Parallel External Memory Suffix Sorting

Juha Kärkkäinen   Dominik Kempa   Simon J. Puglisi

University of Helsinki, Finland

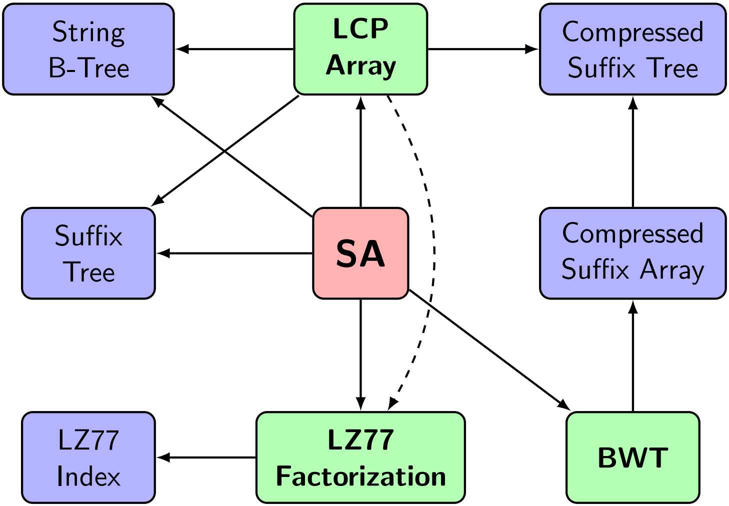
CPM, Ischia, July 2015

Outline

Outline

1. Introduction and Background
2. Internal memory SACA
3. External memory SACA

Central Role of Suffix Array (SA)



Suffix Array Construction

Suffix array construction algorithm (SACA):

Sort all suffixes of a text lexicographically

SA	
6	`$`
5	`a$`
3	`ana$`
1	`anana$`
0	`banana$`
4	`na$`
2	`nana$`

	Internal memory	External memory
Sequential	• Manber, Myers (1993) • Sadakane (1998) • Itoh, Tanaka (1999) • Larsson, Sadakane (1999) • Manzini, Ferragina (2002,2004) • Burkhardt, Kärkkäinen (2003) • Kärkkäinen, Sanders, Burkhardt (2003,2006) • Kim, Sim, Park, Park (2003,2005) • Ko, Aluru (2003,2005) • Hon, Sadakane, Sung (2003,2009) • Schürmann, Stoye (2005,2007) • Maniscalco, Puglisi (2007) • divsufsort: Mori (2008) • Nong, Zhang, Chan (2011) • Nong (2013)	• Gonnet, Baeza-Yates, Snider (1987,1992) • Crauser, Ferragina (1999,2002) • Kärkkäinen, Sanders, Burkhardt (2003,2006) • Dementiev, Kärkkäinen, Mehnert, Sanders (2005,2008) • Ferragina, Gagie, Manzini (2010,2012) • Beller, Zwerger, Gog, Ohlebusch (2013) • eSAIS: Bingmann, Fischer, Osipov (2013) • SAscan: Kärkkäinen, K (2014) • Tischler (2014) • Nong, Chan, Zhang, Guan (2014) • Nong, Chan, Hu, Wu (2015)
Parallel	• Kulla, Sanders (2006,2007) • pDC3: Blelloch, Shun (2011) • Osipov (2012) • Deo, Keely (2013) • pSAscan: This paper	• pSAscan: This paper

Outline

1. Introduction and Background
2. Internal memory SACA
3. External memory SACA

Internal memory

Sequential

- Manber, Myers (1993)
- Sadakane (1998)
- Itoh, Tanaka (1999)
- Larsson, Sadakane (1999)
- Manzini, Ferragina (2002,2004)
- Burkhardt, Kärkkäinen (2003)
- Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Kim, Sim, Park, Park (2003,2005)
- Ko, Aluru (2003,2005)
- Hon, Sadakane, Sung (2003,2009)
- Schürmann, Stoye (2005,2007)
- Maniscalco, Puglisi (2007)
- divsufsort: Mori (2008)
- Nong, Zhang, Chan (2011)
- Nong (2013)

Parallel

- Kulla, Sanders (2006,2007)
- pDC3: Blelloch, Shun (2011)
- Osipov (2012)
- Deo, Keely (2013)
- pSAscan: This paper

External memory

Sequential

- Gonnet, Baeza-Yates, Snider (1987,1992)
- Crauser, Ferragina (1999,2002)
- Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Dementiev, Kärkkäinen, Mehnert, Sanders (2005,2008)
- Ferragina, Gagie, Manzini (2010,2012)
- Beller, Zwerger, Gog, Ohlebusch (2013)
- eSAIS: Bingmann, Fischer, Osipov (2013)
- SAscan: Kärkkäinen, K (2014)
- Tischler (2014)
- Nong, Chan, Zhang, Guan (2014)
- Nong, Chan, Hu, Wu (2015)

