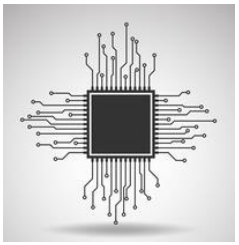
Εθνικο & Καποδιστριακο Πανεπιστημιο Αθηνων
Τμημα Πληροφορικης & Τηλεπικοινωνιων
Μαθημα: **Αρχιτεκτονικη Υπολογιστων ΙΙ**



Εργασία 2 (υποχρεωτική) – Κρυφές Μνήμες

Ακαδημαϊκο Ετος 2023 – 2024
(εκφωνηση) Δευτερα 18 Δεκεμβριου 2023
(παραδοση στο eclass μεχρι) **Παρασκευη 19 Ιανουαριου 2024**

Επώνυμο	Όνομα	Αριθμός Μητρώου	Email
Νταής	Παύλος	sdi2100122	pavlosdais@gmail.com
Αρκουλής	Κωνσταντίνος	sdi2100008	arkoulis.kostas02@gmail.com

Πληροφορίες για τις Υποχρεωτικές Εργασίες του μαθήματος

- Οι υποχρεωτικές εργασίες του μαθήματος είναι **δύο**. Σκοπός τους είναι η κατανόηση των εννοιών του μαθήματος με χρήση αρχιτεκτονικών προσομοιωτών. Η πρώτη υποχρεωτική εργασία αφορά τη διοχέτευση (pipelining) και η δεύτερη (αυτή) αφορά τις κρυφές μνήμες (cache memories).
- Οι δύο εργασίες είναι υποχρεωτικές και η βαθμολογία του μαθήματος θα προκύπτει από το γραπτό (60%), την εργασία της διοχέτευσης (20%), και την εργασία των κρυφών μνημών (20%).
- Κάθε ομάδα μπορεί να αποτελείται **από 1 έως και 3 φοιτητές** (**η υποβολή να γίνει μόνο από έναν φοιτητή εκ μέρους όλης της ομάδας – μην κάνετε υποβολές όλοι**). Συμπληρώστε τα στοιχεία όλων των μελών της ομάδας στον παραπάνω πίνακα. Όλα τα μέλη της ομάδας πρέπει να έχουν ισότιμη συμμετοχή και να γνωρίζουν τις λεπτομέρειες της υλοποίησης.
- Για την εξεταστική Σεπτεμβρίου δεν θα δοθούν άλλες εργασίες. Το Σεπτέμβριο εξετάζεται μόνο το γραπτό.
- Σε περίπτωση αντιγραφής θα μηδενίζονται όλες οι ομάδες που μετέχουν σε αυτή.
- Η παράδοση της **Εργασίας Κρυφών Μνημών** πρέπει να γίνει **ηλεκτρονικά μέχρι τα μεσάνυχτα της προθεσμίας και μόνο στο eclass** (να ανεβάσετε ένα μόνο αρχείο zip ή rar με την τεκμηρίωσή σας σε PDF και τους κώδικές σας). **Μην περιμένετε μέχρι την τελευταία στιγμή – κάθε εργασία απαιτεί τον χρόνο της**.

Ζητούμενο

Το ζητούμενο της εργασίας είναι η **βέλτιστη κατανάλωση ενέργειας** για συγκεκριμένη υπολογιστική εργασία. Η σχεδίαση του υλικού και του λογισμικού και η αξιολόγηση θα γίνει στον προσομοιωτή QtMips.

Η υπολογιστική εργασία είναι **ο υπολογισμός της πράξης X=X+Y*Z μεταξύ τριών τετραγωνικών πινάκων τάξης n** (δηλαδή διαστάσεων nxn). Οι αρχικοί πίνακες και το αποτέλεσμα θα βρίσκονται στο τμήμα δεδομένων του προγράμματος.

Δεδομένων τριών τετραγωνικών πινάκων ακεραίων αριθμών X, Y, και Z διαστάσεων nxn ο καθένας το πρόγραμμα να υπολογίζει την παραπάνω παράσταση. Να εξετάζεται το ενδεχόμενο υπερχείλισης κατά τη διάρκεια εκτέλεσης των πράξεων και το πρόγραμμα να τερματίζει εάν συμβεί υπερχείλιση.

Η MIPS CPU στην οποία θα εκτελείται το πρόγραμμά σας έχει σταθερή σχεδίαση στον πυρήνα της διοχέτευσης: branch predictor των 2-bit με BHT που προσπελάζεται με 5 bit και επίλυση των διακλαδώσεων στο στάδιο EX και πλήρη μονάδα κινδύνων με ανίχνευση και προώθηση. Η εκτέλεση κάθε εντολής στον επεξεργαστή δαπανά ενέργεια 1 picojoule.

Καλείσθε να ρυθμίσετε τη διαμόρφωση του συστήματος μνήμης ώστε να επιτευχθεί η βέλτιστη κατανάλωση ενέργειας. Ο επεξεργαστής διαθέτει δύο επίπεδα κρυφής μνήμης (ξεχωριστή L1 για εντολές και δεδομένα και ενιαία L2). Η προσπέλαση της κύριας μνήμης διαρκεί 35 κύκλους και η προσπέλαση της L2 διαρκεί 6 κύκλους. Οι κρυφές μνήμες L1 έχουν ίδιο συνολικό μέγεθος (καθεμία μπορεί να έχει μέγεθος s*8KB, όπου s=1 ή 2), ίδιο μέγεθος μπλοκ (που μπορεί να είναι w=2, 4 ή 8 λέξεις) και ίδια συσχετιστικότητα (που μπορεί να είναι a=1, 2 ή 4). Κάθε προσπέλαση της L1 (είτε εντολών είτε δεδομένων) δαπανά ενέργεια (s*w*0.5*a)/4 picojoule. Η κρυφή μνήμη L2 έχει μέγεθος 64KB και έχει ίδιο μέγεθος μπλοκ και συσχετιστικότητα με τις L1 caches. Κάθε προσπέλαση της L2 δαπανά ενέργεια 2*w*a picojoule. Κάθε προσπέλαση στην κύρια μνήμη δαπανά ενέργεια 320 picojoule. Σε όλες τις μνήμες η πολιτική εγγραφής είναι ετερόχρονη (write back) και με κατανομή σε εγγραφή (write allocate) και η αντικατάσταση LRU.

