ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ

Προηγμένα Θέματα
Αρχιτεκτονικής Υπολογιστών

4η Άσκηση
Ακ. έτος 2011-2012

Γρηγόρης Λύρας Α.Μ.: 03109687

21 Ιουλίου 2012

Εισαγωγή

I) Προσομοίωση

Ο κώδικας που μας δόθηκε ήταν ο ακόλουθος:

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * 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));

    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(i=0; i<N; i++) {
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                C[i][j] += A[i][k]*B[k][j];
    }
    MAGIC_BREAKPOINT;
    return 0;
}
```

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

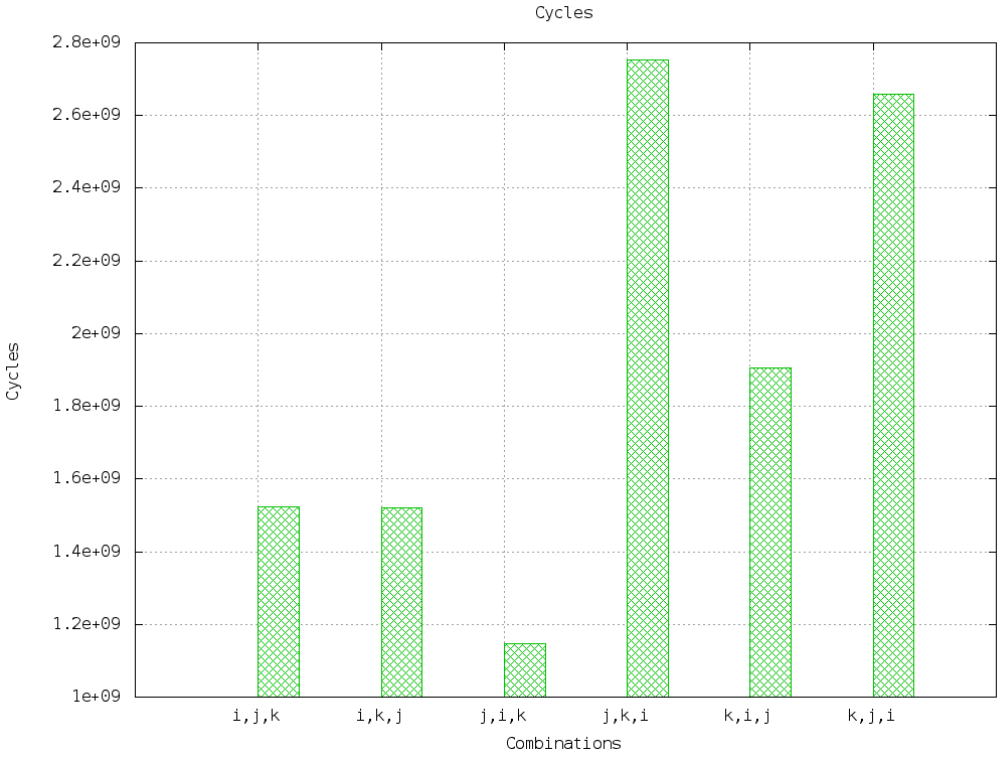
III) Loop Interchange

Για την προσομοίωση κάναμε 6 εκτελέσεις για κάθε διάταξη των i,j,k όπως φαίνεται στον πίνακα 4.

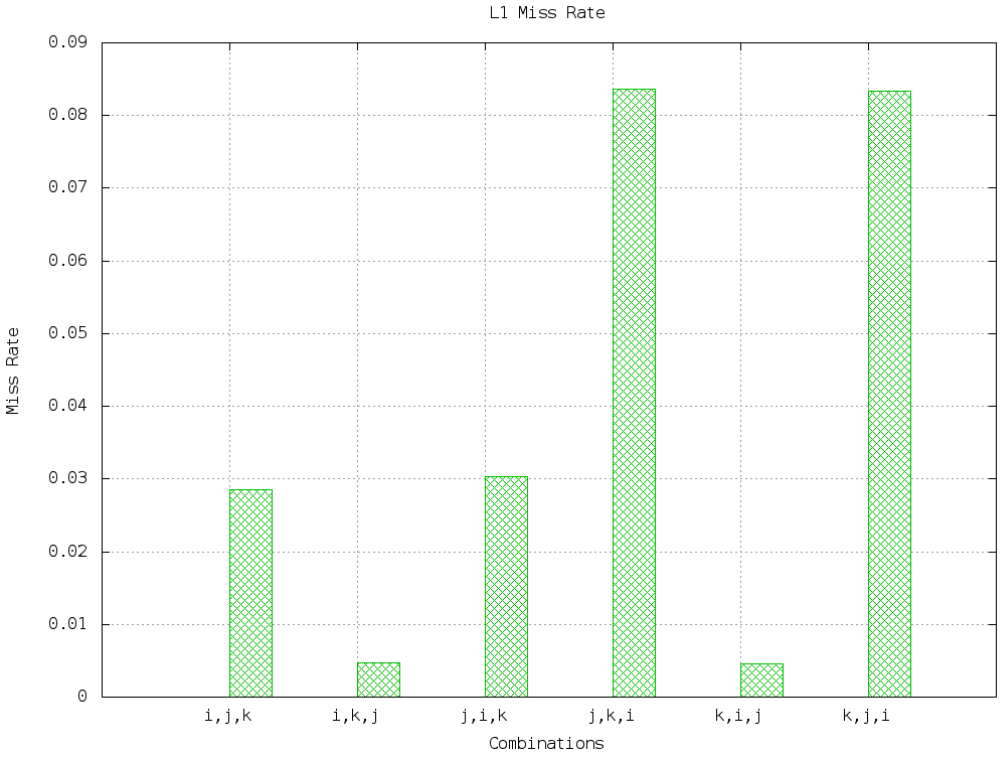
Σειρά προσομοίωσης	Διάταξη
1	i,j,k
2	i,k,j
3	j,i,k
4	j,k,i
5	k,i,j
6	k,j,i

