Υ-07 Παράλληλα Συστήματα
Αρχιτεκτονική σύγχρονων πυρήνων επεξεργαστών

Αρης Ευθυμίου

Διαδικαστικά

- Ιστοσελίδα μαθήματος:
  http://www.cs.uoi.gr/~plmy07/
- Διαφάνειες μαθημάτων, κτλ

Γρήγορη εκτέλεση σειριακού κώδικα

- Ενας πυρήνας εκτελεί ένα απλό σειριακό πρόγραμμα (νήμα)
- Σκοπός της μικρο-αρχιτεκτονικής είναι η ταχύτερη εκτέλεση
- Πως γίνεται αυτό; Ο πυρήνας
  - Επιτρέπει επικάλυψη ή και παράλληλη εκτέλεση εντολών, αλλά τα αποτελέσματα της εκτέλεσης του νήματος είναι ακριβώς τα ίδια με το αν πράγματι εκτελούνταν σειριακά
  - Αυξάνει τον ταυτοχρονισμό (concurrency) έτσι ώστε να μεγιστοποιήσει το παραπάνω

Γιατί;

- Θα δούμε πως ένας σύγχρονος πυρήνας είναι δομημένος ώστε να εκτελεί γρήγορα σειριακό κώδικα
- Γιατί;
  - Οι πυρήνες των υπολογιστών του εμπορίου είναι πιο σύνθετοι απ'ότι καλύπτεται συνήθως σε προπτυχιακά μαθήματα αρχιτεκτονικής
  - Τα διάφορα «κόλπα» της μικροαρχιτεκτονικής πυρήνα επηρεάζουν σε μεγάλο βαθμό την παράλληλη επεξεργασία
- Ενα άλλο θέμα είναι η οργάνωση κρυφών μνημών
  - Παράλληλες μηχανές μοιρασμένης μνήμης δίνουν μεγάλο βάρος στην οργάνωση κρυφών μνημών

Κύκλος προσκόμισης-εκτέλεσης

Η εκτέλεση εντολών ακολουθεί τα παρακάτω απλά βήματα

- Προσκόμιση της εντολής από την μνήμη (fetch)
- Αποκωδικοποίηση, ανάγνωση τελεστέων από καταχωρητές (decode – operand read)
- Εκτέλεση ή υπολογισμός διεύθυνσης μνήμης (execute)
- Προσπελαση μνήμης (memory access)
- Εγγραφή αποτελέσματος στο αρχείο καταχωρητών (write-back)

Διοχέτευση - pipelining

- Τα προηγούμενα βήματα, γενικά, δεν μπορούν να γίνουν λιγότερα
- Αλλά μπορούμε να τα επικαλύψουμε
- Μία εντολή χρειάζεται περίπου τον ίδιο χρόνο για να εκτελεστεί
- Τί κερδίζουμε;
  - μεγαλύτερο ρυθμό εκτέλεσης εντολών
  - Οι (δυναμικές) εντολές ενός προγράμματος είναι πάρα πολλές
  - Αυξάνοντας το ρυθμό εκτέλεσης, ολοκληρώνουμε την εκτέλεση του προγράμματος πιο γρήγορα

Οριο διοχέτευσης

- Μπορούμε να σπάσουμε την εκτέλεση σε περισσότερα βήματα;
- Ναι και αυτό γινόταν μέχρι πριν κάποια χρόνια,
  - (super-pipelining) όταν οι επεξεργαστές της Intel έτρεχαν με ~4GHz
- αλλά.....
- Χρειάζονται καταχωρητές που προσθέτουν καθυστέρηση
  - μεγάλο μέρος του κύκλου ρολογιού δαπανάται σε «άχρηστη» δουλειά
- Οι εντολές έχουν εξαρτήσεις μεταξύ τους
  - δεδομένων, ελέγχου (διακλαδώσεις)

Το επόμενο βήμα

- Ιδέα: Αντί να προσκομίσουμε 1 εντολή γιατί όχι 2 ή περισσότερες
  - και να εκτελούμε ταυτόχρονα
- Ονομάζεται παραλληλισμός επιπέδου εντολής (instruction-level parallelism ILP)
- Εχει αρκετό κόστος σε υλικό, αλλά ο νόμος του Moore είναι μαζί μας!

