Karlsruhe Institute of Technology

# Massive Suffix Array Construction with Thrill

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students | @ SPP 1736

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

# Example $T = [\texttt{dbadcbccbabdcc\$}]$

| $\mathrm{SA}_\mathrm{i}$ | $T_{\mathrm{SA}_\mathrm{i}\ldots\mathrm{n}}$ |
|---|---|
| 14 | $ |
| 9 | a b d c c $ |
| 2 | a d c b c c b a b d c c $ |
| 8 | b a b d c c $ |
| 1 | b a d c b c c b a b d c c $ |
| 5 | b c c b a b d c c $ |
| 10 | b d c c $ |
| 13 | c $ |
| 7 | c b a b d c c $ |
| 4 | c b c c b a b d c c $ |
| 12 | c c $ |
| 6 | c c b a b d c c $ |
| 0 | d b a d c b c c b a b d c c $ |
| 3 | d c b c c b a b d c c $ |
| 11 | d c c $ |

bwUniCluster
512 x 16 cores, 64 GB RAM
© KIT (SCC)

# Suffix Sorting with DC3: Example

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $T = [$ | d | b | a | c | b | a | c | b | d | \$ | \$ $]$ |

$= [\, t_i \,]_{i=0,\ldots,n-1}$

| | | | | | | |
|---|---|---|---|---|---|---|
| triples | (bac,1), | (bac,4), | (bd\$,7), | (acb,2) | (acb,5), | (d\$\$,8) |
| sorted | (acb,2) | (acb,5), | (bac,1), | (bac,4), | (bd\$,7), | (d\$\$,8) |
| equal 0/1 | 0 | 0 | 1 | 0 | 1 | 1 |
| prefix sum | 0 | 0 | 1 | 1 | 2 | 3 |

$R = \boxed{1\ 1\ 2}\ \boxed{0\ 0\ 3}\ \$$

$r_1\ r_4\ r_7\ r_2\ r_5\ r_8$

$\mathrm{SA}_R = 3\ 4\ 0\ 1\ 2\ 5\ \$$ $\qquad \mathrm{ISA}_R = \boxed{2\ 3\ 4}\ \boxed{0\ 1\ 5}\ \$$

$S_0 = [(\mathtt{d}, \mathtt{b}, 2, 0, 0), (\mathtt{c}, \mathtt{b}, 3, 1, 3), (\mathtt{c}, \mathtt{b}, 4, 5, 6)]$ $\quad (t_i, t_{i+1}, r_{i+1}, r_{i+2}, i)$

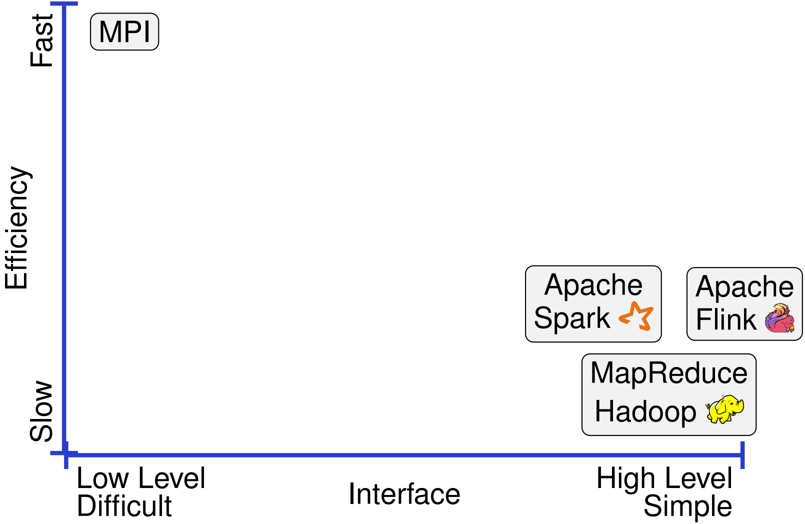
$S_1 = [(2, \mathtt{b}, 0, 1), (3, \mathtt{b}, 1, 4), (4, \mathtt{b}, 5, 7)]$ $\quad (r_{i+1}, t_{i+1}, r_{i+2}, i+1)$

