Hi-C interaction matrix correction using ICE in Rust

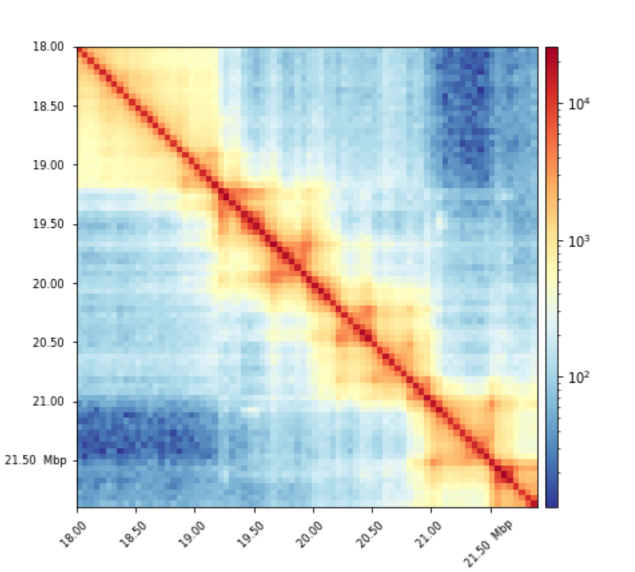
Bachelor thesis defense

Felix Karg

17. Juli 2019

University of Freiburg

Hi-C Contact Matrix



Content

Content

High-Throughput 3C (Hi-C)

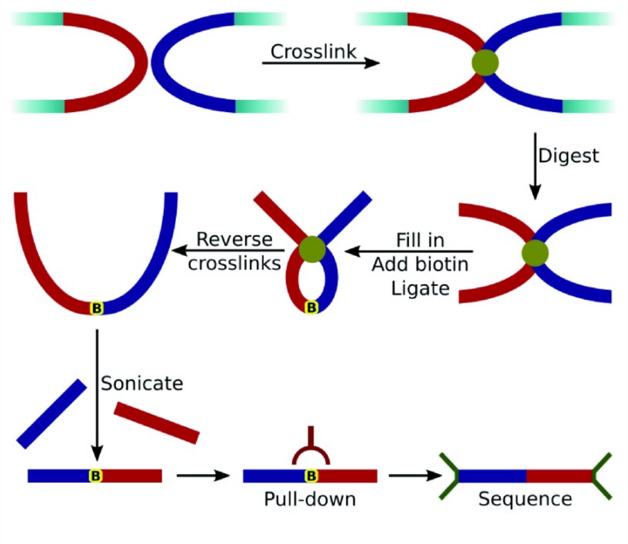


Image adapted from [1]

HiCExplorer

Helpful tools, especially for:

- Data correction
- Analysis
- Visualization

Content

ICE as described in Imakaev et al. 2012 [2]

- O_{ij}: raw data
- T_{ij}: relative contact probabilities
- B_i, B_j: cumulative biases

ICE as described in Imakaev et al. 2012 [2]

Goal: Obtain B and T_{ij}.

Do this by explicitly solving:

$$O_{ij} = B_i B_j T_{ij} \tag{1}$$

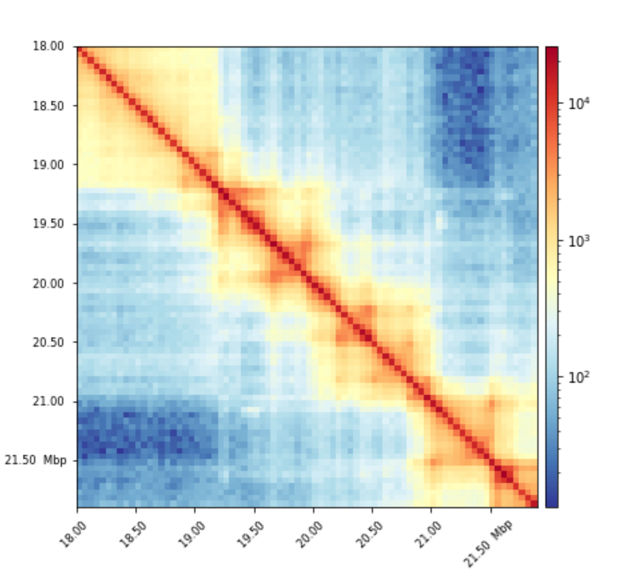
$$\sum_{i=1,|i-j|>1}^{N} T_{ij} = 1 \tag{2}$$

