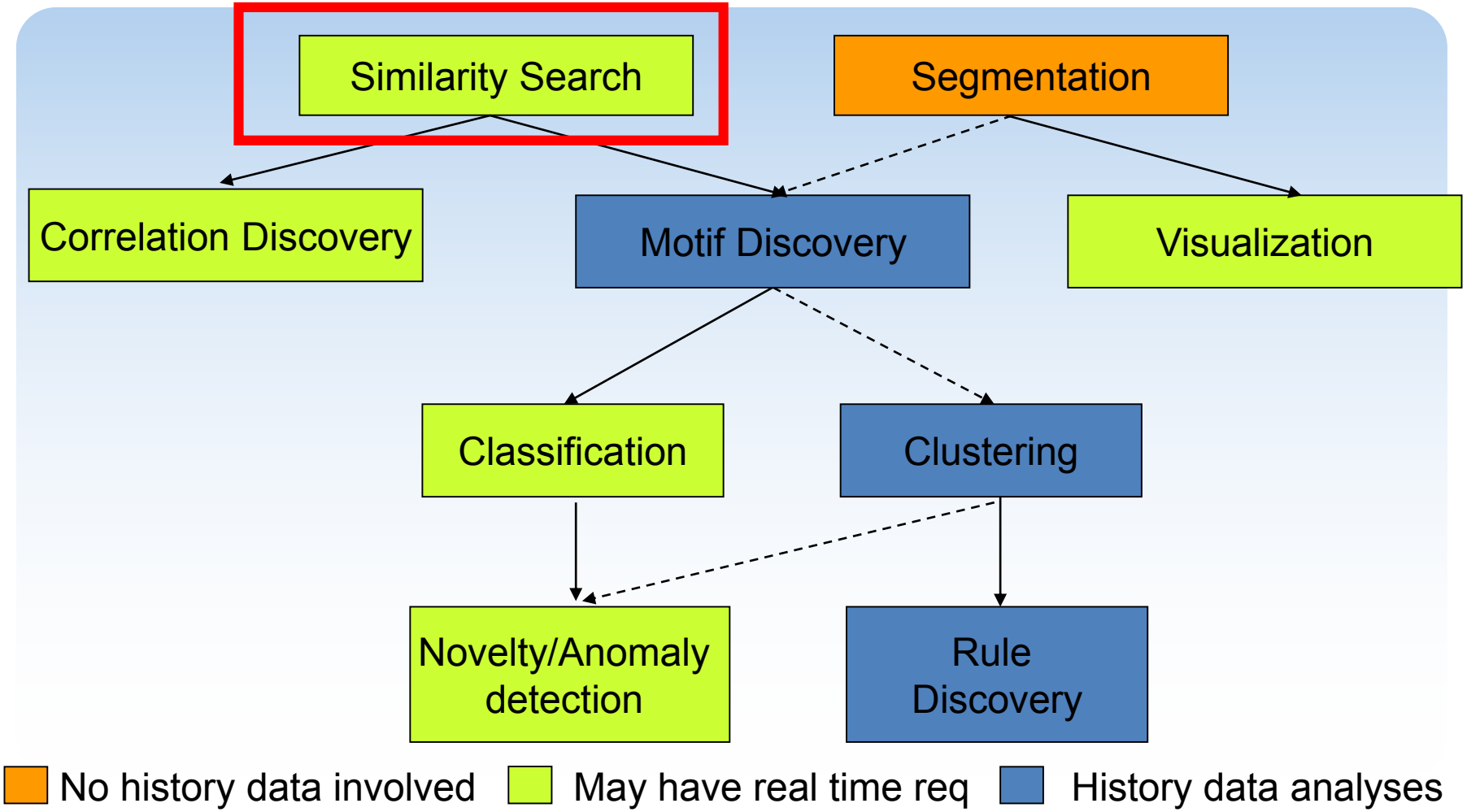
Accelerating Subsequence Similarity Search Based on Dynamic Time Warping Distance with FPGA

Zilong Wang[1], Sitao Huang[1], Lanjun Wang[2], Hao Li[2], Yu Wang[1], Huazhong Yang[1]
[1]E.E. Dept., TNLIST, Tsinghua University; [2]IBM Research China

Similarity search, or subsequence retrieval, is one of the most important sub-routines in time series data mining. there are many applications that need to find some patterns in the time series.

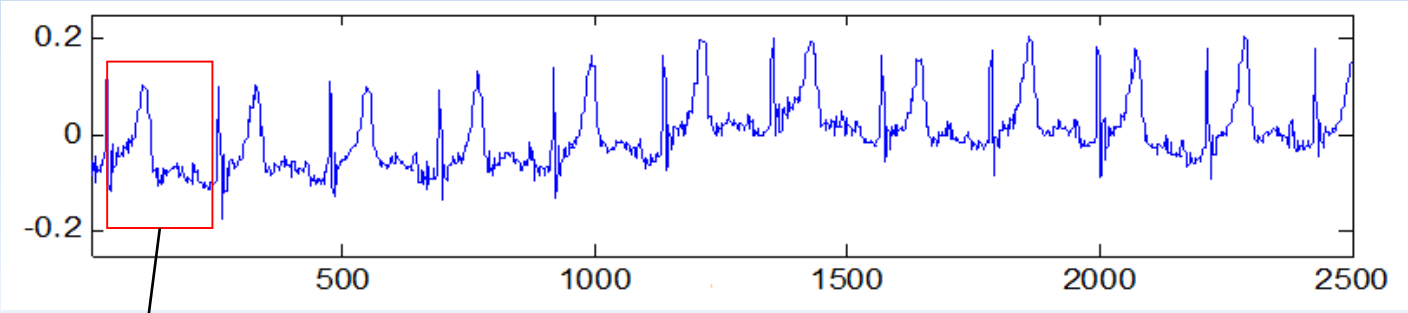
Background: Time Series Data Mining

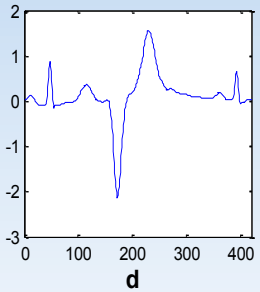


For example, we need to find a special heartbeat, which indicates a heart disease, from the electrocardiogram (ECG).
Firstly, we need to pick out sub-sequences with a sliding window

Background： Subsequence Similarity Search

- Time series (Electrocardiogram) & pattern (heartbeat)

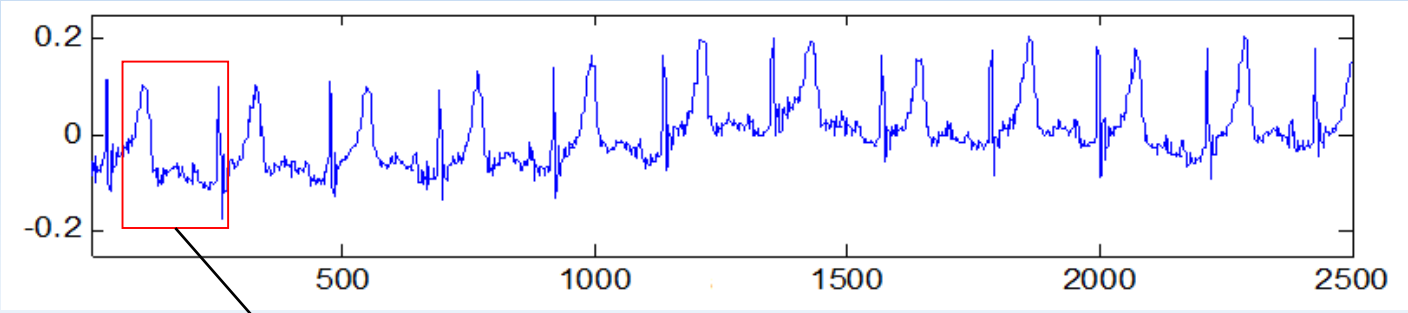


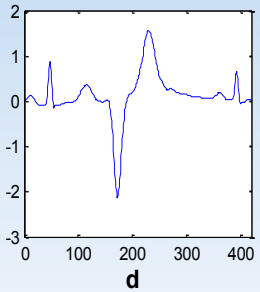


- Pick out sub-sequences with sliding window

Background： Subsequence Similarity Search

- Time series (Electrocardiogram) & pattern (heartbeat)

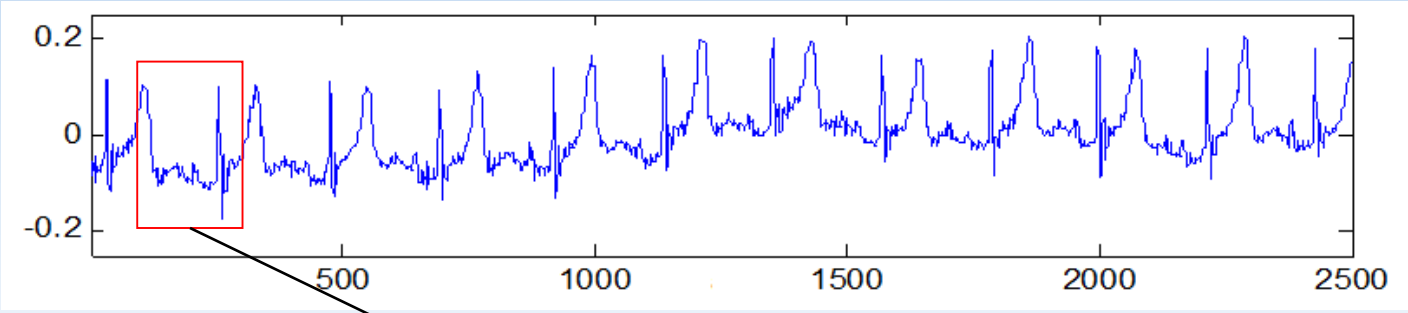


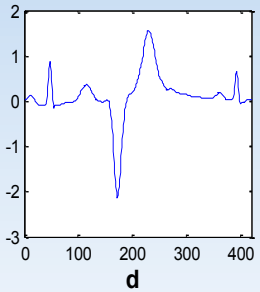


- Pick out sub-sequences with sliding window

Background： Subsequence Similarity Search

- Time series (Electrocardiogram) & pattern (heartbeat)

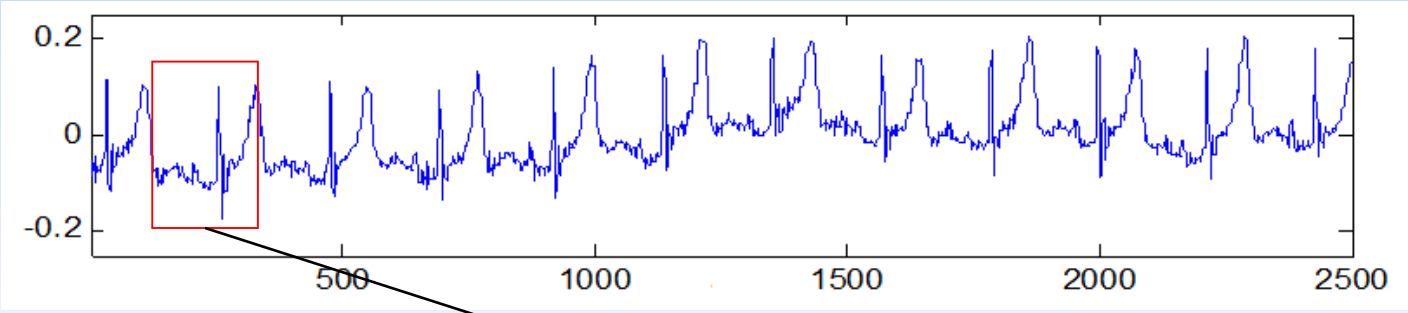


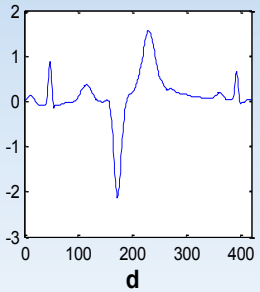


- Pick out sub-sequences with sliding window

Background： Subsequence Similarity Search

- Time series (Electrocardiogram) & pattern (heartbeat)

