ΠΑΡΑΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ ΠΡΟΒΛΗΜΑΤΑ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

1η Εργαστηριακή Άσκηση

Ο αλγόριθμος των κ-μέσων «K-MEANS»

Ον/μο: Γεώργιος Σκόνδρας

Τμήμα: Μηχανικών Η/Υ και Πληροφορικής

Α.Μ: 1020408

Σημείωση: Για την υλοποίηση της άσκησης χρησιμοποιήθηκε το υπολογιστικό σύστημα του Πανεπιστημίου για το οποίο μας δόθηκε πρόσβαση για τις ανάγκες της συγκεκριμένης άσκησης

Λειτουργικό Σύστημα: Ubuntu 20.04.1 LTS

Compiler: gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

Οι κώδικες για την κάθε εργασία βρίσκονται στα αρχεία kmeans.c*

Δηλαδή ο κώδικας για την Εργασία 1 είναι στο αρχείο kmeans1.c για την Εργασία 2 στο kmeans2.c κ.ο.κ.

Εργασία 1

- Οι απαραίτητοι πίνακες που χρησιμοποιούνται δηλώθηκαν ως global κι έτσι οι συναρτήσεις που τους χρησιμοποιούν δεν τους δέχονται σαν ορίσματα. Αυτοί είναι οι εξής:

float Vec[N][Nv] Πίνακας που περιέχει όλα τα διανύσματα στα οποία θα εφαρμόσουμε clustering.

float Center[Nc][Nv] Πίνακας που περιέχει τα κέντρα του κάθε cluster

int Classes[N] Πίνακας ακεραίων που σε κάθε θέση περιέχει τον αριθμό του cluster στο οποίο ανήκει το αντίστοιχο διάνυσμα του πίνακα Vec.

- Οι απαραίτητες συναρτήσεις που θα χρησιμοποιήσουμε είναι οι εξής:

void InitializeVectors(void) Συνάρτηση που αρχικοποιεί τα διανύσματα με τυχαίες τιμές στο διάστημα [-10,10).

void InitializeCenters(void) Συνάρτηση που θέτει κάποια διανύσματα σαν κέντρα των αντίστοιχων cluster.

void CalculateDistance(void) Συνάρτηση που για κάθε διάνυσμα του πίνακα Vec υπολογίζει τις αποστάσεις από κάθε κέντρο και ανάλογα με τη μικρότερη απόσταση τα ορίζει στην ανάλογη κλάση στον πίνακα Classes

void UpdateCenters(void); Συνάρτηση που υπολογίζει τα νέα κέντρα των clusters μετά το classification που κάνει η συνάρτηση ***CalculateDistance()*** με βάση το κέντρο μάζας των διανυσμάτων του κάθε cluster.

int Terminate(void); Συνάρτηση που ελέγχει τη σύγκλιση του αλγορίθμου και τον τερματίζει.

Εργασία 2

Σε αυτό το σημείο υλοποιούνται και κάποιες βοηθητικές συναρτήσεις, που είναι οι εξής:

void Print(float * array, int rows, int cols); Συνάρτηση που παίρνει σαν όρισμα ένα δείκτη σε πίνακα και τις διαστάσεις του και τυπώνει το περιεχόμενό του. Θα τη χρησιμοποιήσουμε για να τυπώσουμε τις τιμές των πινάκων Vec και Centers για να ελέγξουμε για τη σωστή εκτέλεση του αλγορίθμου

void PrintClasses(void); Αντίστοιχη συνάρτηση με την Print, αλλά για την τύπωση του πίνακα Classes.

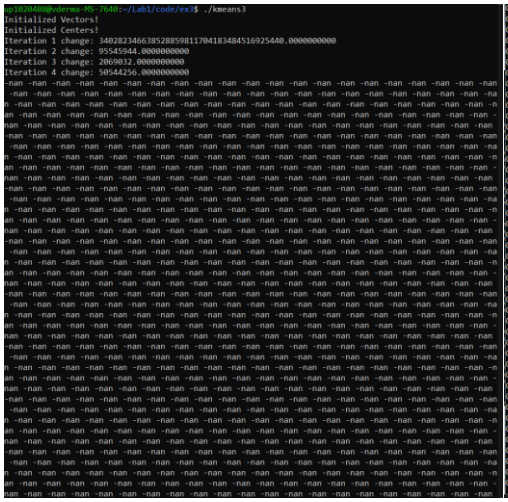
Επίσης αλλάχθηκε το όνομα της συνάρτησης ***void CalculateDistance(void)*** και έγινε ***float Classification(void);*** Η οποία είναι float, ώστε να μπορεί να επιστρέφει το άθροισμα των αποστάσεων των διανυσμάτων από τα κέντρα τους.

