



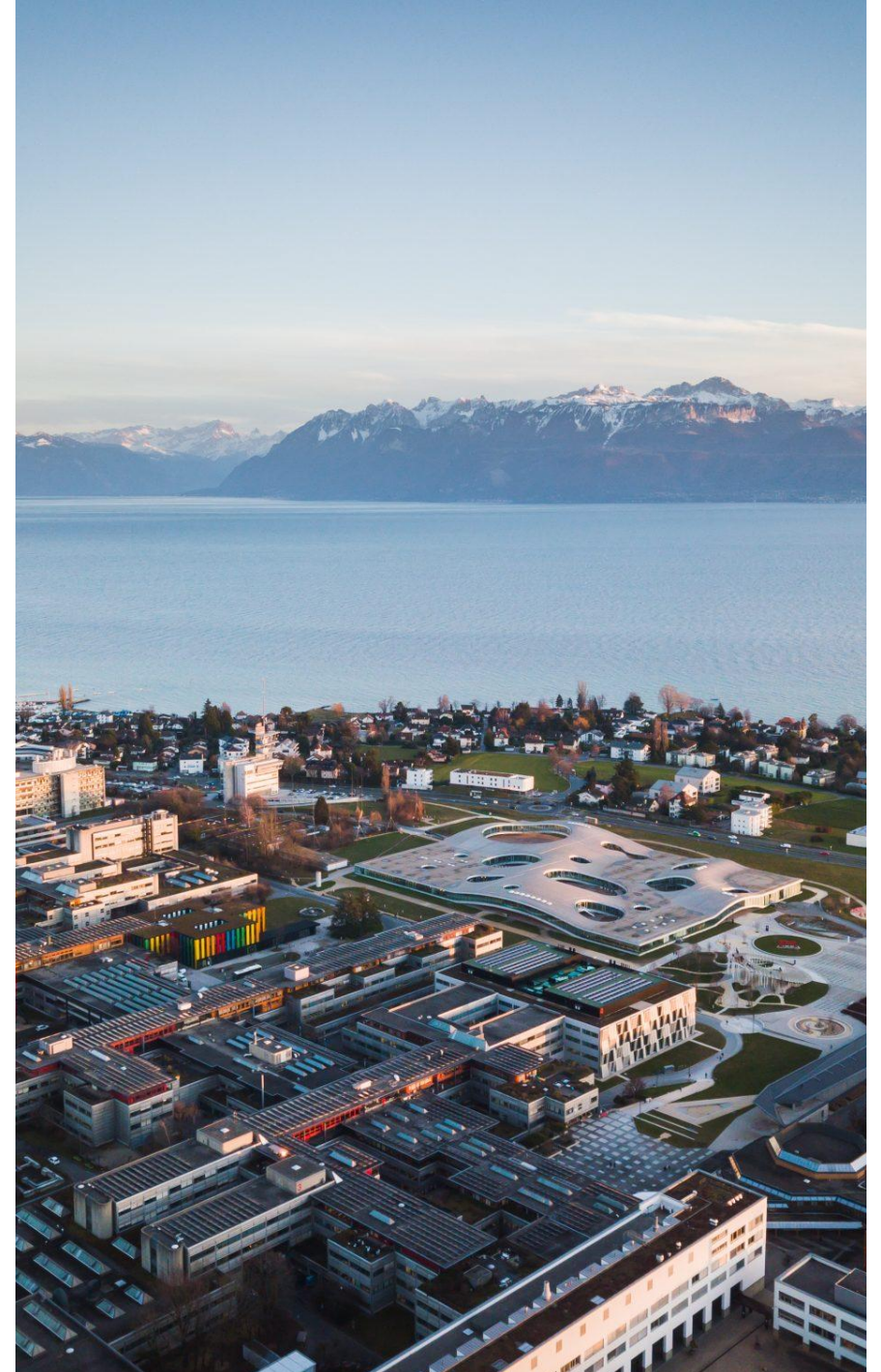
Final Project Presentation

Alejandro López Rodríguez, Pedro Palacios Almendros

EE-390(a): Lab on hardware-software digital systems codesign

Outline

- Single-worker acceleration
 - Specimen caching
 - Compression
 - Smith-Waterman algorithm
 - Linear systolic array
 - SW, Interrupts & Linux driver
- Multi-worker acceleration
 - 1. Across comparisons
 - 2. Across DB sequences
 - 3. Decoupling accelerators
- Results & Conclusion



Single-worker acceleration

1

Specimen caching

- **Cache all specimens in BRAMs**
 - Only load specimens from DRAM once
 - Use BRAM cached specimens for computing scores
- **Use 3 different AXI ports for maximum DRAM bandwidth**
 - Sequences (shared by specimens and DB)
 - Lengths (shared by specimens and DB)
 - Scores
- **We assume that specimens fit in BRAM**
 - Can be easily fixed by adding stride + offset for writing output and calling several times the accelerator
 - We preferred to focus on more important optimizations

Compression

- Use **only 2 bits** to represent a nucleobase
 - 8 bits \rightarrow 2 bits: 4x space savings
- Great **resource savings**:
 - Systolic array registers and LUTs, specimen BRAM cache, less DRAM bandwidth pressure
- Two compression strategies:
 - 1. Variable-size sequences**: More compression, at the cost of more complex hardware and burst access complexity (need manual `hls::burst_maxi`)
 - 2. Fixed-size sequences**: Less compression (assumes all sequences have 32 elements \rightarrow 64 bits), but substantially easier hardware: easier pipelining and bursting

