UC San Diego
JACOBS SCHOOL OF ENGINEERING

DP-HLS: A High-Level Synthesis Framework for Accelerating Dynamic Programming Algorithms in Bioinformatics

Speakers (in order): Anshu Gupta**, Yingqi Cao*, Jason Liang*

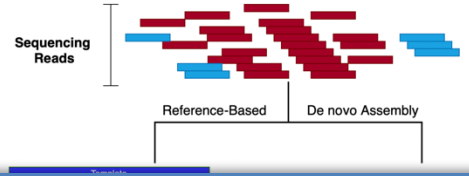
*Department of Electrical and Computer Engineering
**Department of Computer Science and Engineering

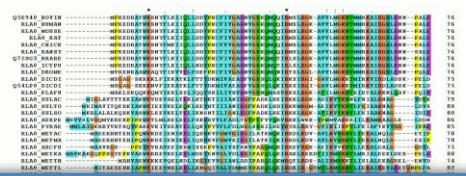
November 2024

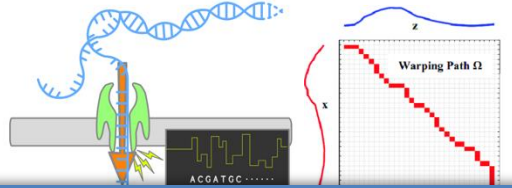
Table of Contents

Table of Contents

Sequence Alignment in Bioinformatics







Dynamic Programming is the underlying algorithm for these applications

Comparative genomics

RNA structure analysis

UC San Diego
JACOBS SCHOOL OF ENGINEERING

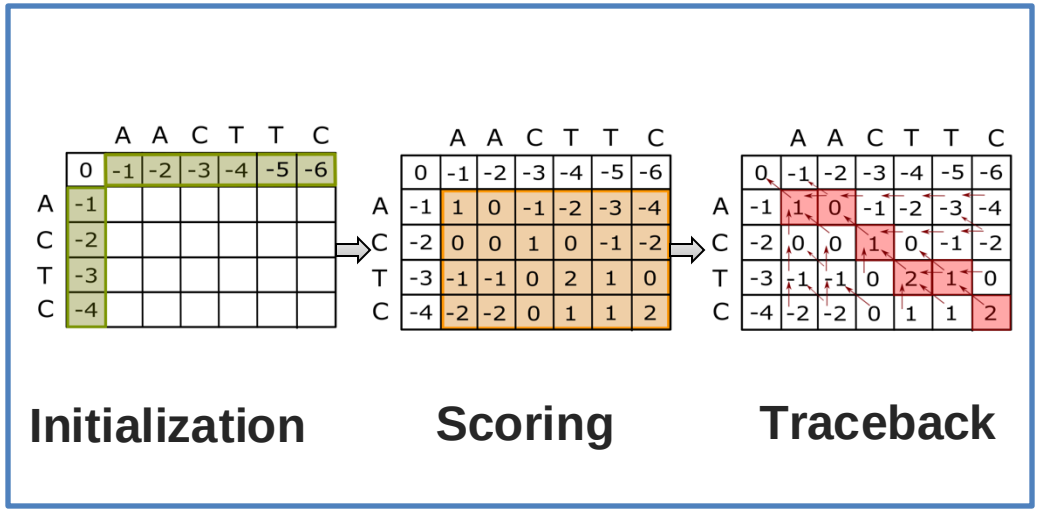
2-D Dynamic Programming Paradigm

- **Problem**: Compare two sequences – ACTC and AACTTC
- **Solution ?**

Input Sequences

Query: ACTC

Reference: AACTTC



Output Traceback Path

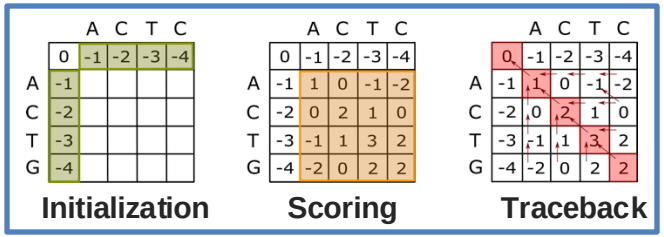
A A C T T C

| | | |.| |

A - C T - C

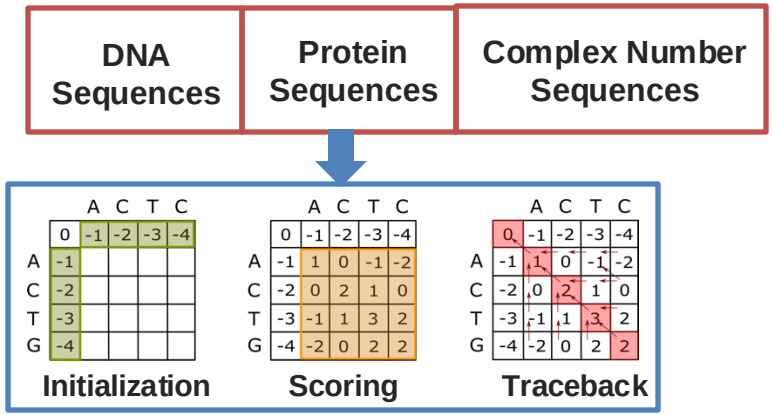
Example: Needleman-Wunsch Algorithm

Variations in 2-D DP Paradigm



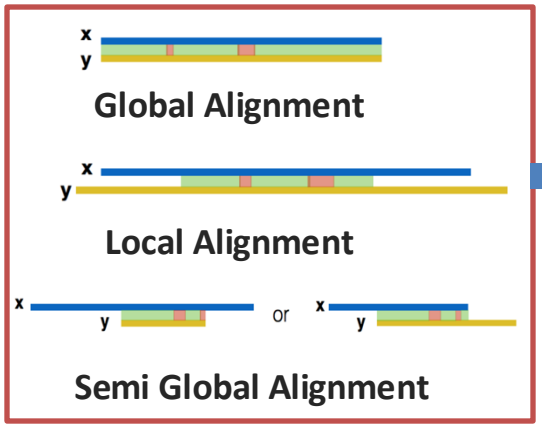
Needleman-Wunsch Algorithm

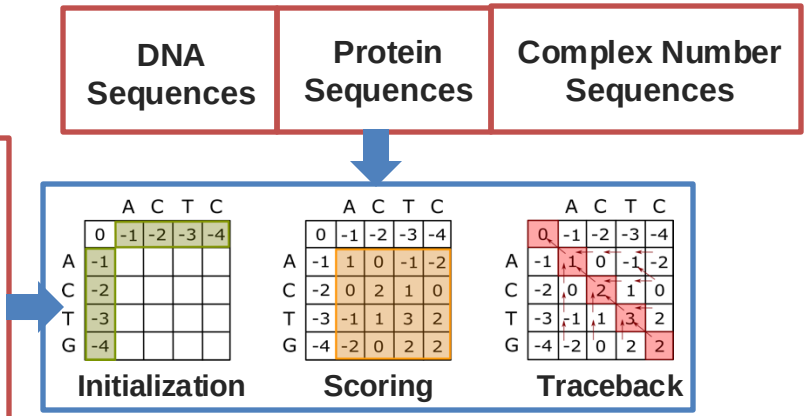
Variations in 2-D DP Paradigm



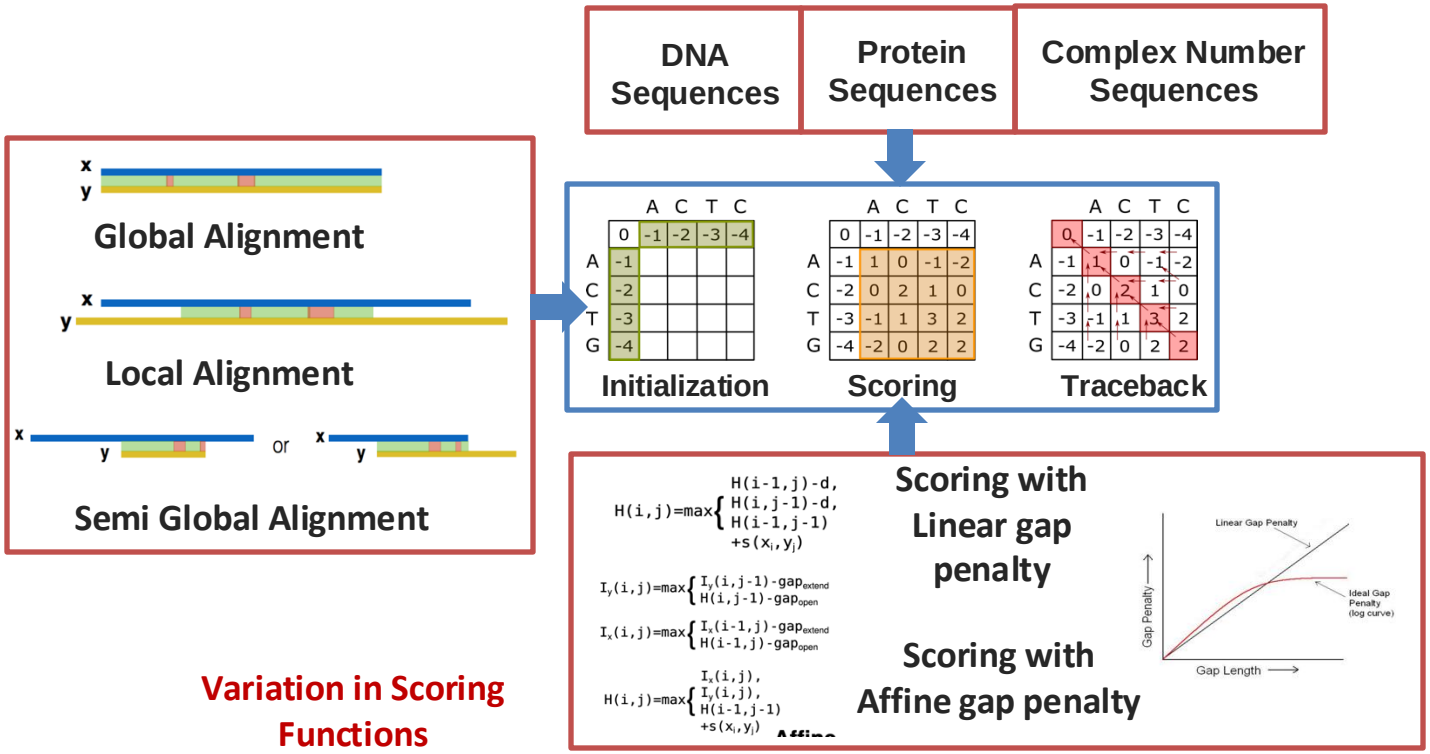
Variations in 2-D DP Paradigm

Variation in Initialization

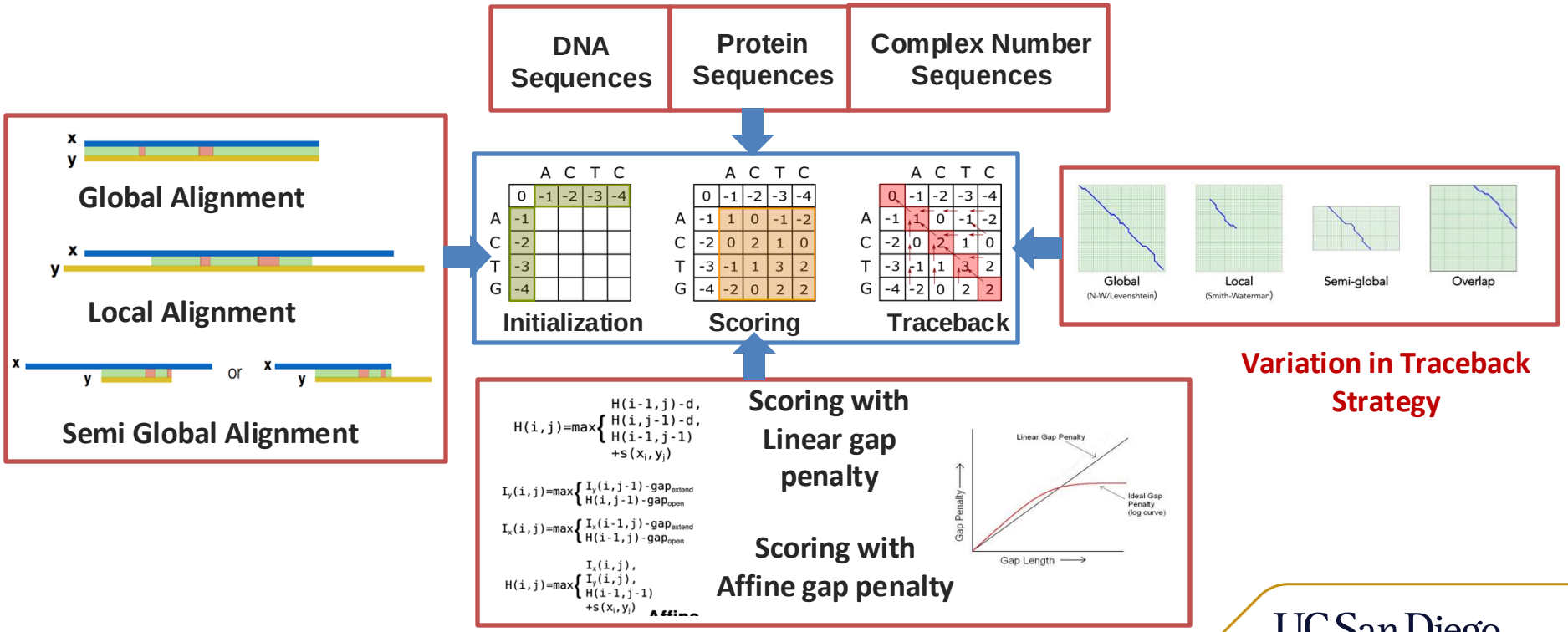




Variations in 2-D DP Paradigm

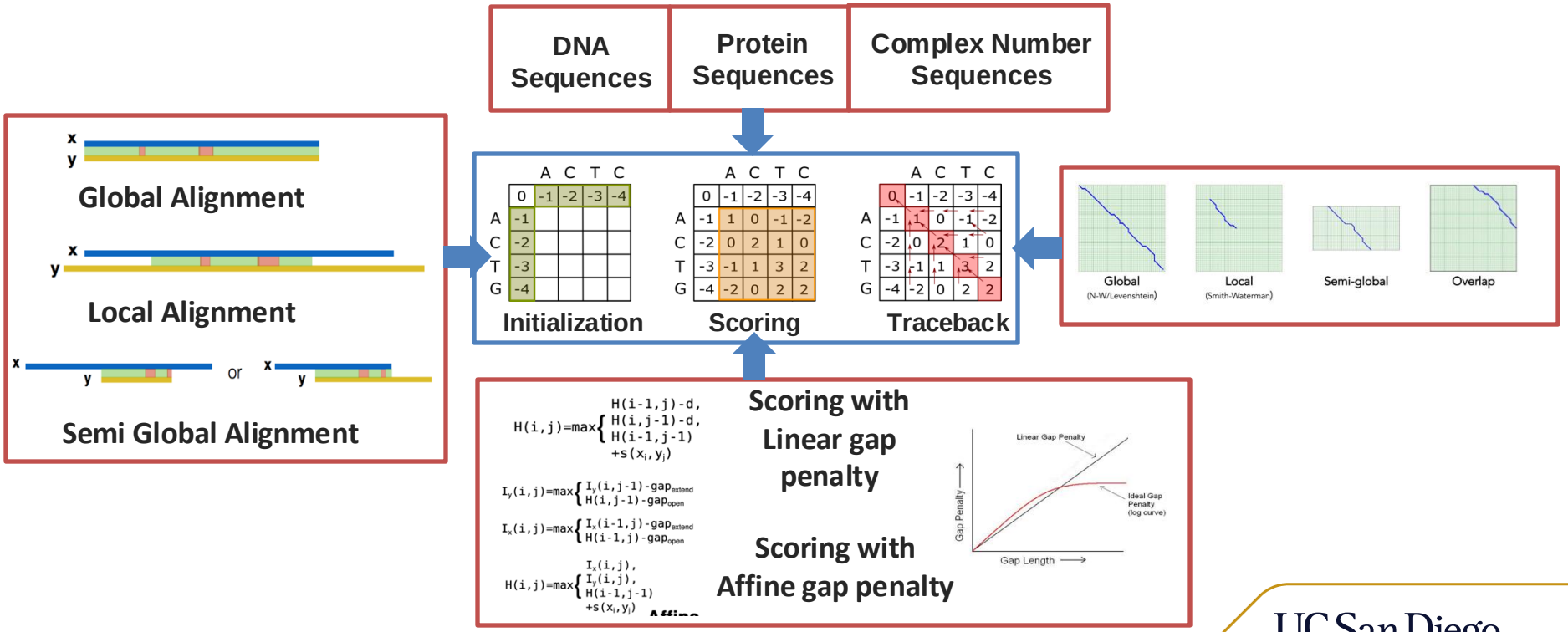


Variations in 2-D DP Paradigm



UC San Diego
JACOBS SCHOOL OF ENGINEERING

Variations in 2-D DP Paradigm



UC San Diego
JACOBS SCHOOL OF ENGINEERING

List of Algorithmic variations in 2-D DP Paradigm

Sl No.	Input Alphabets	Kernels	State-of-the-art Tools	Applications	Modifications in DP-HLS
1	DNA	Global Linear Alignment	BLAST, EMBOSS Stretcher	Similarity Search	N/A
2	DNA	Global Affine Alignment	BLAST, EMBOSS Needle	Accurate Similarity Search	Scoring
3	DNA	Local Linear Alignment	BLAST, FASTA, BLAT	Homology Search	Initialization and Traceback
4	DNA	Local Affine Alignment	BLAST, LASTZ	Whole Genome Alignment	Scoring, Initialization and Traceback
5	DNA	Global Two-piece Affine Alignment	Minimap2	Long Genome Alignment	Scoring
6	DNA	Overlap Alignment	CANU, Flye	Genome Assembly	Initialization and Traceback
7	DNA	Semi-global Alignment	BWA-MEM	Short Read Alignment	Initialization and Traceback

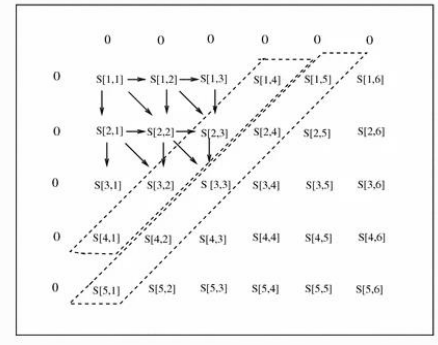
Sl No.	Input Alphabets	Kernels	State-of-the-art Tools	Applications	Modifications in DP-HLS
8	Seq. Profiles	Profile Alignment	CLUSTAL-W, MUSCLE	Multiple Sequence Alignment	Sequence Alphabet and Scoring