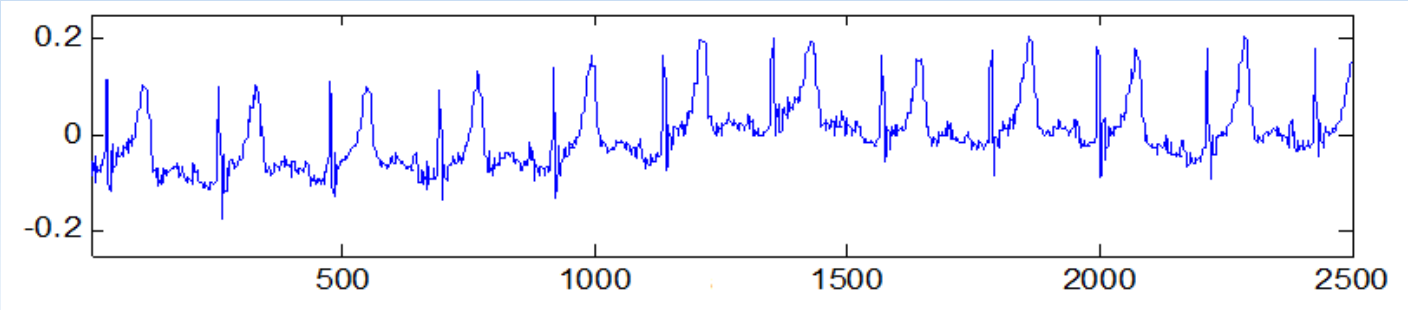


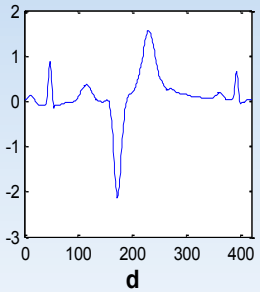


- Pick out sub-sequences with sliding window

Background： Subsequence Similarity Search

- Time series (Electrocardiogram) & pattern (heartbeat)





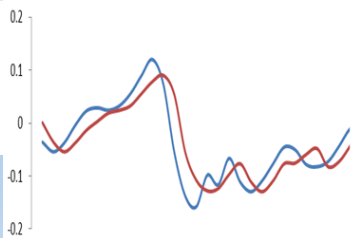
- Pick out sub-sequences with sliding window

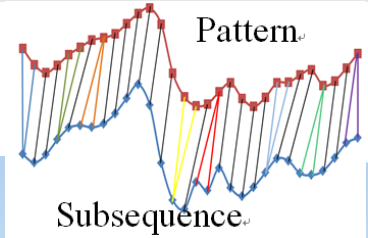
- Compare these sub-sequences with the pattern

- A proper distance metric is needed to define the similarity
  - e.g. Euclid distance

But there is increasing evidence that Dynamic Time Warping is the best distance metric in most domains.
DTW distance is defined as D（M, M）,as the following formula:
The time complexity of DTW is up to O(M square), so it is the bottleneck of many applications.

Background: DTW



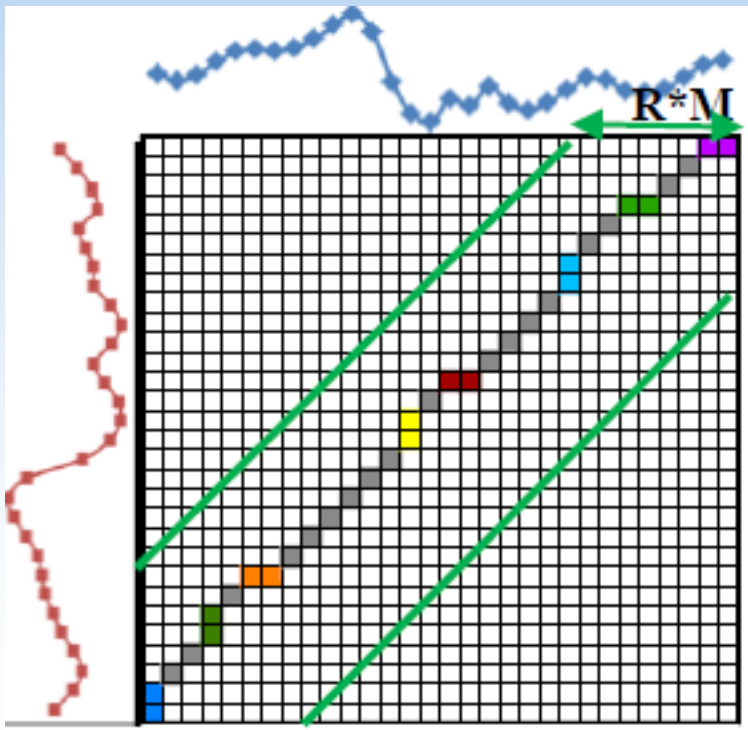


- Dynamic Time Warping (DTW) is the best distance measure in most domains[4]

$P = p_1, p_2, p_3 \ldots p_M$; $S = s_1, s_2, s_3 \ldots s_M$

$DTW\ distance = D(M, M)$;

$$D(i, j) = dist(s_i, p_j) + \min \begin{cases} D(i-1, j) \\ D(i, j-1) \\ D(i-1, j-1) \end{cases}$$



where $D(0,0) = 0$; $D(i, 0) = D(0, j) = \infty$, $1 \le i \le M, 1 \le j \le M$;

M is the pattern length.

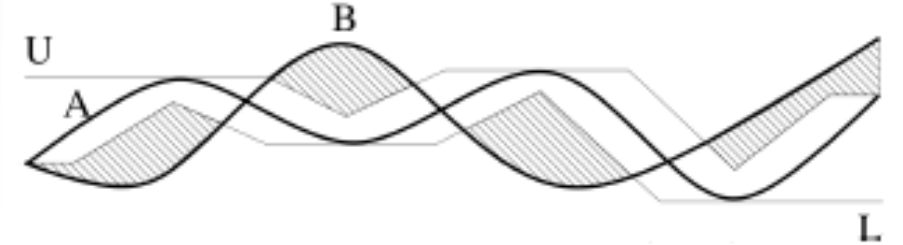
Either *absolute distance* or *square distance* can be used as *dist()*

A classic way to speed up subsequence similarity search based on an expensive distance is to use a cheap-to-compute distance to estimate the lower bound.

Background: Lower Bound

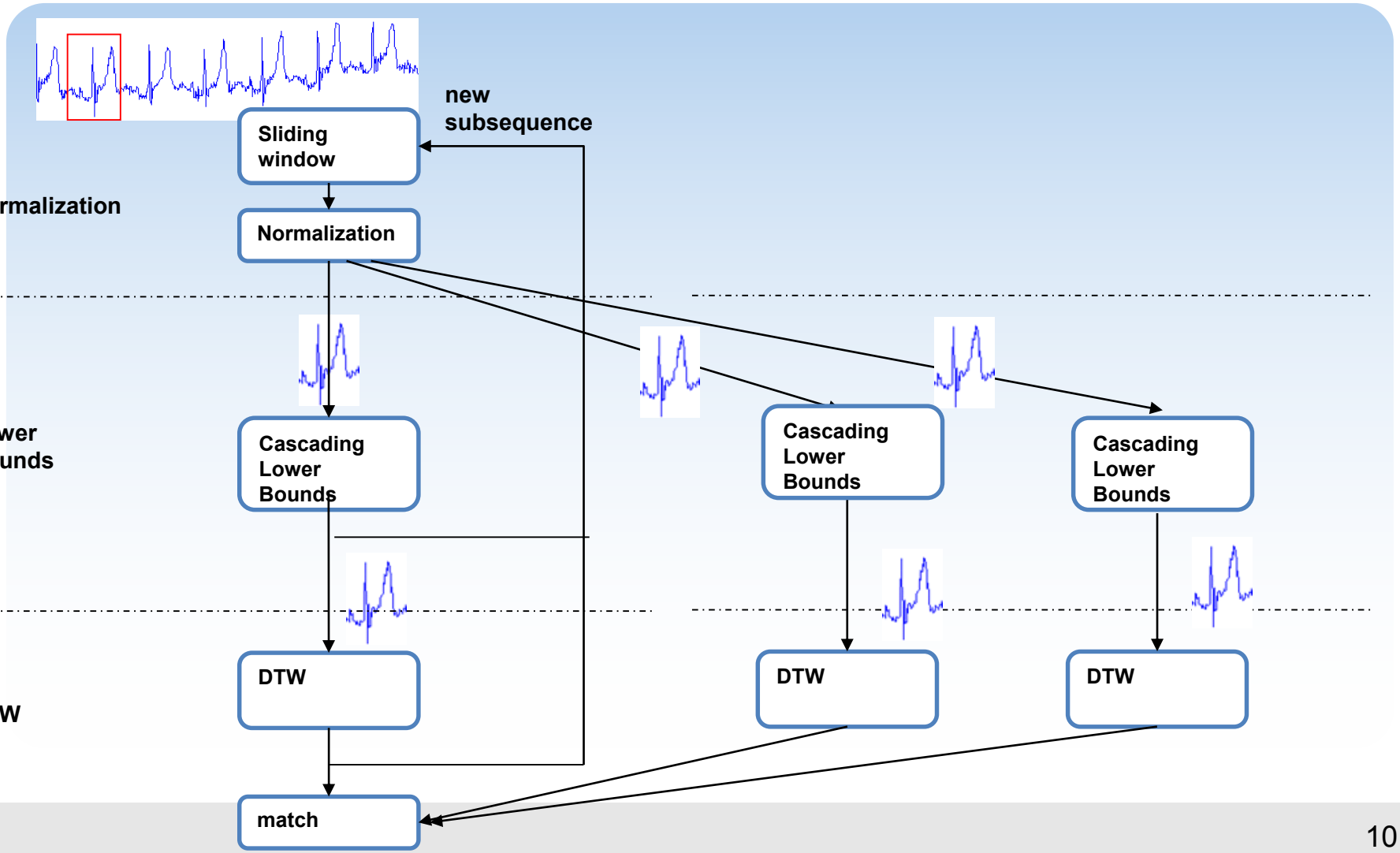
- LB technique tries to estimate the **lower bound** of DTW distance in a cheap way. (larger value=> less similarity)
- If lower bound exceeds the threshold, the real DTW distance will also exceed the threshold, so unpromising subsequences can be pruned off without the DTW calculation.

- LB_Kim: the first point, the last point, the max point and the min point.
- LB_Keogh: Construct an upper envelope and a lower envelope of the pattern, and the accumulated parts falling out of the envelopes is defined as the LB_Keogh



Most of the existing software implementations share the similar algorithm framework:
The existing software-based parallel implementations, such as multi-core, GPU, try to dispatch sub-sequences starting from different positions to different processing elements. This coarse-grained parallelism may lead to a heavy global data-transfer burden, as one sub-sequence may consist of many points.

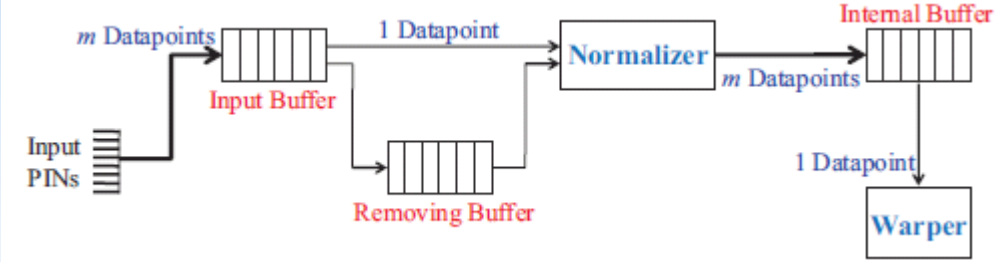
Related Work



It can not support on-line updating. The parameter modification, project re-compilation and FPGA re-configuration may take several hours in total.

