Block Matrix Multiplication

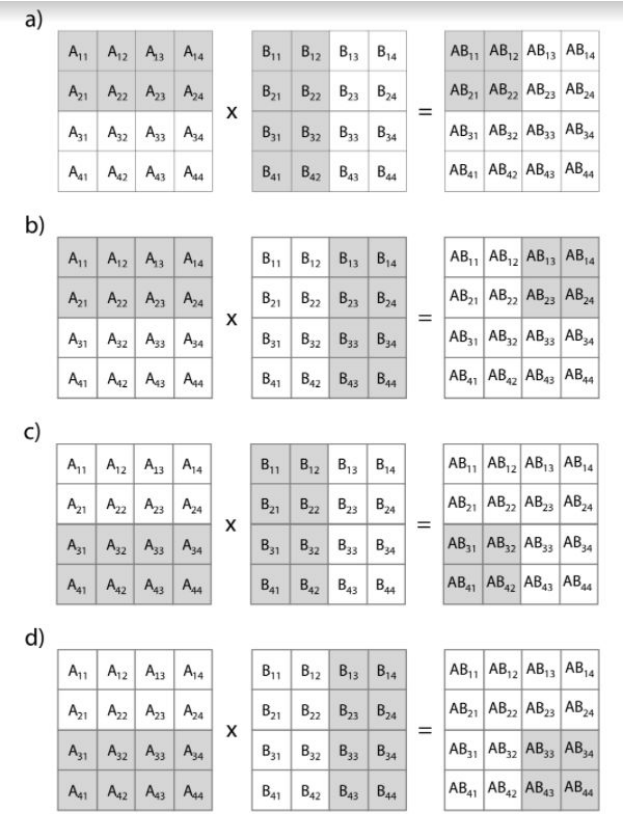
HLS Course

賴志杰

2022/3/30

Block Matrix Multiplication

1. divide the input arrays A and B into blocks
2. compute a portion of the product AB.
3. stream the next set of blocks, compute another portion of AB until the entire matrix multiplication is complete



Why Matrix Blocking

- an easy way to optimize the structure, ex. loop unrolling
- choose to block a matrix according to the natural structure of the matrix
- choose the blocking sizes that match the available on-chip memory size or to the number of multiply-add operators

Outline

- Explain the Original Code/System/Pragmas
- Analyze the Timing/Performance/Utilization
- Share How I Optimize the Design

Outline

- **Explain the Original Code/System/Pragmas**
- Analyze the Timing/Performance/Utilization
- Share How I Optimize the Design

Codes

- Header File
- Kernel Function File
- Testbench File

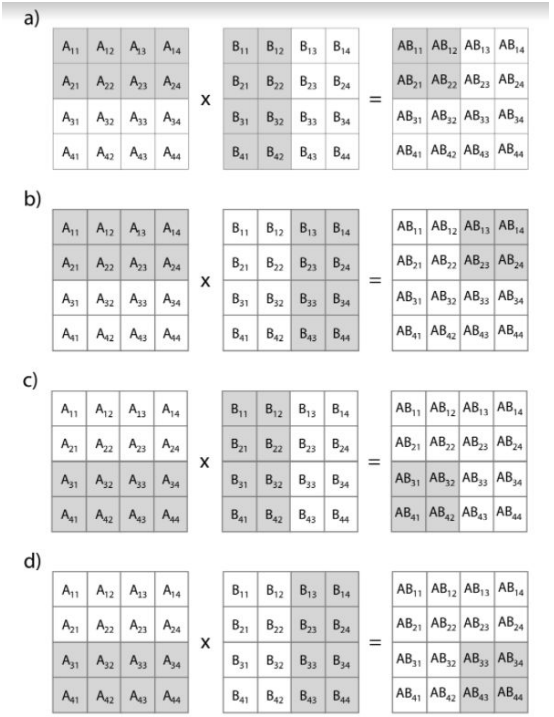
block_mm.cpp	Fresh commit of version 1.0	4 years ago
block_mm.h	Fresh commit of version 1.0	4 years ago
blockmatmul_test.cpp	Fresh commit of version 1.0	4 years ago
blockmatmul_test_init.cpp	Fresh commit of version 1.0	4 years ago