9	Complex Nos.	Dynamic Time Warping Algorithm	SquiggleKit	Basecalling	Sequence Alphabet and Scoring
10	DNA	Viterbi Algorithm	HMMER, Augustus	Remote Homology Search, Gene Prediction	Scoring (no Traceback)
11	DNA	Banded Global Linear Alignment	BLAST, Bowtie	Fast Similarity Search	Scoring and Initialization
12	DNA	Banded Local Affine Alignment	Minimap2	Long Read Assembly	Initialization and Scoring (no Traceback)
13	DNA	Banded Global Two-piece Affine Alignment	Minimap2	Long Read Assembly	Scoring, Initialization and Traceback
14	Integers	Semi-global DTW	SquiggleFilter, RawHash	Basecalling	Sequence Alphabet and Scoring
15	Amino Acids	Local Linear Alignment with protein sequences	EMBOSS Water, BLASTp, DIAMOND	Protein Sequence Alignment	Sequence Alphabet and Scoring

Hardware Acceleration of 2-D DP Algorithms

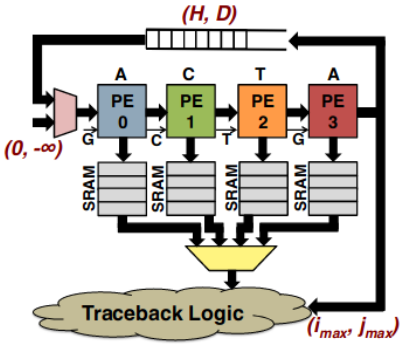
2-D DP algorithm is computationally expensive!

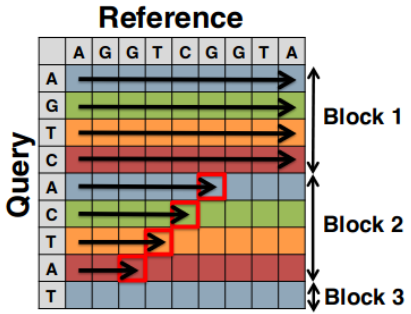
Several existing work accelerates these algorithms on specialized hardware – ASICs/FPGAs.

Maps to **Linear Systolic Array Architecture**



Wavefront Parallelism





UC San Diego
JACOBS SCHOOL OF ENGINEERING

Hardware Acceleration of 2-D DP Algorithms

Examples of some existing works accelerating 2-D DP algorithms

X. Fei, Z. Dan, L. Lina, M. Xin, and Z. Chunlei, "**Fpgasw: accelerating**

A. Haghi, S. Marco-Sola, L. Alvarez, D. Diamantopoulos, C. Hagleit-

Non-configurable,
Requires hardware re-design from scratch

Y. Turakhia, S. D. Goenka, G. Bejerano, and W. J. Dally, "**Darwin-wga: A co-processor provides increased sensitivity in whole genome alignments with high speedup**," in 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2019, pp. 359–372.

International Symposium on Microarchitecture (MICRO), 2020, pp. 937–950.

T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K.-Y. Chen, R. Das, D. Blaauw, and S. Narayanasamy, "**SquiggleFilter: An Accelerator for Portable Virus Detection**," Sep. 2021, arXiv:2108.06610 [q-bio]. [Online].

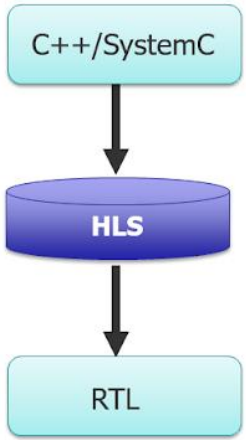
High-Level Synthesis

Automated Design Process

Macroarchitecture specifications in C/C++ to RTL level structure

Vary constraints to generate multiple RTL implementations with single C/C++ algorithm

Optimal constraints required for optimal hardware design



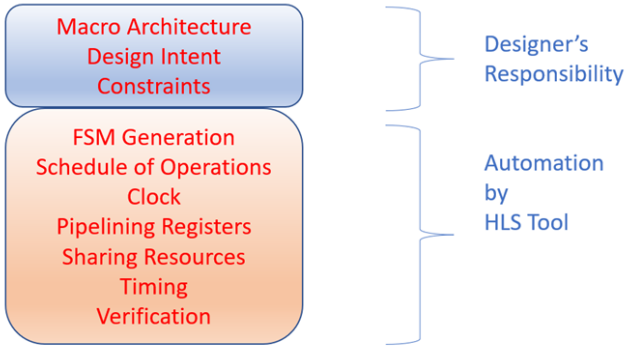


Table of Contents

Objective

Develop a generic framework to accelerate 2-D DP algorithms, which is:

- Highly Programmable
- Support many applications
- Create efficient hardware designs
- No hardware expertise
- Improve productivity

