Shouji: A Fast and Efficient Pre-Alignment Filter for Sequence Alignment

Mohammed Alser, Hasan Hassan, Akash Kumar,

Onur Mutlu, Can Alkan

Bioinformatics 2019

presented by Tobias Oberdörfer

Outline

Background on Bioinformatics

Problem of Read Mapping

Motivation & Goal for Shouji

Key Ideas

- Pre-alignment filter using sliding window approach
- Hardware accelerator for this pre-alignment filter

Results & Conclusion

Discussion

Future work & Questions

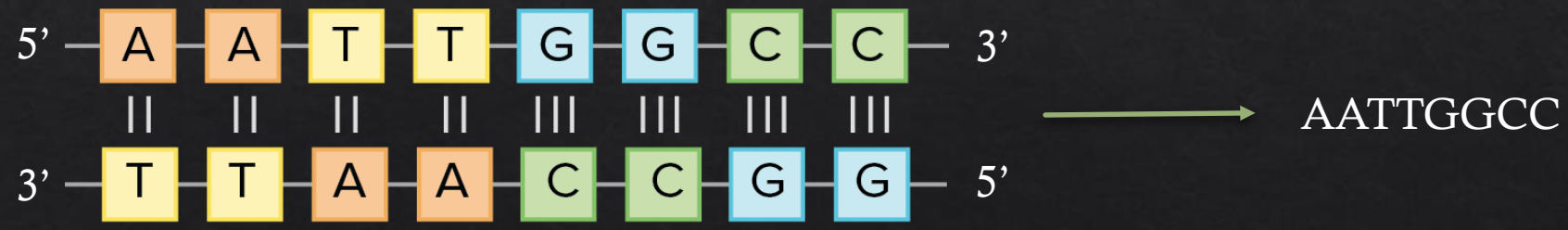
Background on Bioinformatics

What is Bioinformatics?

Definition of Bioinformatics [1]:

"Bioinformatics is an interdisciplinary field that develops methods and software tools for understanding biological data, in particular when the data sets are large and complex."

These datasets can be amino acid or nucleotide sequences



[1] Definition from Wikipedia

Read Mapping

Detection of **differences and similarities** between read and reference sequences.

Read: **TTTTACTGTTCTCCCTTTGAATACAATATATCTATATTTCCCTCTG…**
Reference: **TTTTACTGTTCTCCCTTTGAAATGACAATATATCTATATTTCCCTCTG…**

Read: **TTTTACTGTTCTCCCTTTGAAXTXACAATATATCTATATTTCCCTCTG…**
Reference: **TTTTACTGTTCTCCCTTTGAAATGACAATATATCTATATTTCCCTCTG…**

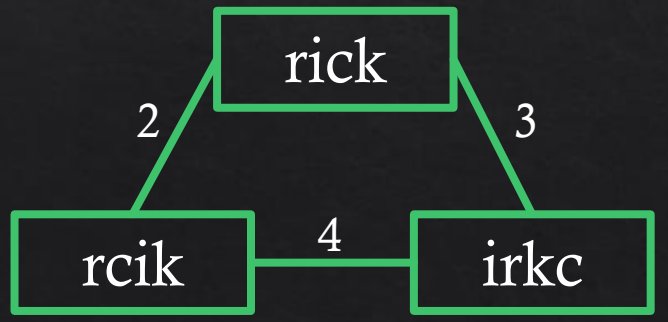
Read Mapping

Read alignment is **useful for many different applications**, e.g.:

- Detect gene mutations
- Strain detection of viruses or bacteria

Not only in bioinformatics, but wherever text comparison is used:

- OCR error correction
- Auto-correction



Levenshtein distance

How to do Read Mapping?

Two closely related ways of working on read mapping

- **Edit distance**
  - **Minimum number of edits** to change one sequence into the other
  - Edit operations include deletion, insertion, and substitution
  - Can have **different orders** with the same number of edits
- **Pairwise alignment**
  - **Use scoring function** on type of operation to order edits
  - Search for **optimal arrangement**
  - Requires backtracking

Problem of Read Mapping

Really slow!

Enumerating all possible prefixes is necessary for the optimum!

Proof idea via contradiction:

- Assume we found the optimum without enumerating all possible prefixes.
- Now let one prefix, which is not enumerated, be part of the optimum.