Header File

```
#ifndef _BLOCK_MM_H_
#define _BLOCK_MM_H_
#include "hls_stream.h"
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;

typedef int DTYPE;
const int SIZE = 8;
const int BLOCK_SIZE = 4;

typedef struct {
    DTYPE a[BLOCK_SIZE]; } blockvec;

typedef struct {
    DTYPE out[BLOCK_SIZE][BLOCK_SIZE]; } blockmat;

void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,
blockmat & ABpartial, DTYPE iteration);
#endif
```

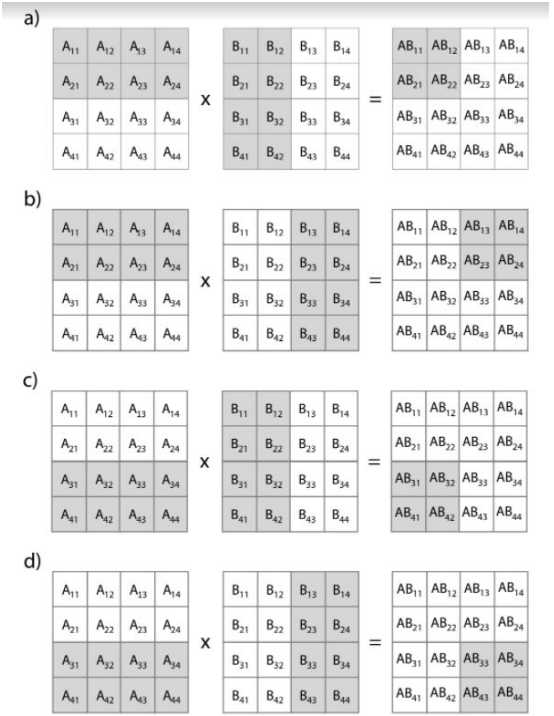
Kernel File

```
#include "block_mm.h"
void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,
                 blockmat &ABpartial, int it) {
    #pragma HLS DATAFLOW
    int counter = it % (SIZE/BLOCK_SIZE);
    static DTYPE A[BLOCK_SIZE][SIZE];
    if(counter == 0){ //only load the A rows when necessary
        loadA: for(int i = 0; i < SIZE; i++) {
            blockvec tempA = Arows.read();
            for(int j = 0; j < BLOCK_SIZE; j++) {
                #pragma HLS PIPELINE II=1
                A[j][i] = tempA.a[j];
            }
        }
    }
    DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
    partialsum: for(int k=0; k < SIZE; k++) {
        blockvec tempB = Bcols.read();
        for(int i = 0; i < BLOCK_SIZE; i++) {
            for(int j = 0; j < BLOCK_SIZE; j++) {
                AB[i][j] = AB[i][j] + A[i][k] * tempB.a[j];
            }
        }
    }
    writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
        for(int j = 0; j < BLOCK_SIZE; j++) {
            ABpartial.out[i][j] = AB[i][j];
        }
    }
}
```



