Παράλληλος και Δικτυακός υπολογισμός

Hybrid K-means with OpenMP and MPI

Ασημίνα Βουρονίκου

Ελένη Ζησιού

2012-2013

1. Ο αλγόριθμος K-means

1.1 Περιγραφή Αλγορίθμου

Το data clustering είναι μια περιγραφική εργασία εξόρυξης γνώσης για ομαδοποίηση αντικειμένων έτσι ώστε τα αντικείμενα μιας ομάδας να είναι παρόμοια (ή σχετικά) μεταξύ τους και διαφορετικά από (ή μη σχετικά με) τα αντικείμενα άλλων ομάδων. Πολλοί ερευνητές έχουν δουλέψει στην περιοχή του παράλληλου data clustering. Ο αλγόριθμος K-means (K-κέντρο) είναι ένας από τους απλούστερους και πιο δημοφιλής αλγορίθμους partitioning βασισμένος στην απόσταση. Το μεγαλύτερο κομμάτι αυτής της εργασίας στοχεύει στην επιτάχυνση της σύγκλισης και στην αύξηση της επίδοσης του αλγορίθμου. Η διαδικασία ακολουθεί ένα απλό και εύκολο τρόπο για να κατηγοριοποιήσει ένα συγκεκριμένο σύνολο δεδομένων μέσα από έναν ορισμένο αριθμό από συστάδες (υποθετικά k clusters) καθορισμένα εκ των προτέρων. Η βασική ιδέα είναι να καθοριστούν k centroids, ένα για κάθε ομάδα. Αυτά τα κέντρα βάρους (centroids) θα πρέπει να τοποθετηθούν με ένα "έξυπνο" τρόπο διότι η διαφορετική θέση προκαλεί διαφορετικό αποτέλεσμα. Έτσι, η καλύτερη επιλογή είναι να τοποθετηθούν όσο το δυνατόν μακριά ο ένας από τον άλλο. Το επόμενο βήμα είναι να ληφθεί κάθε σημείο που ανήκει σε ένα δεδομένο σύνολο δεδομένων και να συσχετιστεί στο πλησιέστερο centroid . Όταν τελειώσουν όλα τα σημεία, το πρώτο βήμα έχει ολοκληρωθεί και έχει γίνει μια πρώτη ομαδοποίηση. Σε αυτό το σημείο θα πρέπει να υπολογιστούν εκ νέου k νέα κέντρα βάρους των συστάδων που προκύπτουν από το προηγούμενο βήμα. Εφόσον έχουμε αυτά τα k νέα κέντρα βάρους, μια νέα δέσμευση πρέπει να γίνει μεταξύ των ίδιων σημείων του συνόλου δεδομένων και του πλησιέστερου νέου κέντρου βάρους. Με αυτόν τον τρόπο δημιουργείται ουσιαστικά ένας βρόχος. Ως αποτέλεσμα αυτού του βρόχου μπορεί να παρατηρήσουμε ότι τα κέντρα βάρους αλλάζουν τη θέση τους βήμα-βήμα μέχρι να μην μπορούν να γίνουν νέες αλλαγές. Με άλλα λόγια να μην παρατηρείται καμία νέα κίνηση όσον αφορά τα centroids. Ο αλγόριθμος αυτός αποσκοπεί στην ελαχιστοποίηση μιας αντικειμενικής συνάρτησης, και πιο συγκεκριμένα μιας τετραγωνισμένης συνάρτησης σφάλματος.

Η αντικειμενική συνάρτηση:

$$\mathbf{J} = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

όπου $\left\| x_i^{(j)} - c_j \right\|^2$ είναι μια επιλεγμένη μετρική απόστασης (συνήθως η Ευκλείδεια) ανάμεσα σε ένα σημείο δεδομένων $x_i^{(j)}$και το κέντρο του cluster c_j, και χρησιμεύει ως δείκτης απόστασης των n σημείων δεδομένων από τα αντίστοιχα κέντρα διασποράς τους.

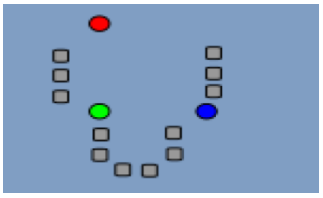
1.2 Βήματα του αλγορίθμου

Ο αλγόριθμος μπορεί να οργανωθεί ([5], [7])σε τρία βασικά βήματα:

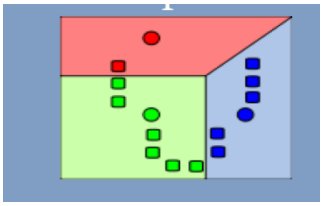
1. Τοποθέτησε K σημεία στο ίδιο χώρο όπου είναι συγκεντρωμένα τα δεδομένα προς συσταδοποίηση. Τα σημεία αυτά αποτελούν αρχικά τα centroids των clusters.
2. Αντιστοίχισε κάθε δεδομένο στο cluster που έχει το πιο κοντινό κέντρο βάρους. Όταν όλα τα δεδομένα έχουν ανατεθεί, υπολόγισε εκ νέου τις θέσεις των K - centroids .
3. Επανέλαβε τα βήματα 1 και 2 μέχρι τα centroids πλέον να μην μετακινούνται. Αυτό παράγει ένα διαχωρισμό των αντικειμένων σε clusters από τα οποία η μετρική που πρέπει να ελαχιστοποιηθεί μπορεί πλέον να υπολογιστεί.