Να σημειωθεί επίσης, πως στη συνάρτηση ***InitializeCenters()*** που αρχικοποιεί τα κέντρα χρησιμοποιήθηκε απλά η εντολή ***memcpy(Center, Vec, sizeof(float) * Nc * Nv)*** Υπάρχει και υλοποίησή της με for loops και έλεγχο να μην επαναλαμβάνονται τα διανύσματα και βρίσκεται σχολιασμένη.

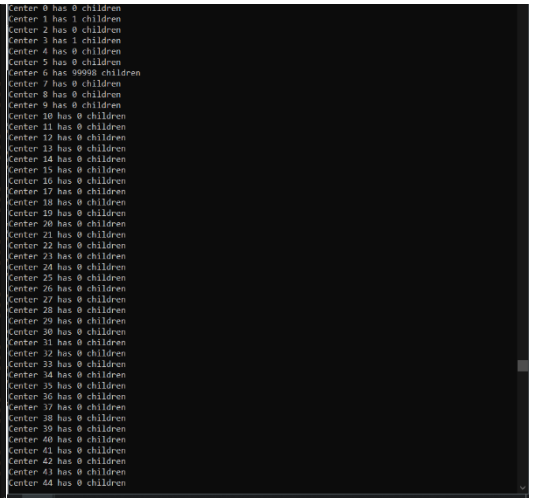
Επίσης δεν θα υλοποιηθεί η συνάρτηση ***Terminate()*** καθώς ο έλεγχος για τη σύγκλιση του αλγορίθμου θα γίνει συγκρίνοντας την απόλυτη τιμή των αποστάσεων που επιστρέφει η συνάρτηση ***Classification()*** μετά από κάθε εκτέλεση του αλγορίθμου K-means.

Εργασία 3

Σε αυτό το σημείο με τη χρήση του debugger καθώς και βοηθητικών τυπώσεων κατά την εκτέλεση του αλγορίθμου παρατηρήθηκε πως για μεγάλο πλήθος διανυσμάτων πολλών διαστάσεων με αλλαγή του εύρους των τυχαίων τιμών από [-10,10)(όπως στο Ερώτημα 2) σε [-1,1) ή [-2,2) παρατηρήθηκε πως δημιουργούνταν κάποια clusters στα οποία δεν ανήκε κανένα διάνυσμα. Αυτό συμβαίνει επειδή για τον υπολογισμό των νέων κέντρων κάναμε χρήση του κέντρου μάζας των διανυσμάτων. Για τον υπολογισμό των κέντρων μάζας διαιρούσαμε με το πλήθος των στοιχείων του κάθε cluster. Όταν αυτό είναι 0, γίνεται διαίρεση με το 0 και παρατηρούνται τιμές -nan στα δεδομένα. Επίσης κατασκευάστηκε η βοηθητική συνάρτηση ***void CenterFreq(void);*** Η οποία τυπώνει για κάθε cluster, πόσα διανύσματα ανήκουν σε αυτό. Παρακάτω φαίνονται τα αποτελέσματα της εκτέλεσης του κώδικα για τις τιμές που ζητούνται (N= 10000, Nv=1000, Nc=100) και για 4 επαναλήψεις του αλγορίθμου, που ήταν το σημείο που ξεκινούσε το πρόβλημα. Ο κώδικας βρίσκεται στο αρχείο kmeans3.c



Εικόνα 1: Οι -nan τιμές του πίνακα Centers, που προκύπτουν μετά από διαίρεση με το 0.



Εικόνα 1: Ο αριθμός των στοιχείων του κάθε cluster που τυπώνεται με τη βοήθεια της συάρτησης CenterFreq()

Συνεπώς σε αυτό το σημείο, θα πρέπει να αποτρέψουμε τη διαίρεση με το 0 που γίνεται στη συνάρτηση ***UpdateCenters()***

Για να το πετύχουμε αυτό, όταν ανανεώνονται τα κέντρα και υπολογίζεται το κέντρο μάζας, πρώτα ελέγχεται αν κάποιο cluster δεν έχει διανύσματα. Αν αυτό δεν ισχύει, τότε το κέντρο υπολογίζεται όπως πριν. Αντίθετα όταν ισχύει, τότε σαν κέντρο αυτού του cluster ορίζεται ένα διάνυσμα που ανήκει στο cluster με τα περισσότερα στοιχεία. Επίσης γίνεται έλεγχος να μην επιστρέφεται το ίδιο διάνυσμα σε περίπτωση που έχουμε πολλαπλά clusters χωρίς στοιχεία.