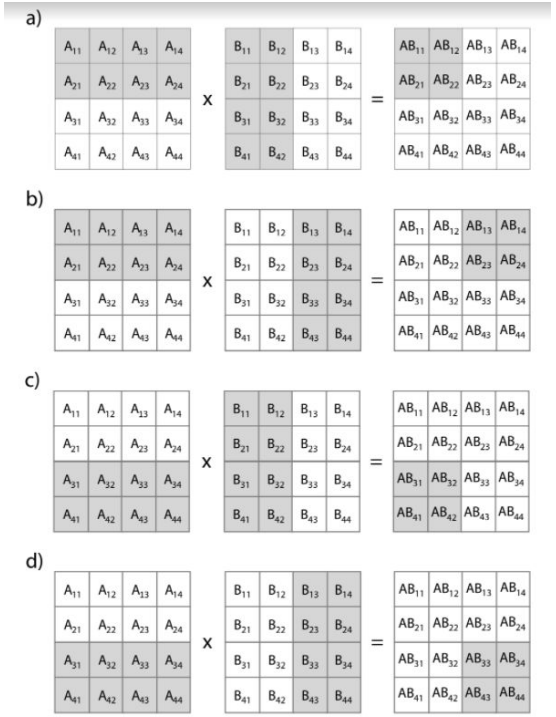
Kernel File

```
#include "block_mm.h"
void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,
                 blockmat &ABpartial, int it) {
    #pragma HLS DATAFLOW
    int counter = it % (SIZE/BLOCK_SIZE);
    static DTYPE A[BLOCK_SIZE][SIZE];
    if(counter == 0){ //only load the A rows when necessary
        loadA: for(int i = 0; i < SIZE; i++) {
            blockvec tempA = Arows.read();
            for(int j = 0; j < BLOCK_SIZE; j++) {
                #pragma HLS PIPELINE II=1
                A[j][i] = tempA.a[j];
            }
        }
    }
    DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
    partialsum: for(int k=0; k < SIZE; k++) {
        blockvec tempB = Bcols.read();
        for(int i = 0; i < BLOCK_SIZE; i++) {
            for(int j = 0; j < BLOCK_SIZE; j++) {
                AB[i][j] = AB[i][j] + A[i][k] * tempB.a[j];
            }
        }
    }
    writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
        for(int j = 0; j < BLOCK_SIZE; j++) {
            ABpartial.out[i][j] = AB[i][j];
        }
    }
}
```



Kernel File

```
#include "block_mm.h"
void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,
                 blockmat &ABpartial, int it) {
    #pragma HLS DATAFLOW
    int counter = it % (SIZE/BLOCK_SIZE);
    static DTYPE A[BLOCK_SIZE][SIZE];
    if(counter == 0){ //only load the A rows when necessary
        loadA: for(int i = 0; i < SIZE; i++) {
            blockvec tempA = Arows.read();
            for(int j = 0; j < BLOCK_SIZE; j++) {
                #pragma HLS PIPELINE II=1
                A[j][i] = tempA.a[j];
            }
        }
    }
    DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
    partialsum: for(int k=0; k < SIZE; k++) {
        blockvec tempB = Bcols.read();
        for(int i = 0; i < BLOCK_SIZE; i++) {
            for(int j = 0; j < BLOCK_SIZE; j++) {
                AB[i][j] = AB[i][j] + A[i][k] * tempB.a[j];
            }
        }
    }
    writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
        for(int j = 0; j < BLOCK_SIZE; j++) {
            ABpartial.out[i][j] = AB[i][j];
        }
    }
}
```



Testbench1

```
#include "block_mm.h"
#include <stdlib.h>
using namespace std;

void matmatmul_sw(DTYPE A[SIZE][SIZE], DTYPE B[SIZE][SIZE],
DTYPE out[SIZE][SIZE]){
    DTYPE sum = 0;
    for(int i = 0; i < SIZE; i++){
        for(int j = 0; j<SIZE; j++){
            sum = 0;
            for(int k = 0; k < SIZE; k++){
                sum = sum + A[i][k] * B[k][j];
            }
            out[i][j] = sum;
        }
    }
}

int main() {
    int fail = 0;
    hls::stream<blockvec> strm_matrix1("strm_matrix1");
    hls::stream<blockvec> strm_matrix2("strm_matrix2");
    blockvec strm_matrix1_element, strm_matrix2_element;
    blockmat block_out;
    DTYPE A[SIZE][SIZE], B[SIZE][SIZE];
    DTYPE matrix_swout[SIZE][SIZE], matrix_hwout[SIZE][SIZE];

    initmatrices: for(int i = 0; i < SIZE; i++){
        for(int j = 0; j < SIZE; j++){
            A[i][j] = rand() % 512;
            B[i][j] = rand() % 512;
            matrix_swout[i][j] = 0;
            matrix_hwout[i][j] = 0;
        }
    }

    //the remainder of this testbench is displayed in the next figure
```

