Implementation of the Smith-Waterman Algorithm On FPGA

B07705049 林稜凱

https://github.com/Leng-Kai/Smith-Waterman-HLS

Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Introduction to Smith-Waterman Algorithm

```
GAP = -1
MATCH = 2
MISMATCH = -1

str1 = "ATCCT"
str2 = "ACCTT"
```

```
alignment #1 :

  str1:  A  T  C  C  T
  str2:  A  C  C  T  T
         2 -1  2 -1  2
  total: 4 (3*MATCH + 2*MISMATCH)
```

```
alignment #2 :

  str1:  A  T  C  C  T  _
  str2:  A  _  C  C  T  T
         2 -1  2  2  2 -1
  total: 6 (4*MATCH + 2*GAP)
```

Introduction to Smith-Waterman Algorithm

		G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0
G	0	2	1	0	0	0	2	2
T	0	1	4	3	2	1	1	1
T	0	0	3	6	5	4	3	2
G	0	2	2	5	5	...		
A	0							
C	0							
T	0							

```
GAP = -1
MATCH = 2        // s(x, y) if x == y
MISMATCH = -1    // s(x, y) if x != y

dp[i][j]
= max
{
  dp[i-1][j-1] + s(a[i], b[j]),
  dp[i-1][j]   + GAP,
  dp[i][j-1]   + GAP,
  0
}
```

Introduction to Smith-Waterman Algorithm

		G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0
G	0	2	1	0	0	0	2	2
T	0	1	4	3	2	1	1	1
T	0	0	3	6	5	4	3	2
G	0	2	2	5	5	...		
A	0							
C	0							
T	0							

+2 ↘, -1 ↓, -1 →

```
GAP = -1
MATCH = 2        // s(x, y) if x == y
MISMATCH = -1    // s(x, y) if x != y

dp[i][j]
= max
{
↘ dp[i-1][j-1] + s(a[i], b[j]),
↓ dp[i-1][j]   + GAP,
→ dp[i][j-1]   + GAP,
  0
}
```

Time complexity = O(mn)

Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Initial Implementation

```
20  void compute_matrices(
21      char *string1, char *string2,
22      int *max_index, int *similarity_matrix, short *direction_matrix)
23  {
24  #pragma HLS INTERFACE m_axi port=string1 offset=slave bundle=gmem
25  #pragma HLS INTERFACE m_axi port=string2 offset=slave bundle=gmem
26  #pragma HLS INTERFACE m_axi port=max_index offset=slave bundle=gmem
27  #pragma HLS INTERFACE m_axi port=similarity_matrix offset=slave bundle=gmem
28  #pragma HLS INTERFACE m_axi port=direction_matrix offset=slave bundle=gmem
29
30  #pragma HLS INTERFACE s_axilite port=string1 bundle=control
31  #pragma HLS INTERFACE s_axilite port=string2 bundle=control
32  #pragma HLS INTERFACE s_axilite port=max_index bundle=control
33  #pragma HLS INTERFACE s_axilite port=similarity_matrix bundle=control
34  #pragma HLS INTERFACE s_axilite port=direction_matrix bundle=control
35
36  #pragma HLS INTERFACE s_axilite port=return bundle=control
```

Initial Implementation

```
42      for(index = N; index < MATRIX_SIZE; index++) {
43          dir = CENTER;
44          val = 0;
45
46          i = index % N; // column index
47          j = index / N; // row index
48
49          if(i == 0) {
50              // first column
51              west = 0;
52              northwest = 0;
53          } else {
54
55              // all columns but first
56              north = similarity_matrix[index - N];
57              match = ( string1[i] == string2[j] ) ? MATCH : MISS_MATCH;
58
59              test_val = northwest + match;
60              if(test_val > val){
61                  val = test_val;
62                  dir = NORTH_WEST;
63              }
64
65              test_val = north + GAP_d;
66              if(test_val > val){
67                  val = test_val;
68                  dir = NORTH;
69              }
70
71              test_val = west + GAP_i;
72              if(test_val > val){
73                  val = test_val;
74                  dir = WEST;
75              }
76
77              similarity_matrix[index] = val;
78              direction_matrix[index] = dir;
79              west = val;
80              northwest = north;
81              if(val > max_value) {
82                  max_index[0] = index;
83                  max_value = val;
84              }
85          }
86      }
87  }
```

preparing dp[i-1][j-1], dp[i-1][j], dp[i][j-1]

Initial Implementation

```
dp[i][j]
= max
{
  dp[i-1][j-1] + s(a[i], b[j]),
  dp[i-1][j]   + GAP,
  dp[i][j-1]   + GAP,
  0
}
```

```
42      for(index = N; index < MATRIX_SIZE; index++) {
43          dir = CENTER;
44          val = 0;
45
46          i = index % N; // column index
47          j = index / N; // row index
48
49          if(i == 0) {
50              // first column
51              west = 0;
52              northwest = 0;
53          } else {
54
55              // all columns but first
56              north = similarity_matrix[index - N];
57              match = ( string1[i] == string2[j] ) ? MATCH : MISS_MATCH;
58
59              test_val = northwest + match;
60              if(test_val > val){
61                  val = test_val;
62                  dir = NORTH_WEST;
63              }
64
65              test_val = north + GAP_d;
66              if(test_val > val){
67                  val = test_val;
68                  dir = NORTH;
69              }
70
71              test_val = west + GAP_i;
72              if(test_val > val){
73                  val = test_val;
74                  dir = WEST;
75              }
76
77              similarity_matrix[index] = val;
78              direction_matrix[index] = dir;
79              west = val;
80              northwest = north;
81              if(val > max_value) {
82                  max_index[0] = index;
83                  max_value = val;
84              }
85          }
86      }
87  }
```

finding max

Initial Implementation

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.750 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
8136	166632	81.360 us	1.666 ms	8136	166632	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	8128	166624	2 ~ 41	-	-	4064	no

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1102	-
FIFO	-	-	-	-	-
Instance	2	-	738	940	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	397	-
Register	-	-	950	-	-
Total	2	0	1688	2439	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	1	4	0

Detail

Cosimulation Report for 'compute_matrices'

Result

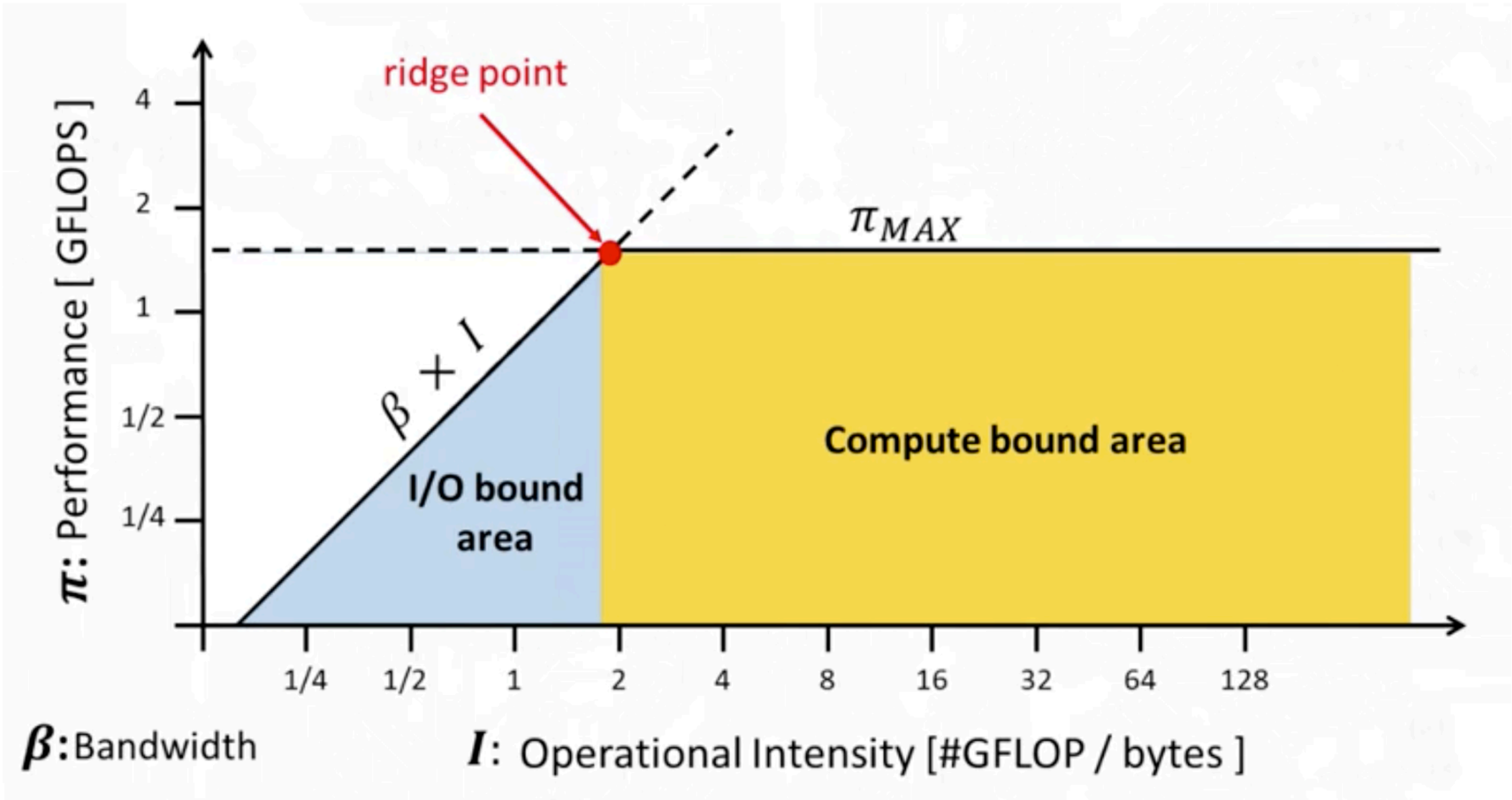
		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	240688	240763	240809	240689	240762	240810