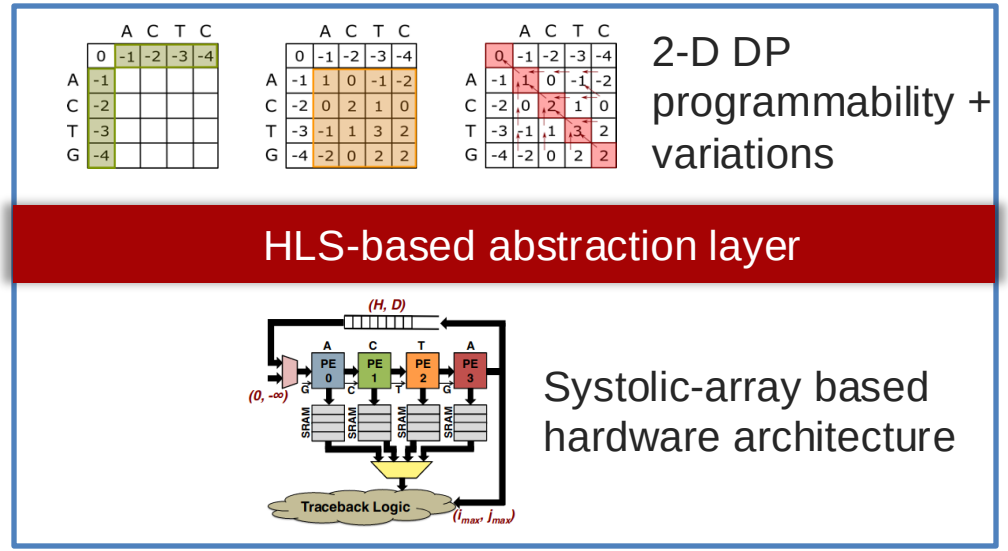
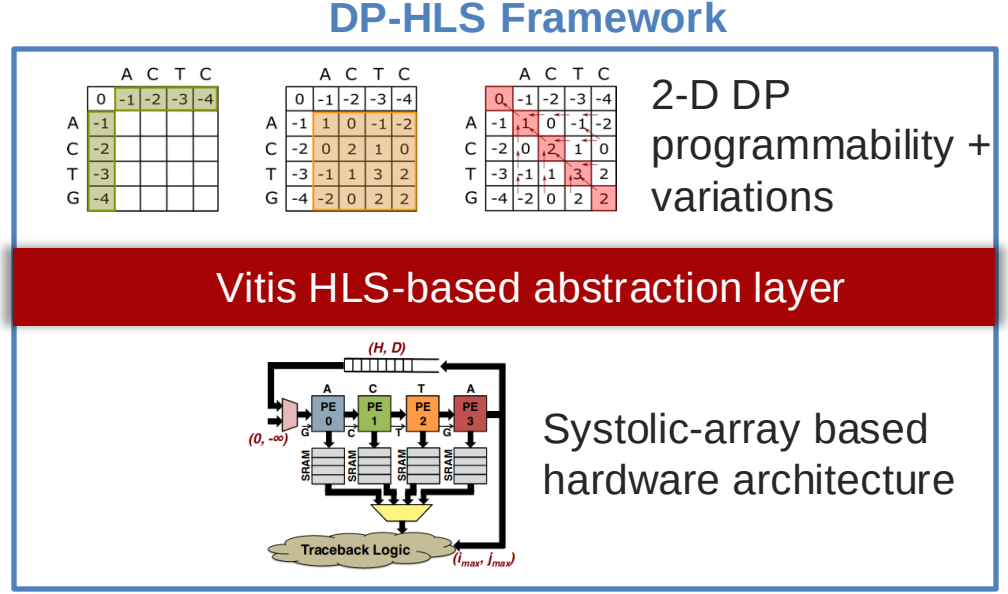


Table of Contents

DP-HLS - Overview

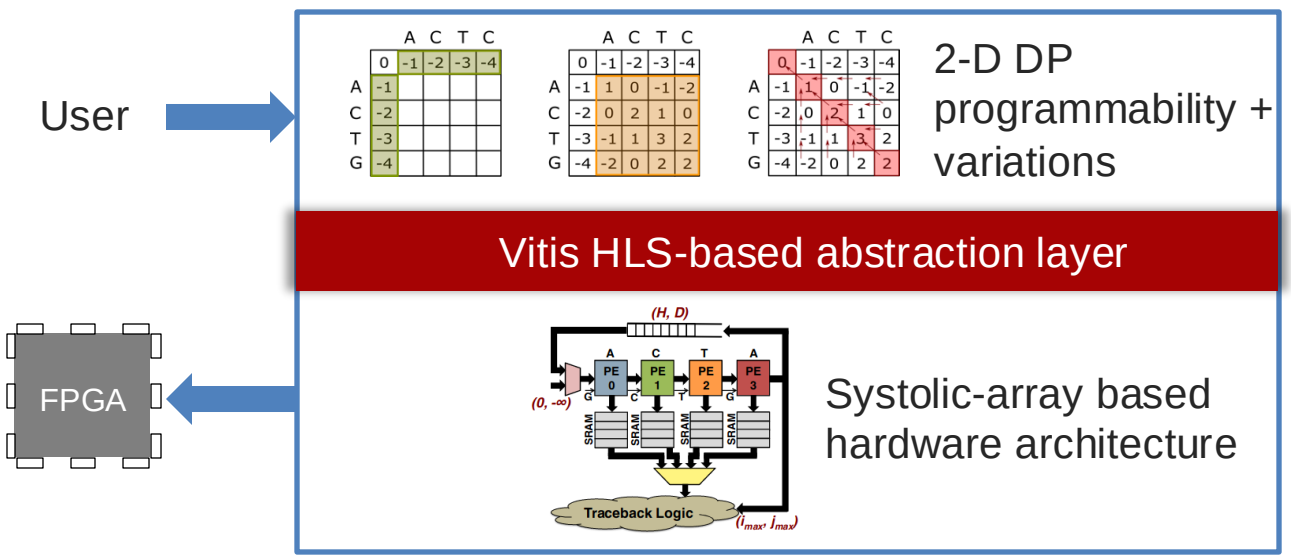
DP-HLS is a generic framework to accelerate 2-D DP algorithms, which is:

- Highly Programmable
- Support many applications
- Create efficient hardware designs
- No hardware expertise
- Improve productivity



DP-HLS - Overview

DP-HLS is a generic framework to accelerate 2-D DP algorithms



DP-HLS Front-end
(C/C++ based customizable API)

DP-HLS Back-end
(contains pre-defined HLS optimization directives)

DP-HLS – How it works

Step 1: Specify kernel configurations
Step 2: Simulate the kernel

DP-HLS Front-end design

Step 3: Perform synthesis and co-simulation
Step 4: Analyze the implementation results

Vitis HLS Tool

Step 5: Generate bitstream and deploy to FPGA

AWS F1 FPGA

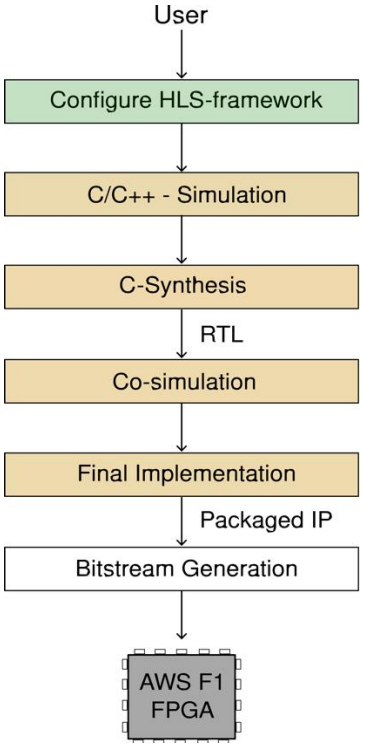
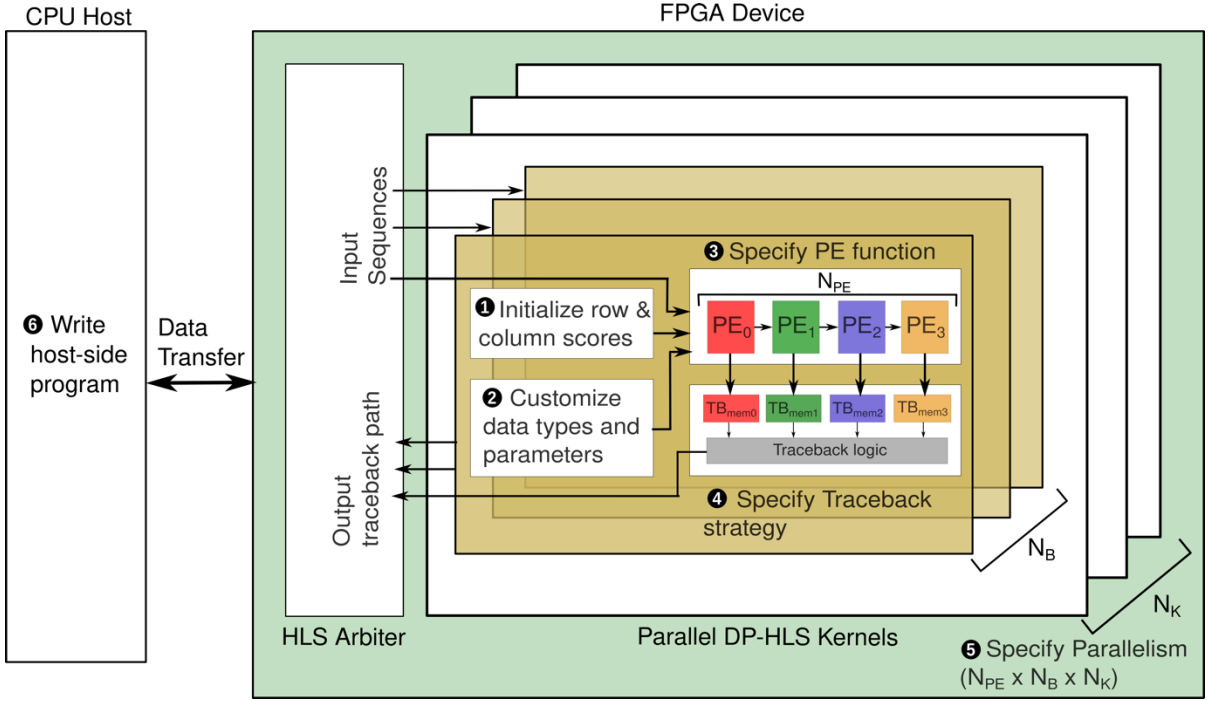


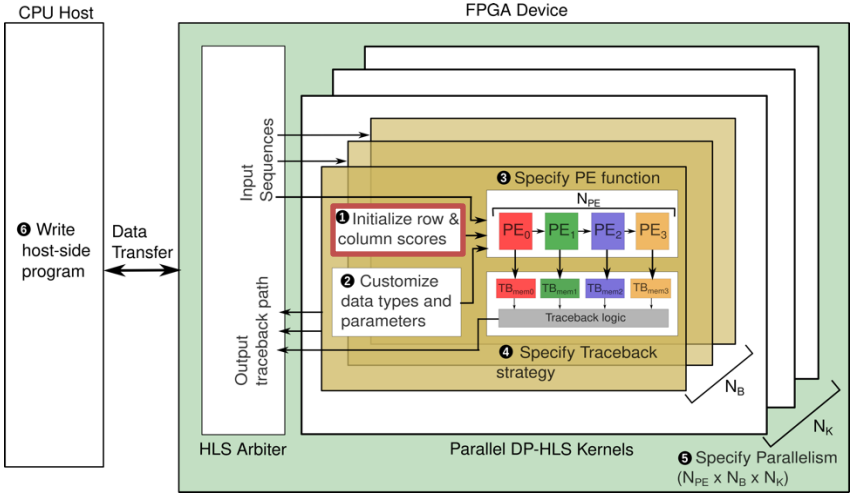
Table of Contents

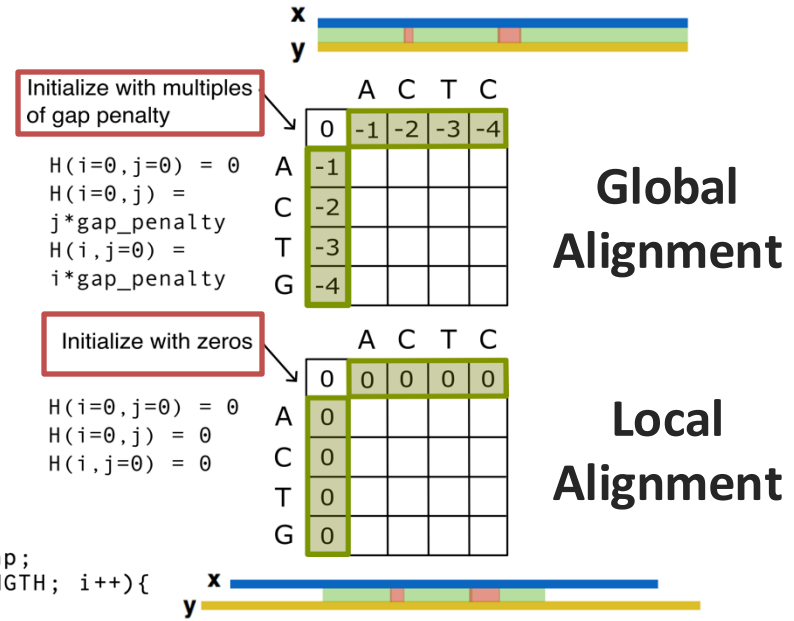
DP-HLS - Front-end



DP-HLS - Front-end

1. Initialize row and column scores



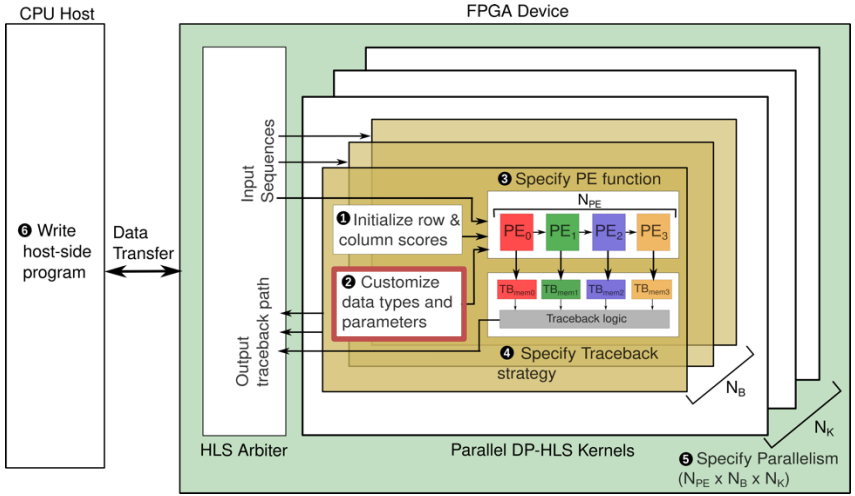



```
type_t gap = scoring_params.linear_gap;
for (int i = 0; i < MAX_REFERENCE_LENGTH; i++){
    init_row_scr[i][0] = i*gap; }
for (int i = 0; i < MAX_QUERY_LENGTH; i++){
    init_col_scr[i][0] = i*gap; }
```