Edit distance and pairwise alignment are **non-additive measures**

- Calculating the sum for a divided sequence is not necessarily the same as for calculating it directly on the original sequence

Our best option

Enumerating all possible prefixes is necessary for the optimum!

Use Dynamic Programming (short DP)

- Use a matrix to remember already calculated prefixes
- Most implementations in **quadratic space and time complexity** $O(m^2)$ [1]
- Fastest known algorithm[2] still takes $O(m^2/\log^2 m)$
- May **require backtracking** to read optimum from penalty

[1] Big O notation is a mathematical notation that describes the limiting behaviour of a function
[2] Masek, W. J. and Paterson, M. S. (1980) A faster algorithm computing string edit distances

Our best option

Algorithms using Dynamic Programming

- For pairwise alignment: Needleman–Wunsch or Smith–Waterman
- For edit distance : Levenshtein distance

Acceleration of Dynamic Programming algorithms in previous work

- Calculate only the necessary parts of the DP-matrix
  - E.g. Banded Smith-Waterman is only calculating some diagonals
- Hardware-accelerators in FPGA, using SIMD or GPU
  - Parallelism for DP is hard, as there are data-dependencies between cells

Is there another way?

Use pre-alignment filtering heuristics to **eliminate sequences before** using the computationally-expensive optimal alignment algorithm.

Recent work in this direction:

- SHD a SIMD-friendly bit-vector filter (Xin et al., 2015)
- GRIM-Filter exploiting 3D-stacked DRAM (Kim et al., 2018)
- GateKeeper (Alser et al., Bioinformatics 2017)
- MAGNET (Alser et al., July 2017)

Motivation & Goal

Motivation

However, most of the previous work regarding DP

- **Simplify the scoring function** of the algorithm used
- Provide **non-optimal sequence alignment**

And regarding pre-alignment filtering

- Are **not scalable** enough
- **Accuracy** is **too low** for most applications

Goal

Hence the goal is to **reduce time spent** on calculating optimal alignment while still **maintaining high filtering accuracy** using a new pre-alignment filter approach

Key Ideas

Key Ideas

High filtering accuracy by **correctly detecting common subsequences**

- By rapidly excluding dissimilar sequences from the optimal alignment calculation using a **sliding search window** approach

Hardware accelerator for this new pre-alignment filter

- Judicious use of the parallelism friendly architecture of modern FPGAs
- Clever observation for calculation of filter steps

Filtering Strategy

The **pigeonhole principle**

Example with 10 pigeons in 9 holes

Because 10 > 9 holds, the pigeonhole principle says that at least one hole has more than one pigeon. [1]

[1] Example and picture taken from Wikipedia Pigeonhole principle

Filtering Strategy

Based on the **pigeonhole principle**:

If two sequences **differ by *E* edits**,

Then **at most *E*+1** non overlapping common subsequences **can exist**

And the **total length** of those must be **at least m-*E***,

where m sequence length and *E* the edit distance threshold

Example with $E = 1$ and $m = 7$:
Read: TACXTGT
Reference: TACTGTA

Common subsequences:
TAC and **TGT**
Total length: 6 = 7-1

Filtering Strategy

Based on the **pigeonhole principle**:

If two sequences **differ by *E* edits**:

Then **at most *E*+1** non overlapping common subsequences **can exist**

And the **total length** of those must be **at least m-*E***

where m sequence length and *E* the edit distance threshold

Example with $E = 1$ and $m = 7$:
Read: TACXXTG
Reference: TACTGTA

Common subsequences:
TAC and **TG**
Total length: $5 < 6 = 7-1$

Filtering Strategy

Based on the **pigeonhole principle**:

If two sequences **differ by *E* edits**:

Then **at most *E*+1** non overlapping common subsequences **can exist**

And the **total length** of those must be **at least m-*E***

where m sequence length and *E* the edit distance threshold

This in turn means for our filter, that **any two sequences, that have less than m-*E* total overlapping length** of the common subsequences, **can be rejected** without further thought.

