



# ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ

Προηγμένα Θέματα  
Αρχιτεκτονικής Υπολογιστών

4<sup>η</sup> Άσκηση  
Ακ. έτος 2011-2012

Γρηγόρης Λύρας Α.Μ.: 03109687

21 Ιουλίου 2012

# Εισαγωγή

## I) Προσομοίωση

Ο κώδικας που μας δόθηκε ήταν ο ακόλουθος:

```
1  /* .....  
2  * File Name : partA.c  
3  * Creation Date : 16-07-2012  
4  * Last Modified : Mon 16 Jul 2012 01:46:39 PM EEST  
5  * Created By : Greg Liras <gregliras@gmail.com>  
6  * .....*/  
7  
8  #include <stdio.h>  
9  #include <stdlib.h>  
10 #define __MAGIC_CASSERT(p) do { \   
11     typedef int __check_magic_argument[(p) ? 1 : -1]; \   
12 } while (0)  
13  
14 #define MAGIC(n) do { \   
15     __MAGIC_CASSERT(!(n)); \   
16     __asm__ __volatile__ ("xchg %bx,%bx"); \   
17 } while (0)  
18  
19 #define MAGIC_BREAKPOINT MAGIC(0)  
20  
21  
22  
23 inline int min(int a, int b)  
24 {  
25     if(a<=b) return a;  
26     else return b;  
27 }  
28 void init_matrix(float **mat, int n)  
29 {  
30     unsigned int i,j;  
31     for(i=0; i<n; i++)  
32         for(j=0; j<n; j++)  
33             mat[i][j] = (float)(i+j);  
34 }  
35 int main(int argc, char **argv)  
36 {  
37     float **A,**B,**C;  
38     int i,j,k,N;  
39     N=atoi(argv[1]);  
40     A=(float**)malloc(N*sizeof(float*));  
41  
42     for(i=0; i<N; i++)  
43         A[i]=(float*)malloc(N*sizeof(float));  
44  
45     B=(float**)malloc(N*sizeof(float*));  
46  
47     for(i=0; i<N; i++)  
48         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 }
```

## II) 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:

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

|               |            |
|---------------|------------|
| Cycles        | 1523400216 |
| L1 miss ratio | 0.029      |
| L2 miss ratio | 0.015      |