Parallel

- pSAscan: This paper

divsufsort

- Currently, the fastest sequential internal memory SACA
- Time complexity: $\mathcal{O}(n \log n)$
- Available 32- and 64-bit versions using $5n$ and $9n$ bytes of RAM

divsufsort

- Currently, the fastest sequential internal memory SACA
- Time complexity: $\mathcal{O}(n \log n)$
- Available 32- and 64-bit versions using $5n$ and $9n$ bytes of RAM

divsufsort

- Currently, the fastest sequential internal memory SACA
- Time complexity: $\mathcal{O}(n \log n)$
- Available 32- and 64-bit versions using $5n$ and $9n$ bytes of RAM

pDC3

- Parallel version of DC3 algorithm of Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Fastest parallel SACA multi-core architecture
- Optimal $\mathcal{O}(n)$ work
- Implementation by Blelloch and Shun
- RAM usage: $21n$ (32-bit), $41n$ (64-bit)
- (divsufsort: $5n$ (32-bit), $9n$ (64-bit))

pDC3

- Parallel version of DC3 algorithm of Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Fastest parallel SACA multi-core architecture
- Optimal $\mathcal{O}(n)$ work
- Implementation by Blelloch and Shun
- RAM usage: $21n$ (32-bit), $41n$ (64-bit)
- (divsufsort: $5n$ (32-bit), $9n$ (64-bit))

pDC3

- Parallel version of DC3 algorithm of Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Fastest parallel SACA multi-core architecture
- Optimal $\mathcal{O}(n)$ work
- Implementation by Blelloch and Shun
- RAM usage: $21n$ (32-bit), $41n$ (64-bit)
- (divsufsort: $5n$ (32-bit), $9n$ (64-bit))

pDC3

- Parallel version of DC3 algorithm of Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Fastest parallel SACA multi-core architecture
- Optimal $\mathcal{O}(n)$ work
- Implementation by Blelloch and Shun
- RAM usage: $21n$ (32-bit), $41n$ (64-bit)
- (divsufsort: $5n$ (32-bit), $9n$ (64-bit))

pDC3

- Parallel version of DC3 algorithm of Kärkkäinen, Sanders, Burkhardt (2003,2006)
- Fastest parallel SACA multi-core architecture
- Optimal $\mathcal{O}(n)$ work
- Implementation by Blelloch and Shun
- RAM usage: $21n$ (32-bit), $41n$ (64-bit)
- (divsufsort: $5n$ (32-bit), $9n$ (64-bit))

SAscan: general overview

1. Divide text into blocks
2. Construct SA for each block
3. Merge each block SA into full SA

SAscan: general overview

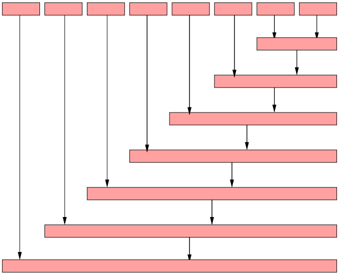
1. Divide text into blocks
2. Construct SA for each block
3. Merge each block SA into full SA

SAscan: general overview

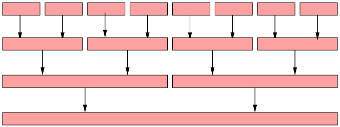
1. Divide text into blocks
2. Construct SA for each block
3. Merge each block SA into full SA

SAscan: general overview

1. Divide text into blocks
2. Construct SA for each block
3. Merge each block SA into full SA
    - multiple merge schedules possible



Unbalanced merge



Fully balanced merge

SAscan: parallel in-RAM version

1. Divide text into blocks
2. Construct SA for each block for all blocks in parallel
3. Merge each block SA into full SA

SAscan: parallel in-RAM version

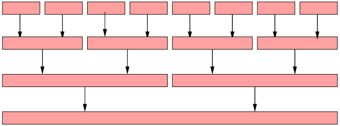
1. Divide text into blocks
2. Construct SA for each block for all blocks in parallel
3. Merge each block SA into full SA

SAscan: parallel in-RAM version

1. Divide text into blocks
2. Construct SA for each block for all blocks in parallel
3. Merge each block SA into full SA

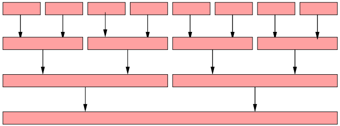
SAscan: parallel in-RAM version

1. Divide text into blocks
2. Construct SA for each block for all blocks in parallel
3. Merge each block SA into full SA
   - balanced schedule
   - how to parallelize?

