HLS LabA Design Optimization

Matrix Multiplication

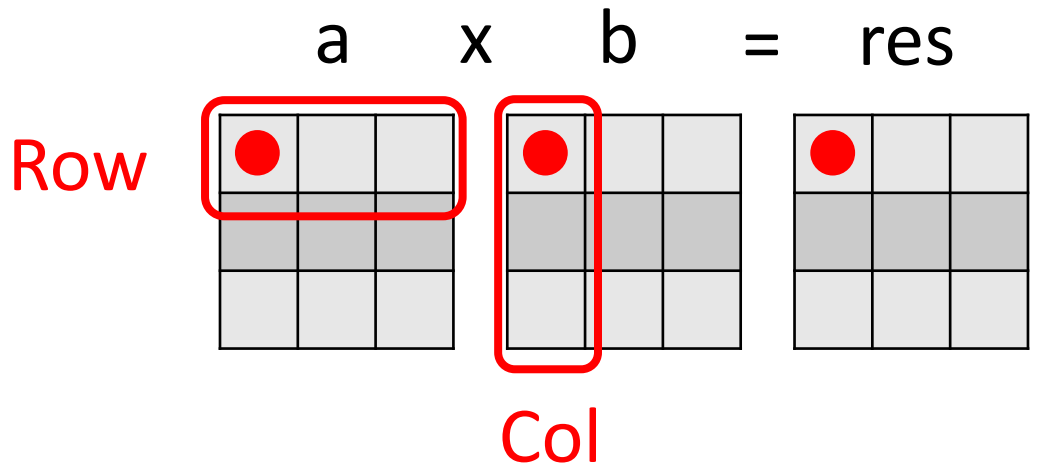
2022/03/30

Outline

- Original code explanation
- Optimized code explanation
- Performance analysis
- Tradeoff

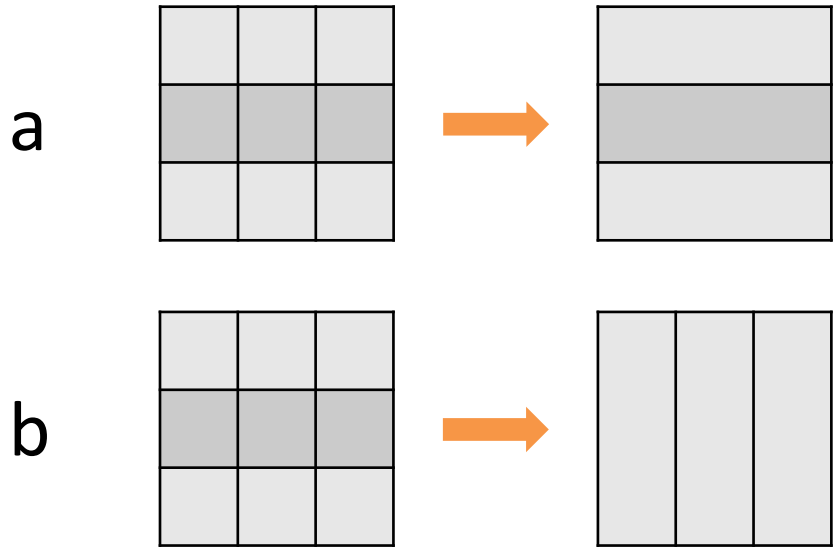
Original code explanation

```
46  #include "matrixmul.h"
47
48  void matrixmul(
49          mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
50          mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
51          result_t res[MAT_A_ROWS][MAT_B_COLS])
52  {
53    // Iterate over the rows of the A matrix
54     Row: for(int i = 0; i < MAT_A_ROWS; i++) {
55          // Iterate over the columns of the B matrix
56          Col: for(int j = 0; j < MAT_B_COLS; j++) {
57              res[i][j] = 0;
58              // Do the inner product of a row of A and col of B
59              Product: for(int k = 0; k < MAT_B_ROWS; k++) {
60                  res[i][j] += a[i][k] * b[k][j];
61              }
62          }
63      }
64  }
```