Export the report(.html) using the Export Wizard

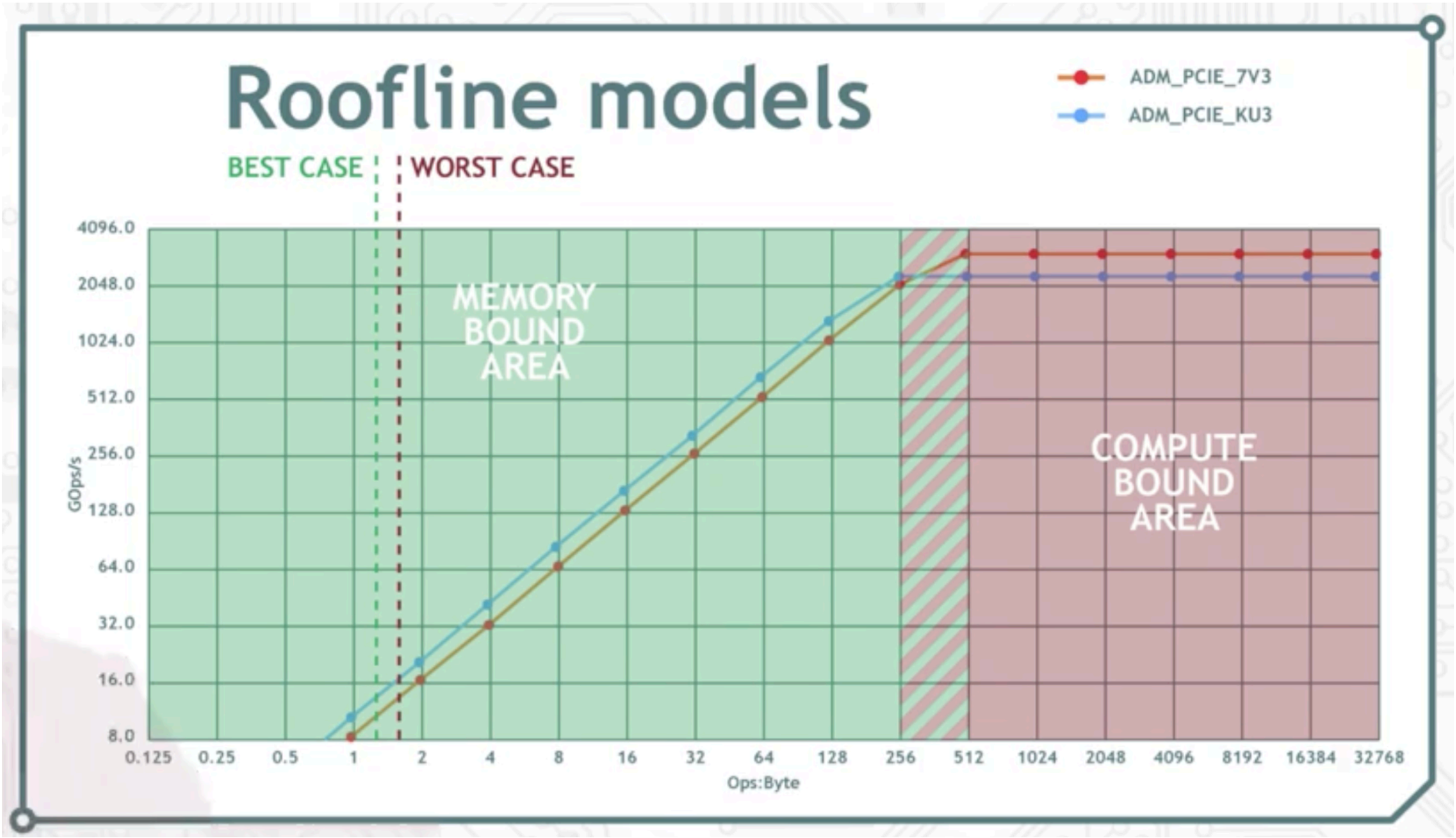
Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Roofline Model



Roofline Model



Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Systolic Array Architecture

		G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0
G	0	2	1	0	0	0	2	2
T	0	1	4	3	2	1	1	1
T	0	0	3	6	5	4	3	2
G	0	2	2	5	5	…		
A	0							
C	0							
T	0							

unroll? pipeline?

