Εργαστήριο 8

Χειμερινό Εξάμηνο 2016-2017

1. Προσομοίωση λειτουργίας μνήμης cache (1 μονάδα)

Υποθέστε ότι κατά την εκτέλεση ενός κώδικα παράγονται οι ακόλουθες διαδοχικές διευθύνσεις μνήμης (στο δεκαδικό σύστημα), οι οποίες απευθύνονται σε μια κοινή κρυφή μνήμη εντολών και δεδομένων μεγέθους 64 bytes:

207, 212, 416, 289, 204, 222, 461, 109, 280, 303, 2, 480

Οι διευθύνσεις αυτές είναι διευθύνσεις bytes που προσπελαύνονται κατά τη φάση προσπέλασης μνήμης από εντολές προσπέλασης ενός byte. Υποθέτοντας ότι η κρυφή μνήμη είναι αρχικά άδεια και ότι τα δεδομένα προσκομίζονται όταν απαιτούνται, βρείτε τι τιμές υπάρχουν στην cache (tag, valid bits, data array) μετά από τις παραπάνω προσπελάσεις. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος 512 bytes.
Η μνήμη cache είναι 2-way set associative με cache block μεγέθους 4 bytes και LRU στρατηγική απελευθέρωσης.
Σας συνιστώ να χρησιμοποιήσετε τους πίνακες στην διαφάνεια 23 της διάλεξης για την μνήμη cache για την επίλυση της άσκησης.

2. Χαρακτηριστικά της μνήμης cache (1,5 μονάδα)

Σας δίνουμε τις 4 παρακάτω ακολουθίες διευθύνσεων που δημιουργούνται από τις εντολές load ενός προγράμματος που εκτελείται σε έναν επεξεργαστή με Data Cache. Η τελευταία στήλη του πίνακα δείχνει το cache hit ratio για κάθε ακολουθία. Χρησιμοποιώντας αυτήν την πληροφορία, η άσκηση αυτή σας ζητάει να υπολογίσετε τις παρακάτω παραμέτρους της Data Cache, θεωρώντας ότι αρχικά η Cache είναι κενή.

- Associativity A: 1, 2 ή 4-way
- Μέγεθος του Cache Block B: 1, 2, 4, 8, 16 ή 32 bytes
- Μέγεθος της Cache S: 256 ή 512 bytes
- Πολιτική αντικατάστασης ενός cache block: LRU ή FIFO (το πρώτο χρονικά στοιχείο που μπήκε στο set θα φύγει).

Σημειώστε ότι οι παρακάτω διευθύνσεις είναι σε bytes και ότι όλες οι προσπελάσεις της μνήμης διαβάζουν ένα byte.

Ακολουθία #	Διευθύνσεις	Cache Hit Ratio
1	0, 2, 4, 7, 32, 40	33%
2	0, 512, 1024, 1536, 2048, 1536, 1024, 512, 0, 1024	10%
3	0, 64, 128, 256, 512, 256, 128, 64, 0, 512	30%
4	0, 512, 1024, 0, 1536, 0, 2048, 0	25%

3. Συμπεριφορά κώδικα σε ιεραρχία μνήμης (1,5 μονάδα)

Ο παρακάτω κώδικας εκτελείται σε ένα σύστημα με L1 Data Cache 16KB, direct-mapped με 32-byte blocks. Να υπολογίσετε πόσα cache misses (read και write) θα προκαλέσει συνολικά στην L1 Data Cache η εκτέλεση του παρακάτω κώδικα. Ποιο είναι το miss rate της L1 Data Cache;

```
double a[3][100], b[101][3];

for (i = 0; i < 3; i++)
   for (j = 0; j < 100; j++)
       a[i][j] = b[j][0] * b[j+1][0];
```

3. Ανάλυση απόδοσης του προγραμματος K-Means για image compression (6 μονάδες)