$S_2 = [(0, \mathtt{a}, \mathtt{c}, 3, 2), (1, \mathtt{a}, \mathtt{c}, 4, 5), (5, \mathtt{d}, \$, \text{-}1, 8)]$ $\quad (r_{i+2}, t_{i+2}, t'_{i+3}, r'_{i+4}, i+2)$

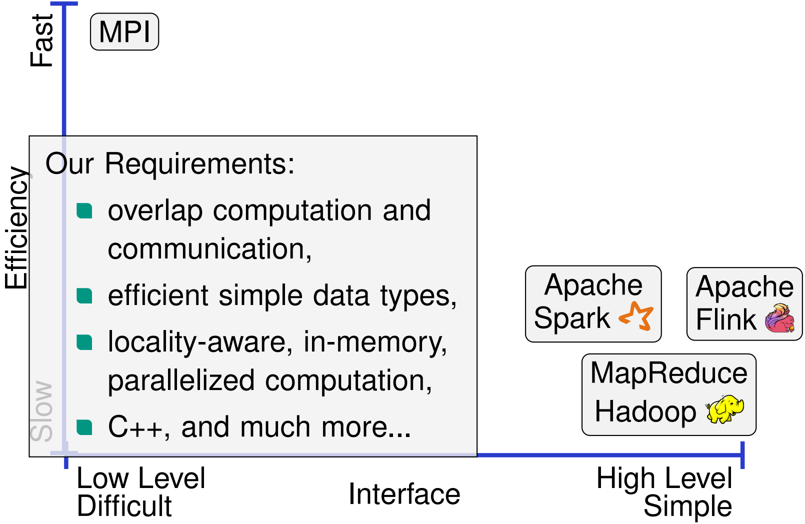
$\mathrm{SA}_T = \mathrm{Merge}(\mathrm{Sort}(S_0), \mathrm{Sort}(S_1), \mathrm{Sort}(S_2))$

$\boxed{\Theta(\mathrm{sort}(n))}$

# Big Data Batch Processing



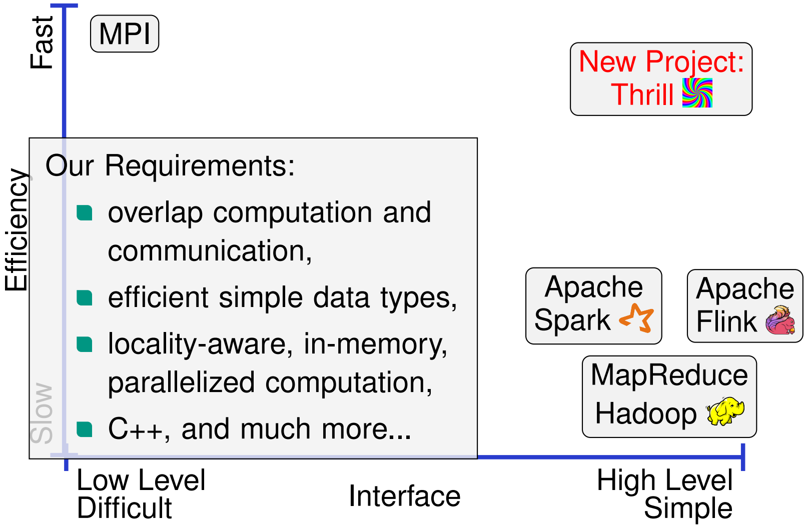
# Big Data Batch Processing



Our Requirements:

- overlap computation and communication,
- efficient simple data types,
- locality-aware, in-memory, parallelized computation,
- C++, and much more...