Στα σχήματα που ακολουθούν απεικονίζονται με τη σειρά τα βήματα του αλγορίθμου που αναφέρθηκαν [6]. Στο Σχήμα 1, k σημεία επιλέγονται τυχαία ως κέντρα. Στη συνέχεια k- clusters δημιουργούνται συνδέοντας κάθε σημείο με το κοντινότερο κέντρο (Σχήμα 2). Όπως φαίνεται στο Σχήμα 3, υπολογίζεται το νέο κέντρο και επαναλαμβάνεται η διαδικασία. Στο Σχήμα 4 δεν παρατηρείται πλέον καμία μετακίνηση επομένως ο αλγόριθμος τερματίζει έχοντας δημιουργήσει τις τελικές συστάδες των clusters.

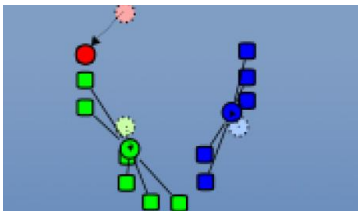
Παράδειγμα:



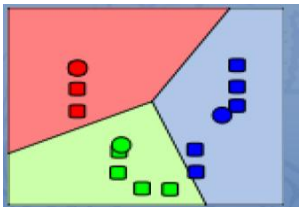
Σχήμα 1 : K -σημεία διαλέγονται τυχαία ως κέντρα



Σχήμα 2 : K-clusters δημιουργούνται συνδέοντας κάθε σημείο με το κοντινότερο κέντρο



Σχήμα 3 : Υπολογίζεται το νέο κέντρο του κάθε cluster



Σχήμα 4 : Επανάληψη μέχρι να μην υπάρχει μετακίνηση

1.3 Πολυπλοκότητα του αλγορίθμου

Λαμβάνοντας υπόψη την υπολογιστική πολυπλοκότητα του αλγορίθμου ([8], [9]), ο k-means, για data-points διάστασης d είναι :

- NP-hard στο γενικό Ευκλείδειο χώρο *d* ακόμη και για 2 clusters
- NP-hard για ένα γενικό αριθμό από clusters *k* ακόμη και στο επίπεδο

Η πολυπλοκότητα είναι O(n* k*I*d)

Όπου n = αριθμός αντικειμένων, k = αριθμός clusters, I = αριθμός επαναλήψεων, d = διάσταση. Για σταθερά k,d ο αλγόριθμος παρατηρείται ότι μπορεί να ολοκληρώσει το clustering σε ακριβώς **$O(n^{dk+1}$ log n)** χρόνο**.**

1.4 Πρόταση

Σήμερα τα περισσότερα συστήματα υψηλών επιδόσεων έχουν ιεραρχικά δομημένο hardware όπου τα shared-memory nodes, με πολλές multicores CPUs, συνδέονται μεταξύ τους μέσω ενός network infrastructure. Ο παράλληλος προγραμματισμός πρέπει να συνδυάσει distributed memory παραλληλισμό στη διασύνδεση των κόμβων, με shared-memory παραλληλισμό στα εσωτερικά ενός κόμβου. Προκλήσεις που παρουσιάζονται είναι η μείωση του overhead συγχρονισμού μεταξύ των threads στο εσωτερικό ενός κόμβου καθώς και η ελαχιστοποίηση του overhead επικοινωνίας μεταξύ των κόμβων.

Συγκεκριμένα για τον αλγόριθμο K-means που αναλύσαμε παραπάνω αν και μπορεί να αποδειχθεί ότι η διαδικασία θα τερματίζει πάντα, ο k-means αλγόριθμος δεν βρίσκει κατ 'ανάγκην τη βέλτιστη διαμόρφωση που αντιστοιχεί στην ελάχιστη αντικειμενική συνάρτηση. Ο αλγόριθμος είναι επίσης σημαντικά ευαίσθητος στα αρχικά τυχαία κέντρα που επιλέγονται. Όσο το σύνολο δεδομένων κλιμακώνει γίνεται όλο και πιο δύσκολο να χρησιμοποιήσουμε τον k-means για ένα τόσο μεγάλο ποσό δεδομένων. Αυτό οδηγεί σε ανάγκη παραλληλισμού του αλγορίθμου για εκμετάλλευση αφενός της υπολογιστικής ισχύς που διαθέτουν τα σημερινά υπολογιστικά συστήματα και αφετέρου της φύσης του αλγορίθμου που προσφέρεται για παραλληλισμό.

Με βάση τις παραπάνω παρατηρήσεις και το γεγονός ότι τα σημεία δεδομένων είναι ανεξάρτητα μεταξύ τους προτείνουμε την παράλληλη εκδοχή του αλγορίθμου, και πιο συγκεκριμένα μια **υβριδική λύση** προγραμματίζοντας σε **OpenMP και MPI.** Το MPI θα χρησιμοποιηθεί για επικοινωνία μεταξύ των clusters και το OpenMP για την παραλληλοποίηση της δουλειάς στο εσωτερικό των κόμβων.

