Πανεπιστήμιο Θεσσαλίας

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Οργάνωση και Σχεδίαση Υπολογιστών
Lab #8

Ανάλυση απόδοσης του προγράμματος **K-means για image compression**

Ζουνίδης Αθανάσιος: 03873
Δεληγιάννης Βίκτωρας: 03893

Διδάσκων: Καθηγητής Μπέλλας Νικόλαος

Ερώτημα Α: Μετρήσεις & Αποτελέσματα

K = 2

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	21.38%	6.27%	326.198s
hf_rgb	22.67%	7.71%	366.962s
lfh	20.65%	6.95%	152.655s
lfv	20.59%	6.22%	265.371s

K = 4

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	22.34%	6.39%	530.567s
hf_rgb	23.75%	6.78%	581.825s
lfh	23.11%	6.19%	492.916s
lfv	22.81%	6.91%	484.382s

K = 8

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	25.98%	7.44%	932.32s
hf_rgb	26.63.75%	8.09%	1013.472s
lfh	24.59%	6.53%	849.321s
lfv	23.93%	8.56%	785.092s

Ερώτημα Α: Παρατηρήσεις

1. Παρατηρούμε ότι με την αύξηση του K αυξάνεται γραμμικά ο χρόνος του compression για όλες τις φωτογραφίες. Αυτό συμβαίνει διότι, τα pixel ομαδοποιούνται σε περισσότερα cluster.

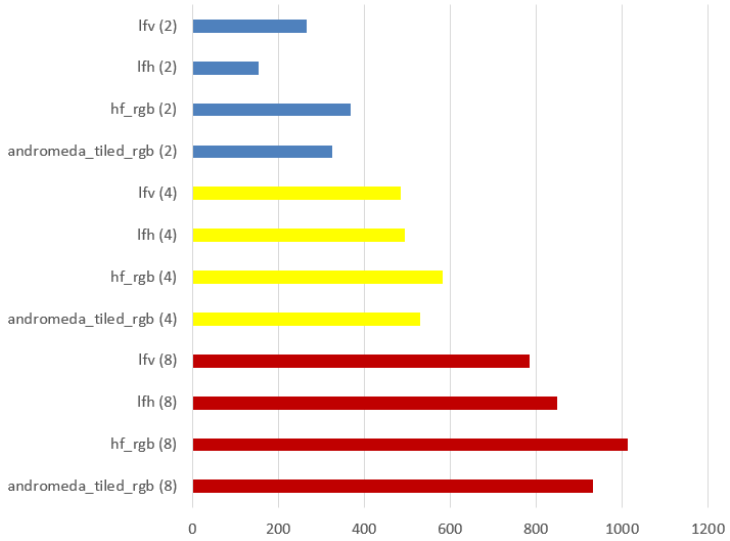


Figure 1: Χρόνος εκτέλεσης φωτογραφιών για K = 2 , K = 4 , K = 8 .

2. Για την τιμή K = 4 παρατηρούμε παρόμοια ποσοστά load misses στην L1-Dcache και στην LL-cache για όλες τις φωτογραφίες. Αντίστοιχα και στον χρόνο δεν υπάρχουν μεγάλες διαφορές. Παρόλάυτα παρατηρούμε ότι η hf_rgb είναι λίγο πιο αργή , λόγω του γεγονότος ότι είναι η μοναδική φωτογραφία που δεν έχει κάποιο pattern.

3. Επίσης, παρατηρούμε ότι η lfv εκτελείται πιο γρήγορα από την lfh παρότι αποτελούνται από το ίδιο μοτίβο με μοναδική διαφορά τον προσανατολισμό αυτού. Ο λόγος που συμβαίνει αυτό είναι ότι η lfh που έχει το μοτίβο της κάθετα (horizontally), εκμεταλλεύεται το γεγονός ότι – πριν τις βελτιστοποιήσεις - ο αλγόριθμος διατρέχει τις φωτογραφίες ανά column πρώτα.