Πίνακας 3: Δυνατές Διατάξεις

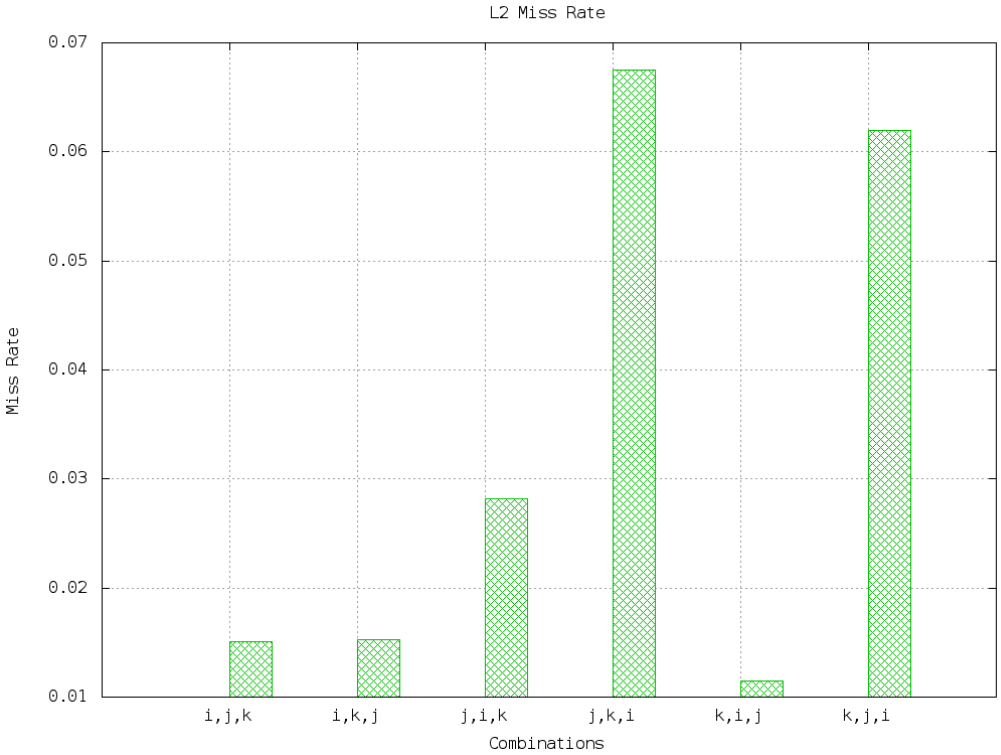
Τα αποτελέσματα φαίνονται στα ακόλουθα σχήματα.



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

IV) Cache Blocking

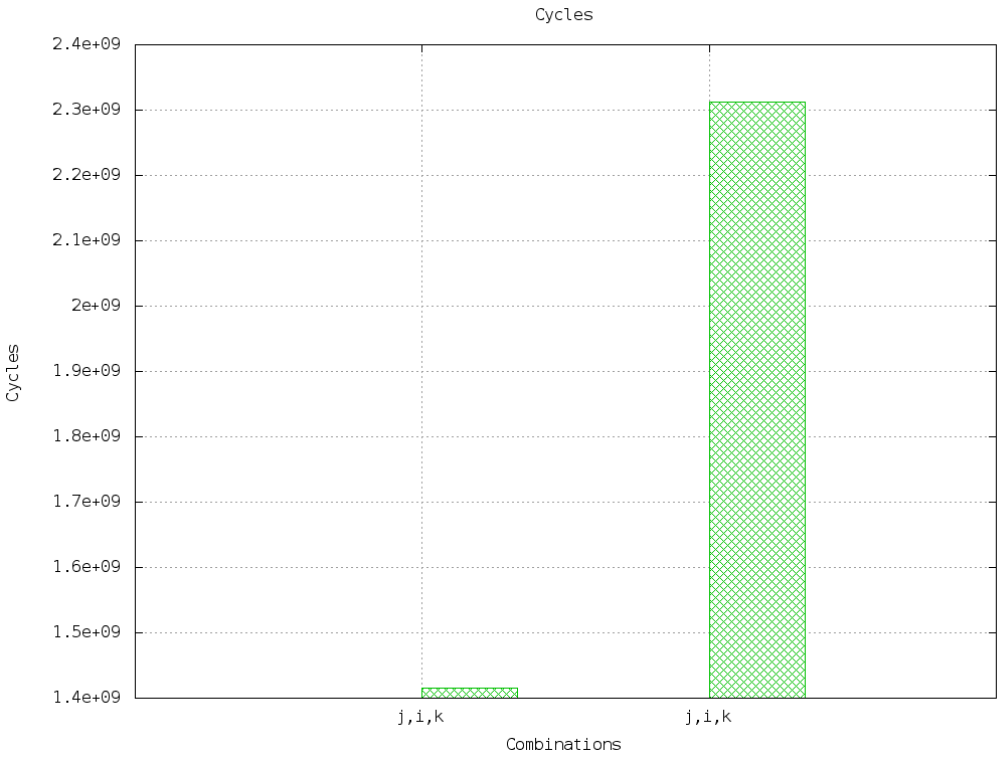
Η L1 cache έχει μέγεθος 32KB, και line size 64 bytes. Κάνουμε πράξεις μεταξύ αριθμών κινητής υποδιαστολής σε 32bit σύστημα συνεπώς κάθε ένας έχει μέγεθος 4bytes. Κάθε cache line χωράει 16 αριθμούς συνεπώς για να εκμεταλλευτούμε καλύτερα την τοπικότητα των αναφορών θα εκτελούμε κάθε loop σε 16δες. Έτσι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

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : partA.c
 * Creation Date : 16-07-2012
 * Last Modified : Fri 20 Jul 2012 01:19:53 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;
    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+=16) {
        stopj = startj + 16;
        stopj = stopj <= N ? stopj : N;
```

