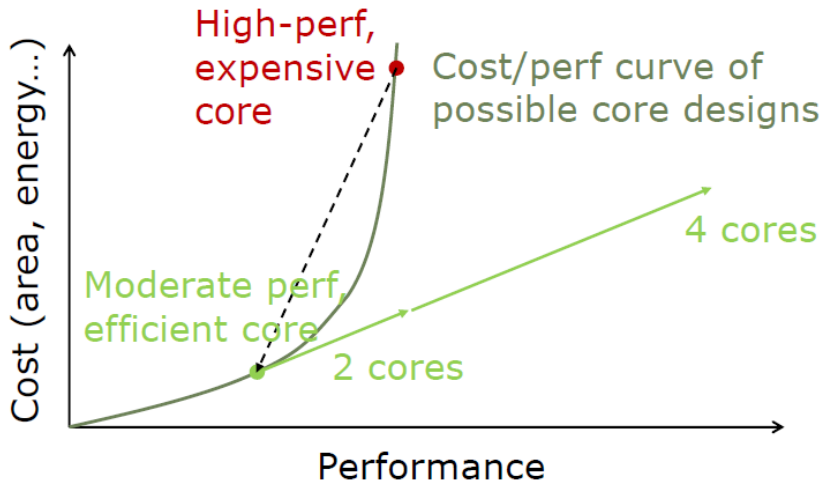
Μνήμη για Transactions

Μάιος 2023

1

Γιατί Πολυπύρηνα Συστήματα ?



2

Αλλά... Ο Παράλληλος Προγραμματισμός είναι Δύσκολος...

- Καταμερισμός του αλγορίθμου σε tasks
- Απεικόνιση των tasks σε νήματα
- Εισαγωγή συγχρονισμού (locks, barriers,…) για να αποφευχθούν data races και συγχρονισμό της σειράς εκτέλεσης των task
- Συναφή προβλήματα: επεκτασιμότητα, τοπικότητα, deadlock, livelock, δικαιοσύνη, συναγωνισμός (races), μεταφερσιμότητα, κλπ...

3

Παράδειγμα: Πίνακες με Hash

- Σειριακή υλοποίηση
- Έχει κινδύνους με εκτέλεση από παράλληλα threads
  - Ταυτόχρονες εισαγωγές και εύρεση (lookup) μπορεί να δημιουργήσουν races
  - Απαιτείται συγχρονισμός

```
V lookup(K key) {
    int idx = hash(key);
    For (;; idx++) {
        if (buckets[idx].empty)
        return NOT_FOUND;
        if (buckets[idx].key == key)
        return buckets[idx].val;
    }
}
```

4

Πίνακες με Hash: Thread-safe με Χρήση Coarse-grain Locks

- Πρέπει να προστεθούν ακόμα lock/unlock ζευγάρια σε όλες τις άλλες λειτουργίες του πίνακα, δηλ. Insert, remove, …
- Αποτέλεσμα: σειριοποιήση λειτουργιών

```
V lookup(K key) {
    int idx = hash(key);
    V result = NOT_FOUND;
    lock(mutex);
    For (;; idx++) {
        if (buckets[idx].empty)
        break;
        if (buckets[idx].key == key)
        result = buckets[idx].val;
        break;
    }
    unlock(mutex);
    return result;
}
```

5

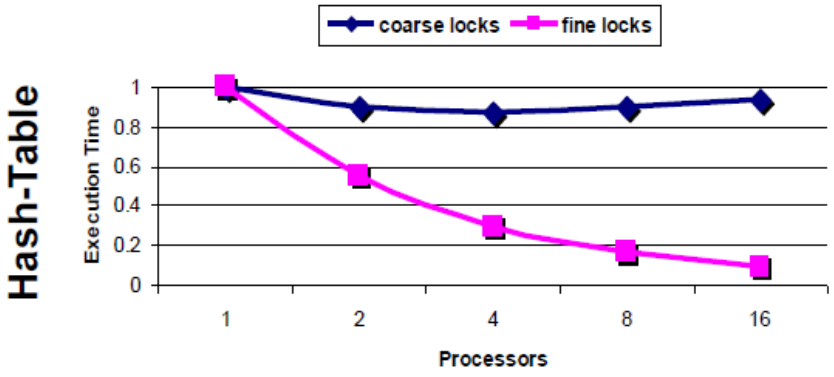
Πίνακες με Hash: Thread-safe με Χρήση Fine-grain Locks

