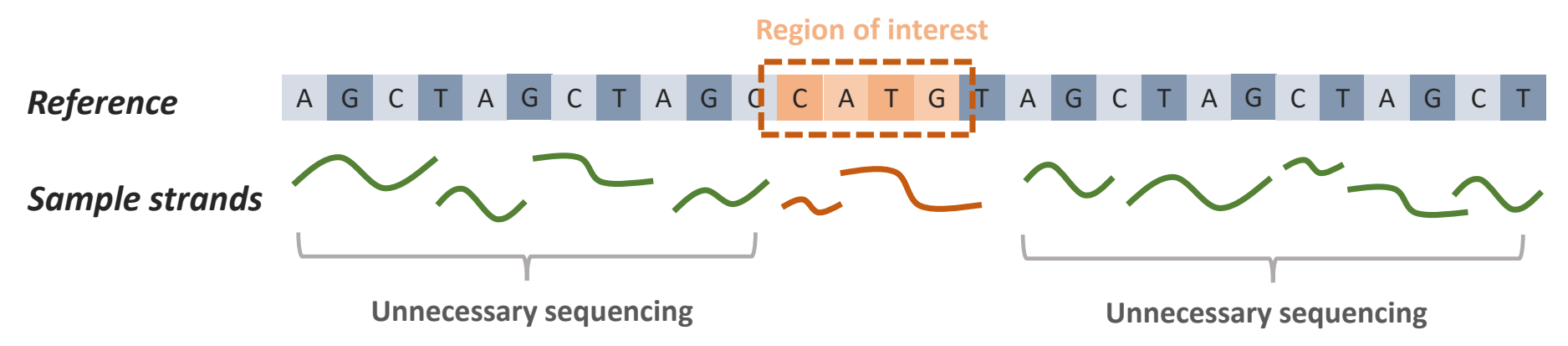
Hardware Accelerated Real-time Selective Genome Sequencing

Po Jui Shih **Supervisor: Sri Parameswaran** **Assessor: Hui Guo**
Co-supervisors: Hasindu Gamaarachchi, Hassaan Saadat

Introduction

Selective sequencing with nanopore technology enables efficient targeted genome analysis