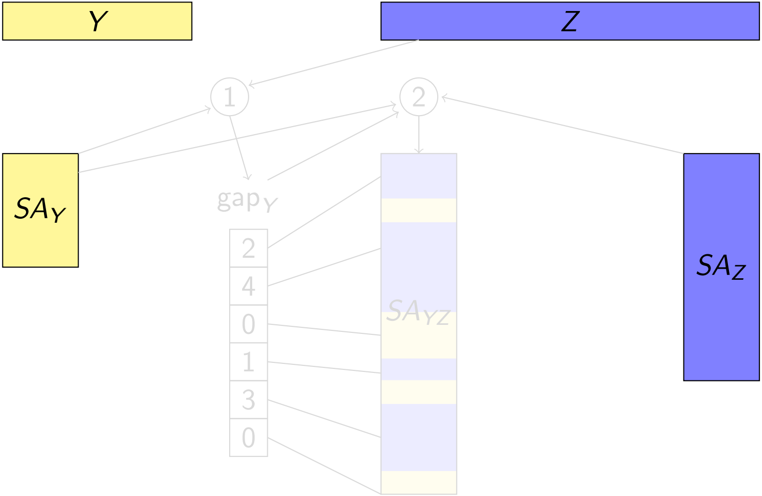


SAscan: parallel in-RAM version

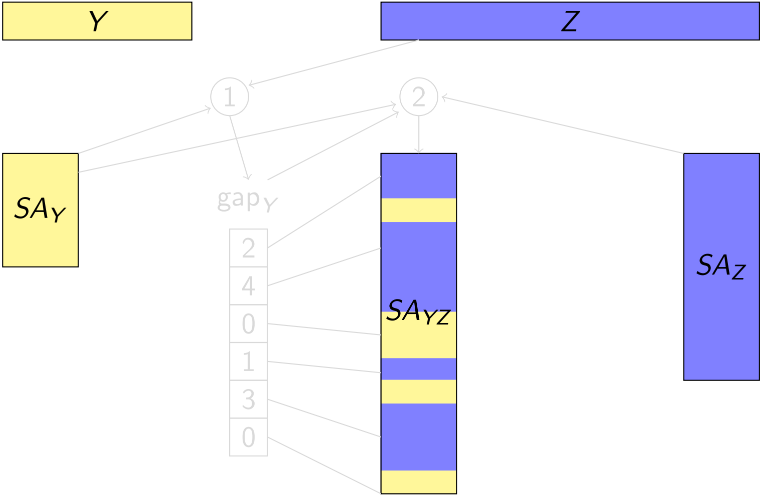
1. Divide text into blocks
2. Construct SA for each block for all blocks in parallel
3. Merge each block SA into full SA
   - balanced schedule
   - how to parallelize?



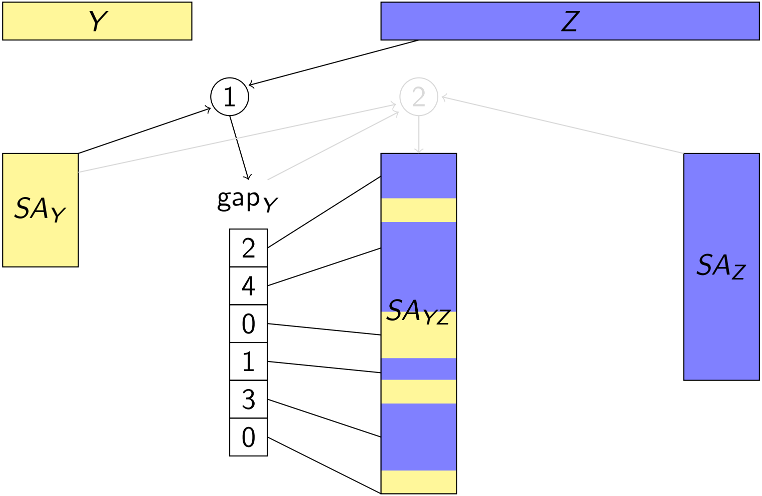
Merging SA of two blocks



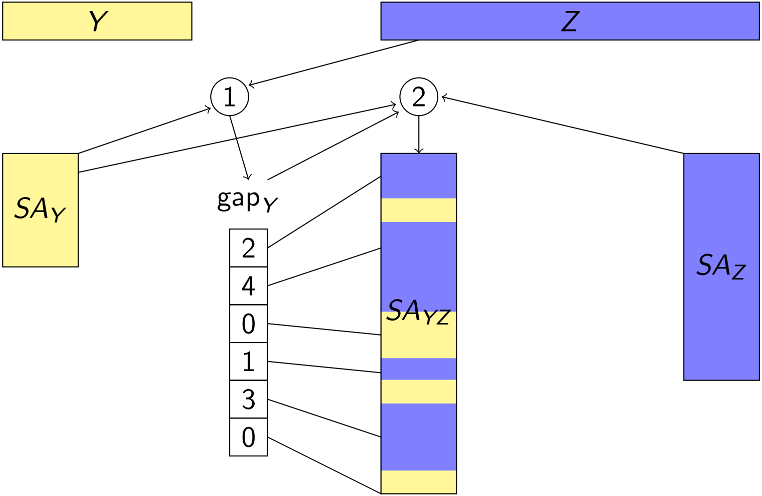
Merging SA of two blocks



Merging SA of two blocks



Merging SA of two blocks



Merging SA of two blocks: parallel in-RAM version

1. $SA_Y + Z \rightarrow \mathrm{gap}_Y$
   - backward search with Y as text and Z as pattern [Ferragina, Gagie & Manzini (2010,2012)]
   - key property: needs rank data structure on BWT of Y
   - parallelization: start backward search from multiple positions
2. $SA_Y + SA_Z + \mathrm{gap}_Y \rightarrow SA_{YZ}$
   - parallelization: possible but complex, details omitted

Merging SA of two blocks: parallel in-RAM version

1. $SA_Y + Z \rightarrow \text{gap}_Y$
   - backward search with Y as text and Z as pattern [Ferragina, Gagie & Manzini (2010,2012)]
   - key property: needs rank data structure on BWT of Y
   - parallelization: start backward search from multiple positions
2. $SA_Y + SA_Z + \text{gap}_Y \rightarrow SA_{YZ}$
   - parallelization: possible but complex, details omitted

