Συνέπεια σε Παράλληλες Αρχιτεκτονικές Πολλών Πυρήνων

Μάιος 2023

Πολλοί Πυρήνες: Παράλληλη Επεξεργασία

- Μοντέλο Παράλληλου Επεξεργαστή Ασθενούς Σύζευξης
  - Χωρίς κοινόχρηστο χώρο διευθύνσεων
  - Προγραμματιστικό μοντέλο συνήθως στηρίζεται σε πέρασμα μυνημάτων
- Μοντέλο Παράλληλου Επεξεργαστή Ισχυρής Σύζευξης
  - Κοινόχρηστος χώρος διευθύνσεων, παραδοσιακά συμμετρική πολυεπεξεργασία (SMP)
  - Προγραμματιστικό μοντέλο όπως σε μονο-επεξεργαστές
  - Οι λειτουργίες σε κοινόχρηστα δεδομένα απαιτούν συγχρονισμό

  - Συνέπεια: οι caches πρέπει να κρατούν τα αντίγραφα μιας τιμής (data) συγχρονισμένα
  - Συγχρονισμός: locks, atomic operations
  - Συνέπεια του υλικού στις λειτουργίες/στόχους του προγραμματιστή (για πολλές τιμές)

Στόχοι της Παράλληλης Αρχιτεκτονικής

- Οι στόχοι της παράλληλης αρχιτεκτονικής γενικά περιλαμβάνουν:
  - Σχεδίαση αρχιτεκτονικής που ξεπερνούν τα bottlenecks που υπάρχουν σε ακολουθιακή (μονή) αλλά και σε παράλληλη εκτέλεση με στόχο τις καλύτερες επιδόσεις και την αποδοτικότητα
  - Διευκόλυνση του προγραμματιστή ώστε να αναπτύσσει σωστά προγράμματα και με υψηλές επιδόσεις παράλληλες εφαρμογές
- Εμπόδια: συγχρονισμός (δεν μπορεί να παραλληλοποιηθεί), ανισορροπία στον φόρτο επεξεργασίας, συναγωνισμός για διαθεσιμότητα των πηγών (μνήμη, δίσκο, περιφερειακά,...)

3

Σειρά Εκτέλεσης στη Μνήμη σε Μονο-Επεξεργαστή

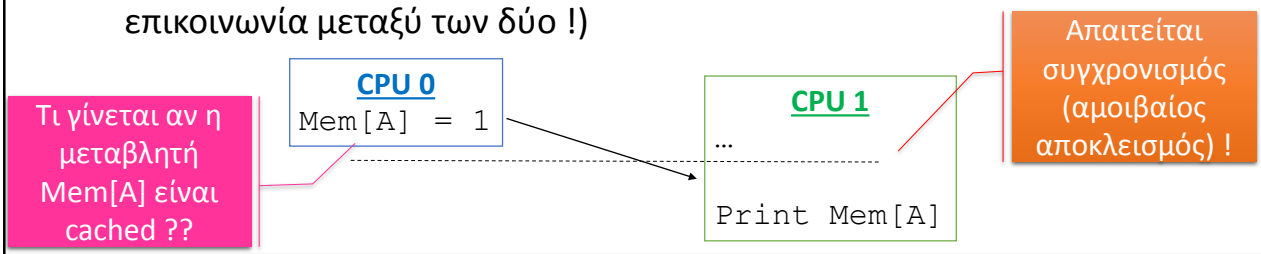
- Σειριακή εκτέλεση: το υλικό εκτελεί τις προσβάσεις στη μνήμη (Loads και Stores) με την σειρά που είναι καθορισμένη στο σειριακό πρόγραμμα
  - Ακόμα και οι επεξεργαστές εκτός-σειράς (OoO) δεν αλλάζουν την σειριακή λογική: στο σημείο της ολοκλήρωσης εκτέλεσης των εντολών εκτός σειράς τις ολοκληρώνει με την σειρά που προβλέπεται στο πρόγραμμα
- Πλεονεκτήματα: σε κάθε εκτέλεση του προγράμματος επαναλαμβάνεται ντετερμινιστικά η ίδια σειριακή εκτέλεση
- Μειονεκτήματα: μειωμένη απόδοση, κυρίως λόγω καθυστέρησης σε speculative εκτέλεση