2. Υλοποίηση Σειριακού Κώδικα

2.1 Σειριακός κώδικας - Optimizations

Κατά την υλοποίηση του σειριακού κώδικα επιλέχθηκαν να γίνουν κάποιες βελτιστοποιήσεις όσον αφορά την αρχικοποίηση των centroids και τη συνθήκη τερματισμού του αλγορίθμου. Πιο συγκεκριμένα :

1. Ο αλγόριθμος αρχικοποίησης KKZ:

Για την αρχικοποίηση βασιστήκαμε στον αλγόριθμο KKZ [1], που συνοψίζεται στα παρακάτω βήματα:

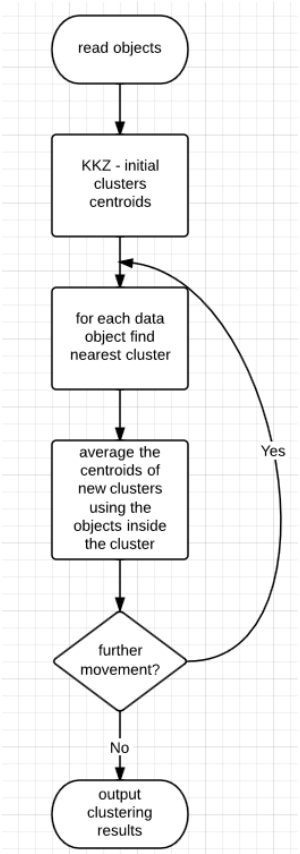
Βήμα 1 :Υπολόγισε τις νόρμες όλων των διανυσμάτων στο dataset. Διάλεξε το vector με την μεγαλύτερη νόρμα ως πρώτο centroid.

Βήμα 2: Υπολόγισε τις αποστάσεις όλων των διανυσμάτων στο dataset από το πρώτο centroid. Διάλεξε το vector με την μεγαλύτερη απόσταση ως δεύτερο centroid.

Βήμα 3: Για ένα σύνολο από centroids μεγέθους i με i=2,3 …υπολόγισε την απόσταση κάθε διανύσματος από κάθε centroid και κράτα την μικρότερη. Από όλες τις αποστάσεις η μεγαλύτερη είναι το (i+1) centroid. Συνέχισε μέχρι να υπολογιστούν K- centroids.

2.Τερματισμός επαναλήψεων του αλγορίθμου :

Η δεύτερη βελτιστοποίηση αφορά τις επαναλήψεις που γίνονται για να υπολογιστούν τα centroid για κάθε data point. Γενικά στον αλγόριθμο οι επαναλήψεις σταματούν με βάση ένα tolerance. Στην υλοποίησή μας οι επαναλήψεις να σταματούν όταν δεν παρατηρείται καμία κίνηση. Κάθε φορά που υπολογίζεται το καινούριο centroid για κάθε διάνυσμα αυξάνουμε έναν μετρητή. Εάν σε κάποια επανάληψη ο μετρητής παρατηρηθεί να έχει την τιμή 0, τότε ο αλγόριθμος τερματίζει. Αυτός ο τρόπος είναι πιο αποδοτικός, καθώς εγγυάται την καλύτερη δυνατή τοποθέτηση των αντικειμένων στα clusters.



Εικόνα 1 : *Διάγραμμα ροής αλγορίθμου*

4. Παραλληλοποίηση με OpenMP

Το OpenMP είναι ένα API που υποστηρίζει multi-platform πολυεπεξεργαστικό προγραμματισμό κοινής μνήμης, σε C, C + +, και Fortran, στις περισσότερες αρχιτεκτονικές επεξεργαστών και λειτουργικά συστήματα, συμπεριλαμβανομένων των Solaris, AIX, HP-UX, GNU / Linux, Mac OS X και Windows. Αποτελείται από ένα σύνολο compiler directives, ρουτινών βιβλιοθήκης, και μεταβλητών περιβάλλοντος που επηρεάζουν την runtime συμπεριφορά. Ο προγραμματιστής είναι υπεύθυνος για το χειρισμό των παράλληλων περιοχών και το συγχρονισμό των νημάτων.

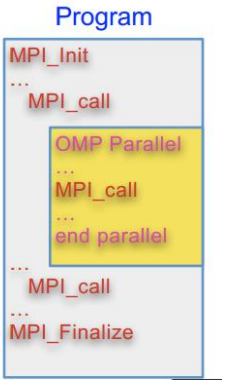
Στρατηγική παραλληλοποίησης

- Ο υπολογισμός του κοντινότερου cluster για κάθε αντικείμενο γίνεται παράλληλα με OpenMP parallel for
- Η μέτρηση των αλλαγών στα clusters γίνεται ατομικά με omp atomic.
- Η άθροιση των αντικειμένων του κάθε cluster καθώς και η μέτρηση των εμφανίσεων τους γίνεται ατομικά με omp atomic.
- Ο υπολογισμός των νέων centroids γίνεται παράλληλα με OpenMP parallel for.

5. Υβριδική Υλοποίηση OpenMP-MPI

Το βασικό μοντέλο για τον σχεδιασμό μιας υβριδικής εφαρμογής περιλαμβάνει τα εξής βήματα:

- Αρχικοποίηση MPI
- Ορισμός των περιοχών που μπορούν να παραλληλοποιηθούν με OpenMP εντός της MPI διαδικασίας
- Οι σειριακές περιοχές είναι το κύριο νήμα ή το MPI task.
- Το MPI rank είναι γνωστό σε όλα τα νήματα
- Κλήση της MPI βιβλιοθήκης σε σειριακές και παράλληλες περιοχές.
- Οριστικοποίηση MPI



Εικ.1 Hybrid model

Στο παρόν υβριδικό μοντέλο, τα data points είναι κατανεμημένα ισόποσα ανάμεσα στα clusters. Η επαναληπτική αντιστοίχηση στα centroids γίνεται παράλληλα με την χρησιμοποίηση των OpenMP directives. Η συμπεριφορά αυτής την υλοποίησης μελετάται με τη χρονοβελτίωση και την αποδοτικότητα, για διάφορα μεγέθη προβλημάτων.

