ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ

Προηγμένα Θέματα
Αρχιτεκτονικής Υπολογιστών

4η Άσκηση
Ακ. έτος 2011-2012

Γρηγόρης Λύρας Α.Μ.: 03109687

21 Ιουλίου 2012

Μέρος Α΄

1 Εισαγωγή

Στην άσκηση αυτή προσομοιώνουμε με χρήση του Simics τον πολλαπλασιασμό δύο τετραγωνικών πινάκων Α και Β χρησιμοποιώντας διάφορες τεχνικές βελτιστοποίησης.

2 Περιβάλλον Προσομοίωσης

Για το εκτελέσιμο χρησιμοποιούμε γλώσσα C και gcc, χωρίς optimization flags.

```
target$ gcc -O1 -o executable partA.c
```

2.1 Προσομοίωση

Ο κώδικας που μας δόθηκε ήταν ο ακόλουθος:

```c
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : partA.c
 * Creation Date : 16-07-2012
 * Last Modified : Mon 16 Jul 2012 01:46:39 PM EEST
 * Created By : Greg Liras <gregliras@gmail.com>
 _.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-.*/

#include <stdio.h>
#include <stdlib.h>
#define __MAGIC_CASSERT(p) do {                         \
    typedef int __check_magic_argument[(p) ? 1 : -1];   \
} while (0)

#define MAGIC(n) do {                       \
    __MAGIC_CASSERT(!(n));                  \
    __asm__ __volatile__ ("xchg %bx,%bx");  \
} while (0)

#define MAGIC_BREAKPOINT MAGIC(0)



inline int min(int a, int b)
{
    if(a<=b)return a;
    else return b;
}
void init_matrix(float **mat, int n)
{
    unsigned int i,j;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            mat[i][j] = (float)(i+j);
}
int main(int argc, char **argv)
{
    float **A,**B,**C;
    int i,j,k,N;
    N=atoi(argv[1]);
    A=(float**)malloc(N*sizeof(float*));

    for(i=0; i<N; i++)
        A[i]=(float*)malloc(N*sizeof(float));

    B=(float**)malloc(N*sizeof(float*));

    for(i=0; i<N; i++)
        B[i]=(float*)malloc(N*sizeof(float));
```

```
49
50      C=(float**)malloc(N*sizeof(float*));
51
52      for(i=0; i<N; i++)
53          C[i]=(float*)malloc(N*sizeof(float));
54
55      fprintf(stderr, "Initializing matrices...\n");
56      init_matrix(A, N);
57      init_matrix(B, N);
58      init_matrix(C, N);
59      MAGIC_BREAKPOINT;
60      for(i=0; i<N; i++) {
61          for(j=0; j<N; j++)
62              for(k=0; k<N; k++)
63                  C[i][j] += A[i][k]*B[k][j];
64      }
65      MAGIC_BREAKPOINT;
66      return 0;
67  }
```

2.2 Ιεραρχία μνήμης και μοντέλο απόδοσης

Χρησιμοποιήσαμε την ιεραρχία μνήμης όπως μας δίνεται στο Παράρτημα. Αυτή έχει δύο επίπεδα κρυφής μνήμης L1 (2 way set associative 64 bytes $\times$ 512 lines write-through LRU policy) και L2 (4 way set associative 128 bytes $\times$ 1024 lines write-back LRU policy).

	assoc	line size	lines	size
L1 instruction cache	2	64	512	32768 = 32 KB
L1 data cache	2	64	512	32768 = 32 KB
L2 cache	4	128	1024	131072 = 128 KB

Πίνακας 1: Cache Hierarchy

Στο μοντέλο που προσομοιώνουμε οι κύκλοι υπολογίζονται χωρίς penalty από το Simics. Για να υπολογίσουμε τους κύκλους θεωρούμε πως πρόσβαση στην L1 cache γίνεται σε 1 κύκλο, στην L2 σε 20 κύκλους και στη μνήμη σε 300 κύκλους οπότε έχουμε τον τύπο:

$$Cycles = Inst + L1_{Accesses} * L1_{Time} + L2_{Accesses} * L2_{Time} + Mem_{Accesses} * Mem_{Time} \quad (1)$$

2.3 Αρχική έκδοση

Προσομοιώσαμε τον αρχικό κώδικα, όπως φαίνεται παραπάνω και πήραμε τα ακόλουθα αποτελέσματα.

