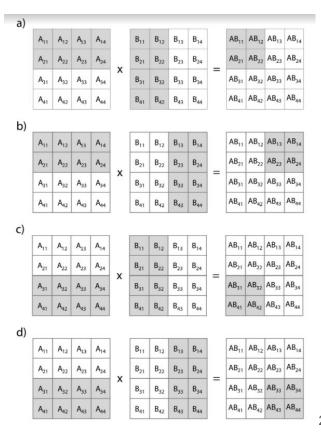
Block Matrix Multiplication HLS Course

賴志杰 2022/3/30

Block Matrix Multiplication

- 1. divide the input arrays A and B into blocks
- 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

block mm.cpp

- Header File
- Kernel Function File
- Testbench File

_			, ,
□ bloc	ck_mm.h	Fresh commit of version 1.0	4 years ago
🗅 bloo	ckmatmul_test.cpp	Fresh commit of version 1.0	4 years ago
Ď bloo	ckmatmul_test_init.cpp	Fresh commit of version 1.0	4 years ago

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
```

7

```
#include "block_mm.h"

void blockmatmul(hls::stream<blockvec> &Arows, hls::stream<blockvec> &Bcols,
```

Kernel File

```
blockmat &ABpartial, int it) {
#pragma HLS DATAFLOW
int counter = it % (SIZE/BLOCK_SIZE);
static DTYPE A[BLOCK_SIZE][SIZE];
                                                                                   a)
if(counter == 0){ //only load the A rows when necessary
    loadA: for(int i = 0; i < SIZE; i++) {
                                                                                                                               AB, AB, AB, AB,
         blockvec tempA = Arows.read();
         for(int j = 0; j < BLOCK_SIZE; j++) {
              #pragma HLS PIPELINE II=1
                                                                                     b)
              A[j][i] = tempA.a[j];
                                                                                         A<sub>11</sub> A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
                                                                                                            B<sub>31</sub> B<sub>32</sub>
                                                                                         A<sub>41</sub> A<sub>42</sub> A<sub>43</sub> A<sub>44</sub>
                                                                                                            B<sub>41</sub> B<sub>42</sub> B<sub>43</sub>
DTYPE AB[BLOCK\_SIZE][BLOCK\_SIZE] = \{ 0 \};
partialsum: for(int k=0; k < SIZE; k++) {
                                                                                             A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
    blockvec tempB = Bcols.read();
                                                                                             A<sub>22</sub> A<sub>23</sub> A<sub>24</sub>
    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];
                                                                                     d)
                                                                                         A<sub>11</sub> A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
                                                                                                                              AB21 AB22 AB23 AB
writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
                                                                                                               B<sub>42</sub> B<sub>43</sub>
    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];
                                                                            a)
    if(counter == 0){ //only load the A rows when necessary
        loadA: for(int i = 0; i < SIZE; i++) {
                                                                                                                    AB, AB, AB, AB,
            blockvec tempA = Arows.read();
            for(int j = 0; j < BLOCK_SIZE; j++) {
                #pragma HLS PIPELINE II=1
                                                                              b)
                A[j][i] = tempA.a[j];
                                                                                  A<sub>11</sub> A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
                                                                                                     B<sub>32</sub> |
                                                                                  A<sub>41</sub> A<sub>42</sub> A<sub>43</sub> A<sub>44</sub>
                                                                                                   B<sub>41</sub> B<sub>42</sub> B<sub>43</sub>
    DTYPE AB[BLOCK_SIZE][BLOCK_SIZE] = { 0 };
                                                                              c)
    partialsum: for(int k=0; k < SIZE; k++) {
                                                                                      A<sub>12</sub> A<sub>13</sub>
        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];
                                                                              d)
    writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
                                                                                                      B<sub>42</sub> B<sub>43</sub>
        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];
                                                                              a)
    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
                                                                                b)
                A[j][i] = tempA.a[j];
                                                                                    A<sub>11</sub> A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
                                                                                                        B<sub>32</sub> |
                                                                                    A<sub>41</sub> A<sub>42</sub> A<sub>43</sub> A<sub>44</sub>
                                                                                                     B<sub>41</sub> B<sub>42</sub> B<sub>43</sub>
    DTYPE AB[BLOCK\_SIZE][BLOCK\_SIZE] = \{ 0 \};
    partialsum: for(int k=0; k < SIZE; k++) {
                                                                                       A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
        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];
                                                                                d)
                                                                                        A<sub>12</sub> A<sub>13</sub> A<sub>14</sub>
   writeoutput: for(int i = 0; i < BLOCK_SIZE; i++) {
                                                                                                        B<sub>42</sub>
        for(int j = 0; j < BLOCK_SIZE; j++) {
            ABpartial.out[i][j] = AB[i][j];
```

AB, AB, AB, AB,

AB₂₁ AB₂₂ AB₂₃ AB₂
AB₃₁ AB₃₂ AB₃₃ AB₃