Υπερβαθμωτοί (superscalar)

- Προσκομίζουν πολλές εντολές ταυτόχρονα
- Εξετάζουν αν έχουν εξαρτήσεις μεταξύ τους
- Εκτελούν, **με τη σειρά**, όσες ανεξάρτητες εντολές μπορούν ταυτόχρονα
- Η εκτέλεση κάθε «πακέτου» εντολών επικαλύπτεται με αυτή του προηγούμενου
  - δηλαδή έχουμε και διοχέτευση

Ορια ILP

- Τώρα που έχουμε τη δυνατότητα παράλληλης και επικαλυπτώμενης εκτέλεσης εντολών, πόσο «μακρυά» μπορούμε να πάμε;
- Οχι πολύ!
- Οι εντολές έχουν εξαρτήσεις:
  - το αποτέλεσμα μιας εντολής χρειάζεται σε μια άλλη που δεν απέχει πολύ στο πρόγραμμα (ή ο παραγωγός αργεί π.χ. cache miss)
  - Πολύ συχνά (κάθε 5-6 εντολές) το πρόγραμμα παίρνει αποφάσεις που αλλάζουν τη ροή εκτέλεσης (διακλαδώσεις). Πόσες εντολές να εκτελέσει ο πυρήνας όταν δεν ξέρει προς τα πού θα αλλάξει η ροή του νήματος;

Σπρώχνοντας τα όρια του ILP

- Για να αξιοποιήσουμε το υλικό που επιτρέπει παράλληλη εκτέλεση εντολών πρέπει να επεκτείνουμε τον ταυτοχρονισμό (concurrency) του προγράμματος

Πώς;

- Αλλάζοντας τη σειρά εκτέλεσης εντολών
  - Αν η επόμενη εντολή περιμένει κάποιο αποτέλεσμα και η μεθεπόμενη είναι ανεξάρτητη, ας εκτελεστεί
  - Πρόβλημα: πώς θα δίνουμε την εντύπωση σειριακής εκτέλεσης;
- Προβλέποντας τις διακλαδώσεις (branch prediction)
  - οδηγεί σε εκτέλεση βασισμένη σε εικασίες (speculative execution)
  - Πρόβλημα: αν κάναμε λάθος πρόβλεψη;

Πρόβλεψη διακλάδωσης

- Πολλές διακλαδώσεις προκαλούν επαναλλήψεις (loops)
- οι επαναλλήψεις εκτελούνται πολλές φορές
- Αρα μπορούμε να προβλέψουμε ότι η διακλάδωση θα γίνει (taken)
- Αλλες διακλαδώσεις έχουν κάποια σχέση με προηγούμενες
  - π.χ. if (a < 7) {….} if (a <= 5) {…}
  - Αν η πρώτη if είναι αληθής, τότε θα είναι και η δεύτερη

Υλοποίηση πρόβλεψης

- Πίνακας προηγούμενης κατεύθυνσης
  - συνήθως 4 πιθανές τιμές: strongly taken, taken, not-taken, strongly not-taken
  - Πρέπει να κάνουμε 2 φορές λάθος για να άλλαξουμε πρόβλεψη
- Διευθυνση του πίνακα από (μέρος της) διεύθυνσης της εντολής που προσκομίζουμε και από τα αποτελέσματα των Ν προηγούμενων διακλαδώσεων
  - κρατούνται σε ένα καταχωρητή ολίσθησης
- Επιπλέον πίνακας με τη διεύθυνση προορισμού της διακλάδοσης (αν γίνει)

Υλοποίηση πρόβλεψης 2

- Η εντολή διακλάδωσης εκτελείται κανονικά
  - το ίδιο και οι εντολές από τον προορισμό της
- Οταν φτάσει στο στάδιο εκτέλεσης
  - ανανεώνει τον πίνακα πρόβλεψης
  - αν η πρόβλεψη ήταν σωστή, τέλος
  - αν ήταν λάθος, πρέπει να ακυρώσει όλες τις εντολές που προσκομίστηκαν μετά από αυτή και να δόσει τη σωστή διεύθυνση προορισμού στο τμήμα προσκόμισης
- Τα λάθη πληρώνονται ακριβά!