UC San Diego
JACOBS SCHOOL OF ENGINEERING

DP-HLS - Front-end

2. Customize data types and parameters



- Sequence Alphabet: DNA or RNA or Complex Sequence

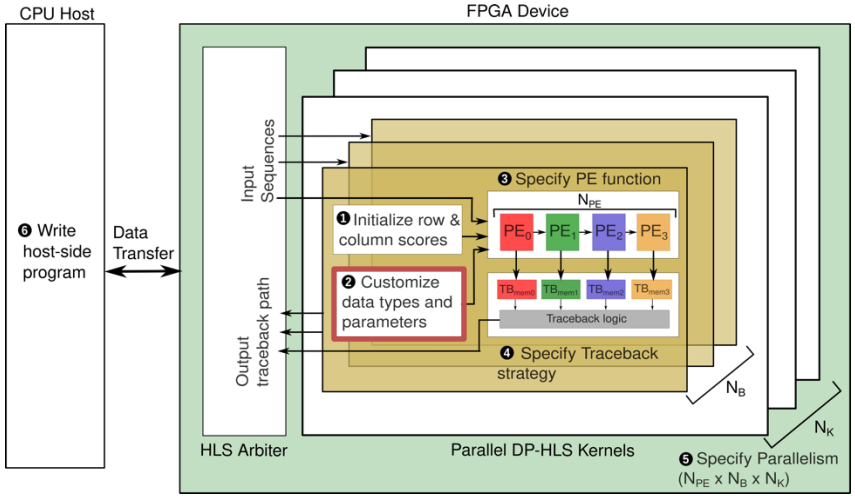
```
typedef ap_uint<2>
    char_t;
```

- Number of scoring layers (`N_LAYERS`)
- Scoring parameters

```
struct ScoringParams {
    type_t mismatch;
    type_t match;
    type_t linear_gap;
} params;
```

DP-HLS - Front-end

2. Customize data types and parameters

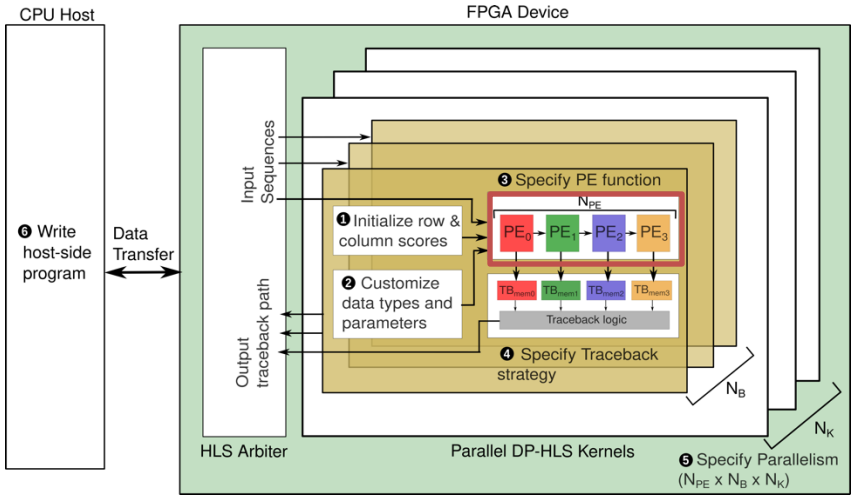


- Maximum Sequence Length (`MAX_REFERENCE_LENGTH` & `MAX_QUERY_LENGTH`)
- Traceback pointer data types and states

```
enum TB_STATE {
    MM, INS, DEL
} tb_next_state,
    tb_curr_state;
```

DP-HLS - Front-end

3. Specify PE function



$$H(i,j)=\max\begin{cases}H(i-1,j)-d,\\H(i,j-1)-d,\\H(i-1,j-1)+s(x_i,y_j)\end{cases}$$

Scoring with Linear gap penalty

$$I_y(i,j)=\max\begin{cases}I_y(i,j-1)-gap_{extend}\\H(i,j-1)-gap_{open}\end{cases}$$

$$I_x(i,j)=\max\begin{cases}I_x(i-1,j)-gap_{extend}\\H(i-1,j)-gap_{open}\end{cases}$$

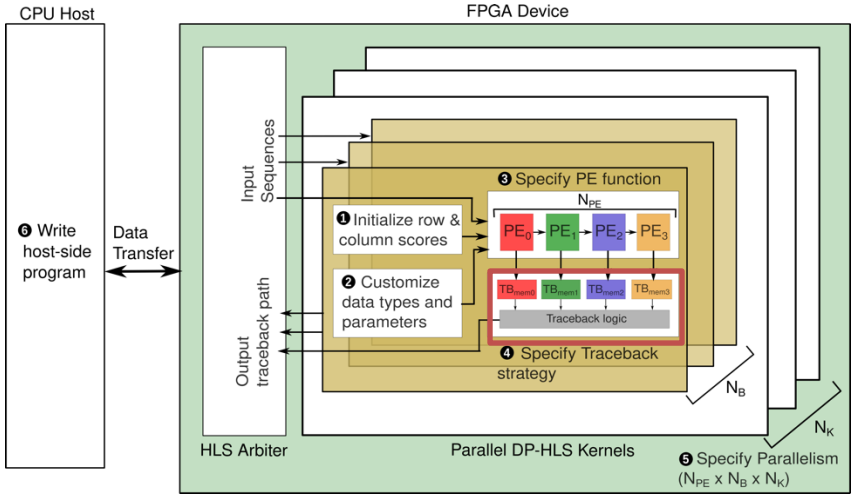
$$H(i,j)=\max\begin{cases}I_x(i,j),\\I_y(i,j),\\H(i-1,j-1)+s(x_i,y_j)\end{cases}$$

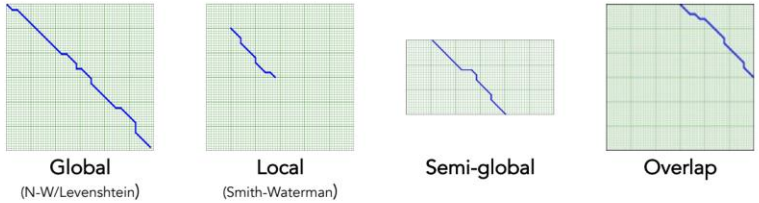
Scoring with Affine gap penalty

```
type_t linear_gap = params.linear_gap;
type_t ins = dp_mem_left[0] + linear_gap;
type_t del = dp_mem_up[0] + linear_gap;
type_t match = dp_mem_diag[0] + (lc_qry_val ==
    lc_ref_val) ? params.match : params.mismatch;
```

DP-HLS - Front-end

4. Specify Traceback Strategy

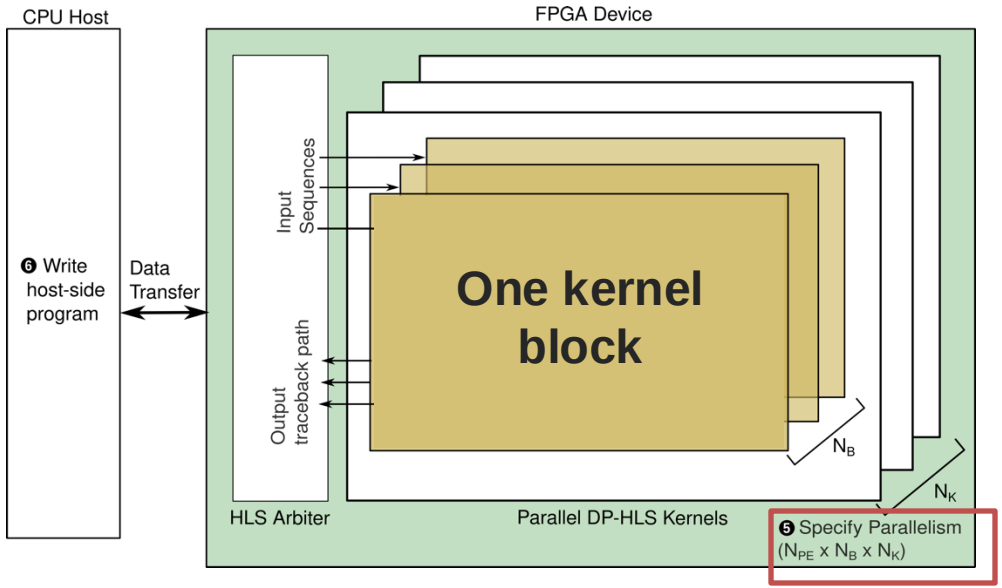





```
if (tb_state == TB_STATE::MM){
    if (tb_ptr == TB_DIAG){tb_move = AL_MMI; }
    else if (tb_ptr == TB_UP){tb_move = AL_DEL; }
    else if (tb_ptr == TB_LEFT){tb_move = AL_INS;}
    else if (tb_ptr == TB_END) {tb_move = AL_END;}
    else {tb_move = AL_END;}
    tb_state = TB_STATE::MM;}
```

DP-HLS - Front-end

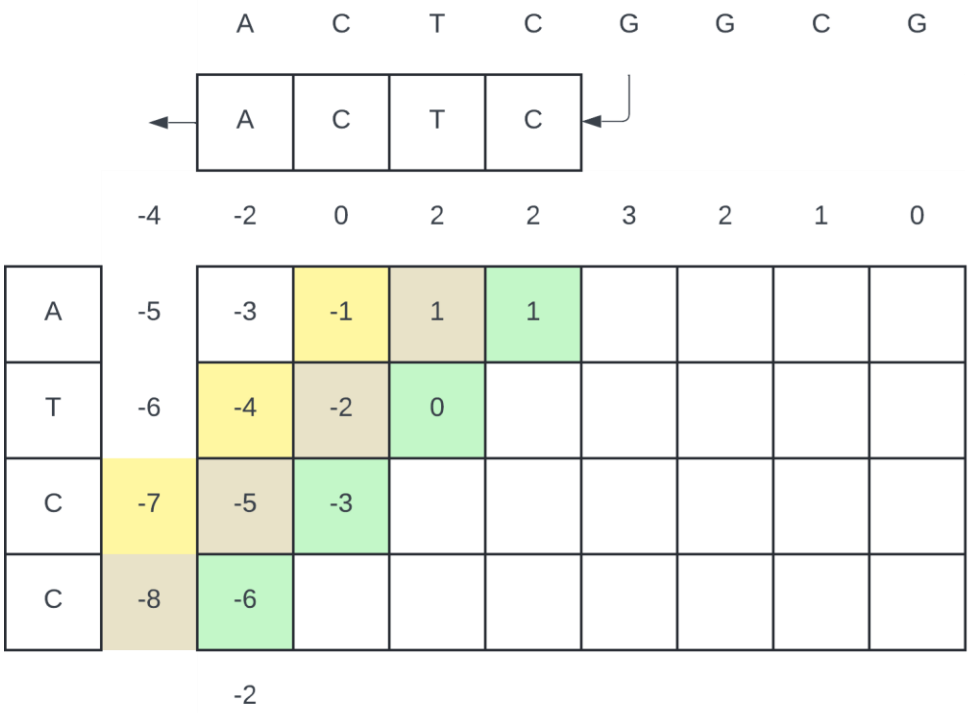
5. Specify parallelism

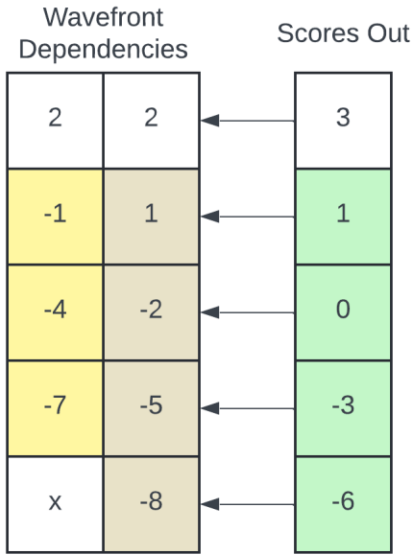


- N_{PE} → Number of processing elements → **Defines Inner-loop parallelism**
- N_B → Number of parallel blocks
- N_K → Number of kernels executed in parallel
- N_B and N_K → **Defines outer-loop parallelism**