Related Work

- The first and only work[2] using FPGA to accelerate DTW is generated by a C-to-RTL tool

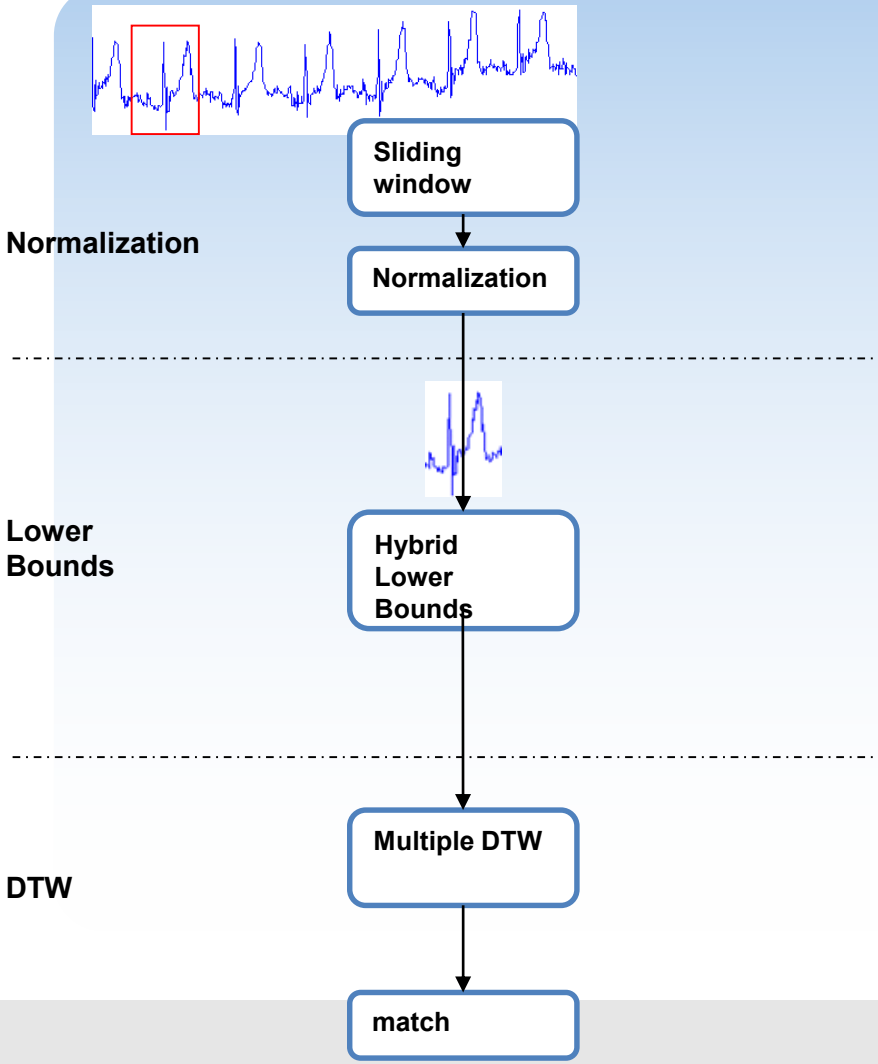


- *"The Warper module (DTW module) is implemented as a systolic array. A systolic array consists of data processing units connected in a matrix fashion"*

  - The lack of insight into FPGA limits the scalability and flexibility:
    - it can not support patterns of length larger than 1024.
    - It can not support on-line patterns updating, if the length of the new pattern is changed.

In the first phase, we use a incremental formula to do normalization, as the right figure.
The sub-sequence is subtracted by the mean, and divided by the standard deviation.
In the second phase, we use a hybrid lower bound consisting of LB_partial DTW, LB_Keogh and reversed LB_Keogh

Algorithm



$$\mu = \frac{1}{m}\left(\sum_{i=1}^{k} s_i - \sum_{i=1}^{k-m} s_i\right)$$

$$\sigma^2 = \frac{1}{m}\left(\sum_{i=1}^{k} s_i^2 - \sum_{i=1}^{k-m} s_i^2\right) - \mu^2$$

$$s_i = \frac{s_i - \mu}{\sigma}$$

- LB_partial DTW
- LB_Keogh/ reversed LB_Keogh

$U_i = max\ \{P_{i-R}, P_{i-R+1} \ldots P_{i+R-1}, P_{i+R}\};$

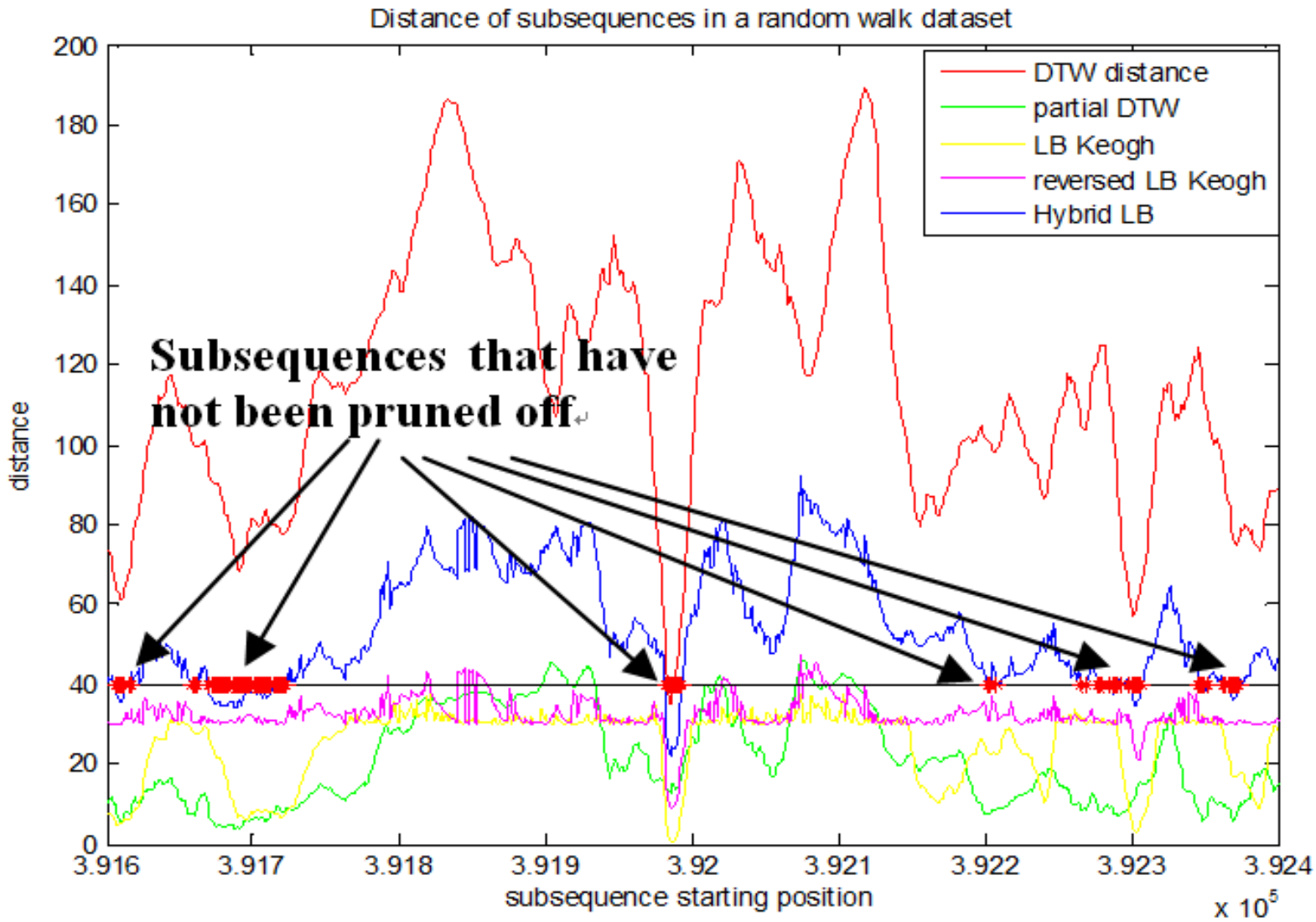
$L_i = min\ \{P_{i-R}, P_{i-R+1} \ldots P_{i+R-1}, P_{i+R}\};$

$$D_i = \begin{cases} S_i - U_i & if\ S_i > U_i \\ L_i - S_i & if\ L_i > S_i \\ 0 & else \end{cases}$$

$LB\_Keogh(P, S) = sum\{D_1, D_2 \ldots, D_Y\}$

Suppose we have finished lower bound calculation of all the sub-sequences,
The red line is the real DTW distance of all the sub-sequences, the blue line is the hybrid lower bound, and the threshold is 40.
We can find, the sub-sequences that have not been pruned off are usually located in a continue interval.

Algorithm: lower bound



If two paths grow to the same cell, the longer path is replaced by the shorter path,.
Note the final goal is to find the most similar sub-sequence, instead of the DTW distance of every sub-sequence.

Algorithm: SPRING

Y. Sakurai et al. propose a **computation-reuse algorithm called SPRING** [3]

- Only one point is different between two neighboring subsequences

- Merge N M-by-M matrixes into single N-by-M matrix. N paths grow at the same time

- N *

26	19	23	16	12
18	19	19	7	5
15	22	18	3	5
14	21	13	3	5
12	13	7	2	4

=>

26(1)	19(1)	23(2)	16(2)	12(2)	14(2)	12(2)	6(2)	14(2)	17(8)
18(1)	19(1)	19(2)	7(2)	5(2)	9(2)	6(2)	11(2)	10(8)	8(8)
15(1)	22(1)	18(2)	3(2)	5(2)	5(2)	8(2)	17(2)	7(8)	4(8)
14(1)	21(1)	13(2)	3(2)	5(2)	5(2)	8(2)	17(2)	6(8)	4(8)
12(1)	13(2)	7(2)	2(2)	4(2)	4(2)	7(2)	14(8)	4(8)	3(8)

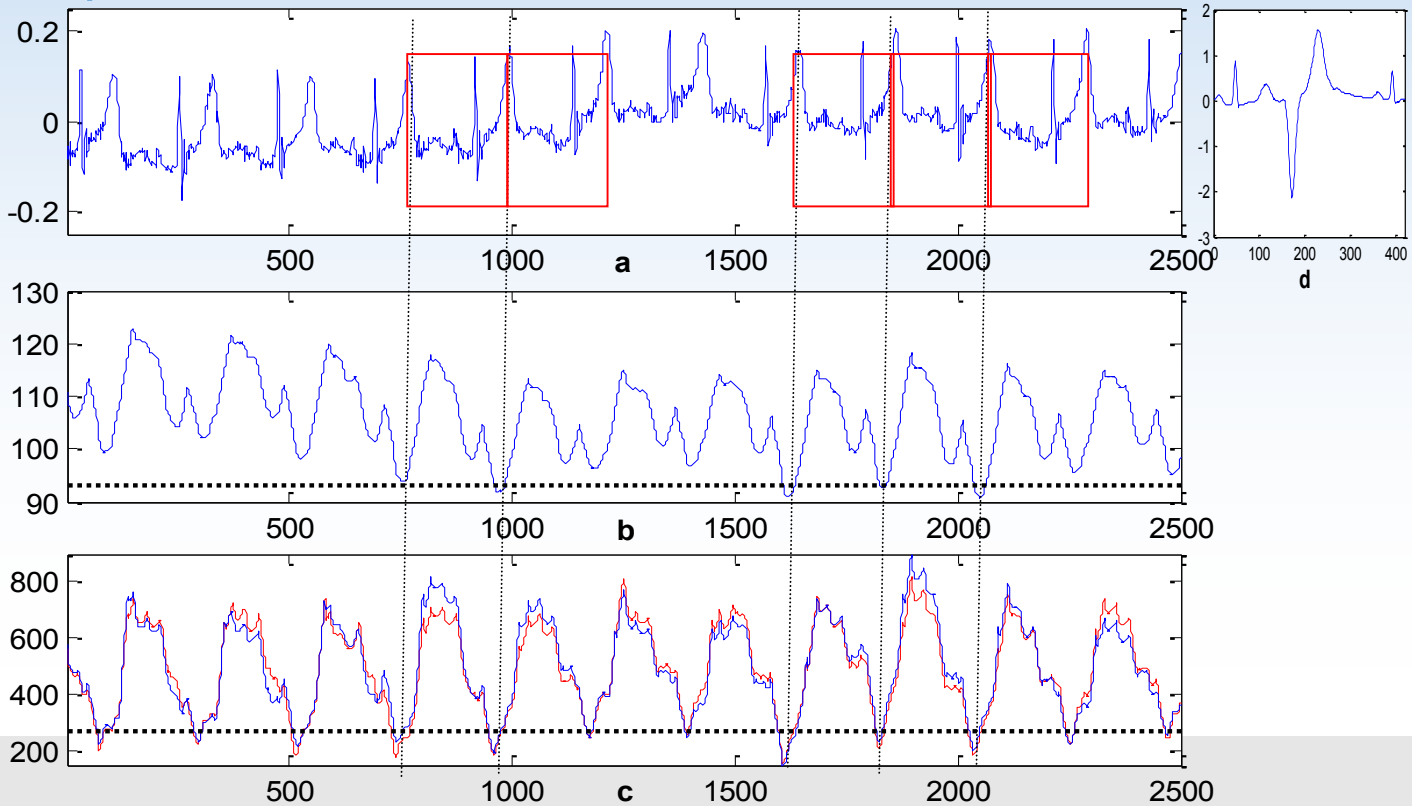
- It reduces the time complexity from O(N*M*M) to O(N*M)

- The sequences can't be normalized in stream

In our opinion, the false result is caused by the time-varying offset, instead of the time-invariant offset. With this motivation, we make a assumption that if the offset or the amplitude can be approximately seen as time-invariant among C continue sub-sequences, these C sub-sequences can be normalized as a group.
The third figure prove this assumption works well, both normalization lead to accurate recognition.