Merging SA of two blocks: parallel in-RAM version

1. $SA_Y + Z \to \text{gap}_Y$
   - backward search with Y as text and Z as pattern [Ferragina, Gagie & Manzini (2010,2012)]
   - key property: needs rank data structure on BWT of Y
   - parallelization: start backward search from multiple positions
2. $SA_Y + SA_Z + \text{gap}_Y \to SA_{YZ}$
   - parallelization: possible but complex, details omitted

Merging SA of two blocks: parallel in-RAM version

1. $SA_Y + Z \rightarrow \mathrm{gap}_Y$
   - backward search with Y as text and Z as pattern [Ferragina, Gagie & Manzini (2010,2012)]
   - key property: needs rank data structure on BWT of Y
   - parallelization: start backward search from multiple positions
2. $SA_Y + SA_Z + \mathrm{gap}_Y \rightarrow SA_{YZ}$
   - parallelization: possible but complex, details omitted

Merging SA of two blocks: parallel in-RAM version

1. $SA_Y + Z \rightarrow \text{gap}_Y$
   - backward search with Y as text and Z as pattern [Ferragina, Gagie & Manzini (2010,2012)]
   - key property: needs rank data structure on BWT of Y
   - parallelization: start backward search from multiple positions
2. $SA_Y + SA_Z + \text{gap}_Y \rightarrow SA_{YZ}$
   - parallelization: possible but complex, details omitted

Merging SA of two blocks: parallel in-RAM version

1. $SA_Y + Z \rightarrow \text{gap}_Y$
   - backward search with Y as text and Z as pattern [Ferragina, Gagie & Manzini (2010,2012)]
   - key property: needs rank data structure on BWT of Y
   - parallelization: start backward search from multiple positions
2. $SA_Y + SA_Z + \text{gap}_Y \rightarrow SA_{YZ}$
   - parallelization: possible but complex, details omitted

Internal memory pSAscan - summary

- Work: $\mathcal{O}(n \log p)$
- Space usage
  - $9n$ (32-bit), $10n$ (40-bit)
  - (pDC3: $21n$ (32-bit), $41n$ (64-bit))

Internal memory pSAscan - summary

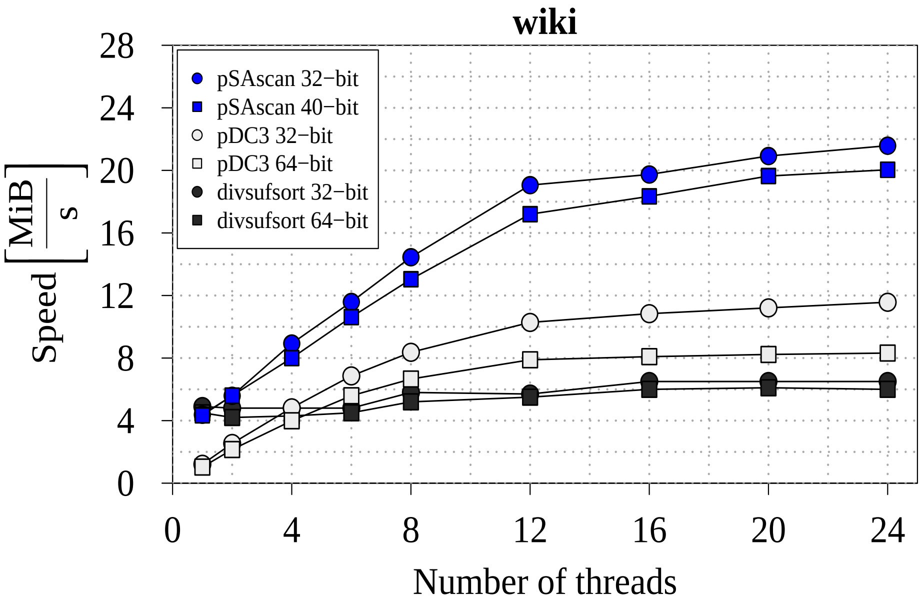
- Work: $\mathcal{O}(n \log p)$
- Space usage
  - $9n$ (32-bit), $10n$ (40-bit)
  - (pDC3: $21n$ (32-bit), $41n$ (64-bit))

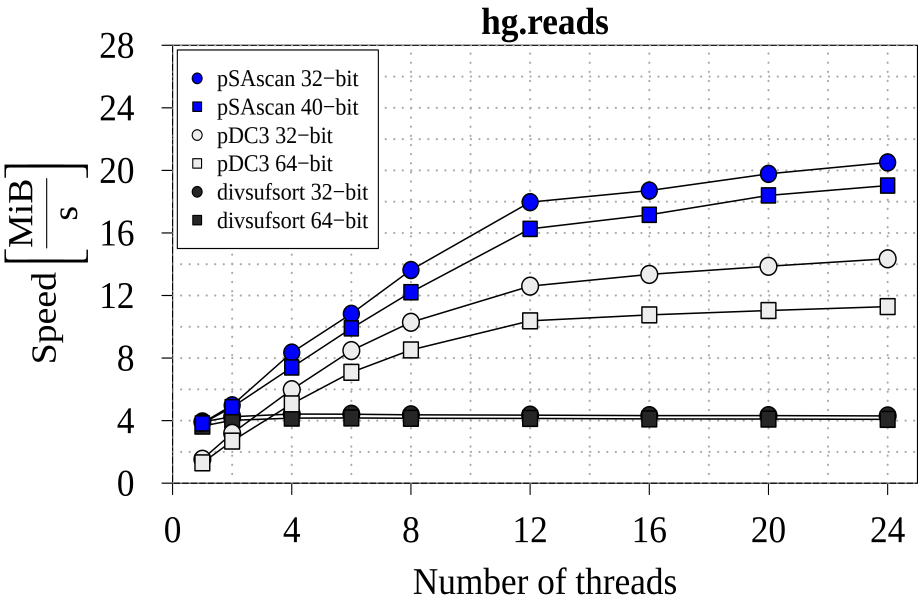
Internal memory pSAscan - summary

- Work: $\mathcal{O}(n \log p)$
- Space usage
  - $9n$ (32-bit), $10n$ (40-bit)
  - (pDC3: $21n$ (32-bit), $41n$ (64-bit))

