ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

4η ΑΣΚΗΣΗ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2019-2020, 8ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Τελική Ημερομηνία Παράδοσης: **21/06/2020**

Μέρος Α

1. Εισαγωγή

Στόχος της άσκησης αυτής είναι η εξοικείωση με τους μηχανισμούς συγχρονισμού και τα πρωτόκολλα συνάφειας κρυφής μνήμης (cache coherence protocols) σε σύγχρονες πολυπύρηνες αρχιτεκτονικές. Για το σκοπό αυτό σας δίνεται ένας πολυνηματικός κώδικας με τον οποίο θα υλοποιήσετε διάφορους μηχανισμούς συγχρονισμού, τους οποίους στη συνέχεια θα αξιολογήσετε σε ένα πολυπύρηνο προσομοιώμενο σύστημα με τη βοήθεια του Sniper.

2. Υλοποίηση μηχανισμών συγχρονισμού

Καλείστε να υλοποιήσετε τους μηχανισμούς κλειδώματος Test-and-Set (TAS) και Test-and-Test-and-Set (TTAS), όπως τους διδαχτήκατε στις αντίστοιχες διαφάνειες του μαθήματος.

Συγκεκριμένα, στο αρχείο locks_scalability.c δίνεται έτοιμος κώδικας C ο οποίος χρησιμοποιεί τη βιβλιοθήκη Pthreads (Posix Threads) για τη δημιουργία και διαχείριση των νημάτων λογισμικού. Ο κώδικας δημιουργεί έναν αριθμό από νήματα, καθένα εκ των οποίων εκτελεί μια κρίσιμη περιοχή για ένα συγκεκριμένο αριθμό επαναλήψεων. Για όλα τα νήματα, η είσοδος στην κρίσιμη περιοχή ελέγχεται από μία κοινή μεταβλητή κλειδιού. Πριν την είσοδο στην περιοχή, το κάθε νήμα εκτελεί την κατάλληλη ρουτίνα για την *απόκτηση του κλειδιού* (*lock aqcuire*), ώστε να εισέλθει σε αυτήν κατ' αποκλειστικότητα. Μέσα στην κρίσιμη περιοχή, το κάθε νήμα εκτελεί έναν dummy cpu-intensive υπολογισμό. Κατά την έξοδο από αυτήν εκτελεί την κατάλληλη ρουτίνα για την *απελευθέρωση του κλειδιού* (*lock release*). Δίνοντας τα κατάλληλα flags κατά τη μεταγλώττιση, μπορείτε να παράξετε κώδικα που χρησιμοποιεί κλήσεις σε κλειδώματα TAS, TTAS ή Pthread mutex (MUTEX). Ο τελευταίος από τους μηχανισμούς είναι ήδη υλοποιημένος στην βιβλιοθήκη Pthreads. Εσείς καλείστε να υλοποιήσετε τις ρουτίνες απόκτησης και απελευθέρωσης κλειδιού για τους μηχανισμούς TAS και TTAS.

2.1 Υλοποίηση κλειδωμάτων TAS και TTAS

Πιο συγκεκριμένα, οι ρουτίνες που θα πρέπει να υλοποιήσετε είναι οι **spin_lock_tas_cas**, **spin_lock_tas_ts**, **spin_lock_ttas_cas** και **spin_lock_ttas_cas**. Οι ορισμοί τους δίνονται, ημιτελείς, στο αρχείο lock.h *και εσείς θα πρέπει υλοποιήσετε το κυρίως σώμα τους*. Στο ίδιο αρχείο δίνεται ο ορισμός του τύπου για τη μεταβλητή κλειδιού (spinlock_t), ο ορισμός των σταθερών που σηματοδοτούν αν η κρίσιμη περιοχή είναι κλειδωμένη ή ελεύθερη (LOCKED, UNLOCKED), καθώς και

ο ορισμός της συνάρτησης αρχικοποίησης της μεταβλητής κλειδιού η οποία καλείται πριν τη δημιουργία των νημάτων.