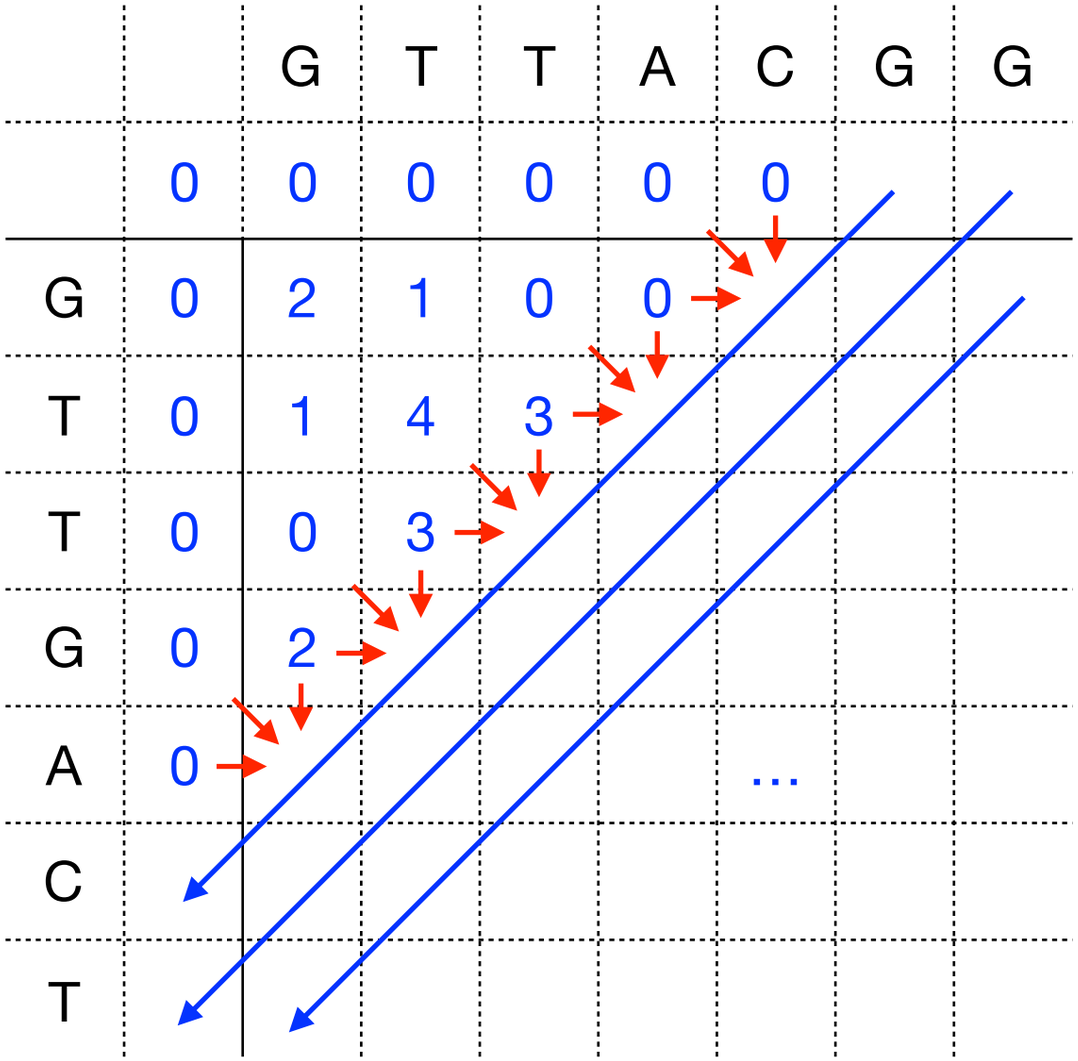
Systolic Array Architecture

		G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0
G	0	2	1	0	0	0	2	2
T	0	1	4	3	2	1	1	1
T	0	0	3	6	5	4	3	2
G	0	2	2	5	5	...		
A	0							
C	0							
T	0							

unroll? pipeline?

```
dp[i][j] depends on dp[i-1][j],
                    dp[i][j-1],
                    dp[i-1][j-1]
```

Systolic Array Architecture



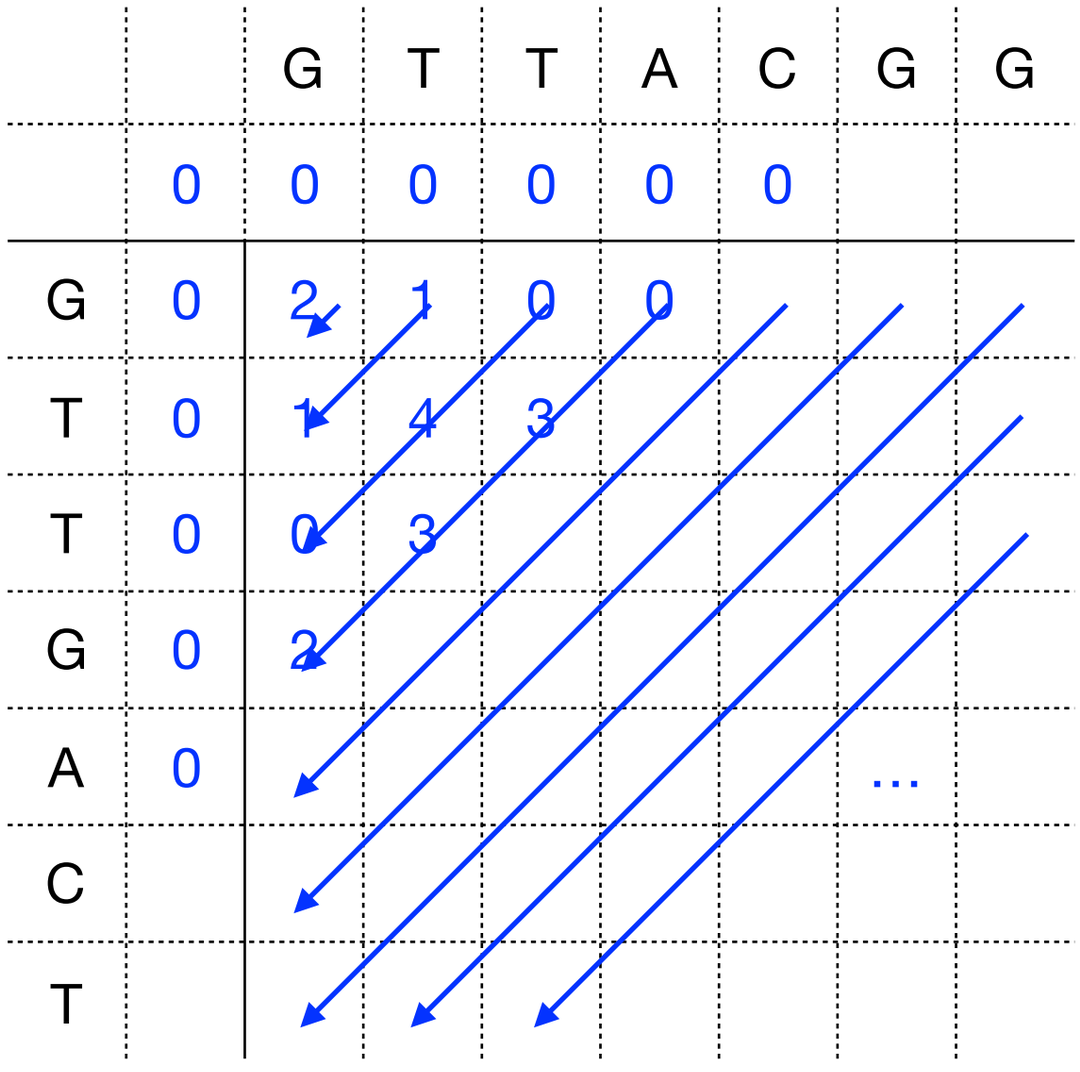
unroll? pipeline?

```
dp[i][j] depends on dp[i-1][j],
                    dp[i][j-1],
                    dp[i-1][j-1]
```