Στρατηγική παραλληλοποίησης

- Αρχικοποίηση του MPI μοντέλου
- Διαμοιρασμός των αντικειμένων στα clusters
- Σε κάθε επανάληψη στο εσωτερικό του κάθε κόμβου:
  - Ο υπολογισμός του κοντινότερου cluster για κάθε αντικείμενο γίνεται παράλληλα με OpenMP parallel for
  - Η μέτρηση των αλλαγών στα clusters γίνεται ατομικά με omp atomic.
  - Η άθροιση των αντικειμένων του κάθε cluster καθώς και η μέτρηση των εμφανίσεων τους γίνεται ατομικά με omp atomic
- Στο τέλος κάθε επανάληψης έχουμε reduction των αποτελεσμάτων
- Ο υπολογισμός των νέων centroids γίνεται μόνο από τον κόμβο 0 και γίνεται broadcast στους υπολοίπους
- Οι επαναλήψεις συνεχίζονται μέχρι να μην υπάρχει μετακίνηση στα clusters.

6. Μετρήσεις

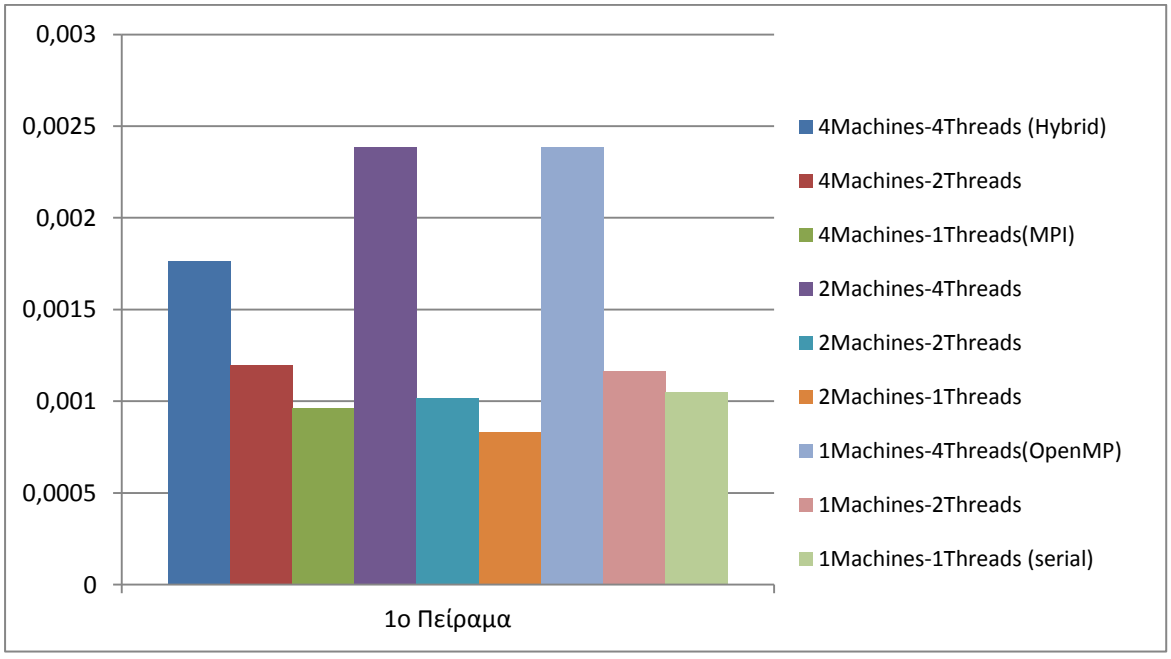
Οι μετρήσεις που παρουσιάζονται έγιναν σε ένα cluster 4 πολυπύρηνων μηχανημάτων. Κάθε μηχάνημα διαθέτει 4 πυρήνες , 8 GB RAM ,Intel Xeon W3550 @3.07GHz επεξεργαστή. Οι μετρήσεις πραγματοποιήθηκαν για 4 τάξεις μεγέθους data set .Για κάθε μια τάξη έγιναν μετρήσεις σε clusters 1,2 και 4 μηχανημάτων και για 1,2 και 4 threads ανά μηχάνημα. Όλες οι περιπτώσεις εξετάστηκαν και συγκρίθηκαν τόσο για τυχαία αρχικοποίηση των centroids όσο και για την αρχικοποίηση μέσω του KKZ αλγορίθμου. Όλες οι μετρήσεις είναι αποτέλεσμα μέσου όρου 5 επαναλήψεων και είναι σε δευτερόλεπτα. Παρατίθενται διαγράμματα της χρονοβελτίωσης (speedup) όλων των περιπτώσεων.