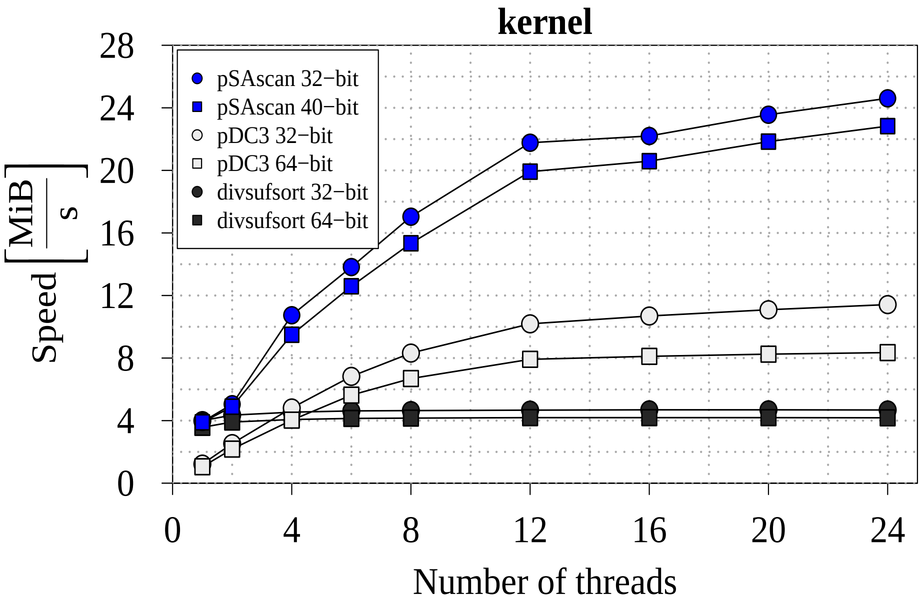
Experiments

(comparison of internal memory parallel suffix array construction algorithms)

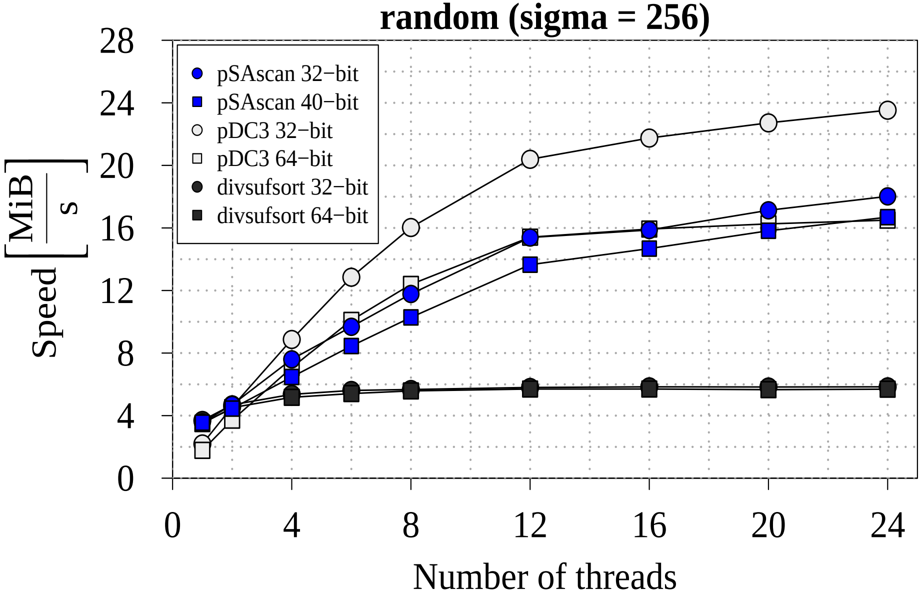

wiki
pSAscan 32−bit
pSAscan 40−bit
pDC3 32−bit
pDC3 64−bit
divsufsort 32−bit
divsufsort 64−bit
Speed [MiB/s]
0
4
8
12
16
20
24
28
Number of threads
0
4
8
12
16
20
24



hg.reads
pSAscan 32−bit
pSAscan 40−bit
pDC3 32−bit
pDC3 64−bit
divsufsort 32−bit
divsufsort 64−bit
Speed [MiB/s]
Number of threads
0
4
8
12
16
20
24
28



kernel
pSAscan 32−bit
pSAscan 40−bit
pDC3 32−bit
pDC3 64−bit
divsufsort 32−bit
divsufsort 64−bit
Speed [MiB/s]
Number of threads
0
4
8
12
16
20
24
28


random (sigma = 256)
pSAscan 32−bit
pSAscan 40−bit
pDC3 32−bit
pDC3 64−bit
divsufsort 32−bit
divsufsort 64−bit
Speed [MiB/s]
Number of threads
0
4
8
12
16
20
24
28

