ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Χειμερινό Εξάμηνο 2022/2023

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

2η Εργαστηριακή Άσκηση

Design Space Exploration με τον gem5

Η εργασία αυτή βασίζεται στην επιτυχή ολοκλήρωση της πρώτης εργαστηριακής άσκησης και για τούτο θα παρατηρήσετε ότι αναφέρεται ως Μέρος Δεύτερο. Συνεπώς, αν για κάποιο λόγο δεν έχετε ολοκληρώσει την πρώτη εργαστηριακή άσκηση, καλείστε να την ολοκληρώσετε προτού προχωρήσετε σε αυτήν εδώ. Στην εργασία αυτή γίνεται η παραδοχή ότι έχετε αποκτήσει μια σχετική εξοικείωση με όσα αναλυτικά αναφέρονται στην πρώτη εργαστηριακή άσκηση (όπως π.χ. το να κάνετε compile ένα πρόγραμμα για να εκτελεστεί στον gem5) και στην παρούσα άσκηση δεν αναλύεται περαιτέρω.

ΕΞΑΙΡΕΤΙΚΑ ΣΗΜΑΝΤΙΚΟ

Όπως μπορεί ήδη να παρατηρήσατε από την πρώτη εργασία, ο gem5 είναι πολύ πιο αργός στην εκτέλεση ενός προγράμματος από ότι αν το τρέχατε κατευθείαν στον υπολογιστή σας. Στη παρούσα άσκηση θα χρειαστεί να τρέξετε τόσο ορισμένα σχετικά μεγάλα προγράμματα, όσο και να πραγματοποιήσετε πολλαπλούς πειραματισμούς με αυτά. Ενώ η πραγματική δουλειά που έχετε να κάνετε δεν είναι μεγάλη, ο χρόνος που θα απαιτηθεί για να τρέξουν τα διαφορετικά προγράμματα είναι σημαντικός. Για τούτο, *συνίσταται εξαιρετικά*, να αρχίσετε την ενασχόληση με την εργασία *νωρίς* γιατί διαφορετικά μπορεί να μην προλάβετε την προθεσμία.

Σε αυτά τα πλαίσια, θα σας είναι ιδιαίτερα χρήσιμο, να αφιερώσετε ορισμένο χρόνο ώστε να αυτοματοποιήσετε κάποιες διαδικασίες. Για παράδειγμα, γράφοντας ένα script για αυτοματοποίηση ορισμένων εργασιών στη γραμμή εντολών (ένα bash shell script) ή ένα Makefile, μπορείτε να κερδίσετε πολύ χρόνο.

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

Βήμα 1ο. Εκτέλεση SPEC CPU2006 Benchmarks στον gem5

Στο δεύτερο μέρος της εργασίας θα ασχοληθείτε με την εκτέλεση μιας σειράς από benchmarks στον gem5. Για τα benchmarks θα χρησιμοποιήσετε ένα υποσύνολο των SPEC cpu2006 benchmarks. Μπορείτε να αναζητήσετε πληροφορίες για τα SPEC cpu2006 benchmarks στον ακόλουθο σύνδεσμο:

https://www.spec.org/cpu2006/

Για την συγκεκριμένη εργασία, έχουμε διαλέξει και προετοιμάσει για εσάς ένα υποσύνολο από τα benchmarks αυτά. Ακολουθείστε τον σύνδεσμο αυτό για να κατεβάσετε τα benchmarks που θα χρησιμοποιήσετε:

http://kition.mhl.tuc.gr:8000/f/72c698c5d1/?raw=1

Εφόσον δουλεύετε σε περιβάλλον Linux για να αποσυμπιέσετε το αρχείο που κατεβάσατε (spec_cpu2006_gem5.tar.gz), τρέξτε την ακόλουθη εντολή:

```
$ tar -xzvf spec_cpu2006_gem5.tar.gz
```

Μέσα στο φάκελο spec_cpu2006 θα βρείτε πέντε φακέλους με τον καθένα να περιέχει ένα benchmark. Στο φάκελο data βρίσκονται τυχόν αρχεία δεδομένων που απαιτεί το benchmark ενώ στον φάκελο src θα βρείτε τους κώδικες και τα makefile που απαιτούνται για να κάνετε compile τους κώδικες και να μπορέσετε να τους χρησιμοποιήσετε στον gem5. Βασική προϋπόθεση είναι να έχετε εκτελέσει τα βήματα του πρώτου μέρους – 1η εργαστηριακή άσκηση – ώστε να έχετε εγκαταστήσει τους ARM cross compilers στο σύστημα που δουλεύετε. Για να γίνει το compile, απλά μπείτε στο σχετικό φάκελο που έχει τους κώδικες που θέλετε και πληκτρολογήστε `make`. Μπορείτε με ασφάλεια να αγνοήσετε κάποια warnings που παρουσιάζονται σε ορισμένους από τους κώδικες.