Τυχαία αρχικοποίηση :

1ο Πείραμα (Data Set : 1200 αντικείμενα, διάσταση : 2, κέντρα : 2 - Επαναλήψεις : 11)

Machines	Data Points	Dimension	Centroids	Threads	Total time (sec)
4	1200	2	2	4	0,001767
4	1200	2	2	2	0,001197
4	1200	2	2	1	0,000960
2	1200	2	2	4	0,002386
2	1200	2	2	2	0,001020
2	1200	2	2	1	0,000829
1	1200	2	2	4	0,002386
1	1200	2	2	2	0,001163
1	1200	2	2	1	0,001050

Πίνακας 1.1

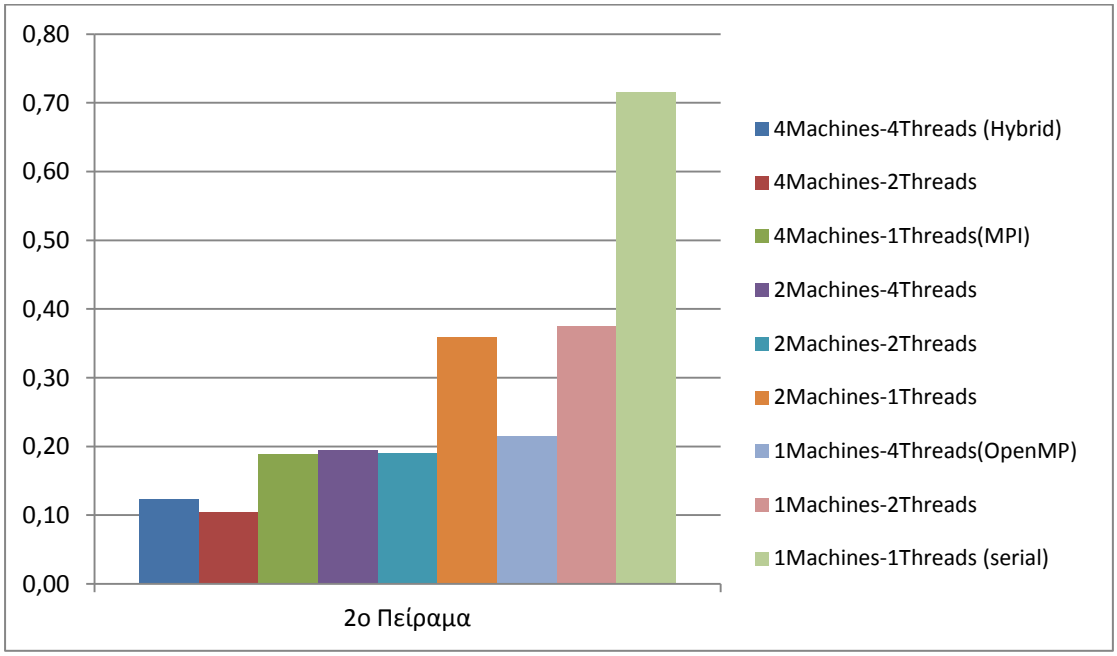


Γράφημα 1.1

2ο Πείραμα (Data Set : 12000 αντικείμενα, διάσταση : 8, κέντρα : 32 - Επαναλήψεις : 34)

Machines	Data Points	Dimension	Centroids	Threads	Total Time (sec)
4	12000	8	32	4	0,123551
4	12000	8	32	2	0,104236
4	12000	8	32	1	0,187950
2	12000	8	32	4	0,194453
2	12000	8	32	2	0,190198
2	12000	8	32	1	0,358045
1	12000	8	32	4	0,214646
1	12000	8	32	2	0,374995
1	12000	8	32	1	0,714961

Πίνακας 1.2

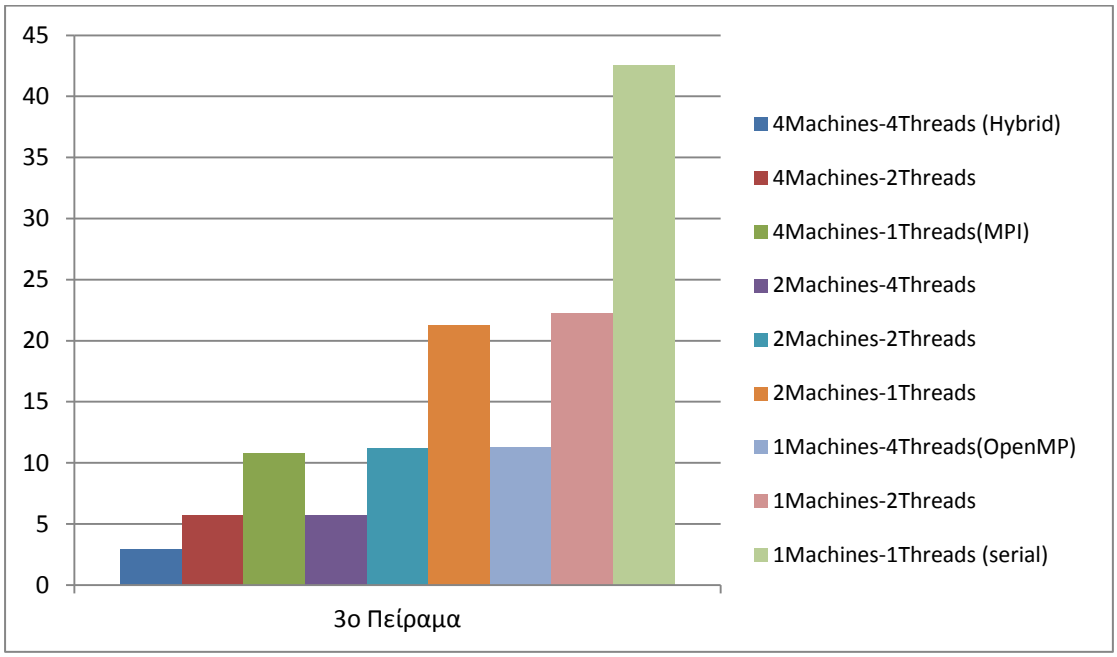


Γράφημα 1.2

3ο Πείραμα (Data Set : 120000 αντικείμενα, διάσταση : 16, κέντρα : 1024 - Επαναλήψεις : 395)

