x-search: A C++ library for fast external string search

Bachelor Thesis – Mai 27, 2023

AUTHOR

Leon Freist

SUPERVISOR

Johannes Kalmbach

Faculty of Engineering, University of Freiburg
Chair of Algorithms and Datastructures
Prof. Dr. Hannah Bast

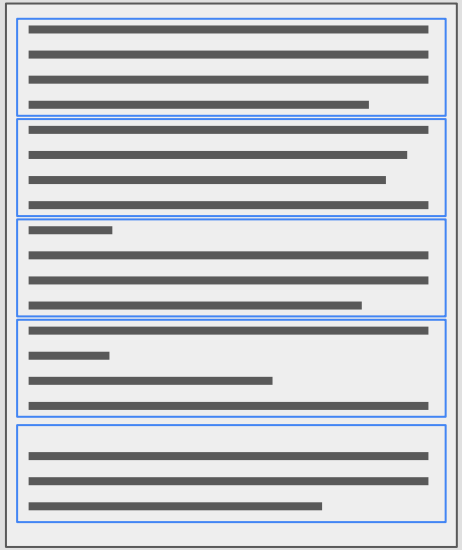
Why External String Search?

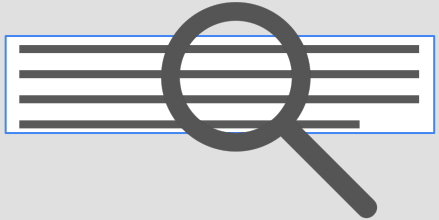
Information Retrieval

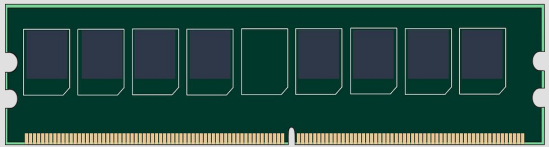
Natural Language Processing

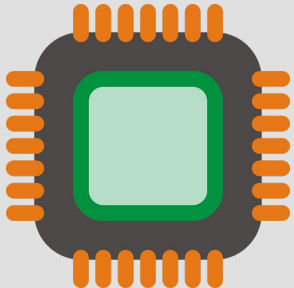
Bioinformatics

Problem









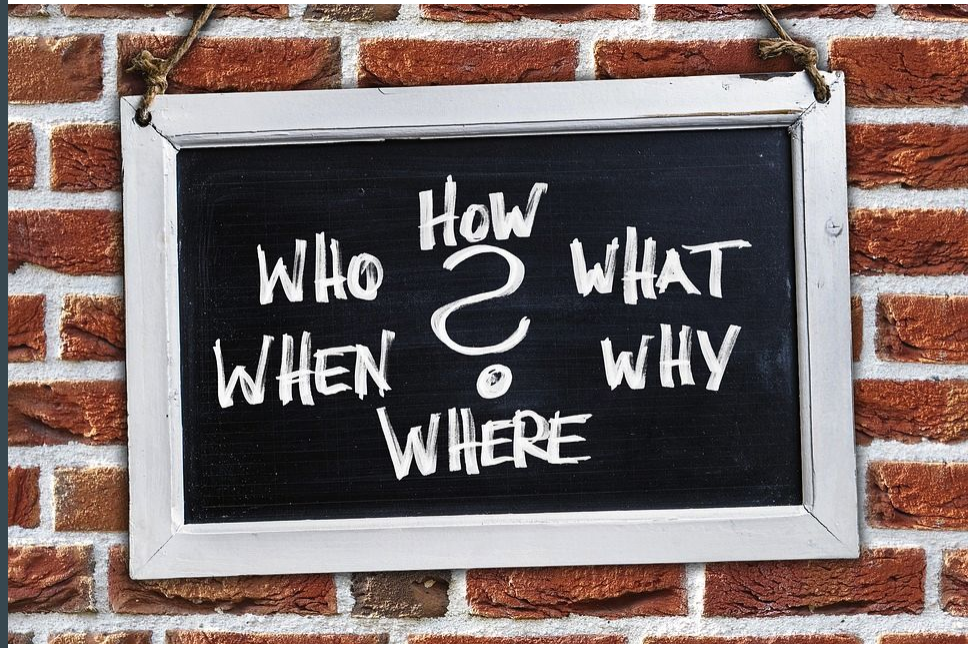
GNU grep Comes Short in Different Terms

- No library
- No multi-thread support for searching a single file
- No direct preprocessing supported
- License issues

Aims for this Thesis

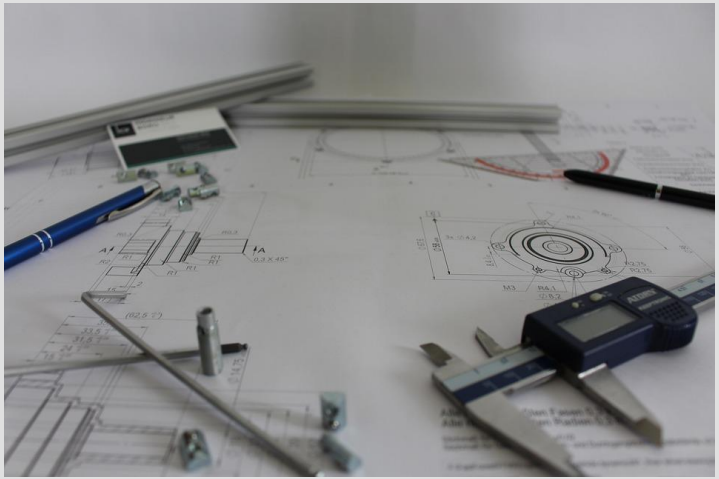
- Implement a C++ library
- Support multi-threading
- Support preprocessing (compression)
- Be faster than GNU grep

Questions?



