3η ΑΣΚΗΣΗ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2021-2022, 8ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Τελική Ημερομηνία Παράδοσης: **10/07/2022**

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 προσομοιωτή "Sniper Multicore Simulator", ο οποίος αξιοποιεί το εργαλείο PIN που χρησιμοποιήσατε στις προηγούμενες ασκήσεις. Λεπτομέρειες και υλικό (παρουσιάσεις, κώδικα, manual κτλ) σχετικά με τον προσομοιωτή μπορείτε να βρείτε εδώ:

http://snipersim.org/w/The_Sniper_Multi-Core_Simulator

2. Εγκατάσταση και χρήση του προσομοιωτή

2.1 Λήψη και εγκατάσταση του sniper

Για την εκτέλεση της άσκησης θα χρησιμοποιήσετε την έκδοση 7.3 του sniper, την οποία μπορείτε να μεταφορτώσετε από το site του εργαστηρίου:

```
$ wget http://www.cslab.ece.ntua.gr/courses/advcomparch/2020/files/askiseis/sniper-7.3.tgz
$ tar xvfz sniper-7.3.tgz
$ cd sniper-7.3
```

Για να χρησιμοποιήσετε το sniper απαιτείται μια έκδοση του pin που περιλαμβάνει και το εργαλείο pinplay και την οποία μπορείτε επίσης να μεταφορτώσετε από το site του εργαστηρίου:

```
$ wget http://www.cslab.ece.ntua.gr/courses/advcomparch/2020/files/askiseis/pinplay-dcfg-3.11-pin-3.11-97998-g7ecce2dac-gcc-linux.tar.bz2
$ tar xvfz pinplay-dcfg-3.11-pin-3.11-97998-g7ecce2dac-gcc-linux.tar.bz2
```

Μετά την λήψη του sniper και του pin προχωρήστε στην μεταγλώττιση του sniper μέσω των παρακάτω εντολών:

```
$ sudo apt-get update
$ sudo apt-get install python make g++ zlib1g-dev libbz2-dev libboost-dev libsqlite3-dev
$ export PIN_HOME=/path/to/pinplay-dcfg-3.11-pin-3.11-97998-g7ecce2dac-gcc-linux/
$ make
```

Όπου /path/to/pin/pinplay-dcfg-3.11-pin-3.11-97998-g7ecce2dac-gcc-linux/ είναι το path στο οποίο βρίσκονται τα αρχεία του pin και του pinplay. Οι παραπάνω οδηγίες μεταγλώττισης έχουν δοκιμαστεί με επιτυχία σε:

- Ubuntu 16.04 με gcc-5.4
- Ubuntu 18.04 με gcc-7

Σε Ubuntu 20.04 η προεπιλεγμένη έκδοση του gcc είναι η έκδοση 9, με την οποία όμως η μεταγλώττιση του sniper αποτυγχάνει. Μπορείτε όμως να χρησιμοποιήσετε την έκδοση 7, η οποία είναι διαθέσιμη στα αποθετήρια της διανομής:

```
$ sudo apt-get install gcc-7 g++-7
$ CC=gcc-7 CXX=g++-7 make
```

2.2 Χρήση του sniper

Αφού ολοκληρωθεί η μεταγλώττιση μπορείτε να τρέξετε τον sniper μέσω του αρχείου run-sniper:

```
$ ./run-sniper
Run program under the Sniper simulator
Usage:
  ./run-sniper  [-n <ncores (1)>]  [-d <outputdir (.)>]  [-c <sniper-config>]  [-c
[objname:]<name[.cfg]>,<name2[.cfg]>,...]  [-c <sniper-options: section/key=value>]  [-
s <script>]  [--roi]  [--roi-script]  [--viz]  [--viz-aso]  [--profile]  [--memory-
profile]  [--cheetah]  [--perf]  [--gdb]  [--gdb-wait]  [--gdb-quit]  [--appdebug]  [--
appdebug-manual]  [--appdebug-enable]  [--follow-execv=1]  [--power]  [--cache-only]  [-
-fast-forward]  [--no-cache-warming]  [--save-output]  [--save-patch]  [--pin-stats]  [-
-wrap-sim=]  [--mpi [--mpi-ranks=<ranks>] [--mpi-exec="<mpiexec -mpiarg...>"] ]  {--
traces=<trace0>,<trace1>,... [--sim-end=<first|last|last-restart (default: first)>]  |
--pinballs=<pinball-basename>,*  |  --pid=<process-pid>  |  [--sift] -- <cmdline> }

Example: $ ./run-sniper -- /bin/ls
```