Machines	Data Points	Dimension	Centroids	Threads	Total Time (sec)
4	120000	16	1024	4	295,333141
4	120000	16	1024	2	567,409306
4	120000	16	1024	1	1074,647152
2	120000	16	1024	4	572,110406
2	120000	16	1024	2	1121,669505
2	120000	16	1024	1	2126,563475
1	120000	16	1024	4	1128,838505
1	120000	16	1024	2	2226,06437
1	120000	16	1024	1	4250,252535

Πίνακας 1.3

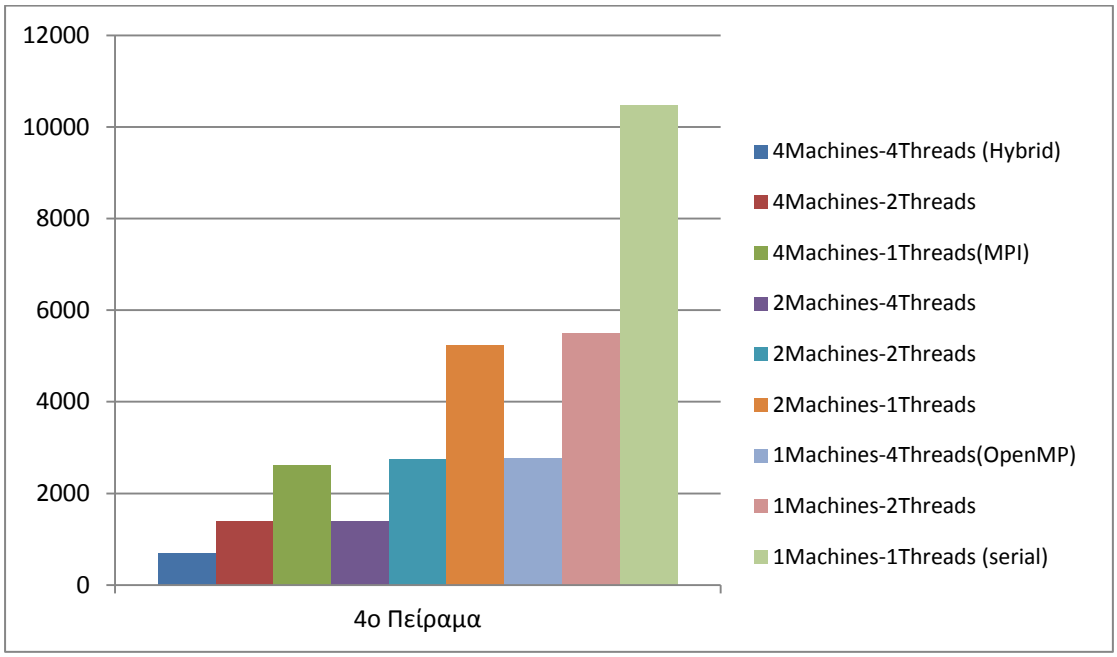


Γράφημα 1.3

4ο Πείραμα (Data Set : 1200000 αντικείμενα, διάσταση : 16, κέντρα : 10000 - Επαναλήψεις : 999)

Machines	Data Points	Dimension	Centroids	Threads	Total Time (sec)
4	1200000	16	10000	4	69869,023594
4	1200000	16	10000	2	138176,8818
4	1200000	16	10000	1	262330,4804
2	1200000	16	10000	4	138229,3753
2	1200000	16	10000	2	275509,7255
2	1200000	16	10000	1	523845,825
1	1200000	16	10000	4	276057,2764
1	1200000	16	10000	2	549321,5985
1	1200000	16	10000	1	1048007,267

Πίνακας 1.4



Γράφημα 1.4

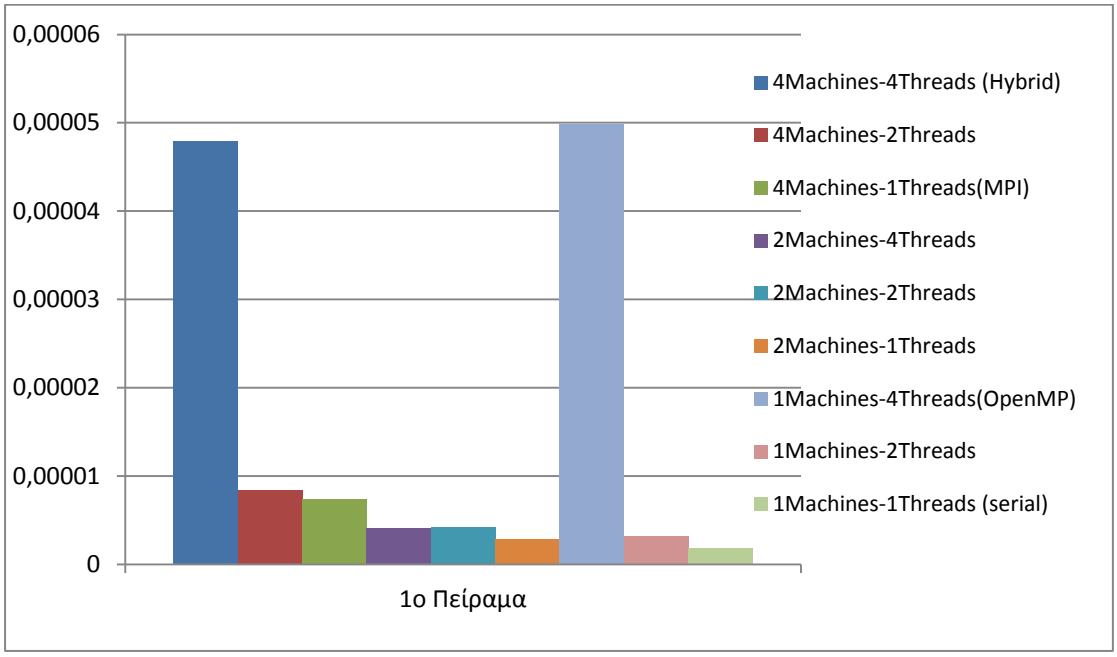
Στους επόμενους πίνακες που ακολουθούν φαίνονται οι χρόνοι για τις αντίστοιχες μετρήσεις όταν η αρχικοποίηση των centroids γίνεται με τον αλγόριθμο KKZ.