Solution

Simple Usage



High Level Viewport

x-search: Basic Usage

```
0  // basic_grep.cpp
1  #include “xsearch/xsearch.h”
2  #include <iostream>
3
4  int main(int argc, char** argv) {
5    auto searcher = xs::extern_search<xs::lines>(argv[1], argv[2], false, 1);
6    for (auto const& line : *searcher->getResult()) {
7      std::cout << line << '\n';
8    }
9  }
```

```
$ basic_grep pattern /path/to/file
```

A Pipeline as Base-Construct

- Reading from file

- Compression
- Decompression

- Substring search

Result
`xs::Result`

Managing partial results provided by the ReturnProcessor

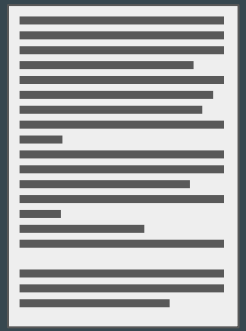
Preprocessing

Reading Data in Chunks
`xs::DataProvider`

Compression
`xs::InplaceProcessor`

Write (Meta) Data
`xs::ReturnProcessor`
`xs::Result`

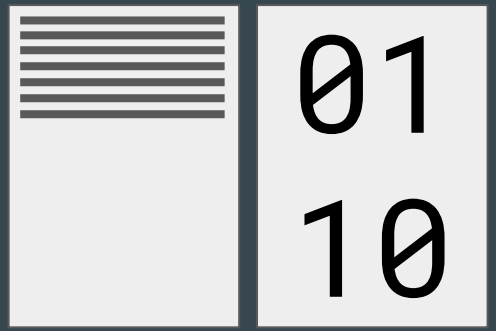
Each chunk ends with a new line character



Compress chunks using

- LZ4
- LZ4 HC
- ZSTD

- Write metadata to a metafile
- Write compressed data to a file



Searching a Preprocessed File

Read Data in Chunks
`xs::DataProvider`

Decompression
`xs::InplaceProcessor`

Search
`xs::ReturnProcessor`

Collect Search Results
`xs::Result`

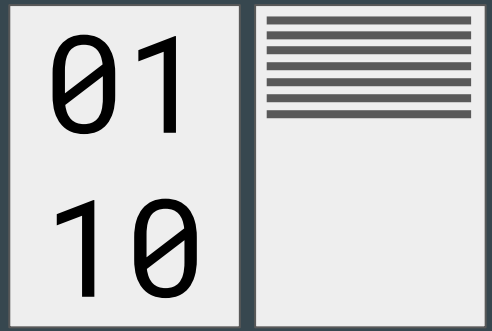
Read Metadata and chunks from input file

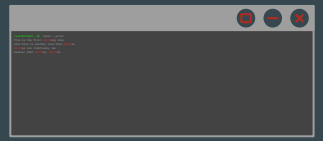
Decompress read chunks

Search according to the specified settings

- Literal
- Regex
- Ignore case

Provide results according to the specified result type





```
std::vector<T>
```

Input — Processing — Output

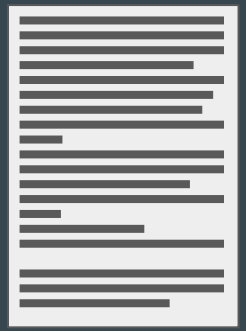
Searching a plain Text File

Read Data in Chunks
`xs::DataProvider`

Search
`xs::ReturnProcessor`

Collect Search Results
`xs::Result`

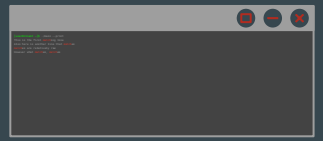
Each chunk ends with a new line character



Search according to the specified settings

- Literal
- Regex
- Ignore case

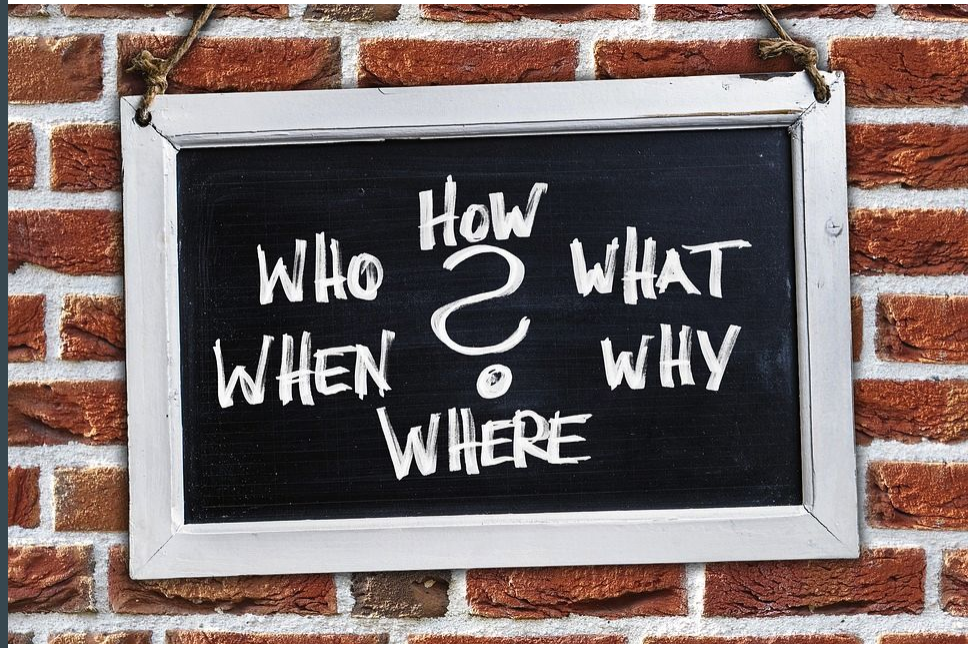
Provide results according to the specified result type



```
std::vector<T>
```

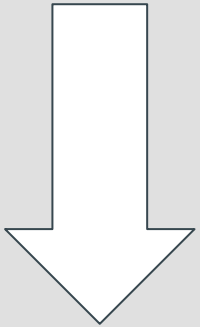
Input — Processing — Output

Questions?