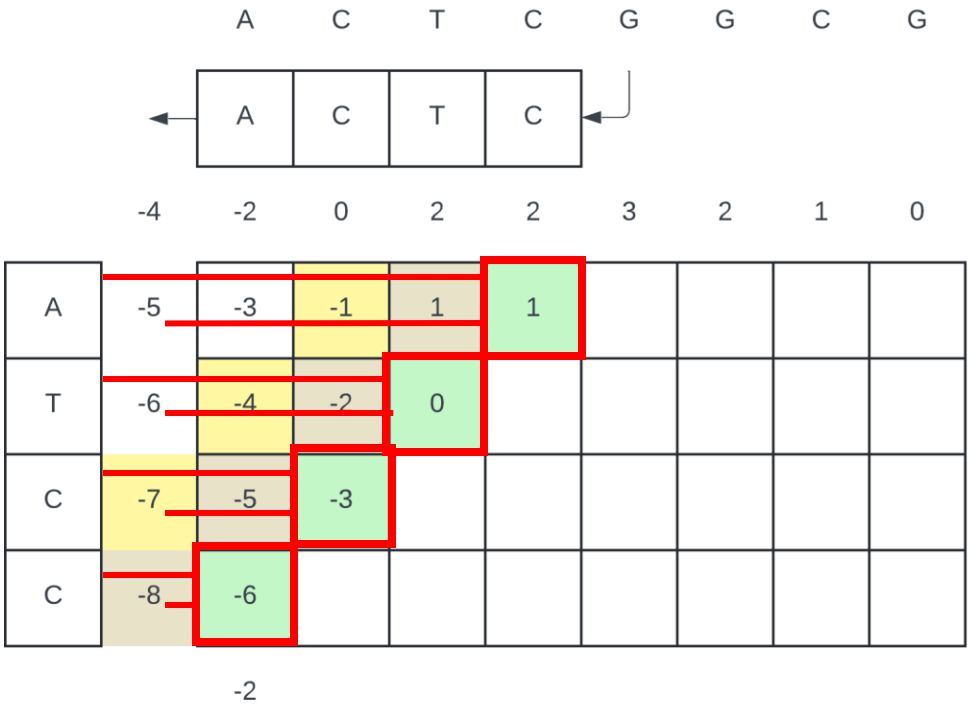
Table of Contents

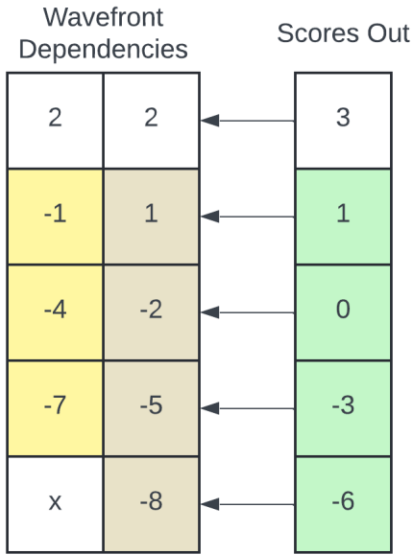
Wavefront Parallelism



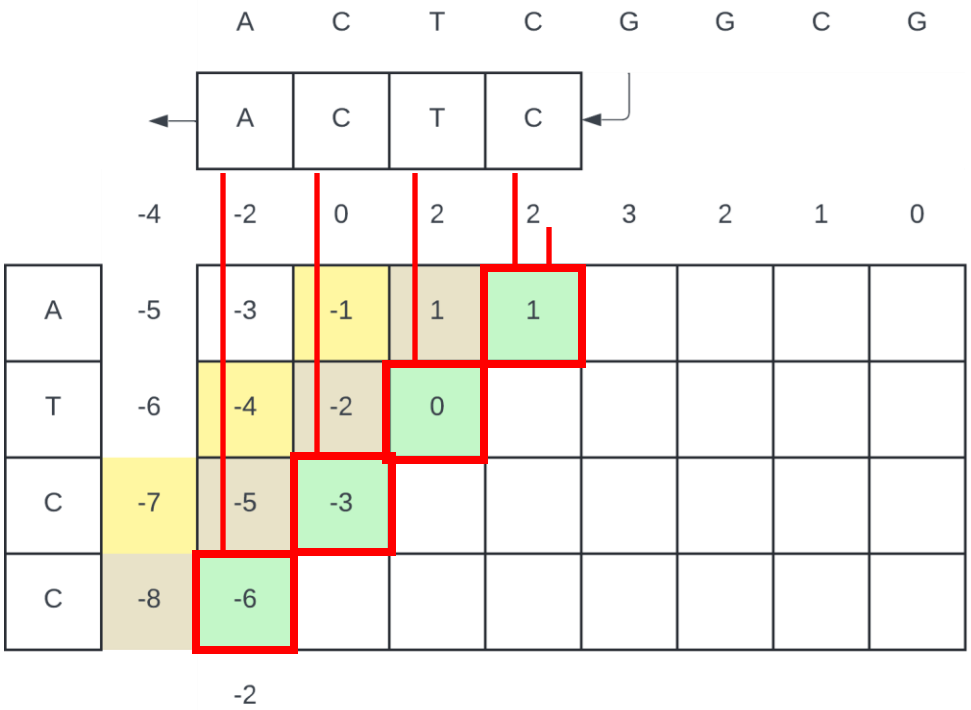


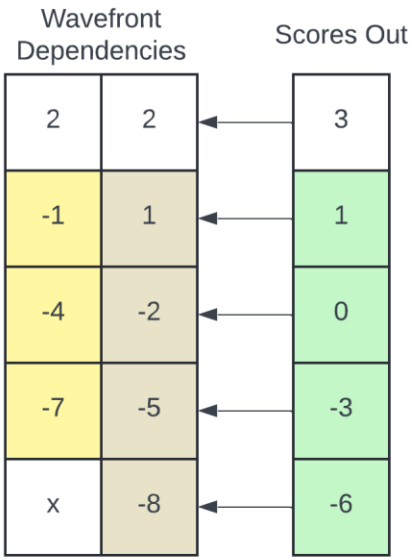
Wavefront Parallelism



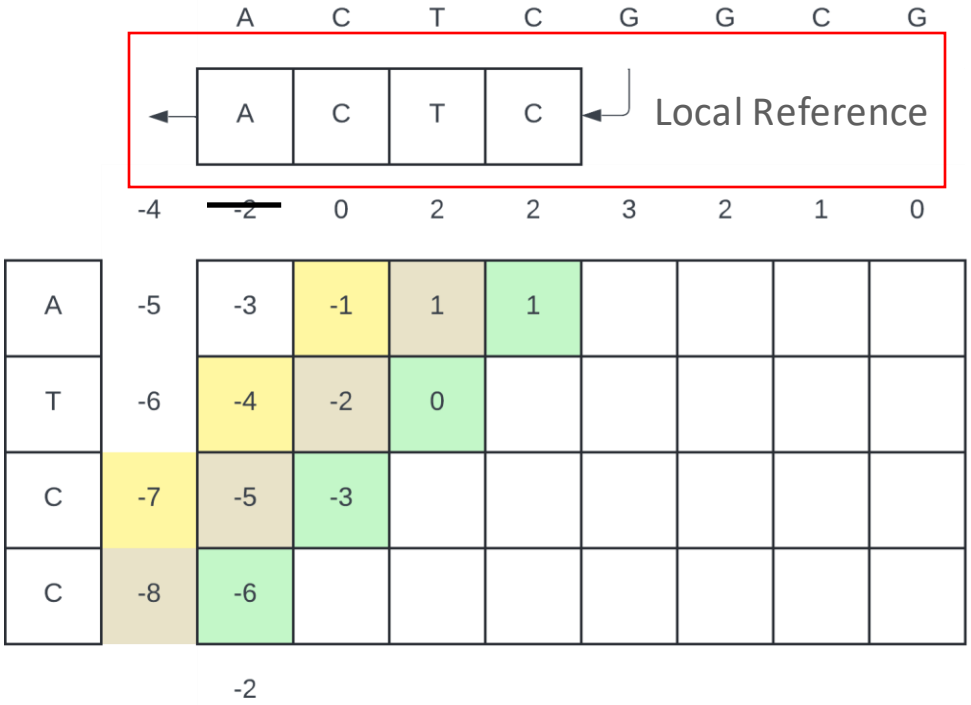


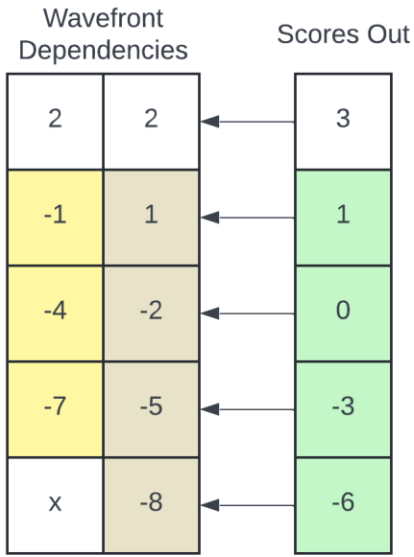