Ο τρόπος με τον οποίο γράφετε ένα πρόγραμμα έχει μεγάλη επίδραση στην απόδοση του προγράμματος όπως αυτή εκφράζεται από τον χρόνο εκτέλεσης. Η λειτουργία της ιεραρχίας μνήμης και του branch predictor έχουν μεγάλη επίδραση στην εκτέλεση ενός προγράμματος. Πολλές φορές ένας καλός optimizing compiler μπορεί να εφαρμόσει βελτιστοποιήσεις σε έναν κώδικα και να αυξήσει σημαντικά την απόδοση, και μάλιστα οι compilers έχουν γίνει πολύ καλοί τώρα τελευταία στο να υλοποιούν τέτοιες βελτιστοποιήσεις. Πολλές φορές όμως αυτό δεν είναι δυνατόν, και μόνο η επέμβαση του ίδιου του προγραμματιστή-εφόσον ξέρει τι κάνει-μπορεί να επιφέρει σημαντικές διαφοροποιήσεις στην απόδοση του συστήματος μας που τρέχει μια συγκεκριμένη εφαρμογή. Η άσκηση αυτή απαιτεί από εσάς ποιοτική ανάλυση ενός φαινομένου βασιζόμενοι σε ποσοτικές μετρήσεις που λάβατε από μια σειρά πειραμάτων. Στην συνέχεια, βελτιστοποίηση του αλγορίθμου μέσω δικών σας παρεμβάσεων με σκοπό την ελαχιστοποιήση του χρόνου εκτέλεσης.

K-Means and Image Compression

Ο αλγόριθμος k-means χρησιμοποιείται για την ομαδοποίηση N στοιχείων (το N είναι συνήθως μεγάλος αριθμός) σε μία συστάδα (cluster) από K σύνολα (K << N). Ο αλγόριθμος, ο οποίος είναι επαναληπτικός, τοποθετεί κάθε ένα από τα N στοιχεία σε ένα από τα k σύνολα. Κάθε σύνολο αντιπροσωπεύεται και από ενα στοιχείο που είναι συνήθως ο μέσος όρος όλων των στοιχείων του συνόλου αυτού.
Ο παρακάτω ψευδοκώδικας υλοποιεί το k-means με εφαρμογή σε image compression[1]:

Input: An (R, G, B) image with width W and Height H (W*H, 3-tuple pixels). Each R, G, B is one byte.
Number of clusters K (Typically K << N).
Output: All W*H pixels assigned to K clusters. Compressed image

Algorithm:

1. Initialize the K cluster centroids with (R, G, B) values from random coordinates of the input image
2. noIterations = 0;
   /* Main loop */
3. while (no more movements of pixels to a different cluster && (no iterations < MAX) {

[1] Για λεπτομέρειες: https://www.youtube.com/watch?v=_aWzGGNrcic

4. Go through each pixel in the image and calculate its nearest centroid; Assign the pixel to the corresponding cluster.
5. Update the centroids of each one of the K clusters to be the mean of all pixels of that cluster
   }

6. Replace all pixels of the original image with the centroid of the cluster they belong to.

Ο αλγόριθμος αυτός καταλήγει με το να αντικαταστήσει κάθε pixel P (εκφρασμένο με τις τιμές του R, G, B) με τις αντίστοιχες τιμές του centroid του cluster στο οποίο ανήκει το P. Στην αρχική εικόνα, κάθε pixel χρειάζεται 3*8 = 24 bits για να αναπαρασταθεί, ενω κάθε εικόνα χρειάζεται 24*W*H bits. Στην τελική εικόνα, θα έχουμε μόνο K διαφορετικές τιμές των pixels, και συνεπώς χρειαζόμαστε $3 * \log_2 K$ bits για κάθε pixel. Εαν, για παράδειγμα, K= 8, τότε θα χρειαστούμε 3*3 = 9 bits για κάθε pixel. Από εκεί και το image compression.

Ο κώδικας που σας δίνεται στο αρχείο *kmeans.c* είναι η υλοποίηση του k-means με εφαρμογή σε image compression. Οι εικόνες αναπαριστώνται σε μορφή BMP (bitmap), όπου κάθε pixel χρησιμοποιεί τρείς τιμές R, G και B. Ένα ξεχωριστό πρόγραμμα *qdbmp.c* περιέχει συναρτήσεις για να επεξεργάζεσται αρχεία μορφής BMP. Επίσης σας δίνουμε και μια σειρά από πραγματικές και τεχνητές εικόνες BMP για να χρησιμοποιηθούν σαν είσοδος στο kmeans.

Πριν αναφερθούμε όμως στα πειράματα που θα πρέπει να κάνετε με το k-means, ας αναλύσουμε πως μπορείτε να καταγράφετε στοιχεία για την εκτέλεση ενός προγράμματος με τους performance counters.