Testbench1

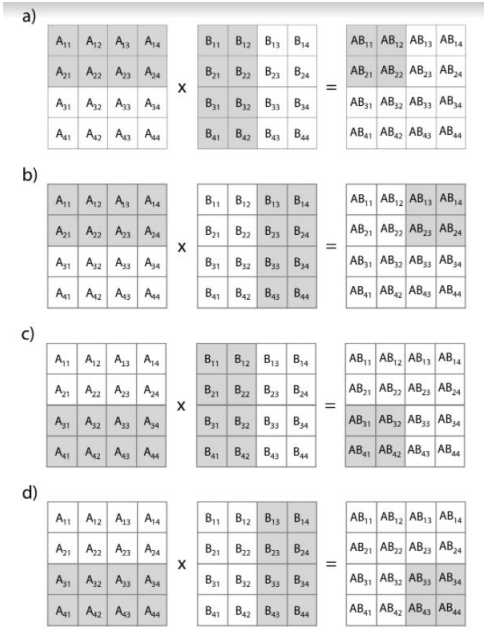
```
#include "block_mm.h"
#include <stdlib.h>
using namespace std;

void matmatmul_sw(DTYPE A[SIZE][SIZE], DTYPE B[SIZE][SIZE],
DTYPE out[SIZE][SIZE]){
    DTYPE sum = 0;
    for(int i = 0; i < SIZE; i++){
        for(int j = 0; j<SIZE; j++){
            sum = 0;
            for(int k = 0; k < SIZE; k++){
                sum = sum + A[i][k] * B[k][j];
            }
            out[i][j] = sum;
        }
    }
}

int main() {
    int fail = 0;
    hls::stream<blockvec> strm_matrix1("strm_matrix1");
    hls::stream<blockvec> strm_matrix2("strm_matrix2");
    blockvec strm_matrix1_element, strm_matrix2_element;
    blockmat block_out;
    DTYPE A[SIZE][SIZE], B[SIZE][SIZE];
    DTYPE matrix_swout[SIZE][SIZE], matrix_hwout[SIZE][SIZE];

    initmatrices: for(int i = 0; i < SIZE; i++){
        for(int j = 0; j < SIZE; j++){
            A[i][j] = rand() % 512;
            B[i][j] = rand() % 512;
            matrix_swout[i][j] = 0;
            matrix_hwout[i][j] = 0;
        }
    }

    //the remainder of this testbench is displayed in the next figure
```

Testbench1

```
#include "block_mm.h"
#include <stdlib.h>
using namespace std;

void matmatmul_sw(DTYPE A[SIZE][SIZE], DTYPE B[SIZE][SIZE],
DTYPE out[SIZE][SIZE]){
    DTYPE sum = 0;
    for(int i = 0; i < SIZE; i++){
        for(int j = 0; j<SIZE; j++){
            sum = 0;
            for(int k = 0; k < SIZE; k++){
                sum = sum + A[i][k] * B[k][j];
            }
            out[i][j] = sum;
        }
    }
}

int main() {
    int fail = 0;
    hls::stream<blockvec> strm_matrix1("strm_matrix1");
    hls::stream<blockvec> strm_matrix2("strm_matrix2");
    blockvec strm_matrix1_element, strm_matrix2_element;
    blockmat block_out;
    DTYPE A[SIZE][SIZE], B[SIZE][SIZE];
    DTYPE matrix_swout[SIZE][SIZE], matrix_hwout[SIZE][SIZE];

    initmatrices: for(int i = 0; i < SIZE; i++){
        for(int j = 0; j < SIZE; j++){
            A[i][j] = rand() % 512;
            B[i][j] = rand() % 512;
            matrix_swout[i][j] = 0;
            matrix_hwout[i][j] = 0;
        }
    }

    //the remainder of this testbench is displayed in the next figure
```