Evaluation

x-search



xs grep

Dataset

- OpenSubtitles2016
  - 9.3 GiB
  - 337’845’355 lines

Pattern

- Literal: “`Sherlock`”
  - 13’645

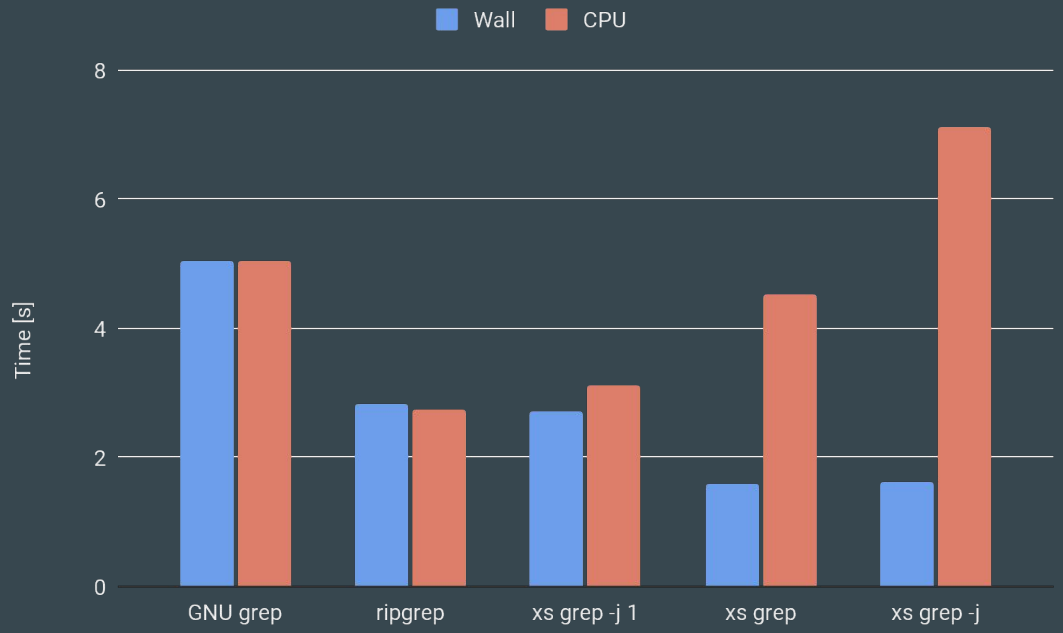
Benchmark Metrics

- Wall Time
- CPU Time

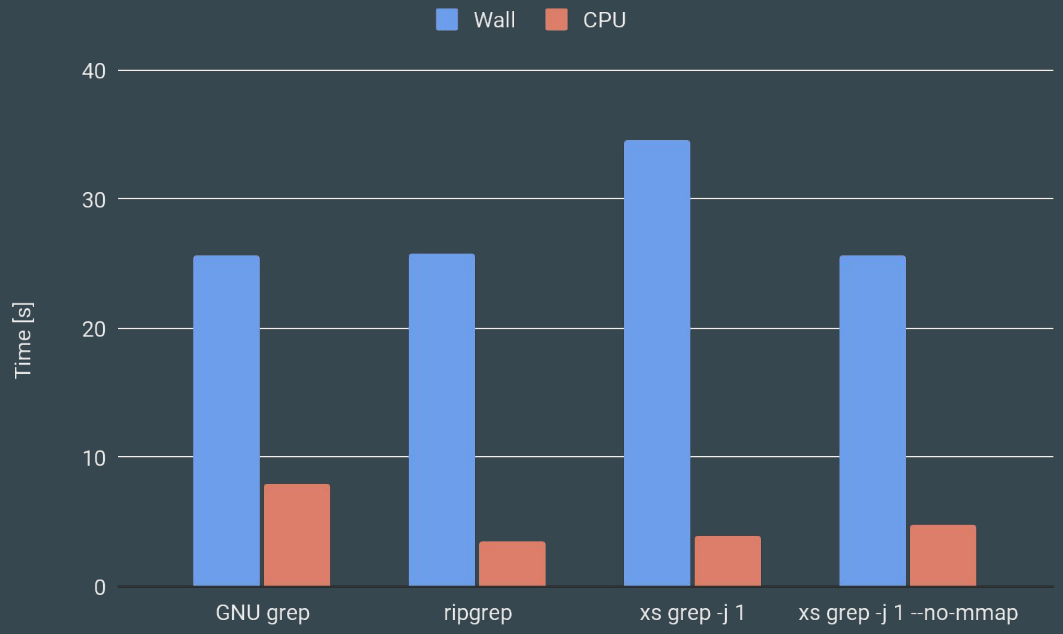
Hardware

- RAID0 NVMe SSD
- HDD

Reading from SSD

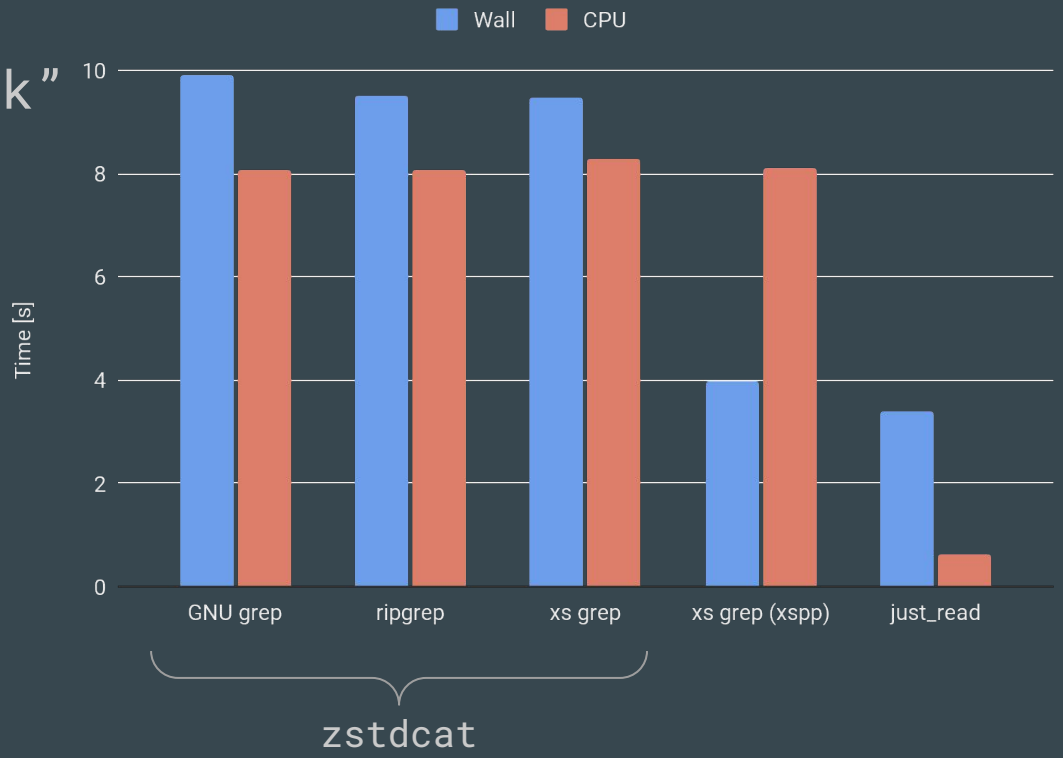


Reading from HDD

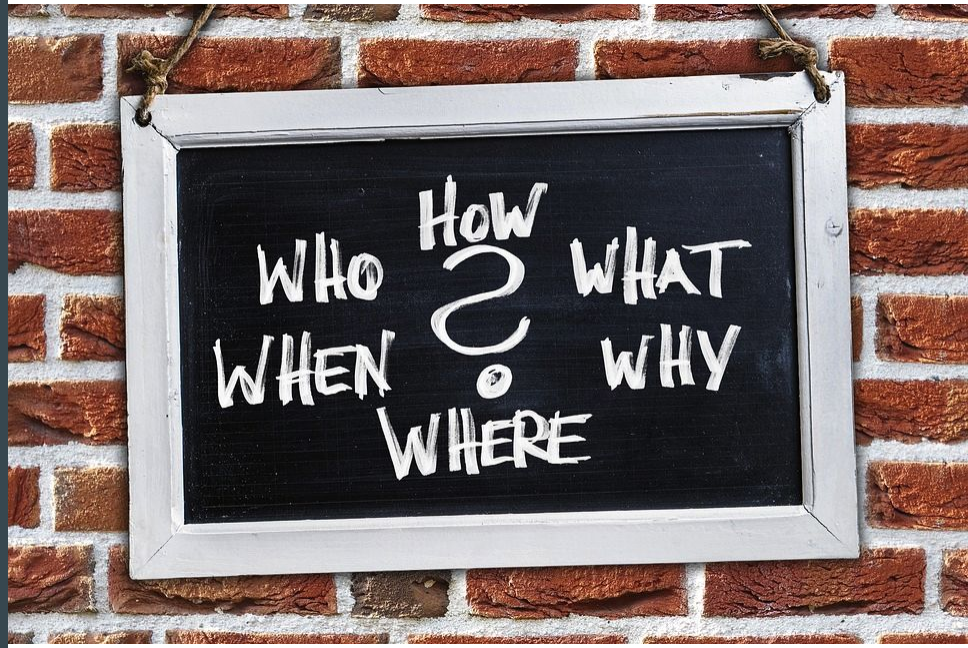


Reading from HDD: Compressed Input

Pattern	“Sherlock”
data.txt	9.3G
data.zst	1.5G
data.xs.zst	1.5G
data.xs.meta	9.6M



Questions?



What else can x-search do?

```
auto searcher = xs::extern_search<T>(...);
```

1. Supporting different search results (T)
    - xs::lines
    - xs::count
    - xs::count_lines
    - xs::match_byte_offsets
    - xs::line_byte_offsets
    - xs::line_indices

What else can x-search do?

```
auto searcher = xs::extern_search<T>(...);
```

1. Supporting different search results (T)
2. Controlling the searcher
   - `searcher.join()`: Wait for the search to finish
   - `searcher.stop()`: Force stop the searcher
   - `searcher.getResult()->begin()`: Access results live

What else can x-search do?

```
auto searcher = xs::extern_search<T>(...);
```

1. Supporting different search results (T)
2. Controlling the searcher
3. Accessing search results

What else can x-search do?

```
auto searcher = xs::extern_search<T>(...);
```

1. Supporting different search results (T)
2. Controlling the searcher
3. Accessing Search Results
4. Providing an extended API
    - Implement custom DataProvider-, InplaceProcessor- and ReturnProcessor-Tasks
    - Implement custom Result-Types

xs grep: A GNU grep-like command line search tool

- Single pattern and single file input
- Command-Line-Options:
  - `-F [--fixed-string]`
  - `-i [--ignore-case]`
  - `-o [--only-matching]`
  - `-n [--line-number]`
  - `-b [--byte-offset]`
  - `-c [--count]`