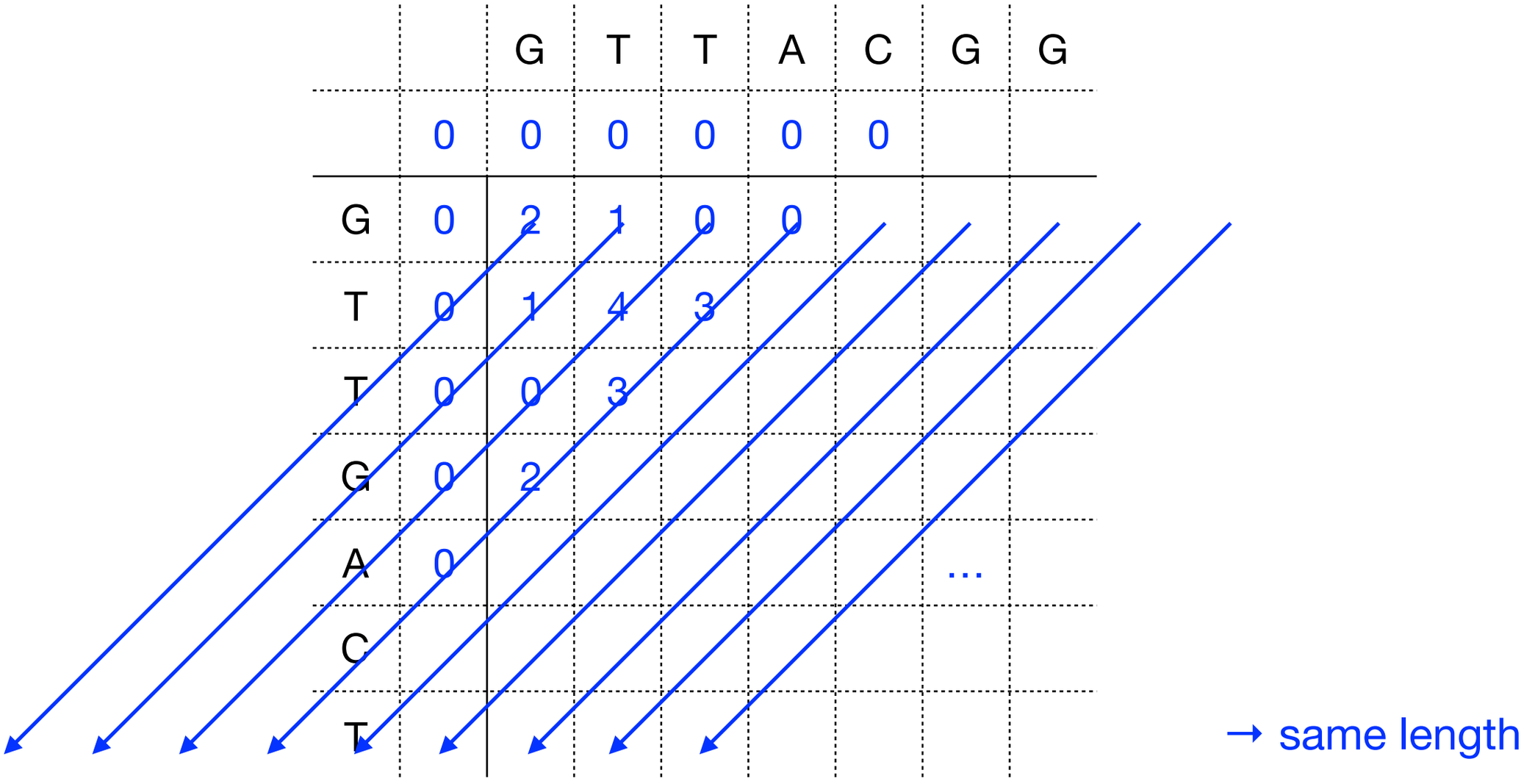
→ independent

Time complexity = O(m+n-1)

Systolic Array Architecture



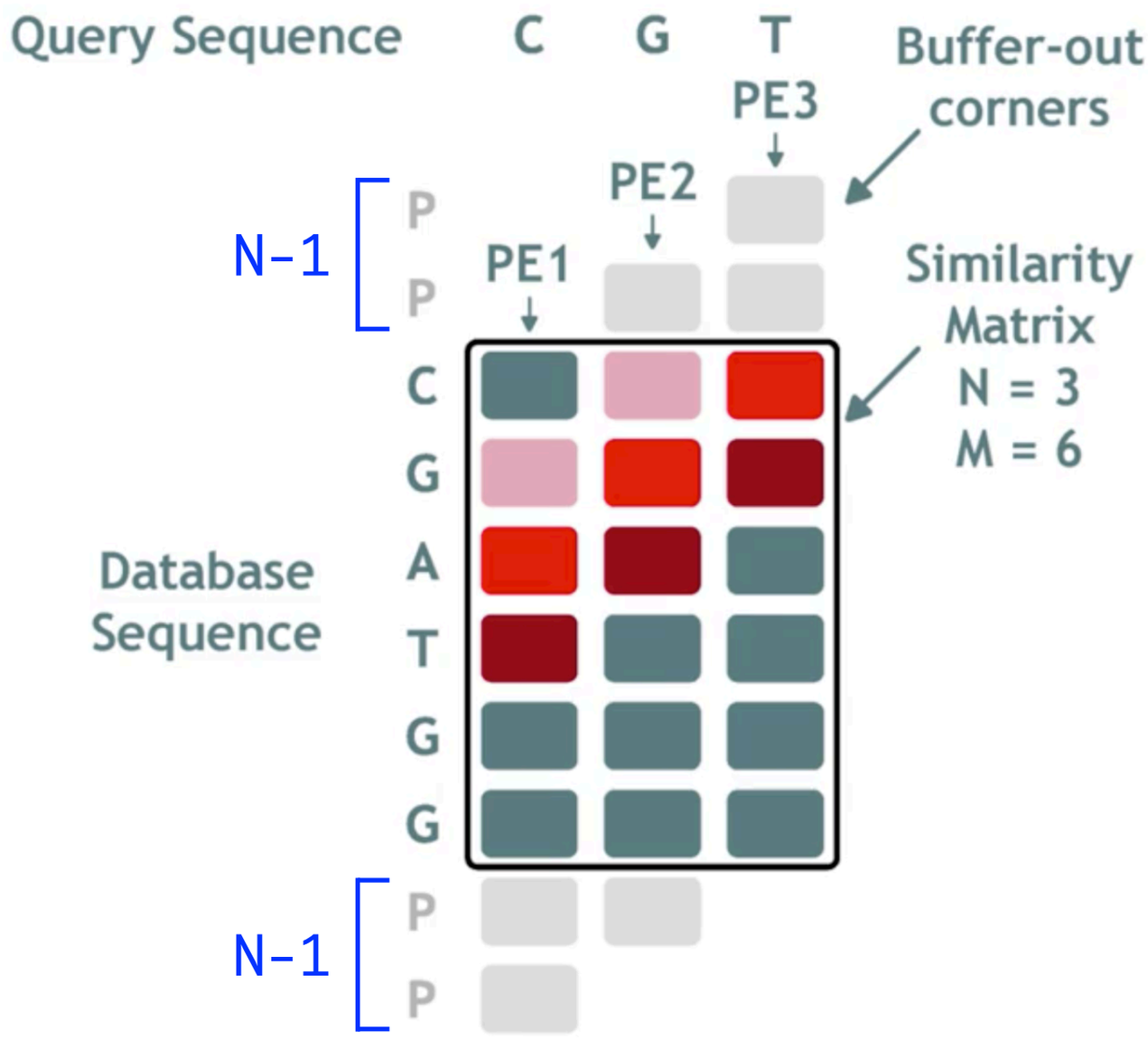
Systolic Array Architecture



→ same length

Systolic Array Architecture

```
17  #define N 64
18  #define M 128
19  #define DATABASE_SIZE M + 2 * (N - 1)
20  #define DIRECTION_MATRIX_SIZE (N + M - 1) * N
21  #define MATRIX_SIZE N * M
```



Systolic Array Architecture

```
91  void compute_matrices( char *string1_g, char *string2_g, ap_uint<512>
        *direction_matrix_g)
92  {
93  #pragma HLS INTERFACE m_axi port=string1_g offset=slave bundle=gmem0
94  #pragma HLS INTERFACE m_axi port=string2_g offset=slave bundle=gmem0
95  #pragma HLS INTERFACE m_axi port=direction_matrix_g offset=slave bundle=gmem1
96
97  #pragma HLS INTERFACE s_axilite port=string1_g bundle=control
98  #pragma HLS INTERFACE s_axilite port=string2_g bundle=control
99  #pragma HLS INTERFACE s_axilite port=direction_matrix_g bundle=control
100
101 #pragma HLS INTERFACE s_axilite port=return bundle=control
102
103     char string1[N];
104 #pragma HLS ARRAY_PARTITION variable=string1 complete dim=1
105     char string2[DATABASE_SIZE];
106 #pragma HLS ARRAY_PARTITION variable=string2 complete dim=1
107
108 //  short direction_matrix[DIRECTION_MATRIX_SIZE];
109 //#pragma HLS ARRAY_PARTITION variable=direction_matrix complete dim=1
110
111     memcpy(string1, string1_g, N*sizeof(char));
112     memcpy(string2, string2_g, DATABASE_SIZE * sizeof(char));
113
114     int north[N+1];
115 #pragma HLS ARRAY_PARTITION variable=north complete dim=1
116     int west[N+1];
117 #pragma HLS ARRAY_PARTITION variable=west complete dim=1
118     int northwest[N+1];
119 #pragma HLS ARRAY_PARTITION variable=northwest complete dim=1
```

Systolic Array Architecture

```
126     ap_uint<512> compressed_diag[1];
127
128     init_dep_for:for(int i = 0; i <= N; i++){
129         north[i] = 0;
130         west[i] = 0;
131         northwest[i] = 0;
132     }
133
134     int directions_index = 0;
135
136     num_diag_for: for(int num_diagonals = 0; num_diagonals < N + M - 1;
            num_diagonals++){
137 #pragma HLS inline region recursive
138 #pragma HLS PIPELINE
139
140         calculate_diagonal(num_diagonals, string1, string2, northwest, north,
                west, directions_index, compressed_diag);
141         store_diagonal(directions_index,  direction_matrix_g, compressed_diag);
142         directions_index ++;
143     }
144
145 //  memcpy(direction_matrix_g, direction_matrix, DIRECTION_MATRIX_SIZE *
        sizeof(short));
146
147     return;
148 }
149 }
```