Optimized code explanation

- ARRAY_RESHAPE
  - New array with fewer elements but with greater bit-width




```
46  #include "matrixmul.h"
47
48  void matrixmul(
49          mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
50          mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
51          result_t res[MAT_A_ROWS][MAT_B_COLS])
52  {
53  #pragma HLS ARRAY_RESHAPE variable=b complete dim=1
54  #pragma HLS ARRAY_RESHAPE variable=a complete dim=2
55  #pragma HLS INTERFACE ap_fifo port=a
56  #pragma HLS INTERFACE ap_fifo port=b
57  #pragma HLS INTERFACE ap_fifo port=res
```

AP_MEMORY

Interface	Bitwidth
a_address0	4
a_address1	4
a_q0	8
a_q1	8
b_address0	4
b_address1	4
b_q0	8
b_q1	8
res_address0	4
res_d0	16

Bit-width from 8-bit to 24-bit

AP_MEMORY

Interface	Bitwidth
a_address0	2
a_q0	24
b_address0	2
b_address1	2
b_q0	24
b_q1	24
res_address0	4
res_address1	4
res_d0	16
res_d1	16

Optimized code explanation

- INTERFACE ap_fifo
  - Enable steam data transfer

HW Interfaces

AP_MEMORY

Interface	Bitwidth
a_address0	4
a_address1	4
a_q0	8
a_q1	8
b_address0	4
b_address1	4
b_q0	8
b_q1	8
res_address0	4
res_d0	16

HW Interfaces

AP_FIFO

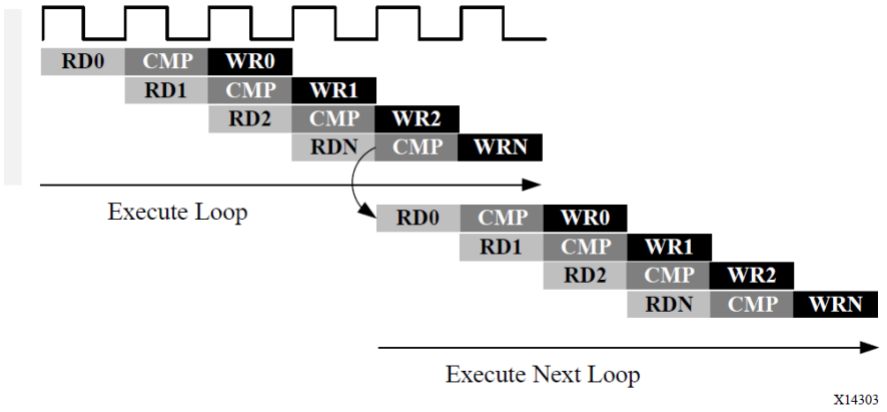
Interface	Data Width
a	8
b	8
res	16

```
46  #include "matrixmul.h"
47
48  void matrixmul(
49          mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
50          mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
51          result_t res[MAT_A_ROWS][MAT_B_COLS])
52  {
53  #pragma HLS ARRAY_RESHAPE variable=b complete dim=1
54  #pragma HLS ARRAY_RESHAPE variable=a complete dim=2
55  #pragma HLS INTERFACE ap_fifo port=a
56  #pragma HLS INTERFACE ap_fifo port=b
57  #pragma HLS INTERFACE ap_fifo port=res
```

Optimized code explanation

- PIPELINE rewind
  - No pause between loops

Figure 55: **Loop Pipelining with Rewind Option**




```
58    mat_a_t a_row[MAT_A_ROWS];
59    mat_b_t b_copy[MAT_B_ROWS][MAT_B_COLS];
60    int tmp = 0;
61    // Iterate over the rowa of the A matrix
62    Row: for(int i = 0; i < MAT_A_ROWS; i++) {
63      // Iterate over the columns of the B matrix
64      Col: for(int j = 0; j < MAT_B_COLS; j++) {
65  #pragma HLS PIPELINE rewind
66        // Do the inner product of a row of A and col of B
67        tmp=0;
68        // Cache each row (so it's only read once per function)
69        if (j == 0)
70          Cache_Row: for(int k = 0; k < MAT_A_ROWS; k++)
71            a_row[k] = a[i][k];
72         // Cache all cols (so they are only read once per function)
73        if (i == 0)
74          Cache_Col: for(int k = 0; k < MAT_B_ROWS; k++)
75            b_copy[k][j] = b[k][j];
76        Product: for(int k = 0; k < MAT_B_ROWS; k++) {
77          tmp += a_row[k] * b_copy[k][j];
78        }
79        res[i][j] = tmp;
80      }
81    }
82  }
```