Ο ρυθμός ρολογιού του επεξεργαστή παραμένει σταθερός για οποιαδήποτε από τις παραπάνω διαμορφώσεις.

Να υπολογίσετε τη **συνολική ενέργεια που δαπανά το πρόγραμμά σας** για τιμές των διαστάσεων των πινάκων που αρχίζουν από n=8 και αυξάνονται κατά 2 (8, 10, 12, …) μέχρι τον μεγαλύτερο αριθμό που επιθυμείτε. Προσπαθήστε να συγγράψετε τον κώδικά σας και να ρυθμίσετε τις παραμέτρους των κρυφών μνημών **ώστε η συνολική δαπανώμενη ενέργεια να είναι η μικρότερη δυνατή**. Αν θέλετε, μπορείτε να συγγράψετε διαφορετικό κώδικα για διαφορετικές τιμές του n (στην περίπτωση αυτή να παραδώσετε όλα τα διαφορετικά προγράμματα).

Να συμπληρώσετε τον παρακάτω πίνακα με **τις σχεδιαστικές αποφάσεις σας** για τις κρυφές μνήμες.

Χαρακτηριστικό	**Τιμές**
L1 caches μέγεθος καθεμίας (8KB ή 16KB – τιμή του s)	
L1 caches, L2 cache (μέγεθος μπλοκ – τιμή του w, συσχετιστικότητα – τιμή του a)	

Να συμπληρώσετε τον παρακάτω πίνακα (ή/και να δώσετε σχετικά διαγράμματα) με **τα αποτελέσματά σας για τις διάφορες τιμές του n που δοκιμάσατε**.

n (διάσταση πινάκων)	**Χρόνος εκτέλεσης (κύκλοι)**	**Ενέργεια (picojoules)**
8		
10		
12		
14		
16		
18		
…		
…		

Τεκμηρίωση

[Σύντομη τεκμηρίωση της λύσης σας ενδεικτικά (όχι αυστηρά) μέχρι **10 σελίδες ξεκινώντας από την επόμενη σελίδα** – μην αλλάζετε τη μορφοποίηση του κειμένου (**και παραδώστε την τεκμηρίωση σε αρχείο PDF**). Η τεκμηρίωσή σας πρέπει να περιλαμβάνει παραδείγματα ορθής εκτέλεσης και σχολιασμό για την επίλυση του προβλήματος και την επίτευξη του ζητούμενου. Μπορείτε να χρησιμοποιήσετε εικόνες, διαγράμματα και ό,τι άλλο μπορεί να βοηθήσει στην εξήγηση της δουλειάς σας.]

Παραδοτέο

- **Report.pdf**: η αναφορά της εργασίας αυτής.
- **matrix_mul8p.s**: το assembly πρόγραμμα που ζητείται, για μεγέθη πινάκων από 8 μέχρι 26 (8+).
- **matrix_mul28p.s**: το assembly πρόγραμμα που ζητείται, για μεγέθη πινάκων από 28 μέχρι 32 καθώς και 64 (28+).
- **generetor.c**: ένα script σε C που δημιουργήθηκε για την παραγωγή datasets (καθώς και την λύση τους) κατάλληλων μεγεθών σαν input στα προγράμματά μας.
- **arrays8+**: Φάκελος με τα δεδομένα εισόδου και των αναμενόμενων αποτελεσμάτων εξόδου, για μεγέθη πινάκων από 8 μέχρι 26, που προέκυψαν από την εκτέλεση του script.
- **arrays26+**: Φάκελος με τα δεδομένα εισόδου και των αναμενόμενων αποτελεσμάτων εξόδου, για μεγέθη πινάκων από 28 μέχρι 32 καθώς και 64, που προέκυψαν από την εκτέλεση του script.
- **Measurements.xlsm**: Οι μετρήσεις μας με διάφορες επιλογές associativity (**a**), μέγεθος block (**w**) και μέγεθος cache (**s**) για τα διάφορα μεγέθη πινάκων του dataset μας.

Γενικές Παρατηρήσεις

1. Ο τύπος της κατανάλωσης της ενέργειας προκύπτει από τον παρακάτω τύπο:

$$E = \frac{s \cdot w \cdot a \#(\text{L1 hits})}{8} + 2 \cdot w \cdot a \cdot \#(\text{L2 hits}) + 320 \cdot \#(\text{DRAM/L2 misses}) + \#(\text{Instructions})$$
$$= w \cdot a \left(\frac{s \cdot \#(\text{L1 hits})}{8} + 2 \cdot \#(\text{L2 hits}) \right) + 320 \cdot \#(\text{DRAM/L2 misses}) + \#(\text{Instructions})$$

Όπου, **E** η συνολική ενέργεια που καταναλώνει το πρόγραμμα σε picojoules, **a** το degree of associativity, **w** το μέγεθος block σε πλήθος words, **s** ο πολλαπλασιαστικός παράγοντας της χωρητικότητας της *L1* μνήμης. Ως προσπέλαση στην κύρια μνήμη, ορίζουμε τα *L2* misses. Ο τύπος αυτός προκύπτει άμεσα από τις οδηγίες της εκφώνησης και από μετέπειτα διευκρινήσεις του διδάσκοντα.

