ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΜΟΥ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ (ECE 415)

Ακαδημαϊκό έτος 2021-2022

2η Εργαστηριακή Άσκηση

Παραλληλοποίηση ολοκληρωμένης ακολουθιακής εφαρμογής με χρήση OpenMP.

Φοιτητές:

Ηλιάδης Ηλίας, ΑΕΜ: 2523

Μακρής Δημήτριος-Κων/νος – ΑΕΜ: 2787

Περιεχόμενα καταλόγου

Στον κατάλογο της εργαστηριακής άσκησης περιέχεται:

1) **log_file:** αρχείο το οποίο περιέχει τους χρόνους εκτέλεσης της κάθε βελτιστοποίησης με αριθμό πυρήνων 4, 8 ,16 ,32 και 64 αντίστοιχα.
2) **script.sh:** shell script το οποίο δημιουργήσαμε για την παραγωγή και την οργάνωση των μετρήσεων που πραγματοποιήθηκαν στο σύστημα csl-artemis.
3) **4 κατάλογοι** ο καθένας απο τους οποίους παραπέμπει σε μια βελτιστοποίηση. Ο εκάστοτε κατάλογος περιέχει τα κατάλληλα αρχεία για να εκτελεστεί η εφαρμογή και ένα Makefile για την ορθή μεταγλώττιση του κώδικα.
4) **Ένας κατάλογος (seq_kmeans)** που περιέχει τον ακολουθιακό κώδικα και τα απαραίτητα αρχεία για την ορθή εκτέλεση του.

Μεταγλώττιση κώδικα

Η μεταγλώττιση πραγματοποιείται στον κάθε φάκελο ξεχωριστά με την χρήση του εκάστοτε makefile. Για την μεταγλώττιση χρημοποιήθηκε ο icc compiler με OPT_FLAG **-O3.** Για την εκτέλεση του κώδικα αρκεί η εκτέλεση της εντολής **make run** στον κάθε φάκελο. Η εκτέλεση πραγματοποιείται σε όλες τις περιπτώσεις με τις παραμέτρους: **-q -o -b -n 2000 Image_data/texture17695.bin**

Εισαγωγή

Σκοπός της 2ης εργαστηριακής άσκησης ήταν η παραλληλοποίηση του κώδικα υλοποίησης του αλγορίθμου clustering k-means, και η περαιτέρω βελτιστοποίησή του, όπου αυτό κρίνεται εφικτό.

Αρχικά, με χρήση του profiler vtune, εντοπίσαμε σημεία του κώδικα τα οποία καταλαμβάνουν μεγάλο χρόνο εκτέλεσης, ξεκινώντας απο τον

υπολογισμό της ευκλείδιας απόστασης του εκάστοτε στοιχείου απο το κάθε cluster στη συνάρτηση euclid_dist_2 στο αρχείο seq_kmeans.c. Η απόπειρα παραλληλισμού του βρόγχου στη συγκεκριμένη συνάρτηση ωστόσο, δεν επέφερε κάποια μείωση του χρόνου εκτέλεσης, καθώς προκαλείται μεγάλο overhead από τις διαδικασίες fork και join της επαναδημιουργίας threads σε κάθε κλήση της (λαμβάνοντας υπόψην και το πολύ μικρό μέγεθος του βρόγχου for).

