2022-2023

Σχεδίαση υλικού λογισμικού Εργαστήριο 1ο

ΝΙΚΟΛΑΟΣ ΓΙΑΝΝΟΠΟΥΛΟΣ 9629 NGIANNOP@ECE.AUTH.GR
ΔΗΜΗΤΡΙΟΣ ΑΝΔΡΟΝΙΚΟΥ 9836 DIMITRIOSA@ECE.AUTH.GR

Εισαγωγή

Στο 1^{ο} εργαστήριο θα υλοποιήσουμε των αλγόριθμο υπολογισμού πολλαπλασιασμών πινάκων καθώς θα αναλύσουμε με βάση το εργαλείο Vivado HLS 2019.2

Υλοποίηση

```
#include <stdio.h>
#include <stdint.h>


#define lm 6
#define ln 6
#define lp 6

#define n 1<<ln
#define m 1<<lm
#define p 1<<lp

//#define ONLY_ARRAY_PARTITION
//#define ONLY_PIPELINE
//#define ONLY_UNROLL

//#define do_all

#define do_all2 // ln = lp = lm = 6

//A is NxM matrix
//B is MxP

//Output C NxP

void MATRIX_MUL(uint8_t A[n][m] ,uint8_t B[m][p], uint32_t C[n][p]){

#ifdef ONLY_ARRAY_PARTITION
// Do Array Partition
	// check only ln
	if(ln == 1 && lp == 1 && lm == 6){
#pragma HLS array_partition variable=A block factor= 2
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 2
	}else if(ln == 2 && lp == 1 && lm == 6){
#pragma HLS array_partition variable=A block factor= 4
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 2
	}
	else if(ln == 3 && lp == 1 && lm == 6){
#pragma HLS array_partition variable=A block factor= 8
#pragma HLS array_partition variable=B block factor= 64
```

```
#pragma HLS array_partition variable=C block factor= 2
      }

      else if(ln == 4 && lp == 1 && lm == 6){
#pragma HLS array_partition variable=A block factor= 16
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 2
      }

      else if(ln == 5 && lp == 1 && lm == 6){
#pragma HLS array_partition variable=A block factor= 32
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 2
      }

      else if(ln == 6 && lp == 1 && lm == 6){
#pragma HLS array_partition variable=A block factor= 64
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 2
      }
      //Checking for lp
      else if(ln == 2 && lp == 2  && lm == 6){
#pragma HLS array_partition variable=A block factor= 4
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 4
      }

      else if(ln == 3 && lp == 3  && lm == 6){
#pragma HLS array_partition variable=A block factor= 8
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 8
      }

      else if(ln == 4 && lp == 4  && lm == 6){
#pragma HLS array_partition variable=A block factor= 16
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 16
      }
      else if(ln == 5 && lp == 5  && lm == 6){
#pragma HLS array_partition variable=A block factor= 32
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 32
      }
      else if(ln == 6 && lp == 6  && lm == 6){
#pragma HLS array_partition variable=A block factor= 64
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 64
      }

#endif

#ifdef ONLY_PIPELINE
#pragma HLS pipeline II=1
#endif

#ifdef do_all
#pragma HLS array_partition variable=A block factor= 16
#pragma HLS array_partition variable=B block factor= 64
#pragma HLS array_partition variable=C block factor= 16
```

```
#pragma HLS pipeline II=5
#endif

	for (int i = 0 ; i < n; i++){
#ifdef ONLY_UNROLL
#pragma HLS unroll region skip_exit_check
#endif

#ifdef do_all
#pragma HLS unroll region skip_exit_check
#endif
		for(int k = 0; k < m ; k++){
			for(int j = 0; j < p; j++){
				C[i][j] += A[i][k] * B[k][j];
			}
		}
	}
}
```

Χρησιμοποιήσαμε τα Define ώστε να γίνεται ποιο εύκολη ανάλυση για την βελτιστοποίηση, οπού τα ONLY_ARRAY_PARTITION κάνει μόνο array partition και στο τέλος κάνουμε όλα μαζί με το define do_all και για την περιπτώσεις που έχουμε ln = lp = lm = 6 με την βέλτιστη λύση χρησιμοποιούμε το define do_all. Το αρχείο με όνομα MATRIX_MUL.cpp το οποίο αποτελείται από 3^{ης} πίνακες. Οι 2 είναι οι πίνακες εισόδου και ο τρίτος C είναι ο πίνακας που παίρνουμε τα αποτελέσματα.