Αυτό υλοποιήθηκε με τη συνάρτηση ***void FixEmptyClusters(void)*** που εκτελείται μετά από κάθε εκτέλεση της ***Classification()*** και της βοηθητικής συνάρτησης ***int PopularVector(int vecClass, int vecNumber)***

Η υλοποίηση αυτή βρίσκεται στο αρχείο kmeans3_1.c και οδηγεί σε clusters που έχουν πάντα παιδιά. Επίσης παρατηρήθηκε πως το πλήθος των στοιχείων του κάθε cluster, εμφανίζει σχετικά

ομοιόμορφη συμπεριφορά. Παρακάτω φαίνεται μέρος της εκτύπωσης της συνάρτησης ***CenterFreq()***

```
Center 0 has 934 children
Center 1 has 1085 children
Center 2 has 1222 children
Center 3 has 977 children
Center 4 has 913 children
Center 5 has 1199 children
Center 6 has 877 children
Center 7 has 1070 children
Center 8 has 931 children
Center 9 has 768 children
Center 10 has 862 children
Center 11 has 1068 children
Center 12 has 1101 children
Center 13 has 1102 children
Center 14 has 1120 children
Center 15 has 1063 children
Center 16 has 823 children
Center 17 has 876 children
Center 18 has 1239 children
Center 19 has 1071 children
Center 20 has 1162 children
Center 21 has 925 children
Center 22 has 992 children
Center 23 has 1055 children
Center 24 has 1063 children
Center 25 has 771 children
Center 26 has 1155 children
Center 27 has 646 children
Center 28 has 903 children
Center 29 has 1123 children
Center 30 has 1084 children
Center 31 has 905 children
Center 32 has 1226 children
Center 33 has 1047 children
Center 34 has 1116 children
Center 35 has 1197 children
Center 36 has 1084 children
Center 37 has 1185 children
Center 38 has 1113 children
Center 39 has 1203 children
Center 40 has 1023 children
Center 41 has 1082 children
Center 42 has 938 children
```

Εικόνα 3: Πλέον όλα τα clusters έχουν στοιχεία

Εργασία 4

Εκτελώ τον κώδικα του προηγούμενου ερωτήματος μέχρι να συγκλίνει(και όχι για συγκεκριμένο αριθμό επαναλήψεων όπως πριν). Αυτή τη φορά δίνω την παράμετρο -pg στον compiler, εκτελώ τον κώδικα και στη συνέχεια τρέχω την παρακάτω εντολή gprof και βλέπω τα αποτελέσματα του profiler στο αρχείο analysis.txt

```
up1020408@vderma-MS-7640:~/Lab1/final/ex4$ gprof kmeans4 gmon.out > analysis.txt
```

```
granularity: each sample hit covers 2 byte(s) for 0.00% of 1031.12 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00 1031.12                 main [1]
              1022.43    0.00      16/16          Classification [2]
                 7.64    0.00      16/16          UpdateCenters [3]
                 1.05    0.00       1/1           InitializeVectors [4]
                 0.00    0.00       1/1           InitializeCenters [6]
-----------------------------------------------
              1022.43    0.00      16/16          main [1]
[2]     99.2 1022.43    0.00      16         Classification [2]
                 0.00    0.00      16/16          FixEmptyClusters [5]
-----------------------------------------------
                 7.64    0.00      16/16          main [1]
[3]      0.7    7.64    0.00      16         UpdateCenters [3]
-----------------------------------------------
                 1.05    0.00       1/1           main [1]
[4]      0.1    1.05    0.00       1         InitializeVectors [4]
-----------------------------------------------
                 0.00    0.00      16/16          Classification [2]
[5]      0.0    0.00    0.00      16         FixEmptyClusters [5]
-----------------------------------------------
                 0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         InitializeCenters [6]
-----------------------------------------------
```

Εικόνα 4: Αποτελέσματα Profiler

Όπως ήταν αναμενόμενο, η συνάρτηση Classification εκτελείται το 99.2 % του χρόνου. Αυτό είναι αναμενόμενο, καθώς περιέχει ένα τριπλό for loop ως προς όλες τις διαστάσεις N,Nv,Nc.

Εργασία 5

Σε αυτό το σημείο θα προσπαθήσουμε να επιταχύνουμε την εκτέλεση του προγράμματος.