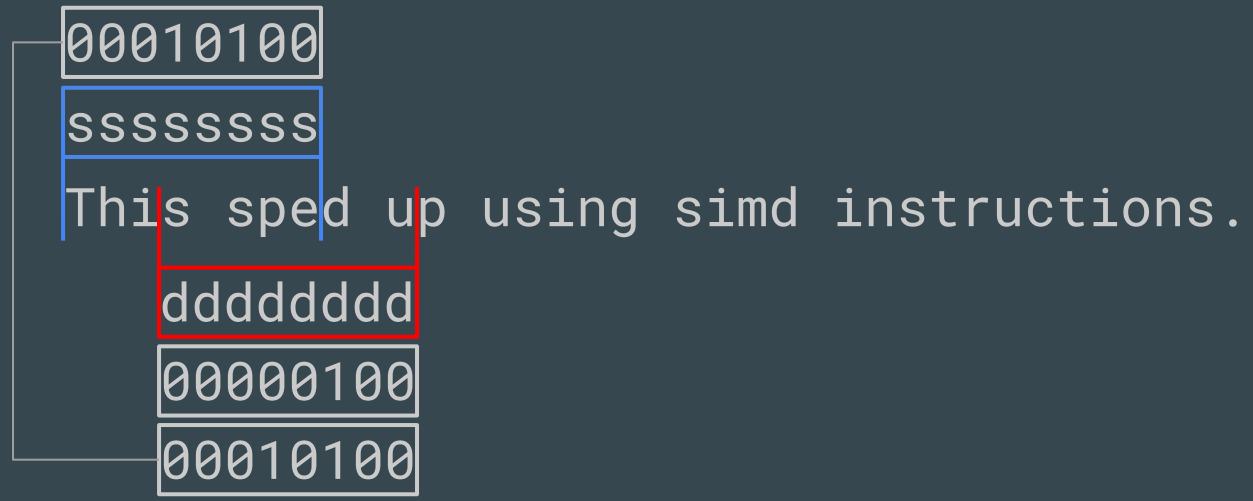
  } GNU grep equivalents

  - `-j [--threads]`
  - `--max-readers`
  - `-m [--metafile]`
  - `--no-mmap`

  } Additional arguments

SIMD Substring Search Algorithm

Pattern: “simd”

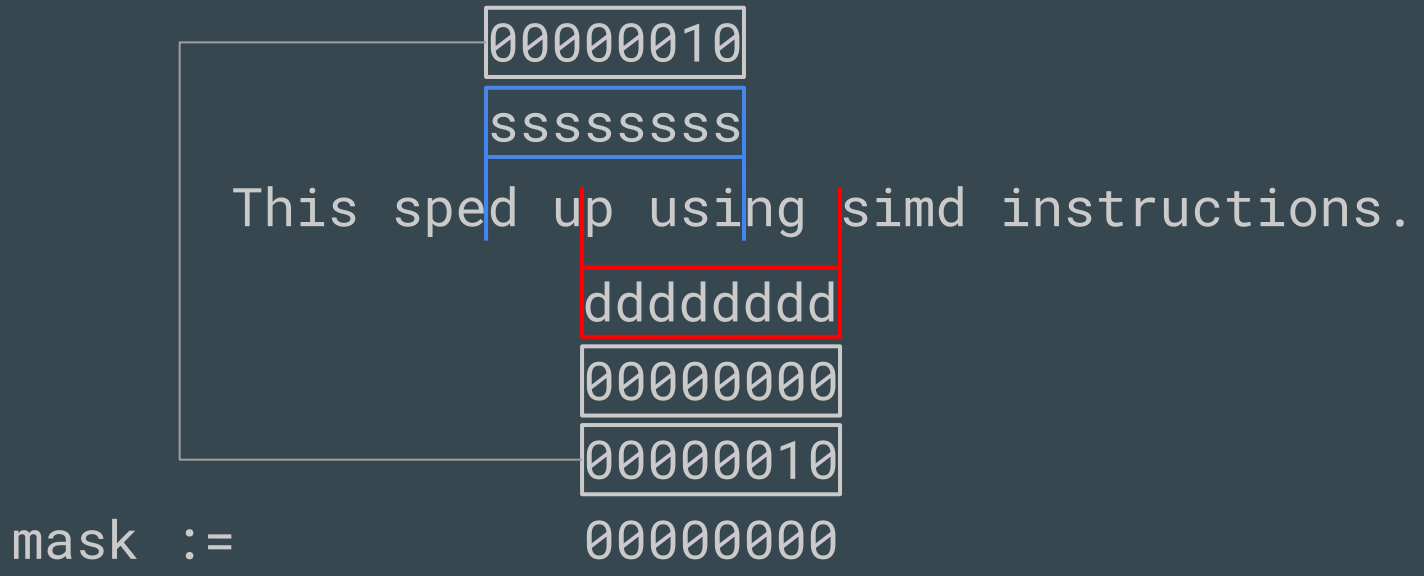


mask := 00000100

```
memcmp(data + __builtin_ctz(mask) + 1, pattern + 1, 2)
```

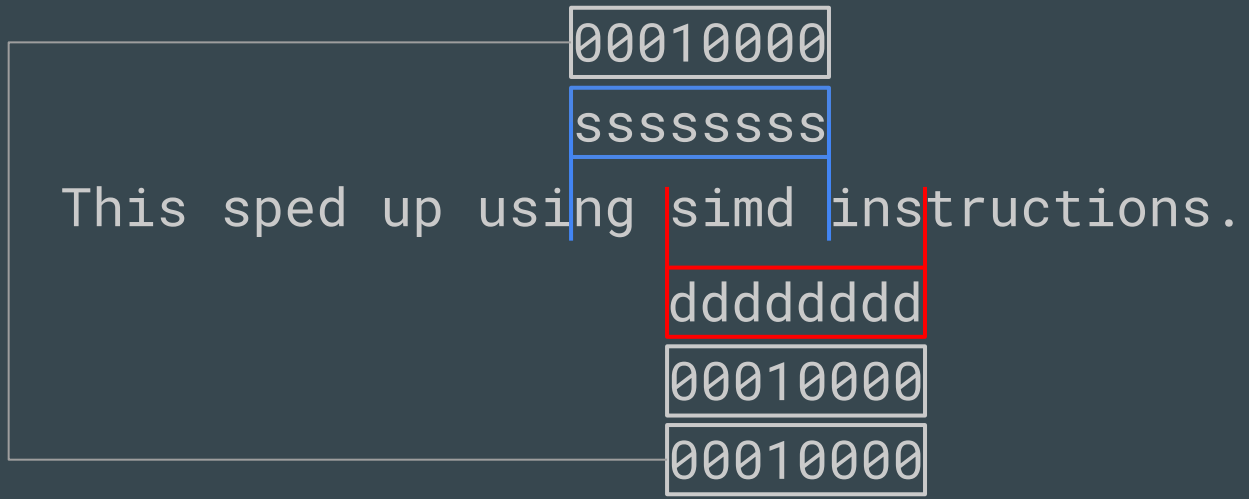
SIMD Substring Search Algorithm

Pattern: “simd”