Filtering Strategy

Implemented in three steps:

1) Build the neighborhood map
2) Find common subsequences using sliding window approach
3) Accept or reject pair after summing up windows

Filtering Strategy

1) Build the **neighborhood map**

For edit distance threshold *E* and m length of sequences

- Represented as a **m-by-m matrix N** of binary values
- Row represents the **T**ext sequence and columns the **P**attern sequence

$$N[i,j] = \begin{cases} 0, & \text{if } P[i] = T[j] \\ 1, & \text{if } P[i] \neq T[j] \end{cases}$$

Calculation formula of neighborhood map entry

	G	G	T	G	C	A	G	A	G	C	T	C
G	0	0	1									
G												
T												
G												
A												
G												
A												
G												
T												
T												
G												
T												

Filtering Strategy

1) Build the **neighborhood map**

$$N[i,j] = \begin{cases} 0, & \text{if } P[i] = T[j] \\ 1, & \text{if } P[i] \neq T[j] \end{cases}$$

Calculation formula of neighborhood map entry

Because of the length requirement of m-E, we only need the diagonals, i.e. i and j should satisfy **$1 \leq i \leq m$** and **$i\text{-}E \leq j \leq i\text{+}E$**

	G	G	T	G	C	A	G	A	G	C	T	C
G	0	0	1	0								
G	0	0	1	0	1							
T	1	1	0	1	1	1						
G	0	0	1	0	1	1	0					
A		1	1	1	1	0	1	0				
G			1	0	1	1	0	1	0			
A				1	1	0	1	0	1	1		
G					1	1	0	1	0	1	1	
T						1	1	1	1	1	0	1
T							1	1	1	1	0	1
G								1	0	1	1	1
T									1	1	0	1

Filtering Strategy

2) **Find common subsequences** using the sliding window approach

- Common subsequence in neighborhood is represented as **consecutive zeros** in a diagonal
- Sliding window of size four found to be optimal

	G	G	T	G	C	A	G	A	G	C	T	C
G	0	0	1	0								
G	0	0	1	0	1							
T	1	1	0	1	1	1						
G	0	0	1	0	1	1	0					
A		1	1	1	1	0	1	0				
G			1	0	1	1	0	1	0			
A				1	1	0	1	0	1	1		
G					1	1	0	1	0	1	1	
T						1	1	1	1	1	0	1
T							1	1	1	1	0	1
G								1	0	1	1	1
T									1	1	0	1

Filtering Strategy

3) **Sum up** windows and **accept or reject** the two sequences depending on the sum

- Done using the so called **Shouji bit-vector**
- Sum can be calculated using a 4-bit hardware lookup table for the sliding windows

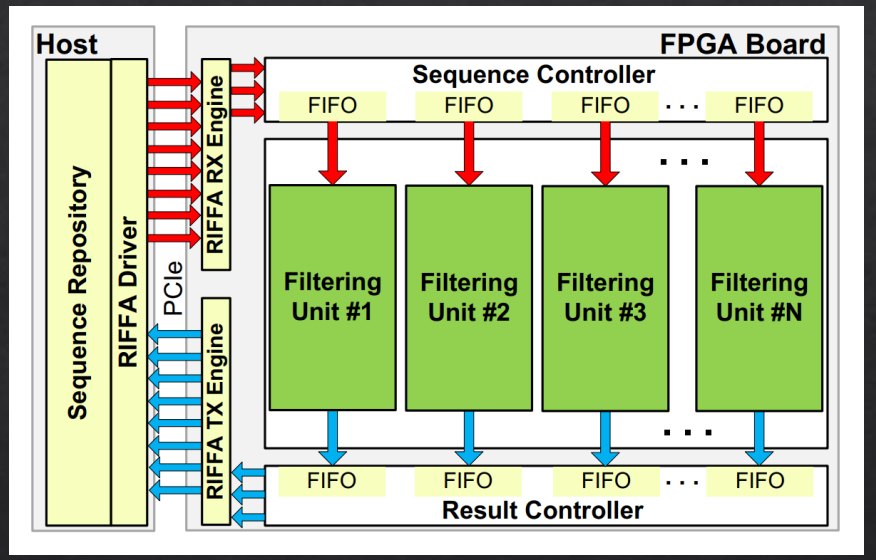
	G	G	T	G	C	A	G	A	G	C	T	C
G	0	0	1	0								
G	0	0	1	0	1							
T	1	1	0	1	1	1						
G	0	0	1	0	1	1	0					
A		1	1	1	1	0	1	0				
G			1	0	1	1	0	1	0			
A				1	1	0	1	0	1	1		
G					1	1	0	1	0	1	1	
T						1	1	1	1	1	0	1
T							1	1	1	1	0	1
G								1	0	1	1	1
T									1	1	0	1