Η υλοποίηση των ζητούμενων ρουτίνων θα πραγματοποιηθεί χρησιμοποιώντας τα atomic intrinsics του gcc και συγκεκριμένα τις εξής 2 συναρτήσεις:

1. *int __sync_lock_test_and_set(int *ptr, int newval);*
2. *int __sync_val_compare_and_swap(int *ptr, int oldval, int newval);*

Η πρώτη συνάρτηση υλοποιεί μία ατομική λειτουργία ανταλλαγής (exchange), δηλαδή:

ατομικά {
*γράψε τη newval στον *ptr και επέστρεψε την παλιά τιμή του *ptr*
}

Η δεύτερη συνάρτηση υλοποιεί μία ατομική λειτουργία compare-and-swap, δηλαδή:

ατομικά {
*αν η τρέχουσα τιμή του *ptr είναι oldval, τότε γράψε τη newval στον *ptr.*
*σε κάθε περίπτωση επέστρεψε την παλιά τιμή του *ptr*
}

2.2 Περιγραφή του προγράμματος

Το πρόγραμμα locks_scalability.c δέχεται τα εξής ορίσματα γραμμής εντολών:

- **nthreads**: ο αριθμός των threads που θα δημιουργηθούν
- **iterations**: ο αριθμός των επαναλήψεων που θα εκτελεστεί η κρίσιμη περιοχή από κάθε νήμα
- **grain_size**: καθορίζει τον όγκο των dummy, cpu-intensive υπολογισμών, και κατ' επέκταση το μέγεθος της κρίσιμης περιοχής

Στον κώδικα του προγράμματος υπάρχουν κατάλληλα compile-time directives τα οποία ορίζουν τη συμπερίληψη ή όχι κατά το χρόνο μεταγλώττισης ενός τμήματος του κώδικα. Με αυτόν τον τρόπο μπορούμε να ενσωματώσουμε σε ένα μόνο αρχείο διαφορετικές εκδόσεις του προγράμματος. Τα directives που χρησιμοποιούνται, και η σημασία τους, είναι τα εξής:

- SNIPER | REAL: ενεργοποιούν τα κατάλληλα τμήματα κώδικα για εκτέλεση του προγράμματος στον Sniper ή σε πραγματικό σύστημα, αντίστοιχα
- TAS_CAS | TAS_TS | TTAS_CAS | TTAS_TS | MUTEX: ενεργοποιούν τις κλήσεις στους μηχανισμούς συγχρονισμού TAS, TTAS και Pthread Mutex, αντίστοιχα
- DEBUG: ενεργοποιεί την εκτύπωση debugging μηνυμάτων

Σε αρχικό στάδιο, στη συνάρτηση main γίνονται οι απαραίτητες αρχικοποιήσεις, όπως η δέσμευση δομών και η αρχικοποίηση της μεταβλητής κλειδιού. Τα δεδομένα εισόδου που προορίζονται για κάθε νήμα ξεχωριστά αποθηκεύονται στη δομή ***targs_t***. Συγκεκριμένα, η δομή περιλαμβάνει το πεδίο my_id που εκφράζει την ταυτότητα ενός νήματος, και αρχικοποιείται σε μια τιμή ξεχωριστή για κάθε νήμα (0 έως nthreads-1).

Σε δεύτερη φάση, ακολουθεί η δημιουργία των νημάτων με χρήση της **pthread_create**. Κάθε νήμα που δημιουργείται αρχίζει και εκτελεί τη συνάρτηση **thread_fn**. Μέσα σε αυτήν, το νήμα αρχικά θέτει το *cpu affinity* του, ορίζει δηλαδή σε ένα bitmask (mask) το σύνολο των cpus του συστήματος ή του simulator όπου μπορεί να εκτελεστεί. Για τους σκοπούς της άσκησης θέλουμε μια "1-1" απεικόνιση ανάμεσα στα νήματα του προγράμματος και τις διαθέσιμες cpus, επομένως κάθε νήμα θέτει το affinity του σε μία και μόνο cpu, αυτή που έχει το ίδιο id με το my_id του. Δηλαδή, το νήμα 0 θα εκτελεστεί στη cpu 0, το νήμα 1 στη cpu 1, κ.ο.κ..