Cycles	1523400216
L1 miss ratio	0.029
L2 miss ratio	0.015

Πίνακας 2: Μετρικές πρώτης εκτέλεσης

3 Τεχνικές Βελτιστοποίησης

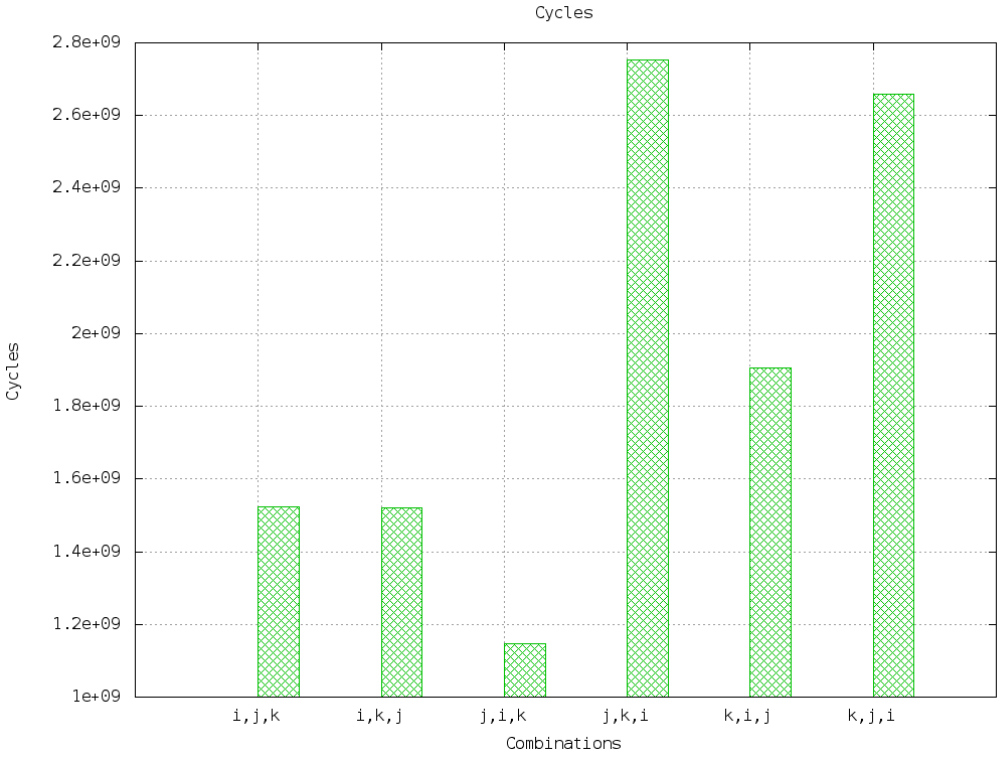
3.1 Loop Interchange

Για την προσομοίωση κάναμε 6 εκτελέσεις για κάθε διάταξη των i,j,k όπως φαίνεται στον πίνακα 3.