Πίνακας 2: Μετρικές πρώτης εκτέλεσης

## Τεχνικές Βελτιστοποίησης

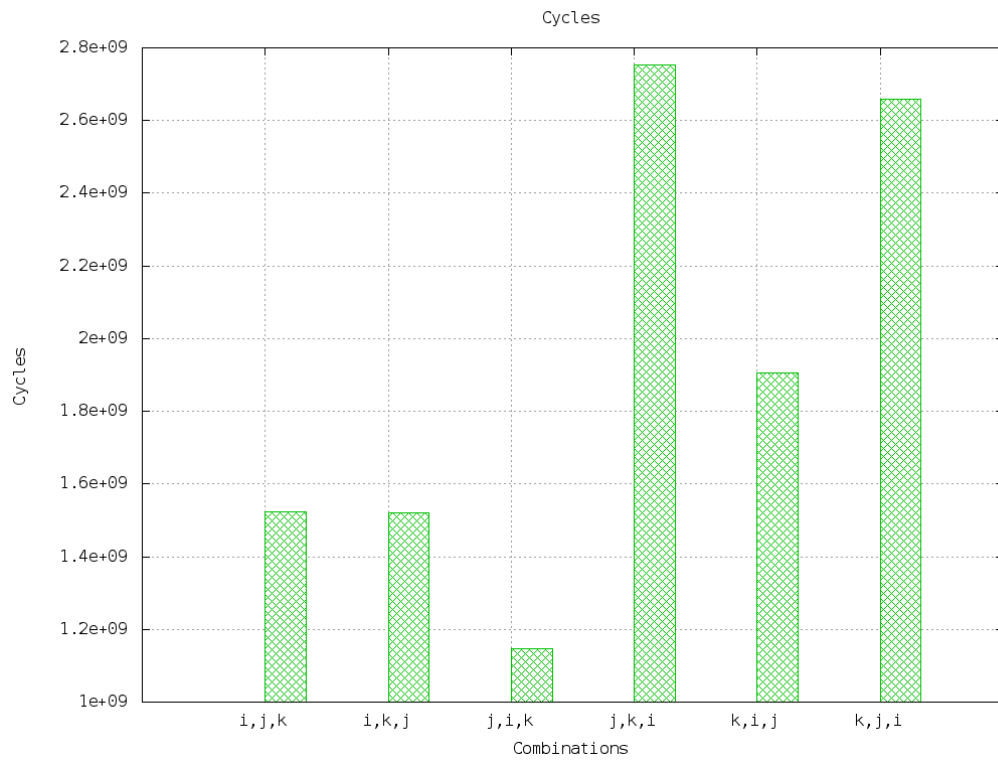
### III) Loop Interchange

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

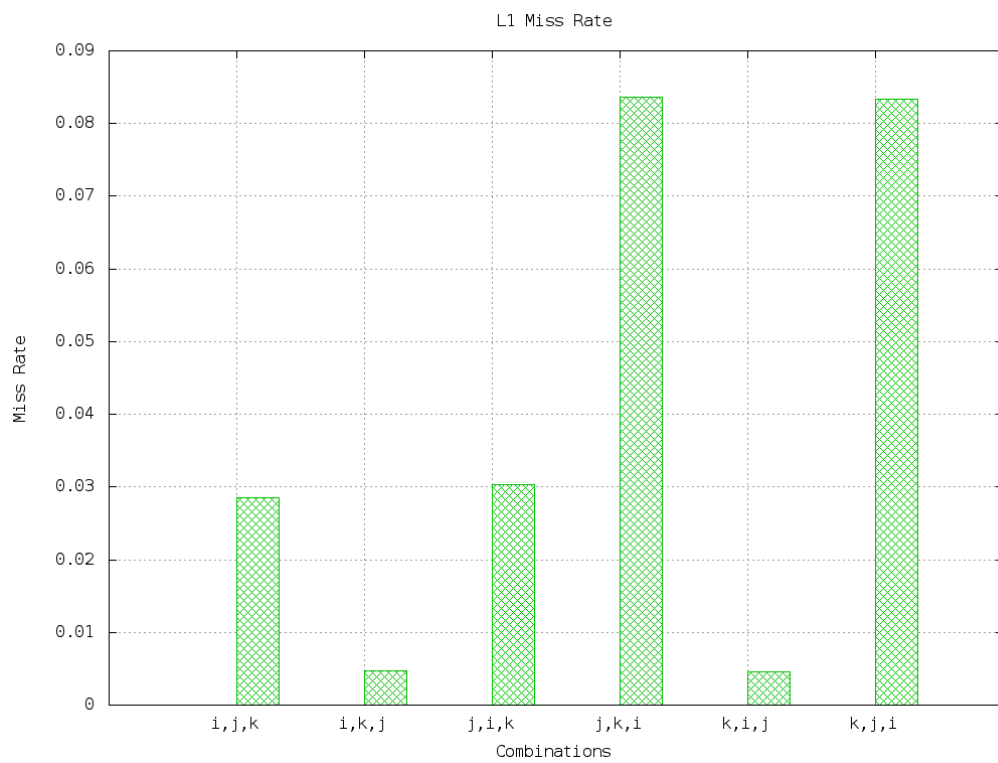
| Σειρά προσομοίωσης | Διάταξη |
|--------------------|---------|
| 1                  | i,j,k   |
| 2                  | i,k,j   |
| 3                  | j,i,k   |
| 4                  | j,k,i   |
| 5                  | k,i,j   |
| 6                  | k,j,i   |