Ένα ακόμα αρκετά χρονοβόρο σημείο που βρέθηκε ήταν στη συνάρτηση find_nearest_cluster στο αρχείο seq_kmeans.c, η οποία περιέχει ένα for loop με μήκος ίσο με το πλήθος των clusters. Εσωτερικά του βρόγχου υπολογίζεται επαναληπτικά η ευκλείδια αποσταση που αναφέρθηκε παραπάνω. Επίσης παρατηρήθηκε ότι η συνάρτηση find_nearest_cluster καλείται τόσες φορές όσες και το σύνολο των σημείων (numObjs), το οποίο δεν είναι αμελητέο. Κατά την παραλληλοποίηση του εν λόγω βρόγχου παρατηρήθηκε το ίδιο ακριβώς πρόβλημα με τη δημιουργία και καταστροφή των threads, αυξάνοντας σημαντικά το overhead κι έτσι και τον χρόνο εκτέλεσης. Επιπρόσθετα, για τη σωστή διαχείριση του κώδικα ήταν απαραίτητη η χρήση critical clause στο σημείο υπολογισμού του min_dist, που σημαίνει ότι ο κώδικας των threads μέχρι ένα βαθμό διατρέχεται σειριακά, και γι'αυτό δεν παρατηρείται βελτίωση στον χρόνο εκτέλεσης.

<u>Σημείωση-1:</u> Στις παραπάνω παραλληλοποιήσεις δοκιμάστηκε και ο διαχωρισμός των "Parallel" και "For" directives, ωστόσο και πάλι το overhead παρέμεινε σε υψηλά επίπεδα.

<u>Βελτιστοποίηση 1η - parallel_kmeans_1:</u>

Στη συνέχεια, θεωρήσαμε λογικό να εστιάσουμε εσωτερικά του βρόγχου do–while της συνάρτησης seq_kmeans, όπου υλοποιείται το μεγαλύτερο ποσοστό του αλγορίθμου k-means, παραλληλοποιώντας τα δύο for loops με την χρήση των αναγκαίων παραμέτρων private για τις μεταβλητές index και j, όπως και του reduction clause για την μείωση κόστους κατα την άυξηση της μεταβλητής delta, συνενόνωντας τις τιμές delta του κάθε thread κατα την ολοκλήρωση του πρώτου for loop. Επίσης για την ορθή λειτουργία του κώδικα εντός του parallel region, κρίθηκε αναγκαία η χρήση των παραμέτρων atomic και critical καθώς υπάρχουν σημαντικά data dependencies, όπως και race conditions μεταξύ των νημάτων. Επίσης, χρησιμοποιήσαμε το single nowait directive στο τέλος του parallel region, ώστε το κάθε thread να μην χρειάζεται να περιμένει να φτάσουν τα υπόλοιπα στο barrier του single block, και να πραγματοποιεί ο υπολογισμός του delta απο ένα μόνο thread, με αποτέλεσμα να μειωθεί ο χρόνος σε σχέση με τον αρχικό κώδικα. Οι υλοποίηση του εν λόγω παραλληλισμού απεικονίζεται παρακάτω.

```
#pragma omp parallel

{

    #pragma omp for private(index, j) \
                    reduction(+:delta) \
                schedule(static)
    for (i=0; i<numObjs; i++) {
        /* find the array index of nestest cluster center */
        index = find_nearest_cluster(numClusters, numCoords, objects[i],
                                     clusters);

        /* if membership changes, increase delta by 1 */
        if (membership[i] != index) delta += 1.0;

        /* assign the membership to object i */
        membership[i] = index;

        #pragma omp atomic
        /* update new cluster center : sum of objects located within */
        newClusterSize[index]++;

        #pragma omp critical
        {
            for (j=0; j<numCoords; j++)
                newClusters[index][j] += objects[i][j];
        }
    }
```

```
    /* average the sum and replace old cluster center with newClusters */

    #pragma omp for private(j) \
                schedule(static)

    for (i=0; i<numClusters; i++) {
        for (j=0; j<numCoords; j++) {
            if (newClusterSize[i] > 0)
                clusters[i][j] = newClusters[i][j] / newClusterSize[i];
            newClusters[i][j] = 0.0;   /* set back to 0 */
        }
        newClusterSize[i] = 0;   /* set back to 0 */
    }
#pragma omp single nowait
{
        delta /= numObjs;
}
}
```

Βελτιστοποίηση 2η - parallel_kmeans_2:

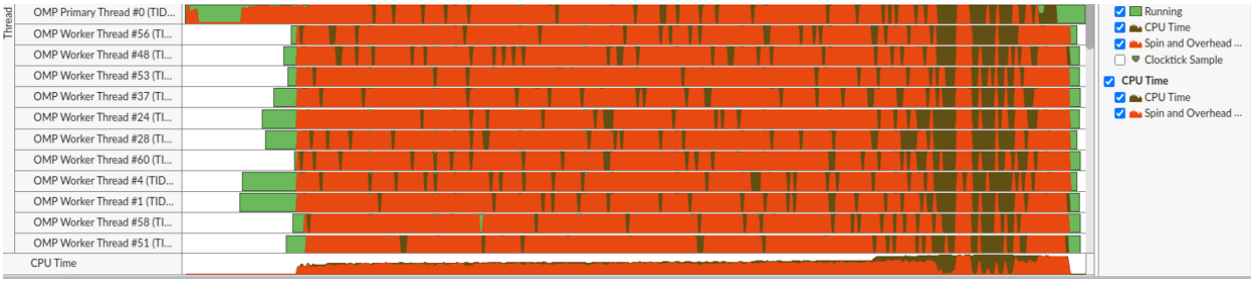
Στο συγκεκριμενο στάδιο βελτιστοποίησης πειραματιστήκαμε με το scheduling των δυο for loops. Αποδείχθηκε με βάση τις μετρήσεις ότι η αλλαγή του scheduling απο static σε dynamic, προκαλεί αποδοτικότερο διαμοίρασμο του φόρτου εργασίας μεταξύ των threads, επιτυγχάνοντας έτσι πολύ καλύτερη αξιοποίηση των πυρήνων του συστήματος. Η αλλαγή αυτή γίνεται αντιληπτή στο Effective CPU Utilization Histogram που προσφέρει ο vtune profiler, στην τελευταία ενότητα όπου απεικονίζονται τα διαγράμματα της πειραματικής αξιολόγησης. Η «δεξιά μετατόπιση» του ιστογράμματος αυτής της βελτιστοποίησης σε σχέση με το ιστόγραμμα της προηγούμενης, δείχνει ότι κατα την δίαρκεια εκτέλεσης του αλγορίθμου αξιοποιείται ταυτόχρονα μεγαλύτερο πλήθος πυρήνων σε σχέση με το static scheduling. Τέλος παρατηρήθηκε ότι όσο το μέγεθος του chunk στο scheduling αυξανόταν, ο χρόνος εκτέλεσης μειωνόταν, συνεπώς το καλύτερο αποδείχτηκε το default, δηλαδή 1.

Σημείωση-2: Σε αυτό το στάδιο βελτιστοποίησης θεωρήσαμε ότι η δημιουργία των threads εντός του do-while δημιουργεί σημαντικό

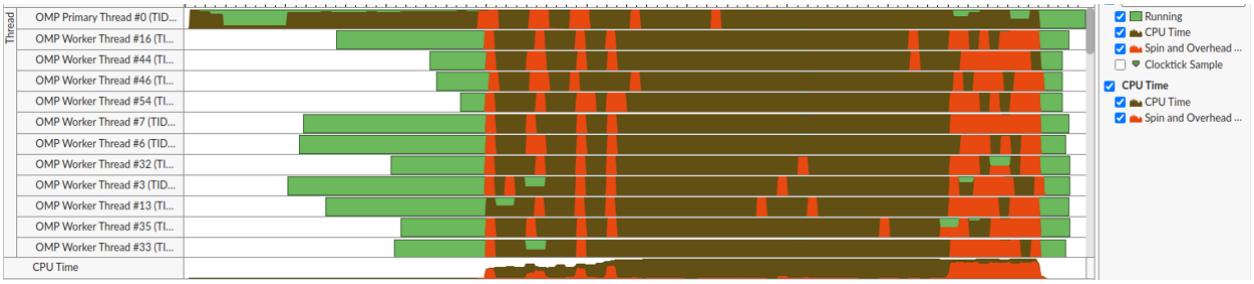
overhead στον συνολικό χρόνο εκτέλεσης. Συνεπώς θεωρήσαμε λογικό να μεταφέρουμε την δημιουργία των thread εξωτερικά του βρόγχου, πράγμα το οποίο τελικά δεν επέφερε βελτιστοποίηση στον χρόνο εκτέλεσης.