```
#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;
        }
}</pre>
```

```
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;
    }
}
</pre>
```

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

```
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;
```

```
#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;
}
```

```
int main() {
   int row, col, it = 0;
                                                                                                                                              AB, AB, AB, AB,
   for(int it1 = 0; it1 < SIZE; it1 = it1 + BLOCK_SIZE) {
       for(int it2 = 0; it2 < SIZE; it2 = it2 + BLOCK_SIZE) {
                                                                                                                                               AB<sub>31</sub> AB<sub>32</sub> AB<sub>33</sub> AB<sub>3</sub>
           row = it1; //row + BLOCK_SIZE * factor_row;
                                                                                                                                              AB41 AB42 AB43 AB44
           col = it2; //col + BLOCK_SIZE * factor_col;
                                                                                                           b)
           for(int k = 0; k < SIZE; k++) {
                                                                                                                                              AB21 AB22 AB23 AB2
               for(int i = 0; i < BLOCK_SIZE; i++) {
                                                                                                                                              AB31 AB32 AB33 AB3
                   if(it % (SIZE/BLOCK_SIZE) == 0) strm_matrix1_element.a[i] = A[row+i][k];
                   strm_matrix2_element.a[i] = B[k][col+i];
                                                                                                          c)
               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];
                                                                                                                                              AB, AB, AB, AB,
           it = it + 1:
   matmatmul_sw(A, B, matrix_swout);
   for(int i = 0; i < SIZE; i++)
       for(int j = 0; j < SIZE; j++)
```

```
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;
}</pre>
```

// The beginning of the testbench is shown in the previous figure

```
// 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	DSF	FF	LUT	URAM
🛛 🔯 blockmatmul		es e		-	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			no					
▲ @ Loop_2_proc2					151	1.510E3		151		no		48	4155	1967	
■ Loop_2_proc2_Pipeline_1					18	180.000		18		no				50	
🕑 Loop 1					16	160.000			16	yes					
▲ ⑤ Loop_2_proc2_Pipeline_partial	sum 📆 II Violation				130	1.300E3				no		48	4141	1798	
partialsum	📆 II Violation	Resource Limitation			128	1.280E3	16		8	yes					
▲ ⑤ Loop_writeoutput_proc	📆 II Violation				16	160.000				no			8656	33166	
writeoutput	📆 II Violation	Memory Dependency			14	140.000				yes					

Co-simulation report.

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
▲ 図 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
	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
riteoutput	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];
        }
}</pre>
```

		Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Interval	Trip Count	Pipelined	BRAM	DSP	FF		URAM
4 8	🔋 blockmatmul					233	2.330E3	152		dataflow	4	48	13095	35414	0
	Block_split55_proc1														0
	D S loadA						800.000								
	● Loop_2_proc2						1.510E3							1967	0
	■ Coop_2_proc2_Pipeline_1						180.000								0
	D Loop 1						160.000								
	■ O Loop 2 proc2 Pipeline partialsum	m II Violation					1.300E3								0
	partialsum	1 Violation	Resource Limitation				1.280E3			yes					-
	Loop_writeoutput_proc	1 Violation					160.000						8656	33166	0
	P writeoutput	📆 II Violation	Memory Dependency				140.000								-

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	
■ Loop_2_proc2	186	229	165	147	147	147
■ Loop_2_proc2_Pipeline_1 ② Loop 1	186	229	165	16	16	16
■ Loop_2_proc2_Pipeline_partialsum ② partialsum	186	229	165	128	128	128
■ Loop_writeoutput_proc	186	229	165	14	14	14
R writeoutput	186	229	165	15	15	15

