ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

www.cslab.ece.ntua.gr

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Εξετάσεις Ιουνίου 2011

Διάρκεια 2:45' ώρες

Οι εξετάσεις θα πραγματοποιηθούν ΧΩΡΙΣ την παρουσία βιβλίων, βοηθημάτων ή άλλου είδους σημειώσεων. Το μόνο που επιτρέπεται να έχετε μαζί σας είναι ένα φύλλο Α4 στο οποίο μπορείτε να έχετε γράψει ό,τι έχετε κρίνει πιο σημαντικό για το μάθημα και θέλετε να το έχετε ως βοήθημά σας. Απαγορεύεται η ανταλλαγή οποιουδήποτε αντικειμένου κατά την ώρα της εξέτασης, ούτε και των φύλλων Α4 που είναι ατομικά.

Θέμα 1ο (15%)

Α. Ποια τεχνική χρησιμοποιεί ο αλγόριθμος Tomasulo για να αντιμετωπίσει τους WAR και WAW hazards; Γιατί δεν χρησιμοποιεί την ίδια τεχνική για να αντιμετωπίσει και τους RAW hazards;

Ο Tomasulo χρησιμοποιεί register renaming για να αντιμετωπίσει τους WAR, WAW hazards. Η τεχνική αυτή αλλάζει τον destination register, και έτσι οι 2 εντολές που προκαλούν το hazard δεν χρειάζεται πλέον να προσπελάσουν τον ίδιο καταχωρητή. Η αλλαγή αυτή όμως δεν επηρεάζει τις RAW εξαρτήσεις, μιας και όπως και να μετονομάσουμε τον destination καταχωρητή, η εντολή που διαβάζει θα εξακολουθήσει να χρειάζεται την τιμή που θα γράψει η προηγούμενη εντολή. Επομένως το register renaming δεν επιλύει τους RAW hazards.

Β. Ποια η διαφορά μεταξύ Reservation Stations και Reorder Buffer;

Τα RS αποθηκεύουν μια εντολή και τα ορίσματα της μέχρι να μπορέσει η εντολή αυτή να εκτελεστεί. Αντίθετα, ο ROB αποθηκεύει την κατάσταση της εντολής (και μετά την εκτέλεση και το αποτέλεσμά της) μέχρι η εντολή να κάνει commit.

Γ. Τι είναι ο Branch Target Buffer (BTB); Πώς βελτιώνει η χρήση του την απόδοση ενός επεξεργαστή; Γιατί δε θα δούλευε αποδοτικά για την πρόβλεψη των διευθύνσεων επιστροφής από ρουτίνες/συναρτήσεις;

Ο BTB αποθηκεύει το target address προηγούμενων αλμάτων που ήταν Taken. Η απόδοση του επεξεργαστή βελτιώνεται, καθώς δε χρειάζεται να περιμένει τον υπολογισμό της διεύθυνσης του target και μπορεί να προχωρήσει κατευθείαν στη φόρτωση της επόμενης εντολής όταν το αποτέλεσμα του branch προβλεφθεί (ή γίνει γνωστό). Για αυτό το λόγο δε θα δούλευε αποδοτικά για την πρόβλεψη των διευθύνσεων επιστροφής από ρουτίνες, μιας και η διεύθυνση επιστροφής δεν είναι πάντα ίδια και εξαρτάται από το σημείο του κώδικα (της εκτέλεσης) στο οποίο κλήθηκε η συγκεκριμένη ρουτίνα/συνάρτηση.

Δ. Θεωρήστε ένα loop το οποίο εκτελείται αρκετές φορές σε ένα πρόγραμμα. Κάθε εκτέλεση του loop, περιλαμβάνει 10 επαναλήψεις. Κάθε επανάληψη περιλαμβάνει 5 εντολές άλματος υπό συνθήκη, τα αποτελέσματα των οποίων φαίνονται στον παρακάτω πίνακα.