Ερώτημα Β: Μετρήσεις και Αποτελέσματα

K = 2

Εικόνες	**L1 - Dcache**	**LL -cache**	**D-cache prefetches**	**seconds time elapsed**
hf_rg_p2	22.28%	71.10%	759x10^6	396.623s
hf_rgb_p2p1	22.95%	4.91%	841x10^6	254.724s

K = 4

Εικόνες	**L1 - Dcache**	**LL -cache**	**D-cache prefetches**	**seconds time elapsed**
hf_rg_p2	23.65%	65.04%	776x10^6	528.883s
hf_rgb_p2p1	24.18%	6.07%	849x10^6	410.353s

K = 8

Εικόνες	**L1 - Dcache**	**LL -cache**	**D-cache prefetches**	**seconds time elapsed**
hf_rg_p2	25.28%	59.94%	786x10^6	789.286s
hf_rgb_p2p1	25.65%	7.20%	840x10^6	683.37s

Ερώτημα Β: Παρατηρήσεις

1. Στην πρώτη εικόνα (hf_rgb_p2) όπου το width είναι 2^{15} έχουμε ευθυγράμμιση της cache και αρκετά conflicts, δηλαδή block που προσπαθούνε να μπουν στο ίδιο set της cache. Εν αντιθέσει στην δεύτερη εικόνα (hf_rgb_p2p1), η cache δεν είναι ευθυγραμμισμένη και λόγω αυτού έχουμε περισσότερα prefetches και μικρότερο χρόνο εκτέλεσης.

Ερώτημα Γ: Βελτιστοποιήσεις

1. Αντιστροφή των βρόγχων, ώστε πρώτα να ανατρέχεται κάθε γραμμή και μετά κάθε στήλη. Έτσι μειώνονται τα cache misses στην L1 , επειδή η μνήμη ανατρέχεται σειριακά και τα στοιχεία κάθε γραμμής βρίσκονται σε διαδοχικές θέσεις μνήμης. Αυτό εκμεταλλεύεται την τοπικότητα της μνήμης, οδηγώντας σε καλύτερη απόδοση του συστήματος λόγω της αποτελεσματικότερης χρήσης της κρυφής μνήμης.

2. Αφαίρεση της τετραγωνικής ρίζας στον υπολογισμό της απόστασης.
   Απλοποιούμε τον υπολογισμό της απόστασης, καθώς δεν μας ενδιαφέρει η ακρίβεια της πράξης Αντικαθιστούμε τον τύπο:

$$\sqrt{(r - \text{means}[i].r)^2 + (g - \text{means}[i].g)^2 + (b - \text{means}[i].b)^2}$$

με:

$$(r - \text{means}[i].r)^2 + (g - \text{means}[i].g)^2 + (b - \text{means}[i].b)^2$$

   και υψώνουμε στο τετράγωνο το mindist , καθώς θέλουμε να συγκρίνουμε μόνο τις αποστάσεις από τα διάφορα κέντρα και όχι την πραγματική ευκλείδεια απόσταση. Έτσι, ο αλγόριθμος υλοποιείται ταχύτερα, βελτιώνοντας την απόδοση του συστήματος.

3. Αντικαθιστούμε το κάλεσμα της κοστοβόρας συνάρτησης BMP_GetPixelRGB με τον κώδικα της, αφαιρώντας μερικούς περιττούς υπολογισμούς (πχ. bytes_per_pixel).

4. Ενσωματώνουμε το τρίτο στάδιο (ανανέωση mean κάθε cluster) στο δεύτερο στάδιο. Με αυτόν τον τρόπο δεν χρειάζεται να διατρέξουμε τη φωτογραφία ξανά.

5. Αξιοποίηση του -Ofast flag στην μεταγλώττιση.

6. Επιπλέον, μπορούμε να μειώσουμε τον αριθμό των max iterations (από 10 σε 8). Έτσι, καταφέρνουμε να μειώσουμε περεταίρω τον χρόνο εκτέλεσης χωρίς να παρατηρείτε σημαντική διαφορά στην ποιότητα του compression.