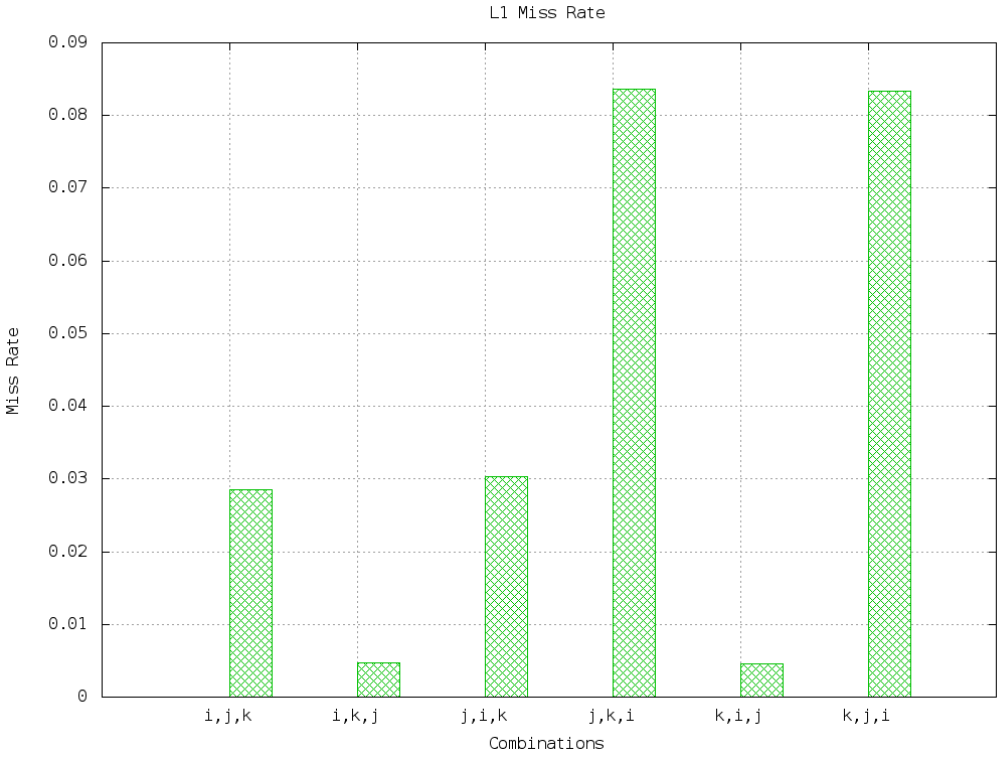
Σειρά προσομοίωσης	Διάταξη
1	i,j,k
2	i,k,j
3	j,i,k
4	j,k,i
5	k,i,j
6	k,j,i

Πίνακας 3: Δυνατές Διατάξεις

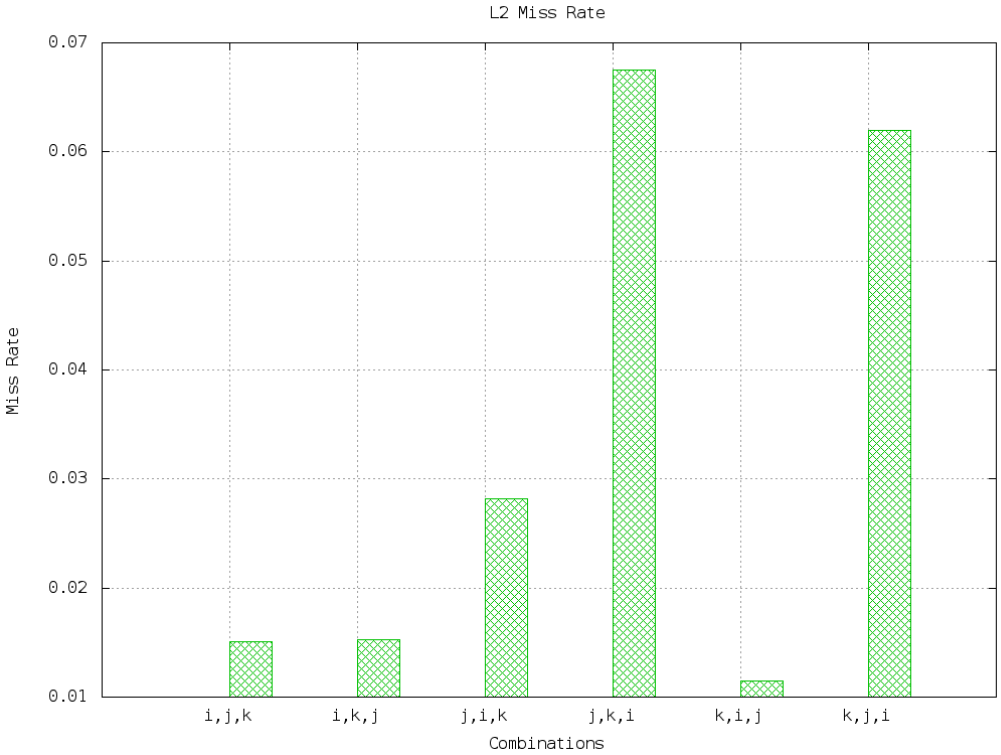
Τα αποτελέσματα φαίνονται στα ακόλουθα σχήματα.