Testbench2

```
// The beginning of the testbench is shown in the previous figure
int main() {
    int row, col, it = 0;
    for(int it1 = 0; it1 < SIZE; it1 = it1 + BLOCK_SIZE) {
        for(int it2 = 0; it2 < SIZE; it2 = it2 + BLOCK_SIZE) {
            row = it1; //row + BLOCK_SIZE * factor_row;
            col = it2; //col + BLOCK_SIZE * factor_col;

            for(int k = 0; k < SIZE; k++) {
                for(int i = 0; i < BLOCK_SIZE; i++) {
                    if(it % (SIZE/BLOCK_SIZE) == 0) strm_matrix1_element.a[i] = A[row+i][k];
                    strm_matrix2_element.a[i] = B[k][col+i];
                }
                if(it % (SIZE/BLOCK_SIZE) == 0) strm_matrix1.write(strm_matrix1_element);
                strm_matrix2.write(strm_matrix2_element);
            }
            blockmatmul(strm_matrix1, strm_matrix2, block_out, it);

            for(int i = 0; i < BLOCK_SIZE; i++)
                for(int j = 0; j < BLOCK_SIZE; j++)
                    matrix_hwout[row+i][col+j] = block_out.out[i][j];
            it = it + 1;
        }
    }

    matmatmul_sw(A, B, matrix_swout);

    for(int i = 0; i<SIZE; i++)
        for(int j = 0; j<SIZE; j++)
            if(matrix_swout[i][j] != matrix_hwout[i][j]) { fail=1; }

    if(fail==1) cout << "failed" << endl;
    else cout << "passed" << endl;

    return 0;
}
```



Testbench2

```
// The beginning of the testbench is shown in the previous figure
int main() {
    int row, col, it = 0;
    for(int it1 = 0; it1 < SIZE; it1 = it1 + BLOCK_SIZE) {
        for(int it2 = 0; it2 < SIZE; it2 = it2 + BLOCK_SIZE) {
            row = it1; //row + BLOCK_SIZE * factor_row;
            col = it2; //col + BLOCK_SIZE * factor_col;

            for(int k = 0; k < SIZE; k++) {
                for(int i = 0; i < BLOCK_SIZE; i++) {
                    if(it % (SIZE/BLOCK_SIZE) == 0) strm_matrix1_element.a[i] = A[row+i][k];
                    strm_matrix2_element.a[i] = B[k][col+i];
                }
                if(it % (SIZE/BLOCK_SIZE) == 0) strm_matrix1.write(strm_matrix1_element);
                strm_matrix2.write(strm_matrix2_element);
            }
            blockmatmul(strm_matrix1, strm_matrix2, block_out, it);

            for(int i = 0; i < BLOCK_SIZE; i++)
                for(int j = 0; j < BLOCK_SIZE; j++)
                    matrix_hwout[row+i][col+j] = block_out.out[i][j];
            it = it + 1;
        }
    }

    matmatmul_sw(A, B, matrix_swout);

    for(int i = 0; i<SIZE; i++)
        for(int j = 0; j<SIZE; j++)
            if(matrix_swout[i][j] != matrix_hwout[i][j]) { fail=1; }

    if(fail==1) cout << "failed" << endl;
    else cout << "passed" << endl;

    return 0;
}
```

Outline

- Explain the Original Code/System/Pragmas
- **Analyze the Timing/Performance/Utilization**
- Share How I Optimize the Design