- Locks για κάθε bucket
- Αποτέλεσμα: υπερβολικές καθυστερήσεις για σεριοποίηση λειτουργιών (δεν επιτρέπονται ταυτόχρονα διαβάσματα στο ίδιο bucket…)

```
V lookup(K key) {
    int idx = hash(key);
    V result = NOT_FOUND;
    For (;; idx++) {
        lock(buckets[idx].mutex);
        if (buckets[idx].empty) {
        unlock(buckets[idx].mutex);
        break; }
        if (buckets[idx].key == key) {
        result = buckets[idx].val;
        unlock(buckets[idx].mutex);
        break;}
        unlock(buckets[idx].mutex);
    }
    return result;
}
```

6

Επιδόσεις με Locks



7

Έλεγχος Ταυτόχρονης Πρόσβασης

- Απαιτείται υλοποίηση τεχνικών ελέγχου όταν υπάρχει ταυτόχρονη πρόσβαση (races) σε κοινόχρηστα δεδομένα!
- Λύσεις?
  - Σταμάτημα (stalls): αμοιβαίος αποκλεισμός, εξασφάλιση ότι μόνο ένα thread είναι μέσα στο critical section, όλα τα υπόλοιπα thread περιμένουν
  - Εκτέλεση με υποθέσεις (speculate)
    - Υπόθεση: ότι δεν θα συμβούν συναγωνισμοί κατά τη διάρκεια εκτέλεσης σε critical section
    - Έλεγχος: ανίχνευση όταν συμβεί συναγωνισμοί σε δεδομένα κοινόχρηστα
    - Ανάκαμψη: αν συμβεί συναγωνισμός, τότε επιστροφή στο παρελθόν, ειδάλως commit

8

Transactional Memory

- Ονομάζεται ως transaction στην μνήμη μια ατομική και απομονωμένη ακολουθία από προσβάσεις στην μνήμη
  - Εμπνευσμένη από την ίδια ανάγκη και ονομασία των database transactions
- Ατομικότητα (όλα ή τίποτα)
  - Στο commit, όλες οι εγγραφές στην μνήμη γίνονται αμέσως
  - Σε ακύρωση, καμία εγγραφή δεν εμφανίζεται
- Απομόνωση
  - Κανένας άλλος κώδικας-thread δεν μπορεί να παρέμβει (ούτε να δει) τις εγγραφές πριν γίνουν commit
- Σειριοποίηση
  - Τα transactions φαίνονται να γίνονται commit με την σειρά
  - Η ακριβής σειρά δεν είναι εγγυημένη

9

Προγραμματισμός με TM

```
void deposit(account, amount) {
    lock(account.mutex);
    int t = bank.get(account);
    t = t + amount;
    bank.put(account, t);
    unlock(account.mutex);
}
```

```
void deposit(account, amount) {
    atomic {
        int t = bank.get(account);
        t = t + amount;
        bank.put(account, t);
    }
}
```

- Συγχρονισμός διατυπωμένος από τον προγραμματιστή
  - Ο προγραμματιστής λέει τον συγχρονισμό αλλά όχι το πως θα γίνει
  - Δεν χρειάζεται δήλωση ή διαχείριση για locks
- Το σύστημα υλοποιεί τον συγχρονισμό
  - Συνήθως υλποιείται μέσω εκτέλεσης με υπόθεση
  - Οι επιδόσεις ίσως συναντήσουν συναγωνισμό λόγω R-W ή W-W