Shouji bit-vector:	0	0	0	0	1	0	0	0	0	1	0	1

Hardware Accelerator

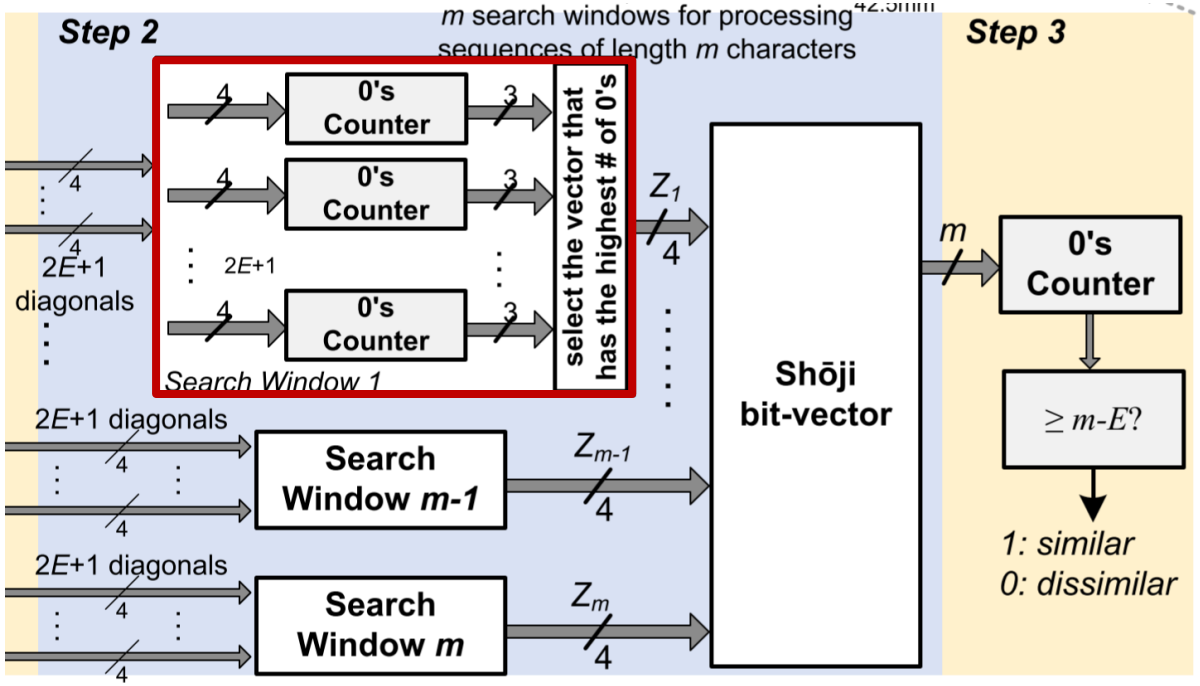
Accelerator is **built up modular**:

- Sequence controller
  - Gets the input sequences from PCIe
  - Distributes the sequences among the filtering units
- n filtering units
  - Can be **as many as fit** on the FPGA
- Result controller
  - Accumulated results of filtering units
  - Sends the results back over PCIe



Filtering Unit

- For each diagonal in a search window, we have a counter.
  - Implemented as hardware-lookup-tables
- In total **m search windows over the neighborhood map**
- Shouji bit-vector to accumulate sum



Results & Conclusions

Results

The following things were evaluated:

- **Filtering accuracy** of different algorithms
- FPGA **resource utilization** for hardware accelerators
- **Execution time** of four mentioned pre alignment filters for filtering and end-to-end