2. Ο *generator.c* παράγει τους μονοδιάστατους πίνακες *X*, *Y*, *Z* τους οποίους και τυπώνουμε στο αρχείο arrays.txt. Έχουμε επιλέξει η αναπαράσταση των πινάκων να είναι με μονοδιάστατο τρόπο στον *generator* καθώς με αυτό τον τρόπο μπορούμε με ένα max size πίνακα να παίρνουμε υποπίνακες του πολύ εύκολα, απλά αλλάζοντας το size του πίνακα (tag **N** του data segment) στο πρόγραμμα assembly μας, λαμβάνοντας κιόλας υπόψη τον σειριακό τρόπο με τον οποίο παίρνουμε τους input πίνακες από το data segment. Στη συνέχεια στον *generator*, για κάθε μέγεθος πίνακα, παίρνουμε τους αντίστοιχους 2-διάστατους υποπίνακες από τους 1-διάστατους *X*, *Y*, *Z*, γίνεται η πράξη και γράφουμε το αποτέλεσμα σε 16-δική μορφή, στο αντίστοιχο αρχείο.

Για την σύγκριση του τελικού αποτελέσματος και τον έλεγχο της ορθότητας του προγράμματος μας, αρκεί ο χρήστης να ελέγξει και να συγκρίνει τις τιμές του data segment από την διεύθυνση 0x100000 και μετά, με το αντίστοιχο (ως προς το N) solution_arrayN.txt file. Σημαντικό είναι να τονίσουμε ότι σε κάθε περίπτωση του N, το τελευταίο στοιχείο του πίνακα αποτελέσματος Z βρίσκεται στην διεύθυνση 0x100000 + N*N*4 – 1, τα στοιχεία παρακάτω (για τις περιπτώσεις όπου N != 32 στο matrix_mul8p.s και N != 64 στο matrix_mul28p.s, περισσότερα για αυτά παρακάτω) είναι τα στοιχεία που έχει επιπλέον ο αρχικός πίνακας λόγω της σειριακής διαχείρισης που κάνουμε και εξηγήσαμε μόλις και **όχι** exceeding of boundaries του αναμενόμενου πίνακα Z λόγω του προγράμματός μας.

Έχουμε κάνει παραμετρικό το πλήθος από πίνακες που θα παράγει ο *generator* καθώς και το μέγεθος τους, αρκεί να γίνουν οι κατάλληλες αλλαγές στις defined τιμές SIZE, END_SIZE, STEP στην αρχή της γεννήτριας. Επιπλέον,

έχει επιλεχθεί να μην αρχικοποιείται η γεννήτρια (srand) για να διατηρούμε ίδιους τους πίνακες μας για τις μετρήσεις μας σε κάθε εκτέλεση του generator. Περισσότερα στον κώδικα και παρακάτω.

Για τον υπολογισμό των # sets σε κάθε cache ανάλογα το configuration της, κινηθήκαμε με την γνωστή μεθοδολογία που έχουμε πει και στο μάθημα, δηλαδή εφαρμόσαμε απλά τον τύπο:

sets_of_cache = (size_of_cache) / (size_of_set) = (size_of_cache) / (**a** * block_size_of_cache)

= (size_of_cache) / (**a** * **w** * size_of_word)

Μάλιστα, φτιάξαμε ένα απλό scriptακι το οποίο μας βοήθησε σε αυτόν τον υπολογισμό που γίνεται από μια φορά για όλα τα configurations αλλά δεν θεωρήσαμε σχετικό ή άξιο το εν λόγω script οπότε και δεν τοποθετήθηκε στο παραδοτέο.

3. Παραθέτουμε δύο προγράμματα, το πρώτο (***matrix_mul8p.s***) αφορά arrays μεγέθους {8, 10, 12, 14, 16, 18, 20, 22, 24, 26} και το δεύτερο (***matrix_mul28p.s***) arrays μεγέθους {28, 30, 32, 64}. Αρχικά, είχαμε σκοπό να εξετάσουμε μεγέθη πινάκων μέχρι και 26. Στη συνέχεια, όμως, αποφασίσαμε πως θα ήταν καλό να προσθέσουμε και άλλα, πιο μεγάλα μεγέθη, για να αναδείξουμε την χρησιμότητα του Cache Blocking αλγορίθμου της προσέγγισης μας (περισσότερα στην αμέσως επόμενη ενότητα) καθώς παρατηρήσαμε πως έχουμε σχετικά μεγάλη cache (= 8KB ακόμα και για **s** = **1**) και οι μικροί αυτοί πίνακες χωράνε σχεδόν εξ' ολοκλήρου μέσα στη *L1*. Επομένως, για να μη πετάξουμε τις μετρήσεις μας, καθώς αυτές θα άλλαζαν αν αλλάζαμε και τους πίνακες που τροφοδοτούν το πρώτο πρόγραμμα, αποφασίσαμε να προσθέσουμε ένα νέο πρόγραμμα, δηλαδή το *matrix_mul28p.s*, με τον ίδιο ακριβώς αλγόριθμο, αλλά, με διαφορετικό data segment ως προς τους πίνακες εισόδου. Επιπλέον, παρότι που η εκφώνηση αναφέρει ρητά ότι οι διαστάσεις των πινάκων αυξάνονται με βήμα 2, επιλέξαμε, για να έχουμε και μεγάλους πίνακες στο working set μας, να παραθέσουμε μετά το n = 32, την περίπτωση n = 64, παραλείποντας τα ενδιάμεσα n για λόγους που είναι εύκολα αντιληπτοί.