ICE as described in Imakaev et al. 2012 [2]

$$
T_{ij} = \begin{bmatrix}
d & d_{+1} & t_{1,3} & \dots & \dots & t_{1,n} \\
d_{-1} & d & d_{+1} & \dots & \dots & t_{2,n} \\
\vdots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \vdots \\
t_{n-1,1} & \dots & \dots & d_{-1} & d & d_{+1} \\
t_{n,1} & \dots & \dots & t_{n,n-2} & d_{-1} & d
\end{bmatrix}
$$

$$
\sum_{i=1,|i-j|>1}^{N} T_{ij} = 1
$$

ICE as described in Imakaev et al. 2012 [2]



Content

Rust: Short History

- Started out in 2006
- Mozilla started sponsoring in 2009
- Self-compiled in 2011
- 1.0 released in 2015

Rust: Language Features

- Syntax seems similar to C/C++
- Semantically closer to ML/Haskell
- Memory-Safety (no NULL)
- Memory Management (RAII)
- Ownership
- Borrowing
- Lifetimes

Rust: Comparison

Speed coparison	C	Rust	C++
n-body	7.49	**5.72**	8.18
binary-trees	3.48	**3.15**	3.79
pidigits	**1.75**	**1.75**	1.89
reverse-complement	1.78	1.61	**1.55**
spectral-norm	1.98	**1.97**	1.98
fannkuch-redux	**8.61**	10.23	10.08
k-nucleotide	5.01	5.25	**3.76**
fasta	1.36	1.47	**1.33**
mandelbrot	1.65	1.96	**1.50**
regex-redux	**1.46**	2.43	1.82
Fastest in:	3/10	4/10	4/10

Runtime measured in **seconds**. Numbers for C from [3] and for C++ from [4]. Both show the same numbers for Rust.

Rust: Advantages and Disadvantages

General Advantages:

- High-Level Features
- Fast Language
- Safe Memory handling
- Strong Type system

General Disadvantages:

- Young ecosystem
- Steep learning curve
- Higher initial compile times
- Language Features not yet available

Rust: Advantages and Disadvantages

Advantages for this Project:

- Own CSRMatrix implementation
- No big dependencies
- Faster and smaller, very specific

Disadvantages for this Project:

- No general implementation of CSRMatrix
- Only subset of features when compared to SciPy implementation

Content

API: Rust to Python

There is three main ways to use Rust from Python:

- rust-cpython
- pyO3
- dylib

rust-cpython

- Similar to C-Python headers
- Grants access to the Python GIL
- renaming needed
- stable rust
- easy package creation

pyO3

- fork from rust-cpython
- Grants access to the Python GIL
- renaming needed
- nightly rust
- easy package creation

dylib

- recommended in the Rust book
- no renaming needed
- stable rust
- No access to the Python GIL
- package creation possible

API comparison for Rust in Python

API Comparison	rust-cpython	pyO3	dylib
Memory from Python	Yes	Yes	**Optional**
renaming needed	Yes	Yes	**No**
stable Rust	**Yes**	No	**Yes**
platform-specific	Yes	Yes	**No**
implementation effort	Medium	Medium	**Low**
creating python packages	**Easy**	**Easy**	Normal
Good in:	2/6	1/6	**5/6**