Systolic Array Architecture

```
33  void calculate_diagonal(int num_diagonals, char string1[N], char
        string2[DATABASE_SIZE], int northwest[N + 1], int north[N + 1], int west[N +
        1], int directions_index, ap_uint<512> compressed_diag[1]){
34
35      int databaseLocalIndex = num_diagonals;
36      int from, to;
37      from = N * 2 - 2;
38      to = N * 2 - 1;
39
40      calculate_diagonal_for: for(int index = N - 1; index >= 0; index --){
41          int val = 0;
42
43          unsigned int q = string1[index];
44          unsigned int db = string2[databaseLocalIndex];
45
46          if(num_diagonals < N - 1 && databaseLocalIndex < N - 1 - num_diagonals)
                db = 9;
47
48          const short match = (q == db) ? MATCH : MISS_MATCH;
49          const short val1 = northwest[index] + match;
50          const short val2 = north[index] + GAP_d;
51          const short val3 = west[index] + GAP_i;
```

Systolic Array Architecture

```
53          if(val1 > val && val1 >= val2 && val1 >= val3){
54              //val1
55              northwest[index + 1] = north[index];
56              north[index] = val1;
57              west[index + 1] = val1;
58              compressed_diag[0].range(to,from) = NORTH_WEST;
59  //          directionDiagonal[index] = NORTH_WEST;
60          } else if (val2 > val && val2 >= val3) {
61              //val2
62              northwest[index + 1] = north[index];
63              north[index] = val2;
64              west[index + 1] = val2;
65              compressed_diag[0].range(to,from) = NORTH;
66  //          directionDiagonal[index] = NORTH;
67          }else if (val3 > val){
68              //val3
69              northwest[index + 1] = north[index];
70              north[index] = val3;
71              west[index + 1] = val3;
72              compressed_diag[0].range(to,from) = WEST;
73  //          directionDiagonal[index] = WEST;
74          }else{
75              //val
76              northwest[index + 1] = north[index];
77              north[index] = val;
78              west[index + 1] = val;
79              compressed_diag[0].range(to,from) = CENTER;
80  //          directionDiagonal[index] = CENTER;
81          }
82          databaseLocalIndex ++;
83          from -= 2;
84          to -= 2;
85      }
86
87  }
```

Systolic Array Architecture

M=128, N=32

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.750 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
440	440	4.400 us	4.400 us	440	440	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- memcpy..string1_g	32	32	2	1	1	32	yes
- memcpy..string2_g	190	190	2	1	1	190	yes
- init_dep_for	33	33	1	1	1	33	yes
- num_diag_for	160	160	3	1	1	159	yes

Systolic Array Architecture

M=128, N=32

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	7472	-
FIFO	-	-	-	-	-
Instance	8	-	1282	128430	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	4229	-
Register	-	-	8523	-	-
Total	8	0	9805	140131	0
Available	280	220	106400	53200	0
Utilization (%)	2	0	9	263	0

Detail

Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Input Compression

```
28 void set_char_main(unsigned int  * array, int index, unsigned char val){
29     switch(val){
30         case 'A' : array[index / 16] |= (0 << ((index % 16) * 2));
31         break;
32         case 'C' : array[index / 16] |= (1 << ((index % 16) * 2));
33         break;
34         case 'G' : array[index / 16] |= (3 << ((index % 16) * 2));
35         break;
36         case 'T' : array[index / 16] |= (2 << ((index % 16) * 2));
37         break;
38     }
39
40 }
```

8 bit/character (char)

↓ input compression

2 bit/character

Input Compression

```
40      calculate_diagonal_for: for(int index = N - 1; index >= 0; index --){
41          int val = 0;
42
43          unsigned int q = string1[index];
44          unsigned int db = string2[databaseLocalIndex];
```

↓ input compression

```
42      calculate_diagonal_for: for(int index = N - 1; index >= 0; index --){
43          int val = 0;
44          const short q = string1[index/NUM_ELEM].range((index%NUM_ELEM) * 2 + 1,
                (index%NUM_ELEM) * 2);
45          short db =
                string2
                [databaseLocalIndex/NUM_ELEM].range((databaseLocalIndex%NUM_ELEM) * 2
                + 1, (databaseLocalIndex%NUM_ELEM) * 2);
```

Input Compression

```
91  void compute_matrices( char *string1_g, char *string2_g, ap_uint<512>
        *direction_matrix_g)
92  {
93  #pragma HLS INTERFACE m_axi port=string1_g offset=slave bundle=gmem0
94  #pragma HLS INTERFACE m_axi port=string2_g offset=slave bundle=gmem0
95  #pragma HLS INTERFACE m_axi port=direction_matrix_g offset=slave bundle=gmem1
96
97  #pragma HLS INTERFACE s_axilite port=string1_g bundle=control
98  #pragma HLS INTERFACE s_axilite port=string2_g bundle=control
99  #pragma HLS INTERFACE s_axilite port=direction_matrix_g bundle=control
100
101 #pragma HLS INTERFACE s_axilite port=return bundle=control
102
103     char string1[N];
104 #pragma HLS ARRAY_PARTITION variable=string1 complete dim=1
105     char string2[DATABASE_SIZE];
106 #pragma HLS ARRAY_PARTITION variable=string2 complete dim=1
107
108 //  short direction_matrix[DIRECTION_MATRIX_SIZE];
109 //#pragma HLS ARRAY_PARTITION variable=direction_matrix complete dim=1
110
111     memcpy(string1, string1_g, N*sizeof(char));
112     memcpy(string2, string2_g, DATABASE_SIZE * sizeof(char));
```

↘ input compression

```
93  void compute_matrices( ap_uint<512> *string1_g, ap_uint<512> *string2_g,
        ap_uint<512> *direction_matrix_g)
94  {
95  #pragma HLS INTERFACE m_axi port=string1_g offset=slave bundle=gmem0
96  #pragma HLS INTERFACE m_axi port=string2_g offset=slave bundle=gmem0
97  #pragma HLS INTERFACE m_axi port=direction_matrix_g offset=slave bundle=gmem1
98
99  #pragma HLS INTERFACE s_axilite port=string1_g bundle=control
100 #pragma HLS INTERFACE s_axilite port=string2_g bundle=control
101 #pragma HLS INTERFACE s_axilite port=direction_matrix_g bundle=control
102
103 #pragma HLS INTERFACE s_axilite port=return bundle=control
104
105     ap_uint<512> string1[N / NUM_ELEM  + 1];
106 //#pragma HLS ARRAY_PARTITION variable=string1 complete dim=1
107     ap_uint<512> string2[(DATABASE_SIZE)/ NUM_ELEM + 1];
108 //#pragma HLS ARRAY_PARTITION variable=string2 complete dim=1
109
110 //  short direction_matrix[DIRECTION_MATRIX_SIZE];
111 //#pragma HLS ARRAY_PARTITION variable=direction_matrix complete dim=1
112
113     memcpy(string1, string1_g, (N/NUM_ELEM + 1) * 64);
114     memcpy(string2, string2_g, ((DATABASE_SIZE) / NUM_ELEM + 1) * 64);
```

Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Shift Register

```
146     ap_uint<512> compressed_diag[1];
147     ap_uint<512> shift_db[N/NUM_ELEM];
148
149     init_dep_for:for(int i = 0; i <= N; i++){
150 #pragma HLS PIPELINE
151         north[i] = 0;
152         west[i] = 0;
153         northwest[i] = 0;
154     }
155
156     init_db:for(int i = 0; i < N/NUM_ELEM; i++){
157 #pragma HLS PIPELINE
158         shift_db[i] = string2[i];
159     }
```

Shift Register

```
162     num_diag_for: for(int num_diagonals = 0; num_diagonals < N + M - 1;
            num_diagonals++){
163 #pragma HLS inline region recursive
164 #pragma HLS PIPELINE
165
166         calculate_diagonal(num_diagonals, string1, string2, northwest, north,
                west, directions_index, compressed_diag, shift_db,
                direction_matrix_g);
167
168         update_database(string2, shift_db, num_diagonals);
169     }
```