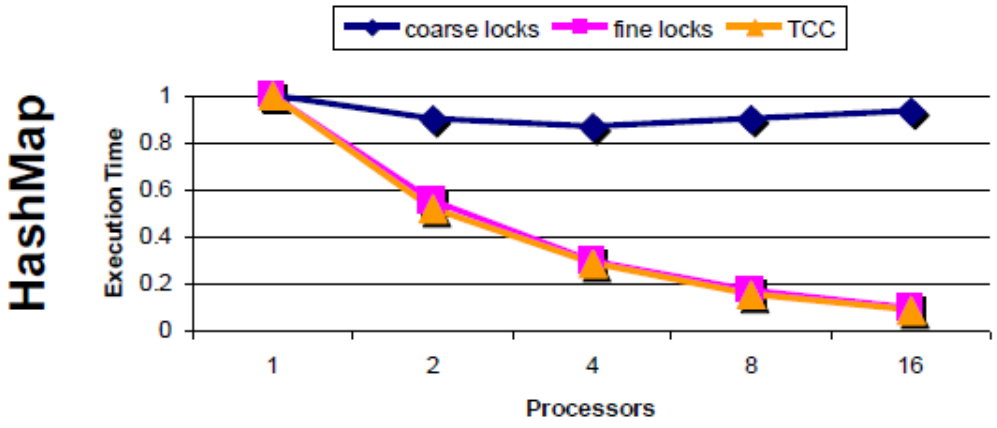
10

Πλεονεκτήματα TM

- Εύκολος συγχρονισμός για τον προγραμματιστή
  - Ο προγραμματιστής δηλώνει, το σύστημα το υλοποιεί
- Καλές επιδόσεις
  - Επιδόσεις τουλάχιστον τόσο καλές όσο τα fine-grain locks
  - Δεν υπάρχει tradeoff μεταξύ επιδόσεων και ορθότητας
- Ασφάλεια στη δόμηση
  - Ασφαλές στην σύνδεση, ανάπτυξη κομματιών λογισμικού

11

Locks vs Transactions



12

Βασικές Τεχνικές Υλοποίησης TM

- Χρήση speculation
- Βασικές απαιτήσεις
  - Διαφορετικές εκδόσεις δεδομένων
  - Ανίχνευση και επίλυση συγκρούσεων
- Επιλογές υλοποίησης
  - Με υλικό (hardware TM)
  - Με λογισμικό (software TM)
  - Υβριδικές λύσεις: επιτάχυνση STM με hardware

13

Τεχνικές Διαχείρισης Δεδομένων

- Διαχείριση νέων (uncommitted) και παλαιών (committed) εκδόσεων των δεδομένων όταν έχουμε ταυτόχρονα transactions
- Ανυπόμονη διαδικασία εκδόσεων (undo-log)
  - Η μνήμη γίνεται update κατευθείαν
  - Κρατιέται πληροφορία για να γίνει επαναφορά (undo-log)
- Τεμπέλικη διαδικασία εκδόσεων (με write-buffer)
  - Κρατιούνται τα δεδομένα σε write buffer μέχρι να γίνει το commit
  - Η μνήμη γίνεται update στο commit