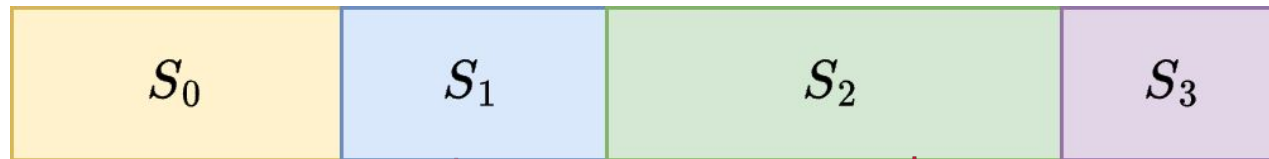
A \rightarrow 00

T \rightarrow 01

G \rightarrow 10

C \rightarrow 11

Variable size:



Fixed size:



Smith-Waterman algorithm

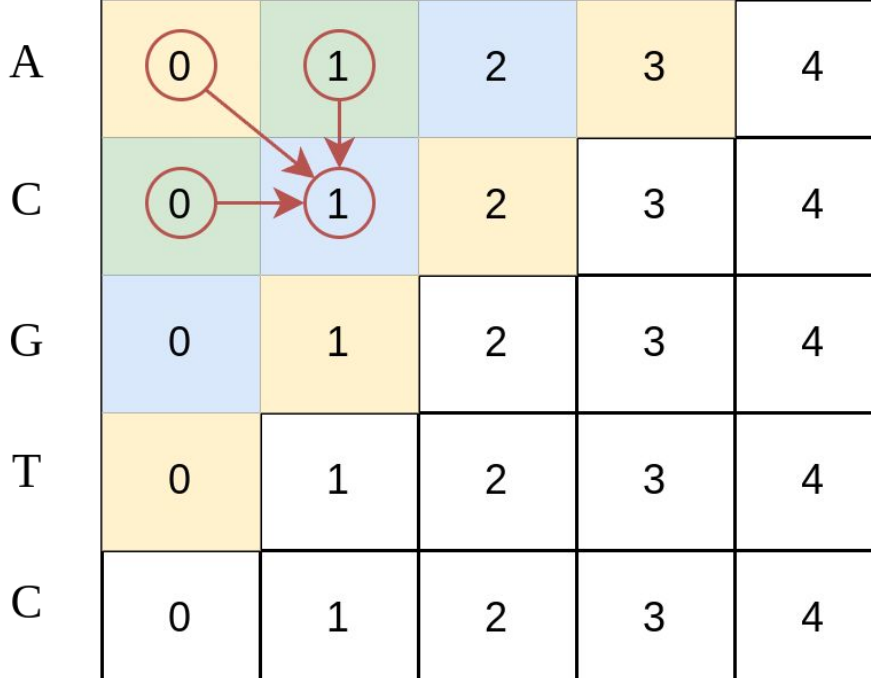
$$S[a, b] = \begin{cases} +1, & \text{if } a = b \\ -1, & \text{if } a \neq b \end{cases}$$

- The **backtracking** is **redundant** and can be eliminated: it will always return the max score
- **Only the maximum is needed**, the full matrix doesn't have to be stored / computed
- **Anti-diagonals** of the matrix can be **computed in parallel**
- If a matrix element can never be the maximum, it doesn't have to be accurately computed

$$M[i, j] = \max \begin{cases} 0 \\ M[i-1, j-1] + S(A[j], B[i]) \\ M[i-1, j] - 1 \\ M[i, j-1] - 1 \end{cases}$$

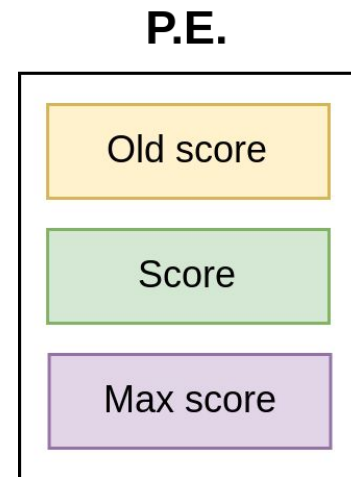
Sequence A

	A	T	G	T	C
Sequence B A	0	1	2	3	4
C	0	1	2	3	4
G	0	1	2	3	4
T	0	1	2	3	4
C	0	1	2	3	4



Linear systolic array

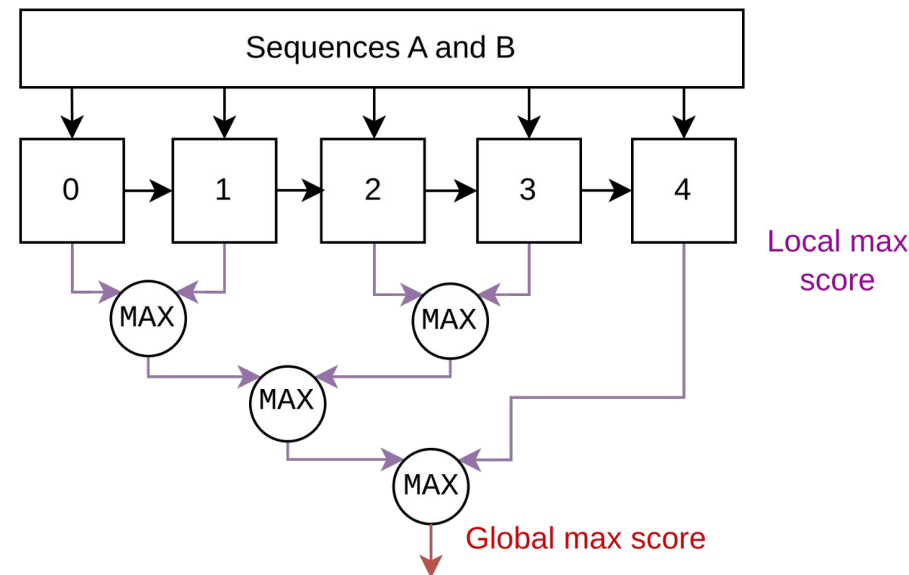
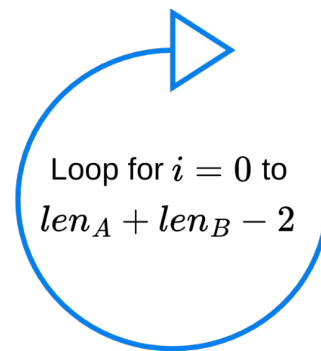
- The **systolic array** computes a **full anti-diagonal** in **1 cycle**
- Each P.E. contains **current score**, **old score** and **local max score** in that P.E (to avoid dependencies)
- After the whole matrix has been processed, the local P.E. maximums are aggregated in a **max reduction tree**
- In HLS, describe from right to left to not overwrite. Used UNROLL and PIPELINE