Shift Register

```
33  void update_database(ap_uint<512> *database, ap_uint<512> *shift_db, int
        num_diagonals){
34      int startingIndex = N + num_diagonals;
35      update_database:for(int i = 1; i < N; i++){
36  #pragma HLS PIPELINE
37          shift_db[(i-1)/NUM_ELEM].range(((i-1)%NUM_ELEM)*2+1, ((i-1)%NUM_ELEM)*2)
                = shift_db[i/NUM_ELEM].range((i%NUM_ELEM)*2+1, (i%NUM_ELEM)*2);
38      }
39      shift_db[(N-1)/NUM_ELEM].range(((N-1)%NUM_ELEM) * 2 + 1,((N-1)%NUM_ELEM) * 2)
            = database[startingIndex/NUM_ELEM].range((startingIndex%NUM_ELEM) * 2 +1,
            (startingIndex%NUM_ELEM) *2);
40  }
```

Outline

1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Comparison

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.750 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
8136	166632	81.360 us	1.666 ms	8136	166632	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	8128	166624	2 ~ 41	-	-	4064	no

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1102	-
FIFO	-	-	-	-	-
Instance	2	-	738	940	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	397	-
Register	-	-	950	-	-
Total	2	0	1688	2439	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	1	4	0

Detail

Cosimulation Report for 'compute_matrices'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	240688	240763	240809	240689	240762	240810

Export the report(.html) using the Export Wizard

Comparison

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.750 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
440	440	4.400 us	4.400 us	440	440	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- memcpy..string1_g	32	32	2	1	1	32	yes
- memcpy..string2_g	190	190	2	1	1	190	yes
- init_dep_for	33	33	1	1	1	33	yes
- num_diag_for	160	160	3	1	1	159	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	7472	-
FIFO	-	-	-	-	-
Instance	8	-	1282	128430	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	4229	-
Register	-	-	8523	-	-
Total	8	0	9805	140131	0
Available	280	220	106400	53200	0
Utilization (%)	2	0	9	263	0

Detail

Cosimulation Report for 'compute_matrices'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	781	784	785	782	785	786

Export the report(.html) using the Export Wizard

Comparison

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.750 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
231	231	2.310 us	2.310 us	231	231	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval		
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count
- memcpy.string1.V.addr.string1_g.V	3	3	3	1	1	2
- memcpy.string2.V.addr.string2_g.V	7	7	3	1	1	6
- init_dep_for	33	33	1	1	1	33
- num_diag_for	162	162	5	1	1	159

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	8515	-
FIFO	-	-	-	-	-
Instance	8	-	1282	18690	-
Memory	0	-	128	6	0
Multiplexer	-	-	-	2025	-
Register	0	-	5784	32	-
Total	8	0	7194	29268	0
Available	280	220	106400	53200	0
Utilization (%)	2	0	6	55	0

Detail

Cosimulation Report for 'compute_matrices'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	286	289	290	287	290	291

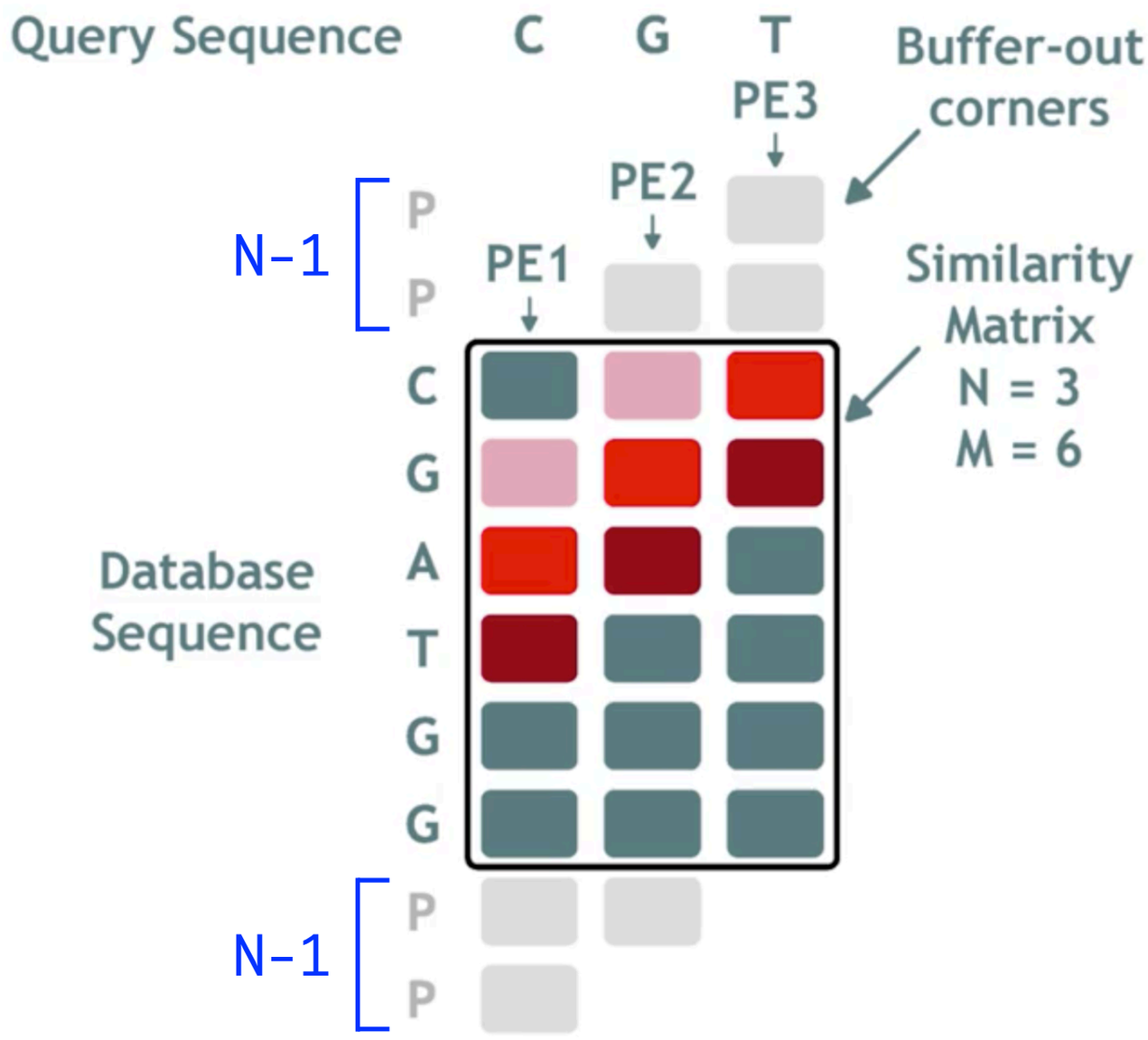
Export the report(.html) using the Export Wizard