Βελτιστοποίηση 3η- parallel_kmeans_3:

Μία ακόμη παρατήρηση ήταν ότι ο χρόνος βελτιώθηκε όταν σε ορισμένες μαθηματικές πράξεις (όπως αυτή που αναφέραμε για τους πίνακες newClusters) αντικαταστάθηκε το critical με το atomic, καθώς μειώθηκε περαιτέρω το overhead που προκύπτει από την είσοδο/έξοδο του κάθε thread στο critical section (χρόνος αρκετά σημαντικός αν σκεφτούμε πως η χρήση των unnamed critical sections περιβάλλεται γύρω από ένα μοναδικό lock). Έτσι, εκμεταλλευόμενοι την ικανότητα του atomic να παρεμποδίζει οποιοδήποτε άλλο thread προσπαθεί να “επέμβει” στην ανάθεση μνήμης στην συγκεκριμένη πράξη, πραγματοποιήθηκε η μείωση αυτή στο χρόνο εκτέλεσης. Η εν λόγω αλλαγή , με βάση τις μετρικές του vtune, αύξησε σημαντικά το ποσοστό παραλληλισμού του προγράμματος αξιοποιώντας ταυτόχρονα ακόμα μεγαλύτερο πλήθος πυρήνων (Ιστόγραμμα βελτιστοποίησης parallel-kmeans3). Επίσης αυξήθηκε σημαντικά το ποσοστό χρήσης του υλικού (hardware) καθώς παρατηρείται μεγάλη μείωση του Overhead & Spin Time του κάθε νήματος, όπως φαίνεται στο παρακάτω γράφημα του profiler.



Δραστηριότητα του κάθε νήματος, Βελτιστοποίηση #2



Δραστηριότητα του κάθε νήματος, Βελτιστοποίηση #3

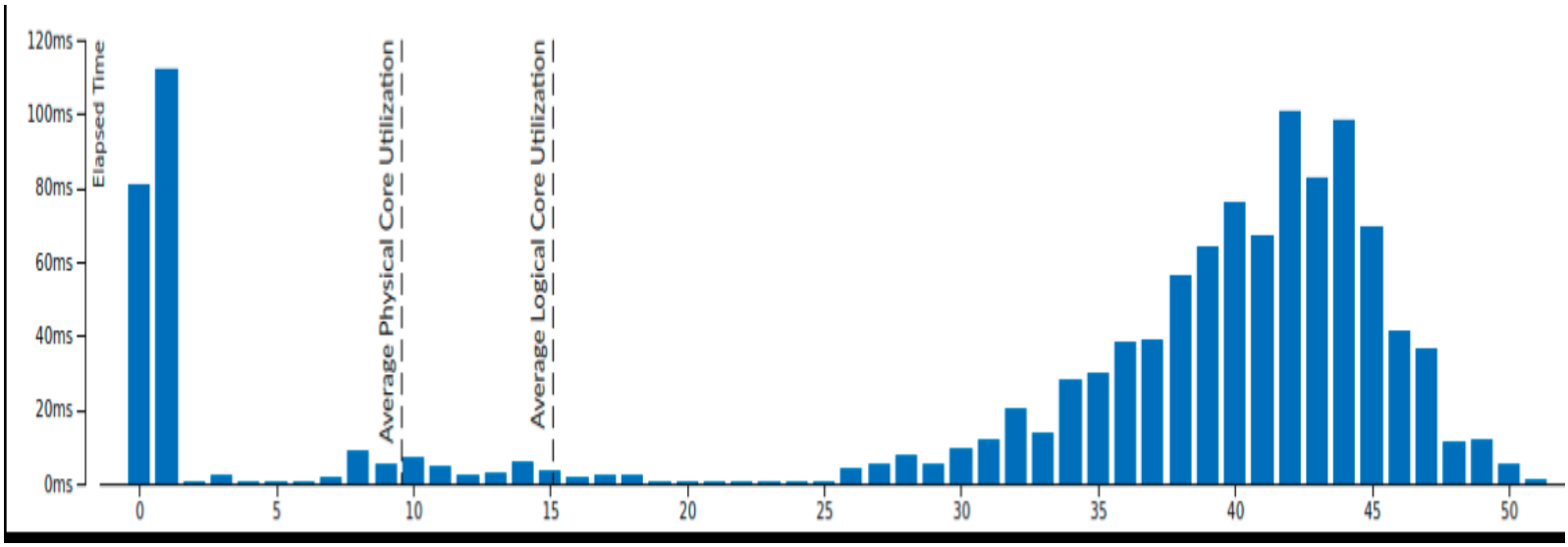
Επιπλέον, παραλληλοποιώντας τους δύο βρόγχους που αρχικοποιούν σους πίνακες membership και newClusters, αυξάνουμε το ποσοστό παραλληλοποίησης του κώδικα πετυχαίνοντας μικρή μείωση του χρόνου εκτέλεσης. Η επιλογή παραλληλοποίησης των δύο for loop με static και guided scheduling παρατηρήσαμε ότι επιφέρει το γρηγορότερο αποτέλεσμα.