	Επανάληψη									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Branch 1	T	N	N	N	N	N	N	N	N	N
Branch 2	T	T	T	N	N	N	T	T	T	T
Branch 3	T	N	T	N	T	N	T	N	T	N
Branch 4	T	T	N	N	N	T	T	T	T	N
Branch 5	T	T	T	T	T	T	T	T	T	N

Στην $10^{η}$ επανάληψη το Branch 5 είναι NT και η εκτέλεση φεύγει από το loop. Υποθέστε ότι δεν υπάρχουν άλλες εντολές άλματος υπό συνθήκη στον κώδικα, καθώς και ότι δεν εμφανίζονται φαινόμενα aliasing μιας και το κάθε branch έχει τον δικό του predictor. Από τους dynamic predictors που εξετάσαμε στο μάθημα διαλέξτε αυτόν που θα δώσει το καλύτερο ποσοστό πρόβλεψης για τις παρακάτω εντολές:
i) Branch 1

Το Branch1 εκτελείται πάντα μετά το Branch5 και είναι πάντα το αντίθετο του. Έτσι ένας (1,1) global predictor θα έκανε πάντα σωστή πρόβλεψη (σίγουρα μετά την πρώτη εκτέλεση του loop ανεξαρτήτως αρχικοποίησης).
[Αντίθετα ένας 2-bit predictor, θα έκανε πάντα 1 λάθος κάθε φορά που θα έμπαινε ξανά στο loop].

ii) Branch 2

Το αποτέλεσμα του Branch2 εξαρτάται πάντα από το αποτέλεσμα των προηγούμενων 3 branches (Branch4, Branch5 του προηγούμενου iteration και Branch1 του ίδιου iteration). Συγκεκριμένα, έχουμε

Branch4	***Branch5***	***Branch1***	***Αποτέλεσμα Branch2***
N	*N*	*T*	*T*
T	*T*	*N*	*T*
N	*T*	*N*	*N*

Επομένως ένας (3,1) global predictor θα έκανε πάντα σωστές προβλέψεις.

iii) Branch 5

Το Branch5 είναι σχεδόν πάντα T. Θα χρησιμοποιήσουμε έναν 2-bit predictor, ο οποίος θα κάνει μόνο 1 λάθος σε κάθε εκτέλεση του loop (κάθε φορά στην έξοδο).

Σε περίπτωση που παραπάνω από ένας predictors δίνουν το καλύτερο αποτέλεσμα, διαλέξτε τον φθηνότερο. Εξηγείστε τις επιλογές σας.

Θέμα 2ο (25%)

Α. Αναφέρατε ένα πλεονέκτημα και ένα μειονέκτημα των χαλαρών μοντέλων συνέπειας μνήμης σε σχέση με το μοντέλο της ακολουθιακής συνέπειας.

Πλεονέκτημα: επιτρέπουν βελτιστοποιήσεις στην απόδοση (π.χ. bypassing των missing writes μέσω write-buffers)
Μειονέκτημα: τα προγράμματα απαιτούν σχολαστικό tuning από τον προγραμματιστή προκειμένου αφενός να τηρείται η σωστή σημασιολογία (που εξασφαλίζεται αυτομάτως από την SC), και αφετέρου να παίρνουμε το μέγιστο της απόδοσης (π.χ. εισαγωγή ελάχιστου αριθμού barriers/barriers με τους λιγότερους περιορισμούς).

Β. i) Με ποιο τρόπο βελτιώνει την απόδοση του πρωτοκόλλου MESI το Exclusive state σε σχέση με το πρωτόκολλο MSI;

Η απόδοση βελτιώνεται μέσω της μείωσης του όγκου της κίνησης στο διάδρομο. Η μείωση αυτή επιτυγχάνεται, καθώς όταν ένα block είναι στο E δεν χρειάζεται να δημιουργήσει κάποιος bus transaction σε περίπτωση εγγραφής, μιας και είναι σίγουρο ότι η cache έχει το μοναδικό αντίγραφο.

ii) Στο πρωτόκολλο MESI μπορεί ένα block να μεταβεί από το S στο E, αν οι υπόλοιπες caches που έχουν το ίδιο block το διώξουν (evict); Γιατί;

Η μετάβαση αυτή δεν είναι εφικτή, καθώς δεν υπάρχει η πληροφορία πόσοι κόμβοι/caches μοιράζονται κάθε στιγμή ένα block. Έτσι όταν μια cache μείνει με το μοναδικό S block δεν το γνωρίζει και δεν μπορεί να μεταβεί στο E.