Sequence A

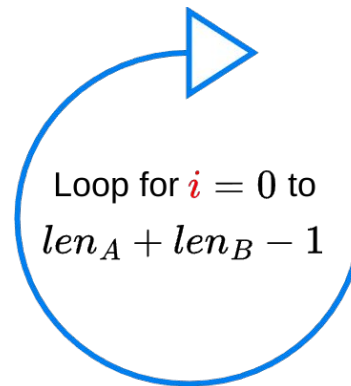
	A	T	G	T	C
A	0	1	2	3	4
C	0	1	2	3	4
G	0	1	2	3	4
T	0	1	2	3	4
C	0	1	2	3	4

Sequence B



Systolic array shift-register

- Naive systolic-array implementation requires iteration-dependent indexing of sequence B: complex hardware (many LUTs)
- Instead, shift the sequence B every iteration and access it at a fixed index (just wires)



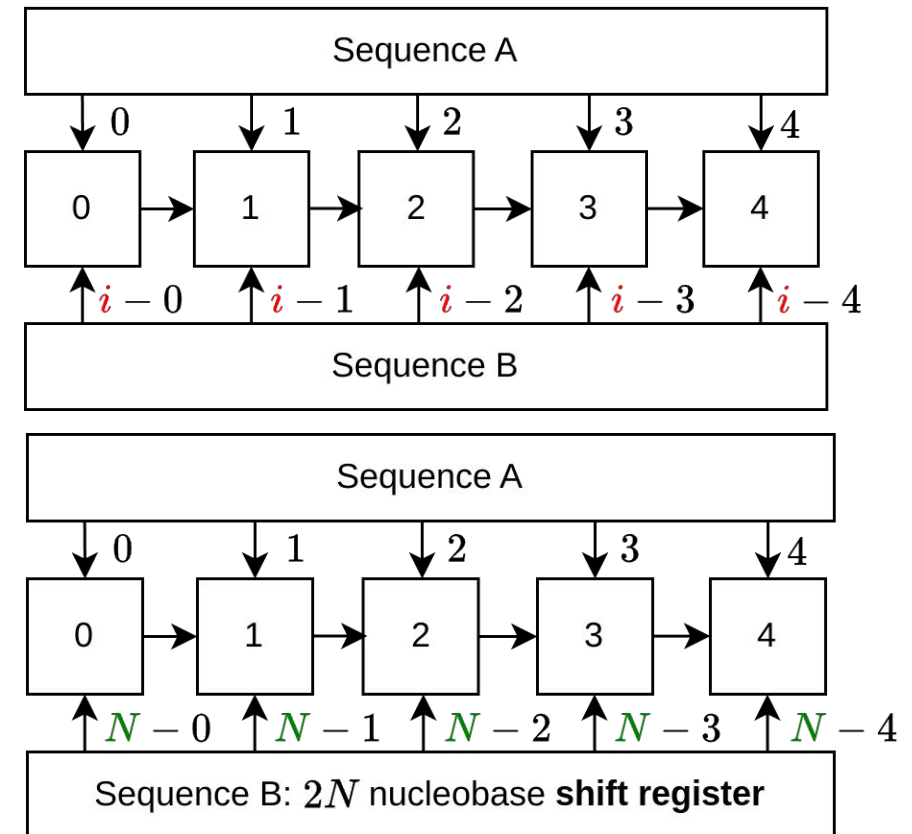
$$0 < len_A \leq N$$

$$0 < len_B \leq N$$

Sequence A

	A	T	G	T	C
A	0	1	2	3	4
C	0	1	2	3	4
G	0	1	2	3	4
T	0	1	2	3	4
C	0	1	2	3	4

Sequence B



Systolic array optimizations

- **Compute out-of-bounds:** avoid ifs to use less LUTs. Out of-bounds elements can never be maximums (can only add 1 in-bounds)
- **5-bit scores:** don't need to use 8-bit scores.
 - Handle *perfect* score of 32 separately
 - Use unsigned integer for scores (handle wrapping)
 - Significantly reduce resource usage (max-reduction)
- **Clock frequency**
- **Manually unroll** iteration 0: less complex HW, remove ifs inside loop.
- **Possible improvements:**
 - Compute several anti-diagonals in parallel through Recursive Variable Expansion
 - Use state-of-the-art algorithm to achieve **higher occupation of systolic array** (up to 3x performance improvement) [1] with open-source Vivado HLS implementations [2].