4. Σχετικά με τον έλεγχο για υπερχείλιση:

- Στην *πρόσθεση* δύο στοιχείων δύο πινάκων, κρατάμε τα πρόσημα των αριθμών πριν την πρόσθεση και στην περίπτωση που οι αριθμοί έχουν ίδιο πρόσημο και στο αποτέλεσμα έχουμε διαφορετικό πρόσημο, αυτό σημαίνει ότι έχει συμβεί υπερχείλιση και το πρόγραμμα τερματίζει βίαια με τιμή στον καταχωρητή *at* = -1.

- Στον *πολλαπλασιασμό* δύο στοιχείων δύο πινάκων, κάνουμε αρχικά μια *mult* και έπειτα συγκρίνουμε τα πρόσημα των *lo* και *hi* καταχωρητών. Σε περίπτωση που τα πρόσημά τους δεν είναι ίδια, αυτό σημαίνει ότι έχει συμβεί υπερχείλιση και το πρόγραμμα τερματίζει βίαια με τιμή στον καταχωρητή *at* = -2.

Για να αποφευχθεί τυχόν overflow και άρα ο βίαιος τερματισμός του προγράμματος στις μετρήσεις μας καθώς και για την ανάδειξη της λειτουργίας του προγράμματος μας, οι τιμές με τις οποίες αρχικοποιούμε τους πίνακες βρίσκονται στο εύρος [-64, +64]. Το οποίο, όμως, μπορεί να αλλάξει ο χρήστης αν θέλει, από το αντίστοιχο macro στον *generator.c*.

Επεξήγηση Λειτουργίας Προγράμματος (Αλγόριθμος Cache Blocking)

- Ο αλγόριθμος που χρησιμοποιούμε είναι ο γνωστός *Cache Blocking (Tiling)* αλγόριθμος για Matrix Multiplication, όπως περιγράφεται και στις διαφάνειες του εργαστηρίου (διαφάνεια 193) αλλά υπάρχει και σε πολλές άλλες πηγές στο διαδίκτυο (π.χ. https://suif.stanford.edu/papers/lam-asplos91.pdf).

- Η βασική ιδέα πίσω απο το *Cache Blocking* είναι πως «σπάμε» το αρχικά μεγάλο πρόβλημα του πολλαπλασιασμού των αρχικών πινάκων σε υποπίνακες μεγέθους *block (Blocking Factor)*, οι οποίοι, όμως, αν επιλεχθούν σωστά, επιτρέπουν στο working set να χωρέσει εξ' ολοκλήρου στην cache. Επομένως, εκμεταλλευόμενοι έτσι τη τοπικότητα των references (*spatial* και *temporal*), οι τιμές που απαιτούνται για τον υπολογισμό κάποιου block (υποπροβλήματος) βρίσκονται ήδη στην cache και η προσκόμιση τους απαιτεί λίγους κύκλους ρολογιού, αυξάνοντας, έτσι, την απόδοση του αλγορίθμου. Στον αντίστοιχο κώδικα γλώσσας C που παραθέτουμε στην αρχή του προγράμματος στα σχόλια, το *block* είναι η διάσταση *block* του αλγορίθμου του προγράμματος, η οποία μπορεί μάλιστα να μεταβληθεί από τον χρήστη αλλάζοντας στο

data segment το tag **B**. Ο μόνος περιορισμός είναι ότι πρέπει να είναι ένας θετικός διαιρέτης της διάστασης του πίνακα.

- Σε αυτό το σημείο, όπως αναφερθήκαμε και λίγο προηγουμένως στο **3)** στις **Γενικές Παρατηρήσεις**, θα πρέπει να τονίσουμε ότι, ξεκινήσαμε αυτή την προσέγγιση του προβλήματος με τον συγκεκριμένο αλγόριθμο, θέλοντας να αξιοποιήσουμε την cache στο έπακρον, κάνοντας δηλαδή optimize τα memory access patterns. Όμως, προς το τέλος των μετρήσεων μας, διαπιστώσαμε πως το καλύτερο *Blocking Factor* για την απόδοση του προγράμματος και -πρωτίστως αλλά άμεσα εξαρτόμενης- της εξοικονόμησης ενέργειας, ήταν το ίδιο το μέγεθος των πινάκων της εκάστοτε περίπτωσης. Αυτό όμως, πολύ απλά, υποβαθμίζει τον αλγόριθμο μας σε απλό matrix multiplication, καθώς τα εξωτερικά loops πραγματοποιούνται από μια μόνο φορά. Ο λόγος –ο οποίος διαπιστώθηκε σε δεύτερο χρόνο- που συμβαίνει αυτό, είναι γιατί η *L1* cache ακόμα και για **s** = **1** είναι αρκετά μεγάλη (8KB) για να «χωρέσει» σχεδόν εξ' ολοκλήρου και τους τρεις πίνακες των περισσότερων περιπτώσεων που εξετάζονται στην παρούσα εργασία, δηλαδή τους πίνακες διάστασης από 8 έως 26. Οπότε, για να μην ξεκινήσουμε επί της ουσίας όλη την δουλειά μας από την αρχή, αποφασίσαμε να κρατήσουμε ότι έχουμε κάνει και να προσθέσουμε ακόμη μεγαλύτερους πίνακες (*matrix_mul28p.s* με n = 28, 30, 32, 64) με σκοπό να αναδείξουμε την χρησιμότητα του αλγορίθμου μας. Παρολ' αυτά, όπως εξάλλου είναι και το ζητούμενο αυτής της άσκησης, τονίζουμε ότι διατηρήσαμε το **ίδιο** global cache configuration (τιμές **s**, **w**, **a**) για όλα τα μεγέθη πινάκων και για όλα τα blocking factors που εξετάζουμε, το οποίο ουσιαστικά δεν αποτελεί βέλτιστη λύση για όλες τις περιπτώσεις αλλά δίνει τα καλύτερα αποτελέσματα στα περισσότερα μεγέθη πινάκων (περισσότερα στις παρακάτω ενότητες).