KKZ αρχικοποίηση :

1ο Πείραμα (Data Set : 1200 αντικείμενα, διάσταση : 2, κέντρα : 2)

Machines	Data Points	Dimension	Centroids	Threads	Iterations	Total Time (sec)
4	1200	2	2	4	2	0,004795
4	1200	2	2	2	2	0,000843
4	1200	2	2	1	2	0,000741
2	1200	2	2	4	2	0,000410
2	1200	2	2	2	2	0,000425
2	1200	2	2	1	2	0,000289
1	1200	2	2	4	2	0,004983
1	1200	2	2	2	2	0,000328
1	1200	2	2	1	2	0,000190

Πίνακας 2.1

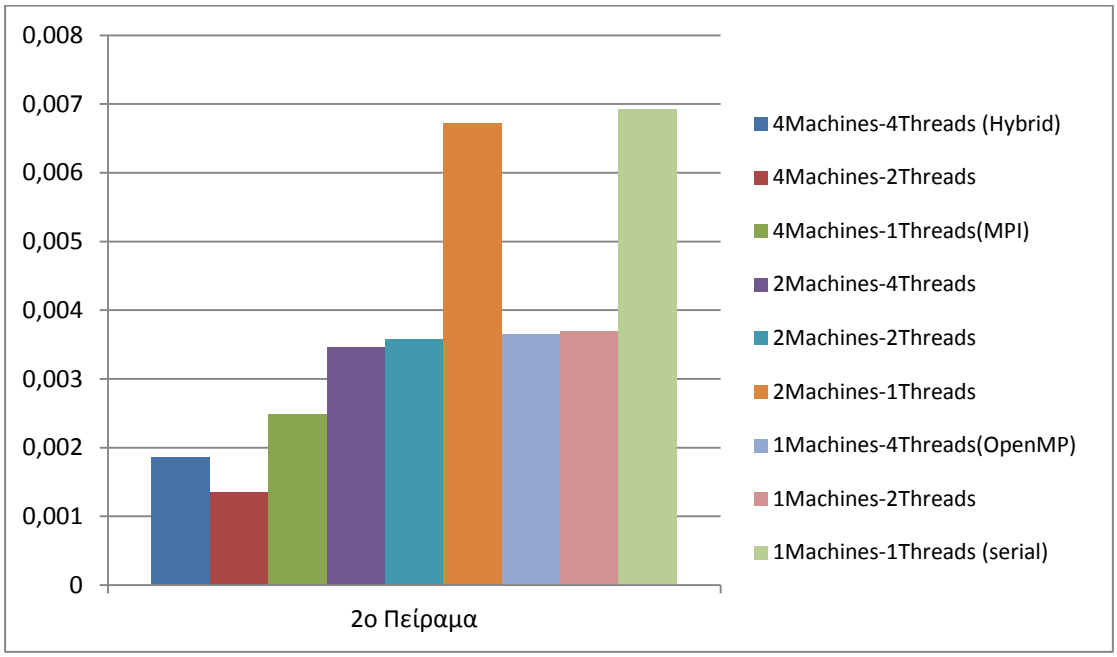


Γράφημα 2.1

2ο Πείραμα (Data Set : 12000 αντικείμενα, διάσταση : 8, κέντρα : 32)

Machines	Data Points	Dimension	Centroids	Threads	Iterations	Total Time (sec)
4	12000	8	32	4	44	0,185831
4	12000	8	32	2	44	0,135532
4	12000	8	32	1	44	0,248927
2	12000	8	32	4	62	0,345258
2	12000	8	32	2	62	0,358263
2	12000	8	32	1	62	0,671434
1	12000	8	32	4	34	0,365213
1	12000	8	32	2	34	0,369116
1	12000	8	32	1	34	0,691679