Στη συνέχεια, τα νήματα που έχουν δημιουργηθεί συγχρονίζονται (**pthread_barrier_wait**) ώστε ένα από αυτά (το πρώτο) να ενεργοποιήσει την λεπτομερή προσομοίωση και την καταγραφή των στατιστικών στην περιοχή που μας ενδιαφέρει (SimRoiStart()), αν η εκτέλεση γίνεται στον Sniper, ή να αρχίσει τη μέτρηση του χρόνου εκτέλεσης, αν η εκτέλεση γίνεται σε πραγματικό σύστημα (gettimeofday). Ακολουθεί ο βρόχος εντός του οποίου εκτελείται η κρίσιμη περιοχή, προστατευόμενη από την εκάστοτε υλοποίηση κλειδώματος. Ομοίως, αφού ολοκληρωθεί η εκτέλεση τα νήματα συγχρονίζονται και πάλι ώστε ένα από αυτά να απενεργοποιήσει τη λεπτομερή προσομοίωση ή την μέτρηση του χρόνου.

2.3 Μεταγλώττιση προγράμματος για προσομοίωση στον Sniper

Στο tarball που σας δίνεται υπάρχουν τα αρχεία locks_scalability.c, lock.h καθώς και το Makefile. Το τελευταίο μπορεί να παράξει είτε κώδικα για προσομοίωση στον Sniper ή για εκτέλεση σε πραγματικό μηχάνημα αναλόγως με την τιμή του IMPLFLAG. Μπορείτε να θέσετε το IMPLFLAG σε –DSNIPER ή –DREAL. Για την παραγωγή κώδικα που κάνει χρήση των μηχανισμών TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS ή Pthread Mutex, αρκεί να θέσετε στο Makefile το LFLAG σε –DTAS_CAS, -DTAS_TS, -DTTAS_CAS, -DTTAS_TS ή -DMUTEX, αντίστοιχα. Τέλος, αφού έχετε θέσει το SNIPER_BASE_DIR ώστε να δείχνει στο path στο οποίο έχετε μεταγλωττίσει τον sniper, δίνοντας την εντολή make στο τερματικό θα παραχθεί το εκτελέσιμο με το όνομα locks.