Γ. Θεωρήστε ένα συμμετρικό πολυεπεξεργαστικό σύστημα με 4 επεξεργαστές που χρησιμοποιεί το πρωτόκολλο MESI και write-back caches. Υποθέστε ότι η μεταβλητή X αρχικά δε βρίσκεται σε καμία από τις caches. Για την ακολουθία των εντολών μνήμης που παρατίθενται στη συνέχεια, παρουσιάστε την κατάσταση των caches και της κύριας μνήμης μετά από την εκτέλεση κάθε αναφοράς. Συγκεκριμένα, για τις caches δείξτε την κατάσταση MESI στην οποία βρίσκεται η cache line που περιέχει την X, ενώ για την μνήμη δείξτε αν η αντίστοιχη διεύθυνση περιέχει έγκυρα ή άκυρα δεδομένα.

Action	**P0 Cache State**	**P1 Cache State**	**P2 Cache State**	**P3 Cache State**	**Memory State**
	I	I	I	I	V
P0 reads X	*E*	*I*	*I*	*I*	*V*
P0 writes X	*M*	*I*	*I*	*I*	*I*
P1 reads X	*S*	*S*	*I*	*I*	*V*
P2 reads X	*S*	*S*	*S*	*I*	*V*
P1 writes X	*I*	*M*	*I*	*I*	*I*
P3 writes X	*I*	*I*	*I*	*M*	*I*
P3 reads X	*I*	*I*	*I*	*M*	*I*

Δ. Θεωρήστε ένα πολυεπεξεργαστικό σύστημα κοινής μνήμης 2 επεξεργαστών καθώς και τον ακόλουθο κώδικα για 2 διεργασίες που εκτελούνται παράλληλα στο σύστημα. Αρχικά ισχύει ότι A=B=flag1=flag2=0.

P1	**P2**
A = 1;	B = 1;
if (B == 0) flag1 = 1;	if (A == 0) flag2 = 1;
print(A, B, flag1, flag2);	

Ποιες από τις εκτυπώσεις δεν είναι συμβατές με το μοντέλο ακολουθιακής συνέπειας και γιατί;
i) (A, B, flag1, flag2) = (1,1,1,1)
Δεν υπάρχει περίπτωση να είναι τα flag1, flag2 ταυτόχρονα ίσα με 1. Αυτό συμβαίνει διότι οποιαδήποτε σειριοποιημένη ακολουθία αναφορών κατά το μοντέλο SC θα ξεκινάει είτε με A=1 είτε με B=1, που σημαίνει ότι τουλάχιστον ένα από τα if blocks δε θα εκτελεστεί τελικά.

ii) (A, B, flag1, flag2) = (1,0,0,0)
Αν B==0, αυτό σημαίνει ότι θα πρέπει να εκτελεστεί flag1=1 πριν την εντολή print. Άρα δεν είναι συμβατό το αποτέλεσμα με την SC.

iii) (A, B, flag1, flag2) = (1,0,1,0)
Μπορεί να συμβεί αν δεν εκτελεστεί καμία από τις εντολές του P2

iv) (A, B, flag1, flag2) = (0,1,0,1)
Για να εκτελεστεί η εντολή print θα πρέπει να έχει αναγκαστικά εκτελεστεί η εντολή A=1. Άρα δεν μπορεί να προκύψει το αποτέλεσμα αυτό.

Θέμα 3ο (30%)

Δίνεται αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το pipeline του επεξεργαστή περιέχει τα στάδια Issue (IS), Execute (EX), Write Result (WR) και Commit (CMT), αγνοούμε δηλαδή τα IF και ID. Ισχύουν επίσης τα ακόλουθα :