SIMD Substring Search Algorithm

Pattern: “simd”



mask :=

```
memcmp(data + __builtin_ctz(mask) + 1, pattern + 1, 2)
```

Data Characteristics

Size	9.3 GiB
Number of Lines	337’845’355

	Bytes per Line
Mean	29.5
Min	1
25% Quartile	14
50% Quartile	24
75% Quartile	38
Max	26’586

Pattern Characteristics

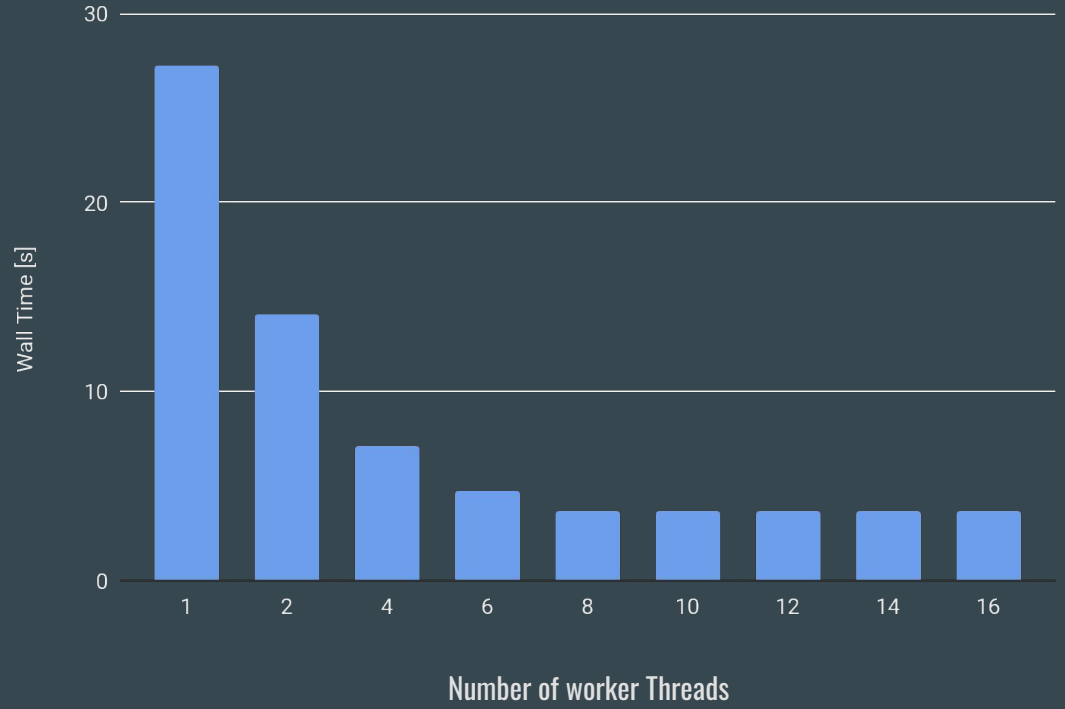
Pattern	Matching Lines	% of Lines
“Sherlock”	13’645	0.004
“ [sS][A-Za-z]*[Kk] ”	1’079’731	0.320

Thread Management

Pattern:

“ `[Ss][A-Za-z]*[Kk]` ”

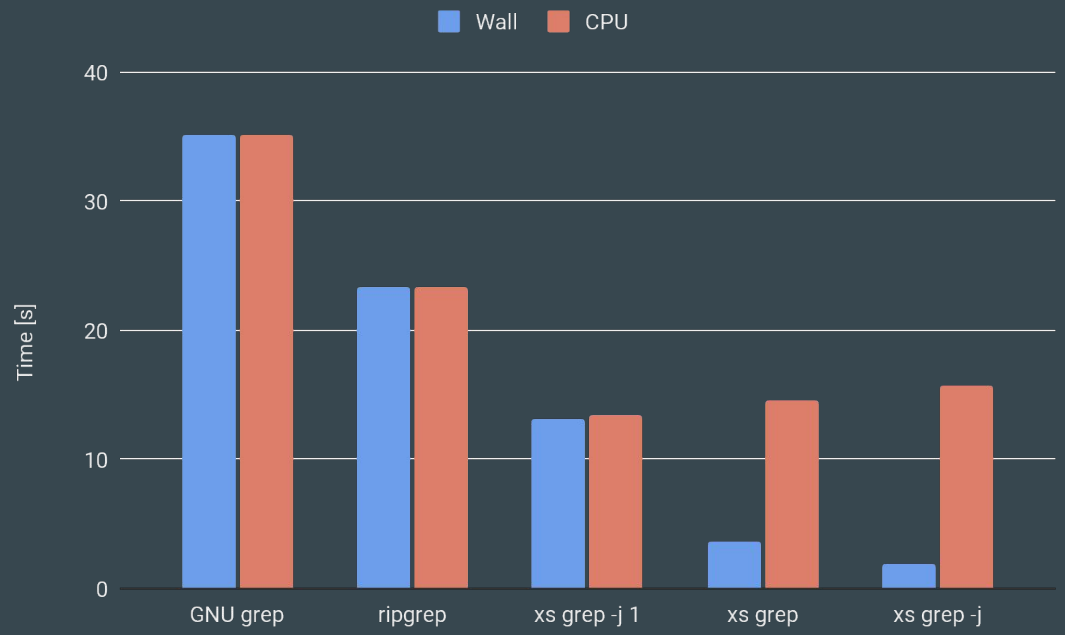
Reading from RAM cache



Reading from SSD

Pattern:

“ [Ss][A-Za-z]*[Kk] ”