Synthesis Report & CoSim Report

Synthesis report

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
blockmatmul				-	233	2.330E3	-	152	-	dataflow	4	48	13095	35414	0
Block_split55_proc1				-	81	810.000	-	81	-	no	0	0	154	235	0
loadA				-	80	800.000	10	-	8	no	-	-	-	-	-
Loop_2_proc2				-	151	1.510E3	-	151	-	no	0	48	4155	1967	0
Loop_2_proc2_Pipeline_1				-	18	180.000	-	18	-	no	0	0	7	50	0
Loop 1				-	16	160.000	1	1	16	yes	-	-	-	-	-
Loop_2_proc2_Pipeline_partialsum	II Violation			-	130	1.300E3	-	130	-	no	0	48	4141	1798	0
partialsum	II Violation	Resource Limitation		-	128	1.280E3	16	16	8	yes	-	-	-	-	-
Loop_writeoutput_proc	II Violation			-	16	160.000	-	16	-	no	0	0	8656	33166	0
writeoutput	II Violation	Memory Dependency	1	-	14	140.000	6	3	4	yes	-	-	-	-	-

Co-simulation report

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	197	229	165	196	228	164
Block_split55_proc1	197	229	165	33	65	1
loadA	207	229	165	33	65	1
Loop_2_proc2	186	229	165	147	147	147
Loop_2_proc2_Pipeline_1	186	229	165	16	16	16
Loop 1						
Loop_2_proc2_Pipeline_partialsum	186	229	165	128	128	128
partialsum						
Loop_writeoutput_proc	186	229	165	14	14	14
writeoutput	186	229	165	15	15	15

Outline

- Explain the Original Code/System/Pragmas
- Analyze the Timing/Performance/Utilization
- **Share How I Optimize the Design**

Original Code vs Optimized Code1(pipeline in partialsum)

```
DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
partialsum: for(int k=0; k < SIZE; k++) {
  blockvec tempB = Bcols.read();
  for(int i = 0; i < BLOCK_SIZE; i++) {
    for(int j = 0; j < BLOCK_SIZE; j++) {
      AB[i][j] = AB[i][j] +  A[i][k] * tempB.a[j];
    }
  }
}
```

```
DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
partialsum: for(int k=0; k < SIZE; k++) {
  #pragma HLS PIPELINE II=1
  blockvec tempB = Bcols.read();
  for(int i = 0; i < BLOCK_SIZE; i++) {
    for(int j = 0; j < BLOCK_SIZE; j++) {
      AB[i][j] = AB[i][j] +  A[i][k] * tempB.a[j];
    }
  }
}
```

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
blockmatmul				-	233	2.330E3	-	152	-	dataflow	4	48	13095	35414	0
Block_split55_proc1				-	81	810.000	-	81	-	no	0	0	154	235	0
loadA				-	80	800.000	10	-	8	no	-	-	-	-	-
Loop_2_proc2				-	151	1.510E3	-	151	-	no	0	48	4155	1967	0
Loop_2_proc2_Pipeline_1				-	18	180.000	-	18	-	no	0	0	7	50	0
Loop 1				-	16	160.000	1	1	16	yes	-	-	-	-	-
Loop_2_proc2_Pipeline_partialsum	II Violation			-	130	1.300E3	-	130	-	no	0	48	4141	1798	0
partialsum	II Violation	Resource Limitation		-	128	1.280E3	16	16	8	yes	-	-	-	-	-
Loop_writeoutput_proc	II Violation			-	16	160.000	-	16	-	no	0	0	8656	33166	0
writeoutput	II Violation	Memory Dependency	1	-	14	140.000	6	3	4	yes	-	-	-	-	-

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
blockmatmul				-	132	1.320E3	-	82	-	dataflow	0	48	13749	35666	0
Block_split76_proc1				-	81	810.000	-	81	-	no	0	0	163	222	0
loadA				-	80	800.000	10	-	8	no	-	-	-	-	-
Loop_2_proc2				-	50	500.000	-	50	-	no	0	48	4540	2144	0
Loop_2_proc2_Pipeline_1				-	18	180.000	-	18	-	no	0	0	7	50	0
Loop 1				-	16	160.000	1	1	16	yes	-	-	-	-	-
Loop_2_proc2_Pipeline_partialsum				-	13	130.000	-	13	-	no	0	48	3998	1671	0
partialsum				-	11	110.000	5	1	8	yes	-	-	-	-	-
Loop_writeoutput_proc	II Violation			-	16	160.000	-	16	-	no	0	0	8656	33166	0
writeoutput	II Violation	Memory Dependency	1	-	14	140.000	6	3	4	yes	-	-	-	-	-