4

Σειρά Εκτέλεσης στη Μνήμη σε Μοντέλο SAS

- Πολλά παράλληλα προγράμματα επικοινωνούν μέσω κοινόχρηστης μνήμης

- Έστω ο CPU 0 ότι γράφει σε μια διεύθυνση και αμέσως μετά ο CPU 1 διαβάζει από αυτή τη διεύθυνση (αυτή η ακολουθία υποννοεί επικοινωνία μεταξύ των δύο !)



Κάθε διάβασμα πρέπει να λαμβάνει την τελευταία τιμή που αποθηκεύτηκε από οποιονδήποτε !

Είναι Σημαντική η Σειρά Εκτέλεσης ?

- Ευκολία στην διόρθωση / ανίχνευση λαθών (debugging)
  - Είναι σημαντικό πολλαπλές εκτελέσεις ενός προγράμματος να δίνουν την ίδια ακολουθία αλλαγών στην μνήμη

- Επιδόσεις και κόστος
  - Η επιβολή ενός αυστηρού μοντέλου σειριοποίησης στην εκτέλεση δυσκολεύει την σχεδίαση υλικού σε διάφορες τεχνικές βελτιστοποίησης, όπως εκτέλεση OoO, caches…

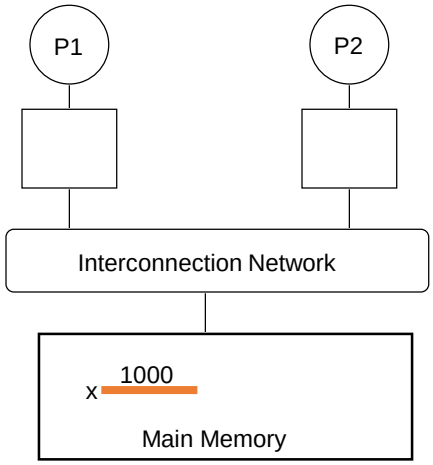
- Ορθή Εκτέλεση
  - Είναι σημαντικό η σειρά εκτέλεσης λειτουργιών στη μνήμη από διάφορους επεξεργαστές να μην οδηγεί σε λάθος αποτελέσματα !

6

Η Έννοια της Συνέπειας της Cache

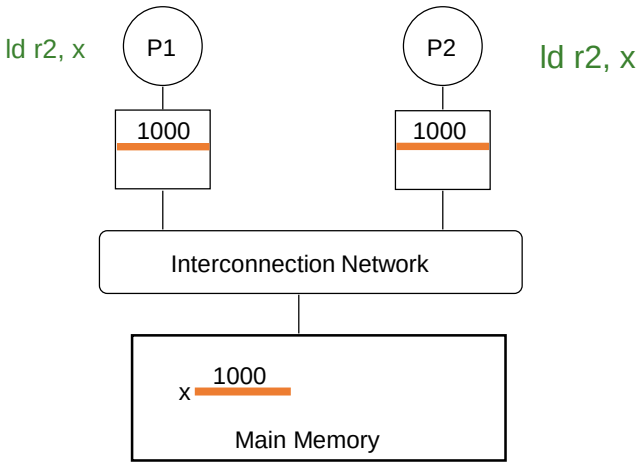
- Θεμελιώδες ερώτημα:

  αν πολλαπλοί επεξεργαστές αντιγράφουν στην cache ίδιο block μνήμης, τότε πως εξασφαλίζεται μια συνεπής κατάσταση των μεταβλητών του block ??