Εκτέλεση εκτός σειράς

- Πώς διαφυλάσεται η (αρχική) σειρά εκτέλεσης;
- Η εκτέλεση μιας εντολής έχει 3 φάσεις:
  - σε εξέλιξη
  - ολοκληρώθηκε (completed)
  - δεσμεύτηκε (committed)
- Εντολές που έχουν ολοκληρωθεί έχουν κάνει αλλαγές και τα αποτελέσματά τους μπορούν να χρησιμοποιηθούν από άλλες εντολές
  - αλλά οι αλλαγές δεν φαίνονται έξω από το νήμα
- Το στάδιο δέσμευσης, γίνεται με τη σειρά του προγράμματος
  - οι αλλαγές είναι πλέον ορατές

Δύο όψεις (states) της μηχανής

- Architectural state – αλλάζει μόνο από δεσμευμένες εντολές
  - η επίσημη κατάσταση του νήματος
  - περιλαμβάνει τους καταχωρητές και τη κύρια μνήμη
- Εικαζόμενη (speculative) state – αλλάζει όταν η εκτέλεση μιας εντολής ολοκληρώνεται
  - Υπάρχει «κρυφή» αποθήκευση μέσα στον πυρήνα

Υλοποίηση 2 όψεων

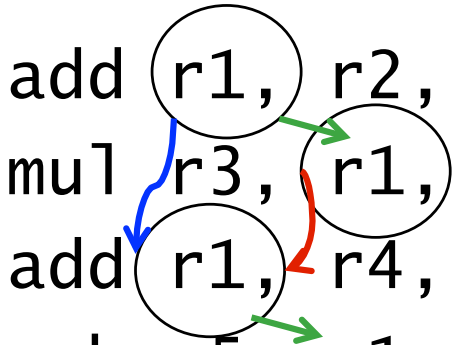
- Δομή δεδομένων σε υλικό (κυκλική ουρά) reorder buffer
- Οι αποκωδικοποιημένες εντολές μπαίνουν με τη σειρά στην αρχή της ουράς
- Οι εντολές που φτάνουν στο τέλος της ουράς (και έχουν ολοκληρωθεί) θεωρούνται δεσμευμένες
  - Τα αποτελεσματά τους «αντιγράφονται» στην architectural state
  - και αφαιρούνται από την ουρά
- Αν μία εντολή φτάσει στο τέλος της ουράς και έχει αποτύχει (exception, branch misprediction)
  - όλες οι εντολές της ουράς αποσύρονται και ξαναπροσκομίζουμε εντολές από την κατάλληλη διεύθυνση

Αποθήκευση τιμών

- Για να υλοποιήσουμε τις δύο όψεις κρατάμε προσωρινές τιμές καταχωρητών από την στιγμή που ολοκληρώνεται η εκτέλεση μιας εντολής μέχρι να δεσμευτεί
  - Είτε κρατάμε προσωρινές τιμές στο reorder buffer και ένα «επίσημο» αρχείο καταχωρητών (register file)
  - ή ένα μεγάλο αρχείο καταχωρητών (physical register file) και ένα πίνακα με τις αντιστοιχίσεις των «αρχιτεκτονικών καταχωρητών» στους φυσικούς
- Είναι αδύνατο να κρατάμε αντίγραφο της κύριας μνήμης
- Πώς μπορούμε να χειριστούμε εντολές αποθήκευσης (store);