Τελικό Configuration-Αιτιολόγηση και Μετρήσεις

Χαρακτηριστικό	Τιμές
L1 caches μέγεθος καθεμίας (8KB ή 16KB – τιμή του s)	**s** = **1**
L1 caches, L2 cache (μέγεθος μπλοκ – τιμή του w, συσχετιστικότητα – τιμή του a)	**w** = **8**, **a** = **2**

Δηλαδή, επιλέξαμε να έχουμε:

- **2**-way associative των **8**KB L1-data Cache
- **2**-way associative των **8**KB L1-instruction Cache
- **2**-way associative των (κλειδωμένα) **64**KB L2-unified Cache

Σε όλες τις ιεραρχίες, το block size είναι σταθερό και ίσο με **8** words. Οπότε, με τους παραπάνω αριθμούς οι αριθμοί των set για την κάθε cache, είναι:

- **128** sets στην L1-data
- **128** sets στη L1-instruction
- **1024** sets στην L2-unified

Η αιτιολόγηση είναι στενά συνδεδεμένη με ότι γνωρίζουμε από την θεωρία αλλά και με τις ίδιες τις μετρήσεις μας και έχεις ως εξής:

- Αρχικά, για το **s** παρατηρήσαμε σχεδόν κατευθείαν ότι η μικρότερη *L1* που μπορούμε να έχουμε είναι μεγέθους 8KB που είναι ήδη αρκετά μεγάλη για την συντριπτική πλειοψηφία των πινάκων που εξετάζονται στην παρούσα εργασία. Επομένως, λαμβάνοτας υπόψη το βάρος που έχει το **s** στον τύπο της ενέργειας, απαλείψαμε τελείως την επιλογή του **s** = **2** οπότε και κινηθήκαμε στην τιμή **s** = **1**. Εξάλλου, όπως ήδη γνωρίζουμε από την θεωρία, αύξηση της συνολικής χωρητικότητας της κρυφής μνήμης, ενώ οδηγεί σε

μείωση του ρυθμού αστοχίας (η οποία δεν θα ήταν αισθητή για τον λόγο που είπαμε προηγουμένως), αυξάνει επίσης την κατανάλωση ισχύος.

- Επίσης, διατηρώντας σταθερές τις τιμές **s**, **w** και αυξάνοντας τον βαθμό της συσχετιστικότητας συνόλου **a** (από **1** σε **2** και από **2** σε **4**), σε γενικές γραμμές θα περιμέναμε επιτάχυνση του προγράμματος (*# CC* και *# Instructions*), όχι όμως κατ'ανάγκη και μείωση της συνολικής ενέργειας αφού, όπως ήδη γνωρίζουμε από την θεωρία, ενώ μειώνονται οι αστοχίες διένεξης, αυξάνεται η κατανάλωση ισχύος και, έτσι, θα πρέπει να ληφθεί υπόψη και η αύξηση του πολλαπλασιαστικού βάρους που δίνει το **a** στον τύπο μας για την ενέργεια. Το σκεπτικό αυτό έρχεται και μας επιβεβαιώνει πάλι ο παρακάτω πίνακας με τις μετρήσεις μας, ο οποίος αναδεικνύει τις μικρές τιμές του **s** (=**1** ή =**2**) ως τις πιο οικονομικές από πλευράς κατανάλωσης ενέργειας.

- Έπειτα, για τις διάφορες τιμές των **w** και **a** κάναμε μετρήσεις για δύο μεγέθη πινάκων, το n = 12 και το n = 24. Οι πίνακες αυτοί επιλέχθηκαν να εκπροσωπήσουν, προσεγγιστικά, τους μικρούς πίνακες και τους μεσαίους πίνακες αντίστοιχα, που εξετάζουμε στην υλοποίηση μας. Αυτό συμβαίνει γιατί, οι τιμές n = 12 και n = 24 βρίσκονται στο μέσο των περιοχών {8, 10, 12, 14, 16} και {18, 20, 22, 24, 26, 28, 30}. Τα αποτελέσματα που πήραμε είναι τα παρακάτω (βρίσκονται στο αρχείο Measurements.xlsm) όπου αναδεικνύεται ως καλύτερο cache configuration με μεγάλη διαφορά, ο συνδυασμός **(s, w, a)** = **(1, 8, 2)** με δεύτερο καλύτερο τον συνδυασμό **(s, w, a)** = **(1, 4, 1)** ο οποίος έχει πολύ κοντινή συμπεριφορά με τον **(1, 4, 2)**. Επιλέξαμε λοιπόν, για να υπάρχει κάποιου είδους σύγκριση στις υπόλοιπες μετρήσεις μας, να εξετάσουμε και τους δύο συνδυασμούς **(1, 8, 2)** και **(1, 4, 1)**. Αυτό που παρατηρήσαμε είναι ότι στην πλειοψηφία των πινάκων (για n = 8, 10, 12, 14, 16, 18, 20, 22, 24, 26) κυριάρχησε ο **(1, 8, 2)** λόγω της καλύτερης επίδοσης του έναντι του **(1, 4, 1)**.

  Στην πραγματικότητα ο **(1, 4, 1)** είναι μια *"value-for-money"* επιλογή από την άποψη ότι δεν δίνει την καλύτερη ταχύτητα στο πρόγραμμα (κάτι που φαίνεται από τα *# CC*) αλλά λόγω των χαμηλών πολλαπλασιαστικών βαρών **w**, **a** στον τύπο της ενέργειας που πολλαπλασιάζονται με τα *# (L1 hits)* και *# (L2 hits)*, προσφέρει μια εξίσου πολύ καλή επιλογή από πλευράς κατανάλωσης ενέργειας. Μάλιστα, λόγω καλύτερης αξιοποίησης της *L2* αλλά και άλλων παραγόντων που θα εξηγήσουμε καλύτερα παρακάτω, ο **(1, 4, 1)** φαίνεται να είναι ο καλύτερος συνδυασμός για μεγάλα μεγέθη πινάκων (n = 32+).