[1] E. Houtgast, V. -M. Sima and Z. Al-Ars, "High Performance Streaming Smith-Waterman Implementation with Implicit Synchronization on Intel FPGA using OpenCL," 2017 IEEE 17th International Conference on Bioinformatics and Bioengineering

[2] <https://gitlab.com/kingma1999/fpga-sw>

SW, Interrupts & Linux driver

- **SW only compresses** nucleobases on file load, **calls HW** and **retrieves results** (to make use of systolic array)
- **Coherence doesn't really matter:** The SW does no work on the HW result, bandwidth is not the bottleneck, and timing does not consider reading the accelerator's result. Used default *non-cacheable DMA buffers*
- Implemented an **interrupt-based Linux driver**
- **0% CPU** usage
- Returned number of performed comparisons returned to user through `raw_copy_to_user`

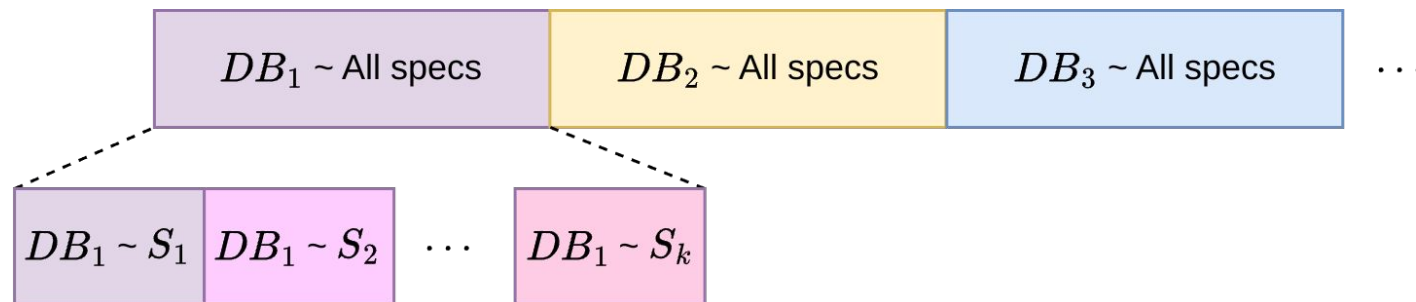
```
raw_copy_to_user(  
    (void*)message.numComparisonsPtr,  
    &numComparisons,  
    sizeof(uint32_t)  
);
```

Multi-worker acceleration

2

Parallelization

- Instantiate **several workers in parallel**
 - Compute several scores in parallel
 - Communicate through **AXI-Stream** (`hls::stream`)
- One **single Vitis IP-core** (instead of multiple accelerators)
 - Better resource usage: don't duplicate shared control, MAXI...
 - Easier interaction with SW: single interrupt, no scheduling
- Need to **choose dimension to parallelize**
 - Specimens or DBs. High performance impact
- Still need to maintain output order: **synchronization**



1. Across comparisons

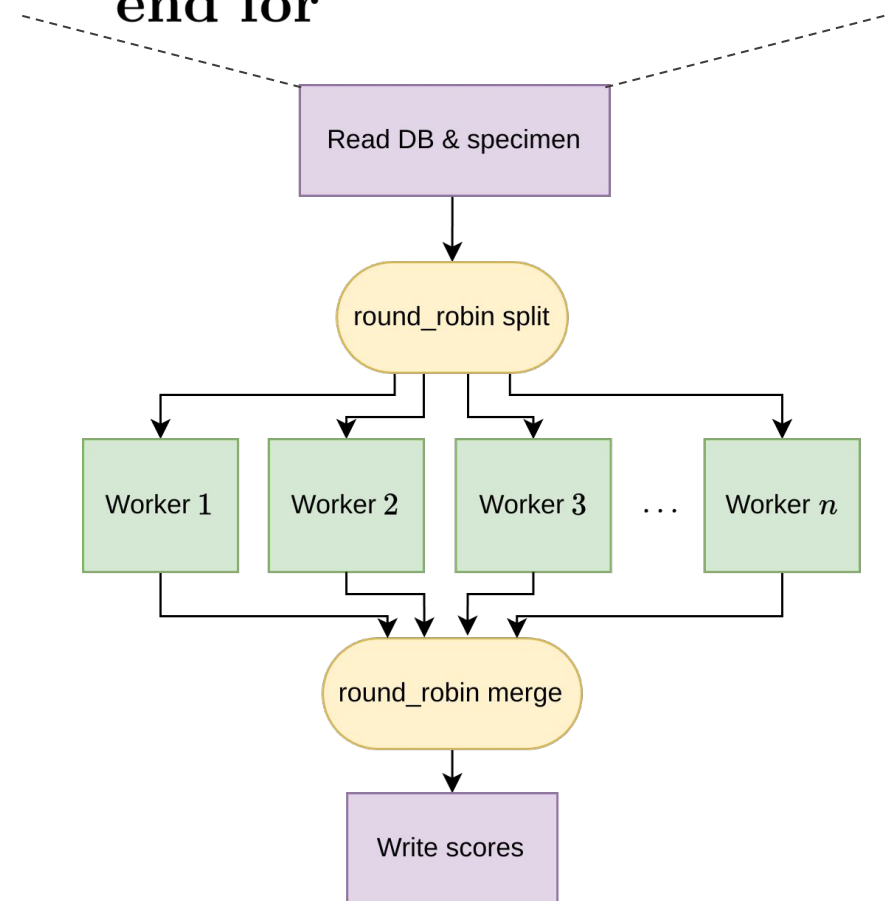
- **Parallelization across every comparison**
 - Each worker gets a single comparison (both DB and specimen) in round_robin fashion.
 - Reader task generates pair of sequences
 - Output already in correct order
- **Synchronization** handled using `hls::task`, `hls::stream`, `hls::split` and `hls::merge`
- **Bottleneck in the reader:** needs to generate every single comparison arguments.
 - Increasing the number of parallel accelerators has little effect on performance.
- Tried changing `hls::split` & `hls::merge` FIFO sizes, and using `load_balance` instead of `round_robin`, but the performance didn't improve.