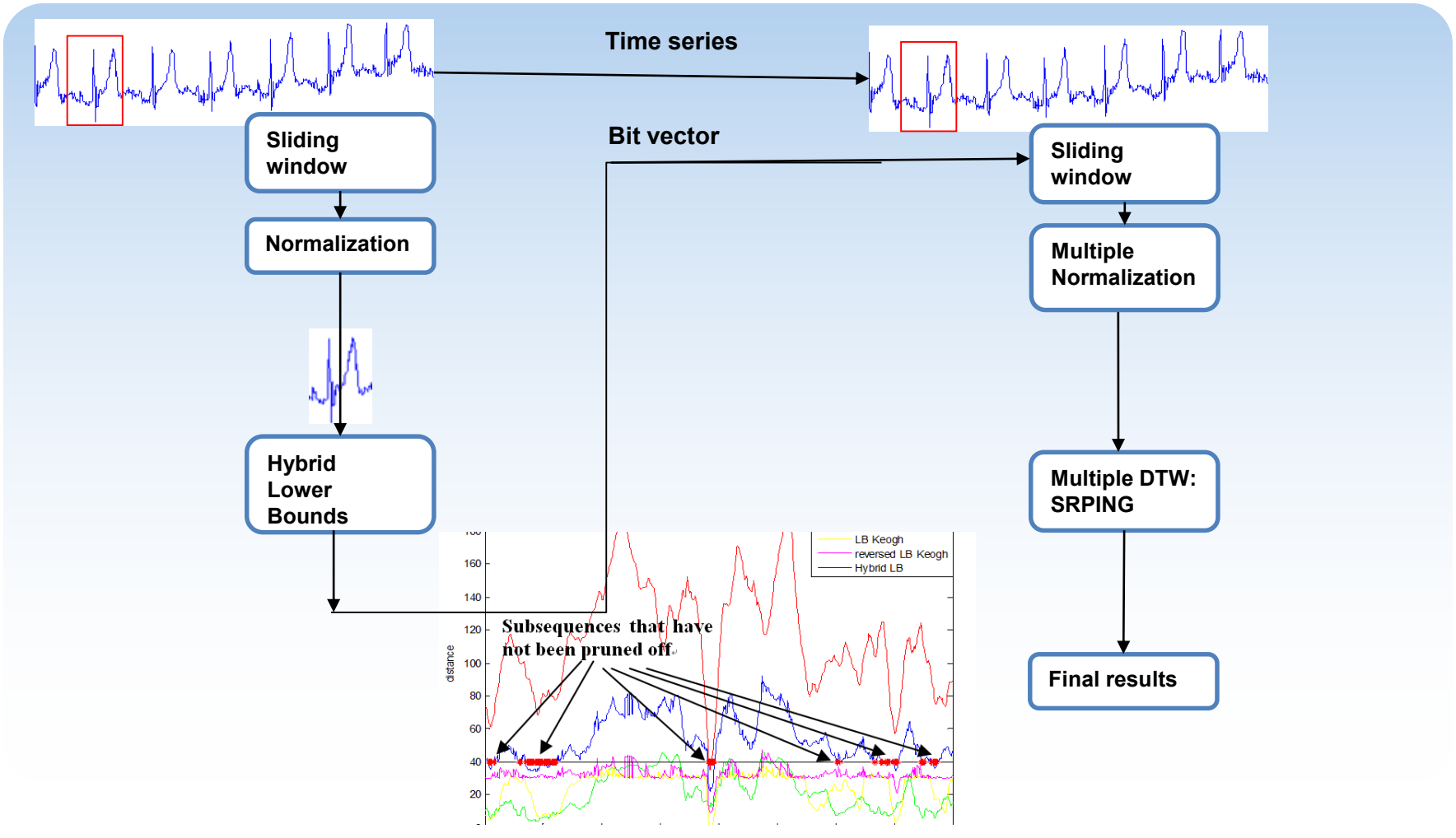
Algorithm: normalization

- **Assumption**: If the offset or the amplitude can be approximately seen as time-invariant among C continue sub-sequences, these C sub-sequences can be normalized at the same time.



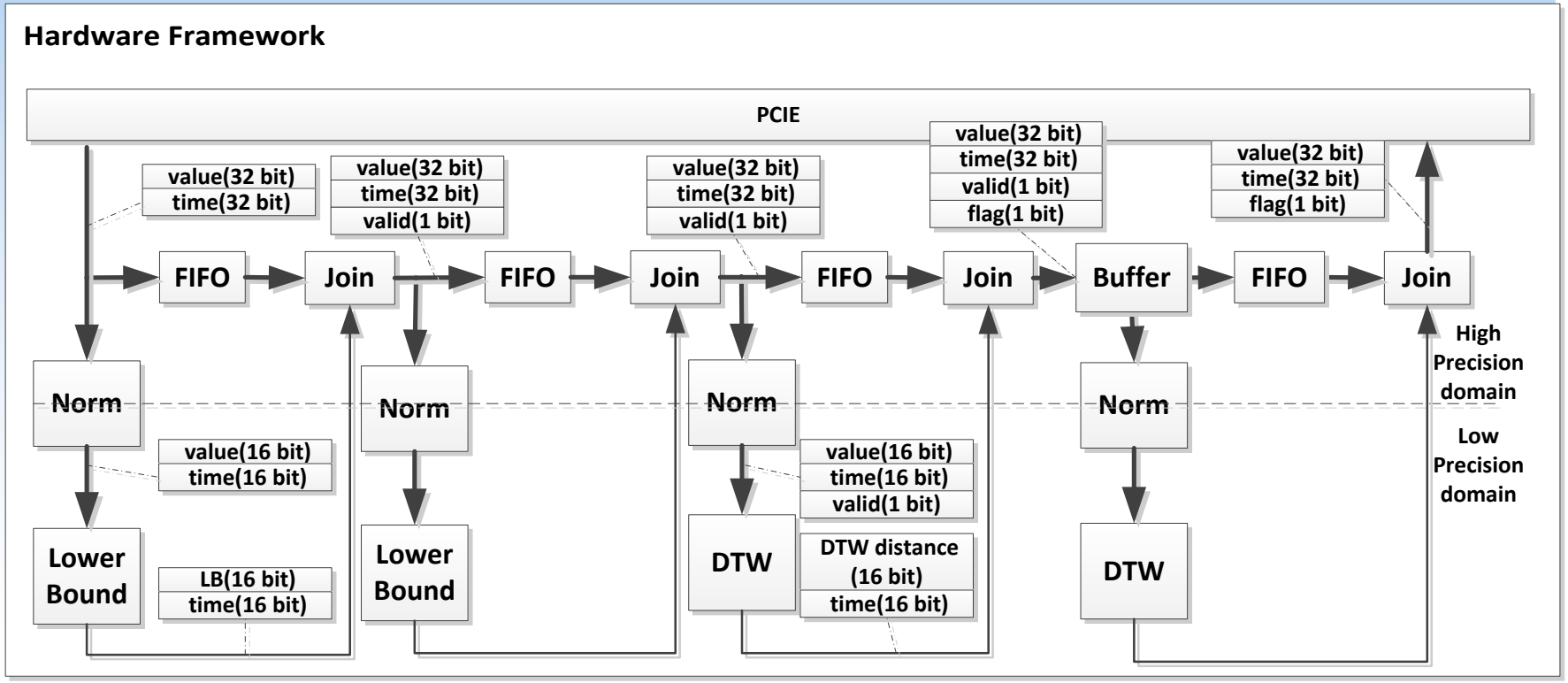
Then we use a bit vector to indicate which sub-sequence have not been pruned off by the Lower bound.
The second sliding window only need to pick out few continue sub-sequences that still need DTW calculation.
These sub-sequences will be normalized as a group, before the multiple DTW calculation.

Algorithm



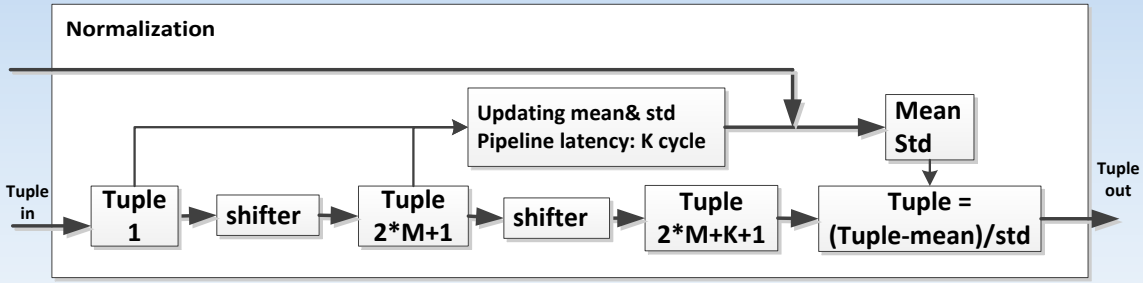
Then we come to the hardware framework.
Compared to the algorithm framework, We place duplicate modules for both lower bound and DTW to improve the throughput of the whole system.