			'Sherlock'			' [sS][A-Za-z]*[kK] '		
			Wall (s)	CPU (s)	Wall (%)	Wall (s)	CPU (s)	Wall (%)
Cached Read	Threads	1	0.938	0.938	**100.0**	27.270	27.276	**100.0**
		2	0.548	1.046	**171.2**	14.082	27.944	**193.7**
		4	0.408	1.438	**229.9**	7.134	28.492	**382.3**
		8	0.426	2.786	**220.2**	3.654	29.054	**746.3**
		16	0.434	2.790	**216.1**	3.696	29.376	**737.8**
	mmap	yes	0.424	1.450	**100.0**			
		no	1.680	2.786	**25.2**			
SSD Read	Threads	1	2.734	3.122	**100.0**	28.876	29.168	**100.0**
		2	1.888	3.542	**144.8**	15.028	30.254	**192.1**
		4	1.612	4.624	**169.6**	7.662	30.816	**376.9**
		8	1.624	7.376	**168.3**	4.002	31.902	**721.5**
		16	1.632	7.164	**167.5**	4.020	32.048	**718.3**
	mmap	yes	1.604	4.498	**100.0**			
		no	3.512	4.574	**45.7**			
HDD Read	Threads	1	34.526	4.188	**100.0**	41.642	28.792	**100.0**
		2	37.534	4.198	**92.0**	45.736	43.178	**91.0**
		4	37.842	3.944	**91.2**	37.044	53.766	**112.4**
		8	37.868	4.196	**91.2**	36.216	57.380	**115.0**
		16	37.834	4.194	**91.3**	36.054	57.736	**115.5**
	mmap	yes	37.472	4.134	**100.0**			
		no	25.674	4.940	**146.0**			

		Cached Read								
		`'Sherlock'`			`'She[r ]lock'`			`' [sS][A-Za-z]*[kK] '`		
		Wall	CPU	Wall %	Wall	CPU	Wall %	Wall	CPU	Wall %
Line	ripgrep	0.836	0.830	**100.0**	1.262	1.256	**100.0**	21.166	21.162	**100.0**
	GNU grep	3.206	3.204	**26.1**	3.930	3.926	**32.1**	33.032	33.026	**64.1**
	xs grep -j 1	0.956	0.964	**87.4**	1.108	1.120	**113.9**	11.314	11.324	**187.1**
	xs grep	0.424	1.458	**197.2**	0.436	1.504	**289.4**	3.022	12.068	**700.4**
	xs grep -j	0.450	2.808	**185.8**	0.442	2.816	**285.5**	1.554	12.280	**1362.0**
Line Number	ripgrep	1.102	1.092	**100.0**	1.526	1.522	**100.0**	21.292	21.286	**100.0**
	GNU grep	5.832	5.828	**18.9**	6.542	6.538	**23.3**	35.924	35.916	**59.3**
	xs grep -j 1	3.886	3.900	**28.4**	4.080	4.092	**37.4**	14.944	14.956	**142.5**
	xs grep	1.286	4.726	**85.7**	1.296	4.758	**117.7**	3.882	15.484	**548.5**
	xs grep -j	0.894	6.144	**123.3**	0.894	6.158	**170.7**	2.010	15.822	**1059.3**
Ignore Case	ripgrep	1.472	1.466	**100.0**	2.468	2.462	**100.0**	21.150	21.146	**100.0**
	GNU grep	13.042	13.040	**11.3**	13.236	13.234	**18.6**	35.374	35.368	**59.8**
	xs grep -j 1	2.216	2.224	**66.4**	11.796	11.806	**20.9**	27.384	27.392	**77.2**
	xs grep	1.714	6.436	**85.9**	3.078	12.184	**80.2**	7.106	28.358	**297.6**
	xs grep -j	1.770	13.146	**83.2**	1.598	12.432	**154.4**	3.632	28.794	**582.3**

		SSD Read								
		`'Sherlock'`			`'She[r ]lock'`			`' [sS][A-Za-z]*[kK] '`		
		Wall	CPU	Wall %	Wall	CPU	Wall %	Wall	CPU	Wall %
Line	ripgrep	2.840	2.760	**100.0**	3.272	3.202	**100.0**	23.384	23.324	**100.0**
	GNU grep	5.054	5.038	**56.2**	5.798	5.784	**56.4**	35.182	35.164	**66.5**
	xs grep -j 1	2.730	3.118	**104.0**	2.894	3.284	**113.1**	13.224	13.422	**176.8**
	xs grep	1.608	4.538	**176.6**	1.614	4.550	**202.7**	3.598	14.630	**649.9**
	xs grep -j	1.626	7.112	**174.7**	1.630	7.406	**200.7**	1.978	15.736	**1182.2**
Line Number	ripgrep	3.174	3.106	**100.0**	3.576	3.504	**100.0**	23.628	23.572	**100.0**
	GNU grep	7.708	7.694	**41.2**	8.424	8.410	**42.5**	37.786	37.770	**62.5**
	xs grep -j 1	5.788	6.174	**54.8**	5.960	6.330	**60.0**	16.940	17.216	**139.5**
	xs grep	2.102	7.946	**151.0**	2.164	8.072	**165.2**	4.556	18.336	**518.6**
	xs grep -j	1.790	12.190	**177.3**	1.766	11.834	**202.5**	2.494	19.760	**947.4**
Ignore Case	ripgrep	3.496	3.428	**100.0**	4.484	4.422	**100.0**	23.402	23.340	**100.0**
	GNU grep	14.928	14.912	**23.4**	15.002	14.988	**29.9**	37.438	37.422	**62.5**
	xs grep -j 1	4.288	4.664	**81.5**	13.610	13.904	**32.9**	28.866	29.160	**81.1**
	xs grep	2.704	10.094	**129.3**	3.618	14.692	**123.9**	7.654	30.806	**305.7**
	xs grep -j	2.720	21.18	**128.5**	1.990	15.790	**225.3**	3.994	31.814	**585.9**