Σχήμα 1: Cycles



Σχήμα 2: L1 Miss Rate



Σχήμα 3: L2 Miss Rate

Cycles	L1 Miss rate	L2 Miss rate	Speedup
1523400216	0.0285	0.0150	1.0
1519770751	0.0046	0.0152	1.00
1145990549	0.0303	0.0281	1.32
2752007165	0.0835	0.0675	0.55
1905041698	0.0045	0.0115	0.79
2659818958	0.0833	0.0619	0.57

Πίνακας 4: Μετρικές όλων των εκτελέσεων

Ανάλογα με τη διάταξη των loops οι πίνακες διασχίζονται κατά γραμμές και κατά στήλες σε διάφορους συνδυασμούς. Μεγαλύτερο speedup παρατηρούμε στη διάταξη j,i,k.

3.2 Cache Blocking

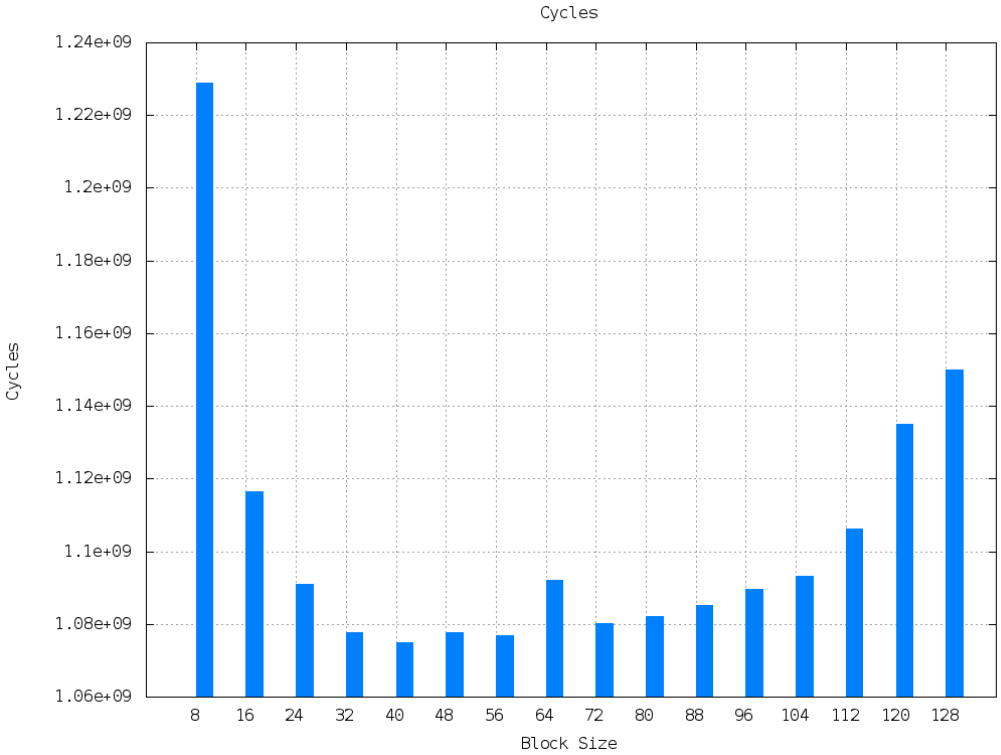
Η L1 cache έχει μέγεθος 32KB, και line size 64 bytes. Κάνουμε πράξεις μεταξύ αριθμών κινητής υποδιαστολής σε 32bit σύστημα συνεπώς κάθε ένας έχει μέγεθος 4bytes. Κάθε cache line χωράει 16 αριθμούς συνεπώς για να εκμεταλλευτούμε καλύτερα την τοπικότητα των αναφορών θα εκτελούμε κάθε loop σε ομάδες. Έτσι θα χρησιμοποιούμε δεδομένα που έχουν ήδη έρθει στην cache.