Outline

	Internal memory	External memory
Sequential	• Manber, Myers (1993) • Sadakane (1998) • Itoh, Tanaka (1999) • Larsson, Sadakane (1999) • Manzini, Ferragina (2002,2004) • Burkhardt, Kärkkäinen (2003) • Kärkkäinen, Sanders, Burkhardt (2003,2006) • Kim, Sim, Park, Park (2003,2005) • Ko, Aluru (2003,2005) • Hon, Sadakane, Sung (2003,2009) • Schürmann, Stoye (2005,2007) • Maniscalco, Puglisi (2007) • divsufsort: Mori (2008) • Nong, Zhang, Chan (2011) • Nong (2013)	• Gonnet, Baeza-Yates, Snider (1987,1992) • Crauser, Ferragina (1999,2002) • Kärkkäinen, Sanders, Burkhardt (2003,2006) • Dementiev, Kärkkäinen, Mehnert, Sanders (2005,2008) • Ferragina, Gagie, Manzini (2010,2012) • Beller, Zwerger, Gog, Ohlebusch (2013) • eSAIS: Bingmann, Fischer, Osipov (2013) • SAscan: Kärkkäinen, K (2014) • Tischler (2014) • Nong, Chan, Zhang, Guan (2014) • Nong, Chan, Hu, Wu (2015)
Parallel	• Kulla, Sanders (2006,2007) • pDC3: Blelloch, Shun (2011) • Osipov (2012) • Deo, Keely (2013) • pSAscan: This paper	• pSAscan: This paper

eSAIS

- Optimal $\mathcal{O}(\mathrm{sort}(n))$ I/O complexity
  - $\mathrm{sort}(n)$ = complexity of sorting n integers
  - $\mathcal{O}\left(\frac{n}{B}\log_{M/B}\frac{n}{B}\right)$ I/Os
  - M = size of RAM, B = size of disk block (in units of $\log n$ bits)
- Implementation by Bingmann, Fischer & Osipov (2013)
- $28n$ bytes of disk space
  - n bytes for input
  - $5n$ bytes output

eSAIS

- Optimal $\mathcal{O}(\mathrm{sort}(n))$ I/O complexity
  - $\mathrm{sort}(n)$ = complexity of sorting n integers
  - $\mathcal{O}\left(\frac{n}{B}\log_{M/B}\frac{n}{B}\right)$ I/Os
  - M = size of RAM, B = size of disk block (in units of $\log n$ bits)
- Implementation by Bingmann, Fischer & Osipov (2013)
- $28n$ bytes of disk space
  - n bytes for input
  - $5n$ bytes output

eSAIS

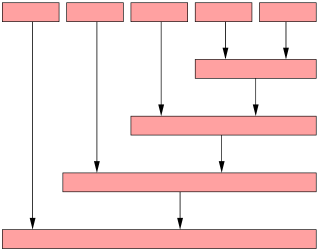
- Optimal $\mathcal{O}(\text{sort}(n))$ I/O complexity
  - $\text{sort}(n)$ = complexity of sorting n integers
  - $\mathcal{O}\left(\frac{n}{B}\log_{M/B}\frac{n}{B}\right)$ I/Os
  - M = size of RAM, B = size of disk block (in units of $\log n$ bits)
- Implementation by Bingmann, Fischer & Osipov (2013)
- $28n$ bytes of disk space
  - n bytes for input
  - $5n$ bytes output

eSAIS

- Optimal $\mathcal{O}(\text{sort}(n))$ I/O complexity
  - $\text{sort}(n)$ = complexity of sorting n integers
  - $\mathcal{O}\left(\frac{n}{B}\log_{M/B}\frac{n}{B}\right)$ I/Os
  - M = size of RAM, B = size of disk block (in units of $\log n$ bits)
- Implementation by Bingmann, Fischer & Osipov (2013)
- $28n$ bytes of disk space
  - n bytes for input
  - $5n$ bytes output

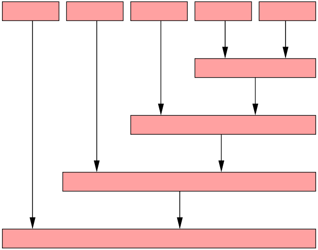
SAscan: EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\implies$ unbalanced merging



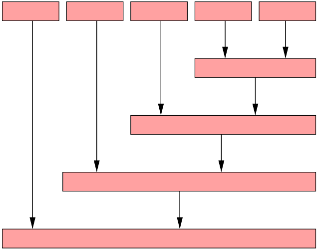
SAscan: EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\Longrightarrow$ unbalanced merging



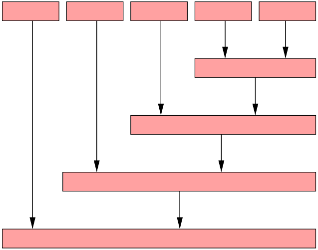
SAscan: EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\implies$ unbalanced merging



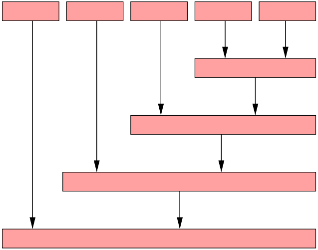
SAscan: EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\implies$ unbalanced merging