Πίνακας 3: Δυνατές Διατάξεις

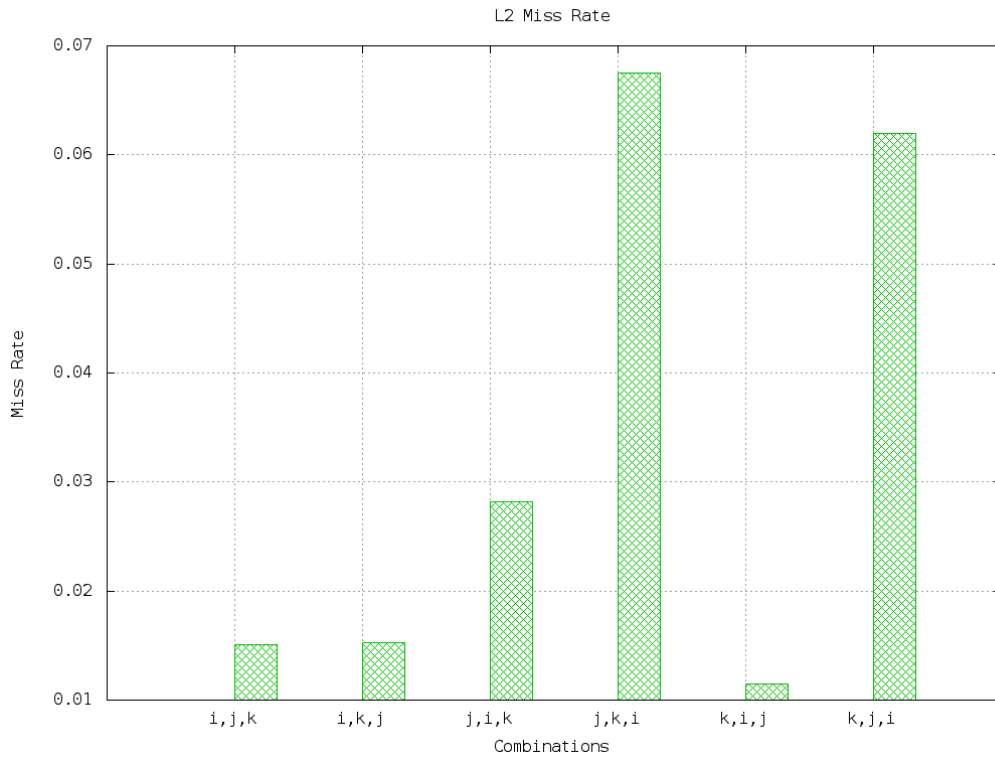
Τα αποτελέσματα φαίνονται στα ακόλουθα σχήματα.



Σχήμα 1: Cycles



Σχήμα 2: L1 Miss Rate



Σχήμα 3: L2 Miss Rate

| Cycles     | L1 Miss rate     | L2 Miss rate    | Speedup        |
|------------|------------------|-----------------|----------------|
| 1523400216 | 0.0285795820412  | 0.0150389941603 | 1.0            |
| 1519770751 | 0.00468022311872 | 0.0152373522682 | 1.00238816611  |
| 1145990549 | 0.0303714959475  | 0.0281574903722 | 1.32933052313  |
| 2752007165 | 0.0835636317199  | 0.06750286478   | 0.553559683774 |
| 1905041698 | 0.00455744098555 | 0.0115127159615 | 0.79966764906  |
| 2659818958 | 0.0833196681295  | 0.0619748730144 | 0.572745829718 |

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.