Αρχικά χρονομετρούμε το πρόγραμμα χωρίς αλλαγές και βελτιστοποιήσεις στον compiler, όπως φαίνεται παρακάτω:

```
up1020408@vderma-MS-7640:~/Lab1/final/ex5$ gcc kmeans5.c -o kmeans5
up1020408@vderma-MS-7640:~/Lab1/final/ex5$ time ./kmeans5

real    16m27,542s
user    16m26,513s
sys     0m0,260s
up1020408@vderma-MS-7640:~/Lab1/final/ex5$
```

Βλέπουμε ότι έχουμε χρόνο εκτέλεσης 16 λεπτά και 27 δευτερόλεπτα

Επειδή σύμφωνα με τον profiler το 99.2% του χρόνου εκτελείται η συνάρτηση ***Classification()*** θα προσπαθήσουμε να επιταχύνουμε τη συνάρτηση αυτή. Πιο συγκεκριμένα θα προσπαθήσουμε να βελτιστοποιήσουμε τον τρόπο που υπολογίζεται η ευκλείδεια απόσταση μέσα στο τριπλό for loop, καθώς εκεί το πρόγραμμα καταναλώνει τον περισσότερο χρόνο.

Δημιουργούμε ένα νέο αρχείο **kmeans5_1.c** και στη συνάρτηση ***Classification()*** αντικαθιστούμε τις εντολές που υπολογίζουν την Ευκλείδεια απόσταση

temp = Vec[i][k] - Center[j][k];

euclDist = temp * temp;

Με μία εντολή που το κάνει αυτό, όπως φαίνεται παρακάτω:

euclDist = (Vec[i][k] - Center[j][k]) * (Vec[i][k] - Center[j][k]);

Επίσης ορίζουμε τη μεταβλητή ***euclDist*** η οποία βρίσκεται στο τριπλό for loop, ως register, ώστε οι ανανεώσεις τις να γίνονται πιο γρήγορα.

Χρονομετρούμε και πάλι με τον ίδιο τρόπο με πριν.

```
up1020408@vderma-MS-7640:~/Lab1/final/ex5$ time ./kmeans5_1

real    13m51,231s
user    13m50,816s
sys     0m0,164s
```

Βλέπουμε πως ο αλγόριθμος εκτελείται σε 13 λεπτά και 51 δευτερόλεπτα, όπως φαίνεται παρακάτω. Υπάρχει αισθητή επιτάχυνση σε σχέση με πριν.

Στη συνέχεια δίνουμε την επιλογή βελτιστοποίηση -O2 στον compiler και χρονομετρούμε ξανά, όπως φαίνεται παρακάτω.

```
up1020408@vderma-MS-7640:~/Lab1/final/ex5$ gcc -O2 kmeans5_1.c -o kmeans5_1
up1020408@vderma-MS-7640:~/Lab1/final/ex5$ time ./kmeans5_1

real    3m26,762s
user    3m26,462s
sys     0m0,236s
up1020408@vderma-MS-7640:~/Lab1/final/ex5$
```

Αυτή τη φορά έχουμε πολύ μεγάλη επιτάχυνση, καθώς ο χρόνος εκτέλεσης έπεσε στα 3 λεπτά και 26 δευτερόλεπτα.

Εργασία 6

Σε αυτό το σημείο θα εκμεταλευτούμε τις δυνατότητες παράλληλης επεξεργασίας SIMD.

Προσθέτω στον κώδικα τις παρακάτω παραμέτρους

#pragma GCC optimize("O3","unroll-loops","omit-frame-pointer","inline", "unsafe-math-optimizations")

#pragma GCC option("arch=native","tune=native","no-zero-upper")

Κάνω compile όπως και χρονομετρώ, όπως φαίνεται παρακάτω.

```
up1020408@vderma-MS-7640:~/Lab1/final/ex6$ gcc kmeans6.c -o kmeans6 -fopt-info-vec-optimized -march=native -mtune=native
kmeans6.c:133:4: optimized: loop vectorized using 16 byte vectors
kmeans6.c:133:4: optimized: loop vectorized using 16 byte vectors
kmeans6.c:166:10: optimized: loop vectorized using 16 byte vectors
kmeans6.c:159:10: optimized: loop vectorized using 16 byte vectors
up1020408@vderma-MS-7640:~/Lab1/final/ex6$ ./kmeans6
up1020408@vderma-MS-7640:~/Lab1/final/ex6$ time ./kmeans6

real    0m49,797s
user    0m49,611s
sys     0m0,164s
```

Παρατηρώ ξανά σημαντική επιτάχυνση. Ο χρόνος εκτέλεση έπεσε στα 49 δευτερόλεπτα.