Register renaming

- Μερικές εξαρτήσεις μεταξύ εντολών δέν είναι πραγματικές!



```
add r1, r2, 10      add ra, r2, 10
mul r3, r1, 10      mul rb, ra, 10
add r1, r4, 20      add rc, r4, 20
sub r5, r1, 20      sub rd, rc, 20
```

- η εξάρτηση μεταξύ των δύο add μέσω του r1 είναι εξάρτηση write-after-write (WAW): το αποτέλεσμα της δεύτερης add είναι το τελικό
- η εξάρτηση μεταξύ της mul και της 2^{ης} add είναι write-after-read (WAR): η mul πρέπει να προλάβει να διαβάσει την τιμή πριν αλλαχθεί από την 2^{η} add
- Απλά δεν έχουμε αρκετούς καταχωρητές!

Store buffer

- Οι εντολές αποθήκευσης γράφουν τη διεύθυνση και τη τιμή στον store buffer και περιμένουν μέχρι η εντολή να δεσμευτεί πριν ξεκινήσουν την εγγραφή στη μνήμη
- Τι συμβαίνει όμως αν μια εντολή φόρτωσης (load) από τη μνήμη εκτελεστεί πριν αδειάσει ο store buffer;
- Αν η διεύθυνση της load είναι ίδια με τη store, η load παίρνει τα δεδομένα από το store buffer!
- Αλλιώς, επειδή η τιμή θα χρειαστεί σύντομα από κάποια άλλη εντολή, η load συχνά ξεπερνάει τη store και ξεκινάει τη προσπέλαση μνήμης πριν από αυτήν

Store buffer - λεπτομέρειες

- Μια εντολή store χρειάζεται δύο τιμές: την τιμή του καταχωρητή που αποθηκεύεται και τη διεύθυνση (συνήθως το άθροισμα ενός καταχωρητή και μιας σταθεράς)
- Μπορεί η μία από τις δύο τιμές να αργεί να υπολογιστεί γιατί εξαρτάται από μια αλυσίδα εντολών
- Πώς θα ξέρουν επόμενες load τη διεύθυνση ή την τιμή;
  - Συντηρητικά, μια load πρέπει να περιμένει όλες τις προηγούμενες store που δεν έχουν διεύθυνση
  - αλλά μερικοί επεξεργαστές μπορεί να επιτρέψουν τη load να προσπελάσει τη μνήμη και μετά ελέγχουν για λάθη!

Κρυφές μνήμες

Η βασική ιδέα είναι απλή:

- Τα προγράμματα έχουν την ιδιότητα της τοπικότητας αναφορών μνήμης (locality of [memory] reference)
  - χρόνου (temporal): μια θέση μνήμης που διαβάζει/γράφει ένα πρόγραμμα είναι πολύ πιθανό ότι θα προσπελαστεί ξανά σύντομα.
  - «χώρου» (spatial): το ίδιο ισχύει για γειτονικές θέσεις μνήμης
- Η ταχύτητα της μνήμης μειώνεται με το μέγεθός της
  - φυσικά υπάρχουν και άλλοι παράγοντες π.χ. η μνήμη DRAM είναι πολύ οικονομικότερη (και πιο αργή) από την SRAM
- Κατασκευάζουμε μια ιεραρχία μνήμης που δίνει την εντύπωση ότι έχουμε μια μεγάλη και γρήγορη μνήμη
  - εκμεταλεύοντας την παραπάνω ιδιότητα

Οργάνωση κρυφής μνήμης

- Μια κρυφή μνήμη αποτελείται από cache blocks (aka cache lines): θέσεις αποθήκευσης για ένα αριθμό (δύναμη του 2) από συνεχόμενα bytes
  - έτσι εκμεταλεύεται την τοπικότητα χώρου
  - κάθε μεταφορά από/προς τη μνήμη μεταφέρει ολόκληρο το block
- Η κρυφή μνήμη είναι οργανωμένη ώς ένας πίνακας από cache sets
  - το set περιέχει τουλάχιστον ένα block
- Ο όρος block χρησιμοποιείται και για τον χώρο αποθήκευσης και για τα δεδομένα που αποθηκεύει