## IV) Cache Blocking

H 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.

```
1  /* .....
2  * File Name : partA.c
3  * Creation Date : 16-07-2012
4  * Last Modified : Fri 20 Jul 2012 05:37:36 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  .....*/
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #define __MAGIC_CASSERT(p) do { \
11     typedef int __check_magic_argument[(p) ? 1 : -1]; \
12 } while (0)
13
14 #define MAGIC(n) do { \
15     __MAGIC_CASSERT(!(n)); \
16     __asm__ __volatile__ ("xchg %bx,%bx"); \
17 } while (0)
18
19 #define MAGIC_BREAKPOINT MAGIC(0)
20
21
22
23 inline int min(int a, int b)
24 {
25     if(a<=b) return a;
26     else return b;
27 }
28 void init_matrix(float **mat, int n)
29 {
30     unsigned int i,j;
31     for(i=0; i<n; i++)
32         for(j=0; j<n; j++)
33             mat[i][j] = (float)(i+j);
34 }
35 int main(int argc, char **argv)
36 {
37     float **A,**B,**C;
38     int i,j,k,N;
39     int starti,stopi;
40     int startj,stopj;
41     int startk,stopk;
42     int BS = atoi(argv[2]);
43     N=atoi(argv[1]);
44     A=(float**)malloc(N*sizeof(float*));
45
46     for(i=0; i<N; i++)
47         A[i]=(float*)malloc(N*sizeof(float));
48
49     B=(float**)malloc(N*sizeof(float*));
50
51     for(i=0; i<N; i++)
52         B[i]=(float*)malloc(N*sizeof(float));
53
54     C=(float**)malloc(N*sizeof(float*));
55
56     for(i=0; i<N; i++)
57         C[i]=(float*)malloc(N*sizeof(float));
58
59     fprintf(stderr, "Initializing matrices...\n");
60     init_matrix(A, N);
61     init_matrix(B, N);
62     init_matrix(C, N);
63     MAGIC_BREAKPOINT;
64     for(startj=0; startj<N; startj+=BS) {
65         stopj = startj + BS;
```