Outline

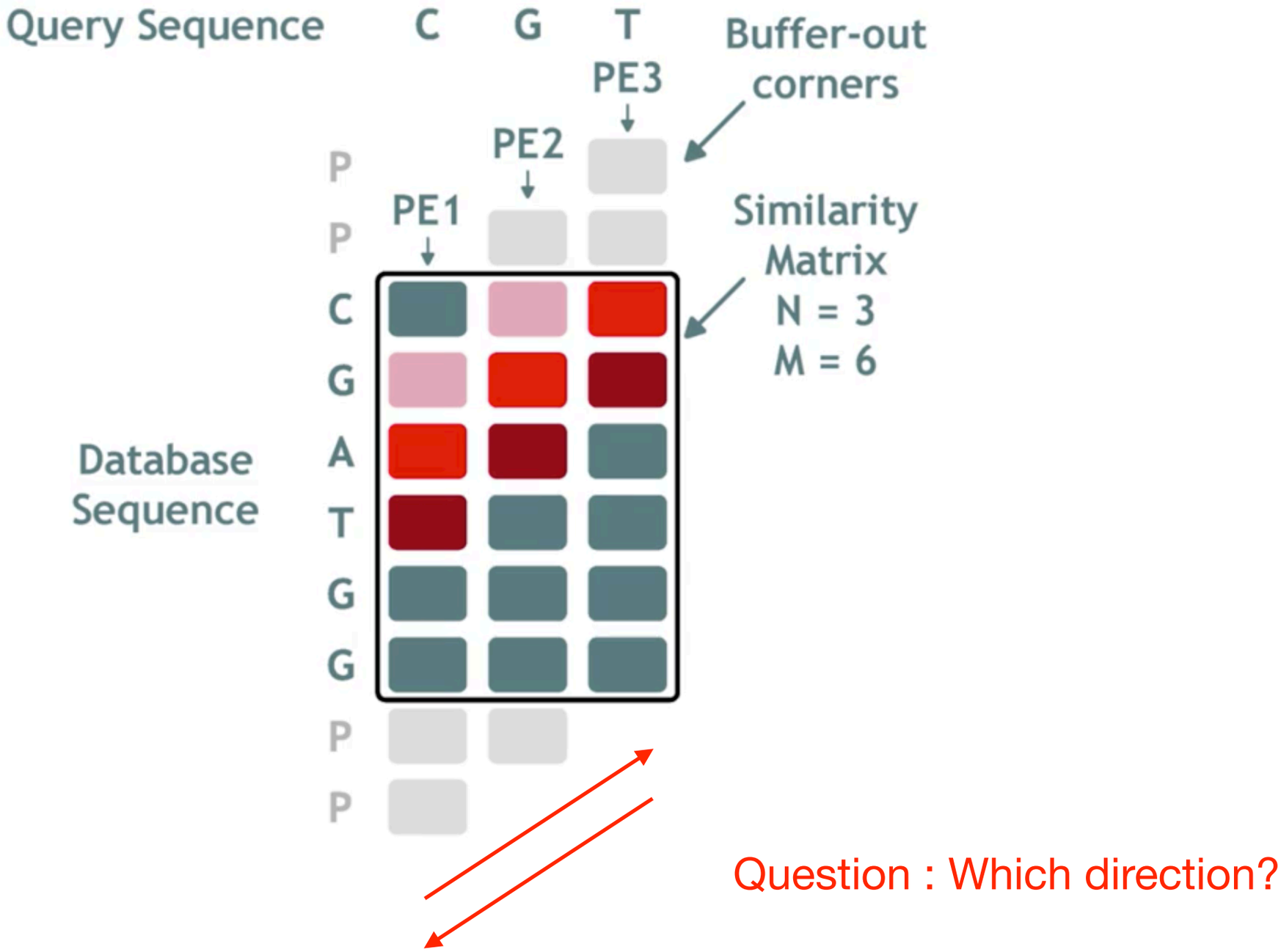
1. Introduction to Smith-Waterman Algorithm
2. Initial Implementation
3. Roofline Model
4. Systolic Array Architecture
5. Input Compression
6. Shift Register
7. Comparison
8. Questions

Systolic Array Architecture

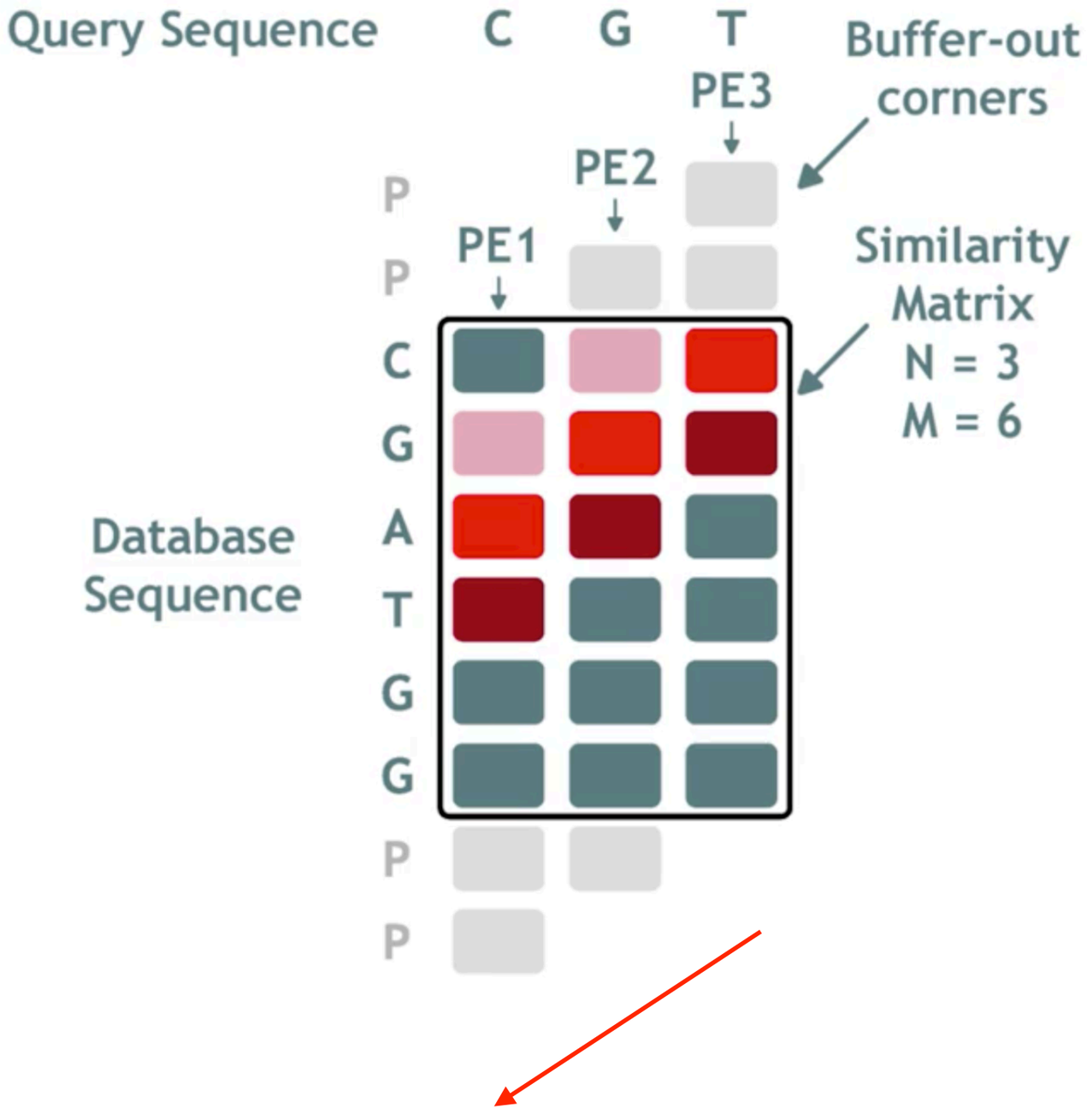
```
17  #define N 64
18  #define M 128
19  #define DATABASE_SIZE M + 2 * (N - 1)
20  #define DIRECTION_MATRIX_SIZE (N + M - 1) * N
21  #define MATRIX_SIZE N * M
```