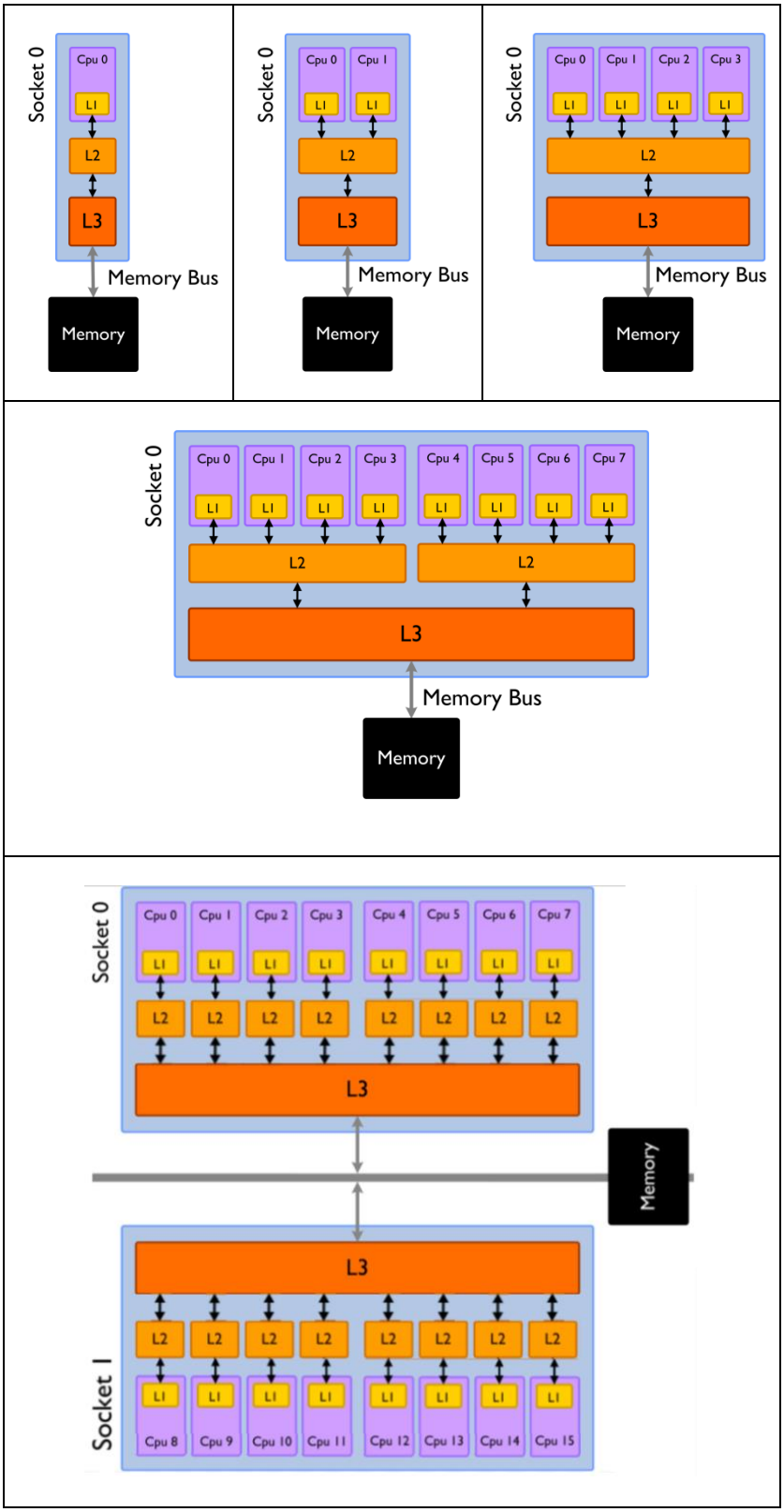
3. Αξιολόγηση επίδοσης

Σε αυτό το μέρος καλείστε να αξιολογήσετε τις επιδόσεις των υλοποιήσεών σας, τόσο ως προς την κλιμακωσιμότητά τους, όσο και ως προς την τοπολογία των νημάτων.

3.1 Σύγκριση υλοποιήσεων

Με τον όρο *κλιμακωσιμότητα* (scalability) εννοούμε πόσο καλά αποδίδει ένα παράλληλο πρόγραμμα καθώς αυξάνεται ο αριθμός των νημάτων από τα οποία αποτελείται. Ακόμα και αν ένα παράλληλο πρόγραμμα είναι σχεδιασμένο για να κλιμακώνει ιδανικά (π.χ., ο χρόνος εκτέλεσης με N νήματα να ισούται με 1/N του χρόνου με 1 νήμα), υπάρχουν διάφοροι εξωγενείς παράγοντες που μπορούν να περιορίσουν την κλιμακωσιμότητά του πολύ κάτω του ιδανικού. Τέτοιοι είναι για παράδειγμα το overhead που σχετίζεται με το πρωτόκολλο συνάφειας, το περιορισμένο bandwidth πρόσβασης στην κύρια μνήμη, κ.ο.κ.. Σε τέτοιες περιπτώσεις, το κόστος μιας λειτουργίας ενός νήματος (π.χ. προσπέλαση μνήμης) είναι πολύ μεγαλύτερο συνήθως όταν στην εκτέλεση συμμετέχουν πολλά νήματα, παρά όταν συμμετέχουν λίγα.

Ζητούμενο του ερωτήματος αυτού είναι η αξιολόγηση και σύγκριση της κλιμακωσιμότητας των μηχανισμών TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS και Pthread mutex για αριθμούς νημάτων 1, 2, 4, 8, 16. Για το λόγο αυτό θα προσομοιώσετε τα πολυπύρηνα συστήματα αντίστοιχου πλήθους πυρήνων που απεικονίζονται στο επόμενο σχήμα και υλοποιούν διαφορετικές πολιτικές διαμοιρασμού των caches. Κάθε τέτοιο σύστημα χρησιμοποιεί το MSI πρωτόκολλο συνάφειας κρυφής μνήμης.



Καλείστε λοιπόν να εκτελέσετε προσομοιώσεις του προγράμματος για όλους τους ακόλουθους συνδυασμούς:

- εκδόσεις προγράμματος: TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS, MUTEX
- iterations: 1000
- nthreads: 1, 2, 4, 8, 16 (σε σύστημα με ισάριθμους πυρήνες)
- grain_size: 1, 10, 100

Για να εκτελέσετε μια προσομοίωση, τρέχετε τον Sniper με όρισμα το επιθυμητό εκτελέσιμο ως εξής:

run-sniper -c <config_script> -n <ncores> --roi –g --
perf_model/l2_cache/shared_cores=<l2sharedcores> –g --
perf_model/l3_cache/shared_cores=<l3sharedcores> -- /path/to/binary/locks <nthreads> 1000 <grain_size>

Το configuration script που θα χρησιμοποιήσετε είναι το ask4.cfg που δίνεται στο tarball.