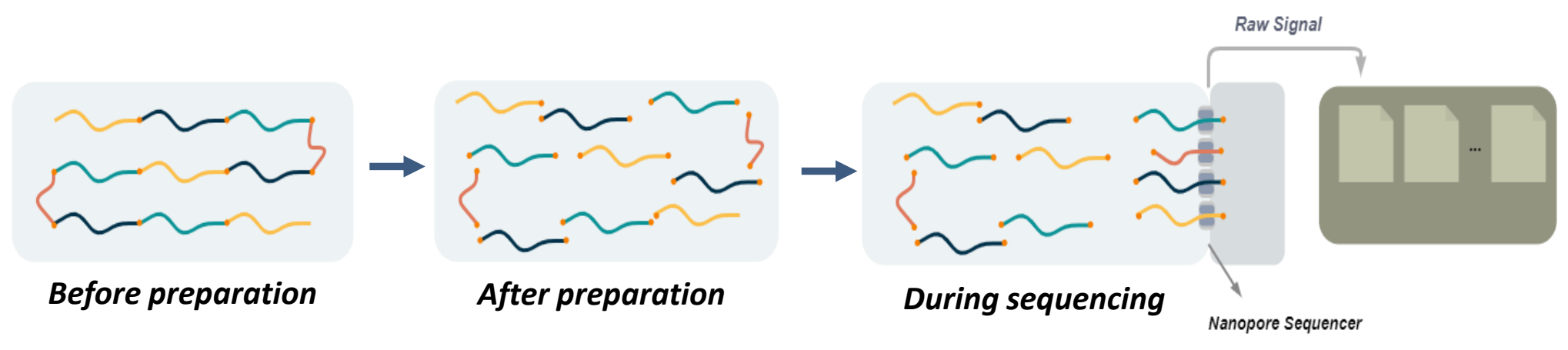


→ Fully ***sequence*** strands within regions of interest; ***reject*** others

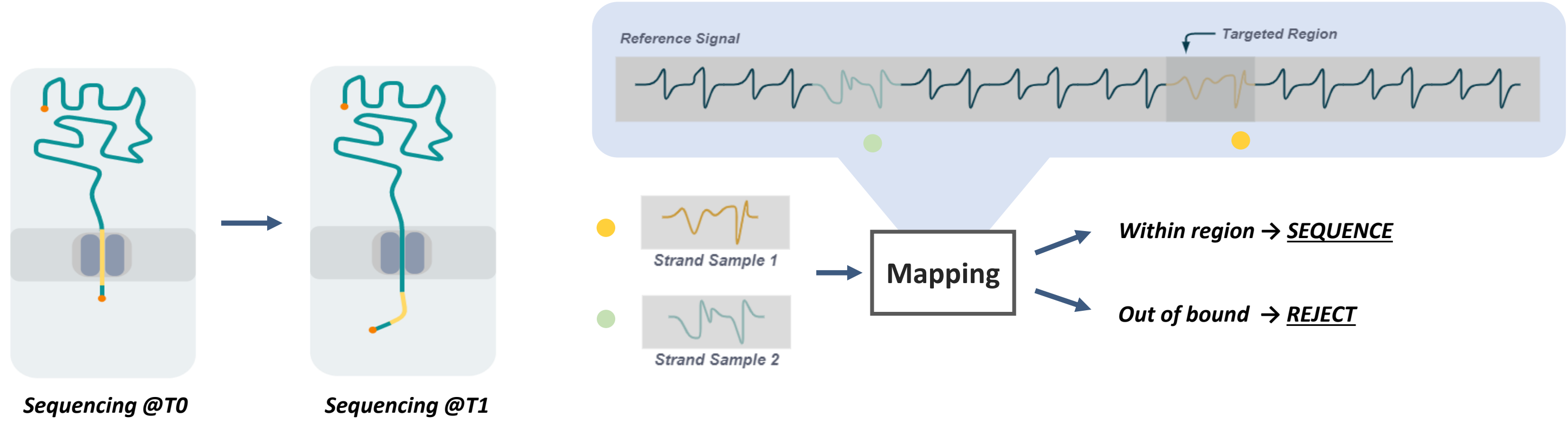
"Needle in a haystack"
e.g. [regions of interest / others]
→ [pathogen / host]
→ [cancer / nontumor]

Nanopore Sequencing

- Simple sample preparation (minimal priori knowledge)
- Real-time data output → real-time analysis
- Able to reject strands at individual nanopore channels



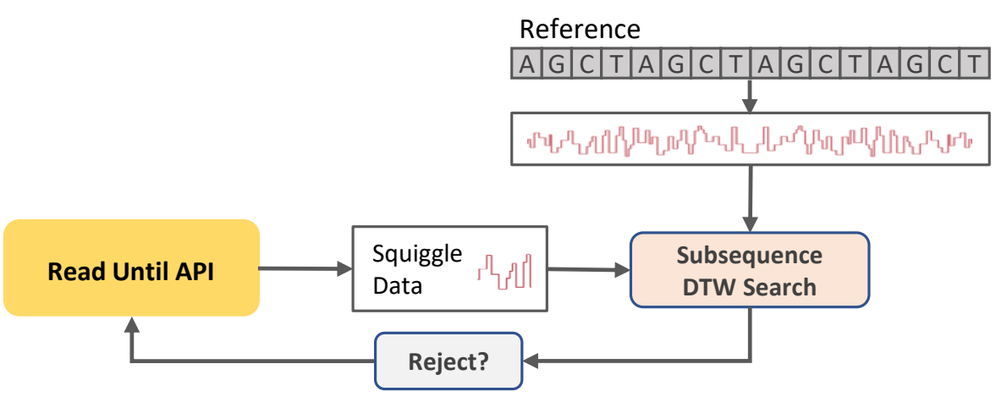
Read Until



Read Until Implementations

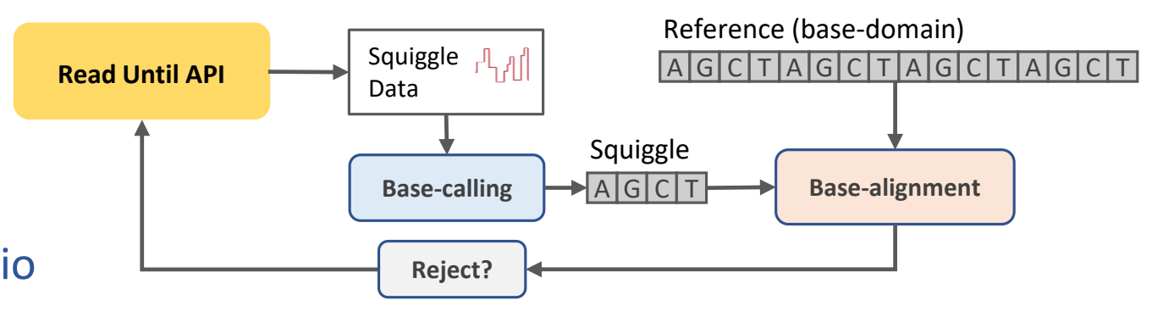
1. Squiggle-domain sequence matching
   - First Read Until implementation (RUscripts [1])
   - Uses **Dynamic Time Warping (*DTW*)** to map sequences

   → Needed 22-core server to keep up with slower sequencing rate, deprecated after sequencing rate ↑



2. Base-domain sequence matching
   - **Base-calls** the squiggle and uses **base-alignment** to map [2]
   - Able to scale to giga-base references

   → Requires high-end GPU to perform real-time base-calling (excessive), loses portability, and has high performance-watt ratio



HARU: The proposed Read Until implementation

- First FPGA accelerated Read Until implementation
- Software-hardware co-design targeting **low-cost** MPSoCs
- Extends the MinION sequencer's **portable** nature
- Low performance requirement for host machine