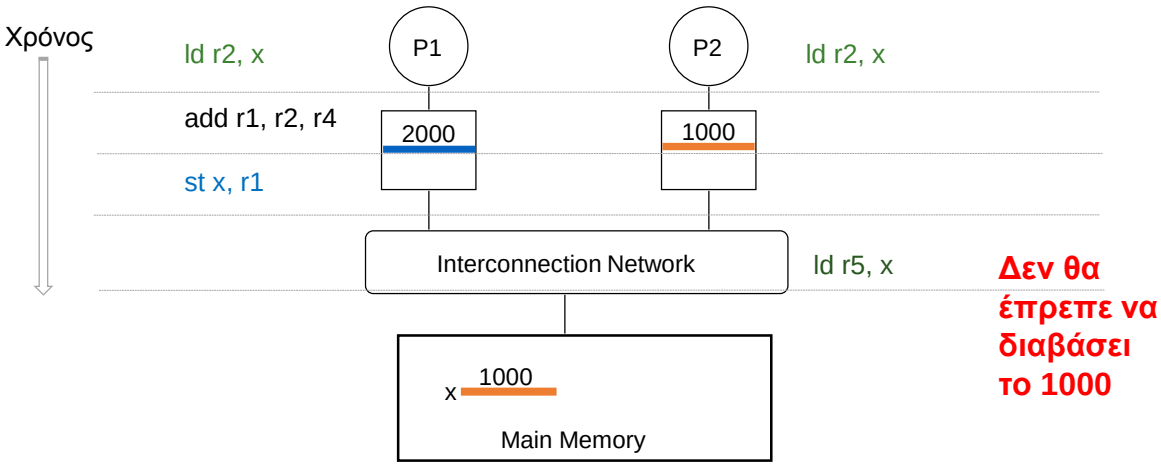
7

Το Πρόβλημα της Συνέπειας στην Cache



8

Το Πρόβλημα της Συνέπειας στην Cache (2)



9

Συνέπεια στην Cache: Δυνατές Λύσεις ?

- Διαχείριση την συνέπειας στις cache μέσω λογισμικού
  - Μπορεί ο προγραμματιστής να εξασφαλίσει συνέπεια αν οι caches δεν είναι ορατές στο προγραμματιστικό μοντέλο ?
  - Μία δυνατή λύση είναι η ISA να υποστηρίζει μια εντολή cache flush
    - Flush-Local A: ακυρώνει/αδειάζει από την cache του CPU το cache block με την δ/νση A
    - Flush-Global A: ακυρώνει/αδειάζει από τις cache όλων των CPU το cache block με την δ/νση A
  - Η λύση μέσω λογισμικού είναι εφικτή αλλά μη πρακτική (προγραμματιστικά..)

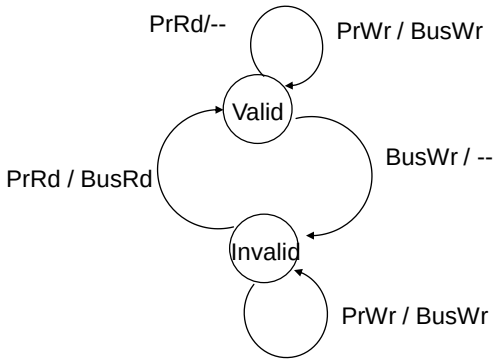
- Στην πραγματικότητα γιατί χρειάζεται η συνέπεια στις cache?
  - Αν είναι όλες οι cache διαμοιρασμένες? -> είναι αδύνατη η σχεδίαση μεγάλης κοινόχρηστης και επεκτάσιμης cache !

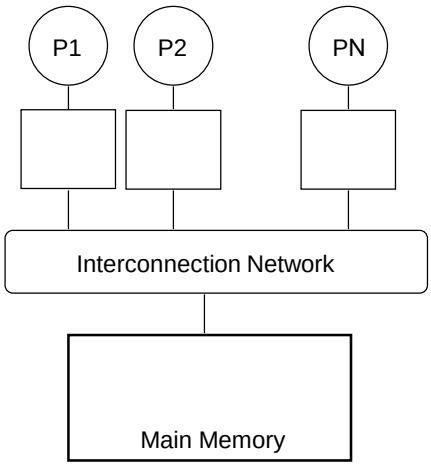
Η συνέπεια στην cache με υποστήριξη από το hardware γενικά είναι η καλύτερη λύση !