Το επόμενο βήμα είναι να εκτελέσετε τα benchmarks στον gem5 χρησιμοποιώντας το configuration script `se.py` που είχατε χρησιμοποιήσει και στην πρώτη εργαστηριακή άσκηση. Καθότι η εκτέλεση των benchmarks στον gem5, αν αυτά εκτελεστούν πλήρως, απαιτεί πάρα πολύ ώρα, θα κάνετε χρήση μιας επιλογής του script που σας επιτρέπει να εκτελέσετε ένα πρόγραμμα μέχρις ότου εκτελεστεί ένας αριθμός από εντολές.

Κάνοντας την βασική υπόθεση ότι έχετε τοποθετήσει το φάκελο spec_cpu2006 μέσα στον φάκελο gem5 που έχετε τον εξομοιωτή σας, εκτελέσετε τα benchmarks χρησιμοποιώντας τις εντολές που παρατίθενται παρακάτω. Με την παράμετρο `-d` ορίζετε το φάκελο στον οποίο θέλετε να αποθηκευτούν τα αποτελέσματα, με την παράμετρο `-c` δίνετε το εκτελέσιμο πρόγραμμα που είναι να τρέξει μέσα στον gem5, με την παράμετρο `-o` δηλώνετε τα ορίσματα που αυτό το πρόγραμμα απαιτεί και τέλος με την παράμετρο `-I` ορίζετε το πλήθος των εντολών του προγράμματος που θα τρέξει ο gem5 (μόλις φτάσει αυτόν τον αριθμό θα τερματίσει την εκτέλεση του). Στις εντολές που παρατίθενται παρακάτω δηλώνεται επίσης ότι θα χρησιμοποιήσετε το minorCPU μοντέλο με caches που συμπεριλαμβάνουν και caches δευτέρου επιπέδου (L2).

```
$ ./build/ARM/gem5.opt -d spec_results/specbzip configs/example/se.py --cpu-
type=MinorCPU --caches --l2cache -c spec_cpu2006/401.bzip2/src/specbzip -o
"spec_cpu2006/401.bzip2/data/input.program 10" -I 100000000


$ ./build/ARM/gem5.opt -d spec_results/specmcf configs/example/se.py --cpu-
type=MinorCPU --caches --l2cache -c spec_cpu2006/429.mcf/src/specmcf -o
"spec_cpu2006/429.mcf/data/inp.in" -I 100000000


$ ./build/ARM/gem5.opt -d spec_results/spechmmer configs/example/se.py --cpu-
type=MinorCPU --caches --l2cache -c spec_cpu2006/456.hmmer/src/spechmmer -o "--
fixed 0 --mean 325 --num 45000 --sd 200 --seed 0
spec_cpu2006/456.hmmer/data/bombesin.hmm" -I 100000000


$ ./build/ARM/gem5.opt -d spec_results/specsjeng configs/example/se.py --cpu-
type=MinorCPU --caches --l2cache -c spec_cpu2006/458.sjeng/src/specsjeng -o
"spec_cpu2006/458.sjeng/data/test.txt" -I 100000000


$ ./build/ARM/gem5.opt -d spec_results/speclibm configs/example/se.py --cpu-
type=MinorCPU --caches --l2cache -c spec_cpu2006/470.lbm/src/speclibm -o "20
spec_cpu2006/470.lbm/data/lbm.in 0 1
spec_cpu2006/470.lbm/data/100_100_130_cf_a.of" -I 100000000
```

Τα αποτελέσματα από τις διαδοχικές αυτές εκτελέσεις του gem5 (δηλαδή τα αρχεία εξόδου stats.txt, config.ini, config.json κτλ που προκύπτουν από κάθε εκτέλεση) θα τοποθετηθούν στους σχετικούς φακέλους που ορίζονται από την παράμετρο -d.