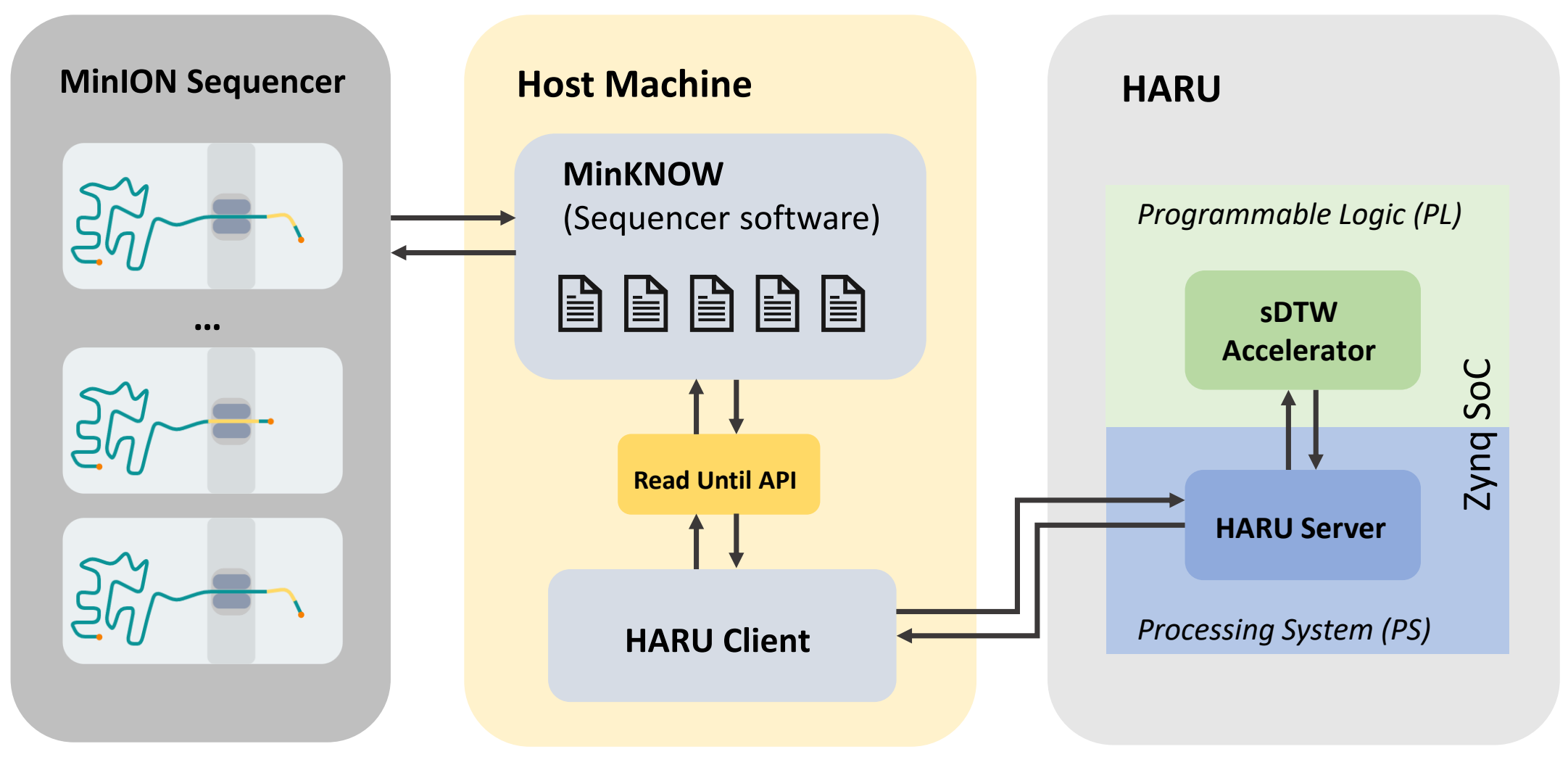
Selective sequencing with MinION + HARU

Contributions

- Minimal requirements for performing targeted small-genome analysis
- Demonstrates the use of High-Level-Synthesis (HLS) for DNA sequencing and analysis acceleration
- Provides an extendible framework for Read Until

HARU: Hardware Accelerated Read Until

Read Until with HARU (MinION + HARU)