SAscan: EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block
     $\implies$ unbalanced merging



SAscan: EM version

- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 5$$

- Lightweight w.r.t. disk space, uses $7.5n$ bytes
    - eSAIS: $28n$

SAscan: EM version

- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 5$$

- Lightweight w.r.t. disk space, uses $7.5n$ bytes
  - eSAIS: $28n$

SAscan: EM version

- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 5$$

- Lightweight w.r.t. disk space, uses $7.5n$ bytes
  - eSAIS: $28n$

SAscan: EM version

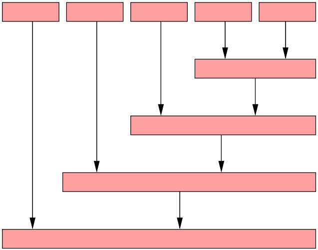
- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 5$$

- Lightweight w.r.t. disk space, uses $7.5n$ bytes
  - eSAIS: $28n$

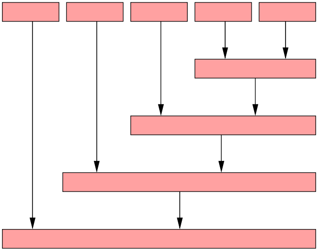
SAscan: parallel EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
   - use internal memory pSAscan
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\Longrightarrow$ unbalanced merging
   - parallization: start backward search from multiple positions



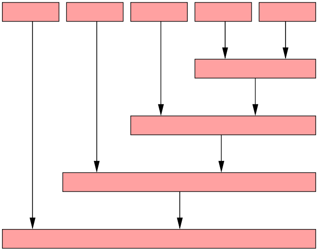
SAscan: parallel EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
   - use internal memory pSAscan
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\Longrightarrow$ unbalanced merging
   - parallization: start backward search from multiple positions



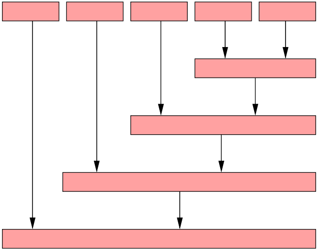
SAscan: parallel EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
   - use internal memory pSAscan
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\Longrightarrow$ unbalanced merging
   - parallization: start backward search from multiple positions



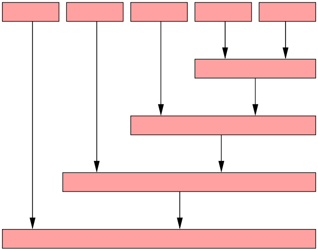
SAscan: parallel EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
   - use internal memory pSAscan
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block $\Longrightarrow$ unbalanced merging
   - parallization: start backward search from multiple positions



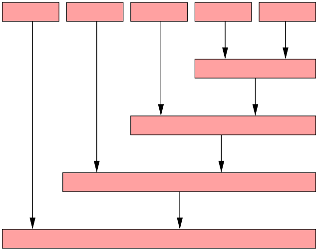
SAscan: parallel EM version

1. Divide text into blocks
   - small enough for RAM processing
2. Construct SA for each block in RAM
   - use internal memory pSAscan
3. Merge each block SA into full SA
   - key property: backward search needs rank data structure on BWT of left hand-size block
     $\Longrightarrow$ unbalanced merging
   - parallization: start backward search from multiple positions



SAscan: parallel EM version

1. Divide text into blocks
    - small enough for RAM processing
2. Construct SA for each block in RAM
    - use internal memory pSAscan
3. Merge each block SA into full SA
    - key property: backward search needs rank data structure on BWT of left hand-size block $\Longrightarrow$ unbalanced merging
    - parallization: start backward search from multiple positions



SAscan: parallel EM version

- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

(on 12-core machine)

- Lightweight w.r.t. disk space, uses 7.5n bytes
  - eSAIS: 28n

SAscan: parallel EM version

- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

(on 12-core machine)

- Lightweight w.r.t. disk space, uses $7.5n$ bytes
  - eSAIS: $28n$

SAscan: parallel EM version

- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

(on 12-core machine)