Σημείωση: Σε περίπτωση που δε χρησιμοποιήσετε Ubuntu 19.10 θα δημιουργηθεί inconsistency στο 3ο benchmark, οπότε επικοινωνήστε με τους βοηθούς εργαστηρίου για να σας δώσουν τις οδηγίες που χρειάζεστε ανάλογα με το distribution σας.

Ερωτήματα

1. Χρησιμοποιήστε τις γνώσεις σας από το πρώτο εργαστήριο και βρείτε στα σχετικά αρχεία τις βασικές παραμέτρους για τον επεξεργαστή που εξομοιώνει ο gem5 όσον αφορά το υποσύστημα μνήμης. Πιο συγκεκριμένα, βρείτε τα μεγέθη των caches (L1 instruction και L1 Data caches καθώς και της L2 cache), το associativity κάθε μίας από αυτές και το μέγεθος της cache line.
2. Καταγράψτε τα αποτελέσματα από τα διαφορετικά benchmarks. Συγκεκριμένα κρατείστε τις ακόλουθες πληροφορίες από κάθε benchmark: (i) χρόνο εκτέλεσης (προσοχή! Το χρόνο που απαιτεί το πρόγραμμα να τρέξει στον εξομοιούμενο επεξεργαστή, όχι τον χρόνο που χρειάζεται ο gem5 να πραγματοποιήσει την εξομοίωση), (ii) CPI (cycles per instruction) και (iii) συνολικά miss rates για την L1 Data cache, L1 Instruction cache και L2 cache. Τις πληροφορίες αυτές μπορείτε να τις αντλήσετε από τα αρχεία stats.txt (Hint: για το πρώτο βρείτε την τιμή sim_seconds και για το τρίτο αναζητήστε εγγραφές σαν αυτή: icache.overall_miss_rate::total). Φτιάξτε γραφήματα που να απεικονίζουν αυτές τις πληροφορίες για το σύνολο των benchmarks. Τι παρατηρείτε;
3. Τρέξτε ξανά τα benchmarks στον gem5 με τον ίδιο τρόπο με προηγουμένως αλλά αυτή τη φορά προσθέστε και την παράμετρο `--cpu-clock=1GHz` και `--cpu-clock=3GHz`. Δείτε τα αρχεία

stats.txt από τις τρεις εκτελέσεις του προγράμματος (την αρχική σας και αυτή με το 1GHz και το 3GHz) και εντοπίστε τις πληροφορίες για το ρολόι. Θα βρείτε δύο εισαγωγές: μία για system.clk_domain.clock και μία για cpu_cluster.clk_domain.clock. Μπορείτε να εξηγήσετε τελικά τί χρονίζεται στο 1GHz/3GHz και τί χρονίζεται στα default GHz; Γιατί πιστεύετε συμβαίνει αυτό? Ανατρέξτε στο αρχείο config.json που αντιστοιχεί στο σύστημα με το 1GHz. Αναζητώντας πληροφορίες για το ρολόι, μπορείτε να δώσετε μια πιο σαφή απάντηση; Αν προσθέσουμε άλλον έναν επεξεργαστή, ποια εικάζετε ότι θα είναι η συχνότητα του; Παρατηρείστε τους χρόνους εκτέλεσης των benchmarks για τα συστήματα με διαφορετικό ρολόι. Υπάρχει τέλειο scaling; Μπορείτε να δώσετε μια εξήγηση αν δεν υπάρχει τέλειο scaling;

4. Τρέξτε ξανά ένα benchmarks το οποίο θα επιλέξετε εσείς στον gem5 με τον ίδιο τρόπο με προηγουμένως αλλά αυτή τη φορά αλλάξτε το memory configuration από DDR3_1600_x64 στο DDR3_2133_x64 (DDR3 με πιο γρήγορο clock) και . Τι παρατηρείτε? Εξηγήστε τα ευρήματά σας.

Βήμα 2ο. Design Exploration – Βελτιστοποίηση απόδοσης

Οι παράμετροι και η οργάνωση του υποσυστήματος μνήμης παίζουν σημαντικό ρόλο στην απόδοση του συστήματος. Θεωρώντας ότι ο επεξεργαστής στο minorCPU μοντέλο μπορεί να εκτελεί μία εντολή ανά κύκλο, CPI μεγαλύτερο του ένα μας δείχνει ότι ο επεξεργαστής αργεί να πάρει εντολές και δεδομένα από τις μνήμες (caches και κύρια μνήμη). Όσο περισσότερα είναι τα misses που παρουσιάζονται στις caches κατά την εκτέλεση ενός προγράμματος, τόσο μεγαλύτερο αρνητικό αντίκτυπο θα έχει στον χρόνο εκτέλεσης του προγράμματος.