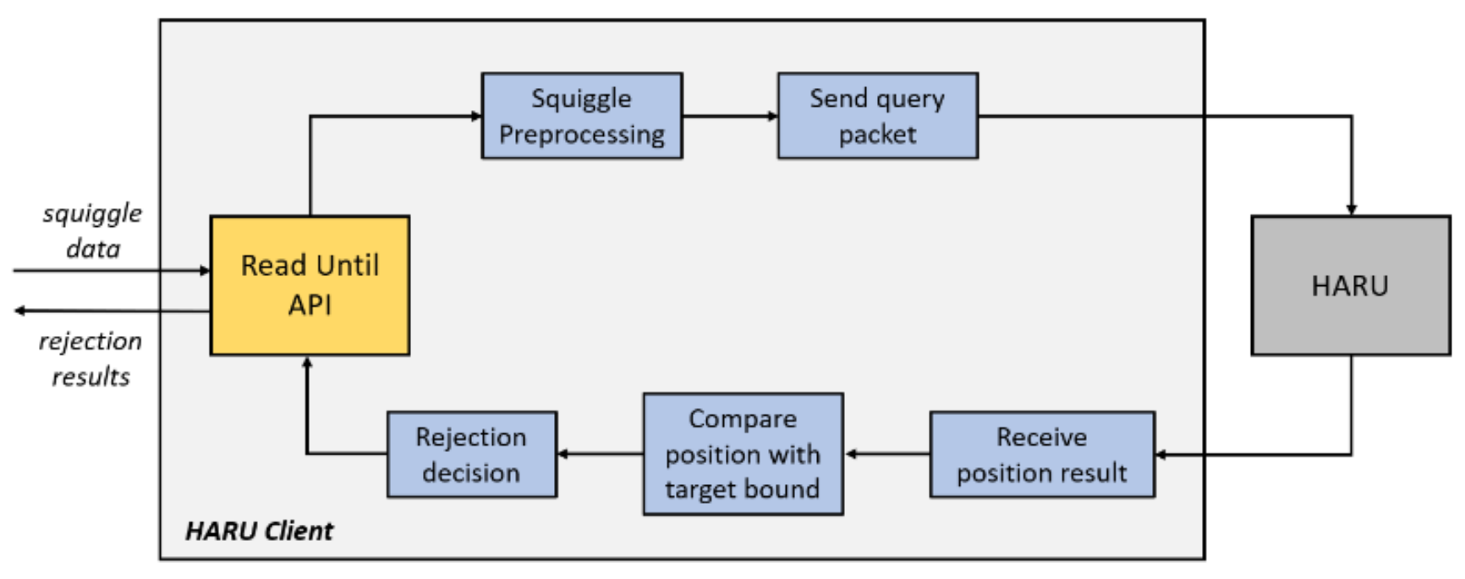
HARU Client

Sequencer → HARU

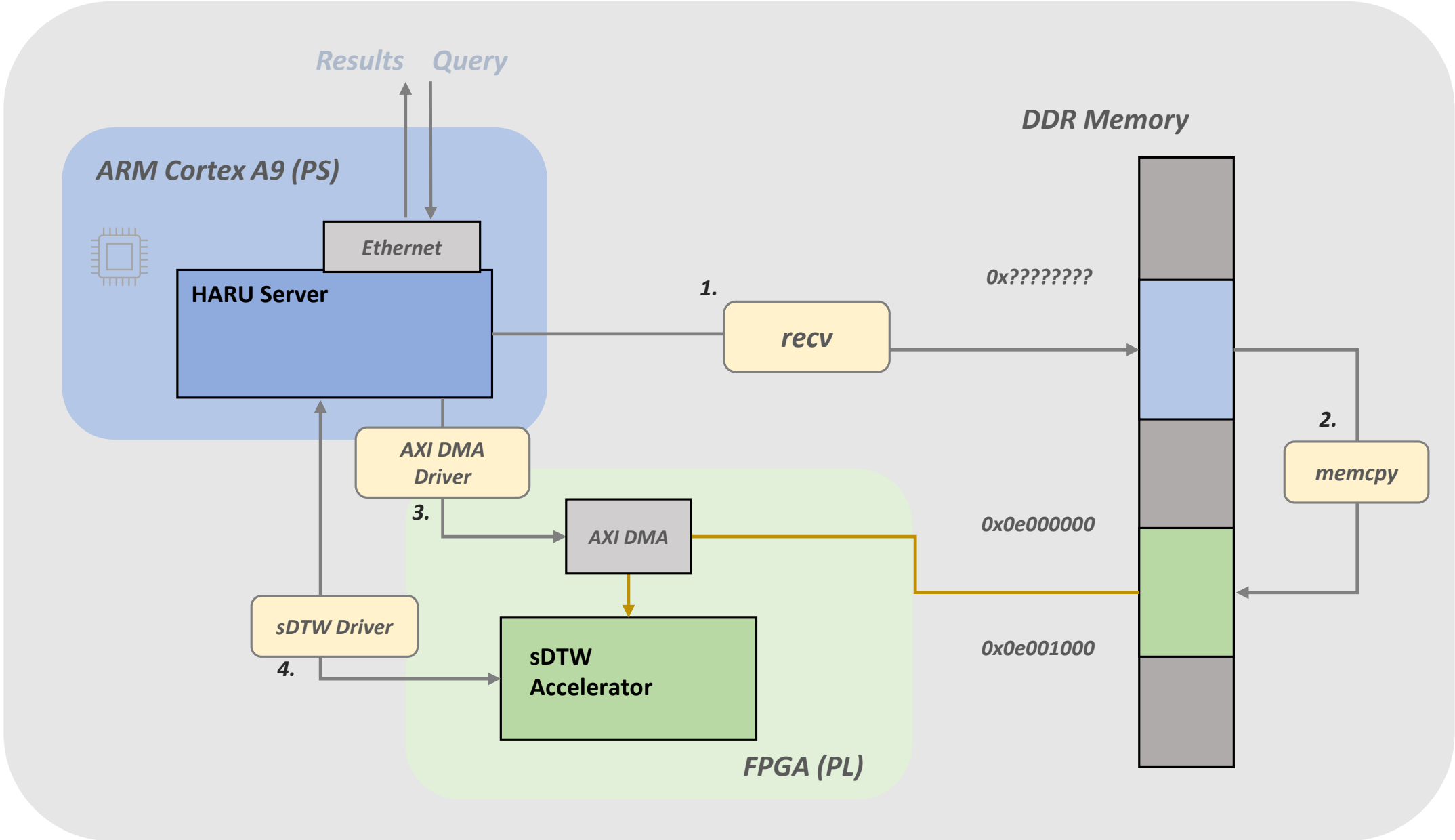
- Collects real-time squiggle data via the Read Until API
- Pre-processes raw data
- Sends data to HARU via Ethernet