Βελτιστοποίηση 4η – parallel_kmeans_test:

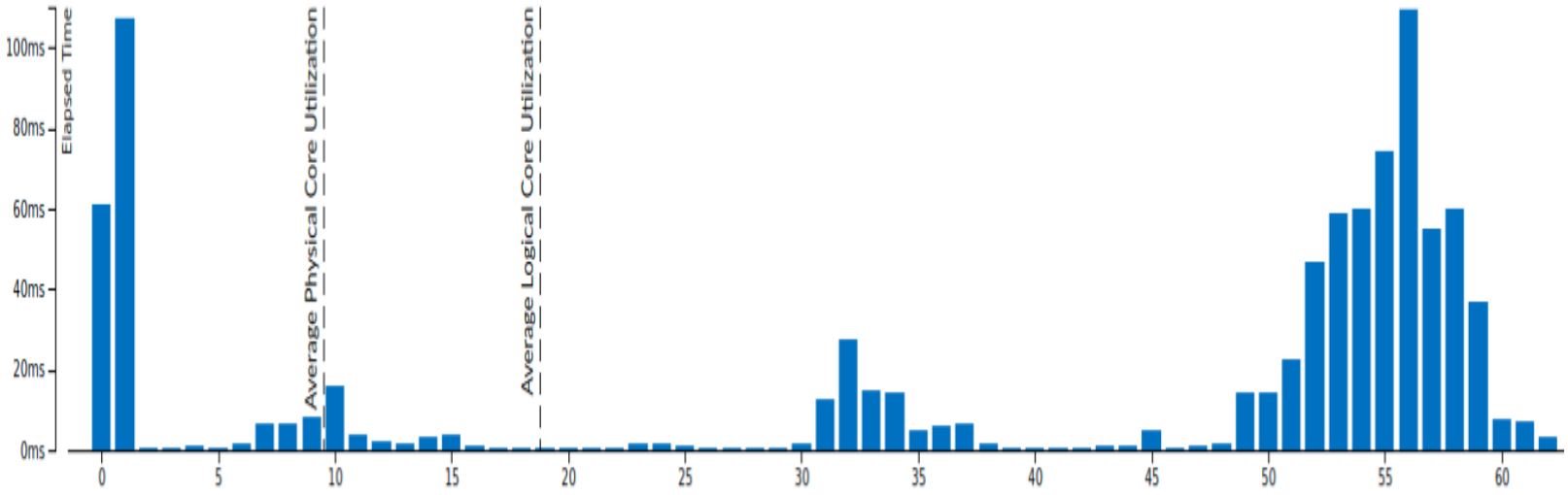
Στην συγκεκριμένη υλοποίηση βρήκαμε πως κατά το τρέξιμο του κώδικα από 64 threads σε συνδυασμό με μερικό loop unrolling χρησιμοποιώντας το direcrive #pragma unroll στο for loop της συνάρτητης e7uclid_dist_2 , πετυχαίνουμε ελάχιστη μείωση σε σχέση με την 3η βελτιστοποίηση. Διεξάγαμε το ίδιο πείραμα με λιγότερο αριθμό πυρήνων, αλλα σε αυτές τις περιπτώσεις δεν παρστηρήθηκε κάποιο είδος βελτιστοποίησης.