Προσπαθήστε να βρείτε με ποιες τιμές από τις παρακάτω παραμέτρους μπορείτε να πετύχετε τη μέγιστη απόδοση στο σύστημα σας για κάθε benchmark:

- L1 instruction cache size
- L1 instruction cache associativity
- L1 data cache size
- L1 data cache associativity
- L2 cache size
- L2 cache associativity
- Μέγεθος cache line

Οι παράμετροι αυτοί μπορούν να δοθούν κατά την κλήση του gem5 ως options του configuration script se.py. Εδώ φαίνεται πώς μπορείτε να πραγματοποιήσετε μια σχετική εκτέλεση με κάποιες τυχαίες τιμές για κάθε παράμετρο:

```
$ ./build/ARM/gem5.opt -d spec_results/speclibm configs/example/se.py --cpu-
type=MinorCPU --caches --l2cache --l1d_size=32kB --l1i_size=64kB --l2_size=512kB
--l1i_assoc=1 --l1d_assoc=1 --l2_assoc=2 --cacheline_size=64 --cpu-clock=1GHz -c
spec_cpu2006/470.lbm/src/speclibm -o "20 spec_cpu2006/470.lbm/data/lbm.in 0 1
spec_cpu2006/470.lbm/data/100_100_130_cf_a.of" -I 100000000
```

Ως μέγιστη απόδοση θεωρούμε το ελάχιστο CPI (όσο πιο κοντά στο 1 μπορεί να φτάσει). Θεωρήστε ότι δεν μπορείτε να ξεπεράσετε τα 256KB ως συνολικό μέγεθος L1 cache (δηλαδή άθροισμα L1 data και L1 instruction – υπόψη αυτά δεν χρειάζεται να είναι ίδια) και τα 4MB ως συνολικό μέγεθος L2 cache.

Ερωτήματα

1. Όπως καταλαβαίνετε ο χώρος των πιθανών συνδυασμών είναι πολύ μεγάλος. Χρησιμοποιήστε τα αποτελέσματα από το πρώτο Βήμα της άσκησης καθώς και τις γνώσεις ή πιθανή μελέτη των benchmarks ώστε να προσπαθήσετε να περιορίσετε τις δοκιμές που θα χρειαστεί να κάνετε. Αιτιολογήστε τις απαντήσεις σας.
2. Παρουσιάστε γραφήματα που δείχνουν την επίδραση κάθε παράγοντα στην απόδοση κάθε benchmark. Προσπαθήστε να εξηγήσετε τα αποτελέσματα που έχουν προκύψει.

Προσπαθήστε να αυτοματοποιήσετε όσο περισσότερο γίνεται τη διαδικασία για να κερδίσετε χρόνο. Χρησιμοποιήστε κάποιο bash script για να βάλετε όλες τις εντολές μέσα που θέλετε και να τις εκτελέσετε όλες μαζί. Για να σας βοηθήσουμε στη συλλογή και επεξεργασία των αποτελεσμάτων, μπορείτε να κατεβάσετε το ακόλουθο script (read_results.sh):

http://kition.mhl.tuc.gr:8000/f/31dfda3055/?raw=1

Για να το τρέξετε χρειάζεται να προσδιορίσετε κάποιο αρχείο που περιγράφει αυτό που θέλετε να παρατηρήσετε. Ας υποθέσουμε ότι για κάθε εκτέλεση του benchmark 401.bzip2 έχετε σώσει τα αποτελέσματα στους φακέλους spec_bzip2_0, spec_bzip2_1, …, spec_bzip2_5 , ότι θέλετε να παρατηρήσετε τις τιμές system.cpu.cpi, system.cpu.dcache.overall_miss_rate::total,

```
[Benchmarks]
spec_bzip2_0
spec_bzip2_1
spec_bzip2_2
spec_bzip2_3
spec_bzip2_4
spec_bzip2_5

[Parameters]
system.cpu.cpi
system.cpu.dcache.overall_miss_rate::total
system.cpu.icache.overall_miss_rate::total
system.l2.overall_miss_rate::total

[Output]
Results_bzip2.txt
```

system.cpu.icache.overall_miss_rate::total και system.l2.overall_miss_rate::total από τα σχετικά αρχεία stats.txt και τα αποτελέσματα θα θέλατε να αποθηκευτούν στο αρχείο results_bzip2.txt. Τότε μπορείτε να γράψετε το ακόλουθο αρχείο conf_script.ini ως εξής:

Φυσικά μπορείτε να βάλετε όσα διαφορετικά benchmark runs θέλετε μέσα. Στη γραμμή εντολών δώστε την ακόλουθη εντολή:

```
$  bash read_results.sh conf_script.ini
```

Το αρχείο results_bzip2.txt που θα παραχθεί θα έχει τη μορφή (οι αριθμοί εδώ είναι τυχαίοι):

```
Benchmarks  system.cpu.cpi  system.cpu.dcache.overall_miss_rate::total
      system.cpu.icache.overall_miss_rate::total
      system.l2.overall_miss_rate::total

Spec_bzip2_0      1.829176    0.008526    0.000483    0.548917

Spec_bzip2_1      1.802639    0.006697    0.000460    0.718968

Spec_bzip2_2      1.802333    0.006697    0.000134    0.725691

Spec_bzip2_3      1.801962    0.006695    0.000114    0.726001

Spec_bzip2_4      1.790316    0.005833    0.000112    0.848716

Spec_bzip2_5      1.790161    0.005829    0.000100    0.849015
```

Βήμα 3ο. Κόστος απόδοσης και βελτιστοποίηση κόστους/απόδοσης

Από τις γνώσεις σας σχετικά με τη σχεδίαση ψηφιακών κυκλωμάτων καταλαβαίνετε εύκολα ότι μια μνήμη μεγέθους A έχει μεγαλύτερο κόστος υλοποίησης σε σχέση με μια μνήμη μεγέθους B με B < A. Αντίστοιχα γνωρίζουμε ότι μια μνήμη μεγάλου μεγέθους είναι πιο αργή από μια μνήμη μικρού μεγέθους και εκεί βασίζεται η λογική των caches και της ιεραρχίας μνημών.

Προσπαθήστε λοιπόν (χρησιμοποιώντας και τη βιβλιογραφία) να εκφράσετε με μια συνάρτηση κόστους την επίδραση των παραγόντων που «πειράξατε» στο Βήμα 2 όσον αφορά το μέγεθος του κυκλώματος και την ταχύτητα. Προσπαθήστε δηλαδή να καταδείξετε τί πρέπει να «πληρώσουμε» για την κάθε επιλογή. Θεωρείστε μια αυθαίρετη μονάδα κόστους.

(Hints: η «τιμή» που πρέπει να πληρώσουμε για μια L1 cache μνήμη είναι αρκετά μεγαλύτερη από αυτό που πληρώνουμε για μια L2. Αντίστοιχα, αυξάνοντας το associativity αυξάνουμε την πολυπλοκότητα, κ.ο.κ.)

[Σημαντική Σημείωση για την αξιολόγηση της προσπάθειας σας: Δεν υπάρχει μια *σωστή* απάντηση με τη μορφή μιας συγκεκριμένης συνάρτησης που έχουμε υπολογίσει και θέλουμε να τη βρείτε. Αυτό που ζητάμε είναι να μπορέσετε να μας δείξετε ότι έχετε μια ποιοτική αντίληψη του κόστους που έχει

κάθε επιλογή, να κάνετε μια επαρκή έρευνα στη βιβλιογραφία για να στηρίξετε όσο μπορείτε πιο ρεαλιστικά την υπόθεση σας και τέλος να προσπαθήσετε με κάποιο τρόπο να της δώσετε κάποια ποσοτικά χαρακτηριστικά που θα σας βοηθήσουν να πάρετε σχεδιαστικές αποφάσεις. Όπως κάνουν ακριβώς και οι αρχιτέκτονες των σύγχρονων επεξεργαστών. 😊]

Αφού κατασκευάσετε τη συνάρτηση κόστους σας, χρησιμοποιήστε τα αποτελέσματα από το Βήμα 2 για κάθε benchmark όσον αφορά το CPI και προσδιορίστε την αρχιτεκτονική που βελτιστοποιεί την απόδοση του συστήματος σε σχέση με το κόστος του. Σχολιάστε τους συμβιβασμούς που θεωρείτε ότι πρέπει να γίνουν και στηρίξτε τις επιλογές σας.