Από τις παραμέτρους που δέχεται αυτό το script μας ενδιαφέρουν κυρίως οι εξής:

- -d <outputdir> : ο φάκελος στον οποίο θα αποθηκευτούν τα στατιστικά της προσομοίωσης
- -c <config-file>: το αρχείο με τις παραμέτρους του συστήματος που προσομοιώνουμε
- -g <options> : ορισμός παραμέτρων της προσομοίωσης
- --viz : ενεργοποιεί τα visualizations για τα αποτελέσματα της προσομοίωσης

2.3 Παραμετροποίηση του sniper

Η παραμετροποίηση των προσομοιωμένων συστημάτων γίνεται με τη χρήση των config files, τα οποία βρίσκονται στο φάκελο **sniper-7.3/config**. Εκεί βρίσκονται τα configurations για διάφορους επεξεργαστές, όπως το **gainestown.cfg** για τον επεξεργαστή gainestown. Μέσα στο config file ορίζονται οι παράμετροι του επεξεργαστή όπως η συχνότητα, το issue width, ο branch predictor κ.α.. Για παράδειγμα, παρακάτω ορίζεται η L1 data cache:

```
perf_model/l1_dcache]
perfect = false
cache_block_size = 64
cache_size = 32
associativity = 8
replacement_policy = lru
…
```

Όσες παράμετροι δεν ορίζονται στο αρχείο αυτό, ορίζονται είτε στο **base.cfg** (χρησιμοποιείται σε κάθε προσομοίωση από το run-sniper) είτε στο **nehalem.cfg** (included από το **gainestown.cfg**). Επίσης, μπορείτε να ορίσετε παραμέτρους της προσομοίωσης μέσω της γραμμής εντολών με τη χρήση της επιλογής *-g* του run-sniper. Σε κάθε εκτέλεση μιας προσομοίωσης δημιουργείται στο φάκελο των αποτελεσμάτων το αρχείο **sim.cfg**, το οποίο περιέχει αναλυτικά όλες τις παραμέτρους με τις τιμές που χρησιμοποιήθηκαν.

2.4 McPAT

Το McPAT (Multi-core Power, Area, Timing) μοντελοποιεί χαρακτηριστικά ενός επεξεργαστή, όπως η κατανάλωση ενέργειας και το μέγεθος που καταλαμβάνουν στο τσιπ οι διαφορετικές δομικές μονάδες του επεξεργαστή. Ο sniper περιλαμβάνει το McPAT, το οποίο χρησιμοποιείται για την εξαγωγή των στατιστικών

μέσα από τον φάκελο μίας προσομοίωσης που έχει ολοκληρωθεί. Στο βοηθητικό κώδικα της άσκησης, δίνεται επιπλέον το script **advcomparch_mcpat.py** το οποίο είναι μία τροποποιημένη έκδοση του **mcpat.py** που περιέχεται στο sniper. Το **advcomparch_mcpat.py** χρησιμοποιεί το εργαλείο gnuplot για την απεικόνιση αποτελεσμάτων με τη μορφή γραφημάτων. Αν το gnuplot δεν είναι ήδη εγκατεστημένο στον υπολογιστή σας, μπορείτε να το εγκαταστήσετε με την παρακάτω εντολή:

```
$ sudo apt-get install gnuplot
```

Αφού αντιγράψετε το advcomparch_mcpat.py στον φάκελο sniper-7.3/tools μπορείτε να το χρησιμοποιήσετε ως εξής:

```
$ /path/to/sniper-7.3/tools/advcomparch_mcpat.py -h
Usage: advcomparch_mcpat.py [-h (help)] [-j <jobid> | -d <resultsdir (default: .)>] [-t
<type: total|dynamic|static|peak|peakdynamic|area>] [-c <override-config>] [-o <output-
file (power{.png,.txt,.py})>]
```

Από τις παραμέτρους που δέχεται αυτό το script μας ενδιαφέρουν κυρίως οι εξής:

- -d <resultsdir> : ο φάκελος της προσομοίωσης.
- -t <type> : το είδος των στατιστικών που θέλουμε να εξάγουμε.
- -o <output-file> : το όνομα που θα έχουν τα αρχεία που θα παραχθούν.