+ Γρήγορα commit

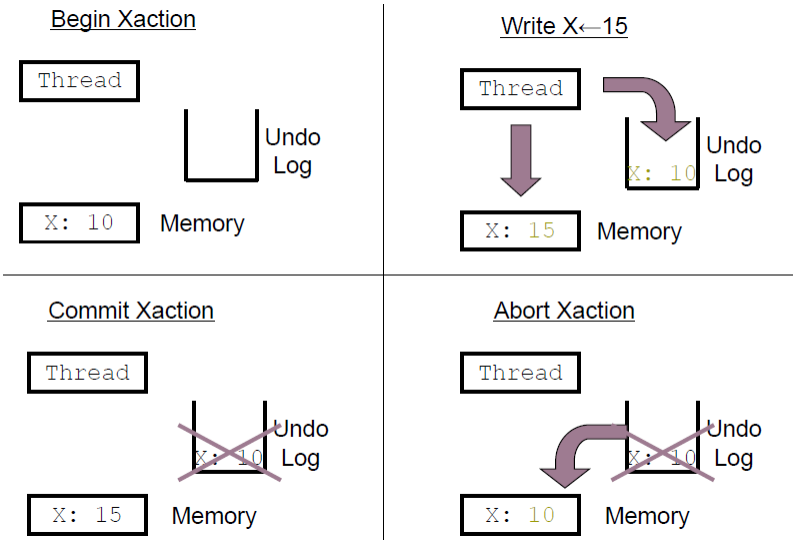
- Αργή επαναφορά

+ Γρήγορη επαναφορά

- Αργά commit

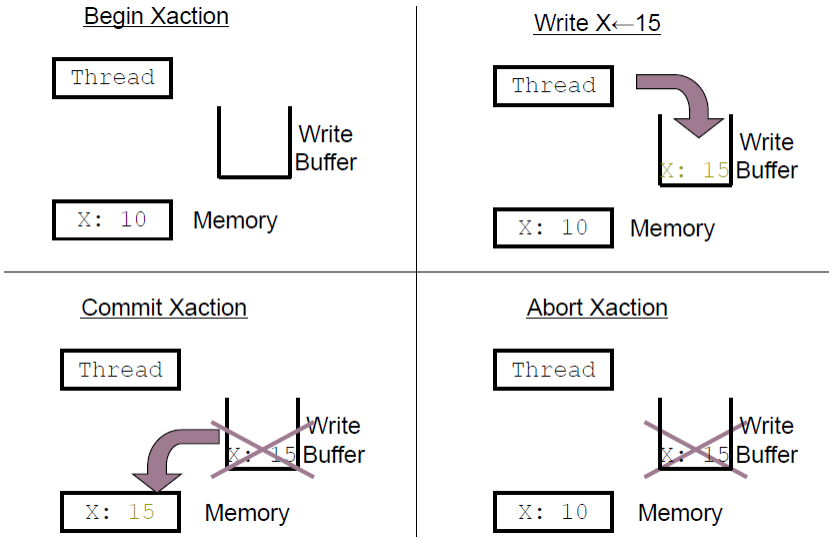
14

Ανυπόμονη Έκδοση



15

Τεμπέλικη Έκδοση



16

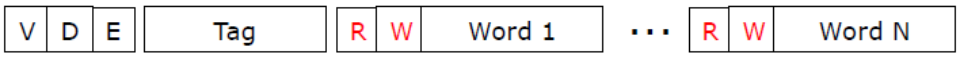
Υλοποίηση TM σε Υλικό (HTM)

- Εκδόσεις δεδομένων με χρήση caches
  - Κρατιέται στην cache ο write-buffer ή το undo-log
  - Κρατιούνται metadata στην cache για τα σετ αναγνώσεων και εγγραφών
- Ανίχνευση συγκρούσεων: χρήση του πρωτοκόλλου coherence
  - Αναζήτηση συγκρούσεων στα transactions
  - Λειτουργεί και με snooping, directory πρωτόκολλα
- Σε ακυρώσεις πρέπει επίσης να αποκαθιστώνται οι καταχωρητές ⇒ register checkpoints

17

Υλοποίηση TM σε Υλικό (HTM) - 2

- Οι γραμμές της cache ανιχνεύουν τα σετ αναγνώσεων και εγγραφών
  - R bit: κρατάει data που διαβάζονται από ένα transaction; set on load
  - W bit: κρατάει data που γράφονται από ένα transaction; set on store
  - R/W bits μπορεί να είναι σε επίπεδο word ή cache-line
  - R/W bits γίνονται clear στο transaction commit ή στο abort