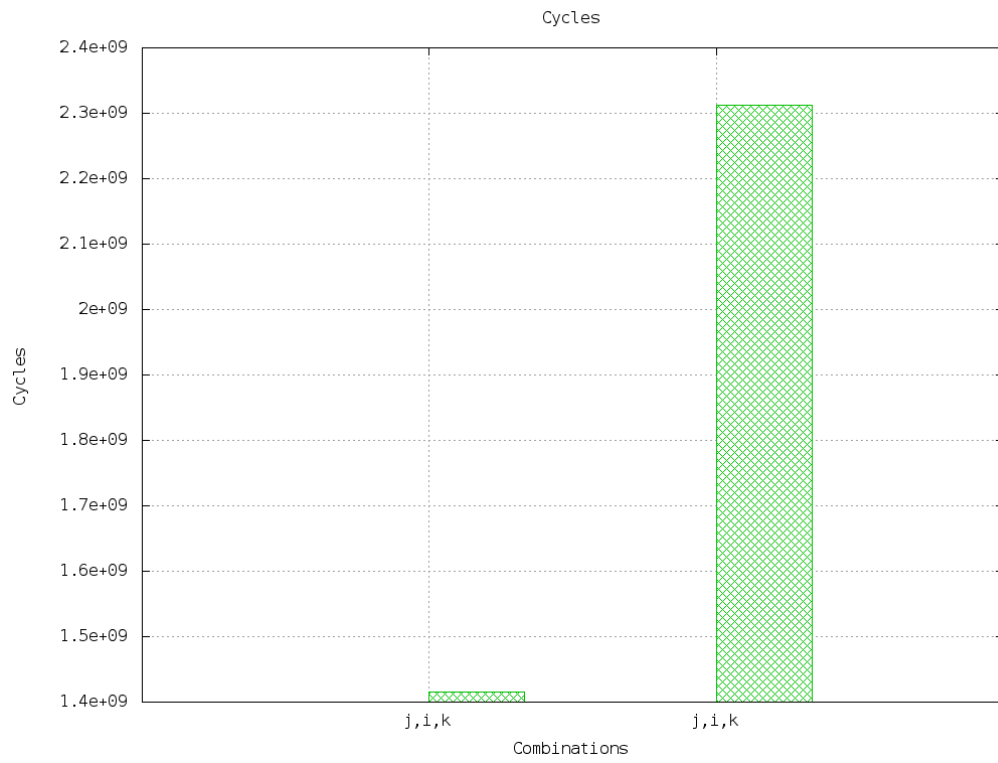
```

66     stopj = stopj <= N ? stopj : N;
67     for(starti=0; starti<N; starti+=BS) {
68         stopi = starti + BS;
69         stopi = stopi <= N ? stopi : N;
70         for(startk=0; startk<N; startk+=BS) {
71             stopk = startk + BS;
72             stopk = stopk <= N ? stopk : N;
73             for(j=start; j<stop; j++)
74                 for(i=start; i<stop; i++)
75                     for(k=start; k<stop; k++)
76                         C[i][j] += A[i][k]*B[k][j];
77         }
78     }
79 }
80 MAGIC_BREAKPOINT;
81 return 0;
82 }

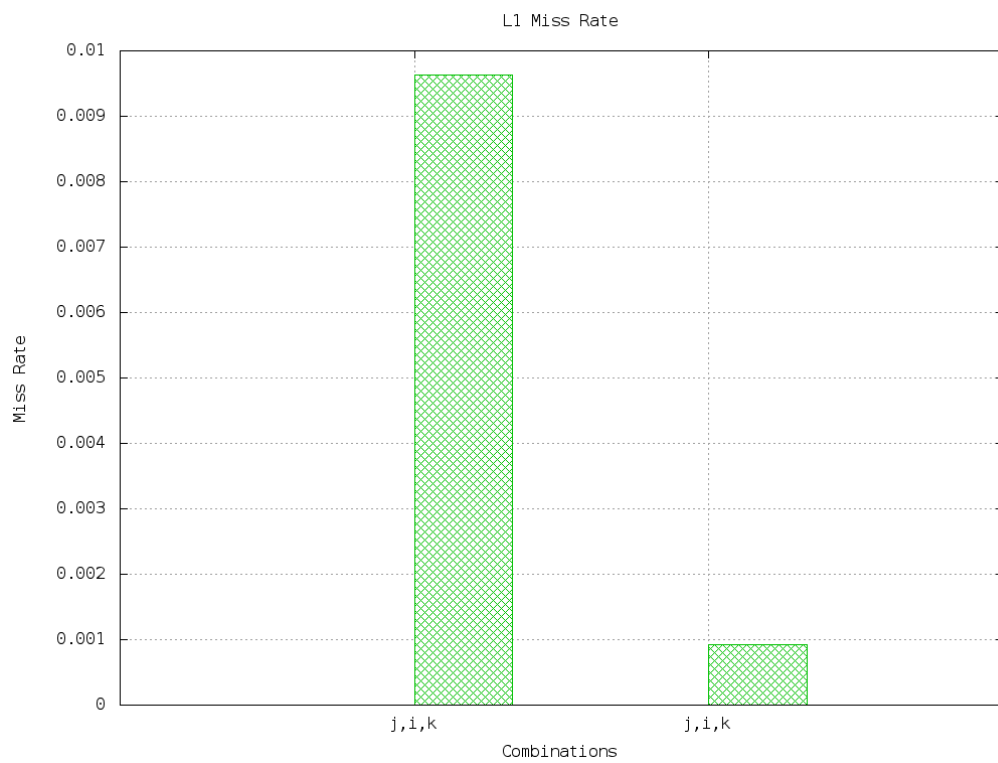
```