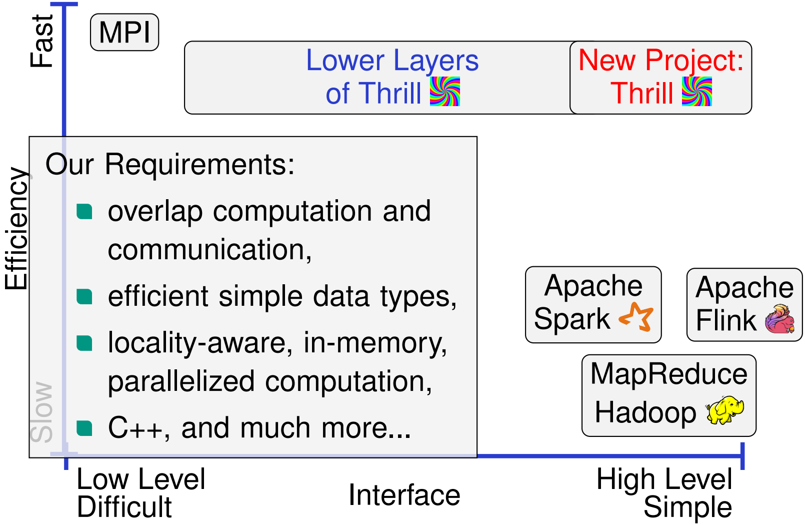
# Big Data Batch Processing



Our Requirements:

- overlap computation and communication,
- efficient simple data types,
- locality-aware, in-memory, parallelized computation,
- C++, and much more...

# Big Data Batch Processing



Fast

MPI

Lower Layers of Thrill

New Project: Thrill

Efficiency

Slow

Apache Spark

Apache Flink

MapReduce Hadoop

Low Level
Difficult

Interface

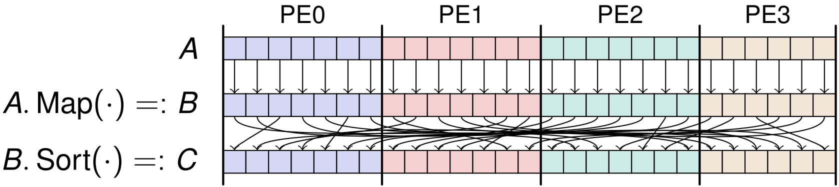
High Level
Simple

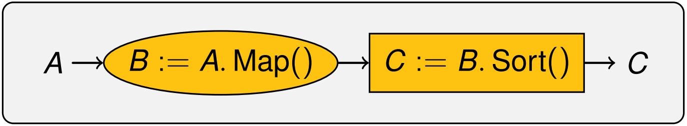
Our Requirements:

- overlap computation and communication,
- efficient simple data types,
- locality-aware, in-memory, parallelized computation,
- C++, and much more...

# Distributed Immutable Array (DIA)

- User Programmer's View:
  - `DIA<T>` = result of an operation (local or distributed).
  - Model: distributed array of items T on the cluster





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

# List of Primitives

- Local Operations (LOp): input is one item, output $\geq$ 0 items.
  Map(), Filter(), FlatMap().
- Distributed Operations (DOp): input is a DIA, output is a DIA.

  Sort() Sort a DIA using comparisons.

  ShuffleReduce() Shuffle with Key Extractor, Hasher, and associative Reducer.

  PrefixSum() Compute (generalized) prefix sum on DIA.