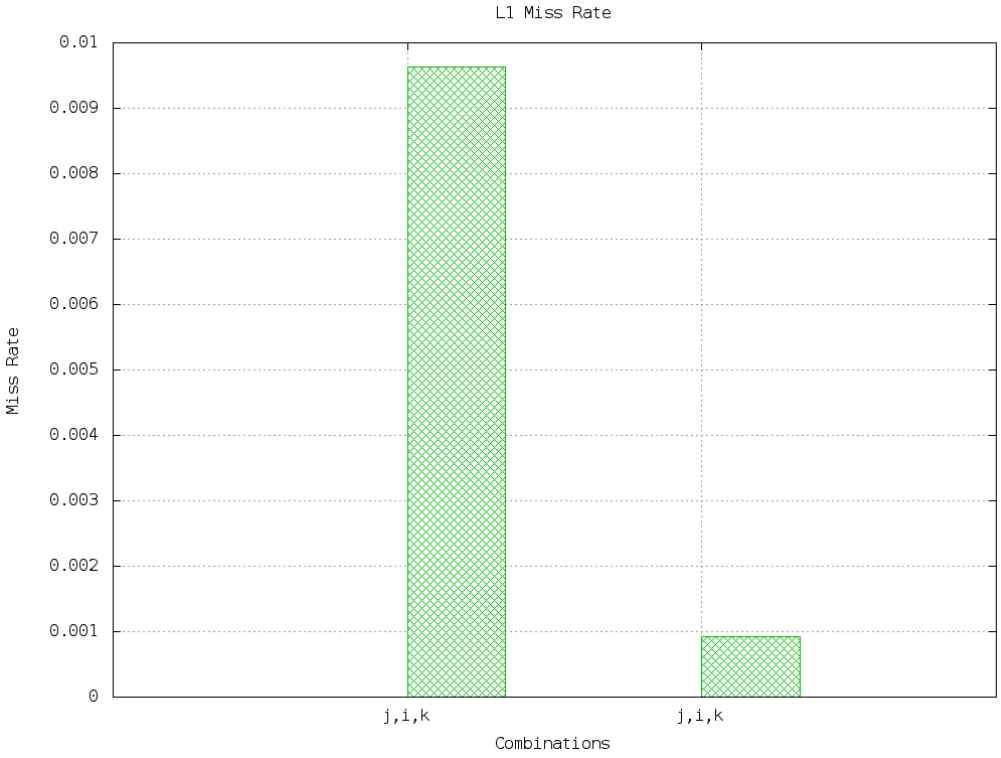
```
66          for(starti=0; starti<N; starti+=16) {
67              stopi = starti + 16;
68              stopi = stopi <= N ? stopi : N;
69              for(startk=0; startk<N; startk+=16) {
70                  stopk = startk + 16;
71                  stopk = stopk <= N ? stopk : N;
72                  for(j=start; j<stop; j++)
73                      for(i=start; i<stop; i++)
74                          for(k=start; k<stop; k++)
75                              C[i][j] += A[i][k]*B[k][j];
76              }
77          }
78      }
79      MAGIC_BREAKPOINT;
80      return 0;
81  }
```