Reader code

```

for  $seq_{DB} \in DB_{DRAM}$  do
  for  $seq_{spec} \in Specs_{cached}$  do
     $out \leftarrow (seq_{DB}, seq_{spec})$ 
  end for
end for

```

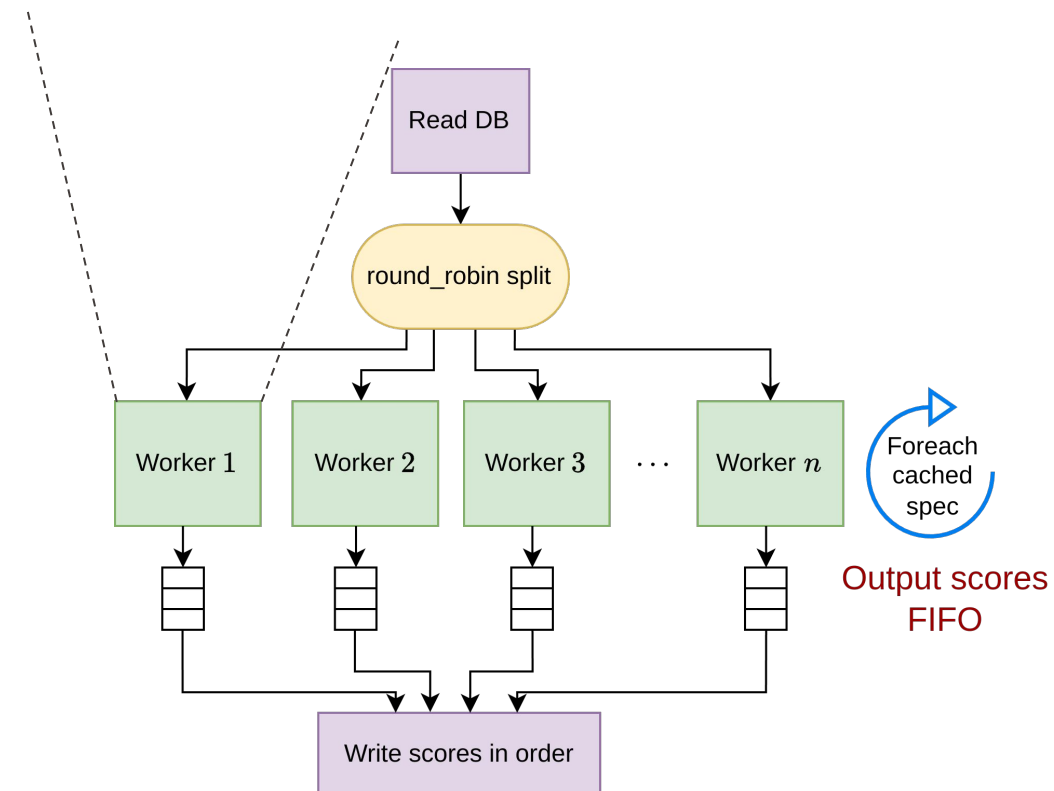


2. Across DB sequences

- **Parallelization across DB sequences**
 - Reader only distributes DB sequence
 - Each worker iterates through all cached specimens to compare with DB
- **Synchronization** handled manually using `hls::stream`
- Need to **replicate cached specimens**. With dual-ported BRAMs, need $\lceil n/2 \rceil$ copies. Manually replicated with bidimensional arrays and `ARRAY_PARTITION`
- The workers **compute 1000 scores at once** (stored in a FIFO), and the writer task outputs them in order, worker by worker.
- **Bottleneck:** All workers have to wait for the slowest one.
- Best result: 1.983 s | CR = 29.98 %
- Limited by number of LUTs (~48 k)