Implementation

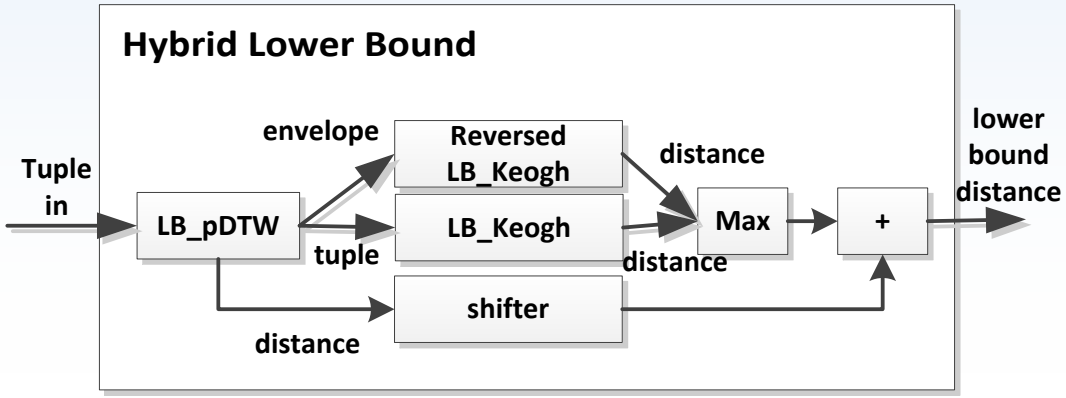


Implementation

- Normalization

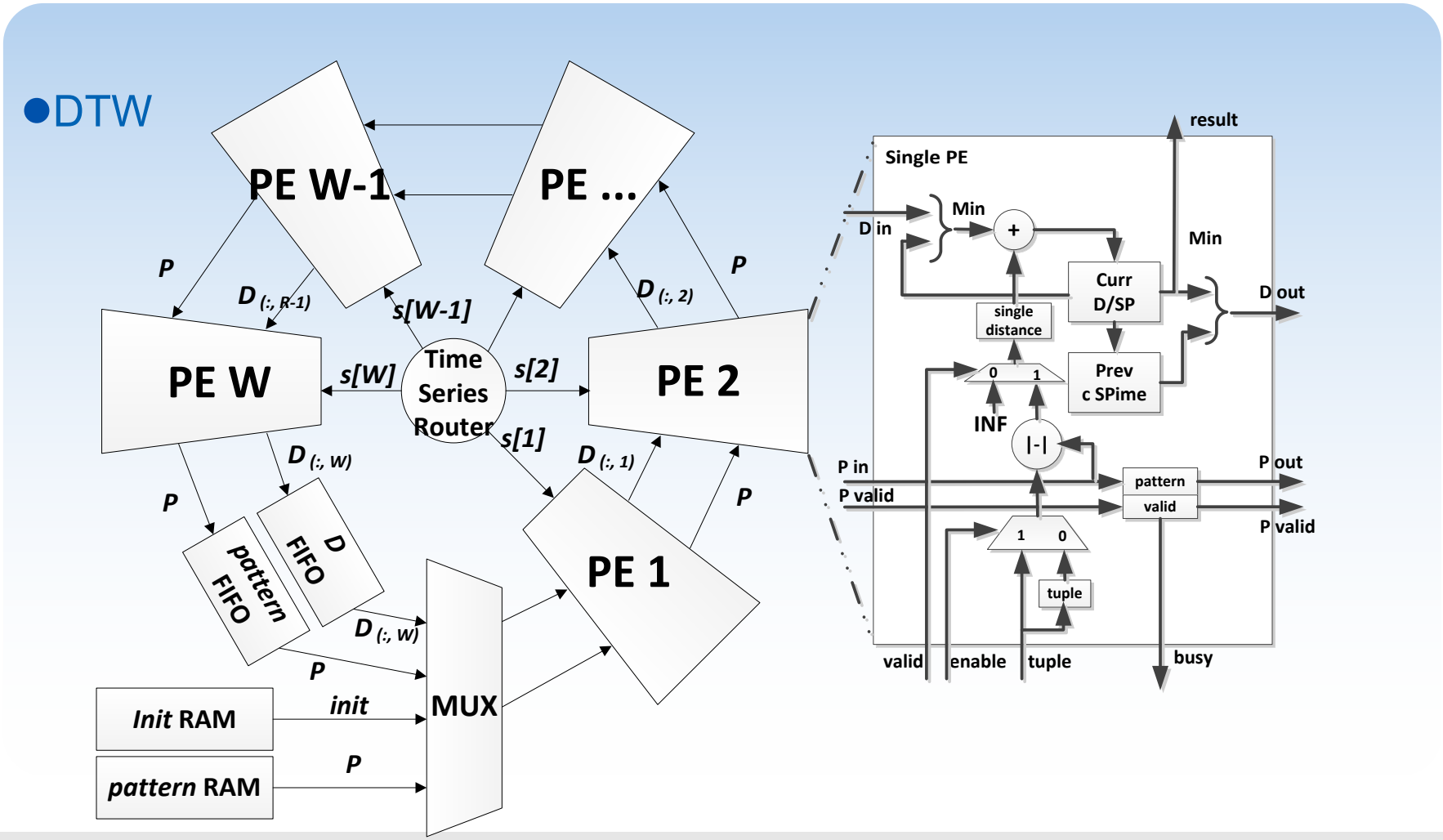


- Lower Bound



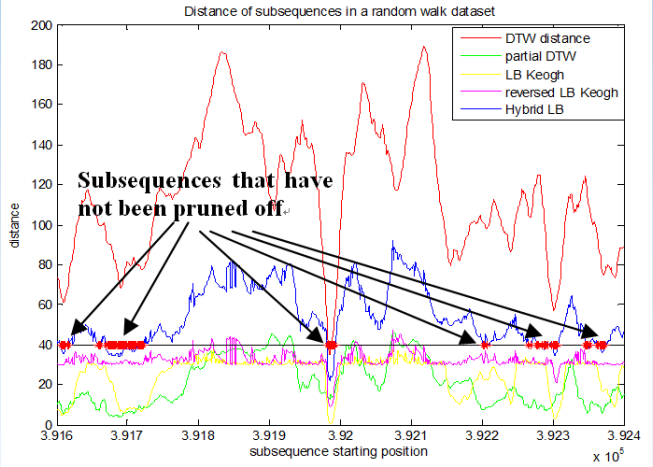
For DTW calculation, we propose a simple but effective structure: PE-ring.
A single PE is only used to calculate one column of the warping matrix, and all the PEs are connected one by one.
A multiplexer is used to send the pattern and the boundary condition into the ring. The FIFO is used to buffer the output of the last PE when all the PEs are busy.

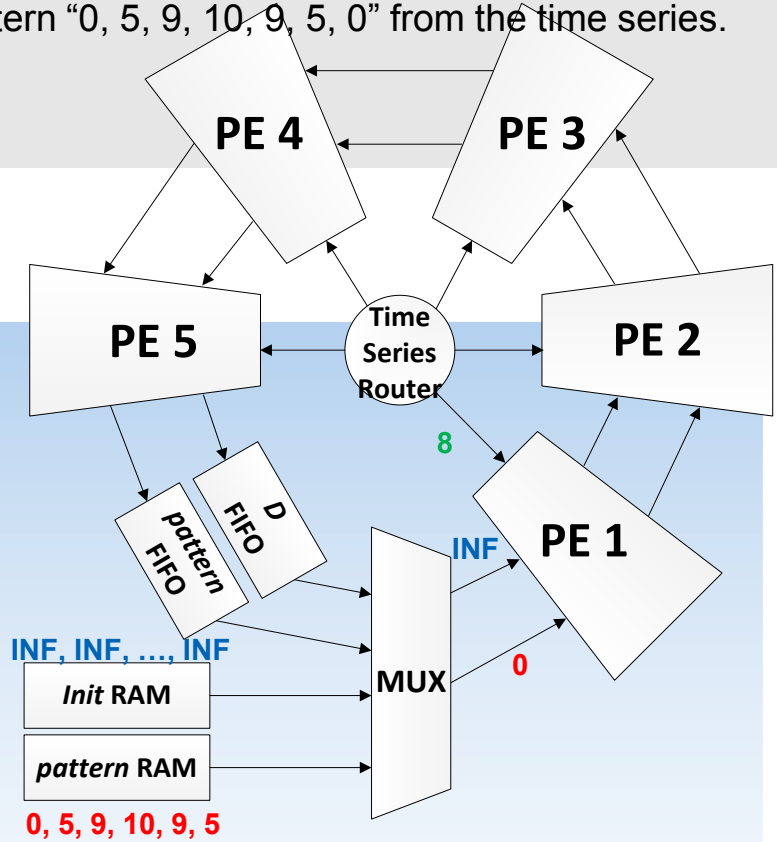
Implementation



Suppose we have 5 PEs in the ring, and we want to find the pattern “0, 5, 9, 10, 9, 5, 0” from the time series.

PE-ring for DTW



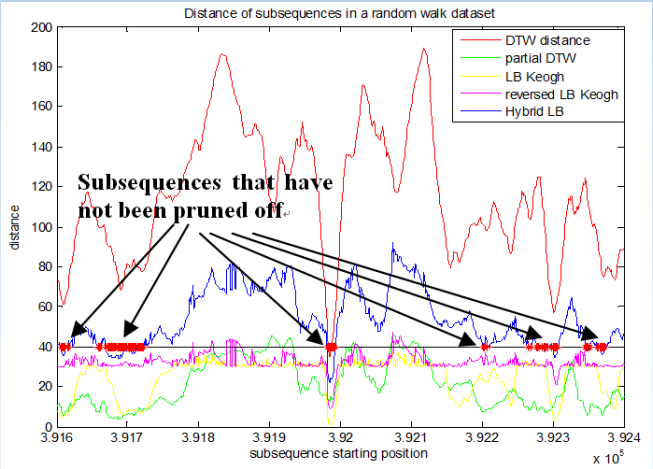


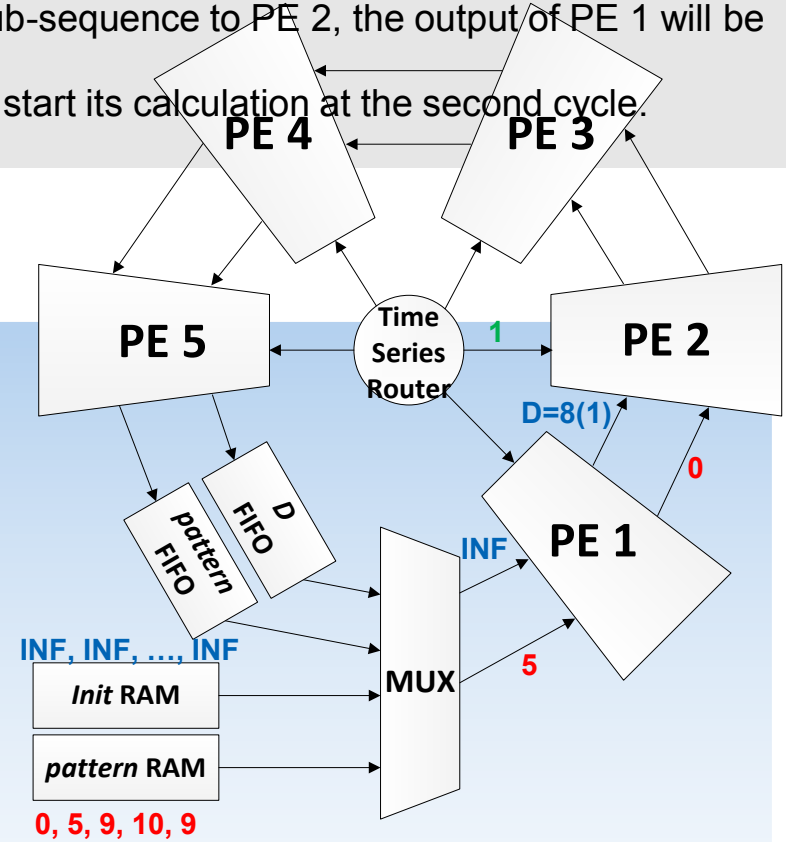
P7=0	INF														
P6=5	INF														
P5=9	INF														
P4=10	INF														
P3=9	INF														
P2=5	INF														
P1=0	INF	8(1)													
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1													

At the second cycle, the router sends the second point of the sub-sequence to PE 2, the output of PE 1 will be treated as the new boundary condition for PE 2.

PE 1 also send the first point of the pattern to PE2, so PE2 can start its calculation at the second cycle.

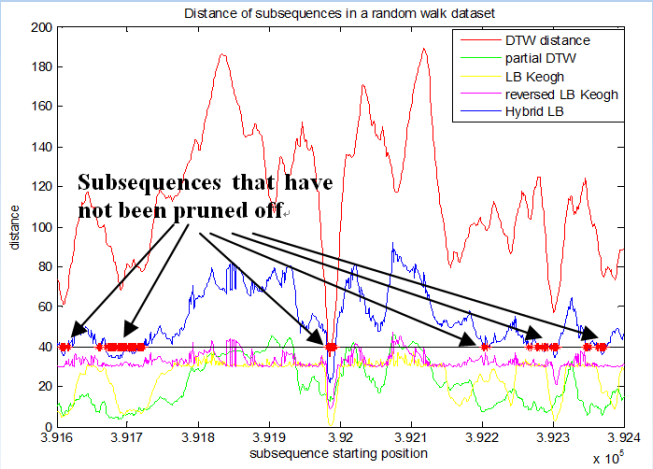
PE-ring for DTW

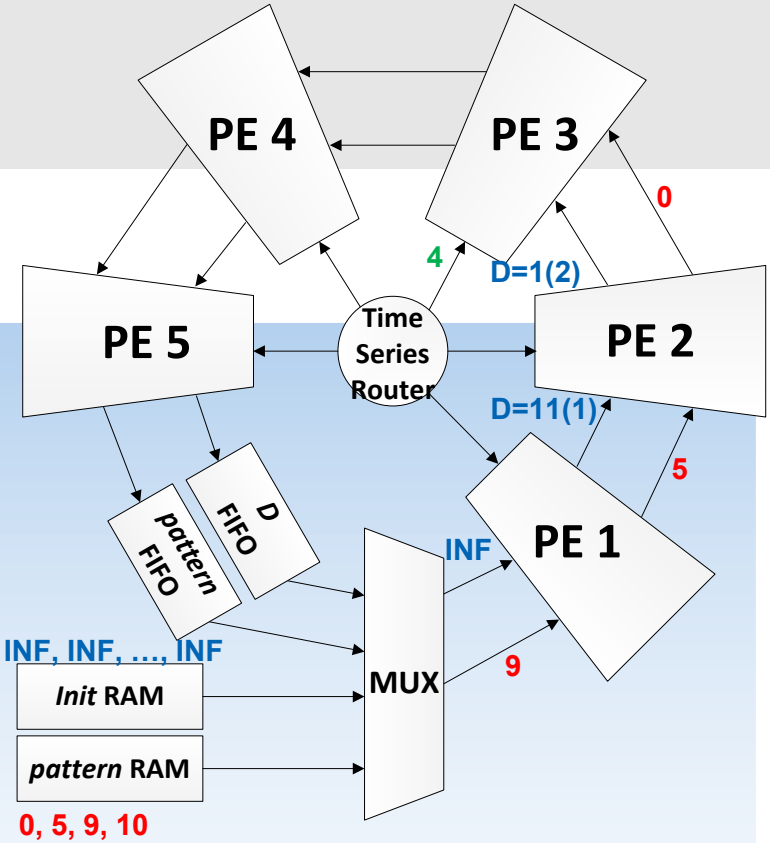




P7=0	INF														
P6=5	INF														
P5=9	INF														
P4=10	INF														
P3=9	INF														
P2=5	INF	11(1)													
P1=0	INF	8(1)	1(2)												
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2												