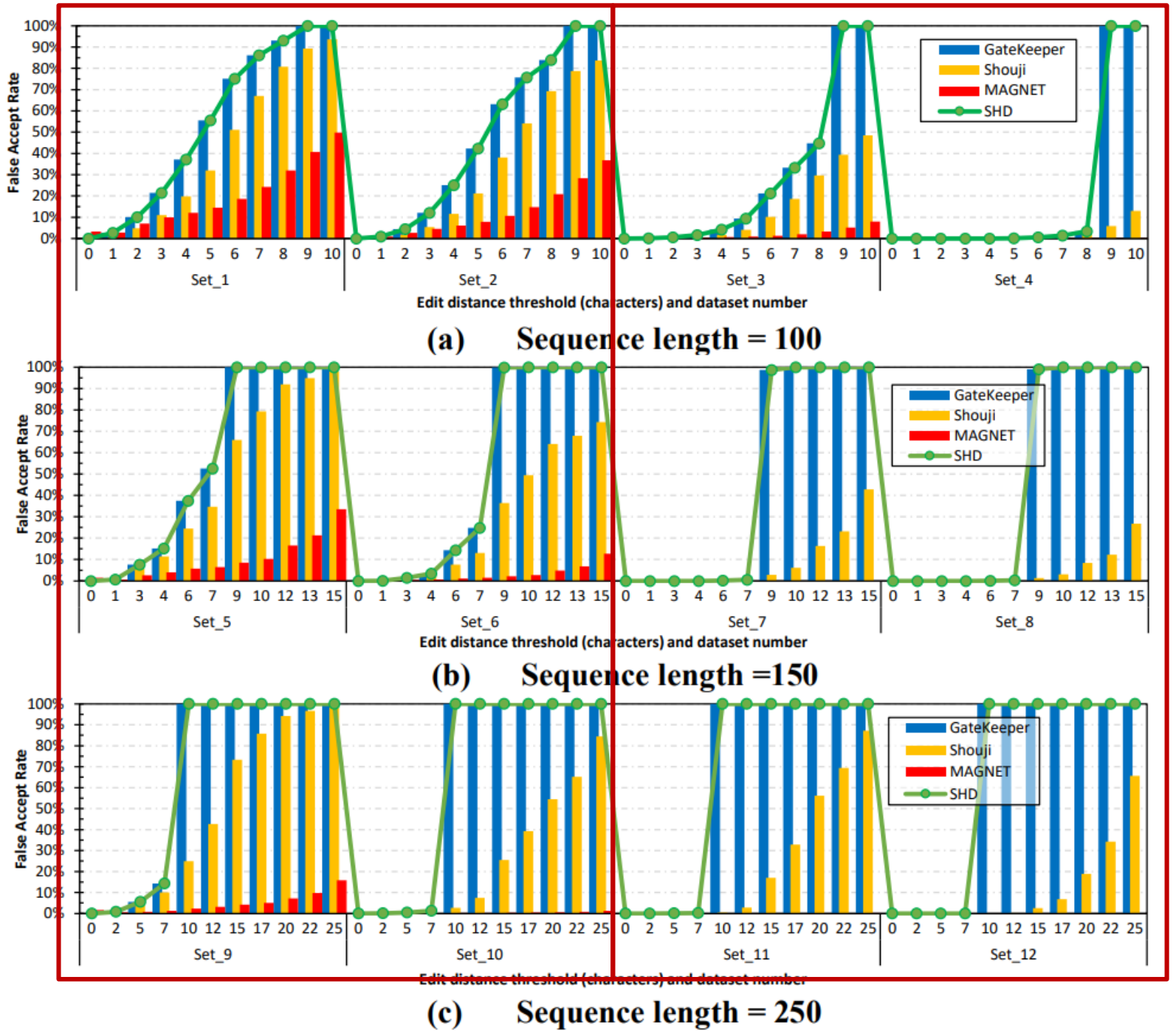
Testbench setup:

- 3.6 GHz Intel i7-3820 CPU
- 8 GB RAM
- Xilinx Virtex 7 VC709 board (Xilinx, 2014) to implement accelerators
- FPGA design using Vivado 2015.4 in synthesizable Verilog

Filtering Accuracy

Accuracy tested by looking at false accept rate

- All algorithms **less accurate at low-edit datasets**
- SHD and GateKeeper useless after some threshold
- Shouji better than SHD and GateKeeper
- MAGNET best for high-edit datasets



FPGA Resource Utilization

2% and 5% edit distances, FPGA on 250MHz

- SHD cannot be directly implemented in FPGA
- MAGNET uses a lot more resources than Shouji per filtering unit
- **Shouji** and GateKeeper are **close in usage**

Table 1: FPGA resource usage for a single filtering unit of Shouji, MAGNET, and GateKeeper, for a sequence length of 100 and under different edit distance thresholds. We highlight the best value in each column.