- Τα IS, WR, CMT απαιτούν 1 κύκλο.
- Το σύστημα περιέχει *περιορισμένο* αριθμό από reservation stations (RS). Συγκεκριμένα, περιέχει 1 RS για προσθέσεις/αφαιρέσεις και 1 RS για πολλαπλασιασμούς/διαιρέσεις floating point αριθμών. Αντίστοιχα, για integer αριθμούς περιλαμβάνονται 3 RS για εντολές διακλάδωσης, αριθμητικές και λογικές εντολές καθώς και 1 RS για πολλαπλασιασμούς/διαιρέσεις.
- Το σύστημα περιλαμβάνει 1 non-pipelined functional unit για πράξεις integer αριθμών. Όλες οι εντολές μεταξύ integer αριθμών διαρκούν 1 κύκλο.
- Το σύστημα περιλαμβάνει 2 non-pipelined floating point functional units, ένα για ADDD / SUBD και ένα για MULD / DIVD. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 4 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 7 κύκλους.
- Για τις εντολές αναφοράς στη μνήμη, στο στάδιο EX γίνεται τόσο ο υπολογισμός της διεύθυνσης αναφοράς όσο και η προσπέλαση στη μνήμη. Το σύστημα περιλαμβάνει ένα Load και ένα Store Queue, τα οποία διαθέτουν 2 θέσεις. Οι εντολές χρησιμοποιούν ένα ξεχωριστό functional unit για τον υπολογισμό της διεύθυνσης και διαρκούν 2 κύκλους στην περίπτωση Hit στην cache και 5 κύκλους σε περίπτωση Miss.
- Οι εντολές διακλάδωσης υπό συνθήκη χρησιμοποιούν τα κατάλληλα RS και FU για αφαίρεση, προκειμένου να υπολογίσουν αν ισχύει η συνθήκη. Η πρόβλεψη για μια εντολή διακλάδωσης υπό συνθήκη γίνεται *ταυτόχρονα* με τη δρομολόγηση της εντολής. Ο έλεγχος της πρόβλεψης γίνεται αμέσως μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR (κύκλος k). Σε περίπτωση σφάλματος, σταματά η εκτέλεση των εντολών του miss-predicted execution path και στον επόμενο κύκλο (κύκλος k+1) δρομολογείται η σωστή εντολή.
- Ο ROB έχει 9 θέσεις.
- Το σύστημα περιλαμβάνει 1 CDB. Σε περίπτωση που παραπάνω από μια εντολές θέλουν να το χρησιμοποιήσουν, τότε προτεραιότητα αποκτά η ‘‘*παλαιότερη*’’ εντολή (αυτή που έγινε issued πρώτη). Θεωρήστε ότι τα branches δεν χρησιμοποιούν το CDB κατά τη διάρκεια του WR σταδίου τους.
- Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί έναν πίνακα από 1-bit predictors με συνολικά 32 εγγραφές (entries). Οι predictors είναι αρχικοποιημένοι στο NT.
- Το σύστημα περιλαμβάνει μια fully associative cache με 32 cache lines και μέγεθος block = 16 bytes. Αρχικά η cache είναι άδεια.
- Ο καταχωρητής R1 περιέχει την διεύθυνση του πρώτου στοιχείου ενός πίνακα αριθμών διπλής ακρίβειας (μήκους 8 bytes ο καθένας). Ο πίνακας είναι ευθυγραμμισμένος.

Δίνεται ο παρακάτω κώδικας :

```
0x00880004        LOOP:       LD    F1, 0(R1)
0x00880008                    ADDD  F2, F1, F1
0x0088000C                    MULD  F4, F2, F1
0x00880010                    ADDI  R1, R1, #8
0x00880014                    ADDI  R2, R2, #1
0x00880018                    ANDI  R4, R2, #1
0x0088001C                    ADDI  R4, R4, #-1
0x00880020                    BNEZ  R4, L1
0x00880024                    LD    F1, 0(R1)
```

```
0x00880028                  ADDD  F4, F4, F1
0x0088002C        L1:       ADDI  R6, R6, #-1
0x00880030                  BNEZ  R6, LOOP
0x00880034                  ST    F4, 0(R1)
0x00880038                  ADD   R4, R5, R7
```

Δίνονται οι αρχικές τιμές R2=0 και R6=2. Εκτελέστε τον παραπάνω κώδικα και δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών σε έναν πίνακα όπως ο παρακάτω :