PE-ring for DTW

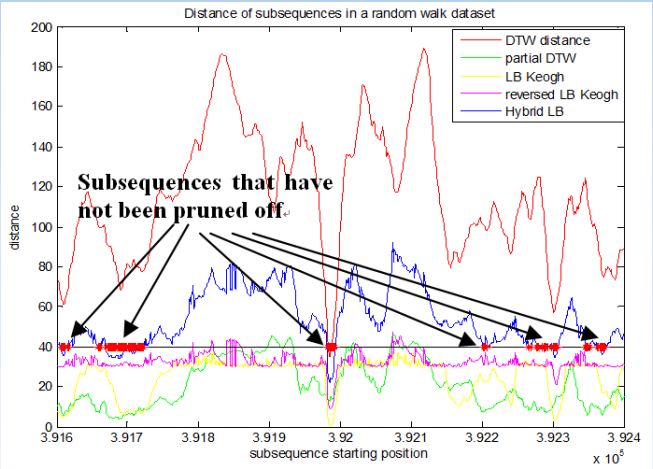


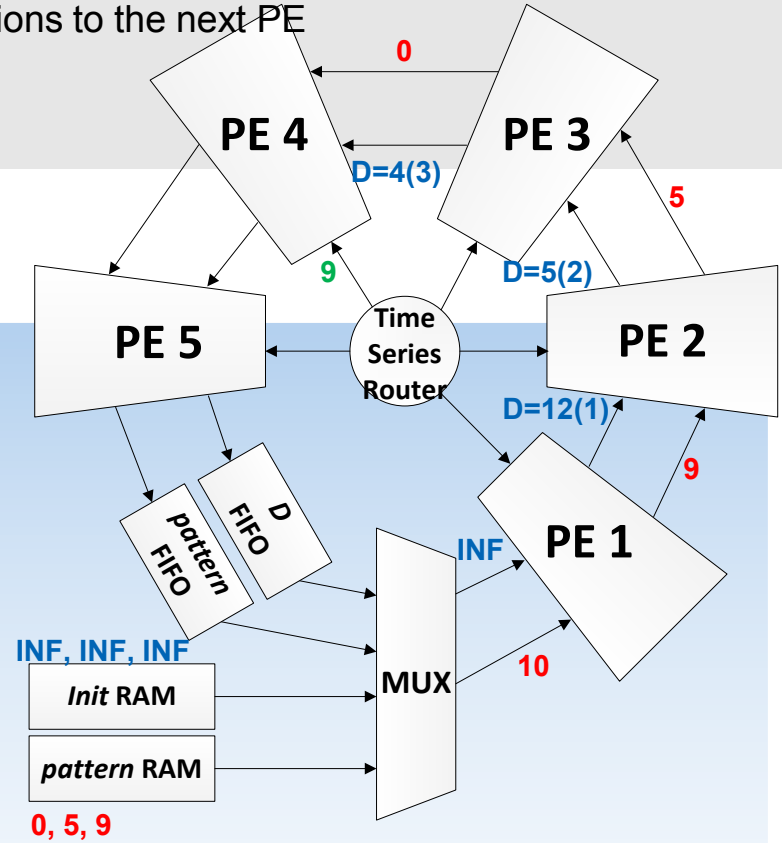


P7=0	INF														
P6=5	INF														
P5=9	INF														
P4=10	INF														
P3=9	INF	12(1)													
P2=5	INF	11(1)	5(2)												
P1=0	INF	8(1)	1(2)	4(3)											
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3											

All the PE transfer the pattern and the updated boundary conditions to the next PE

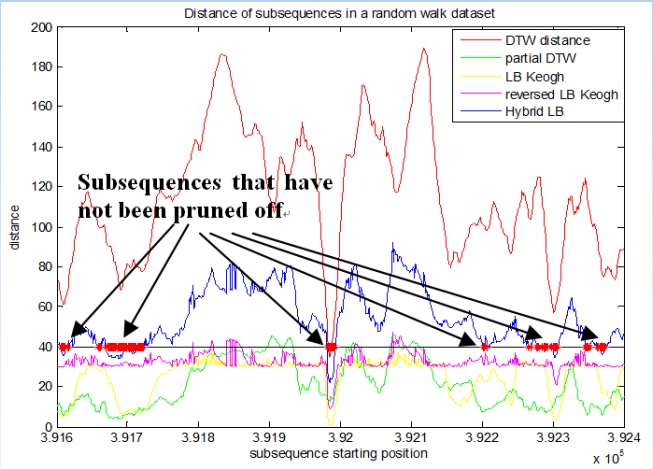
PE-ring for DTW

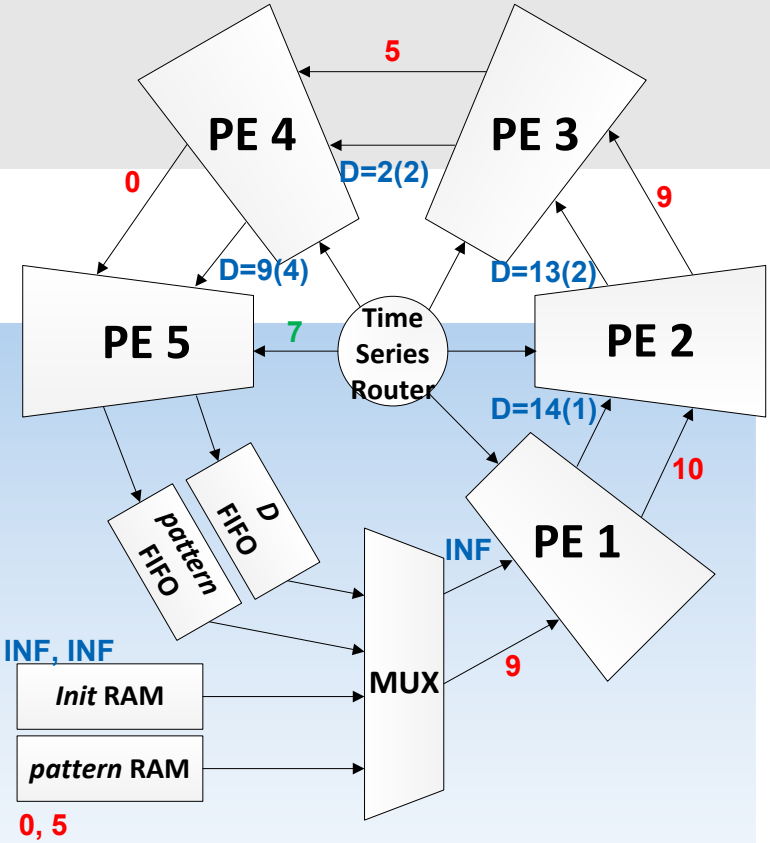




P7=0	INF														
P6=5	INF														
P5=9	INF														
P4=10	INF	14(1)													
P3=9	INF	12(1)	13(2)												
P2=5	INF	11(1)	5(2)	2(2)											
P1=0	INF	8(1)	1(2)	4(3)	9(4)										
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4										

PE-ring for DTW

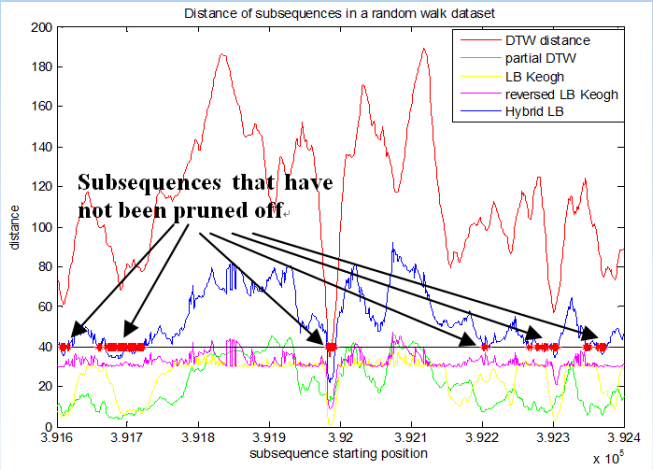


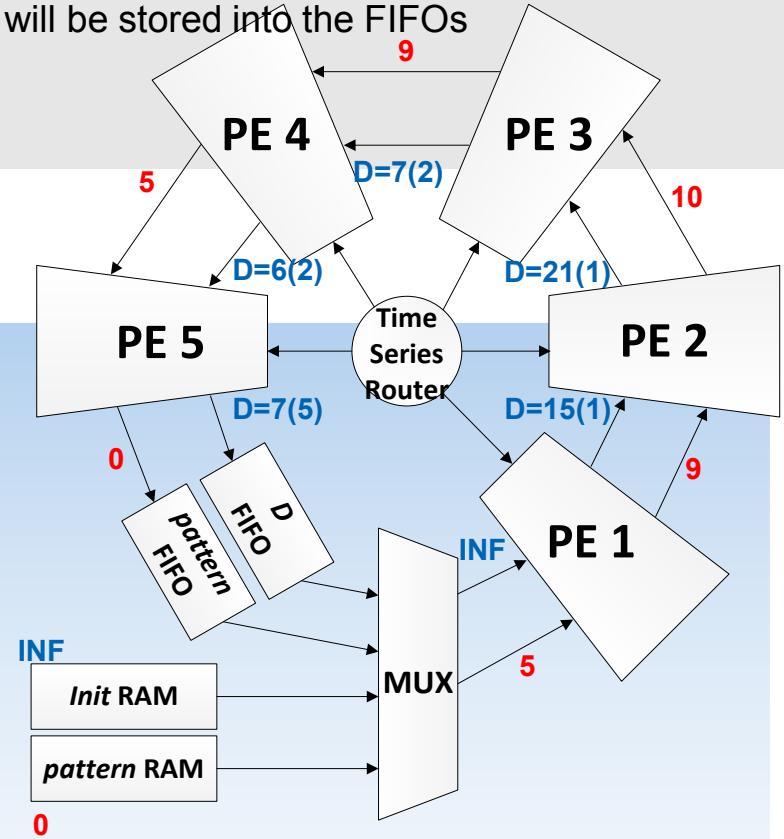


P7=0	INF														
P6=5	INF														
P5=9	INF	15(1)													
P4=10	INF	14(1)	21(1)												
P3=9	INF	12(1)	13(2)	7(2)											
P2=5	INF	11(1)	5(2)	2(2)	6(2)										
P1=0	INF	8(1)	1(2)	4(3)	9(4)	7(5)									
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4	PE5									

At the 6 cycle, as all the PEs are busy, the output of the last PE will be stored into the FIFOs

PE-ring for DTW

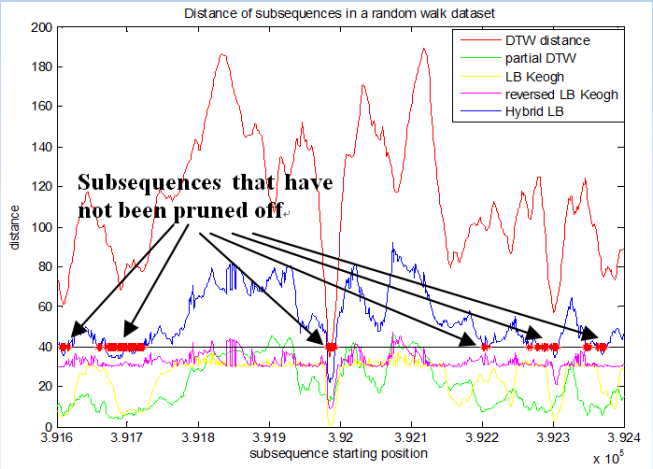


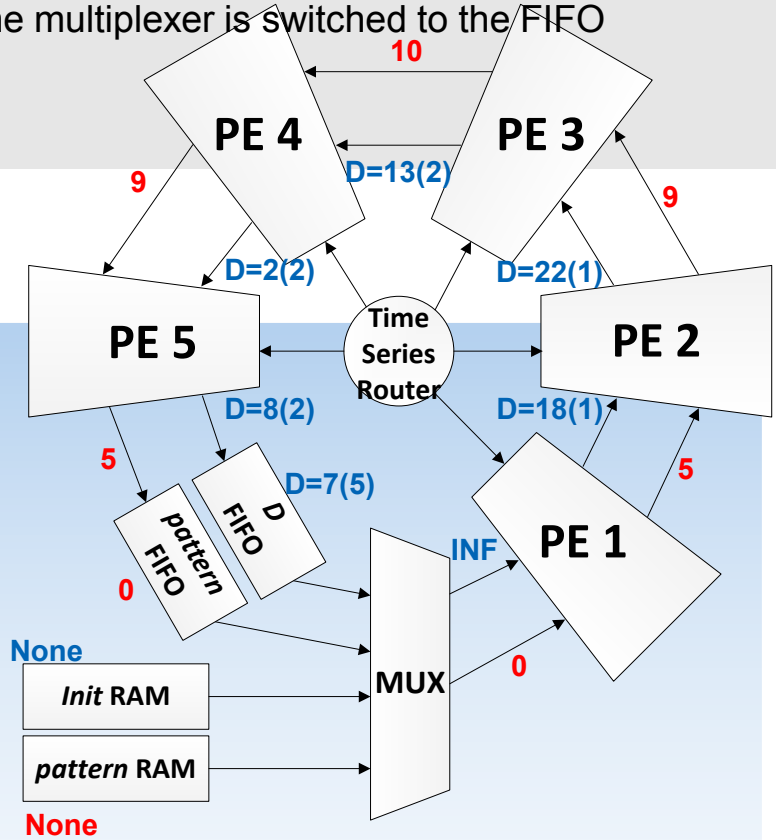


P7=0	INF														
P6=5	INF	18(1)													
P5=9	INF	15(1)	22(1)												
P4=10	INF	14(1)	21(1)	13(2)											
P3=9	INF	12(1)	13(2)	7(2)	2(2)										
P2=5	INF	11(1)	5(2)	2(2)	6(2)	8(2)									
P1=0	INF	8(1)	1(2)	4(3)	9(4)	7(5)									
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4	PE5									