Filter	*E*	Single Filtering Unit		Max. No. of Filtering Units
		Slice LUT	Slice Register	
Shouji	2	0.69%	0.01%	16
	5	1.72%	0.01%	16
MAGNET	2	10.50%	0.8%	8
	5	37.80%	2.30%	2
GateKeeper	2	**0.39%**	**0.01%**	**16**
	5	0.71%	0.01%	16

Filtering Time

Same setup as previous and SHD on a single CPU core

- Shouji as fast as GateKeeper
- Shouji 2 to 8 times faster than MAGNET
- Shouji a lot faster than SHD

Table 2: Execution time (in seconds) of FPGA-based GateKeeper, MAGNET, Shouji, and CPU-based SHD under different edit distance thresholds and sequence lengths. We use set_1 to set_4 for a sequence length of 100 and set_9 to set_12 for a sequence length of 250. We provide the performance results for both a single filtering unit and the maximum number of filtering units (in parentheses).

E	GateKeeper	MAGNET	Shouji	SHD
		Sequence Length = 100		
2	**2.89[a] (0.18[b], 16[c])**	2.89 (0.36, 8)	**2.89 (0.18, 16)**	60.33
5	**2.89 (0.18, 16)**	2.89 (1.45, 2)	**2.89 (0.18, 16)**	67.92
		Sequence Length = 250		
5	**5.78 (0.72, 8)**	5.78 (2.89[d], 2)	**5.78 (0.72[d], 8)**	141.09
15	**5.78 (0.72, 8)**	5.78 (5.78[d], 1)	**5.78 (0.72[d], 8)**	163.82

[a] Execution time, in seconds, for a single filtering unit.

[b] Execution time, in seconds, for maximum filtering units.

[c] The number of filtering units.

[d] Theoretical results based on the resource utilization and data throughput.

Execution Time

Integration of pre-alignment filters in state-of-the-art aligners

- Shouji **best** speedup for **low edit distance threshold**
- Shouji is the **best performing** with up to **18.8x speedup**

Table 3: End-to-end execution time (in seconds) for several state-of-the-art sequence alignment algorithms, with and without pre-alignment filters (Shouji, MAGNET, GateKeeper, and SHD) and across different edit distance thresholds.

E	Edlib	w/ Shouji	w/ MAGNET	w/ GateKeeper	w/ SHD
2	506.66	**26.86**	30.69	36.39	96.54
5	632.95	147.20	**106.80**	208.77	276.51
E	Parasail	w/ Shouji	w/ MAGNET	w/ GateKeeper	w/ SHD
2	1310.96	**69.21**	78.83	93.87	154.02
5	2044.58	475.08	**341.77**	673.99	741.73
E	FPGASW	w/ Shouji	w/ MAGNET	w/ GateKeeper	w/ SHD
2	11.33	**0.78**	1.04	0.99	61.14
5	11.33	**2.81**	3.34	3.91	71.65
E	CUDASW++ 3.0	w/ Shouji	w/ MAGNET	w/ GateKeeper	w/ SHD
2	10.08	**0.71**	0.96	0.90	61.05
5	10.08	**2.52**	3.13	3.50	71.24
E	GSWABE	w/ Shouji	w/ MAGNET	w/ GateKeeper	w/ SHD
2	61.86	**3.44**	4.06	4.60	64.75
5	61.86	14.55	**11.75**	20.57	88.31

Conclusion

- Shouji and MAGNET are asymptotically inexpensive, because they **run in linear time** on the input length **O(m)**
- Shouji and MAGNET significantly **improve filtering accuracy** over GateKeeper and SHD
- **FPGA implementations significant improvement** of speed over the equivalent CPU implementations
- **Shouji is scalable** and can be used in many different pipelines without negatively affecting them

General questions so far?

Discussion

“As such, we hope that it catalyzes the adoption of specialized pre-alignment accelerators in genome sequence analysis.”