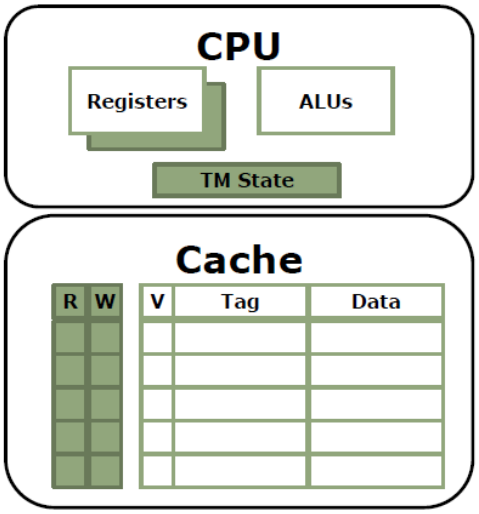


- Αιτήματα συνοχής (coherence) ελέγχουν τα bit R/W για να ανιχνεύσουν συγκρούσεις
  - Shared request to W-word is a read-write conflict
  - Exclusive request to R-word is a write-read conflict
  - Exclusive request to W-word is a write-write conflict

18

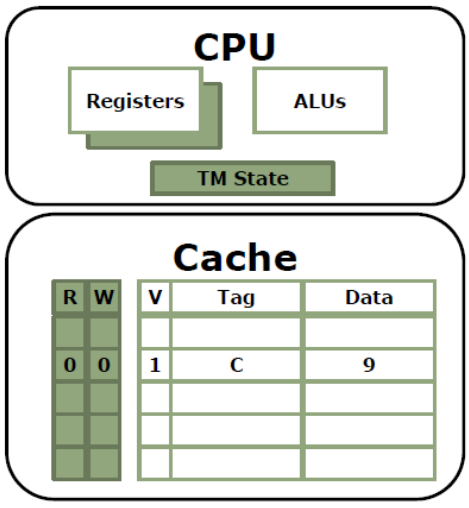
Παράδειγμα HTM: Αισιόδοξα Τεμπέλικο (Lazy Optimistic)

- Αλλαγές στον επεξεργαστή
  - Αποθήκευση καταχωρητών (checkpointing)
  - Ειδικούς καταχωρητές Κατάστασης για TM (στάτους, δείχτες σε handlers, ….)
- Αλλαγές στην cache
  - R/W bits ανά γραμμή της cache
- Με βάση ένα σύστημα διασύνδεσης μέσω διαύλου



19

Εκτέλεση Transaction HTM

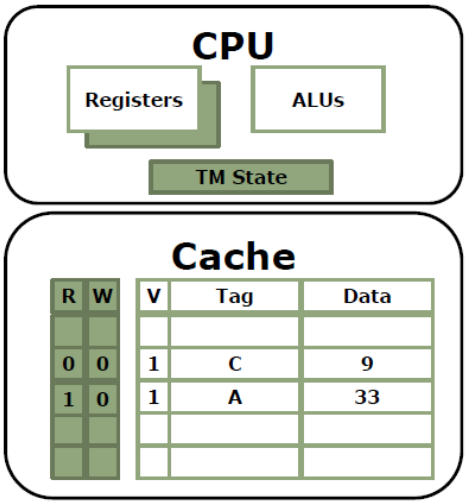



```
⇒  Xbegin
     Load A
     Store B ← 5
     Load C
   Xcommit
```

- Εκκίνηση του transaction
  - Αρχικοποίηση CPU και κατάστασης στην cache
  - Αποθήκευση των καταχωρητών (checkpointing)