Μια Απλή Τεχνική για Συνέπεια

Οι Caches παρατηρούν (snoop) τις λειτουργίες write/read των άλλων cache. Αν ένας επεξεργαστής γράψει σε ένα Block όλες οι άλλες cache το ακυρώνουν από τις δικές τους cache.

- Write-through, no-write-allocate cache
- Actions: PrRd, PrWr, BusRd, BusWr





Σημείωση: ObservedEvent/Action

11

Βασική Ιδέα Τεχνικής Συνέπειας σε Hardware

- Ο Επεξεργαστής/η Cache στέλνει ότι έγραψε μια τιμή σε μια διεύθυνση μνήμης σε όλους τους άλλους επεξεργαστές
- Οι άλλες caches θα πρέπει είτε να ανανεώσουν (update) το τοπικό αντίγραφο της cache τους είτε να το ακυρώσουν (invalidate)
- Πως μπορούμε με ασφάλεια να ανανεώσουμε δεδομένα που είναι αντιγραμμένα σε διάφορες cache ?
  - Πρωτόκολλο Update: στείλε την νέα τιμή σε όλα τα αντίγραφα
  - Πρωτόκολλο Invalidate: εξασφάλιση ότι ένα μοναδικό αντίγραφο υπάρχει σε μία τοπική cache, και ανανέωσε την τιμή του

12

Ανανέωση ή Ακύρωση Αντιγράφων ?

- Ποιοι παράγοντες επηρεάζουν την επιλογή Update ή Invalidate
  - Η συχνότητα των εγραφών και η συμπεριφορά διαμοιρασμού
- Ανανέωση
  - + με την προϋπόθεση ότι το σετ κοινόχρηστων μεταβλητών είναι σταθερό και οι ανανεώσεις σπάνιες, έτσι αποφεύγεται το κόστος να στείλεις σε όλους σήματα ακύρωσης και επαναφόρτωσης στην cache
  - - αν γίνονται εγγραφές χωρίς να γίνονται εμβόλιμες αναγνώσεις από άλλους πυρήνες, τότε τα updates είναι άχρηστα
  - - αν έχουμε write-through τότε το bus αντιμετωπίζει τεράστια κυκλοφορία
- Ακύρωση
  - + όταν μετά από την αποστολή ακυρώσεων, ο πυρήνας έχει αποκλειστική πρόσβαση
  - - αν υπάρχουν πολλές εγγραφές, τότε εμφανίζεται φαινόμενο Ping-pong

13

Τεχνικές Συνέπειας σε Hardware

- Πως εξασφαλίζεται ότι οι κατάλληλες cache γίνονται update?
- Παρακολούθηση του διαύλου (Snoopy Bus)
  - Μέσω διαύλου όλες οι αιτήσεις (αναγνώσεις/εγγραφές) σειριοποιούνται
  - Οι επεξεργαστές παρακολουθούν τις ενέργειες των άλλων επεξεργαστών
- Με Directory
  - Οι επεξεργαστές κάνουν συγκεκριμένα αιτήσεις για blocks
  - Η ιδιοκτησία και ο διαμοιρασμός κάθε Block αποθηκεύεται στο directory
  - Ο directory controller συντονίζει τις ακυρώσεις