At the 7th cycle, the pattern RAM and init RAM are empty, and the multiplexer is switched to the FIFO

PE-ring for DTW

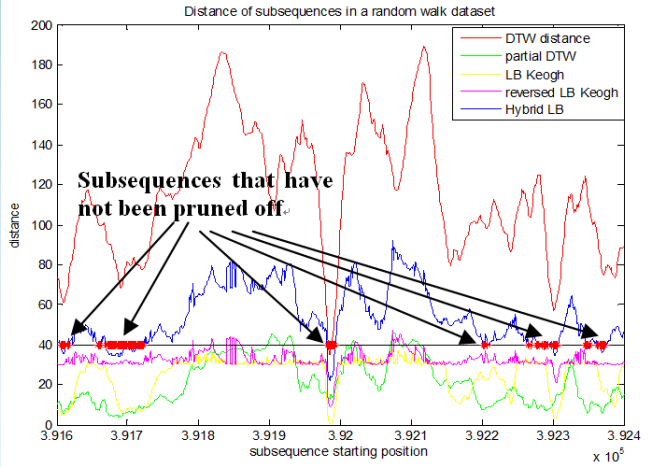


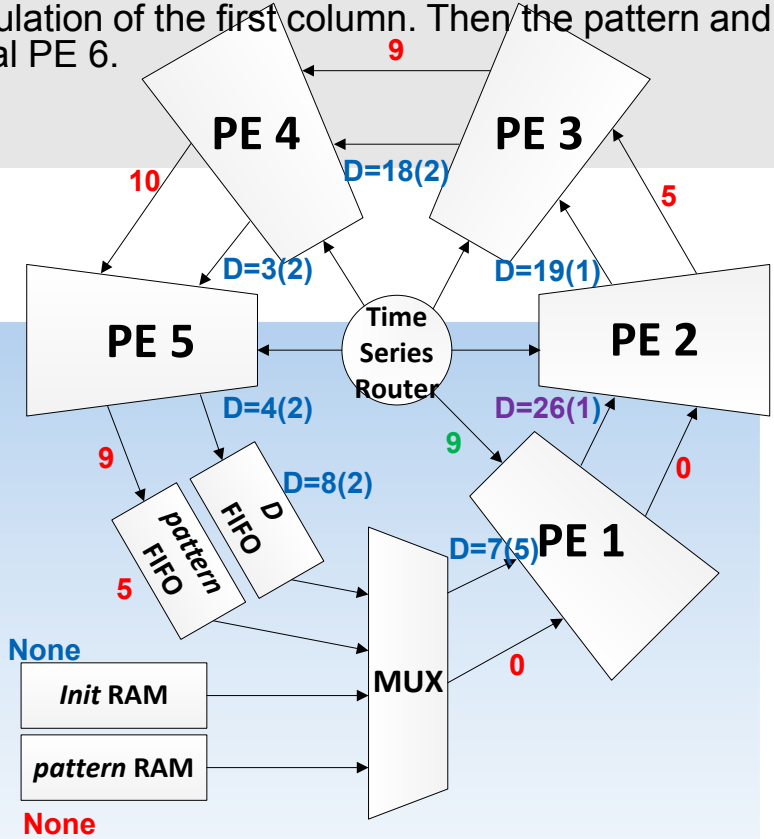


P7=0	INF	26(1)													
P6=5	INF	18(1)	19(1)												
P5=9	INF	15(1)	22(1)	18(2)											
P4=10	INF	14(1)	21(1)	13(2)	3(2)										
P3=9	INF	12(1)	13(2)	7(2)	2(2)	4(2)									
P2=5	INF	11(1)	5(2)	2(2)	6(2)	8(2)									
P1=0	INF	8(1)	1(2)	4(3)	9(4)	7(5)									
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4	PE5									

At the 8 cycle, the PE 1 is idle again because it finishes the calculation of the first column. Then the pattern and boundary in the FIFO is sent to PE 1, and PE 1 works as a virtual PE 6.

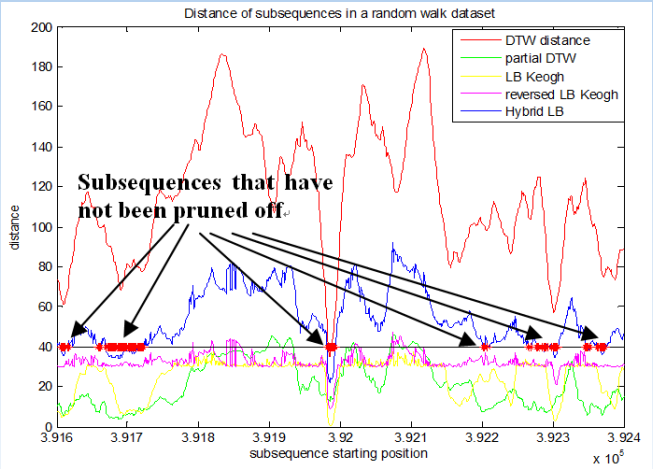
PE-ring for DTW

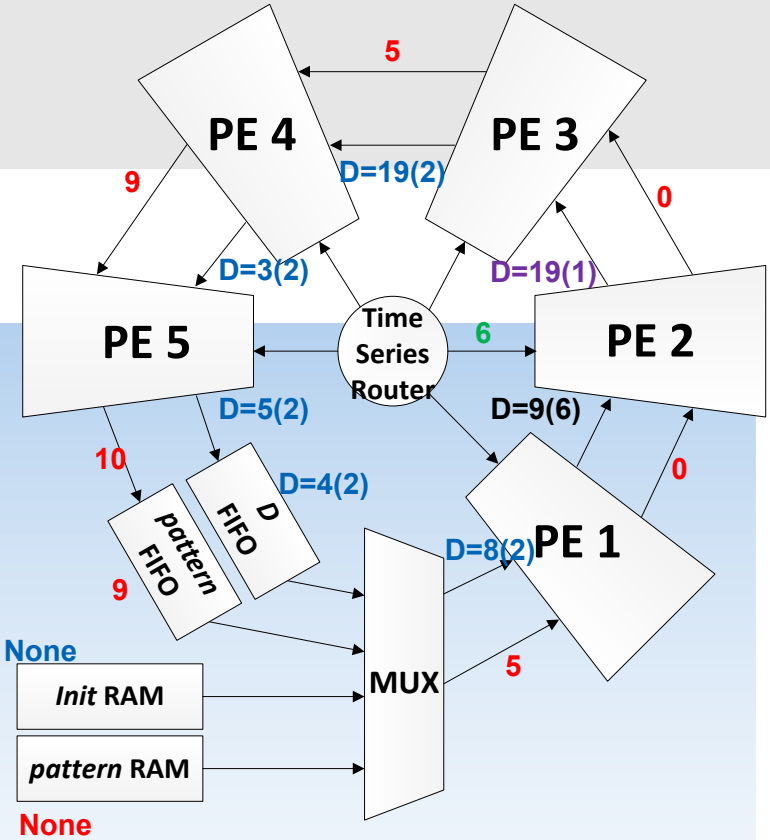




P7=0	INF	26(1)	19(1)												
P6=5	INF	18(1)	19(1)	19(2)											
P5=9	INF	15(1)	22(1)	18(2)	3(2)										
P4=10	INF	14(1)	21(1)	13(2)	3(2)	5(2)									
P3=9	INF	12(1)	13(2)	7(2)	2(2)	4(2)									
P2=5	INF	11(1)	5(2)	2(2)	6(2)	8(2)									
P1=0	INF	8(1)	1(2)	4(3)	9(4)	7(5)	9(6)								
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4	PE5	PE1								

PE-ring for DTW



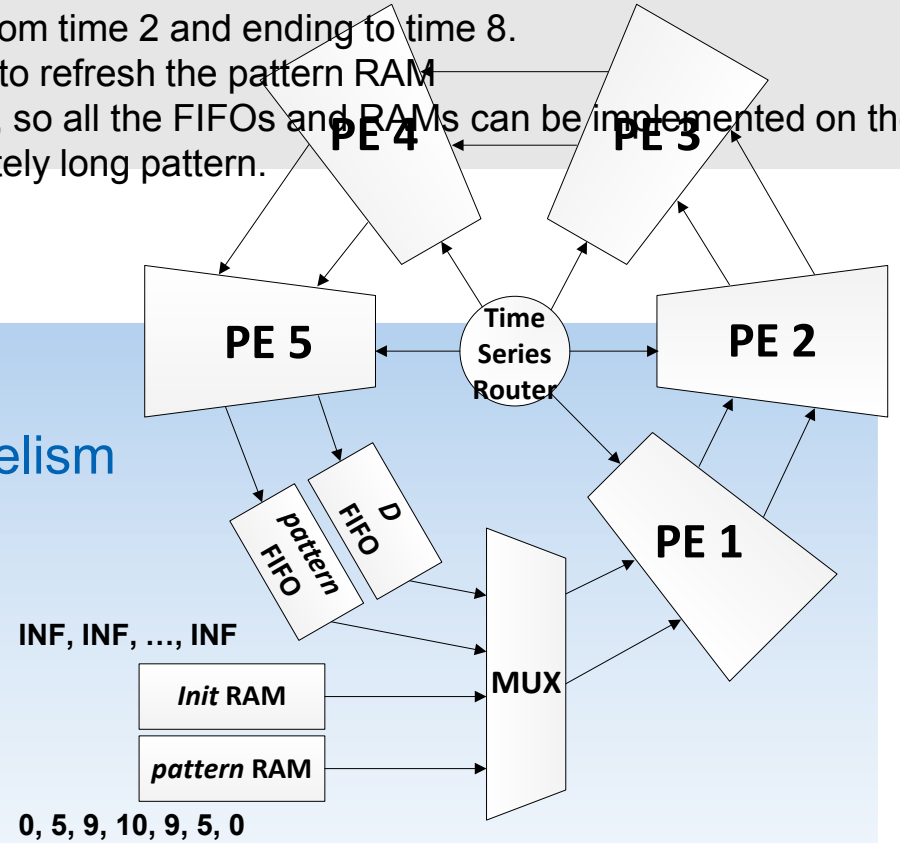


P7=0	INF	26(1)	19(1)	23(2)											
P6=5	INF	18(1)	19(1)	19(2)	7(2)										
P5=9	INF	15(1)	22(1)	18(2)	3(2)	5(2)									
P4=10	INF	14(1)	21(1)	13(2)	3(2)	5(2)									
P3=9	INF	12(1)	13(2)	7(2)	2(2)	4(2)									
P2=5	INF	11(1)	5(2)	2(2)	6(2)	8(2)	11(5)								
P1=0	INF	8(1)	1(2)	4(3)	9(4)	7(5)	9(6)	6(7)							
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4	PE5	PE1	PE2							

Finally, we find the most similar sub-sequence, starting from time 2 and ending to time 8.
When a new pattern of different is wanted, we only need to refresh the pattern RAM
There is no random memory access in the whole system, so all the FIFOs and RAMs can be implemented on the off-chip memory, so this PE-ring can support nearly infinitely long pattern.

PE-ring for DTW

- Fully exploit the fine-grained parallelism
- Flexible parallelism degree.
- Support on-line updating pattern of various lengths



P7=0	INF	26(1)	19(1)	23(2)	16(2)	12(2)	14(2)	12(2)	6(2)	14(2)	17(8)	11(8)	12(8)	14(8)	12(8)
P6=5	INF	18(1)	19(1)	19(2)	7(2)	5(2)	9(2)	6(2)	11(2)	10(8)	8(8)	5(8)	7(8)	9(8)	11(8)
P5=9	INF	15(1)	22(1)	18(2)	3(2)	5(2)	5(2)	8(2)	17(2)	7(8)	4(8)	7(8)	9(8)	11(8)	17(8)
P4=10	INF	14(1)	21(1)	13(2)	3(2)	5(2)	5(2)	8(2)	17(2)	6(8)	4(8)	7(8)	9(8)	11(8)	17(11
P3=9	INF	12(1)	13(2)	7(2)	2(2)	4(2)	4(2)	7(2)	14(8)	4(8)	3(8)	6(8)	8(8)	10(11	11(14
P2=5	INF	11(1)	5(2)	2(2)	6(2)	8(2)	11(5)	7(7)	5(8)	3(8)	7(8)	7(8)	8(11)	9(13)	5(14)
P1=0	INF	8(1)	1(2)	4(3)	9(4)	7(5)	9(6)	6(7)	0(8)	8(9)	9(10)	6(11)	7(12)	7(13)	3(14)
value		8	1	4	9	7	9	6	0	8	9	6	7	7	3
time		1	2	3	4	5	6	7	8	9	10	11	12	13	14
PE		PE1	PE2	PE3	PE4	PE5	PE1	PE2	PE3	PE4	PE5	PE1	PE2	PE3	PE4

Any PE in the ring can be removed without causing functional errors and the saved resource can be allocated to other modules. If there is abundant resource, a new PE can be directly inserted into the ring to improve the performance.