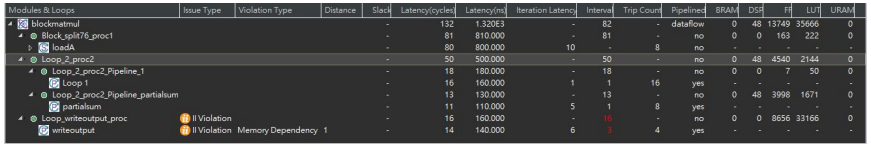
Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	197	229	165	196	228	164
Block_split55_proc1	197	229	165	33	65	1
loadA	207	229	165	33	65	1
Loop_2_proc2	186	229	165	147	147	147
Loop_2_proc2_Pipeline_1	186	229	165	16	16	16
Loop 1						
Loop_2_proc2_Pipeline_partialsum	186	229	165	128	128	128
partialsum						
Loop_writeoutput_proc	186	229	165	14	14	14
writeoutput	186	229	165	15	15	15

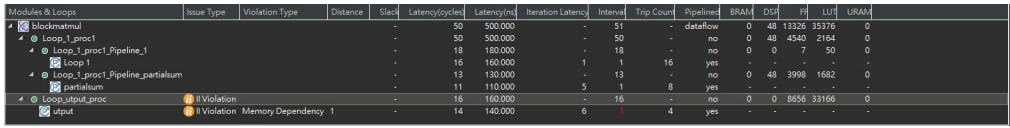
Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	96	128	64	95	127	63
Block_split76_proc1	96	128	64	33	65	1
loadA	106	128	64	33	65	1
Loop_2_proc2	85	128	64	46	46	46
Loop_2_proc2_Pipeline_1	85	128	64	16	16	16
Loop 1						
Loop_2_proc2_Pipeline_partialsum	85	128	64	11	11	11
partialsum	85	128	64	12	12	12
Loop_writeoutput_proc	85	128	64	14	14	14
writeoutput	85	128	64	15	15	15

Code1 vs Optimized Code2(modify loadA)

```
DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
partialsum: for(int k=0; k < SIZE; k++) {
  #pragma HLS PIPELINE II=1
  blockvec tempB = Bcols.read();
  for(int i = 0; i < BLOCK_SIZE; i++) {
    for(int j = 0; j < BLOCK_SIZE; j++) {
      AB[i][j] = AB[i][j] +  A[i][k] * tempB.a[j];
    }
  }
}
```

```
DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
 partialsum: for(int k=0; k < SIZE; k++) {
   #pragma HLS PIPELINE II=1
   blockvec tempA = Arows.read();
   blockvec tempB = Bcols.read();
   for(int i = 0; i < BLOCK_SIZE; i++) {
     for(int j = 0; j < BLOCK_SIZE; j++) {
       AB[i][j] = AB[i][j] +  tempA.a[i] * tempB.a[j];
     }
   }
 }
```





Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	96	128	64	95	127	63
Block_split76_proc1	96	128	64	33	65	1
loadA	106	128	64	33	65	1
Loop_2_proc2	85	128	64	46	46	46
Loop_2_proc2_Pipeline_1	85	128	64	16	16	16
Loop 1						
Loop_2_proc2_Pipeline_partialsum	85	128	64	11	11	11
partialsum	85	128	64	12	12	12
Loop_writeoutput_proc	85	128	64	14	14	14
writeoutput	85	128	64	15	15	15