  Η αύξηση του βαθμού της συσχετιστικότητας από **a** = **1** σε **a** = **2**, και, κυρίως, του *cache block size* από **w** = **4** σε **w** = **8** words, οδηγεί σε σημαντικά καλύτερα αποτελέσματα για τους περισσότερους πίνακες, λόγω της καλύτερης ταχύτητας που προσφέρει αυτός ο συνδυασμός, διότι αυξάνεται το hit ratio της *L1* σε δραματικό βαθμό. Αυτό έχει ως αποτέλεσμα να μειωθεί αρκετά το *# CC* και μερικώς το # Instructions, καθώς και να μειωθεί σε μεγάλο βαθμό το άθροισμα *# (L2 hits) + # (L2 misses)* αφού η *L1* έχει περισσότερα hits. Ο λόγος πίσω από αυτό, είναι ότι το πρόβλημα του πολλαπλασιασμού πινάκων έχει προφανώς μεγάλη χωρική (αλλά και χρονική) τοπικότητα και, σύμφωνα και με την θεωρία μας, η αύξηση του μεγέθους του μπλοκ, μειώνει τις υποχρεωτικές αστοχίες. Έτσι, η επιλογή του **w** = **8** εκμεταλλεύεται ακόμα πιο πολύ (αποδοτικά) την τοπικότητα και δεν οδηγεί δηλαδή σε *pollution* της cache.

  Οπότε, αυτό που συμβαίνει είναι ότι ο συνδυασμός **(1, 8, 2)**, παρόλο τον σχετικά μεγάλο πολλαπλασιαστικό παράγοντα των **w**, **a** στον τύπο της ενέργειας, κάνει απόσβεση αυτών και εν τέλει, υπερτερεί έναντι όλων των άλλων συνδυασμών και σε ταχύτητα και σε ενέργεια (βλ. την από κάτω εικόνα). Οι υπόλοιπες μετρήσεις μας, δεν μας απογοήτευσαν για την διαπίστωση αυτού του συμπεράσματος, καθώς ο **(1, 8, 2)** κυριαρχεί έναντι του **(1, 4, 1)** σε όλους τους πίνακες μεγέθους {8, 10, 14, 16, 18, 20, 22, 26}.

  Εκεί που άρχισε να «χάνει» ο **(1, 8, 2)** έναντι του **(1, 4, 1)**, από πλευράς καταναλισκόμενης ενέργειας (αλλά όχι σε μεγάλο βαθμό) είναι στους πίνακες από n = 28 και πάνω (βλ Measurements.xlsm) με την μεγαλύτερη διαφορά στο μεγαλύτερο n που εξετάζουμε (= 64). Ο λόγος που συμβαίνει το τελευταίο, έχει να κάνει άμεσα με τον προκυπτόμενο τύπο της ενέργειας που αναφέραμε προηγουμένως. Συγκεκριμένα, το χαμηλό πολλαπλασιαστικό βάρος των **w**, **a** υπερτερεί στο αποτέλεσμα έναντι των τιμών *# (L1 hits)*, *# (L2 hits)* και *# (L2 misses)*. Μάλιστα, οι τιμές *# (L1 hits)* και *# (L2 misses)* επηρρεάστηκαν αρνητικά από πλευράς cache, αλλά και πάλι δεν υπερισχύσαν έναντι των βαρών **w**, **a**. Η αρνητική επιρροή που είχε η μετάβαση του configuration από **(1, 8, 2)** σε **(1, 4, 1)** στα *# (L1 hits)* έχει να κάνει για τους ίδιους λόγους που αναφέραμε προηγουμένως (κοιτώντας από την απέναντι σκοπιά). Θα πρέπει να σημειωθεί ότι με τον **(1, 4, 1)** το hit ratio της *L2* είναι εν γένει καλύτερο σχεδόν σε όλες τις μετρήσεις μας. Όμως, αυτό αρχίζει και έχει πραγματική επιρροή στην ενέργεια για τα μεγάλα μεγέθη πινάκων τα οποία, δηλαδή, δεν χωράνε πια εξ' ολοκλήρου στην *L1 cache*. Μάλιστα, με το **(1, 8, 2)** το hit ratio της *L1* λόγω και του μεγέθους της, είναι τόσο υψηλό, που πρακτικά η *L2* δεν προλαβαίνει ποτέ να ζεσταθεί αρκετά (για τα μικρομεσαία μεγέθη πινάκων) οπότε και παραμένει σχεδόν διακοσμητική στις περισσότερες περιπτώσεις, κατί που είναι αρνητικό από πλευράς της *L2-unified* cache αλλά overall τα αποτελέσματα της ταχύτητας και της ενέργειας παραμένουν όπως έχουμε ήδη πει.

s, w, a

	CC	Instructions	L1 (hits)	L2 (hits)	DRAM (L2 misses)	Total Energy
1, 2, 1	498800	406409	52129	3576	949	737425.25
1, 2, 2	485624	406409	54325	204	1093	784963.5
1, 2, 4	484976	406409	54433	27	1162	833114
1, 4, 1	492386	406340	50568	6504	477	636296
1, 4, 2	466940	406340	54809	104	548	638173
1, 4, 4	466610	406340	54864	14	583	703076
1, 8, 1	508074	406306	46632	12646	252	735914
1, 8, 2	457554	406306	55052	48	280	607546
1, 8, 4	457386	406306	55080	6	294	721090
		SIZE = 24				
		BLOCK = 24				