M. Alser et al. in **Shouji: A Fast and Efficient Pre-Alignment Filter for Sequence Alignment (2019)**

This should be the case,
as Shouji is a solution to **speed up the ever more important field** of sequencing in Bioinformatics

Strengths

- Shouji is **scalable** because there is **no interdependence** between windows
- Elegant hardware/software **co-design**
- Shouji **sacrifices no capabilities** of other pipelines/sequence aligners and **can be adapted for any** bioinformatics **pipeline** performing sequence alignment
- Accelerator, in a FPGA, **not using vendor specific functions** and as such can be used on any available architecture supporting FPGAs
- All code and designs are **open-sourced**
- With some knowledge of common subsequences it is an easy paper to understand

Weaknesses

- Shouji is **not so good for long sequences**, but not verified
- **Data movement** is still a big **bottleneck** that was not solved
- High false-accept rate for low-edit sequences
- Specialized hardware chips may discourage the target users
- Specialized hardware can be expensive

Future Work

Future Work

- **Better connection** between accelerators and CPU
  - Newer PCIe channels or more lanes
    - PCIe Gen 5 has four times the transfer rate of Gen 3
      - That would mean the 4-lane connection would have 15.754 GB/s throughput
  - Different and **newer protocols** than RIFFA 2.2

JetStream: An Open-Source High-Performance PCI Express 3 Streaming Library for FPGA-to-Host and FPGA-to-FPGA Communication

Malte Vesper, Dirk Koch
School of Computer Science
The University of Manchester
malte.vesper@postgrad.manchester.ac.uk
dirk.koch@manchester.ac.uk

Kizheppatt Vipin
Mahindra École Centrale
vipin.kizheppatt@mechyd.edu.in

Suhaib A. Fahmy
School of Engineering
University of Warwick
s.fahmy@warwick.ac.uk

VerCoLib: Fast and Versatile Communication for FPGAs via PCI Express

Oğuzhan Sezenlik[1] · Sebastian Schüller[1] · Joachim K. Anlauf[1]

Received: 10 September 2018 / Revised: 6 March 2019 / Accepted: 25 June 2019

Future Work

- Another avenue for future work is the exploration of **pre-alignment filters on longer sequences** (in the thousands of base-pairs)

SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs

Mohammed Alser [1,3], Taha Shahroodi [1], Juan Gómez-Luna [1], Can Alkan [3], and Onur Mutlu [1,2,3]

[1] Department of Computer Science, ETH Zurich, Zurich 8006, Switzerland
[2] Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh 15213, PA, USA
[3] Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey

Future Work

- If sequence alignment moves into the **cloud**, there is a need for **preservation of privacy** for ethical, security and legal reasons.

Secure Cloud Computing for Pairwise Sequence Alignment

Sergio Salinas
Department of Electrical Engineering and Computer Science
Wichita State University
salinas@cs.wichita.edu

Pan Li
Department of Electrical Engineering and Computer Science
Case Western Reserve University
lipan@case.edu

Questions

Questions

Data movement is still a bottleneck. How could we try to reduce it?

- For better performance of the current accelerator just **newer and faster I/O would already** help
- Bringing the **accelerator closer to memory**
  - Building a hardware accelerator with integrated memory
  - Building memory with integrated FPGA/a hardware accelerator
- Build accelerator **close to/in caches**
  - To still improve memory access and throughput without having to build new memory

Questions

Any ideas for reducing the adoption problem? Is it even a problem?

- Authors mention two ways to improve upon this weakness
  - **Closely integrate** the accelerator into sequencers for real-time pre-alignment filtering
  - Offer **cloud computing** with access to advanced FPGA chips
- **Market** the benefits of the **speedup** well and it should not pose a problem
  - Best example are mining accelerators for crypto currencies

Questions

Do you have any ideas where this way of speeding up DP-algorithms could also be useful?

- Many **DP-algorithms can benefit** from different insights of read mapping
  - Banded DP-algorithms are already used sometimes
  - Combining pre filtering with DP-algorithms is also looked at:

Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem

François Clautiaux[a,b,*], Ruslan Sadykov[b,a], François Vanderbeck[a,b], Quentin Viaud[a,b]

[a] *IMB, Université de Bordeaux, 351 cours de la Libération, 33405 Talence, France*
[b] *INRIA Bordeaux - Sud-Ouest, 200 avenue de la Vieille Tour, 33405 Talence, France*

Questions

Can you think of fields that could be similarly in need of string alignment as read mapping in bioinformatics?

- **Natural language processing**
  - OCR error correction
  - Autocorrection in text-based editors or apps
  - Reconstruction of languages using the comparative method
- Social sciences
- Business and marketing research

Shouji: A Fast and Efficient Pre-Alignment Filter for Sequence Alignment

Mohammed Alser, Hasan Hassan, Akash Kumar,

Onur Mutlu, Can Alkan

Bioinformatics 2019