Wavefront Parallelism



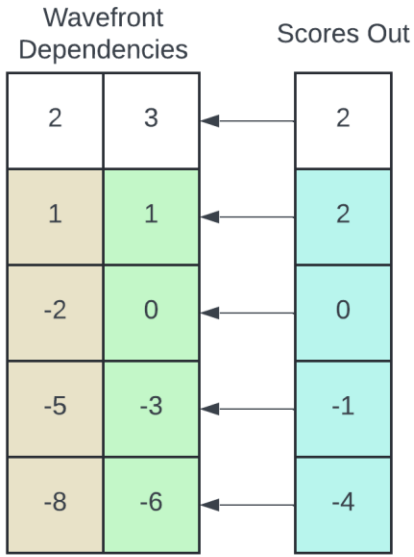


Wavefront Parallelism

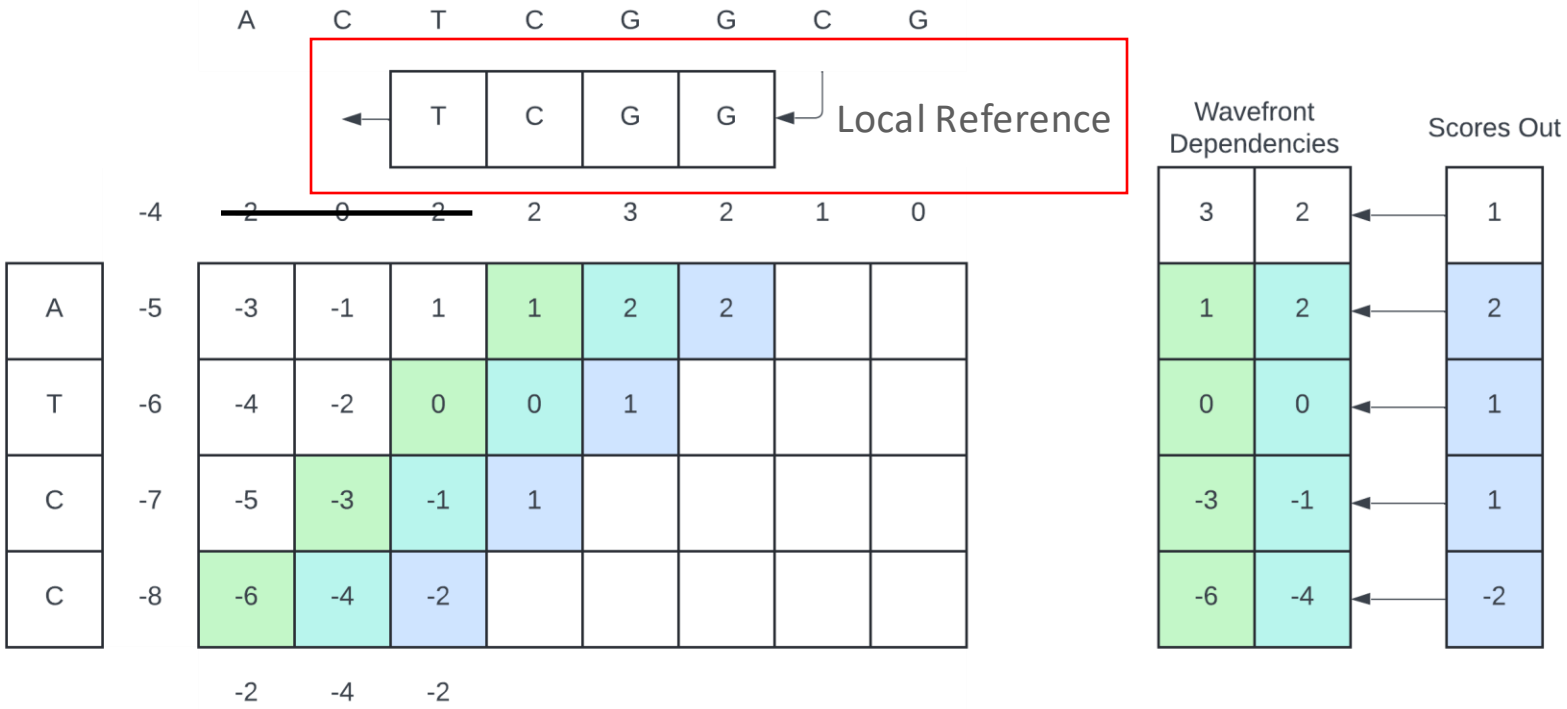




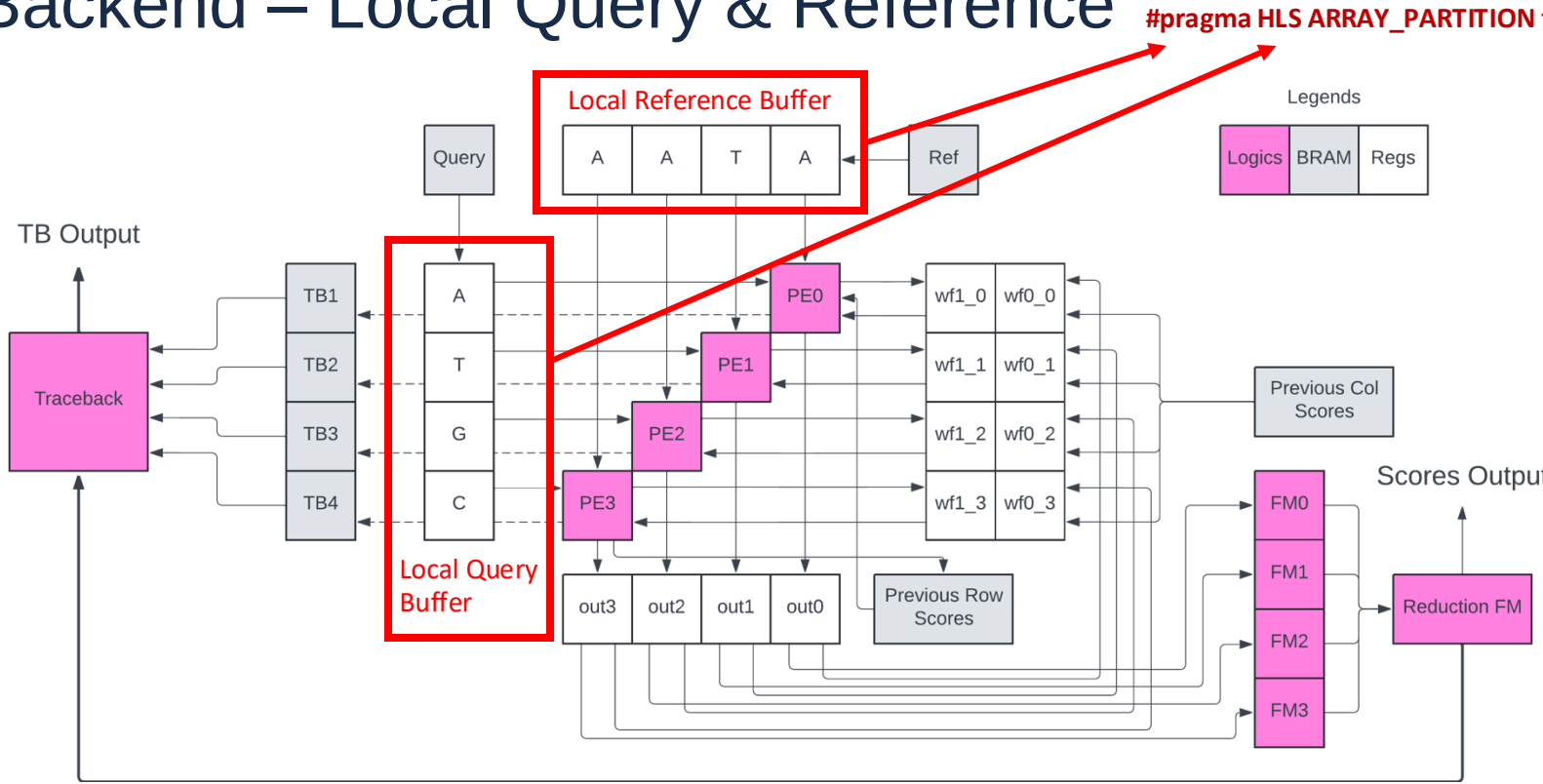
Wavefront Parallelism



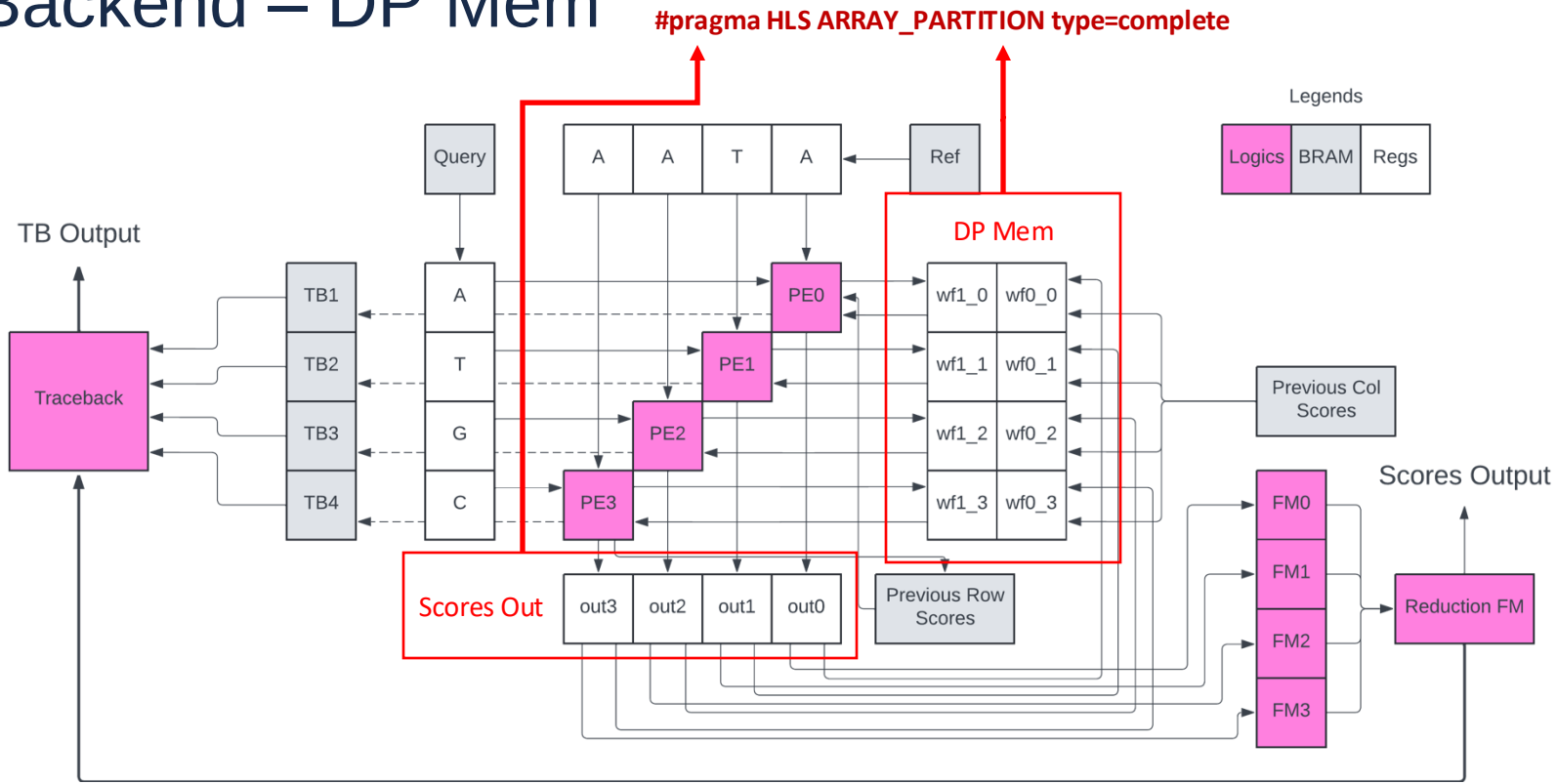
Wavefront Parallelism



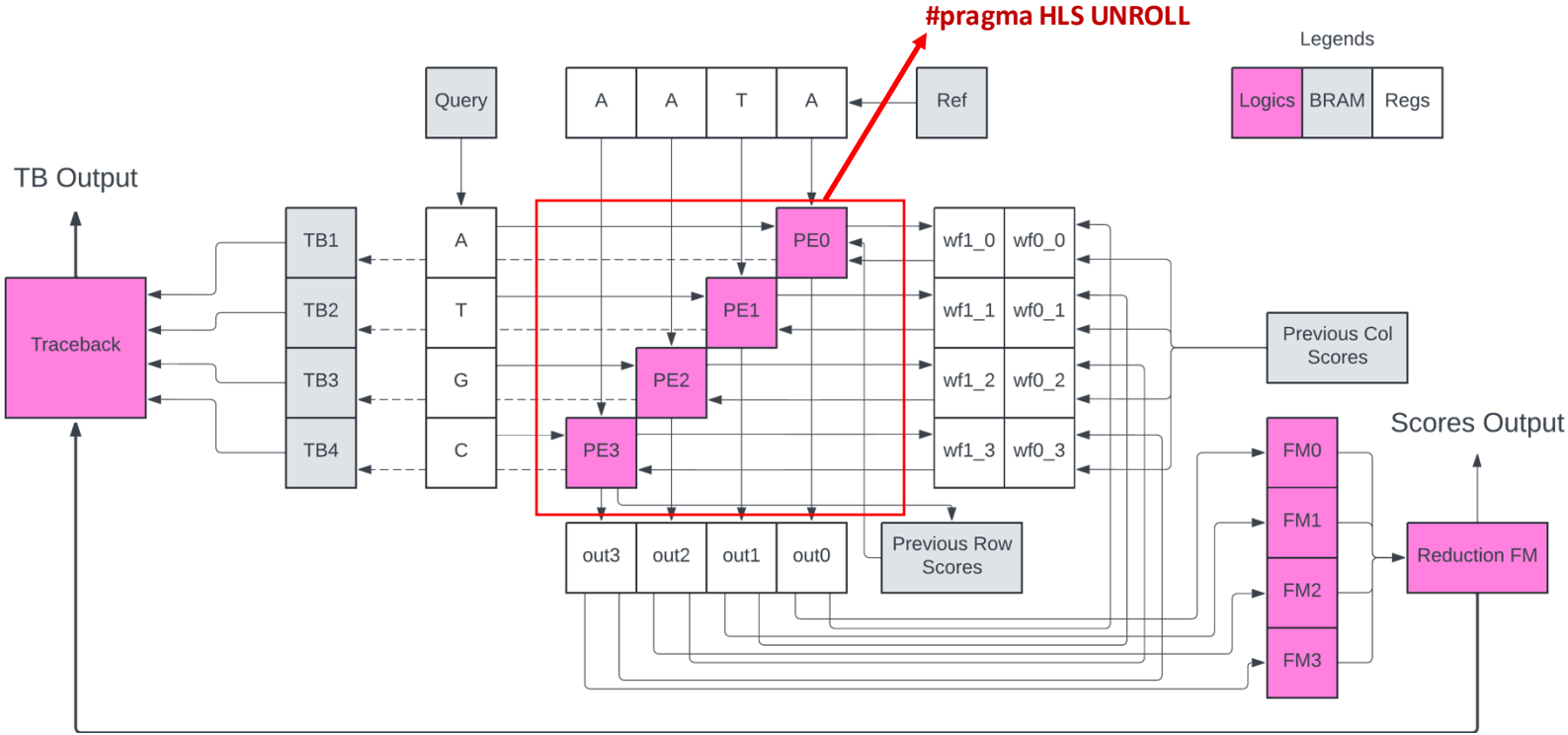