	CC	Instructions	L1 (hits)	L2 (hits)	DRAM (L2 misses)	Total Energy
1, 2, 1	70164	51379	6193	780	277	144687.25
1, 2, 2	67140	51379	6697	0	325	158727.5
1, 2, 4	67140	51379	6697	0	325	162076
1, 4, 1	67818	51310	5844	1452	141	110968
1, 4, 2	62058	51310	6804	0	165	110914
1, 4, 4	62058	51310	6804	0	165	117718
1, 8, 1	69844	51276	5130	2598	72	121014
1, 8, 2	59476	51276	6858	0	84	91872
1, 8, 4	59476	51276	6858	0	84	105588
		SIZE = 12				
		BLOCK = 12				

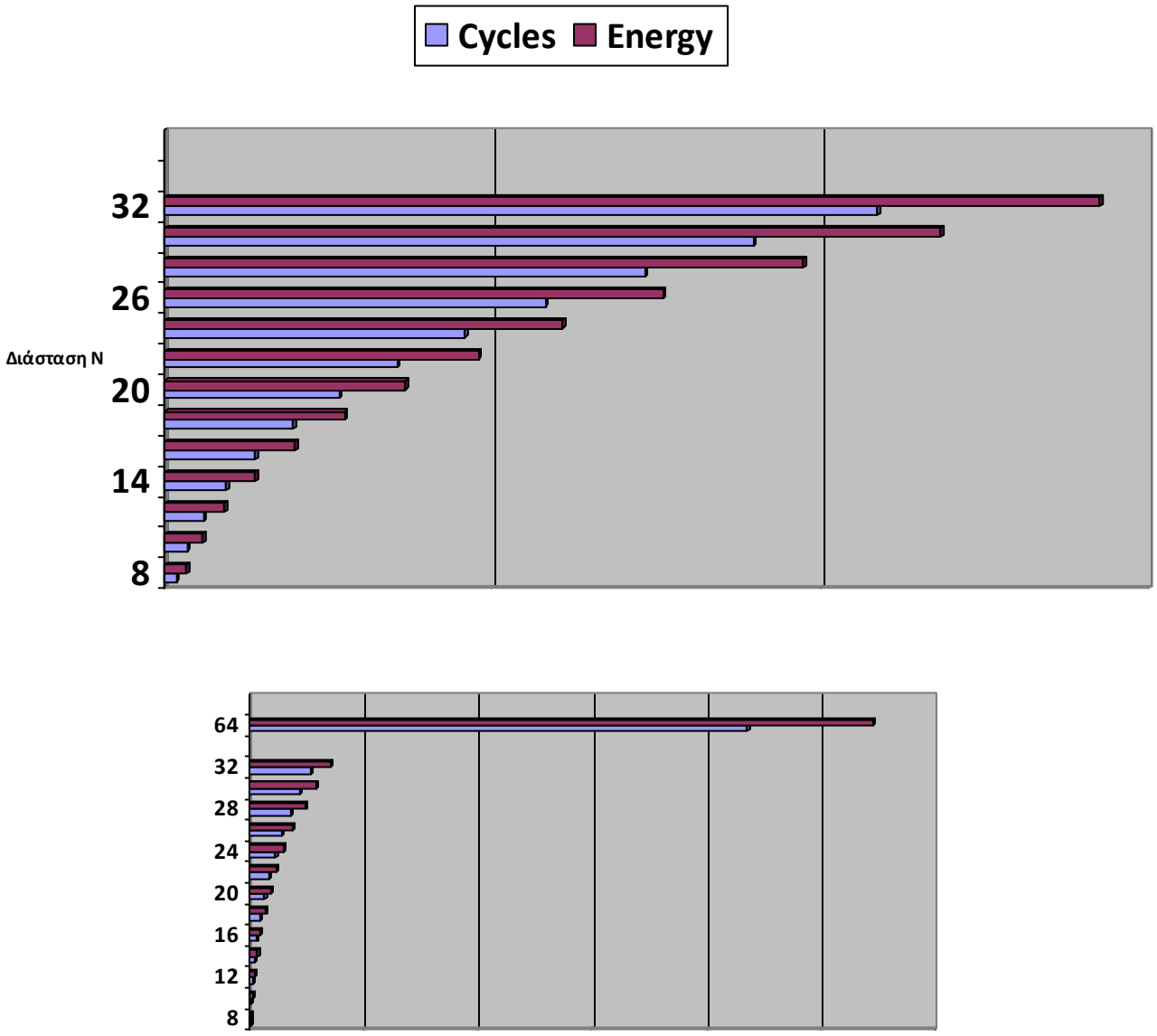
Μετρήσεις και Datasets

Οι μετρήσεις των προγραμμάτων μας βρίσκονται στο αρχείο Measurements.xlsm. Με πράσινο χρώμα (διακεκομμένου) πλαισίου είναι το βέλτιστο -από πλευράς ενέργειας- configuration για το εκάστοτε size πινάκων. Να σημειωθεί ότι για την στήλη Total Energy, έχει χρησιμοποιηθεί συνάρτηση που παίρνει τις τιμές από τις άλλες στήλες της ίδιας γραμμής και χρησιμοποιεί τον τύπο της ενέργειας που έχουμε συζητήσει, για την αυτοματοποίηση των υπολογισμών μας.