Παραδοτέα

Χρησιμοποιήστε τον λογαριασμό στο GitHub που δημιουργήσατε στην πρώτη άσκηση και δημιουργήστε ένα δεύτερο repository για την άσκηση αυτή. Ανεβάστε:

1. ό,τι κώδικες και αρχεία αποτελεσμάτων έχουν προκύψει από τα ερωτήματα που σας ζητούνται να απαντήσετε. Για τους κώδικες που θα ανεβάσετε απαιτείται να έχετε οπωσδήποτε επαρκή σχόλια που να επεξηγούν τι έχετε κάνει και αν αυτό απαιτείται κάποιο documentation.
2. μια αναλυτική αναφορά με τις απαντήσεις στα ερωτήματα. Η αναφορά θα είναι γραμμένη σε ένα αρχείο README.md που θα είναι στο top level του repository σας. Η αναφορά σας θα πρέπει να περιλαμβάνει <u>οπωσδήποτε</u> τις πηγές που χρησιμοποιήσατε για να ολοκληρώσετε την εργασία σας.

<u>ΣΗΜΑΝΤΙΚΟ</u>: Όταν ολοκληρώσετε επιτυχώς την εργασία σας, μπορείτε να πάρετε έναν έξτρα βαθμό, γράφοντας μια κριτική για την εργασία αυτή. Επικεντρωθείτε στο αν μάθατε κάτι, αν τη θεωρείτε απλοϊκή ή δύσκολη, αν υπήρχε κάτι που σας δυσκόλεψε χωρίς λόγο και χωρίς να είναι σχετικό με την εργασία σας, κάποια συμβουλή κτλ. Βάλτε αυτή την κριτική μέσα στην αναφορά σας που είναι στο README.md

<u>Για την παράδοση της άσκησης σας, θα κάνετε submit μέσω του elearning την διεύθυνση του repository σας (όχι τον κώδικα ή οτιδήποτε άλλο ΜΟΝΟ την διεύθυνση στο github)</u>.

Να έχετε υπόψη σας τα εξής:

1. εφόσον όλα τα repositories είναι public, όλοι μπορούν να δουν το κώδικα σας και την εργασία σας. Αυτό σημαίνει ότι θα πρέπει να φροντίσετε να είναι προσεγμένα (θα κριθείτε και για την ποιότητα της εργασίας όσον αφορά την παρουσίαση της)
2. με την ίδια λογική εφόσον όλα είναι public γίνεται προφανές ότι είναι εξαιρετικά εύκολο να αναδειχθούν οι αντιγραφές. Καλό θα ήταν λοιπόν να το λάβετε αυτό υπόψη σας.
3. Η δεύτερη αυτή άσκηση είναι εξαιρετικά απλή στην υλοποίηση αλλά όμως αρκετά απαιτητική όσον αφορά το χρόνο εκτέλεσης. Το σημαντικότερο όμως για την άσκηση αυτή είναι ότι απαιτεί τη κριτική σας σκέψη και την εφαρμογή στη πράξη της θεωρίας που έχετε διδαχθεί καθώς και την ποιοτική κατανόηση όσων αποτελεσμάτων προκύπτουν από την εκτέλεση της άσκησης. Στην εργασία αυτή θα πρέπει να στηρίξετε τις παρατηρήσεις και τις επιλογές σας. Για να το κάνετε αυτό και για να εξάγετε συμπεράσματα που μπορούν να γίνουν αποδεκτά, έχετε δύο τρόπους (και πρέπει να τους χρησιμοποιήσετε και τους δύο): (i) τα πειραματικά αποτελέσματα και (ii) τη

βιβλιογραφία. Έτσι κάθε θέση που διατυπώνετε, θα πρέπει να μπορεί να στηριχθεί είτε με τον έναν είτε με τον άλλο τρόπο.

Για απορίες σε σχέση με τον GEM5 και την εκτέλεση των benchmarks μπορείτε να στέλνετε email στον κ. Καρανάσσο Δημήτρη (dkaranassos@ece.auth.gr). ΣΗΜΑΝΤΙΚΟ: μη στέλνετε τις ολοκληρωμένες εργασίες σας μέσω email βάλτε τες στο repository σας στο github και θα ενημερωθείτε μέσω ανακοίνωσης που να τις καταθέσετε.

ΚΑΛΗ ΕΠΙΤΥΧΙΑ