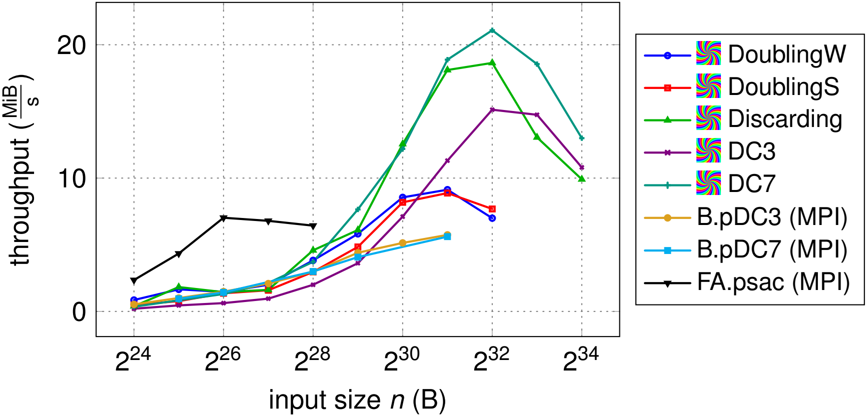
Data-Flow Graph of DC3 with Recursion



T
$T_3 := T.\text{FlatWindow}_3$
construct lexicographic names
$S := T_3.\text{Sort}$
$N' := S.\text{FlatWindow}_2$
$I_S := S.\text{Map}$
$N := N'.\text{PrefixSum}$
$N.\text{Max}$
$T_R := \text{Zip}([\,I_S, N\,])$
$T'_R := T_R.\text{Sort}$
$\text{SA}_R := \text{DC3}(T'_R.\text{Map})$
recursion and calculate ranks
$I'_R := \text{SA}_R.\text{ZipWithIndex}$
$I_R := I'_R.\text{Sort.Map}$
$Z' := \text{ZipWindow}_{[3,2]}([\,T, I_R\,])$
$Z := Z'.\text{Window}_2$
$S'_0 := Z.\text{Map}$
$S'_1 := Z.\text{Map}$
$S'_2 := Z.\text{Map}$
$S_0 := S'_0.\text{Sort}$
$S_1 := S'_1.\text{Sort}$
$S_2 := S'_2.\text{Sort}$
$\text{Merge}([\,S_0, S_1, S_2\,])$
SA_T

create tuples and merge suffix array

Suffix Sorting Wikipedia with 32 Hosts

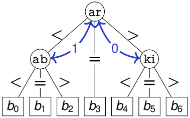


Run on 32 × i3.4xlarge AWS EC2 instances containing 16-core Intel Xeon E5-2686 CPUs with 2.30 GHz, 8 GB of RAM, and 2 × 1.9 TB NVMe SSDs.

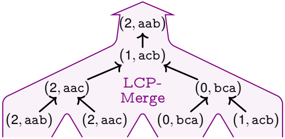
Overview: Main Contributions

Multi-Core Scalable String Sorting

- Parallel Super Scalar String Sample Sort (pS^5) [BS13]

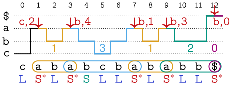


- Parallel Multiway LCP-Merge, Merge Sort, and More [BES17]



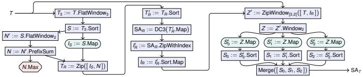
External and Distributed Scalable Suffix Sorting

- Induced Sorting in External Memory: eSAIS [BFO13, BFO16]



- New High-Performance Distributed Framework in C++: Thrill [BAJ+16]

- Distributed External Suffix Sorting



T
$T_3 := T$.FlatWindow$_3$
$S := T_3$.Sort
$N' := S$.FlatWindow$_2$
$I_S := S$.Map
$N := N'$.PrefixSum
N.Max
$T_R :=$ Zip([I_S, N])
$T'_R := T_R$.Sort
SA$_R :=$ DC3(T'_R.Map)
$I'_R :=$ SA$_R$.ZipWithIndex
$I_R := I'_R$.Sort.Map
$Z' :=$ ZipWindow$_{[3,2]}$([T, I_R])
$Z := Z'$.Window$_2$
$S'_0 := Z$.Map
$S'_1 := Z$.Map
$S'_2 := Z$.Map
$S_0 := S'_0$.Sort
$S_1 := S'_1$.Sort
$S_2 := S'_2$.Sort
Merge([S_0, S_1, S_2])
SA$_T$