This PE-ring can also be applied to lower bound. The pattern is replaced by the envelopes, the path only grow along the anti-diagonal.
As lower bound has different performance in different dataset, we can insert more Pes into the bottleneck module to improve the throughput of the whole system.

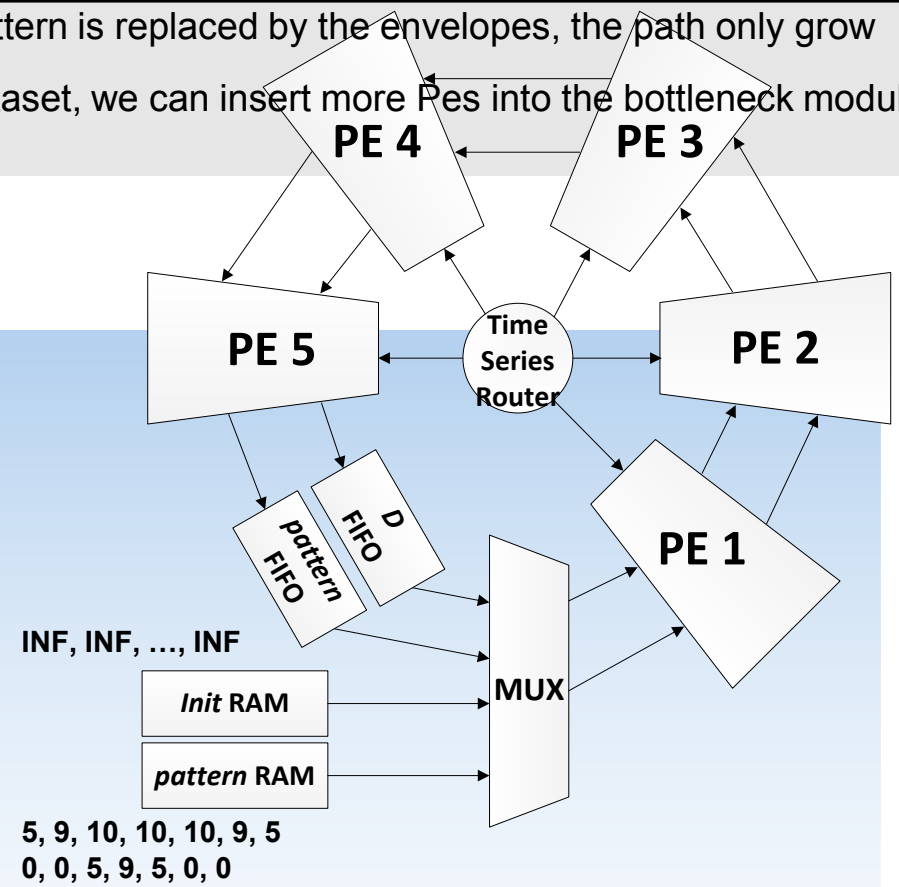
PE-ring for LB_Keogh

$$U_i = max\{P_{i-R}, P_{i-R+1} \ldots P_{i+R-1}, P_{i+R}\};$$

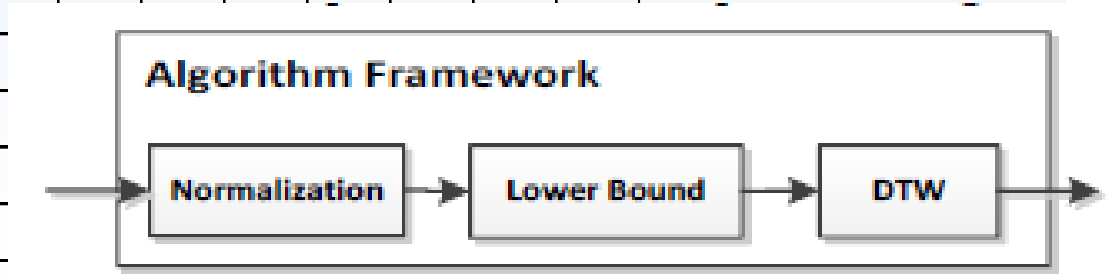
$$L_i = min\{P_{i-R}, P_{i-R+1} \ldots P_{i+R-1}, P_{i+R}\};$$

$$D_i = \begin{cases} S_i - U_i & if\ S_i > U_i \\ L_i - S_i & if\ L_i > S_i \\ 0 & else \end{cases}$$

$$LB(P_{1,Y}, S_{1,Y}) = sum\{D_1, D_2 \ldots, D_Y\}$$



U7=5	L7=0	INF													
U6=9	L6=0	INF													
U5=10	L5=5	INF	INF												
U4=10	L4=9	INF	INF	INF											
U3=10	L3=5	INF	INF	INF	4(1)										
U2=9	L2=0	INF	INF	3(1)	0(2)	0(3)									
U1=5	L1=0	INF	3(1)	0(2)	0(3)	4(4)	2(5)								
value			8	1	4	9	7	9	6	0	8	9	6	7	7
time			1	2	3	4	5	6	7	8	9	10	11	12	13
PE			PE1	PE2	PE3	PE4	PE5								



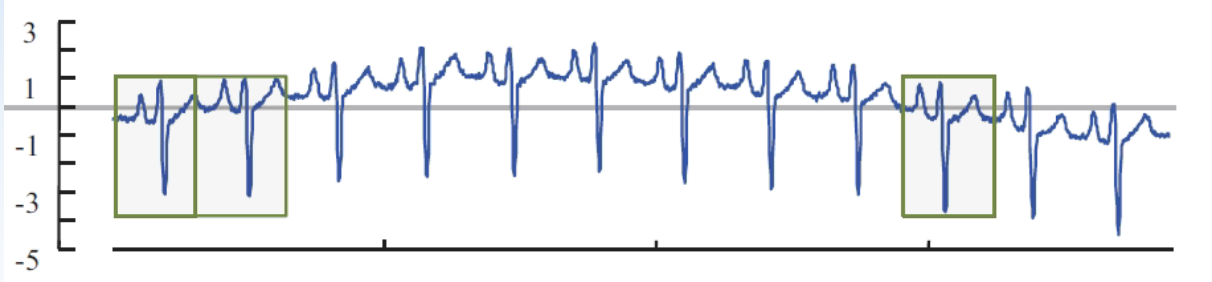
Experiment

- FPGA board: Altera/Terasic DE4
  - Combinational ALUTs 362,568/424,960 (85%)
  - Dedicated logic registers 230,160/424,960 (54%)
  - Memory bits 1,902,512/21,233,664 (9%)
  - frequency: 150MHz

- CPU: intel i7-930 2.8GHz,
  16 GB DDR3 1333MHz ,
  windows 7

Experiment

- Software: T. Rakthanmanon [10] (the best paper of the sigkdd 2012)
- Dataset 1: medical data
  - This dataset has about 8G points, and we need to find a pattern of length 421 with R = 5%



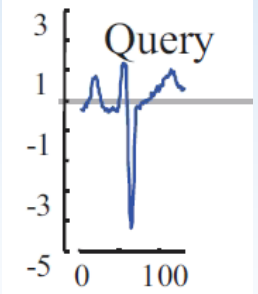


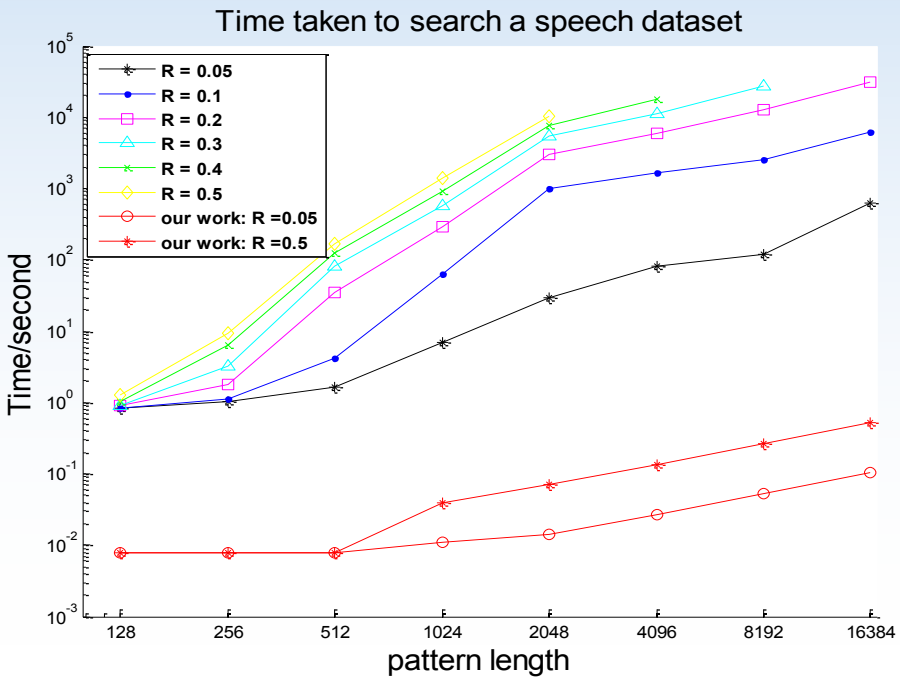
Table 1: Time taken to search one year of ECG data

	UCR_DTW[10]	Our work	Speedup
ECG	18.0 minutes	56 seconds	19.28

They claim that the constraint should be as small as about 5% to prevent pathological warping, while some other researchers insist that there should be no (or larger) constrain to improve the fault tolerance. In our opinion, the constraint *R* is an application-dependent parameter. Though we test their program in cases that *R* is set to be a large one in some dataset, we only show the result as a comparison of computation power in extreme cases, not standing for that the larger constraint can improve the high level accuracy in these applications.

Experiment

- Dataset 2: speech recognition
  - We download the CMU_ARCTIC speech synthesis databases, and construct a speech of 1 minute(1 million points) by splicing together the first 21 utterances of all the 1132 utterances
- Two orders of magnitude (0.827s/0.008s =103) speedup In the case that pattern length is 128, R=5%
- Four orders of magnitude (31716s/0.5s=63432) speedup in the case that pattern length is 16384, R=20%



Experiment

- FPGA and GPU: D.Sart [2]
- Dataset : Electrical Penetration Graph (EPG) signal.
  - This data set has 1,499,000 points, and the pattern length is 360, no constraint (R=100%).

EPG data set	D. Sart [2]	Our work	speedup
GPU	80.39s		7398
FPGA	2.24s	0.011s	203

- More datasets can be seen in the paper

- **Thank you!**

Profiling

- Random walk：1M tuple，pattern length = 128，R=5%

Function / Call Stack	Hardware Event ... CPU_CLK_U...	Hardware Eve... INST_RETIRED...	CPI Rate	Retire Stalls	LLC Miss	Execution Stalls	LLC Load Misses Ser...	Branch Mispredict	Instruction Starvation
main	2,402,201,204	3,918,415,127	0.613						
lb_keogh_cumulative	1,158,922,723	1,065,721,254	1.087						
dtw	138,496,969	115,395,583	1.200						
lb_keogh_data_cumulative	131,618,505	107,358,890	1.226						
lower_upper_lemire	123,818,682	124,413,794	0.995						
lb_kim_hierarchy	107,450,509	79,992,155	1.343						
allrem	65,206,999	51,874,699	1.257						
LdrGetDllHandleEx	2,204,827	0							

- Random walk：1M tuple，pattern length = 128，R=20%

Function / Call Stack	Hardware Ev... CPU_CLK...	Hardware E... INST_RETIR...	CPI Rate	Retire Stalls	LLC Miss	Execution Stalls	LLC Load Misses Ser...	Branch Mispredict	Contested Accesses	Instruction Starvation
main	3,550,613,111	4,774,769,541	0.744							
dtw	3,360,470,893	3,470,674,105	0.968							
lb_keogh_cumulative	787,781,793	800,736,978	0.984							
lb_keogh_data_cumulative	714,456,962	748,348,230	0.955							
lower_upper_lemire	154,867,233	155,769,308	0.994							
lb_kim_hierarchy	125,824,356	142,588,112	0.882							
allrem	26,120,521	28,350,488	0.921							
[Import thunk CIsqrt]	4,803,942	4,777,379	1.006							
NtConnectPort	2,025,026	0								