Backend – Local Query & Reference



Backend – DP Mem



Backend – Create Systolic Array



Backend – Inter-Chunk Dependency

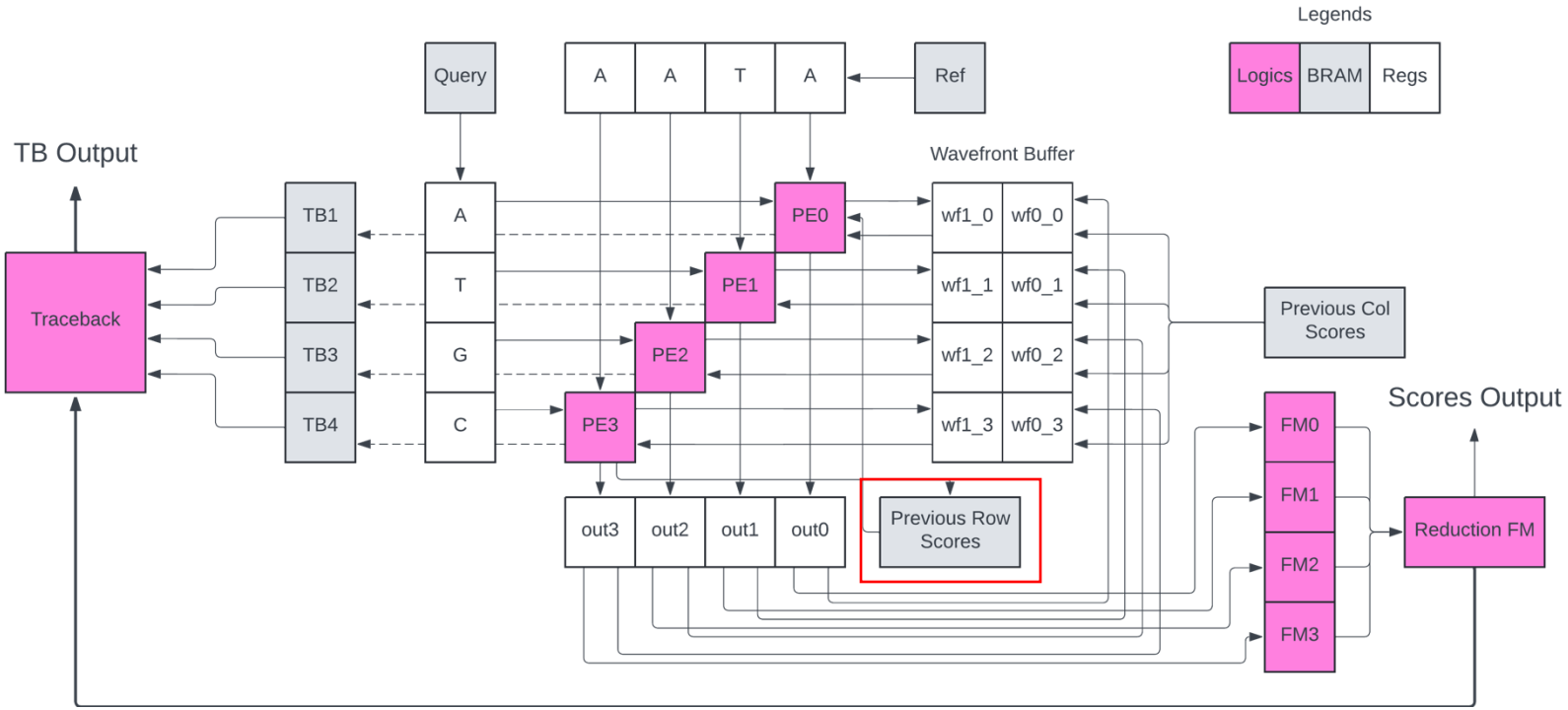
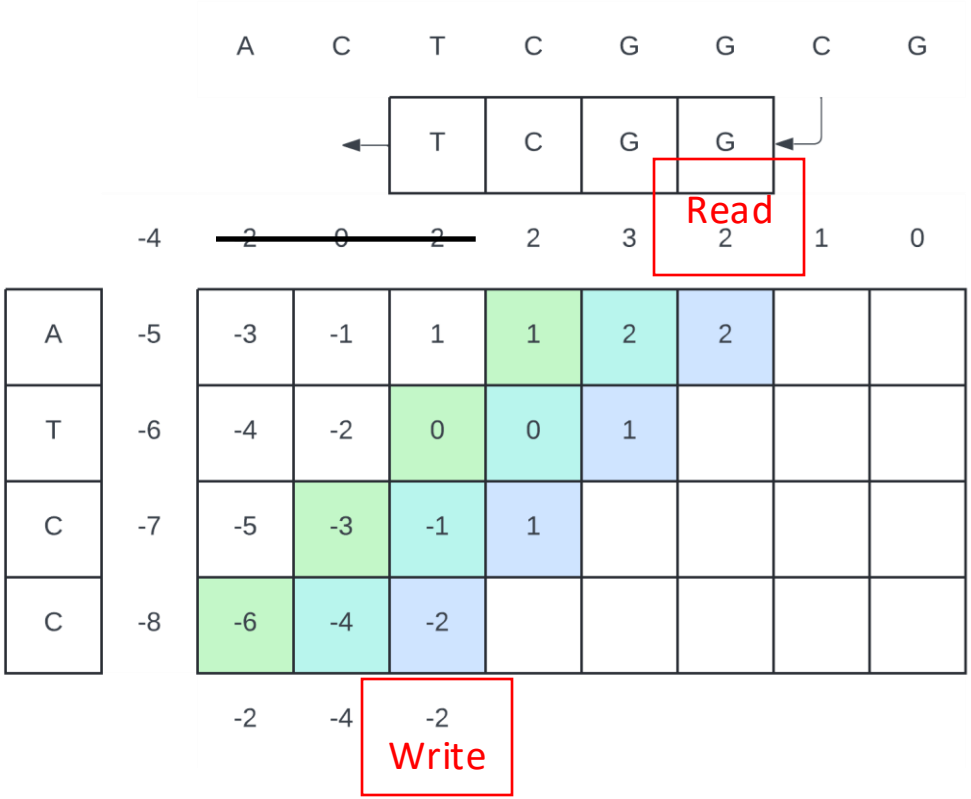
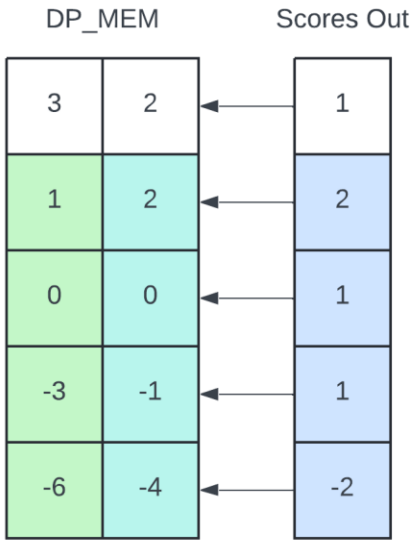
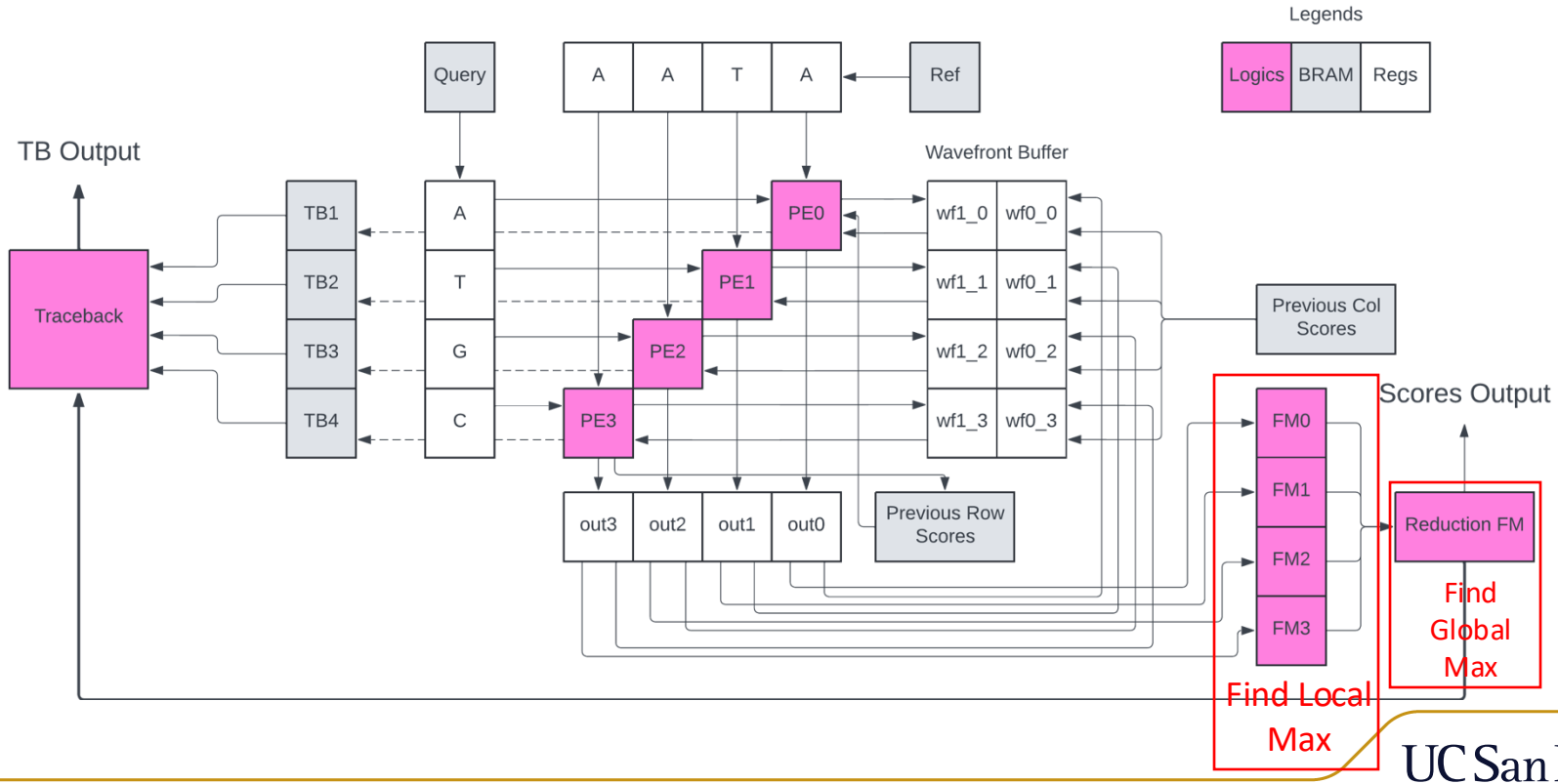