Η εκτέλεση του script παράγει 4 αρχεία:

- **power.png**: Η γραφική αναπαράσταση των διαφόρων στατιστικών.
- **power.py**: Περιλαμβάνει όλα τα στατιστικά που έχουν παραχθεί σε μορφή κατάλληλη ώστε να μπορεί να χρησιμοποιηθεί απευθείας σε python script.
- **power.txt**: Περιλαμβάνει όλα τα στατιστικά που έχουν παραχθεί σε μορφή κειμένου.
- **power.xml**: Το αρχείο που παράγεται από τον sniper και δίνεται ως είσοδος στο McPAT.

Το advcomparch_mcpat.py τυπώνει επίσης στην οθόνη (ή όπου κάνετε redirect την έξοδο) συνοπτικά τα στατιστικά που ορίσατε με το όρισμα –t.

Ένα παράδειγμα χρήσης του δίνεται παρακάτω:

```
$ advcomparch_mcpat.py -d bench_dir -t total -o bench_dir/power > bench_dir/power.total.out

$ ls -al bench_dir
total 172
drwxrwxr-x     2 user user  4096 Μάι  11 13:24 ./
drwxrwxr-x   458 user user 36864 Μάι  11 13:08 ../
-rw-rw-r--     1 user user  6602 Μάι  11 13:13 power.png
-rw-rw-r--     1 user user 18287 Μάι  11 13:13 power.py
-rw-rw-r--     1 user user 12572 Μάι  11 13:13 power.total.out
-rw-rw-r--     1 user user 12572 Μάι  11 13:13 power.txt
-rw-rw-r--     1 user user 19221 Μάι  11 13:13 power.xml
-rw-rw-r--     1 user user  6121 Μάι   7 21:02 sim.cfg
-rw-rw-r--     1 user user  2481 Μάι   7 21:11 sim.info
-rw-rw-r--     1 user user  1900 Μάι   7 21:11 sim.out
-rw-r--r--     1 user user 45056 Μάι   7 21:11 sim.stats.sqlite3
```

```
$ cat bench_dir/power.total.out

                    Power      Energy      Energy %
core-core           10.15 W      0.92  J      41.45%
core-ifetch         1.78 W       0.16  J      7.26%
core-alu            0.40 W       0.04  J      1.65%
core-int            1.07 W       0.10  J      4.37%
core-fp             1.05 W       0.10  J      4.30%
core-mem            0.63 W       0.06  J      2.59%
core-other          0.95 W       0.09  J      3.89%
icache              0.82 W       0.08  J      3.37%
dcache              1.36 W       0.12  J      5.56%
l2                  1.98 W       0.18  J      8.09%
l3                  0.00 W       0.00  J      0.00%
nuca                0.00 W       0.00  J      0.00%
noc                 0.02 W       2.26 mJ      0.10%
dram                4.25 W       0.39  J      17.37%

core                16.04 W      1.46  J      65.52%
cache               4.17  W      0.38  J      17.02%
total               24.49 W      2.23  J      100.00%
```

2.5 Energy-Delay Product

Παραδοσιακά, για την αξιολόγηση της κατανάλωσης ενός επεξεργαστή χρησιμοποιείται ως μετρική η συνολική κατανάλωση ενέργειας σε Joules. Ωστόσο, πολλές φορές απαιτείται η μελέτη της επίδρασης διαφόρων χαρακτηριστικών του επεξεργαστή όχι μόνο στην κατανάλωση αλλά ταυτόχρονα και στην επίδοσή του. Για το σκοπό αυτό έχει προταθεί ως μετρική το energy-delay product (EDP). Το EDP για την εκτέλεση ενός benchmark ορίζεται ως το γινόμενο της ενέργειας επί τον χρόνο εκτέλεσης του benchmark:

$$EDP = Energy(J) * runtime(\text{sec})$$

Αντίστοιχα, αν θέλουμε να δώσουμε περισσότερο βάρος στον χρόνο εκτέλεσης μπορούμε να υψώσουμε το *runtime* στο τετράγωνο, στον κύβο κ.λ.π. Έτσι προκύπτουν τα ED^2P, ED^3P και ούτω καθεξής:

$$ED^2P = Energy(J) * runtime^2(\text{sec})$$
$$ED^3P = Energy(J) * runtime^3(\text{sec})$$

3. 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.

3.1 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*
}

3.2 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.

3.3 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.