14

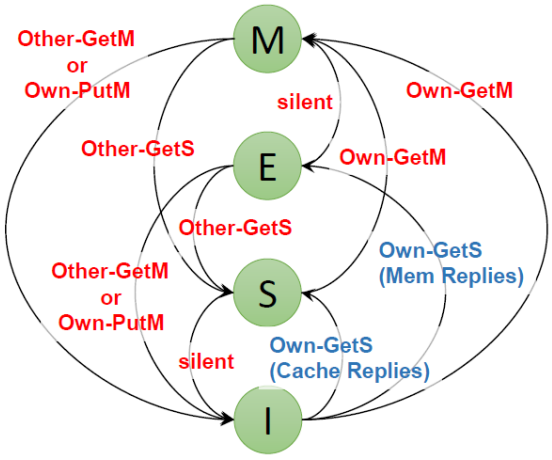
Παράδειγμα: Πρωτόκολλο MESI

- Καταστάσεις: I, E (Exclusive), S (Shared), M (Modified) : ονομάζεται ***MESI***
  - Χρησιμοποιείται ευρέως σε πραγματικές μηχανές
- Χαρακτηριστικά :
  - Η cache γνωρίζει αν ένα αντίγραφο είναι Exclusive (E)
  - Αν κάποια cache έχει ένα αντίγραφο σε κατάσταση E, τότε γίνεται μεταφορά cache-cache
- Πλεονεκτήματα:
  - Αν βρίσκεται σε κατάσταση E δεν χρειάζεται invalidation traffic όταν έχουμε εγγραφή
  - Μειωμένη κυκλοφορία ***upgrade*** για blocks που γίνονται read (πρώτη φορά) και μετά εγγραφή
  - Για ακολουθιακά προγράμματα η κυκλοφορία είναι παρόμοια με έναν μονο-επεξεργαστή
  - Μειωμένη καθυστέρηση μέσω μεταφορών Cache-cache transfer
- Μειονεκτήματα:
  - Πολυπλοκότητα του μηχανισμού που προσδιορίζει το E (exclusiveness)
  - Η μνήμη πρέπει να περιμένει μέχρι να προσδιοριστεί η κατάσταση διαμοιρασμού

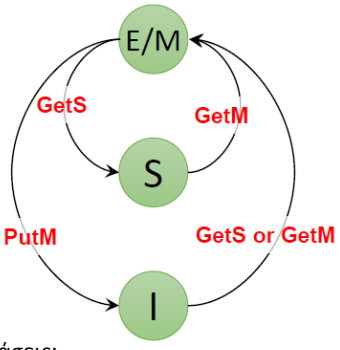
15

MESI, Προδιαγραφές

FSM στον cache controller



FSM στην μνήμη



Καταστάσεις:
Invalid – cache does not have a copy
Shared – cache has a read-only copy; clean
Modified – cache has a copy; writable; dirty
Transactions:
– GetS(hared), GetM(odified), PutM(odified)