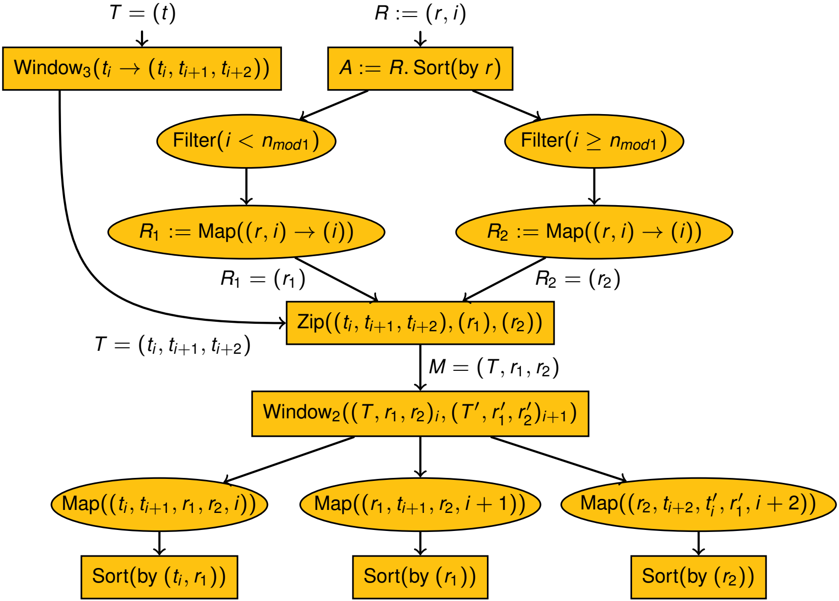
  Window$_k$() Scan all $k$ consecutive DIA items.

  Concat() Concatenate two or more DIAs of equal type.

  Zip() Combine equal sized DIAs item-wise.

  Merge() Merge equal typed DIAs using comparisons.
- Actions: input is a DIA, output: $\geq$ 0 items on master.
  At(), Min(), Max(), Sum(), Sample(), pretty much still open.

# Exert of DC3's Data-Flow Graph



$T = (t)$

$R := (r, i)$

$\text{Window}_3(t_i \rightarrow (t_i, t_{i+1}, t_{i+2}))$

$A := R.\text{Sort}(\text{by } r)$

$\text{Filter}(i < n_{mod1})$

$\text{Filter}(i \geq n_{mod1})$

$R_1 := \text{Map}((r, i) \rightarrow (i))$

$R_2 := \text{Map}((r, i) \rightarrow (i))$

$R_1 = (r_1)$

$R_2 = (r_2)$

$\text{Zip}((t_i, t_{i+1}, t_{i+2}), (r_1), (r_2))$

$T = (t_i, t_{i+1}, t_{i+2})$

$M = (T, r_1, r_2)$

$\text{Window}_2((T, r_1, r_2)_i, (T', r'_1, r'_2)_{i+1})$

$\text{Map}((t_i, t_{i+1}, r_1, r_2, i))$

$\text{Map}((r_1, t_{i+1}, r_2, i + 1))$

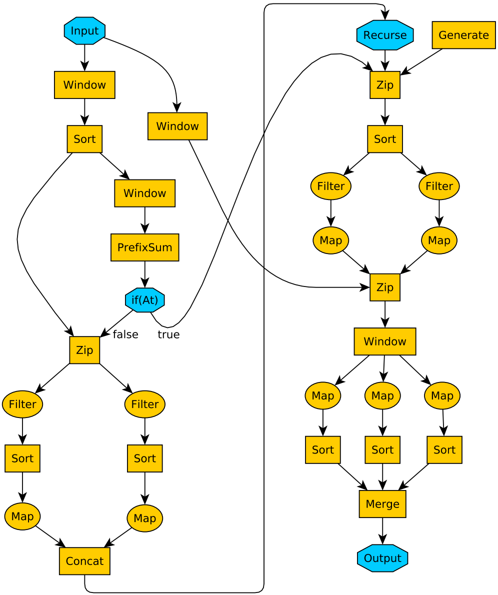
$\text{Map}((r_2, t_{i+2}, t'_i, r'_1, i + 2))$

$\text{Sort}(\text{by } (t_i, r_1))$

$\text{Sort}(\text{by } (r_1))$

$\text{Sort}(\text{by } (r_2))$

# A Suffix Sorting Algorithm: DC3



# Current and Future Work

- Open-Source at http://project-thrill.org and Github.
- Status: prototypes of many DOps work reasonably well.
- Near future: extension to distributed LCP array construction.

- Distributed rank()/select() and wavelet tree construction.
- Distributed query processing.
- Communication efficient distributed operations for Thrill.

Thank you for your attention!

Questions?