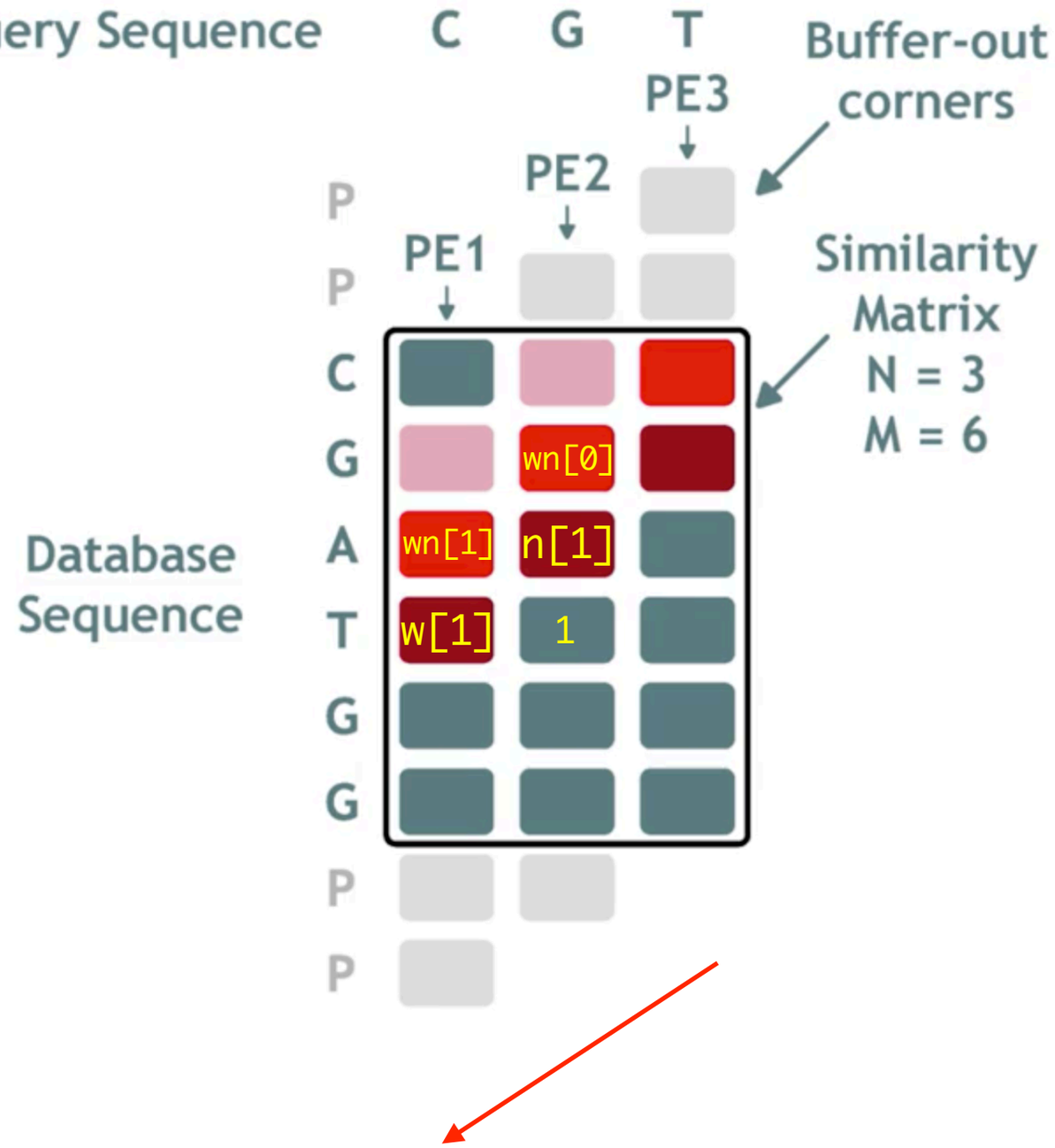
Systolic Array Architecture



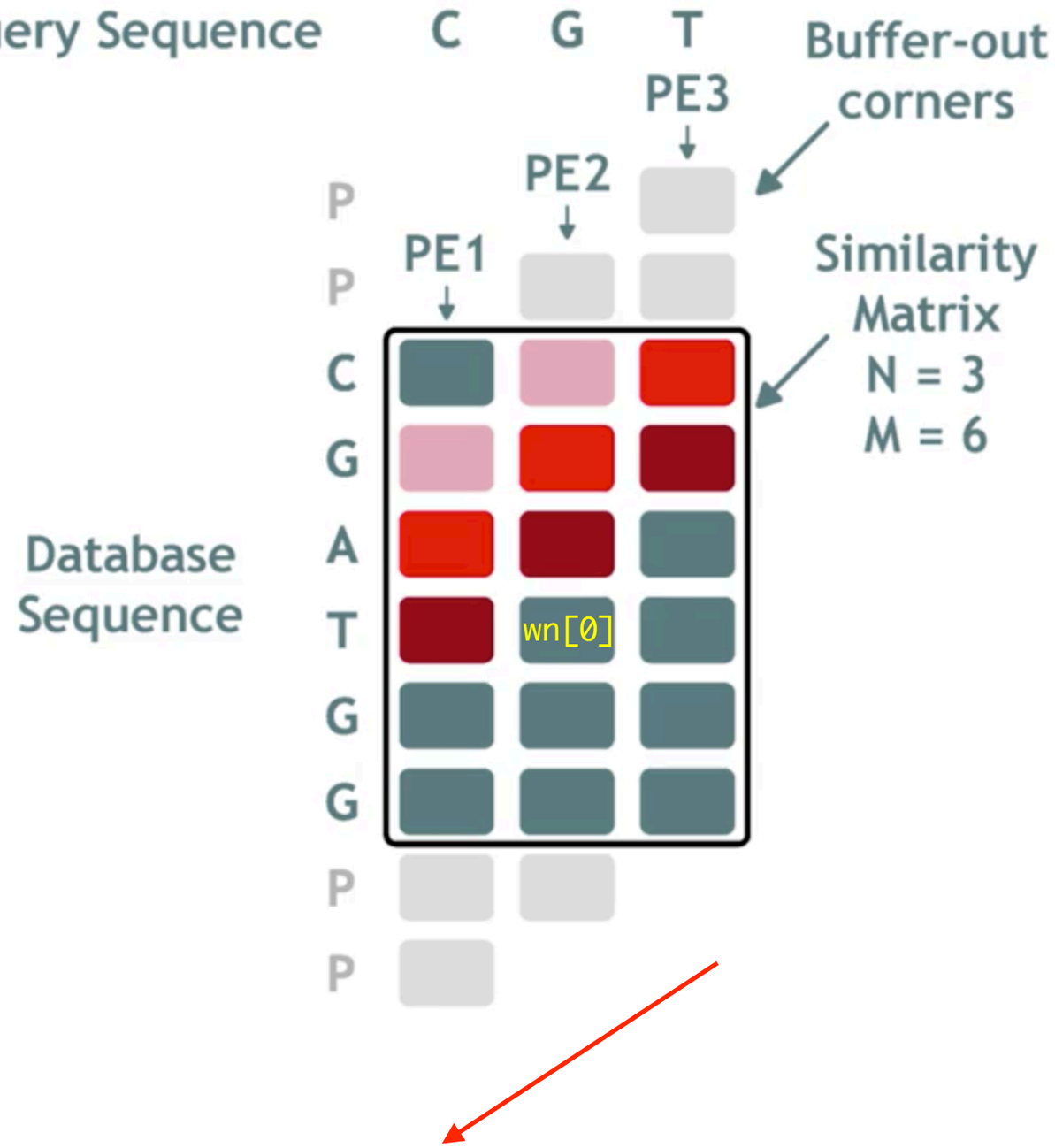
Systolic Array Architecture



Systolic Array Architecture



Systolic Array Architecture



Systolic Array Architecture

```
126     ap_uint<512> compressed_diag[1];
127
128     init_dep_for:for(int i = 0; i <= N; i++){
129         north[i] = 0;
130         west[i] = 0;
131         northwest[i] = 0;
132     }
133
134     int directions_index = 0;
135
136     num_diag_for: for(int num_diagonals = 0; num_diagonals < N + M - 1;
            num_diagonals++){
137 #pragma HLS inline region recursive
138 #pragma HLS PIPELINE
139
140         calculate_diagonal(num_diagonals, string1, string2, northwest, north,
                west, directions_index, compressed_diag);
141         store_diagonal(directions_index,  direction_matrix_g, compressed_diag);
142         directions_index ++;
143     }
144
145 //  memcpy(direction_matrix_g, direction_matrix, DIRECTION_MATRIX_SIZE *
        sizeof(short));
146
147     return;
148 }
149 }
```

Systolic Array Architecture

```
53          if(val1 > val && val1 >= val2 && val1 >= val3){
54              //val1
55              northwest[index + 1] = north[index];
56              north[index] = val1;
57              west[index + 1] = val1;
58              compressed_diag[0].range(to,from) = NORTH_WEST;
59  //          directionDiagonal[index] = NORTH_WEST;
60          } else if (val2 > val && val2 >= val3) {
61              //val2
62              northwest[index + 1] = north[index];
63              north[index] = val2;
64              west[index + 1] = val2;
65              compressed_diag[0].range(to,from) = NORTH;
66  //          directionDiagonal[index] = NORTH;
67          }else if (val3 > val){
68              //val3
69              northwest[index + 1] = north[index];
70              north[index] = val3;
71              west[index + 1] = val3;
72              compressed_diag[0].range(to,from) = WEST;
73  //          directionDiagonal[index] = WEST;
74          }else{
75              //val
76              northwest[index + 1] = north[index];
77              north[index] = val;
78              west[index + 1] = val;
79              compressed_diag[0].range(to,from) = CENTER;
80  //          directionDiagonal[index] = CENTER;
81          }
82          databaseLocalIndex ++;
83          from -= 2;
84          to -= 2;
85      }
86
87  }
```

References

- https://www.coursera.org/learn/fpga-sdaccel-practice
- https://github.com/necst/coursera-sdaccel-practice