Λειτουργία

- Ο πυρήνας ζητάει μια λέξη/byte δίνοντας τη διεύθυνση
- Χρησιμοποιούμε ένα μέρος από τη διεύθυνση για να βρούμε σε πιο set θα βρίσκεται το δεδομένο που θέλουμε
- μπορεί να βρίσκεται σε οποιαδήποτε θέση block μέσα στο set που έχει υποδείξει η διεύθυνση
  - πρέπει να τα ψάξουμε όλα
- Τι ψάχνουμε;
  - το υπόλοιπο μέρος της διεύθυνσης, tag, αποθηκεύεται μαζί με τα δεδομένα του block (είναι διαφορετικό για κάθε block)

Είδη κρυφής μνήμης

- Αν σε κάθε set υπάρχει χώρος για μόνο ένα block, η κρυφή μνήμη λέγεται direct mapped
  - κάθε διεύθυνση μνημης αντιστοιχεί σε ένα μόνο set/block.
- Αν η κρυφή μνήμη έχει μόνο ένα set (με πολλά blocks), λέγεται fully associative
  - Κάθε διεύθυνση μπορεί να αντιστοιχιστεί σε οποιοδήποτε block του (ενός) set, αφού το tag συνοδεύει τα δεδομένα
- Στις υπολοιπες περιπτώσεις λέγεται way associative
  - ο αριθμός (N) των block του set λέγεται associativity
  - κάθε «στήλη» από blocks, λέγεται way
  - Κάθε διεύθυνση μπορεί να αντιστοιχιστεί σε οποιοδήποτε από τα N blocks ενός set

Λειτουργία 2

- Αν το δεδομένο βρίσκεται σε ένα block του set, έχουμε ευστοχία (hit)
  - κάνουμε ανάγνωση/έγγραφη ανάλογα με το αίτημα του πυρήνα
- Αλλιώς έχουμε αστοχία (miss)
  - μεταβιβάζουμε το αίτημα προς τη κύρια μνήμη (για ολόκληρο το block)
  - όταν πάρουμε την απάντηση, αποθηκεύουμε το καινούριο block και απαντάμε στον πυρήνα
- Επειδή γενικά όλα τα blocks του set περιέχουν δεδομένα, πρέπει να διαλέξουμε πιο θα εκδιωχθεί (victim block)
  - Διάφορες «πολιτικές» αντικατάστασης (replacement policies), π.χ. least recently used (LRU)

Εγγραφές

Μπορεί η κρυφή μνήμη να έχει διαφορετικά δεδομένα από την κύρια;

- write-through: κάθε εγγραφή ενημερώνει και τη μνήμη
- write-back: οι εγγραφές γίνονται τοπικά και η μνήμη ενημερώνεται όταν το block αντικαθίσταται

Οι εγγραφές που αστοχούν μπορούν να χειριστούν με διάφορους τρόπους:

- Οπως οι αναγνώσεις: εξασφάλιση μιας θέσης για το block και τοπική εγγραφή (write-allocate)
- Παρακάμπτοντας τη κρυφή μνήμη (no write-allocate)

non-blocking caches

- Μια απλή κρυφή μνήμη όταν χειρίζεται μια αστοχία, παγώνει την εκτέλεση στον πυρήνα
- Επειδή αστοχίες συμβαίνουν αρκετά συχνά, οι σύγχρονοι πυρήνες έχουν κρυφές μνήμες non-blocking
- Υπάρχουν διάφορες κατηγορίες
  - Συνεχίζουν να εξυπηρετούν εύστοχες μόνο προσπελάσεις για όσο διαρκεί η εξυπηρέτηση της αστοχίας
  - Επιτρέπουν πολλαπλές αστοχίες