Διαγράμματα Μετρήσεων

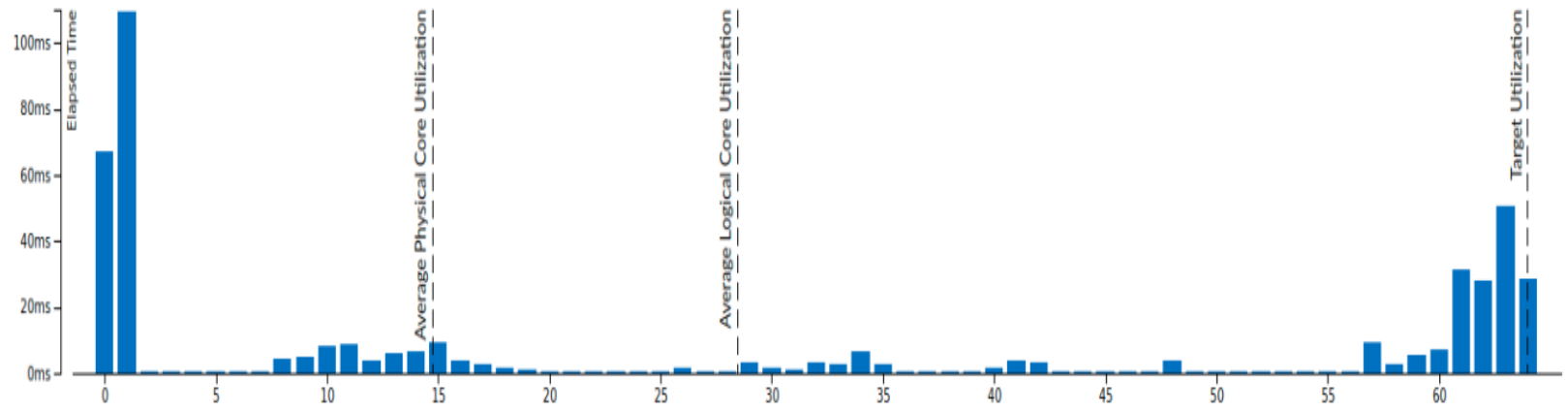
Παρακάτω παραθέτουμε τα αποτελέσματα της πειραματικής αξιολόγησης στις βελτιστοποιήσεις που αναφέραμε, με τα διαγράμματα να αναπαριστούν τη αξιοποίηση τη χρήση των CPUs του συστήματος, καθώς και το τελικό speedup που προσέφερε σε κάθε εκτέλεση η αλλαγή του αριθμού των threads:



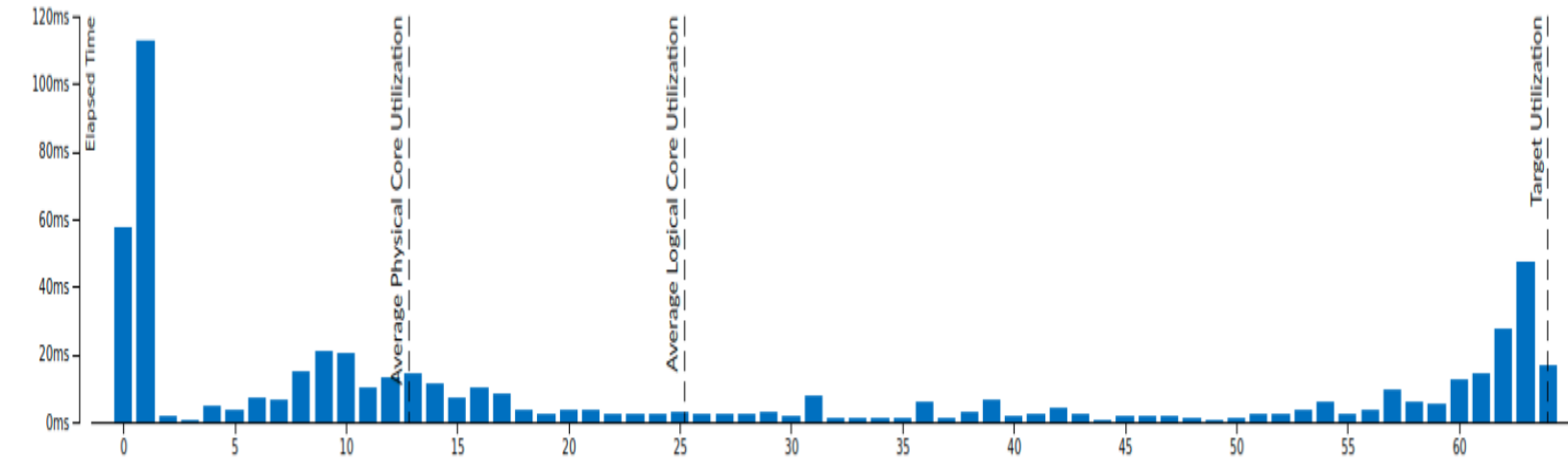
parallel_kmeans_1



parallel_kmeans_2



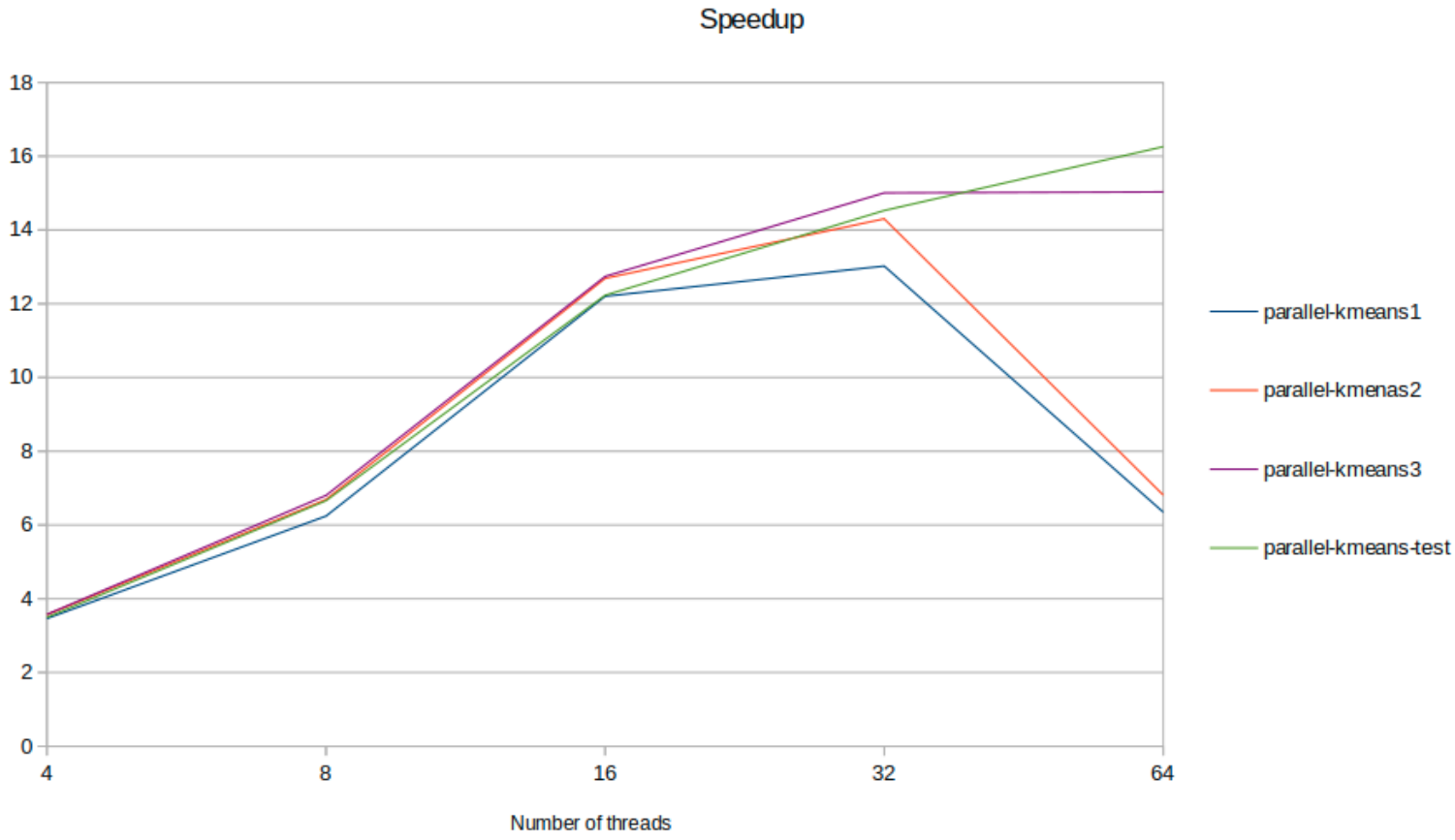
parallel_kmeans_3



parallel_kmeans_test

Παρατηρήσεις σχετικά με την πειραματική διαδικασία:

- Κατά το τρέξιμο των αρχείων, παρατηρήσαμε πως για τα αρχεία txt που περιέχουν λίγα σημεία και για τα οποία δημιουργούνται 4 clusters, όσο μεγαλύτερη γίνεται η παραλληλοποίηση τόσο περισσότερο αυξάνεται ο χρόνος εκτέλεσης, που σημαίνει ότι το overhead της δημιουργίας και καταστροφής των threads υπερβαίνει τον χρόνο της ακολουθιακής εκτέλεσης του κώδικα.
- Από την άλλη, κατά το τρέξιμο των binary αρχείων, τα οποία περιέχουν περισσότερα σημεία και δημιουργούνται 2000 clusters, ο χρόνος μειώνεται σημαντικά.
- Όσον αφορά στο speedup που προσφέρει ο παραλληλισμός, όπως φαίνεται και στο παρακάτω διάγραμμα, όσο αυξάνεται ο αριθμός των threads τόσο αυξάνεται και η επιτάχυνση του κώδικα. Συγκεκριμένα, στα αρχεία όπου συμβάλλουν και άλλες βελτιστοποιήσεις, το speedup παραμένει σταθερό, ή ακόμα και αυξάνεται γραμμικά (αρχεία parallel_kmeans_3 και parallel_kmeans_test αντίστοιχα), ενώ στα 2 πρώτα αρχεία παρατηρούμε πως όταν τα threads ξεπερνούν τα 32, το speedup μειώνεται. Αυτό οφείλεται στο ότι τα συστήματα είναι 2-way hyperthreaded, που σημαίνει ότι κάποια νήματα μοιράζονται το ίδιο core, συνεπώς απαιτείται αρκετός χρόνος για θέματα συγχρονισμού ή/και δεν εκτελούν κώδικα παράλληλα.


Speedup
18
16
14
12
10
8
6
4
2
0
4
8
16
32
64
Number of threads
parallel-kmeans1
parallel-kmenas2
parallel-kmeans3
parallel-kmeans-test