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : partA.c
 * Creation Date : 16-07-2012
 * Last Modified : Fri 20 Jul 2012 05:37:36 PM EEST
 * Created By : Greg Liras <gregliras@gmail.com>
 _.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-.*/

#include <stdio.h>
#include <stdlib.h>
#define __MAGIC_CASSERT(p) do {                         \
    typedef int __check_magic_argument[(p) ? 1 : -1];   \
} while (0)

#define MAGIC(n) do {                       \
    __MAGIC_CASSERT(!(n));                  \
    __asm__ __volatile__ ("xchg %bx,%bx");  \
} while (0)

#define MAGIC_BREAKPOINT MAGIC(0)



inline int min(int a, int b)
{
    if(a<=b)return a;
    else return b;
}
void init_matrix(float **mat, int n)
{
    unsigned int i,j;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            mat[i][j] = (float)(i+j);
}
int main(int argc, char **argv)
{
    float **A,**B,**C;
    int i,j,k,N;
    int starti,stopi;
    int startj,stopj;
    int startk,stopk;
    int BS = atoi(argv[2]);
    N=atoi(argv[1]);
    A=(float**)malloc(N*sizeof(float*));

    for(i=0; i<N; i++)
        A[i]=(float*)malloc(N*sizeof(float));

    B=(float**)malloc(N*sizeof(float*));

    for(i=0; i<N; i++)
        B[i]=(float*)malloc(N*sizeof(float));

    C=(float**)malloc(N*sizeof(float*));

    for(i=0; i<N; i++)
        C[i]=(float*)malloc(N*sizeof(float));