	File Size (GB)		HDD Read 'Sherlock' Wall	CPU	Wall %
Plain	9.3	ripgrep	25.810	3.462	**100.0**
		GNU grep	25.632	7.998	**100.7**
		xs grep	25.662	4.826	**100.6**
ZStandard	1.5	zstcat → ripgrep	9.518	8.068	**269.4**
		zstcat → GNU grep	9.922	8.058	**260.1**
		zstcat → xs grep	9.484	8.294	**272.1**
		xs grep (xspp)	3.976	8.116	**649.1**
LZ4	3.7	lz4cat → ripgrep	8.184	6.332	**315.4**
		lz4cat → GNU grep	13.096	7.896	**197.1**
		lz4cat → xs grep	8.228	7.586	**313.7**
		xs grep (xspp)	8.862	6.438	**291.2**
LZ4 HC	2.1	lz4cat → ripgrep	6.446	5.750	**400.4**
		lz4cat → GNU grep	11.614	6.680	**222.2**
		lz4cat → xs grep	7.102	7.040	**363.4**
		xs grep (xspp)	5.358	5.744	**481.7**

Benchmark Metrics

- Wall Time
- CPU Time

Scenario 0:

One Thread reads data from disk and writes the read data to standard out. The program runs 1 second.

- Wall: 1 seconds
- CPU: 0.1 seconds

Scenario 1:

Two threads simultaneously search two different patterns within a string. The program runs 1 second.

- Wall: 1 second
- CPU: 2 seconds

Boyer-Moore: Bad Character

```
Data:    this is a boyer-moore test!
Pattern: test|   |   |   |   ||       's' = 1
          test   |   |   |   ||       ' ' = 4
              test   |   |   ||       'a' = 4
                  test   |   ||       'y' = 4
                      test   ||       'm' = 4
                          test|       'e' = 2
                            test      ' ' = 4
                               test
```

t	0
e	2
s	1
other	pattern size: 4

Boyer-Moore: Good Suffix

```
Data:    reinesupersauersupesupersupe
Pattern: superrsupe   |     |
              supersupe    |
                       supersupe
                            supersupe
```

e	1
pe	10
upe	11
supe	12
rsupe	9
ersupe	10
persupe	11
upersupe	12

x-search: Basic Usage

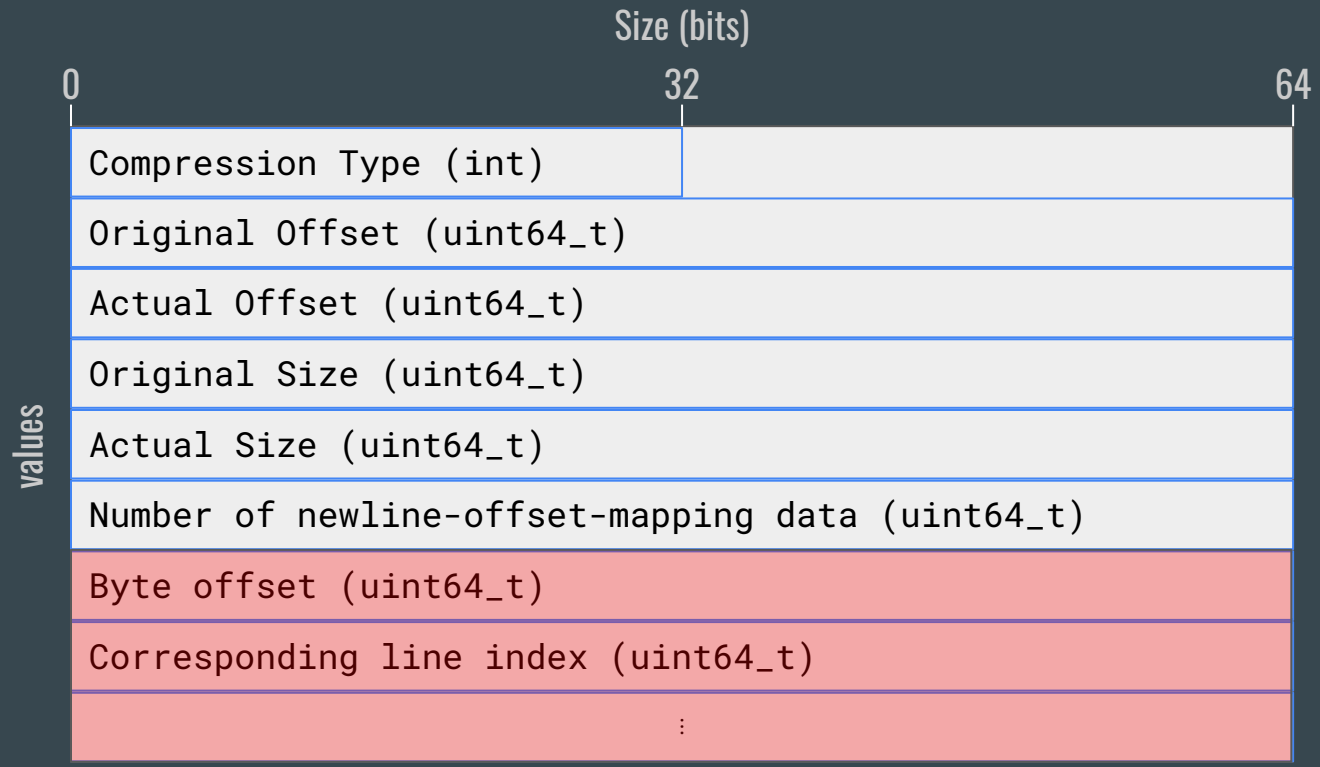
```
[lfreist@xps ~]$ time ./example_grep Sherlock data.txt

real   0m0.933s
user   0m0.640s
sys    0m0.320s


[lfreist@xps ~]$ time grep Sherlock data.txt

real   0m3.572s
user   0m2.626s
sys    0m0.869s
```

Metafile Structure



Appendix