Στην συνέχεια γράφουμε και το testbench αρχείο οπού βλέπουμε αν λειτουργεί ορθά η συνάρτηση MATRIX_MUL.

```
#include <stdio.h>
#include <stdint.h>
#include <random>
#include <iostream>


using namespace std;

#define lm 6
#define ln 6
#define lp 6

#define n 1<<ln
#define m 1<<lm
#define p 1<<lp

void MATRIX_MUL(uint8_t  A[n][m] , uint8_t  B[m][p] , uint32_t  C[n][p]);

int main(){

	uint32_t  C[n][p];
	uint8_t  A[n][m] , B[m][p];
	uint32_t C2[n][p]; // matrix to check if results is correct

	std::minstd_rand simple_rand;
```

```
	std::uniform_int_distribution<uint8_t> uniform_0_255(0,255);

	for(int i = 0; i < n; i++){
		for(int j = 0; j < m; j++){
			A[i][j] =uniform_0_255(simple_rand);
		}
	}

	for(int i = 0; i < m; i++){
		for(int j = 0; j < p; j++){
			B[i][j] = uniform_0_255(simple_rand);
		}
	}

	for(int i = 0; i < n; i++){
		for(int j = 0; j < p; j++){
			C2[i][j] = 0;
			C[i][j] = 0;
		}
	}

	printf("Matrix A\n");
	for(int i = 0; i < n; i++){
		for(int j = 0; j < m; j++){
			//printf("A[%d][%d] = %d\n",i,j,A[i][j]);
			printf("%d ",A[i][j]);
		}
		printf("\n");
	}

	printf("\nMatrix B\n");
	for(int i = 0; i < m; i++){
		for(int j = 0; j < p; j++){
			//printf("B[%d][%d] = %d\n",i,j,B[i][j]);
			printf("%d ",B[i][j]);
		}
		printf("\n");
	}

	MATRIX_MUL(A, B, C);

	bool t = true;
	for (int i = 0 ; i < n; i++){
		for(int k = 0; k < m ; k++){
			for(int j = 0; j < p; j++){
				C2[i][j] += (A[i][k] * B[k][j]);
			}
		}
	}

	for (int i = 0 ; i < n; i++){
			for(int j = 0; j < p; j++){
				if(C2[i][j] != C[i][j]) t = false;
			}
	}
	printf("\nFinal Matrix\n");
	for(int i = 0; i < n; i++){
		for(int j = 0; j < p; j++){
			printf("%d ",C[i][j]);
		}
		printf("\n");
	}
	if(t){
		printf("TEST PASSED!\n");
	}
	else{
		printf("TEST FAILED!\n");
	}
}
```

Για να γίνει σωστή υλοποιήσει του $n= 2^{ln}$, $m= 2^{lm}$,$p = 2^{lp}$ θα πρέπει να γίνει 1 << lj , οπού j κάποιο από τα lm,ln,lp. Αυτό θα εκτελεστεί με την λογική αυτή $1 * 2^{lj}$ αρά αν το lj είναι 1 τότε έχουμε 1 * 2 → 2 οπού είναι σαν να shift-αραμε κατά μια θέση αριστερά το 1 → $(0000\_0001)_2$ →$(0000\_0010)_2$

Καθώς φαινεται και η δηλωση τον πινακων A,B,C & C2 χρησιμοποιούμε την δήλωση αυτόν τον πινάκων με τα καταλληλά μεγέθη

Χρησιμοποιούμε το `minstd_rand` οπού είναι μια απλή πολλαπλασιαστική συμβατή ψευδοτυχαία γεννήτρια αριθμών σε συνδυασμό με το uniform_0_255(0,255) για να καθοριστεί το εύρος των τυχαίων αριθμών.

Καθώς κάνουμε Run C Synthesis για το ερώτημα 2 προκύπτει ότι

Estimated clock period	4.849 ns
Worst case latency	5.326 ms
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	85
Number of LUTs used	171

Ερώτημα 3ο τρέχοντας το C/RTL cosimulation προκύπτουν τα έξεις

Total Execution Time	5326275 ns
Min latency	532609
Avg. latency	532609
Max latency	532609

Ερώτημα 4ο

Αρχικά κάναμε array partition και προκύπτουν τα έξεις αποτελέσματα κρατώντας το lm σταθερό και ίσο με 6

- ln = 1
- lp = 1

Estimated clock period	4.472 ns
Worst case latency	7.730 us
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	48
Number of LUTs used	445