Illustration – Previous Row Scores





Backend – Maximum Scores



Backend – TB Memory

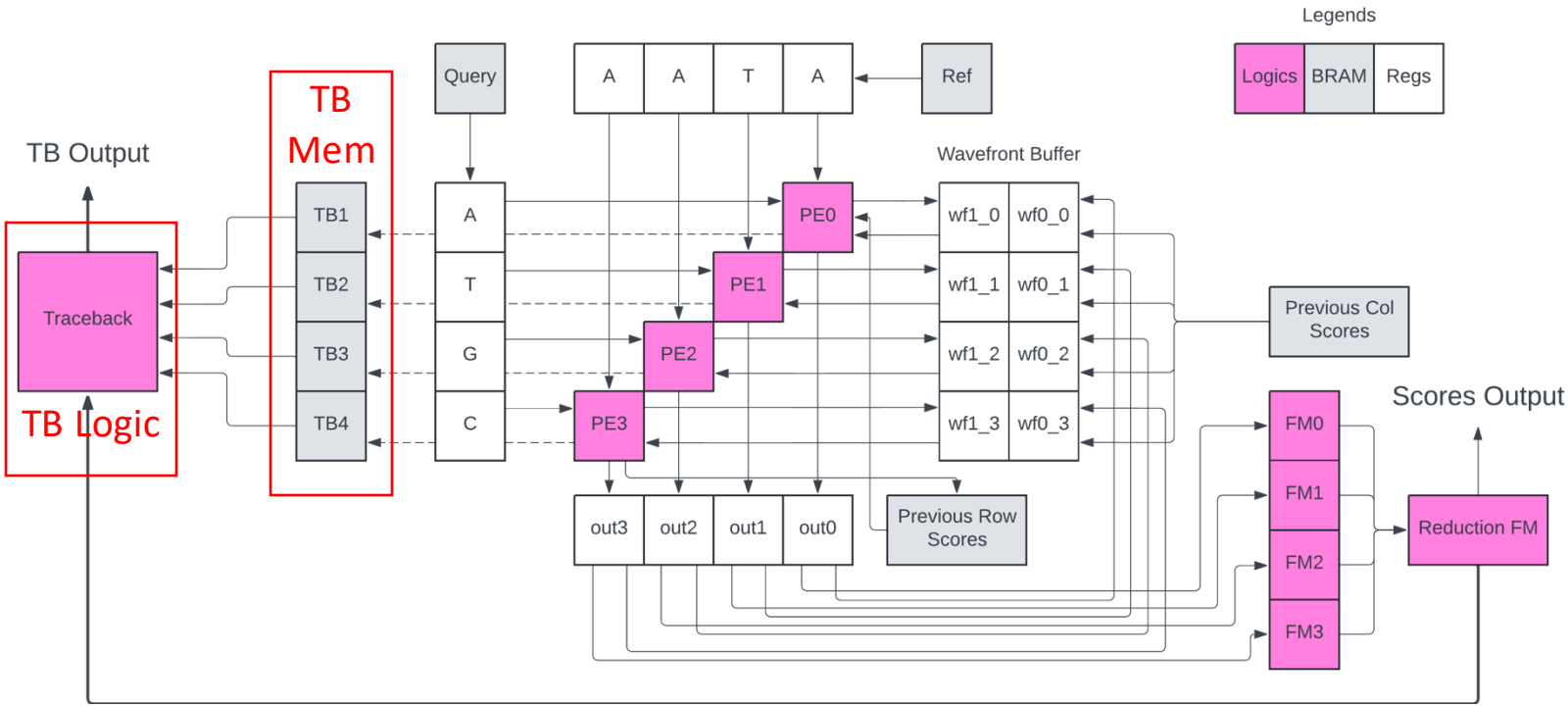
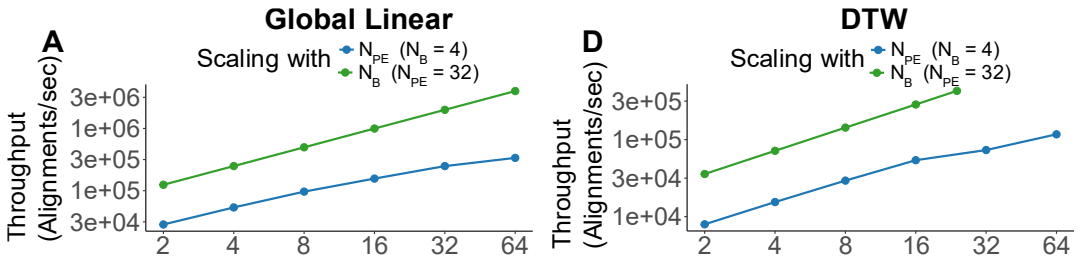


Table of Contents

Results

- **DP-HLS Generates 1-D systolic Array Architecture**

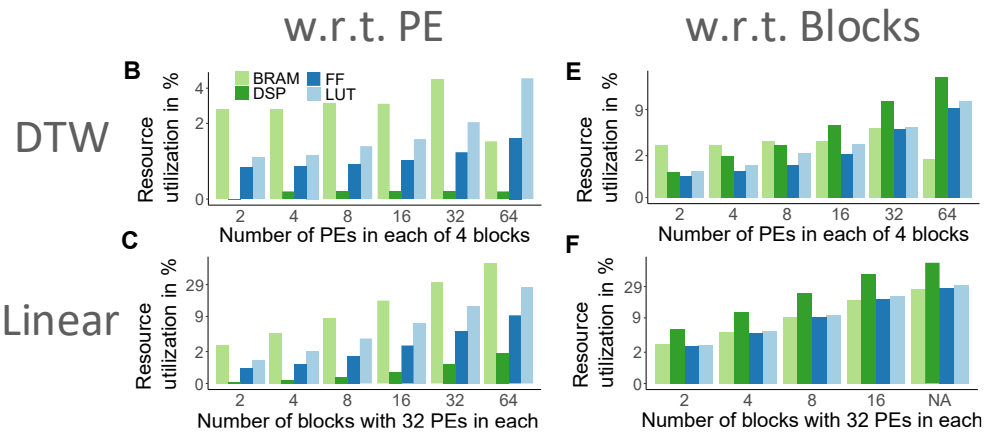


Note:

- V++ and Vitis HLS 2021.2 are used for compilation and cycle count profiling for all DP-HLS kernels.

Results

- DP-HLS Generates 1-D systolic Array Architecture
- **The Scaling is Stable Across Diverse Kernels**

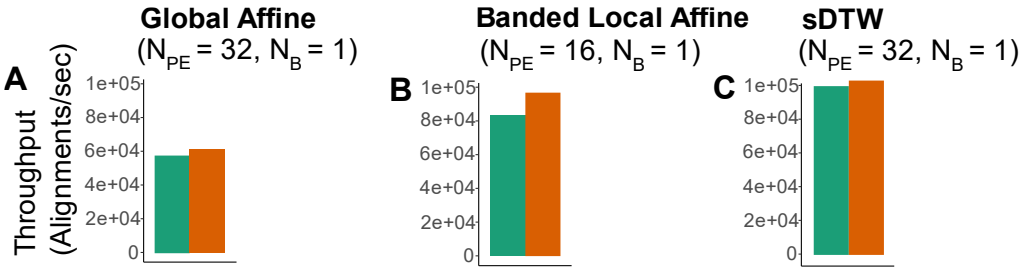


Note:

- V++ and Vitis HLS 2021.2 are used for compilation and cycle count profiling for all DP-HLS kernels.

Results

- DP-HLS Generates 1-D systolic Array Architecture
- The Scaling is Stable Across Diverse Kernels
- **Similar Throughput to RTL**

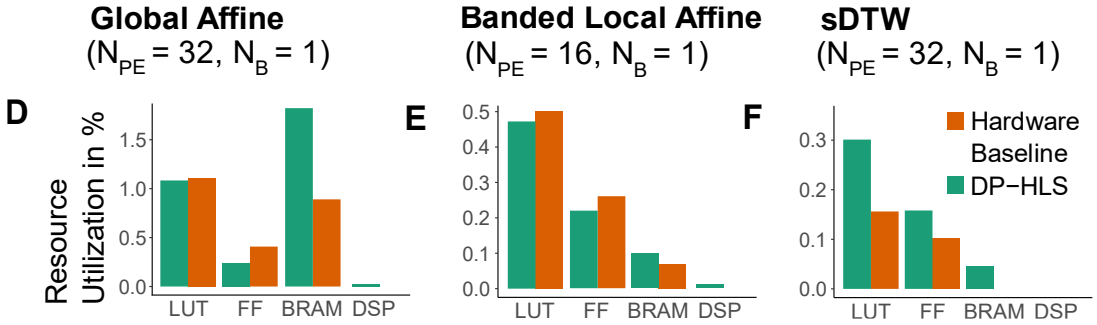


Notes:

- The throughput is measured based on the cycle count in the Vitis HLS cosimulation (DP-HLS) and Vivado Waveform and Iverilog (Hardware baseline)

Results

- DP-HLS Generates 1-D systolic Array Architecture
- The Scaling is Stable Across Diverse Kernels
- Similar Throughput to RTL
- **Comparable Kernel Resource Utilization to RTL**

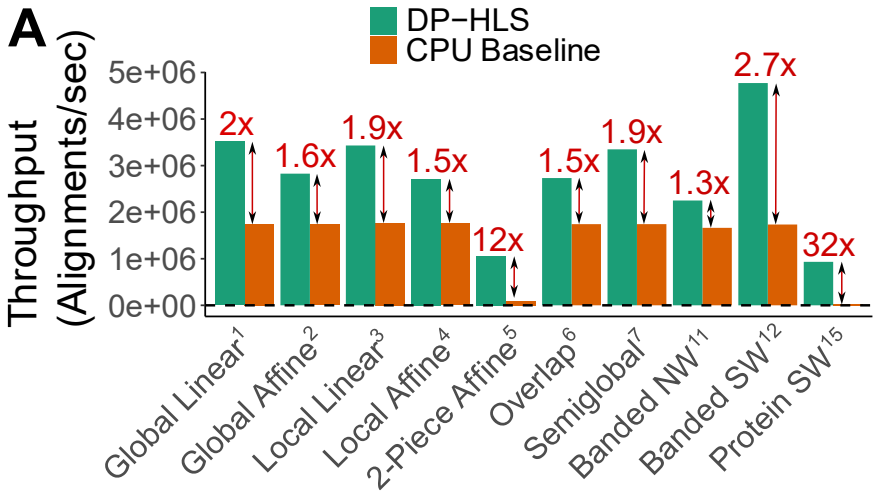


Notes:

- Vivado 2021.2 are used for all hardware baselines. All complied for xcvu9p-flgb2104-2-I device.
- Since the original SquiggleFilter has a rewards bonus that is hard to adapt in DP-HLS, it is removed and becomes a SW kernel with only the diagonal and upper dependency

Results

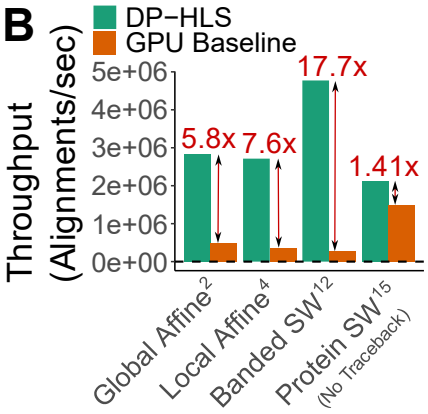
- DP-HLS Generates 1-D systolic Array Architecture
- The Scaling is Stable Across Diverse Kernels
- Similar Throughput to RTL
- Comparable Kernel Resource Utilization to RTL
- **1.3x – 32x Speedup to CPU Baseline**



Notes:
Software baselines are deployed on an AWS compute-optimized **c4g.8xlarge** instance with the same cost as the F1 instance.
Baselines: SeqAn3 [1-4, 6-7, 12, 11], EMBOSS Water [15], Minimap2 [5]

Results

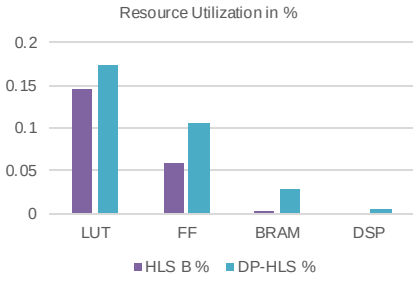
- DP-HLS Generates 1-D systolic Array Architecture
- The Scaling is Stable Across Diverse Kernels
- Similar Throughput to RTL
- Comparable Kernel Resource Utilization to RTL
- 1.3x – 32x Speedup to CPU Baseline
- **1.41x – 17.7x Speedup to GPU Baseline**

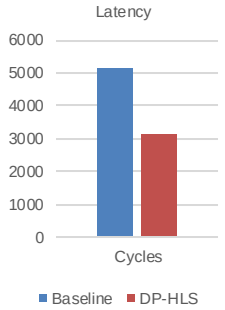


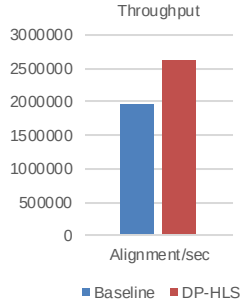
Notes:
GPU baselines are deployed on an AWS P3.2xlarge instance ($3.06/hr).
Baselines: GASAL2 [2, 4, 12], CUDASW4++ [15]

Results

- DP-HLS Generates 1-D systolic Array Architecture
- The Scaling is Stable Across Diverse Kernels
- Similar Throughput to RTL
- Comparable Kernel Resource Utilization to RTL
- 1.3x – 32x Speedup to CPU Baseline
- 1.41x – 17.7x Speedup to GPU Baseline
- **Higher Throughput than HLS Baseline**







	Baseline	DP-HLS
Post Routing Frequency (MHz)	317.6	259.1

Notes:
- The baseline is Vitis Genomics Library (v2021.2) Smith-Waterman kernel

Conclusion

- We present DP-HLS, an **HLS framework** to design **hardware accelerators for DP-based bioinformatics algorithms**.
- The user **doesn't need hardware design backgrounds** to customize any hardware kernels.
- It is **highly scalable** and results in stable throughput and resource utilization scaling.
- The throughput and resource utilization are **comparable to accelerators with RTL designs**.
- Achieves a maximum **32x to 17x iso-cost speedup** compared to CPU and GPU baselines respectively.

Acknowledgments

Thanks for watching!

Advisor:

- Yatish Turakhia (UCSD)

Collaborators:

- AMD-omics group

AMD

ArXiv Preprint

GitHub Repository

DP-HLS

Turakhia Lab