20

Εκτέλεση Transaction HTM

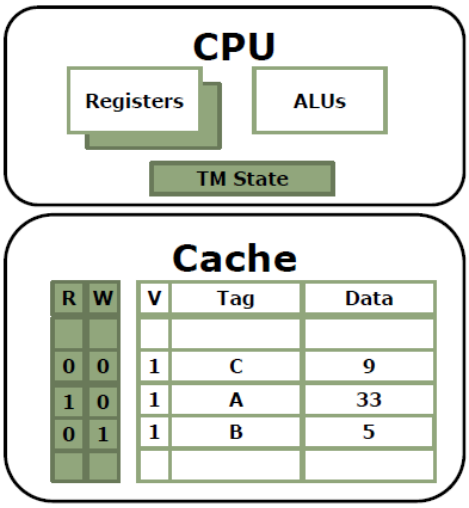


Xbegin
⇒ Load A
Store B ← 5
Load C
Xcommit

- Λειτουργία Διαβάσματος
  - Αν χρειαστεί εξυπηρετείται και το cache miss
  - Γίνεται 1 (set) το R-bit της cacheline

21

Εκτέλεση Transaction HTM

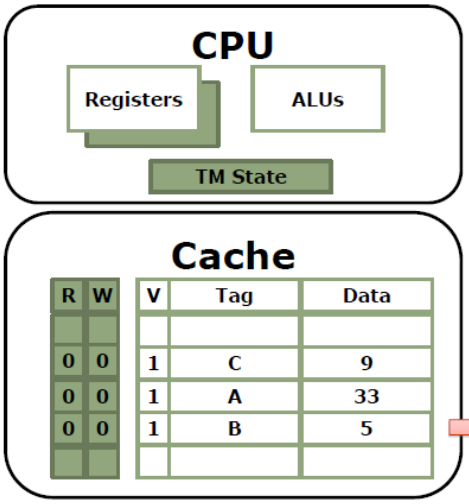


Xbegin
Load A
⇒ Store B ← 5
Load C
Xcommit

- Λειτουργία Εγγραφής
  - Αν χρειαστεί εξυπηρετείται και το cache miss
  - Αν άλλοι πυρήνες έχουν τη γραμμή, τότε θα γίνει shared
  - Γίνεται 1 (set) το W-bit της cacheline

22

Εκτέλεση Transaction HTM

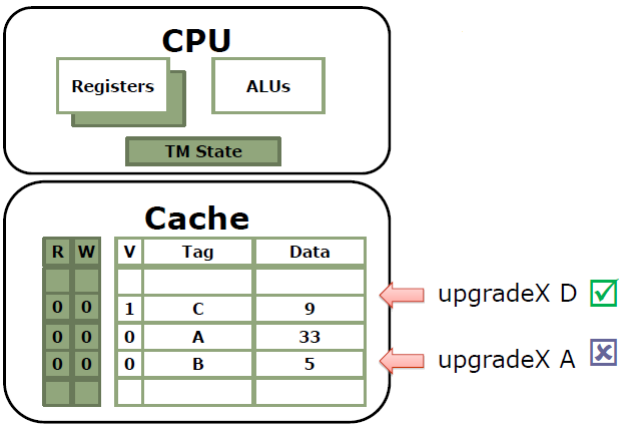



```
Xbegin
    Load A
    Store B ← 5
    Load C
⇒   Xcommit
```

- Ταχύ 2-phase commit
  - Validate: αίτημα για exclusive access στις γραμμές που είναι Write
  - Commit: reset R, W bits, τα write data γίνονται valid

23

Ανίχνευση Σύγκρουσης HTM




```
Xbegin
    Load A
    Store B ← 5
⇒   Load C
Xcommit
```

- Ταχεία ανίχνευση για σύγκρουση και ακύρωση
  - έλεγχος: ανίχνευση για αιτήματα για exclusive access στις γραμμές που είναι Read, Write
  - ακύρωση: invalidate το write-set, reset τα R και W bits, αποκατάσταση από το προηγούμενο checkpoint

24

Πλεονεκτήματα HTM

- Ταχύτατη όταν εμφανίζονται συνήθεις συμπεριφορές (common-case behavior)
  - Zero-overhead tracking of read-set & write-set
  - Zero-overhead versioning
  - Fast commits & aborts without data movement
  - Continuous validation of read-set
- Ισχυρή απομόνωση
  - Conflicts detected on non-transactional loads/stores as well
- Απλοποιείται η multi-core coherence and consistency

25