Linux Performance Counters

Οι performance counters[2] είναι ειδικοί καταχωρητές στους *x86 Intel* επεξεργαστές που μετράνε δυναμικά γεγονότα (events) που συμβαίνουν στην CPU κατά την διάρκεια εκτέλεσης ενός προγράμματος. Τέτοια γεγονότα είναι ο χρόνος εκτέλεσης (execution time), ο αριθμός εκτελούμενων εντολών, το CPI, αριθμός προσπελάσεων στην cache, αριθμός cache misses, αριθμός branch mispredictions, κοκ. Το πλεονέκτημα με τους hardware performance counters είναι ότι μπορούν να προσφέρουν μια ολοκληρωμένη εικόνα του προφίλ εκτέλεσης ενός προγράμματος και να εντοπίσουν τυχόν προβλήματα με την απόδοση του συστήματος δίνοντας έτσι την ευκαιρία σε έναν προγραμματιστή να βελτιώσει την απόδοση.

Θα πρέπει να εγκαταστήσετε το *perf* package του Linux για να μπορέσετε να χρησιμοποιήσετε τους performance counters. Η παρακάτω εντολή εκτυπώνει στην οθόνη του υπολογιστή σας όλα τα γεγονότα (events) που υποστηρίζει το εργαλείο *perf*.

```
% perf list
```

Βλέπουμε για παράδειγμα ότι μπορούμε να μετρήσουμε τον αριθμό των L1 Data Cache loads, L1 Data Cache stores, L1 Data Cache load misses, L1 Data Cache store misses, LLC-loads, LLC-load-misses, branch-misses κοκ. Το σύμβολο LLC αναφέρεται στο τελευταίο επίπεδο cache στον υπολογιστή σας που είναι η L3 cache. Δεν υπάρχει τρόπος να δείτε τα misses άλλου επιπέδου της μνήμης cache.

Η επόμενη εντολή τρέχει το πρόγραμμα k-means με K=4 και στο τέλος εκτυπώνει μια σειρά από σημαντικά στατιστικά για events του προγράμματος. Το $event_i$ είναι ένα από τα events που εκτύπωσε η παραπάνω εντολή perf list.

```
% perf stat –e <event1> -e <event2> -e <eventN> ./kmeans InputImage OutputImage 4
```

[2] https://perf.wiki.kernel.org/index.php/Main_Page

Για παράδειγμα ο παρακάτω κώδικας εκτυπώνει τον αριθμό των loads και τον αριθμό των load misses από την L1 Data Cache (δίνοντας μάλιστα και το miss rate για τα loads). Το ίδιο για τον αριθμό των loads και load misses από το τελευταίο επίπεδο της Cache που είναι το L3 (LLC = L3).

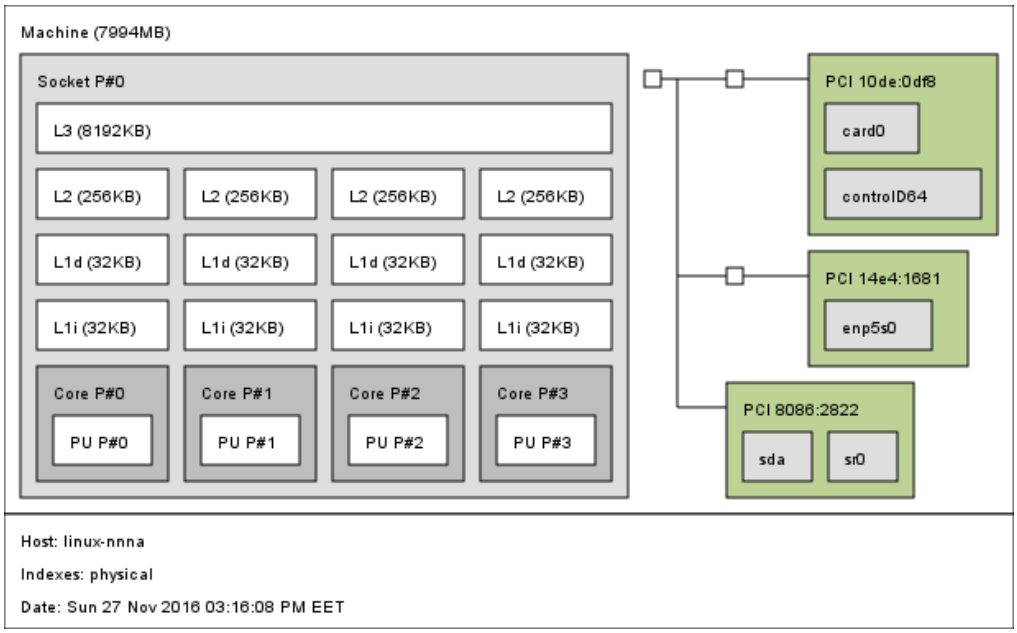
```
% perf stat -e L1-dcache-loads –e L1-dcache-load-misses –e LLC-loads –e LLC-load-misses
./kmeans InputImage OutputImage 4
```

Μπορείτε να χρησιμοποιήσετε τους hardware performance counters για να δείτε που ακριβώς οφείλετε η συμπεριφορά του κώδικα *kmeans* ή για να ελέγξετε εάν οι υποθέσεις σας είναι σωστές.