    fprintf(stderr, "Initializing matrices...\n");
    init_matrix(A, N);
    init_matrix(B, N);
    init_matrix(C, N);
    MAGIC_BREAKPOINT;
    for(startj=0; startj<N; startj+=BS) {
        stopj = startj + BS;
```

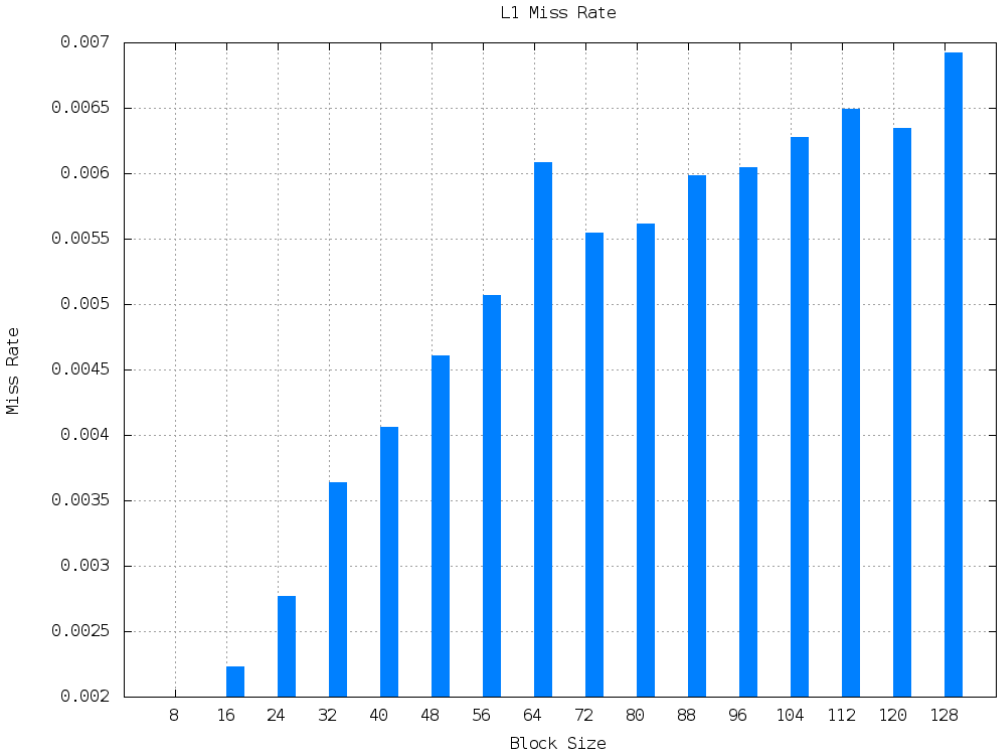
```
        stopj = stopj <= N ? stopj : N;
        for(starti=0; starti<N; starti+=BS) {
            stopi = starti + BS;
            stopi = stopi <= N ? stopi : N;
            for(startk=0; startk<N; startk+=BS) {
                stopk = startk + BS;
                stopk = stopk <= N ? stopk : N;
                for(j=start; j<stop; j++)
                    for(i=start; i<stop; i++)
                        for(k=start; k<stop; k++)
                            C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
    MAGIC_BREAKPOINT;
    return 0;
}
```

Cycles	L1 Miss rate	L2 Miss rate	Speedup
1229008147	0.00200	0.00436	1.0
1116424502	0.00222	0.00355	1.1008
1090971021	0.00276	0.00363	1.1265
1077600084	0.00364	0.00337	1.1405
1074879790	0.00406	0.00347	1.1433
1077724197	0.00460	0.00443	1.1403
1076847331	0.00506	0.00464	1.1413
1092092535	0.00608	0.00734	1.1253
1080181105	0.00554	0.00586	1.1377
1082269217	0.00561	0.00616	1.1355
1085119465	0.00598	0.00724	1.1326
1089671800	0.00604	0.00797	1.1278
1093319686	0.00628	0.00837	1.1241
1106283098	0.00649	0.01040	1.1109
1134938000	0.00634	0.01550	1.0828
1150045093	0.00692	0.01869	1.0686

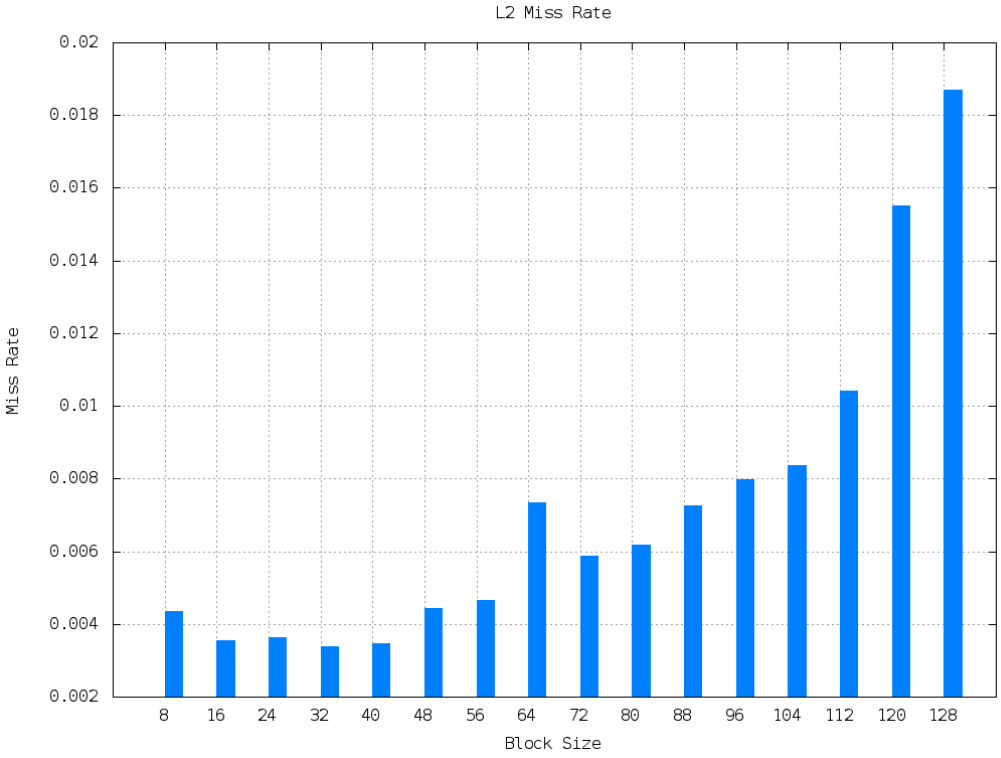
Πίνακας 5: Διαφορά των εκτελέσεων



Σχήμα 4: Cycles



Σχήμα 5: L1 Miss Rate



Σχήμα 6: L2 Miss Rate

Η απλοική εκτέλεση διαρκεί 1523400216 cycles. Η βέλτιστη cache blocked 1074879790. Το speedup που υπολογίζουμε από αυτή τη διαφορά είναι 1.41.

Παρατηρούμε πως η μεταβολή του block size στο πρόγραμμά μας, αν και αυξάνει το πλήθος των εντολών του προγράμματος, αυξάνει την ταχύτητά του καθώς εκμεταλλεύεται καλύτερο την τοπικότητα των αναφορών. Ωστόσο για ακραίες τιμές η επιτάχυνση δεν είναι πολύ σημαντική. Είναι μια τεχνική που μπορεί να εφαρμοστεί ταυτόχρονα με την αναδιάρθρωση των βρόχων αρκεί το pattern να μην είναι τυχαίο αλλά η πρόσβαση στα δεδομένα να μπορεί να μετασχηματιστεί σε αντίστοιχη μορφή.

Μέρος Β΄

1 Πρωτόκολλο MESI

Έχουμε ένα πολυεπεξεργαστικό σύστημα δύο επεξεργαστών που χρησιμοποιεί το πρωτόκολλο MESI. Κάθε επεξεργαστής έχει cache με τα ακόλουθα χαρακτηριστικά.

Cache Size :	4KB
Associativity :	2-way set write-back