Content

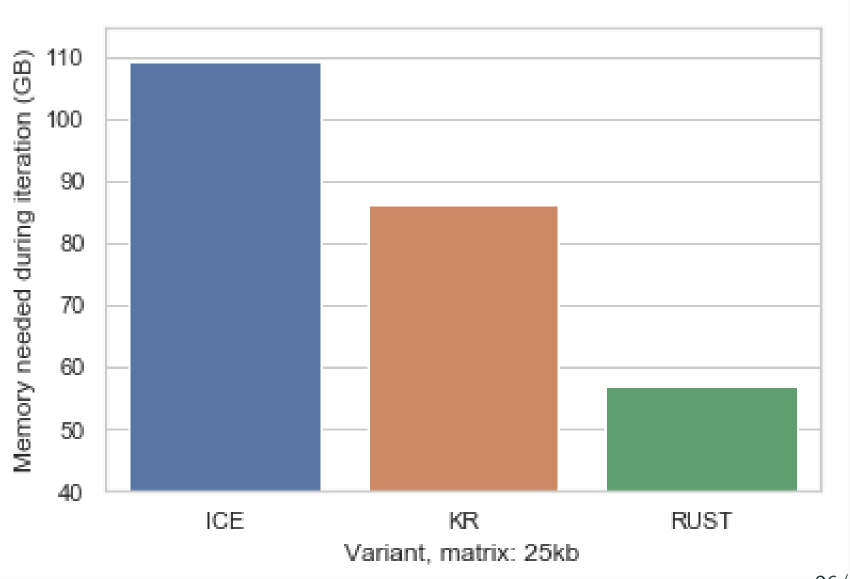
Compared implementations

- 'ICE' - Python implementation
- 'KR' - C++ implementation
- 'RUST' - this implementation

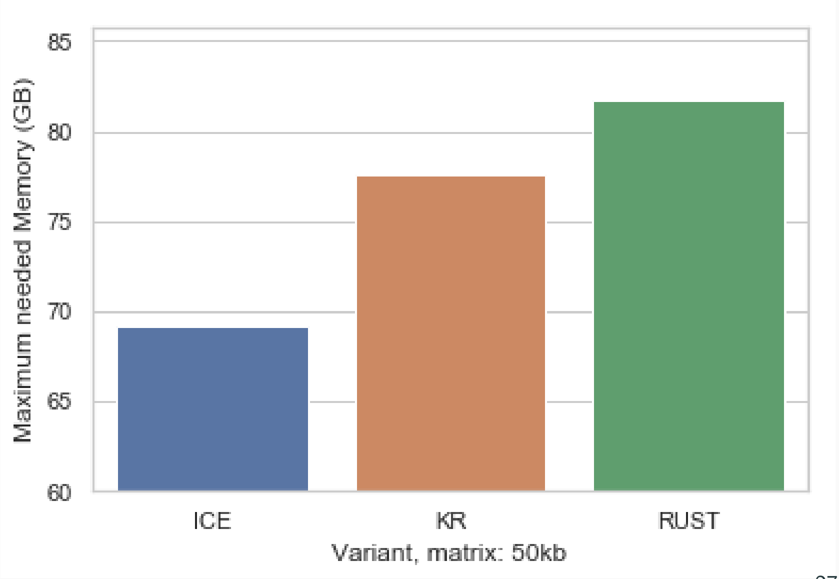
Data for Testing

Name	25kb	50kb
Filesize	1.1 GByte	732 MByte
Size	123,841	61,928
Bin length	25,000	50,000
Non-zero elements	1,530,533,003	1,053,216,825