Επίσης, υπάρχουν και μέθοδοι και εργαλεία σε Linux για να βρείτε τα χαρακτηριστικά της CPU που χρησιμοποιείτε καθώς και την τοπολογία του επεξεργαστή.

```
% cat /proc/cpuinfo      # CPU information in text file
% cat /proc/meminfo     # memory Information in text file
% lstopo                 # Draws topology of the CPU. Tool needs to be installed
```

Το εργαλείο *lstopo* (list topology) έδωσε το παρακάτω output όταν το έτρεξα στο Linux Box που έχω στο γραφείο μου.



Πειράματα για Ανάλυση Απόδοσης και Βελτιστοποίηση

Μαζί με τον κώδικα σας δίνουμε και ένα *Makefile* για να το χρησιμοποιήσετε για να κάνετε compile τον κώδικά σας σε ένα Linux box. Δεν θα πρέπει να χρησιμοποιήσετε Linux virtual machines γιατί θα έχετε προβλήματα με τους hardware counters (οι VMs δεν έχουν απευθείας πρόσβαση στο hardware counters). Θα πρέπει να κάνετε τα παρακάτω πειράματα και να αναλύσετε **ΓΡΑΠΤΩΣ** την συμπεριφορά που βλέπετε κάθε φορά με βάση όσα έχουμε δει στο μάθημα. Η ανάλυση σας θα πρέπει να είναι τεκμηριωμένη και να υποστηρίζεται και απο τα πειράματά σας με τους hardware counters.

a) Να εκτελεσθεί το kmeans με *K=4*, και

i. *InputImage = andromeda_tiled_rgb.bmp*

ii. *InputImage = hf_rgb.bmp*

iii. *InputImage = lfh_rgb.bmp*

Όλες οι εικόνες είναι μεγέθους W=24000, H=7672 pixels (δηλ. πολύ μεγάλες). Να χρησιμοποιήσετε το *perf* για τα πειράματά σας ελέγχοντας όλα τα events που νομίζετε ότι θα σας βοηθήσουν στην ανάλυσή σας.

Τι παρατηρείτε σε σχέση με τον χρόνο εκτέλεσης στις τρείς αυτές περιπτώσεις; Εαν παρατηρείτε στατιστικά σημαντικές διαφορές στην εκτέλεση, που νομίζετε ότι οφείλονται αυτές οι διαφορές; Μπορείτε να χρησιμοποιήσετε και άλλα πειράματα (πχ K=2, K=8) για περισσότερες πληροφορίες. Θα σας βοηθήσει επίσης το να παρατηρήσετε τις ίδιες τις εικόνες για να δείτε το περιεχόμενο τους.

b) Να εκτελεσθεί το kmeans με *K=4*, και

i. *InputImage = hf_rgb_p2.bmp* (32768 x 4096)

ii. *InputImage = hf_rgb_p2p1.bmp* (32769 x 4096*)*

Θα πρέπει να κάνετε ανάλογα πειράματα και ανάλυση και σε αυτήν την ερώτηση όπως και στην προηγούμενη.

c) Ο κώδικας είναι γραμμένος με τρόπο που δεν είναι βέλτιστος όσον αφορά τον χρόνο εκτέλεσης. Η ερώτηση αυτή σας ζητάει να χρησιμοποιήσετε τις γνώσεις σας και την φαντασία σας για να εφαρμόσετε βελτιστοποιήσεις σε επίπεδο source code με έμφαση σε βελτιστοποιήσεις σε for-loop που στοχεύουν την ιεραρχία μνήμης.

Και για να το κάνουμε πιό ενδιαφέρον, θα οργανώσουμε ένα μικρό διαγωνισμό ανάμεσα στις ομάδες για αυτό το ερώτημα. Οι 3 ομάδες φοιτητών που θα έχουν τους καλύτερους χρόνους εκτέλεσης θα πάρουν bonus 3, 2, 1 μονάδες, αντίστοιχα[3].

Downloads

Μπορείτε να ανακτήσετε τα αρχεία που θα χρειαστούν από το:

http://tinyurl.com/huretnr

[3] Ο έλεγχος θα γίνει στους υπολογιστές του εργαστηρίου A1, με Input Image = Andromeda_rgb.bmp, K=4 και flag –O3.