| Cycles     | L1 Miss rate     | L2 Miss rate     | Speedup       |
|------------|------------------|------------------|---------------|
| 1229008147 | 0.00200081611811 | 0.00436512977739 | 1.0           |
| 1116424502 | 0.00222865322992 | 0.00355591913984 | 1.10084304384 |
| 1090971021 | 0.00276906476251 | 0.00363915609977 | 1.12652684933 |
| 1077600084 | 0.00364079131771 | 0.00337975797327 | 1.14050487305 |
| 1074879790 | 0.00406211929743 | 0.00347469774194 | 1.14339125029 |
| 1077724197 | 0.00460946323771 | 0.00443471065927 | 1.14037353009 |
| 1076847331 | 0.0050673609159  | 0.00464953114037 | 1.14130212484 |
| 1092092535 | 0.00608081927452 | 0.00734009216841 | 1.12536997334 |
| 1080181105 | 0.00554504027806 | 0.00586511858615 | 1.13777971241 |
| 1082269217 | 0.00561610467505 | 0.00616771173416 | 1.1355844994  |
| 1085119465 | 0.00598425140451 | 0.00724845710677 | 1.13260169653 |
| 1089671800 | 0.00604447720433 | 0.00797237050637 | 1.12787001279 |
| 1093319686 | 0.0062805931738  | 0.00837639698304 | 1.12410684884 |
| 1106283098 | 0.00649260144475 | 0.0104062454345  | 1.11093457834 |
| 1134938000 | 0.00634545344483 | 0.0155069937752  | 1.08288571446 |
| 1150045093 | 0.00692636946452 | 0.0186948544829  | 1.06866083294 |

Πίνακας 5: Διαφορά των εκτελέσεων

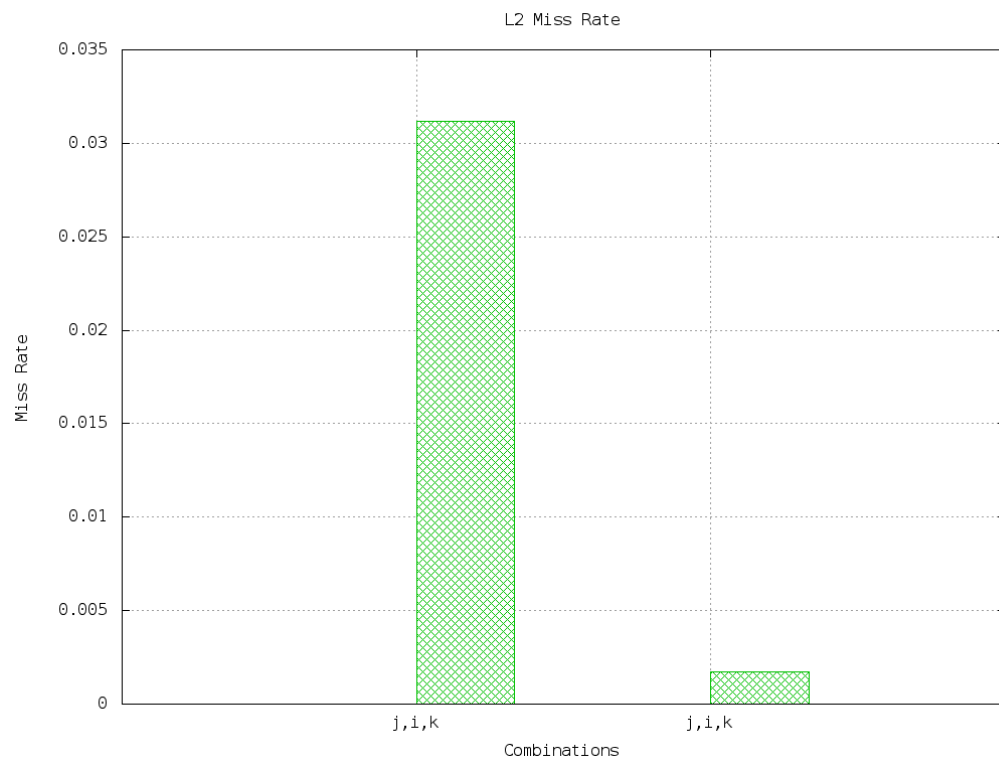


Σχήμα 4: Cycles



Σχήμα 5: L1 Miss Rate





Σχήμα 6: L3 Miss Rate

## Cache Consistency