```
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];
        }
    }
}</pre>
```

	Issue Type		Distance	Latency(cycles)	Latency(ns)		Interval	Trip Count	Pipelined	BRAM	DSP	FF		URAM
				132	1.320E3	-	82		dataflow	0	48	13749	35666	0
▲ @ Block_split76_proc1					810.000									
D S loadA					800.000									
4 ⑤ Loop 2 proc2														
					180.000									
D Loop 1					160.000									
■ Loop_2_proc2_Pipeline_partialsun					130.000									
partialsum					110.000									
■ B Loop_writeoutput_proc	(1) Il Violation				160.000								33166	
writeoutput	(i) II Violation	Memory Dependency			140.000									

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
▲	96	128	64	95	127	63
■ Block_split76_proc1	96	128	64	33	65	
▷ 🕃 loadA	106	128	64	33	65	
▲ © Loop_2_proc2	85	128	64	46	46	46
■ Loop_2_proc2_Pipeline_1 P Loop 1	85	128	64	16	16	16
■ Loop_2_proc2_Pipeline_partialsum	85	128	64	11	11	11
partialsum	85	128	64	12	12	12
▲ ⑤ Loop_writeoutput_proc	85	128	64	14	14	14
riteoutput	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];
        }
    }
}</pre>
```

```
Monther & Lorgo Sun Type | Volation Type | Datance | Sard | LatercyCycled | La
```

```
Modules & Loops
                                                                              Min Latency
 lockmatmul
                                          128
                                                64
 ■ Block split76 proc1
                                                64
   128

■ loop_2_proc2

                                          128
    ■ O Loop 2 proc2 Pipeline 1
                                          128
                                                64
        P Loop 1
    ■ Loop_2_proc2_Pipeline_partialsum 85
                                          128
                                                64
        partialsum
                                          128

■ O Loop writeoutput proc

                                                64
      mriteoutput
```

```
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];
        }
    }
}</pre>
```

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM	
				-	50	500.000		51		dataflow	0	48	13326	35376	0	
						500.000							4540			
■ B Loop_1_proc1_Pipeline_1						180.000										
Coop 1						160,000										
■ Loop_1_proc1_Pipeline_partialsum						130.000							3998			
P partialsum						110.000										
✓ ⑥ Loop_utput_proc	📆 II Violation															
utput	1 Violation	Memory Dependency				140.000				yes						

Modules & Loops	Avg II	MaxII	Min II	Avg Latency	Max Latency	Min Latency
	62	62	62	61	61	61
■ Loop_1_proc1	62	62	62	46	46	46
▲ ② Loop_1_proc1_Pipeline_1 P Loop 1	62	62	62	16	16	16
 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];
        }
}</pre>
```

```
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];
        }
    }
}</pre>
```

```
| Stude & Loops | Stude Type | Violation Type | Distance | Stade | Latency(cycles | Latenc
```

	Issue Type		Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM	
				50	500.000		51		dataflow	0	48	13326	35376	0	
▲ @ Loop_1_proc1					500.000							4540			
■ Loop_1_proc1_Pipeline_1					180.000										
D Loop 1					160.000										
■ Loop_1_proc1_Pipeline_partialsum					130.000							3998			
partialsum					110.000										
	📆 II Violation														
c utput	(1) II Violation	Memory Dependency			140.000										

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
	197	229	165	196	228	164
■ Block_split55_proc1	197	229	165	33	65	1
♪ 🕃 loadA	207	229	165	33	65	
▲ O Loop_2_proc2	186	229	165	147	147	147
■ Loop_2_proc2_Pipeline_1 P Loop 1	186	229	165	16	16	16
■ Loop_2_proc2_Pipeline_partialsum P partialsum	186	229	165	128	128	128
■ Loop_writeoutput_proc	186	229	165	14	14	14
riteoutput	186	229	165	15	15	15

Modules & Loops	Avg II	MaxII	Min II	Avg Latency	Max Latency	Min Latency
▲ 🔯 blockmatmul	62	62	62	61	61	61
■ O Loop_1_proc1	62	62	62	46	46	46
▲ ② Loop_1_proc1_Pipeline_1 ② Loop 1	62	62	62	16	16	16
 Loop_1_proc1_Pipeline_partialsum 	62	62	62	11	11	11
partialsum	62	62	62	12	12	12
▲ ⑤ Loop_utput_proc	62	62	62	14	14	14
🕝 utput	62	62	62	15	15	15

The End

github