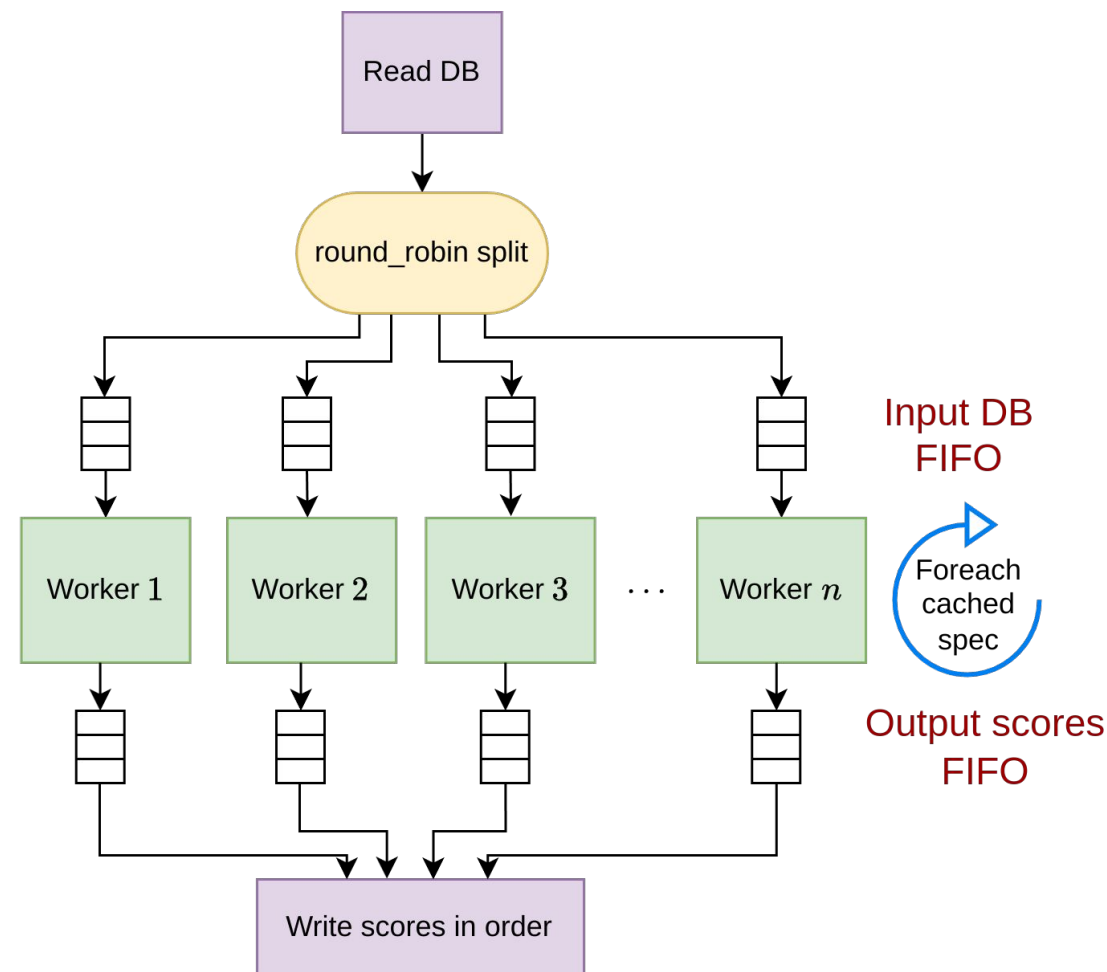
Worker code

```
seqDB ← in.dequeue()
for seqspec ∈ Specscached do
    out.enqueue(score(seqDB, seqspec))
end for
```



3. Decoupling accelerators

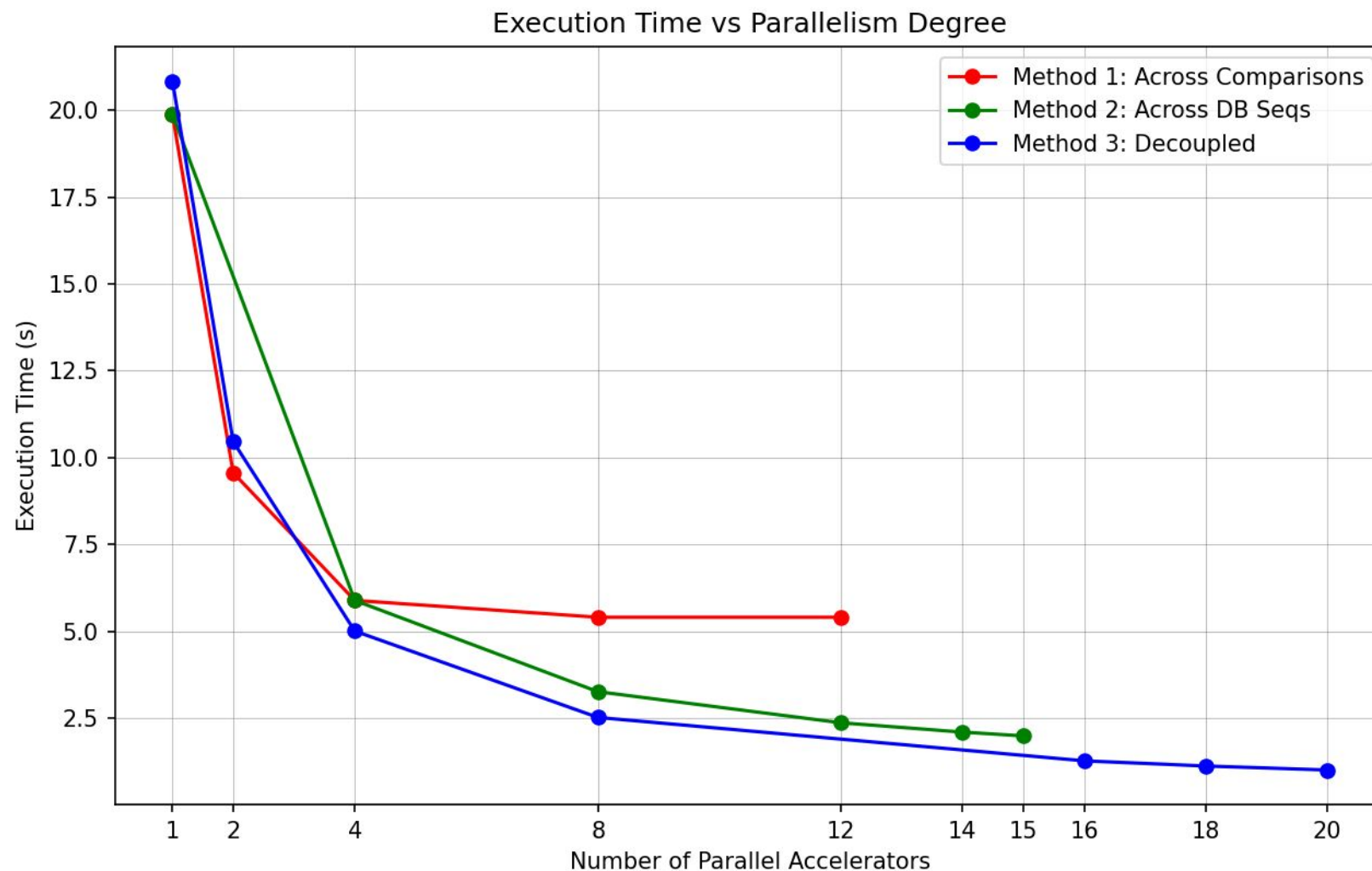
- Still **parallelize across DB sequences**
 - Same conceptual architecture as accelerator 2
- Introduce a **FIFO buffer** between reader and workers
 - If big enough, can hide the difference in sequence lengths
 - Less expensive than reorder buffers
- Achieves almost **linear speed-up** with increasing number of workers.
- Best result: 1.000 s | CR = 43.73 %
- Limited by number of LUTs (~48 k)