HARU → Sequencer

- Receives sequence mapping results from HARU via Ethernet
- Determines whether the position of strand is within a region of interest
- Sends back rejection to sequencer software if not a necessary strand

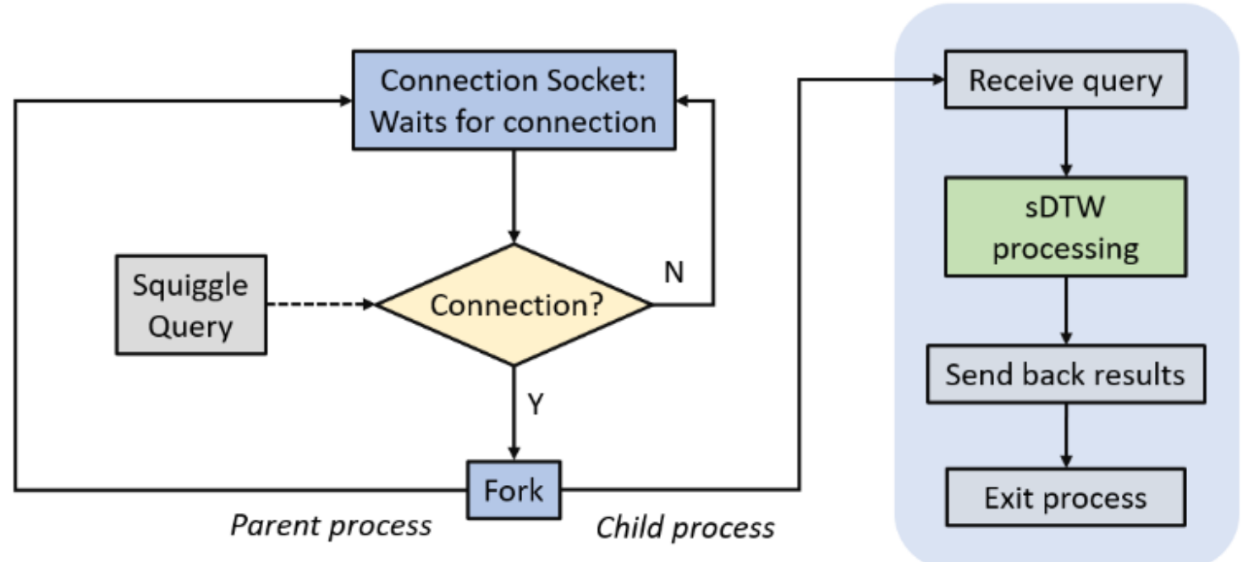


HARU Overview



HARU Server

- Server application running on a custom PetaLinux generated embedded Linux OS on the processing system of the Zynq MPSoC
- Responsible for query request handling
- Sends query over to accelerator via AXI stream (HP AXI)
- Controls the custom sDTW accelerator through custom drivers
- Sends results back to client via Ethernet using the same socket



Documented AXIs throughput:
- MM2S = 399.04 MB/s
- S2MM = 298.59 MB/s

Benchmarked AXIs throughput:
- 333.16 MB/s

sDTW accelerator driver
- HLS generated
- Accessed through UIO

`/dev/class/uio/uio0`

AXI DMA driver
- HARU specific AXI DMA driver
- Accessed through physical address space