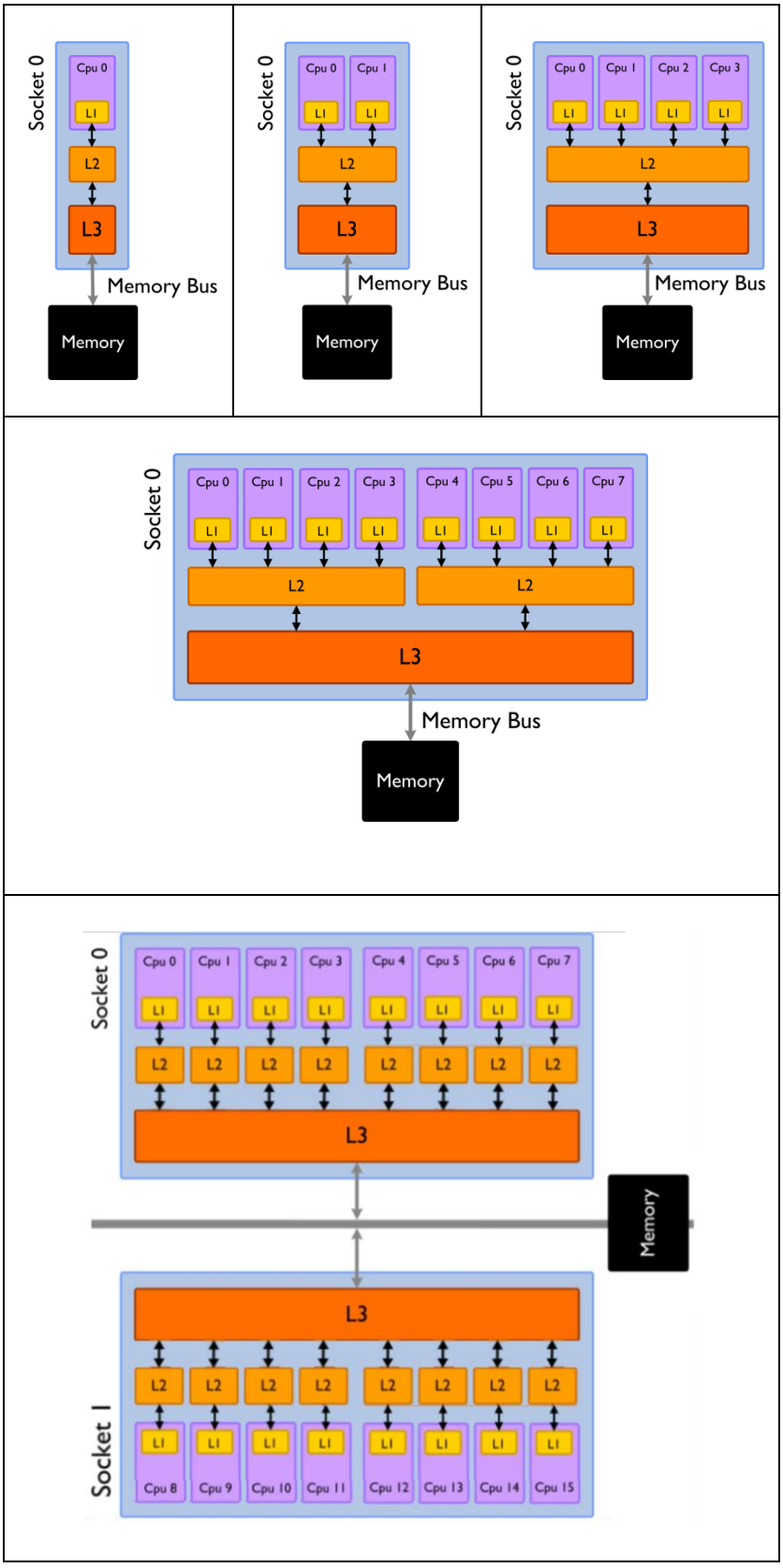
4. Αξιολόγηση επίδοσης

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.

4.1 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 ask3.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 κανονικά και έχουν μέσα τα αποτελέσματα της προσομοίωσης, μπορείτε να αγνοήσετε το error και να χρησιμοποιήσετε κανονικά στη μελέτη σας τα νούμερα που έχουν εξαχθεί.

4.1.1. 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.

4.1.2. 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.

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

4.1.4. 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 4.1.1. 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>

4.2 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.