OP	IS	EX	WR	CMT	Σχόλιο
L.D F1, 0(R1)	1	2-6	7	8	Miss on A[0] (fetch A[0], A[1])
ADDD F2,F1,F1	2	8-11	12	13	RAW(F1)
MULD F4,F2,F1	3	13-19	20	21	RAW(F1,F2)
ADDI R1,R1,#8	4	5-5	6	22	R1→A[1]
ADDI R2,R2,#1	5	6-6	8	23	CDB conflict, R2=1
ANDI R4,R2,#1	6	9-9	10	24	RAW(R2), R4=1
ADDI R4,R4,#-1	7	11-11	13	25	RAW(R4), R4=0, CDB conflict
BNEZ R4,L1	9	14-14	15	26	Integer RS full, RAW(R4), pred = NT, res = NT
LD F1,0(R1)	10	11-12	14	27	Hit on A[1], CDB conflict
ADDD F4,F4,F1	13	21-24	25	28	RAW(F1, F4)
ADDI R6,R6,#-1	14	15-15	16	29	ROB full, R6=1
BNEZ R6, LOOP	22	23-23	24	30	ROB full, pred = NT, res = T
ST F4,0(R1)	23				RAW(F4), flush @ cycle 24
ADD R4, R5, R7	24				Flush @ cycle 24
L.D F1,0(R1)	25	26-27	28	31	Hit on A[1]
ADDD F2,F1,F1	26	29-32	33	34	RAW(F1)
MULD F4,F2,F1	27	34-40	41	42	RAW(F1, F2)
ADDI R1,R1,#8	28	29-29	30	43	R1→A[2]
ADDI R2,R2,#1	29	30-30	31	44	R2=2
ANDI R4,R2,#1	30	32-32	34	45	RAW(R2), CDB conflict, R4=0
ADDI R4,R4,#-1	31	35-35	36	46	RAW(R4), R4=-1
BNEZ R4,L1	32	37-37	38	47	RAW(R4), pred = NT, res = T
L.D F1,0(R1)	33	34-38			Miss on A[2] (fetch A[2], A[3]), flush @ cycle 38
ADDD F4,F4,F1	34				RAW(F1, F4), flush @ cycle 38
ADDI R6,R6,#-1	35	36	37		Flush @ cycle 38
ADDI R6,R6,#-1	39	40-40	42	48	ROB full (until cycle 38) , R6=0
BNEZ R6,LOOP	40	43-43	44	49	RAW(R6), pred = T, res = NT
LD F1,0(R1)	41	42-43	44		Hit on A[2], flush @ cycle 44
ADDD F2,F1,F1	43				RAW(F1), ROB full, flush @ cycle 44
MULD F4,F2,F1	44				Flush @ cycle 44
ST F4,0(R1)	45	46-47	48	50	Hit on A[2]
ADD R4, R5, R7	46	47	48	51	Η ST δε χρησιμοποιεί το CDB.

Θέμα 4ο (20%)

Α. Για κάθε μία από τις ακόλουθες προτάσεις απαντήστε αν είναι σωστή ή λάθος, δικαιολογώντας την απάντησή σας. Υποθέστε ότι όλες οι παράμετροι της κρυφής μνήμης παραμένουν σταθερές εκτός αυτής στην οποίαν αναφέρεται η κάθε ερώτηση.

α. *Ο διπλασιασμός του associativity διπλασιάζει το πλήθος των tags στην cache*
ΛΑΘΟΣ. Ο συνολικός αριθμός των cache lines παραμένει ο ίδιος, επομένως και ο αριθμός των tags.

β. *Ο διπλασιασμός της χωρητικότητας μιας direct-mapped cache συνήθως μειώνει τα conflict misses*
ΣΩΣΤΟ. Ο διπλασιασμός της χωρητικότητας αυξάνει τις cache lines απο L σε 2L. Ως εκ τούτου, οι διευθύνσεις i και i+L απεικονίζονται σε διαφορετικές θέσεις της cache, ενώ στην αρχική κατάσταση έκαναν conflict.

γ. *Ο διπλασιασμός της χωρητικότητας μιας direct-mapped cache συνήθως μειώνει τα compulsory misses*
ΛΑΘΟΣ. Διπλασιάζεται ο αριθμός των cache lines, όμως το μέγεθος της cache line παραμένει το ίδιο. Συνεπώς τα compulsory misses παραμένουν τα ίδια.