ΠΡΟΣΟΧΗ: ο αριθμός των νημάτων που δίνετε σαν 1ο όρισμα στο εκτελέσιμο θα πρέπει να ταυτίζεται με τον αριθμό που δίνετε στο όρισμα -n του προσομοιωτή, ώστε το σύστημα που δημιουργεί ο Sniper να έχει ισάριθμους πυρήνες.

ΣΗΜΕΙΩΣΗ: Σε κάποια τρεξίματα ο sniper εμφανίζει το παρακάτω error:

```
[SIFT_RECORDER]     emulation.cc:41:     void     handleCpuid(LEVEL_VM::THREADID,
LEVEL_VM::PIN_REGISTER*,     LEVEL_VM::PIN_REGISTER*,     LEVEL_VM::PIN_REGISTER*,
LEVEL_VM::PIN_REGISTER*): Assertion `emulated' failed.

Pin app terminated abnormally due to signal 6.
```

Στις περιπτώσεις αυτές, αν τα sim.out αρχεία έχουν δημιουργηθεί κανοικά και έχουν μέσα τα αποτελέσματα της προσομοίωσης, μπορείτε να αγνοήσετε το error και να χρησιμοποιήσετε κανονικά στη μελέτη σας τα νούμερα που έχουν εξαχθεί.

3.1.1. Για κάθε grain size, δώστε το διάγραμμα της κλιμάκωσης του συνολικού χρόνου εκτέλεσης της περιοχής ενδιαφέροντος σε σχέση με τον αριθμό των νημάτων. Συγκεκριμένα, στον x-άξονα θα πρέπει να έχετε τον αριθμό των νημάτων και στον y-άξονα τον χρόνο εκτέλεσης σε κύκλους. Στο ίδιο διάγραμμα θα πρέπει να συμπεριλάβετε τα αποτελέσματα και για τις 5 εκδόσεις.

3.1.2. Τι συμπεραίνετε για την κλιμάκωση του χρόνου εκτέλεσης σε σχέση με τη φύση της εκάστοτε υλοποίησης; Τι συμπεραίνετε για την κλιμάκωση του χρόνου εκτέλεσης σε σχέση με το grain size; Δικαιολογήστε τις απαντήσεις σας.

3.1.3. Χρησιμοποιώντας όπως και στην προηγούμενη άσκηση το McPAT, συμπεριλάβετε στην ανάλυση σας εκτός από το χρόνο εκτέλεσης και την κατανάλωση ενέργειας (Energy, EDP κτλ.)

3.1.4. Μεταγλωττίστε τις διαφορετικές εκδόσεις του κώδικα για *πραγματικό* σύστημα. Εκτελέστε τα ίδια πειράματα με πριν είτε σε ένα πραγματικό σύστημα, που διαθέτει πολλούς πυρήνες, είτε σε ένα πολυπύρηνο VM σε περίπτωση που έχετε πρόσβαση σε κάποιο. Χρησιμοποιήστε τους ίδιους αριθμούς νημάτων (με μέγιστο αριθμό νημάτων ίσο με τον αριθμό των πυρήνων που διαθέτει το μηχάνημά σας) και τα ίδια grain sizes με πριν. Αυτή τη φορά όμως δώστε έναν *αρκετά μεγαλύτερο αριθμό επαναλήψεων* ώστε ο χρόνος της εκτέλεσης να είναι επαρκώς μεγάλος για να μπορεί να μετρηθεί με ακρίβεια (π.χ. φροντίστε ώστε η εκτέλεση με 1 νήμα να είναι της τάξης των μερικών δευτερολέπτων).

Δώστε τα ίδια διαγράμματα με το ερώτημα 3.1.1. Πώς συγκρίνεται η κλιμακωσιμότητα των διαφορετικών υλοποιήσεων στο πραγματικό σύστημα σε σχέση με το προσομοιωμένο; Δικαιολογήστε τις απαντήσεις σας.