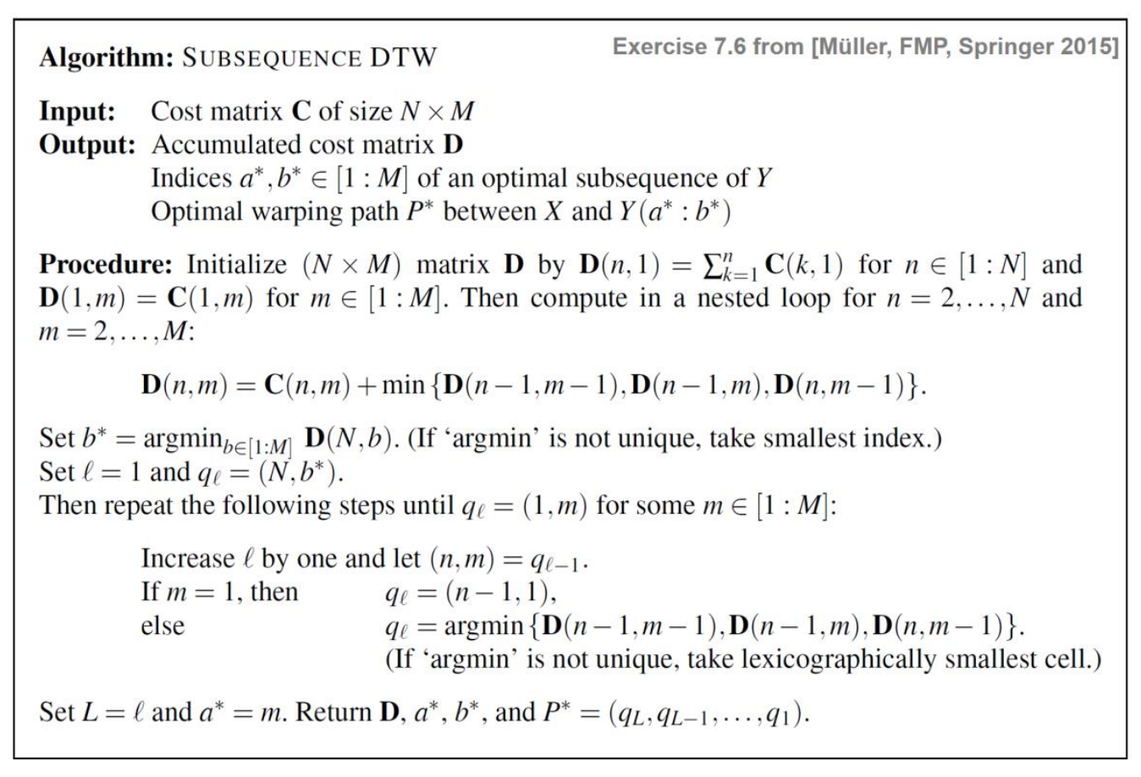
`/dev/mem`

Subsequence DTW Accelerator

Original subsequence DTW algorithm [3]:

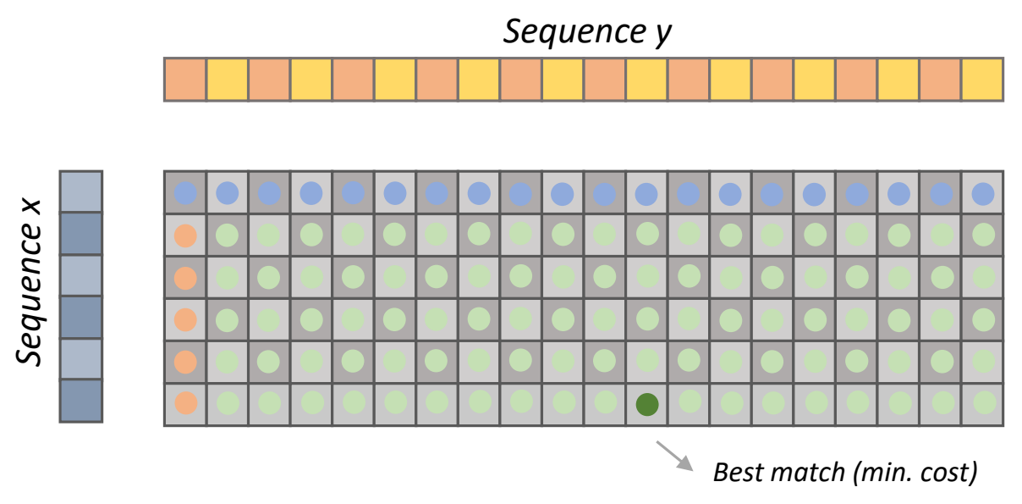
Given two sequences **X**, **Y**
- **X** := (x_1, x_2, ..., x_M) of length **M** ∈ N
- **Y** := (y_1, y_2, ...y_N) of length **N** ∈ N

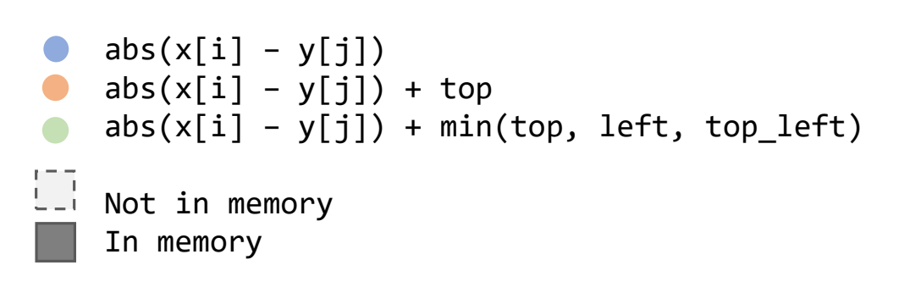
and cost matrix **C** ∈ $R^{M×N}$
- **C**(m, n) := $|x_m\text{-}y_n|$