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	62	62	62	61	61	61
Loop_1_proc1	62	62	62	46	46	46
Loop_1_proc1_Pipeline_1	62	62	62	16	16	16
Loop 1						
Loop_1_proc1_Pipeline_partialsum	62	62	62	11	11	11
partialsum	62	62	62	12	12	12
Loop_utput_proc	62	62	62	14	14	14
utput	62	62	62	15	15	15

Original Code vs Optimized Code

```
DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
partialsum: for(int k=0; k < SIZE; k++) {
  blockvec tempB = Bcols.read();
  for(int i = 0; i < BLOCK_SIZE; i++) {
    for(int j = 0; j < BLOCK_SIZE; j++) {
      AB[i][j] = AB[i][j] +  A[i][k] * tempB.a[j];
    }
  }
}
```

```
DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
 partialsum: for(int k=0; k < SIZE; k++) {
   #pragma HLS PIPELINE II=1
   blockvec tempA = Arows.read();
   blockvec tempB = Bcols.read();
   for(int i = 0; i < BLOCK_SIZE; i++) {
     for(int j = 0; j < BLOCK_SIZE; j++) {
       AB[i][j] = AB[i][j] +  tempA.a[i] * tempB.a[j];
     }
   }
 }
```

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
blockmatmul				-	233	2.330E3	-	152	-	dataflow	4	48	13095	35414	0
Block_split55_proc1				-	81	810.000	-	81	-	no	0	0	154	235	0
loadA				-	80	800.000	10	-	8	no	-	-	-	-	-
Loop_2_proc2				-	151	1.510E3	-	151	-	no	0	48	4155	1967	0
Loop_2_proc2_Pipeline_1				-	18	180.000	-	18	-	no	0	0	7	50	0
Loop 1				-	16	160.000	1	1	16	yes	-	-	-	-	-
Loop_2_proc2_Pipeline_partialsum	II Violation			-	130	1.300E3	-	130	-	no	0	48	4141	1798	0
partialsum	II Violation	Resource Limitation		-	128	1.280E3	16	16	8	yes	-	-	-	-	-
Loop_writeoutput_proc	II Violation			-	16	160.000	-	16	-	no	0	0	8656	33166	0
writeoutput	II Violation	Memory Dependency	1	-	14	140.000	6	3	4	yes	-	-	-	-	-

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
blockmatmul				-	50	500.000	-	51	-	dataflow	0	48	13326	35376	0
Loop_1_proc1				-	50	500.000	-	50	-	no	0	48	4540	2164	0
Loop_1_proc1_Pipeline_1				-	18	180.000	-	18	-	no	0	0	7	50	0
Loop 1				-	16	160.000	1	1	16	yes	-	-	-	-	-
Loop_1_proc1_Pipeline_partialsum				-	13	130.000	-	13	-	no	0	48	3998	1682	0
partialsum				-	11	110.000	5	1	8	yes	-	-	-	-	-
Loop_utput_proc	II Violation			-	16	160.000	-	16	-	no	0	0	8656	33166	0
utput	II Violation	Memory Dependency	1	-	14	140.000	6	3	4	yes	-	-	-	-	-

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	197	229	165	196	228	164
Block_split55_proc1	197	229	165	33	65	1
loadA	207	229	165	33	65	1
Loop_2_proc2	186	229	165	147	147	147
Loop_2_proc2_Pipeline_1	186	229	165	16	16	16
Loop 1						
Loop_2_proc2_Pipeline_partialsum	186	229	165	128	128	128
partialsum						
Loop_writeoutput_proc	186	229	165	14	14	14
writeoutput	186	229	165	15	15	15

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
blockmatmul	62	62	62	61	61	61
Loop_1_proc1	62	62	62	46	46	46
Loop_1_proc1_Pipeline_1	62	62	62	16	16	16
Loop 1						
Loop_1_proc1_Pipeline_partialsum	62	62	62	11	11	11
partialsum	62	62	62	12	12	12
Loop_utput_proc	62	62	62	14	14	14
utput	62	62	62	15	15	15

The End

github