- Lightweight w.r.t. disk space, uses $7.5n$ bytes
  - eSAIS: $28n$

SAscan: parallel EM version

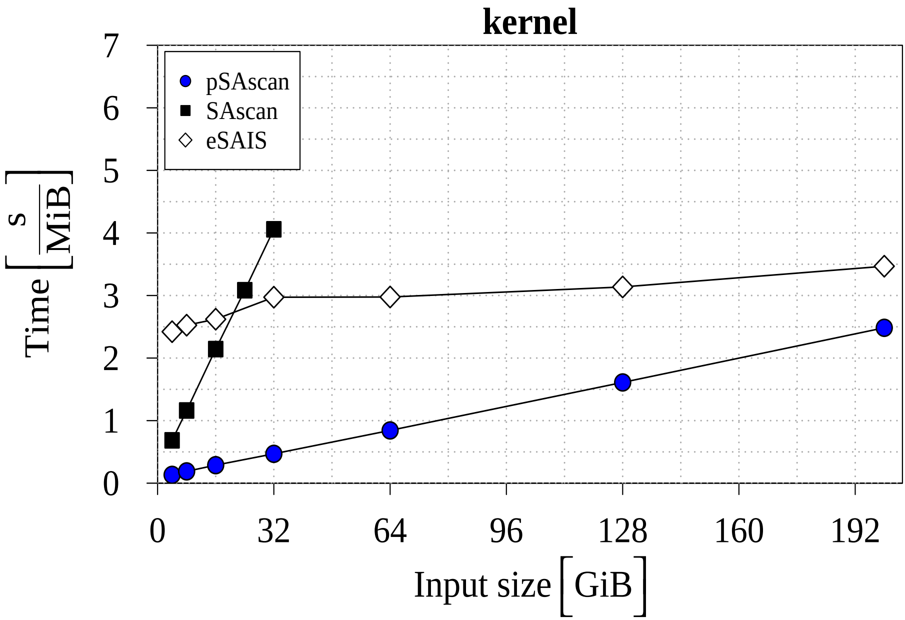
- $\Theta(n/M)$ blocks
- Time and I/O proportional to n^2/M
- Fastest SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

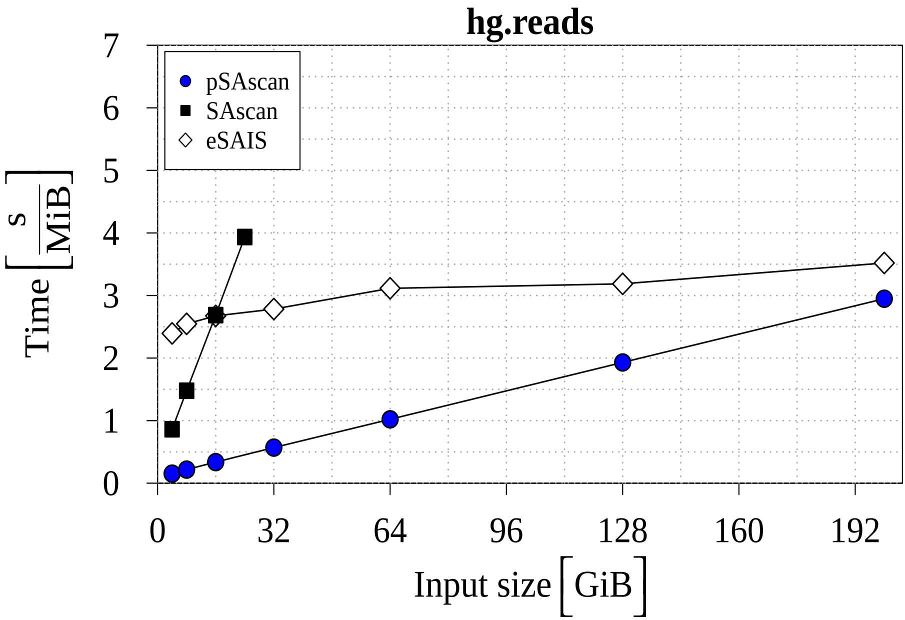
  (on 12-core machine)
- Lightweight w.r.t. disk space, uses $7.5n$ bytes
  - eSAIS: $28n$

Experiments

(comparison of EM suffix array construction algorithms)



kernel
pSAscan
SAscan
eSAIS
Time [s/MiB]
0
1
2
3
4
5
6
7
0
32
64
96
128
160
192
Input size [GiB]



hg.reads
pSAscan
SAscan
eSAIS
Time [s/MiB]
0
1
2
3
4
5
6
7
0
32
64
96
128
160
192
Input size [GiB]

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

- 200GiB prefix of `hg.reads`
  - Runtime

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	8.3 days	4.1 days
pSAscan	7.0 days	0.5 days

  - Peak disk space usage

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	4.6 TiB	4.6 TiB
pSAscan	1.4 TiB	1.4 TiB

  - I/O volume

	3.5 GiB of RAM	120 GiB of RAM
eSAIS	52.0 TiB	36.1 TiB
pSAscan	43.8 TiB	4.9 TiB

- Full instance (1TiB) of `hg.reads`, using 120GiB of RAM

	Runtime	Peak disk usage	I/O volume
pSAscan	8.1 days	7.3 TiB	48.3 TiB

Concluding Remarks

- New internal-memory parallel SACA
  - $2\times$ faster and $\sim 2.5\times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Concluding Remarks

- New internal-memory parallel SACA
  - $2\times$ faster and $\sim 2.5\times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Concluding Remarks

- New internal-memory parallel SACA
  - $2 \times$ faster and $\sim 2.5 \times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Concluding Remarks

- New internal-memory parallel SACA
  - $2\times$ faster and $\sim 2.5\times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Concluding Remarks

- New internal-memory parallel SACA
  - $2\times$ faster and $\sim 2.5\times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Concluding Remarks

- New internal-memory parallel SACA
  - $2\times$ faster and $\sim 2.5\times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Concluding Remarks

- New internal-memory parallel SACA
  - $2 \times$ faster and $\sim 2.5 \times$ less RAM than pDC3
- First external-memory parallel SACA
  - Fastest EM SACA when

$$\frac{\text{text size}}{\text{RAM size}} < 80$$

  - uses ~ 4 times less disk space than eSAIS

Possible further improvements

- smaller rank data structure
- distributed / GPU implementation?

Thank you!

Code:
www.cs.helsinki.fi/group/pads/