Cycles Cycles	1448776923 9385880
L1 miss ratio L1 miss ratio	0.0131 0.0026
L2 miss ratio L2 miss ratio	0.0292 0.0065

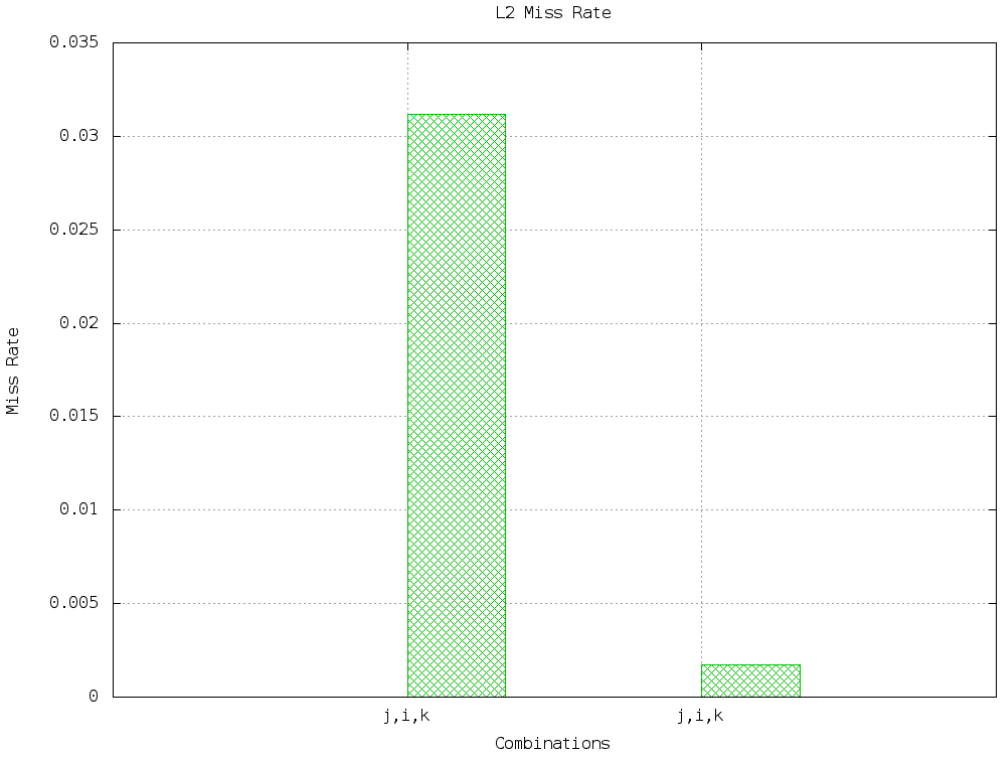
Πίνακας 5: Διαφορά των εκτελέσεων



Σχήμα 4: Cycles



Σχήμα 5: L1 Miss Rate



Σχήμα 6: L3 Miss Rate

Cache Consistency