Απόδοση

- Υπάρχουν διάφορα μέτρα απόδοσης υπολογιστή
  - χρόνος εκτέλεσης, ρυθμός εκτέλεσης, κατανάλωση ενέργειας/ ισχύος (ή θερμότητα), αξιοπιστία, ...
- Εδώ θα μας απασχολήσει κυρίως ο χρόνος εκτέλεσης

Αριθμός εντολών, CPI

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Αριθμός (δυναμικών) εντολών προγράμματος
  - Καθορίζονται από το πρόγραμμα, ISA, μεταφραστή
- Μέσος όρος κύκλων ανά εντολή (CPI)
  - Καθορίζεται από το υλικό
  - Αν το CPI είναι διαφορετικό για κάθε εντολή
    - Το μέσο CPI επηρεάζεται από το «μίγμα εντολών» του προγράμματος

- Οταν συγκρίνουμε μικροαρχιτεκτονικές συγκρίνουμε απευθείας τα CPI
- Αν το σύστημα μνήμης ήταν ιδανικό και δεν υπήρχαν εξαρτήσεις εντολών κτλ, το (ιδανικό) CPI θα ήταν το αντίστροφο του αριθμού εντολών που εκτελούνται ταυτόχρονα
  - συχνά χρησιμοποιείται ο όρος IPC
- Συχνά μετράμε την καθυστέρηση λόγω των παραπάνω προβλημάτων ώς ένα αριθμό που αθροίζεται στο ιδανικό CPI

Πολυνηματικοί πυρήνες

- Γενικά ένας πυρήνας τρέχει ένα μόνο νήμα κάθε φορά
- Το Λ.Σ. αποφασίζει πότε θα αντικαταστήσει το νήμα που τρέχει
  - όταν περάσει ενας προκαθορισμένος χρόνος, ή το νήμα ζητάει είσοδο-έξοδο (που αργεί), ...
- Μπορούμε να εκμεταλευτούμε άδειους κύκλους ρολογιού όταν το (μοναδικό) νήμα δεν μπορεί να προχωρήσει λόγω εξαρτήσεων δεδομένων (cache miss), κ.α.
- Χρειάζεται σχετικά λίγο επιπλέον υλικό: πολλαπλά αρχεία καταχωρητών, PCs, κ.α.

- Η δυνατότητα αυτή ονομάζεται multi-threading ή hyper-threading
- Ετσι δημιουργούνται επιπλέον εικονικοί πυρήνες ανά φυσικό πυρήνα
  - η Intel χρησιμοποιεί αυτή την ορολογία
  - Δεν έχει σχέση με εικονικές μηχανές πάνω από φυσικές μηχανές (Vmware κτλ)
- Τι κερδίζουμε;
  - Ενα νήμα δεν εκτελείται γρηγορότερα. Σε μερικές περιπτώσεις συμβαίνει το αντίθετο (Sun Niagara, αν υπάρχουν πολλά νήματα)
  - Αλλά ο ρυθμός εκτέλεσης νημάτων αυξάνεται

Περίληψη

- Σε ένα σειριακό πρόγραμμα η εκτέλεση εντολών επικαλύπτεται και μερικές εντολές μπορούν να εκτελεστούν ταυτόχρονα
- Η σειρά εκτέλεσης μπορεί να είναι διαφορετική από την σειρά προγράμματος
  - Αυτό μπορεί να ισχύει και για εντολές προσπέλασης μνήμης
  - Μια load μπορεί να μην εμφανιστεί ποτέ στο υποσύστημα μνήμης, αν βρει την τιμή στο store buffer
- Η κρυφή μνήμη «φιλτράρει» προσπελάσεις μνήμης
- Ενας επεξεργαστής μπορεί να είναι χρονομερισμένος από πολλά νήματα (multithreading)

Βιβλιογραφία:

Moshovos, Sohi, “Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Scaling”, Proceedings of the IEEE, vol. 89, no. 11, November 2001