Set Size :	2KB
Block Size :	16 bytes $\rightarrow$ 4 bits block offset
Blocks/way :	2KB / way $\rightarrow 2^{11}/2^4 = 2^7 = 128$ blocks / way
Tag :	5 bits
Policy :	LRU
Address Size :	16 bytes

Πίνακας 6: Cache characteristics

Συνεπώς μπορούμε να βρούμε την αντιστοιχία των διευθύνσεων στην cache.

Addresses	Tag	Index	Block Offset
073C	00000	1110011	1100
0734	00000	1110011	0100
0738	00000	1110011	1000
0730	00000	1110011	0000
1F34	00011	1110011	0100
1F3C	00011	1110011	1100
2730	00100	1110011	0000
273C	00100	1110011	1100

Πίνακας 7: Memory $\mapsto$ Cache

	P0	P1
1	read 073C	
2		read 0734
3		write‘1111’ -> 0734
4	read 0738	
5	write‘2222’ -> 0730	
6		write‘3333’-> 1F34
7	write‘4444’ -> 1F3C	
8	read 073C	
9		write‘5555’-> 2730
10	read 273C	
11		write‘6666’-> 273C

Πίνακας 8: Σειρά προσπελάσεων στη μνήμη

Πίνακας 9: P0: read 073C

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	0000	V
1F3X	0000	0000	0000	0000	V
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	E	0000	0000	0000	0000
Processor 1							

Πίνακας 10: P1: read 0734

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	0000	V
1F3X	0000	0000	0000	0000	V
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	S	0000	0000	0000	0000
Processor 1	0	115/0	S	0000	0000	0000	0000

Πίνακας 11: P1: write ’1111’ to 0734

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	0000	I
1F3X	0000	0000	0000	0000	V
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	I	-	-	-	-
Processor 1	0	115/0	M	0000	0000	0000	1111

Πίνακας 12: P0: read 0738

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	V
1F3X	0000	0000	0000	0000	V
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	S	0000	0000	0000	1111
Processor 1	0	115/0	S	0000	0000	0000	1111

Πίνακας 13: P0: write ’2222’ to 0730

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	0000	0000	0000	V
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
Processor 1	0	115/0	I	-	-	-	-

Πίνακας 14: P1: write ’3333’ to 1F34

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	0000	0000	0000	I
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
Processor 1	0	115/0	I	-	-	-	-
	3	115/1	M	0000	3333	0000	1111

Πίνακας 15: P0: write ’4444’ to 1F3C

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	3333	0000	0000	I
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
	3	115/1	M	0000	3333	0000	4444
Processor 1	0	115/0	I	-	-	-	-
	3	115/1	I	-	-	-	-