Οπού ο χρόνος καθυστερήσεις αυξήθηκε από ms → us

Στην συνέχεια κάνουμε

- In = 2
- Ip = 1

Estimated clock period	4.472 ns
Worst case latency	15.450 us
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	67
Number of LUTs used	723

Για In = 3 , Ip = 1

Estimated clock period	4.472 ns
Worst case latency	30.890 us
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	97
Number of LUTs used	729

Για In = 4 , Ip = 1

Estimated clock period	4.472 ns
Worst case latency	61.770 us
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	151
Number of LUTs used	729

Για In = 5 , Ip = 1

Estimated clock period	5.036 ns
Worst case latency	0.124 ms
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	253
Number of LUTs used	729

Έχουμε βελτιώσει us → ms

Για ln = 6, lp = 1

Estimated clock period	5.036 ns
Worst case latency	0.247 ms
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	451
Number of LUTs used	731

Καθώς αυξάνετε μόνο το ln και παραμένουν σταθερά τα lp , lm βλέπουμε ότι ο χρόνος καθυστερήσεις αυξάνετε αλλά όταν είναι κοντά στο lm τότε μειώνεται.

Καθώς δοκιμάσουμε για ιδίες τιμές ln = lp

- ln = lp = 2 , lm = 6

Estimated clock period	4.472 ns
Worst case latency	25.690 us
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	71
Number of LUTs used	251

- ln = lp = 3 , lm = 6

Estimated clock period	4.472 ns
Worst case latency	92.330 us

Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	116
Number of LUTs used	331

- ln = lp = 4 , lm = 6

Estimated clock period	4.472 ns
Worst case latency	0.348 ms
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	209
Number of LUTs used	547

- ln = lp = 5 , lm = 6

Estimated clock period	5.036 ns
Worst case latency	1.352 ms
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	406
Number of LUTs used	1051

- ln = lp = 6 , lm = 6

Estimated clock period	5.106 ns
Worst case latency	5.326 ms
Number of DSP48E used	1
Number of BRAMs used	0
Number of FFs used	827
Number of LUTs used	1893

Καταλήγουμε ότι αν όλα είναι περίπου ιδιά τότε προκύπτει να κάνουμε βελτιστοποίηση.

Η βέλτιστη λύση που προκύπτει είναι In = Ip = 4 , Im = 6.

Χρησιμοποιώντας pipeline υλοποίηση προκύπτει ότι είναι ανέφικτο με την υλοποίηση In = Ip = 4 & Im = 6

Χρησιμοποιώντας loop unroll προκύπτει ότι

Estimated clock period	8.493 ns
Worst case latency	82.250 us
Number of DSP48E used	528
Number of BRAMs used	0
Number of FFs used	9901
Number of LUTs used	38206

Τώρα χρησιμοποιώντας σε συνδυασμό όλα μαζί (ifdef do_all)προκύπτει το έξεις ,χωρίς Pipeline

Estimated clock period	8.660 ns
Worst case latency	6.570 us
Number of DSP48E used	528
Number of BRAMs used	0
Number of FFs used	8372
Number of LUTs used	43883

Χρησιμοποιούμε Im = 6 =In = Ip = 6 με την βέλτιστη λύση που υλοποιήθηκε παραπάνω αλλά χωρίς pipeline.

Χρησιμοποιήθηκαν τα έξεις

- `#pragma HLS array_partition variable=A block factor= 16`
- `#pragma HLS array_partition variable=B block factor= 64`
- `#pragma HLS array_partition variable=C block factor= 16`
- `#pragma HLS unroll region skip_exit_check`

Δεν γινόταν η χρήση του Pipeline γιατί υπάρχει μεγάλη εξάρτηση μεταξύ τον τιμών και είναι ανέφικτο να γίνει και προκύπτει

Estimated clock period	8.660 ns
Worst case latency	42.250 us
Number of DSP48E used	2112
Number of BRAMs used	0
Number of FFs used	38823
Number of LUTs used	160137
Total Execution Time	42435 ns
Min latency	4225 ns
Avg. latency	4225 ns
Max latency	4225 ns

Συγκρίνοντας την βέλτιστη με την αρχική λύση προκύπτει ότι

Αρχικό:

Estimated clock period	4.849 ns
Worst case latency	5.326 ms
Total Execution Time	5326275 ns
Min latency	532609
Avg. latency	532609
Max latency	532609

Βέλτιστο:

Estimated clock period	8.660 ns
Worst case latency	42.250 us
Total Execution Time	42435 ns
Min latency	4225 ns
Avg. latency	4225 ns
Max latency	4225 ns