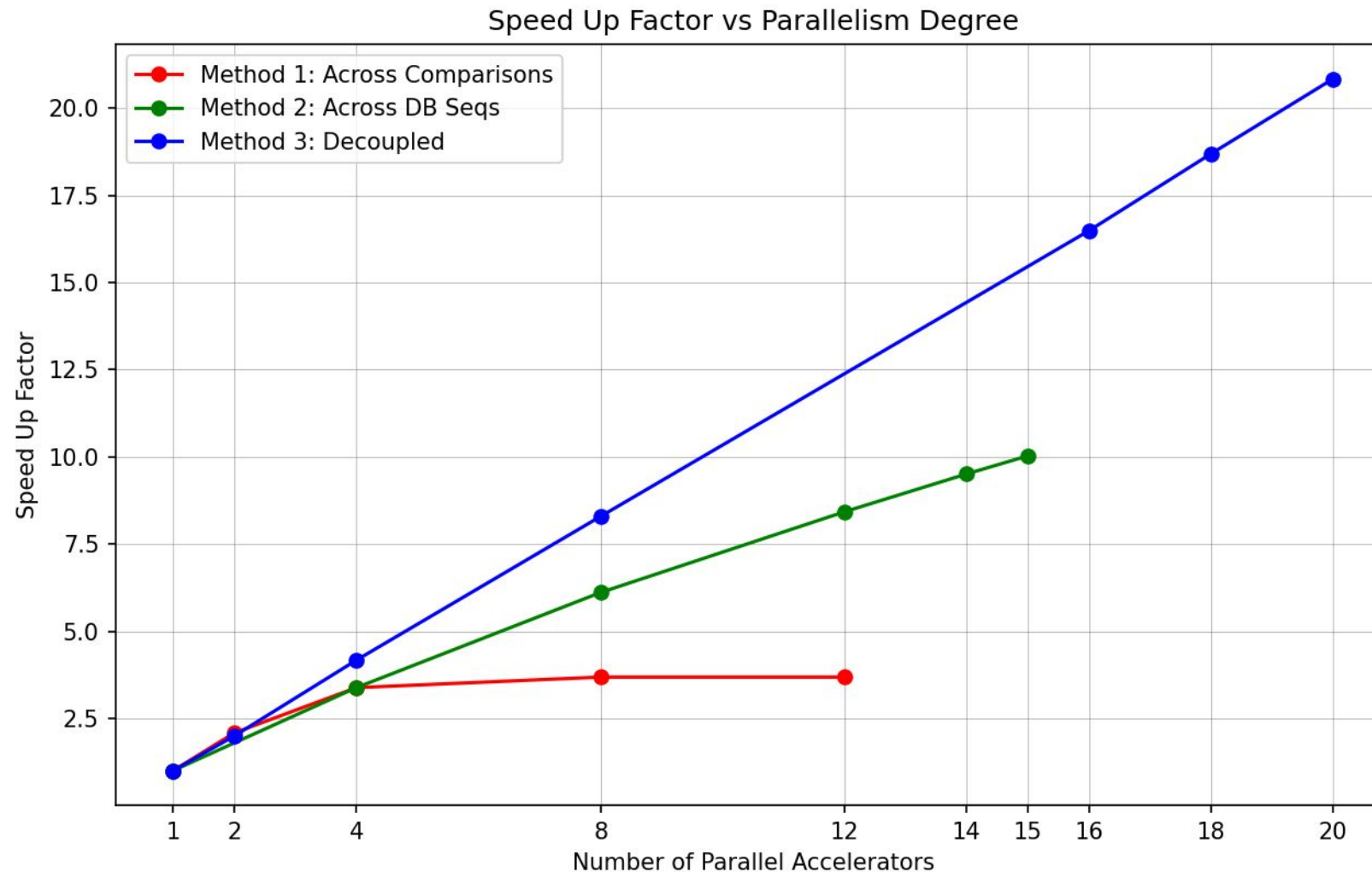
Results & Conclusion

3

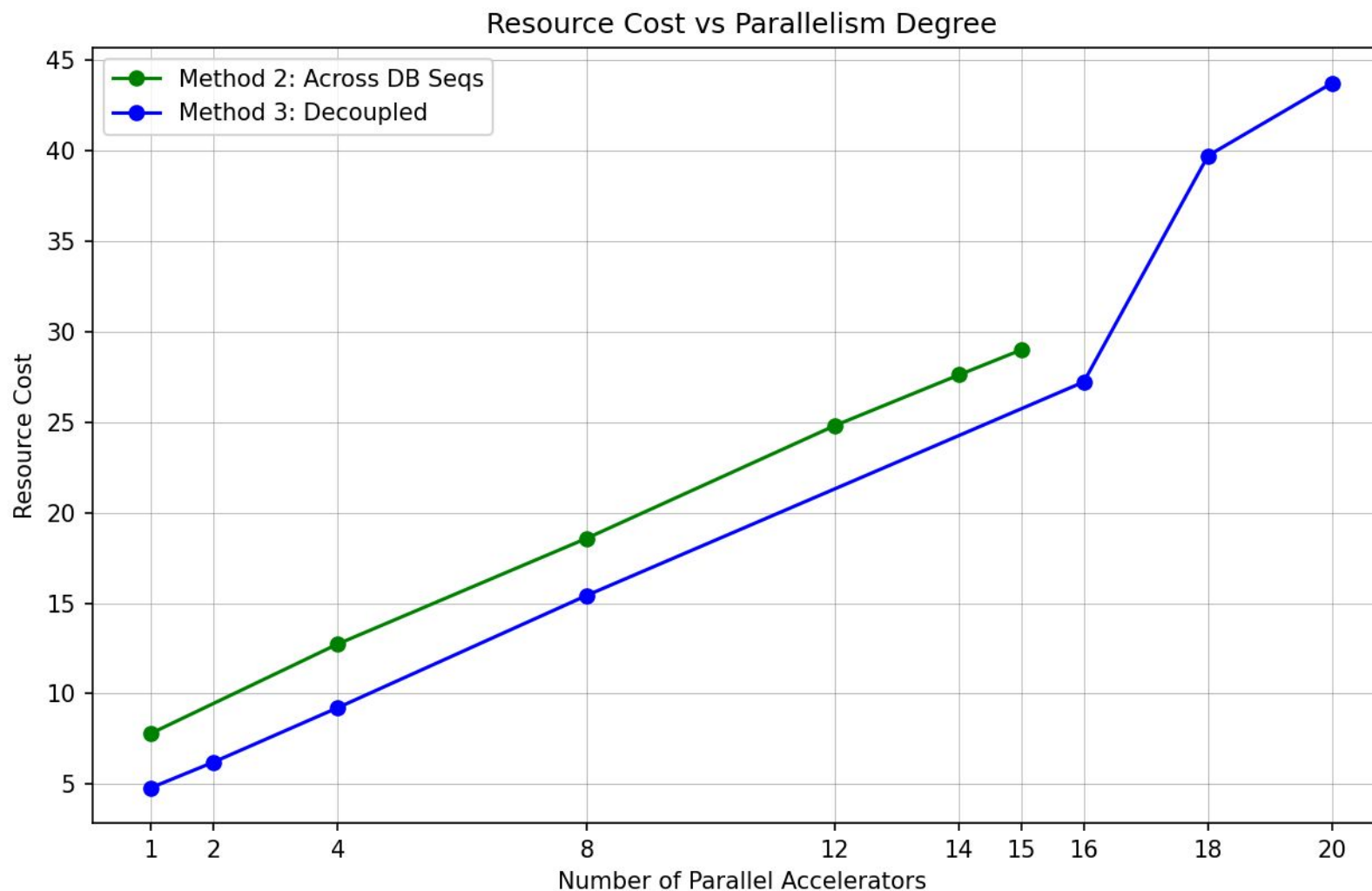
3.1. Execution Time vs Parallelism Degree



3.2. Speed Up Factor vs Parallelism Degree



3.3. Resource Cost vs Parallelism Degree



3.4. Performance summary of the implemented solutions

name	num parallel	time	cpu
02_SeqMatcher_HW_Phase2_Parallel_2	2	9.562	0
03_SeqMatcher_HW_Phase2_Parallel_4	4	5.886	0
03_SeqMatcher_HW_Phase2_Parallel_8	8	5.401	0
03_SeqMatcher_HW_Phase2_Parallel_12	12	5.401	0

name	resources	num parallel	time	cpu	brams	dsps	luts	ffs
02_SeqMatcher_HW_Phase1	7.809	1	19.892	0	19	3	8708	8239
02_SeqMatcher_HW_Phase1_Parallel_4	12.738	4	5.885	0	23	6	17093	13200
02_SeqMatcher_HW_Phase1_Parallel_8	18.569	8	3.256	0	23	6	28739	20930
02_SeqMatcher_HW_Phase1_Parallel_12	24.806	12	2.360	0	27	9	40025	27044
02_SeqMatcher_HW_Phase1_Parallel_15	28.989	15	1.983	0	28.5	9	47959	32291

name	resources	num parallel	time	cpu	brams	dsps	luts	ffs