Τα παρακάτω αποτελέσματα είναι για την τιμή K = 4

1η Βελτιστοποίηση

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	1.45%	93.56%	246.598s
hf_rgb	1.54%	97.61%	288.392s
lfh	1.28%	93.35%	191.734s
lfv	1.54%	97.39%	230.140s

2η Βελτιστοποίηση

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	1.73%	91.69%	159.134s
hf_rgb	1.88%	97.31%	178.742s
lfh	1.64%	91.85%	129.852s
lfv	1.88%	96.45%	138.200s

3η Βελτιστοποίηση

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	2.67%	92.09%	85.640s
hf_rgb	2.75%	96.33%	112.870s
lfh	2.52%	91.04%	59.298s
lfv	2.88%	92.96%	77.274s

4η Βελτιστοποίηση

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	0.59%	94.84%	69.74s
hf_rgb	0.59%	95.03%	77.42s
lfh	0.59%	94.82%	51.36s
lfv	0.59%	95.06%	70.57s

5η Βελτιστοποίηση

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	0.59%	94.40%	62.76s
hf_rgb	0.59%	95.67%	77.73s
lfh	0.59%	93.74%	50.37s
lfv	0.59%	93.25%	58.68s

6η Βελτιστοποίηση

Εικόνες	**L1 - Dcache**	**LL -cache**	**seconds time elapsed**
andromeda_tiled_rgb	0.59%	94.19%	48.44s
hf_rgb	0.59%	92.57%	62.23s
lfh	0.59%	94.90%	41.77s
lfv	0.59%	91.54%	44.65s