Για την εκτέλεση σε πραγματικό μηχάνημα, αφού έχετε μεταγλωττίσει το αρχείο *locks* με IMPLFLAG=-DREAL, αρκεί να τρέξετε την παρακάτω εντολή:

/path/to/binary/locks *<nthreads>* 1000 *<grain_size>*

3.2 Τοπολογία νημάτων

Στόχος του ερωτήματος αυτού είναι η αξιολόγηση της κλιμάκωσης των διαφόρων υλοποιήσεων όταν τα νήματα εκτελούνται σε πυρήνες με διαφορετικά χαρακτηριστικά ως προς το διαμοιρασμό των πόρων.

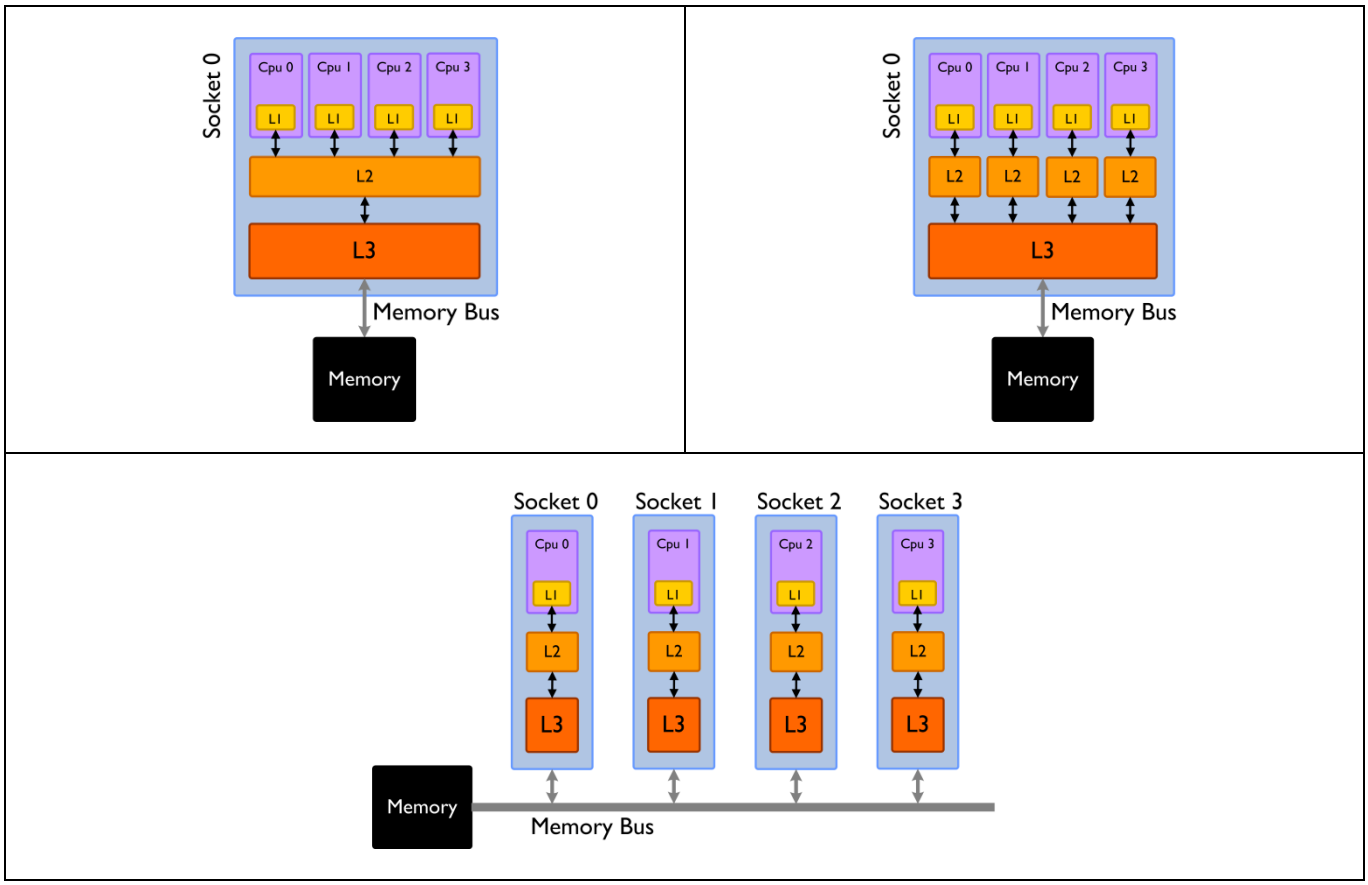
Συγκεκριμένα, θεωρούμε τις εξής πειραματικές παραμέτρους:

- εκδόσεις προγράμματος: TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS, MUTEX
- iterations: 1000
- nthreads: 4
- grain_size: 1

και εξετάζουμε τις ακόλουθες τοπολογίες:

- ***share-all:*** και τα 4 νήματα βρίσκονται σε πυρήνες με κοινή L2 cache
- ***share-L3:*** και τα 4 νήματα βρίσκονται σε πυρήνες με κοινή L3 cache, αλλά όχι κοινή L2
- ***share-nothing:*** και τα 4 νήματα βρίσκονται σε πυρήνες με διαφορετική L3 cache

Οι τοπολογίες αυτές φαίνονται στα παρακάτω σχήματα.



Για την εκτέλεση των προσομοιώσεων χρησιμοποιήστε και αυτή τη φορά το configuration script ask4.cfg, όπως δείξαμε στην ενότητα 3.1, επανακαθορίζοντας όμως συγκεκριμένες παραμέτρους του script που ορίζουν τον τρόπο διαμοιρασμού συγκεκριμένων επιπέδων της ιεραρχίας μνήμης. Συγκεκριμένα, για να μην αλλάζετε το configuration script, μπορείτε να εκτελέσετε τον sniper όπως δείχνουμε παρακάτω προκειμένου να περάσουν οι σωστές τιμές στις αντίστοιχες παραμέτρους:

- Για την τοπολογία share-all:

```
run-sniper -c ask4.cfg -n 4  --roi \
-g --perf_model/l1_icache/shared_cores=1 \
-g --perf_model/l1_dcache/shared_cores=1 \
-g --perf_model/l2_cache/shared_cores=4 \
-g --perf_model/l3_cache/shared_cores=4 \
-- /path/to/binary/locks 4 1000 1
```

- Για την τοπολογία share-L3:

```
run-sniper -c ask4.cfg -n 4 --roi  \
-g --perf_model/l1_icache/shared_cores=1 \
-g --perf_model/l1_dcache/shared_cores=1 \
-g --perf_model/l2_cache/shared_cores=1 \
-g --perf_model/l3_cache/shared_cores=4 \
-- /path/to/binary/locks 4 1000 1
```

- Για την τοπολογία share-nothing:

```
run-sniper -c ask4.cfg -n 4 --roi  \
-g --perf_model/l1_icache/shared_cores=1 \
-g --perf_model/l1_dcache/shared_cores=1 \
-g --perf_model/l2_cache/shared_cores=1 \
-g --perf_model/l3_cache/shared_cores=1 \
-- /path/to/binary/locks 4 1000 1
```

3.2.1. Για τις παραπάνω τοπολογίες, δώστε σε ένα διάγραμμα το συνολικό χρόνο εκτέλεσης της περιοχής ενδιαφέροντος για όλες τις υλοποιήσεις. Χρησιμοποιώντας το McPAT συμπεριλάβετε στην αξιολόγηση σας και την κατανάλωση ενέργειας (Energy, EDP κτλ.). Τι συμπεράσματα βγάζετε για την απόδοση των μηχανισμών συγχρονισμού σε σχέση με την τοπολογία των νημάτων; Δικαιολογήστε τις απαντήσεις σας.

Μέρος Β

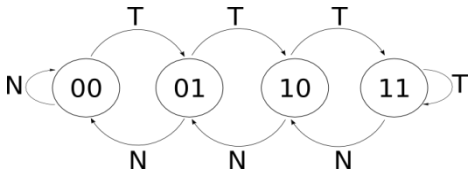
Δίνεται αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το pipeline του επεξεργαστή περιέχει τα στάδια Issue (IS), Execute (EX), Write Result (WR) και Commit (CMT), αγνοούμε δηλαδή τα IF και ID. Ισχύουν επίσης τα ακόλουθα:

- Τα IS, WR, CMT απαιτούν 1 κύκλο.
- Ο επεξεργαστής είναι 2-wide superscalar, μπορεί δηλαδή να δρομολογεί (Issue) και να ολοκληρώνει (Commit) 2 εντολές σε κάθε κύκλο.
- Το σύστημα περιέχει *περιορισμένο* αριθμό από reservation stations (RS). Συγκεκριμένα, περιέχει 3 RS για προσθέσεις/αφαιρέσεις και 2 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 3 pipelined functional units για πράξεις integer αριθμών. Όλες οι εντολές μεταξύ integer αριθμών διαρκούν 2 κύκλους.
- Το σύστημα περιλαμβάνει 2 non-pipelined floating point functional units, 1 για ADDD / SUBD και 1 για MULD / DIVD. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 3 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 5 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, καθένα από τα οποία διαθέτει 2 θέσεις. Οι εντολές χρησιμοποιούν ένα ξεχωριστό pipelined functional unit για τον υπολογισμό της διεύθυνσης και διαρκούν 1 κύκλο στην περίπτωση Hit στην cache και 4 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.
- Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί έναν (1, 2) global history predictor με συνολικά 4 entries, τα οποία φαίνονται στον παρακάτω πίνακα. Ο BHR έχει τιμή ίση με 0. Δίνεται επίσης το FSM διάγραμμα του 2-bit predictor, ο οποίος προβλέπει T για τιμές ≥ 2 και NT για τις υπόλοιπες.

	Column Index	
Row Index	**0**	**1**
0	00	01
1	11	10



Η δεικτοδότηση του πίνακα γίνεται χρησιμοποιώντας τον BHR καθώς και τον κατάλληλο αριθμό low order bits από το PC της εντολής. Ο επεξεργαστής υλοποιεί το ISA του MIPS.

- Το σύστημα περιλαμβάνει μια fully associative cache μεγέθους 32B με block size 16 bytes και πολιτική αντικατάστασης LRU. Αρχικά η cache είναι άδεια.

- Οι καταχωρητές R1, R2 περιέχουν τις διευθύνσεις των πρώτων στοιχείων των πίνακα A και B αντίστοιχα, στους οποίους έχουν αποθηκευτεί αριθμοί διπλής ακρίβειας (μήκους 8 bytes ο καθένας). Οι πίνακες είναι ευθυγραμμισμένοι.

Δίνεται ο παρακάτω κώδικας :

```
0x00448408 LOOP: LD    F0, 0(R1)
0x0044840C       ADDD  F4, F4, F0
0x00448410       LD    F1, 0(R2)
0x00448414       MULD  F4, F4, F1
0x00448418       ANDI  R9, R8, 0x2
0x0044841C       BNEZ  R9, NEXT
0x00448420 IF:   LD    F2, 16(R2)
0x00448424       MULD  F2, F2, F5
0x00448428       ADDD  F4, F4, F2
0x0044842C NEXT: LD    F5, 8(R1)
0x00448430       ADDD  F4, F4, F5
0x00448434       ADDI  R1, R1, 0x8
0x00448438       SUBI  R8, R8, 0x1
0x0044843C       BNEZ  R8, LOOP
0x00448440       SD    F4, 8(R2)
```

Δίνεται η αρχική τιμή R8=1. Εκτελέστε τον κώδικα δίνοντας τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών σε έναν πίνακα όπως ο παρακάτω. Ποια τα τελικά περιεχόμενα της cache;

OP	**IS**	**EX**	**WR**	**CMT**	**Σχόλιο**
L.D F0, 0(R1)	1	2-??	??	??	

Στο πεδίο “Σχόλιο” δικαιολογήστε τυχόν καθυστερήσεις μεταξύ IS-EX, EX-WR και WR-CMT καθώς και ακυρώσεις εντολών

*Παραδοτέο της άσκησης θα είναι ένα ηλεκτρονικό κείμενο (**pdf, docx ή odt**). Στο ηλεκτρονικό κείμενο να αναφέρετε στην αρχή τα στοιχεία σας (**Όνομα, Επώνυμο, ΑΜ**).*

Η άσκηση θα παραδοθεί ηλεκτρονικά στην ιστοσελίδα:

http://www.cslab.ece.ntua.gr/courses/advcomparch/submit

Δουλέψτε ατομικά. Έχει ιδιαίτερη αξία για την κατανόηση του μαθήματος να κάνετε μόνοι σας την εργασία. Μην προσπαθήσετε να την αντιγράψετε από άλλους συμφοιτητές σας.

Μην αφήσετε την εργασία για το τελευταίο Σαββατοκύριακο, απαιτεί αρκετό χρόνο για την εκτέλεση όλων των προσομοιώσεων, ξεκινήστε αμέσως!