δ. *Ο διπλασιασμός του μεγέθους του cache line συνήθως μειώνει τα compulsory misses*
ΣΩΣΤΟ. Ο διπλασιασμός του line size έχει σαν αποτέλεσμα τη φόρτωση περισσότερων δεδομένων κάθε φορά που έρχεται μία cache line στην cache. Αυτό αυξάνει την πιθανότητα δεδομένα που θα κληθούν για πρώτη φορά σε επόμενες εντολές να βρίσκονται ήδη στη γραμμή που φορτώθηκε, μειώνοντας έτσι τα compulsory misses.

Β. Θεωρήστε το ακόλουθο κομμάτι κώδικα:

```
int A[1024], B[1024], C[1024];
for (i=0; i<512; i++)
    A[i+1] += 2*B[i+1] - C[i];
```

Ισχύουν τα εξής:
- Το πρόγραμμα εκτελείται σε έναν επεξεργαστή με μόνο ένα επίπεδο κρυφής μνήμης δεδομένων, η οποία αρχικά είναι άδεια. Η κρυφή μνήμη είναι 2-way set associative, write-allocate και έχει μέγεθος 1KB.
- Το μέγεθος του block είναι 32 bytes, ενώ το μέγεθος ενός int είναι 4 bytes.
- Δήλωση διαδοχικών μεταβλητών στο πρόγραμμα συνεπάγεται αποθήκευσή τους σε διαδοχικές θέσεις στη μνήμη. Οι πίνακες είναι ευθυγραμμισμένοι.
- Η σειρά με την οποία γίνονται οι αναγνώσεις είναι: A[i+1], B[i+1], C[i].

i) Βρείτε το συνολικό ποσοστό αστοχίας (miss rate) για τις αναφορές που γίνονται στη μνήμη στον παραπάνω κώδικα.

#blocks = 1024/32 = 32 , #sets =16 , κάθε block περιέχει 32/4=8 διαδοχικά στοιχεία ενός πίνακα

A[1] B[1] C[0] A[1] m m m m
A[2] B[2] C[1] A[2] h m m m
A[3] B[3] C[2] A[3] h m m m
...
A[7] B[7] C[6] A[7] h m m m

A[8] B[8] C[7] A[8] m m h h
A[9] B[9] C[8] A[9] h h m m
A[10] B[10] C[9] A[10] h m m m

...
A[15] B[15] C[14] A[15] h m m m

A[16] B[16] C[15] A[16] m m h h
A[17] B[17] C[16] A[17] h h m m
A[18] B[18] C[17] A[18] h m m m
...

A[512] B[512] C[511] A[512] m m h h

*#Hits = 6 + 63*10 + 2 = 638, miss rate = (512*4 - 638)/512*4 = 68.8%*

ii) Ποιες από τις παρακάτω τεχνικές βελτιστοποίησης επιπέδου λογισμικού ή υλικού θα ακολουθούσατε προκειμένου να βελτιώσετε την απόδοση του παραπάνω κώδικα; Δικαιολογήστε την επιλογή ή μη-επιλογή κάθε τεχνικής (χωρίς απαραίτητα να υπολογίσετε το νέο ποσοστό αστοχίας).
α. loop blocking
Δε βοηθάει στη μείωση των conflict misses.

β. loop distribution
Μπορεί να εφαρμοστεί στον παραπάνω κώδικα και θα βοηθούσε στη μείωση των conflict misses. Στην ουσία θα ομαδοποιούνταν οι αναφορές στους A, B και C σε 2 loops (ένα με τους A, B και ένα με τους A, C), και έτσι θα υπήρχαν 2 διαφορετικές αναφορές που δε θα ανταγωνίζονταν για τα blocks του κάθε set.

γ. αύξηση του block size στα 64 bytes (με διατήρηση χωρητικότητας)
Δε βοηθάει στη μείωση των conflict misses.

δ. αύξηση associativity σε 4-way (με διατήρηση χωρητικότητας)
Θα βοηθούσε.