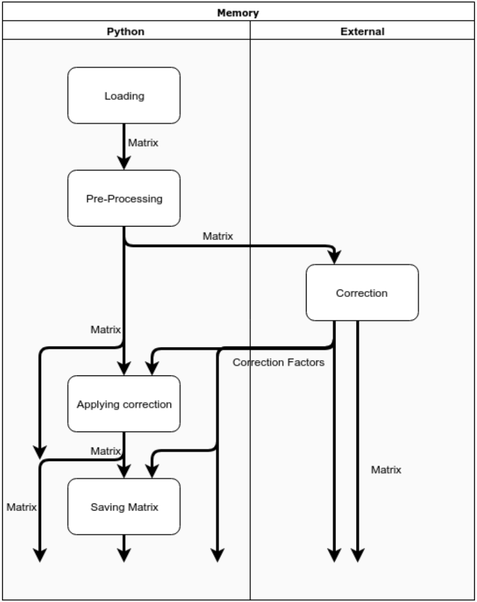
Memory Requirements during correction



Peak Memory Usage



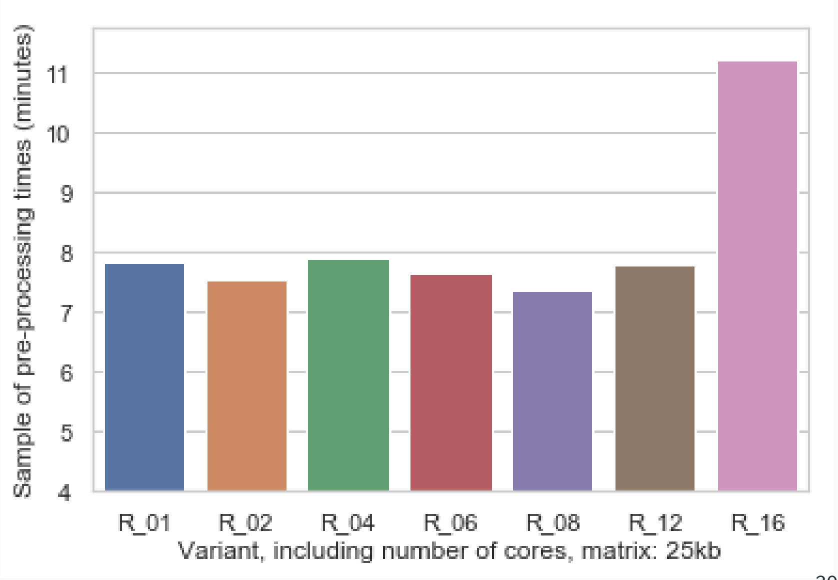
Control Flow Diagram



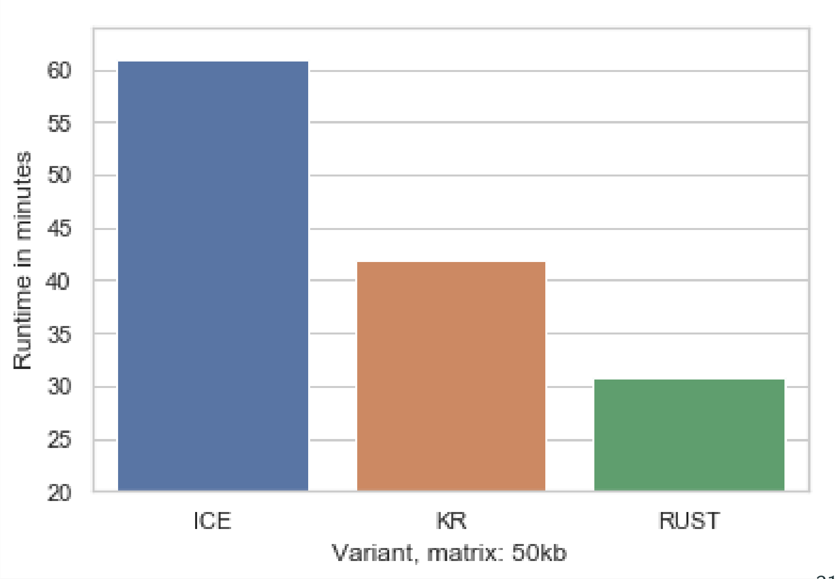
Comparison of Memory needs

Memory needs Comparison	ICE	KR	RUST
During correction (50kb)	54.6	43.1	**39**
Maximum (50kb)	**69.2**	77.6	81.7
During correction (25kb)	110	86	**57**
Maximum (25kb)	**110**	112.7	118.6