03_SeqMatcher_HW_Phase2_DB_parallelism_1_HIGHER_CLK	4.807	1	17.881	0	6	6	5805	6502
03_SeqMatcher_HW_Phase2_DB_parallelism_2_HIGHER_CLK	6.212	2	8.972	0	7.5	6	8013	8423
03_SeqMatcher_HW_Phase2_DB_parallelism_4_HIGHER_CLK	9.221	4	4.503	0	13	6	12077	12120
03_SeqMatcher_HW_Phase2_DB_parallelism_8_HIGHER_CLK	15.414	8	2.263	0	24	6	20687	19196
03_SeqMatcher_HW_Phase2_DB_parallelism_16_HIGHER_CLK	27.213	16	1.138	0	46	6	36341	34231
03_SeqMatcher_HW_Phase2_DB_parallelism_18	39.707	18	1.115	0	60.5	78	42845	41849
03_SeqMatcher_HW_Phase2_DB_parallelism_20	44.216	20	1.000	0	67	86	47058	45996

An aerial photograph of Lake Geneva in Switzerland. In the foreground, the EPFL campus is visible, featuring a large, modern building with a distinctive circular design. The lake extends to the horizon, with snow-capped mountains in the background under a clear sky.

Thank you!

Alejandro López, Pedro Palacios
EE390(a) Lab on HW/SW digital systems codesign