Πίνακας 16: P0: read 073C

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	3333	0000	0000	I
273X	0000	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
	3	115/1	M	0000	3333	0000	4444
Processor 1	0	115/0	I	-	-	-	-
	3	115/1	I	-	-	-	-

Πίνακας 17: P1: write ’5555’ to 2730

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	3333	0000	0000	I
273X	0000	0000	0000	0000	I

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
	3	115/1	M	0000	3333	0000	4444
Processor 1	4	115/0	M	5555	0000	0000	0000
	3	115/1	I	-	-	-	-

Πίνακας 18: P0: read 273C

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	3333	0000	4444	V
273X	5555	0000	0000	0000	V

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
	4	115/1	S	5555	0000	0000	0000
Processor 1	4	115/0	S	5555	0000	0000	0000
	3	115/1	I	-	-	-	-

Πίνακας 19: P0: write ’6666’ to 273C

Memory					
Address	0	4	8	C	Valid
073X	0000	0000	0000	1111	I
1F3X	0000	3333	0000	4444	V
273X	5555	0000	0000	0000	I

Processor	Tag	Set/line	MESI	Data			
Processor 0	0	115/0	M	2222	0000	0000	1111
	4	115/1	M	5555	0000	0000	6666
Processor 1	4	115/0	I	-	-	-	-
	3	115/1	I	-	-	-	-

2 Memory Consistency

Οι δυνατοί συνδιασμοί των τελικών τιμών φαίνονται στους παρακάτω πίνακες.

P0	P1	P0	P1	P0	P1	P0	P1
flag=1		Y=1		X=2		X=2	
	r3=0	flag=1		flag=1		Y=1	
	r4=0		r3=0		r3=2	flag=1	
	flag=0		r4=1		r4=0		r3=2
X=2			flag=0		flag=0		r4=1
Y=1		X=2		Y=1			flag=0
r1=2		r1=2		r1=2		r1=2	
r2=0		r2=0		r2=0		r2=0	
	Z=4		Z=4		Z=4		Z=4
(r1,r2,r3,r4)=(2,0,0,0)		(r1,r2,r3,r4)=(2,0,0,1)		(r1,r2,r3,r4)=(2,0,2,0)		(r1,r2,r3,r4)=(2,0,2,1)	

P0	P1	P0	P1	P0	P1	P0	P1
flag=1		Y=1		X=2		X=2	
	r3=0	flag=1		flag=1		Y=1	
	r4=0		r3=0		r3=2	flag=1	
	Z=4		r4=1		r4=0		r3=2
	flag=0		Z=4		Z=4		r4=1
X=2			flag=0		flag=0		Z=4
Y=1		X=2		Y=1			flag=0
r1=2		r1=2		r1=2		r1=2	
r2=4		r2=4		r2=4		r2=4	
(r1,r2,r3,r4)=(2,4,0,0)		(r1,r2,r3,r4)=(2,4,0,1)		(r1,r2,r3,r4)=(2,4,2,0)		(r1,r2,r3,r4)=(2,4,2,1)	

Ο μοναδικός συνδιασμός για να μπορέσει ο Processor 0 να βγει από το inf loop, είναι ο Processor 1 να εκτελέσει την flag=0.

P0	P1
X=2	
Y=1	
flag=1	
	r3=2
	r4=1
	Z=4
	flag=0
r1=2	
r2=4	
(r1,r2,r3, r4)=(2,4,2,1)	

Για να επιτύχουμε το ίδιο αποτέλεσμα με weak ordering model πρέπει οι εντολές X=2,Y=1 να πραγματοποιηθούν πριν τις r3 = X, r4 = Y. Συνεπώς θα πρέπει να έχουμε δύο SYNCH.

P0	P1
X=2 Y=1 SYNCH flag=1 while(flag==1) r1=X r2=Z	while(flag==0); r3=X r4=Y Z=4 SYNCH flag=0