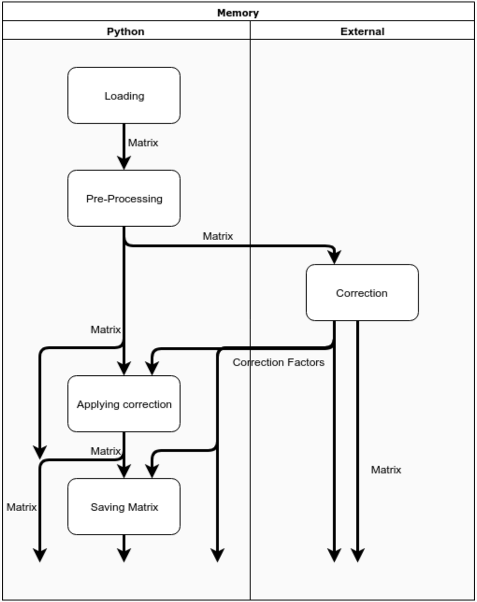
Loading Times



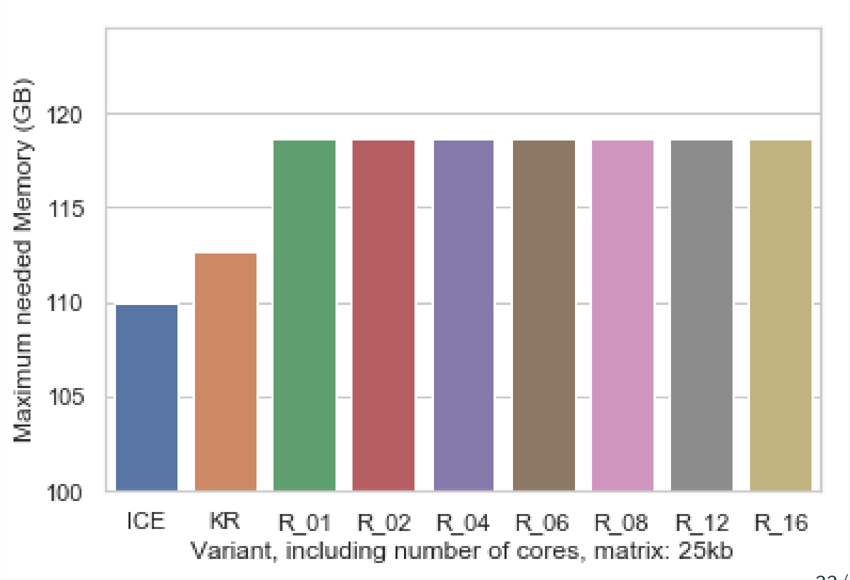
Runtime Length



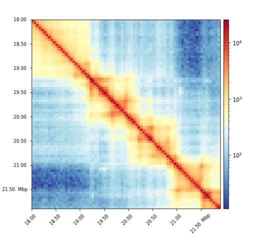
Multicore Runtime Length Comparison



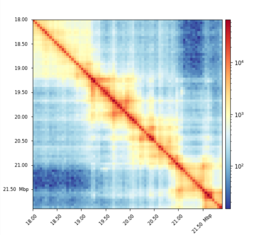
Multicore Memory Comparison



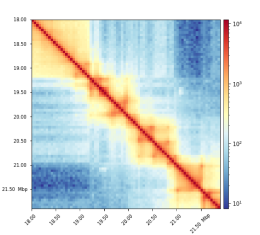
Comparison of Results



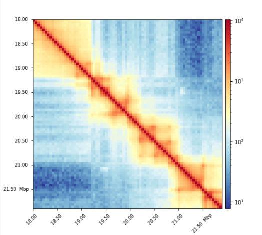
(a) Uncorrected



(b) RUST



(c) KR



(d) ICE

Content

Conclusion Regarding the Integration of RUST

- Test the alternatives more
- Better integration should be possible
- All in all: Went better than expected

Conclusion Regarding Computation Comparisons

- Reduction of memory usage during correction achieved
- Reduction of runtime achieved
- Parallelism does not offer significant benefits yet

Remaining Questions

- Writing Code for faster parallelism
- Speedup when parallelizing more
- Pure implementations needing even less memory?
- Is KR faster for bigger Matrices?

Content

Sources i

- S. Wingett, P. Ewels, M. Furlan-Magaril, T. Nagano, S. Schoenfelder, P. Fraser, and S. Andrews, "Hicup: pipeline for mapping and processing hi-c data," *F1000Research*, vol. 4, 2015.
- M. Imakaev, G. Fudenberg, R. P. McCord, N. Naumova, A. Goloborodko, B. R. Lajoie, J. Dekker, and L. A. Mirny, "Iterative correction of hi-c data reveals hallmarks of chromosome organization," *Nature methods*, vol. 9, no. 10, p. 999, 2012.

Sources ii

- "Rust comparison with c."
  https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust.html, 2019.
  accessed 2019-06-26.
- "Rust comparison with c++."
  https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-gpp.html, 2019.
  accessed 2019-06-26.

Sources iii

- G. Li, L. Cai, H. Chang, P. Hong, Q. Zhou, E. V. Kulakova, N. A. Kolchanov, and Y. Ruan, "Chromatin interaction analysis with paired-end tag (chia-pet) sequencing technology and application," *BMC Genomics*, vol. 15, p. S11, Dec 2014.