Exercise 7.6 from [Müller, FMP, Springer 2015]

```
Algorithm: SUBSEQUENCE DTW
Input:  Cost matrix C of size N × M
Output: Accumulated cost matrix D
        Indices a*, b* ∈ [1 : M] of an optimal subsequence of Y
        Optimal warping path P* between X and Y(a* : b*)

Procedure: Initialize (N × M) matrix D by $D(n,1) = \sum_{k=1}^{n} C(k,1)$ for $n \in [1:N]$ and
$D(1,m) = C(1,m)$ for $m \in [1:M]$. Then compute in a nested loop for $n = 2,\dots,N$ and
$m = 2,\dots,M$:

    $D(n,m) = C(n,m) + \min\{D(n-1,m-1), D(n-1,m), D(n,m-1)\}$.

Set $b^* = \mathrm{argmin}_{b \in [1:M]} D(N,b)$. (If 'argmin' is not unique, take smallest index.)
Set $\ell = 1$ and $q_\ell = (N, b^*)$.
Then repeat the following steps until $q_\ell = (1,m)$ for some $m \in [1:M]$:

    Increase $\ell$ by one and let $(n,m) = q_{\ell-1}$.
    If $m = 1$, then    $q_\ell = (n-1, 1)$,
    else                $q_\ell = \mathrm{argmin}\{D(n-1,m-1), D(n-1,m), D(n,m-1)\}$.
                        (If 'argmin' is not unique, take lexicographically smallest cell.)

Set $L = \ell$ and $a^* = m$. Return D, a*, b*, and $P^* = (q_L, q_{L-1}, \dots, q_1)$.
```



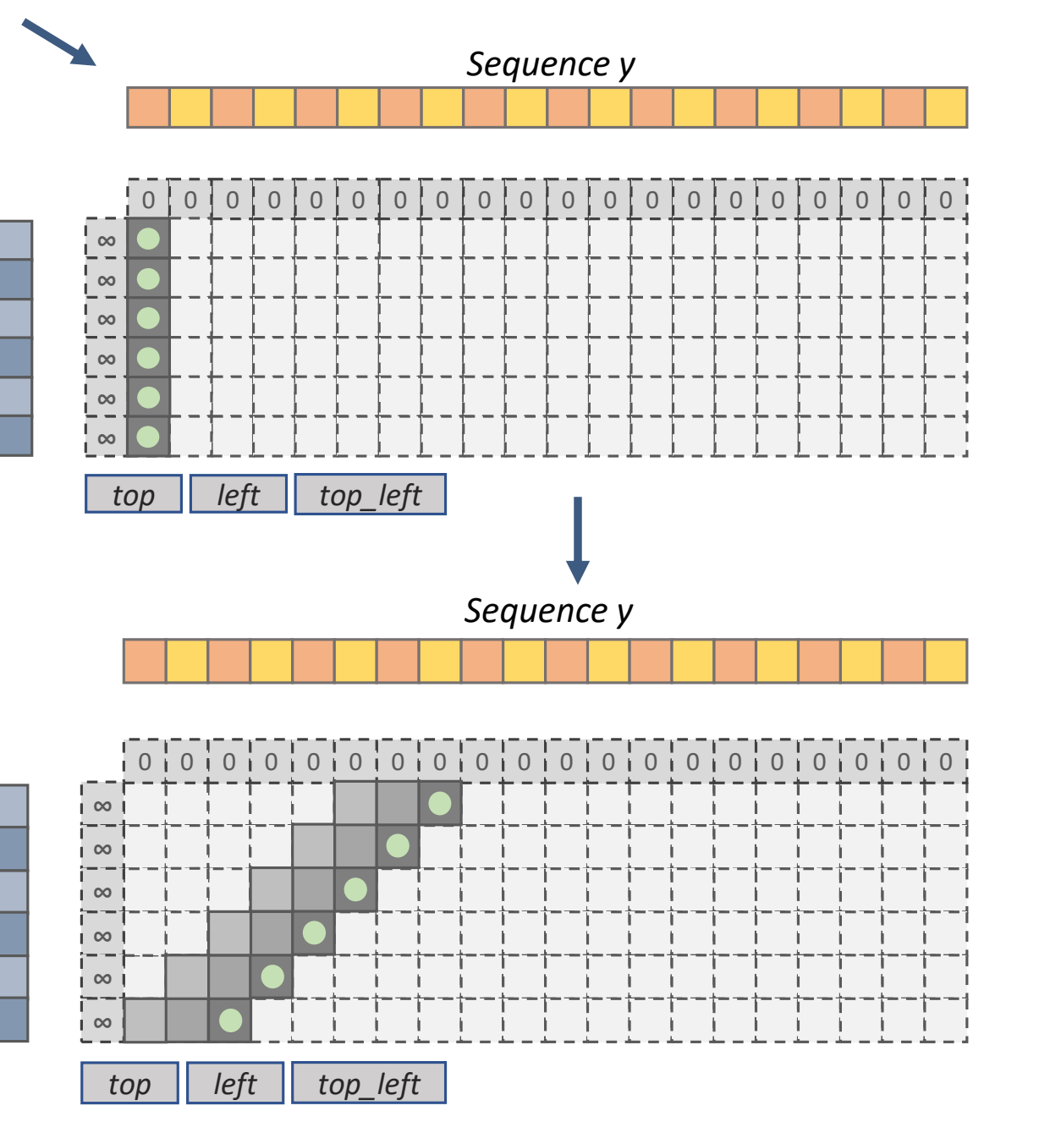


Algorithmic optimisations:
a. Padding for cost matrix
b. Reduce cost matrix to single column

HLS specific optimisations:
a. Pipelining of column computation
b. 16-bit fixed point data type