Αρχικά, όπως προαναφέραμε, κάναμε μια σύγκριση για όλα τα δυνατά CPU configurations για ένα σχετικά μικρό μέγεθος πινάκων (= 12) και για ένα σχετικά μεσαίο (ως προς τα πλαίσια αυτής της εργασίας) μέγεθος πινάκων (= 24). Το συμπέρασμα που βγάλαμε από τις μετρήσεις μας, συμφωνεί με όσα έχουμε πει νωρίτερα και αναδεικνύει τον συνδυασμό **(s, w, a)** = **(1, 8, 2)** ως τον καλύτερο από πλευράς energy efficiency για πίνακες διαστάσεων κοντά στην περιοχή του n = 8+ μέχρι και σίγουρα 26 και δίνει εξίσου ικανοποιητικά αποτελέσματα για μεγαλύτερους πίνακες (n = 28, 30, 32, 64) . Προφανώς, γνωρίζουμε ότι, δεν υπάρχει «*χρυσή συνταγή*» που να δίνει τα καλύτερα δυνατά αποτελέσματα από πλευράς εξοικονόμησης ενέργειας για όλα τα διαφορετικά μεγέθη πινάκων (κάνοντας την παραδοχή ότι διατηρείται ο αλγόριθμος ίδιος, όπως κάναμε εμείς). Επομένως, για τα υπόλοιπα μεγέθη πινάκων πέρα από τα n = 12 και n = 24, παρατίθουμε τις μετρήσεις μας για τα top 2 configurations που έχουμε προαναφέρει. Επιπλέον, στις περιπτώσεις των μεγαλύτερων μας πινάκων (32 και 64) έχουμε προσθέσει επιπλέον μετρήσεις ανάλογα και με το block size του αλγορίθμου, όπου, τώρα, δεν έχει νόημα να είναι απολύτως ίδιο με το n.

Στον παρακάτω πίνακα δίνονται μόνο οι μετρήσεις αυτές του τελικού μας configuration **(1, 8, 2)** που σε γενικές γραμμές κυριάρχησε στην κάθε περίπτωση του N και επιλέξαμε να εκπροσωπήσει αυτή την εργασία. Ακολουθούν δύο γραφήματα του πίνακα αυτού.

n (διάσταση πινάκων)	Χρόνος εκτέλεσης (κύκλοι)	Ενέργεια (picojoules)
8	18536	33512
10	35138	57932
12	59476	91872
14	93330	138388
16	137998	198136
18	195306	274254
20	266824	366380
22	354370	478870
24	457554	607546
26	580669	759420
28	731894	973332
30	897551	1183408
32	1087100	1425188
64	8722160	10931227



Συμπεράσματα-Παρατηρήσεις

- Το συμπέρασμα που αντλούμε από τα παραπάνω γραφήματα, συνάδει με όλα όσα έχουμε πει αλλά και με την διαίσθηση μας:

  Προφανώς, όσο μεγαλώνει το size (N), μεγαλώνει και ο αριθμός των συνολικών cycles για την λήξη του προγράμματος και μεγαλώνει και η συνολική καταναλισκόμενη ενέργεια, η οποία είναι άμεσα εξαρτώμενη από τα cycles. Όμως, η ενέργεια εξαρτάται και από άλλους παράγοντες τους οποίους έχουμε ήδη αναφέρει από τον τύπο της ενέργειας στο **1)** στις **Γενικές Παρατηρήσεις.** Διατηρώντας ίδιο το cache configuration **(s, w, a)** = **(1, 8, 2)** και μεταβάλλοντας μόνο το array size (N), αυτό που βλέπουμε από τις παραπάνω μετρήσεις είναι μια περίπου παραβολοειδής αύξηση δηλαδή της τάξεως ~(N^2-N^3) (καθώς το N αυξάνεται) τόσο στους συνολικούς κύκλους όσο και στην ενέργεια.

- Από τις μετρήσεις του Measurements.xlsm αλλά και γενικά από όλη την εργασία και από αυτά που μάθαμε στην πορεία για τις caches, την απόδοση τους, την κατάναλωση ισχύος και την επιρροή που έχουν σε αυτή οι διάφορες ρυθμίσεις των caches, την τοπικότητα των references κ.α., καταλήγουμε στο ίδιο συμπέρασμα που μαθαίνουμε σε πολλά προβλήματα υπολογιστικής:

  Σε όλα υπάρχει ένα trade-off και δεν υπάρχει «*χρυσή συνταγή*». Επιλέξαμε να αναδείξουμε τον συνδυασμό **(s, w, a)** = **(1, 8, 2)** ως τον καλύτερο και να παρουσίασουμε την υλοποίηση αυτή από αυτή την οπτική, γιατί ήταν ο καλύτερος στην συντριπτική πλειοψηφία των πινάκων που εξετάζουμε, **χωρίς**, δηλαδή, να εννοούμε ότι κυριάρχησε σε **όλα** τα μεγέθη πινάκων.

Για την ακρίβεια, με τα δεδομένα αυτής της άσκησης, καταλήξαμε στο εξής συμπέρασμα:

> *Αν το working set του χρήστη είναι σχετικά μικροί πίνακες, δηλαδή από n = 8 μέχρι και 30, τότε προτιμάται το configuration **(1, 8, 2)**. Αν όμως το working set του χρήστη μεγαλώσει αρκετά, δηλαδή μιλάμε για n = 32 και πάνω, τότε, μπορούμε να επιλέξουμε το configuration **(1, 4, 1)** και να χάσουμε από πλευράς ταχύτητας (# CC) για να κερδίσουμε, όμως, -σημαντικά- στο συνολικό ποσό ενέργειας που καταναλώνεται.*

Κατανομή φόρτου

Όλα τα ζητούμενα της άσκησης αυτής ήρθαν εις πέρας και από τα δύο μέλη της ομάδας, με μικρές -όχι άξιας αναφοράς- διαφορές σε μερικά σημεία. Διατηρήσαμε συνεχή συνεννόηση για οτιδήποτε κάναμε και οι απόφασεις που λαμβάναμε ήταν από κοινού.