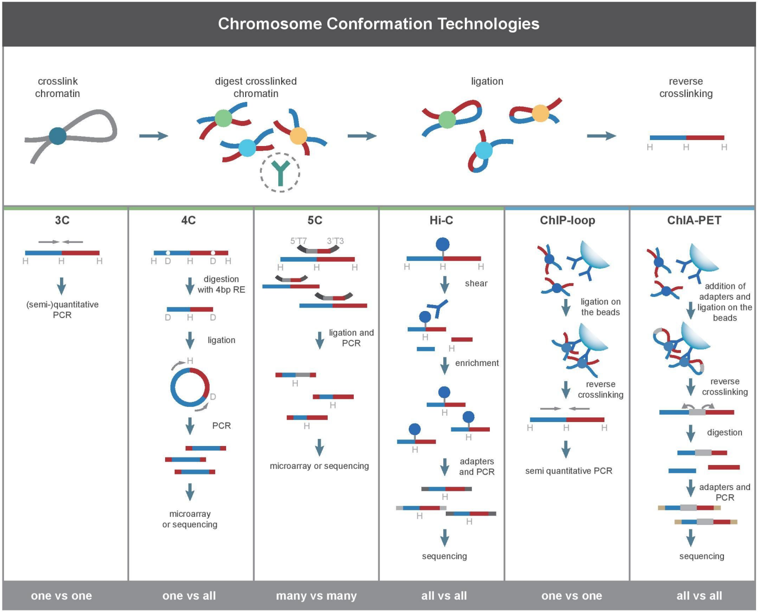

Chromosome Conformation Technologies
crosslink chromatin
digest crosslinked chromatin
ligation
reverse crosslinking
3C
(semi-)quantitative PCR
4C
digestion with 4bp RE
ligation
PCR
microarray or sequencing
5C
5'T7
3'T3
ligation and PCR
microarray or sequencing
Hi-C
shear
enrichment
adapters and PCR
sequencing
ChIP-loop
ligation on the beads
reverse crosslinking
semi quantitative PCR
ChIA-PET
addition of adapters and ligation on the beads
reverse crosslinking
digestion
adapters and PCR
sequencing
one vs one
one vs all
many vs many
all vs all
one vs one
all vs all

Image from [5].

ICE as described in Imakaev et al. 2012 [2]

Each iteration, compute:

$$S_i = \sum_j W_{ij} \tag{3}$$

$$\Delta B_i = S_i / mean(S) \tag{4}$$

$$W_{ij} = W_{ij} / \Delta B_i \Delta B_j \tag{5}$$

$$B_i = B_i \cdot \Delta B_i \tag{6}$$

Rust: Code Example

Code Example 1

```rust
fn main() {
    let mut v = vec![];     // ---|
    v.push("Hello");        // <--|
                            //    |
    let x = &v[0];          // -| |
                            //  | |
                            //  | |
    v.push("world");        // <X-|
    println!("{}", x);      // -| |
}                           // ---|
```

Rust: Code Example

Output Nr. 1

```
error[E0502]: cannot borrow `v` as mutable because it is
also borrowed as immutable
 --> src/main.rs:5:5
  |
5 |     let x = &v[0];
  |              - immutable borrow occurs here
8 |     v.push("world");
  |     ^^^^^^^^^^^^^^^ mutable borrow occurs here
9 |     println!("{}", x);
  |                    - immutable borrow later used here
```

Rust: Code Example

Code Example 2

```rust
fn main() {
    let mut v = vec![];     // ---|
    v.push("Hello");        // <--|
                            //    |
    let x = &v[0];          // -| |
    println!("{}", x);      // -| |
                            //    |
    v.push("world");        // <--|
    println!("{}", v[1]);   // <--|
}                           // ---|
```

Rust: Code Example

Output Nr. 2

```
Hello
world
```

Test Server Specification

Virtual Server Specification	
Available Cores / Threads	16 / 32
Working Memory (RAM)	120 GByte

Processor Specification	
Processor	Intel® Xeon® E5-2630V4
Number of Cores/Threads	10 / 20
Base/Turbo frequency	2.2 GHz / 3.1 GHz