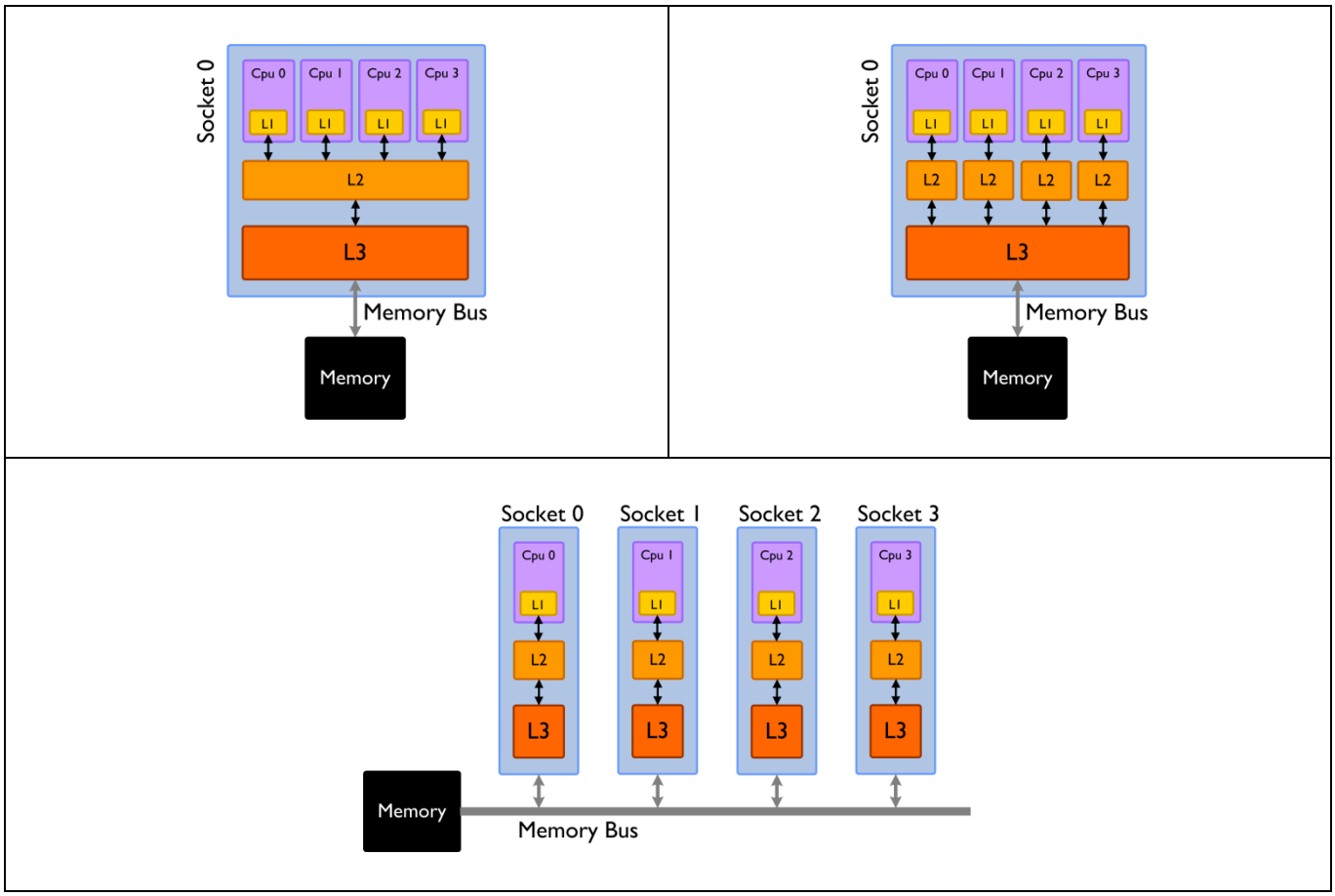
Συγκεκριμένα, θεωρούμε τις εξής πειραματικές παραμέτρους:

- 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 ask3.cfg, όπως δείξαμε στην ενότητα 4.1, 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 ask3.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 ask3.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 ask3.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

4.2.1. 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.

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 στο moodle του μαθήματος:

https://helios.ntua.gr/course/view.php?id=1039

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.

Καθώς τα ίδια εργαλεία είχαν χρησιμοποιηθεί και τις προηγούμενες χρονιές, είναι εξαιρετικά πιθανό αρκετές (αν όχι όλες οι) απορίες που μπορεί να προκύψουν να έχουν ήδη απαντηθεί. Σας συμβουλεύουμε λοιπόν να ψάξετε για απαντήσεις/βοήθεια στα archives της mailing list του μαθήματος, τα οποία βρίσκονται εδώ:

http://lists.cslab.ece.ntua.gr/pipermail/advcomparch/