Τελικά αποτελέσματα: Διαγράμματα

Figure 2: Χρόνος - Στάδιο Βελτίωσης

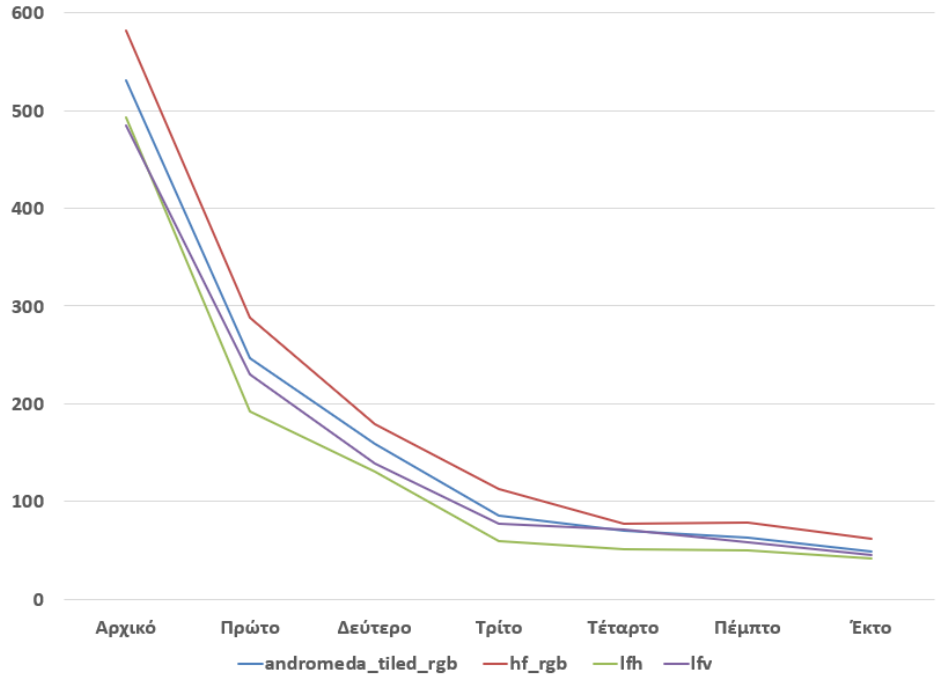


Figure 3: L1 Dcache Misses (%) - Στάδιο Βελτίωσης

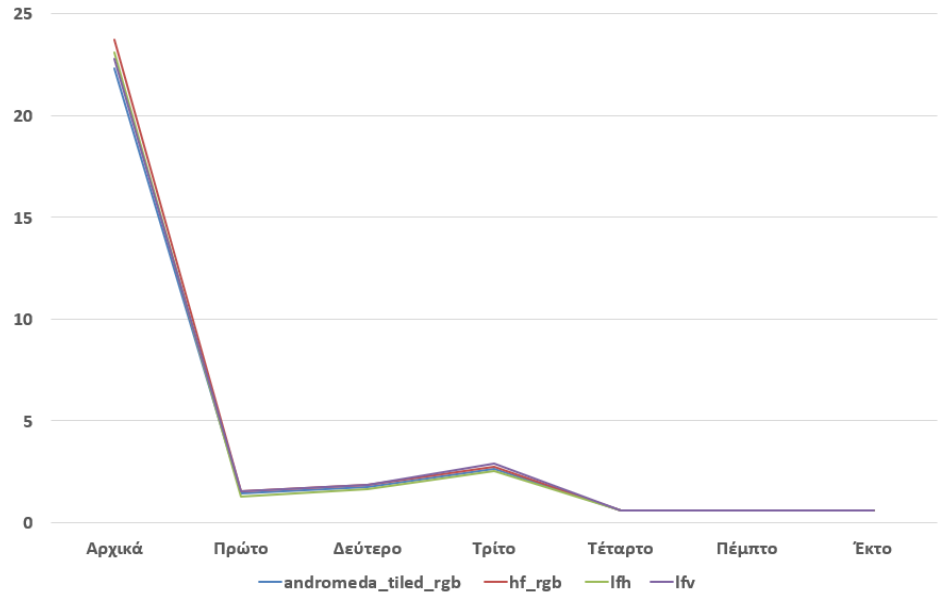
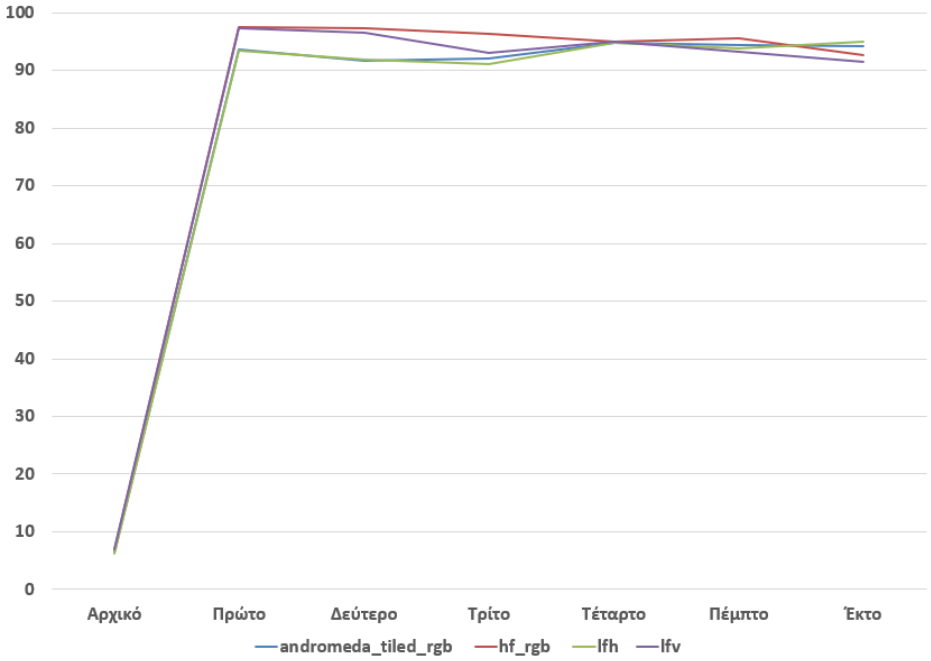


Figure 4: LLC Dcache Misses (%) - Στάδιο Βελτίωσης



Ζουνίδης Αθανάσιος: 03873

Δεληγιάννης Βίκτωρας: 03893