16

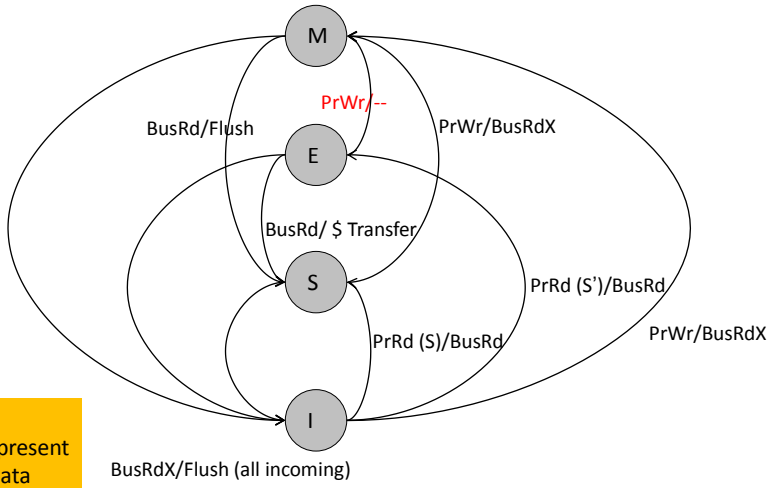
MESI Πρωτόκολλο

Modified:
- 1 owner
- dirty data
- R/W access

Exclusive:
- 1 owner
- clean data
- R/W access

Shared:
- >=1 owner(s)
- clean data
- RO access

Invalid:
- Not present
- No data
- No access

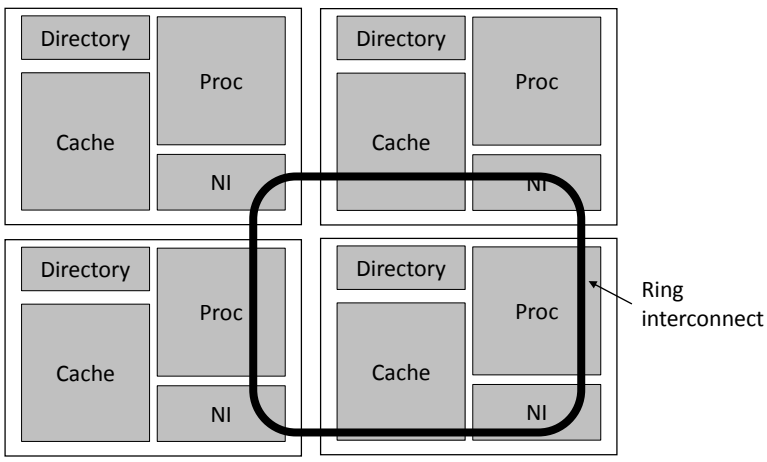


[Culler/Singh96]

17

Πρωτόκολλα με Μεθόδους βάση Directory

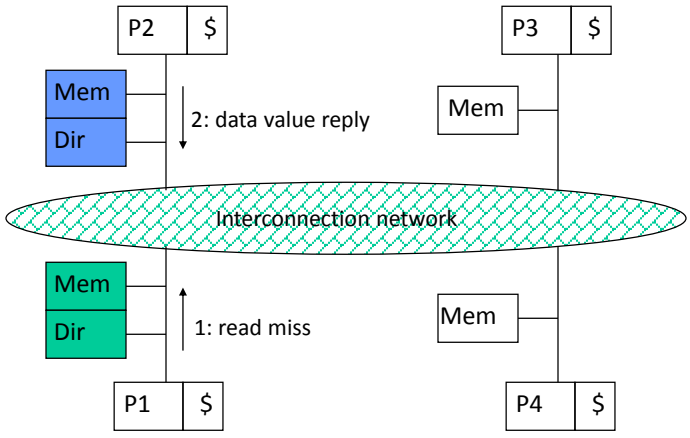
- Οι δίαυλοι είναι απλοί και με μικρό κόστος αλλά μη επεκτάσιμοι...
- Λύση: κατανεμημένη συνέπεια με χρήση Directories
  - Για να εξασφαλιστεί συνέπεια πρέπει να υπάρχει ένα σημείο για σειριοποίηση εγγραφών
  - Η θέση σειριοποίησης μπορεί να είναι διαφορετική ανά block !



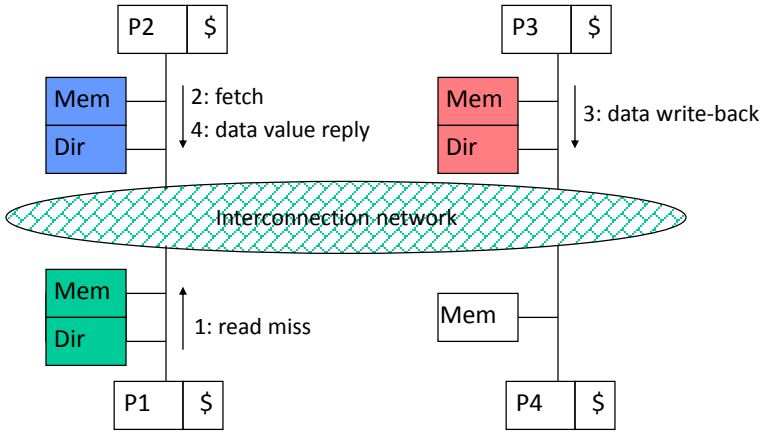
18

19

Λειτουργίες σε Περίπτωση Cache Miss



Read Miss για ένα Απομακρυσμένο Block



Write Miss