Resulting accelerator:
- Oblique PE array with size of **M**
- Cost matrix with size of **3M**
- Oblique PE array propagates through the reference sequence

→ Task latency := (M + N −1) × Initiation Interval



Results and Evaluation

Experiment Details and Results

- Accelerator synthesised using Vivado HLS
- Targets the Xilinx Zynq-7020 device (xc7z020clg484-1)
- Tested on the target enrichment application for the bacteriophage lambda DNA
- Single direction has 48,502 bp, giving a full search space of 97,004 bp

Synthesis Results

	Slice LUTs	Slice Register	Slice	BRAM
Available (Zynq-7020)	53,200	106,400	13,300	140
HARU	32,341 (60.79%)	18.899 (17.76%)	9,615 (72.29%)	32.5 (23.21%)

HLS Latency Estimates

	Cycles	Clock Freq.	Estimated Time
Single directional reference search	48875	90 MHz	0.543 ms
Bi-directional reference search (Zynq-7020)	97755	90 MHz	1.086 ms
Unpack Streamed Query	250	90 MHz	2.778us
Overall Subseek DTW	98005	90 MHz	1.089 ms

Comparison with RUscripts

	RUscripts (reference)		HARU (proposed)		
	Laptop Intel i7-8565U	Desktop Intel i9-10850K	HARU system	Network latency	Overall latency
Avg. sDTW task latency	345.75 ms	136.11 ms	1 ms	3.36 ms	4.36 ms

Key results:
- **Core sDTW**: 345.75x faster than Intel i7 Laptop, 136.11x faster than Intel i9 Desktop
- **Overall**: 79.3x faster than RUscripts on Intel i7 Laptop, 31.22x faster than RUscripts on Intel i9 Desktop

→ *Bottleneck is now the network latency (currently unoptimized)*

Evaluation

Substantial speedup at a low hardware cost

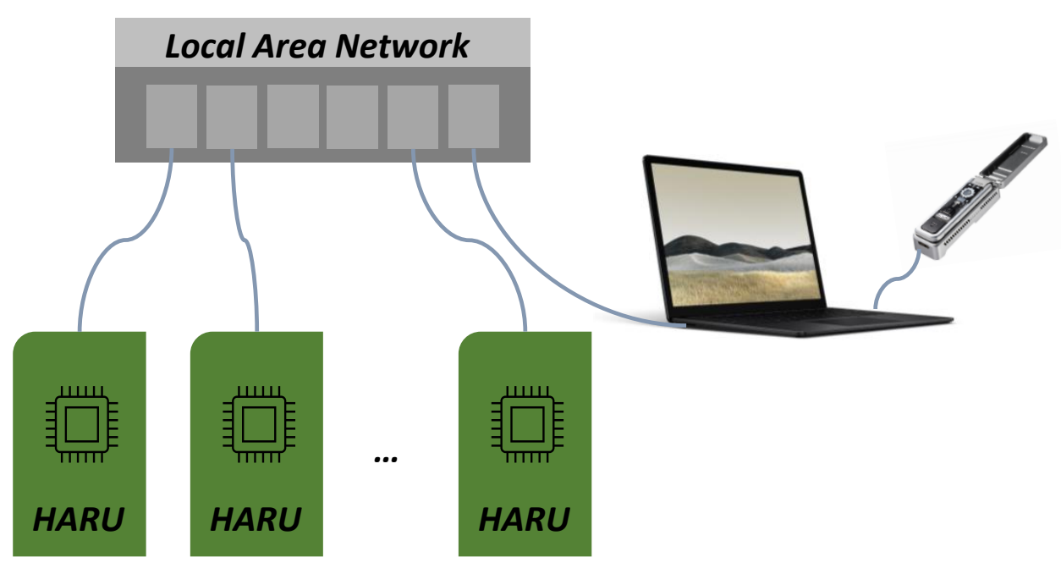
- Subsequence DTW search now linearly dependant to the length of reference sequence
- Cost matrix only requires three times the size of squiggle sequence (subsequence)
- Optimal for smaller genomes (e.g. bacteria, virus)

  → *fast and direct search, can fully store the reference in on-chip memory (no sw-hw transfer overhead)*

Preserves portability while enabling scalability

- Accesses HARU's service through Ethernet
- No harsh requirements for host machine running HARU client
- Scalable by deploying a cluster of MPSoCs running HARU

  → *In-the-field analysis with low hardware requirements*



Provides an extendible low-cost yet high performance-per-watt framework

- HARU demonstrated the use of HLS tools to perform acceleration for DNA sequencing and analysis techniques
- The framework is interchangeable and extendable based on application and algorithmic requirements

References

[1] M. Loose, S. Malla, and M. Stout, "Real time selective sequencing using nanopore technology," BioRxiv, 2016.
[2] A. Payne, N. Holmes, T. Clarke, R. Munro, B. Debebe, and M. W. Loose, "Nanopore adaptive sequencing for mixed samples, whole exome capture and targeted panels.," BioRxiv, 2020.
[3] M. M¨uller, "Dynamic time warping," Information retrieval for music and motion, pp. 69–84, 2007.