Optimized code explanation

- Cache input data
  - Red: first read
  - Blue: data reuse

```
58  mat_a_t a_row[MAT_A_ROWS];
59  mat_b_t b_copy[MAT_B_ROWS][MAT_B_COLS];
60  int tmp = 0;
61  // Iterate over the rowa of the A matrix
62  Row: for(int i = 0; i < MAT_A_ROWS; i++) {
63    // Iterate over the columns of the B matrix
64    Col: for(int j = 0; j < MAT_B_COLS; j++) {
65  #pragma HLS PIPELINE rewind
66      // Do the inner product of a row of A and col of B
67      tmp=0;
68      // Cache each row (so it's only read once per function)
69      if (j == 0)
70        Cache_Row: for(int k = 0; k < MAT_A_ROWS; k++)
71          a_row[k] = a[i][k];
72       // Cache all cols (so they are only read once per function)
73      if (i == 0)
74        Cache_Col: for(int k = 0; k < MAT_B_ROWS; k++)
75          b_copy[k][j] = b[k][j];
76      Product: for(int k = 0; k < MAT_B_ROWS; k++) {
77        tmp += a_row[k] * b_copy[k][j];
78      }
79      res[i][j] = tmp;
```

Row i	0									1									2								
Col j	0			1			2			0			1			2			0			1			2		
Product k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
addr a	00	01	02	00	01	02	00	01	02	10	11	12	10	11	12	10	11	12	20	21	22	20	21	22	20	21	22
addr b	00	10	20	01	11	21	02	12	22	00	10	20	01	11	21	02	12	22	00	10	20	01	11	21	02	12	22
addr res	00	00	00	01	01	01	02	02	02	10	10	10	11	11	11	12	12	12	20	20	20	21	21	21	22	22	22

Optimized code explanation

- Write once
  - Red: write to tmp
  - Blue: write to res

```
58    mat_a_t a_row[MAT_A_ROWS];
59    mat_b_t b_copy[MAT_B_ROWS][MAT_B_COLS];
60    int tmp = 0;
61    // Iterate over the rowa of the A matrix
62    Row: for(int i = 0; i < MAT_A_ROWS; i++) {
63      // Iterate over the columns of the B matrix
64      Col: for(int j = 0; j < MAT_B_COLS; j++) {
65  #pragma HLS PIPELINE rewind
66        // Do the inner product of a row of A and col of B
67        tmp=0;
68        // Cache each row (so it's only read once per function)
69        if (j == 0)
70          Cache_Row: for(int k = 0; k < MAT_A_ROWS; k++)
71            a_row[k] = a[i][k];
72         // Cache all cols (so they are only read once per function)
73        if (i == 0)
74          Cache_Col: for(int k = 0; k < MAT_B_ROWS; k++)
75            b_copy[k][j] = b[k][j];
76        Product: for(int k = 0; k < MAT_B_ROWS; k++) {
77          tmp += a_row[k] * b_copy[k][j];
78        }
79        res[i][j] = tmp;
80      }
81    }
82  }
```

Row i	0																										
Col j	0			1			2			0			1			2			0			1			2		
Product k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
addr a	00	01	02	00	01	02	00	01	02	10	11	12	10	11	12	10	11	12	20	21	22	20	21	22	20	21	22
addr b	00	10	20	01	11	21	02	12	22	00	10	20	01	11	21	02	12	22	00	10	20	01	11	21	02	12	22
addr res	00	00	00	01	01	01	02	02	02	10	10	10	11	11	11	12	12	12	20	20	20	21	21	21	22	22	22

Performance analysis

	Latency	Interval	DSP	FF	LUT
Lab1	24	25	2	117	393
Lab2	14	9	2	243	543
ARRAY_RESHAPE	15	16	6	248	367
ap_fifo	33	34	2	62	201
ARRAY_RESHAPE + ap_fifo	23	24	6	172	289
rewind	23	18	2	162	405
cache	24	25	2	239	667
Loop_flatten	18	19	6	441	472

Red is min, blue is max.

Tradeoff

- For FPGA hardware, 3x3 matrix multiplication utilizes ~0%.
- Use the fastest one: Optimized code from Lab2
  - ARRAY_RESHAPE
  - INTERFACE ap_fifo
  - PIPELINE rewind
  - Cache input data
  - Write once

Thank you!

hsu880105/HLS_LabA_Design_Optimization (github.com)