Πίνακας 2.2

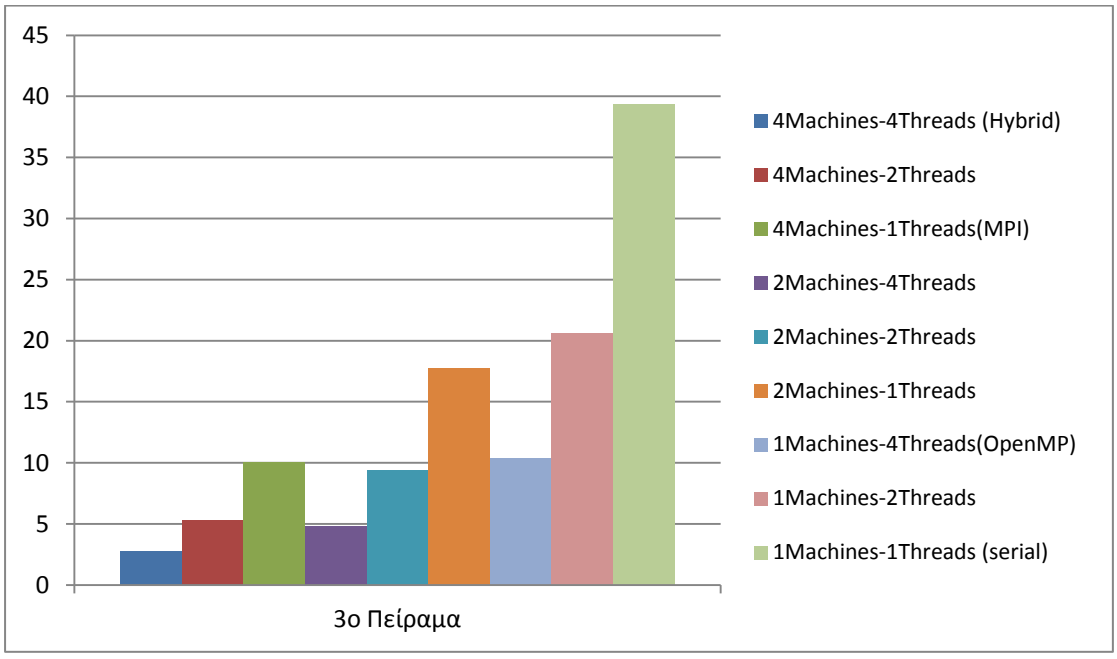


Γράφημα 2.2

3ο Πείραμα (Data Set : 120000 αντικείμενα, διάσταση : 16, κέντρα : 1024)

Machines	Data Points	Dimension	Centroids	Threads	Iterations	Total Time (sec)
4	120000	16	1024	4	370	274,259329
4	120000	16	1024	2	370	531,497309
4	120000	16	1024	1	370	1006,631509
2	120000	16	1024	4	330	477,024125
2	120000	16	1024	2	330	937,0909789
2	120000	16	1024	1	330	1776,62265
1	120000	16	1024	4	366	1040,770364
1	120000	16	1024	2	366	2062,631796
1	120000	16	1024	1	366	3938,208678

Πίνακας 2.3

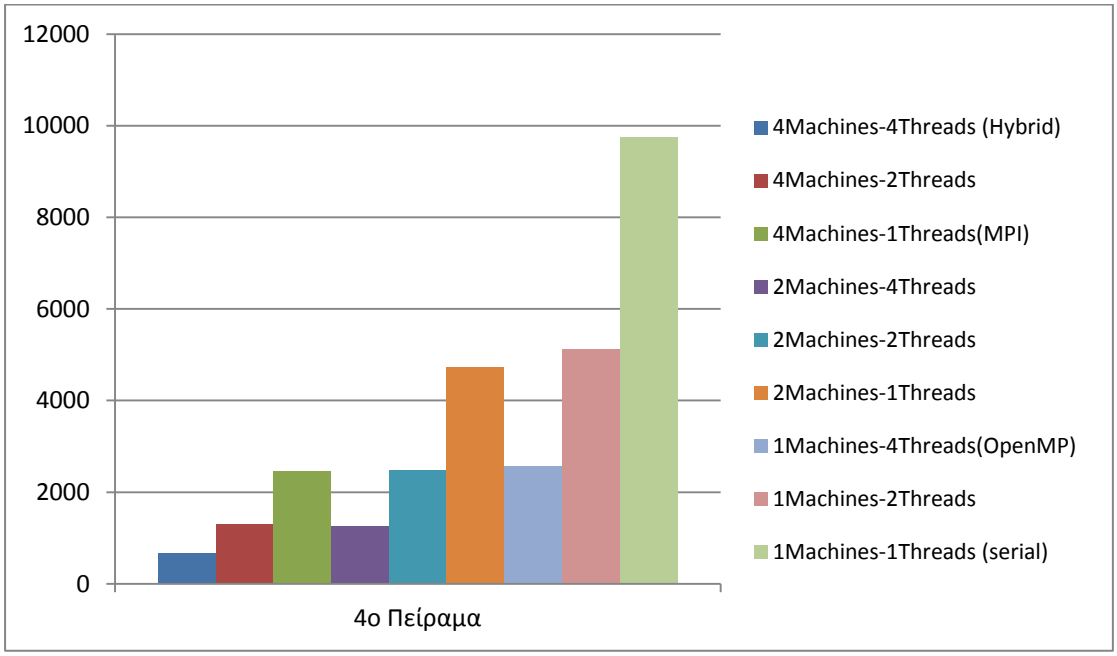


Γράφημα 2.3

4ο Πείραμα (Data Set : 1200000 αντικείμενα, διάσταση : 16, κέντρα : 10000)

Machines	Data Points	Dimension	Centroids	Threads	Iterations	Μέτρηση (sec)
4	**1200000**	**16**	**10000**	**4**	**937**	**65532,80736**
4	**1200000**	**16**	**10000**	**2**	**937**	**129600,5125**
4	**1200000**	**16**	**10000**	**1**	**937**	**246049,7058**
2	**1200000**	**16**	**10000**	**4**	**902**	**124807,688**
2	**1200000**	**16**	**10000**	**2**	**902**	**248757,8759**
2	**1200000**	**16**	**10000**	**1**	**902**	**492738,4343**
1	**1200000**	**16**	**10000**	**4**	**930**	**256990,2573**
1	**1200000**	**16**	**10000**	**2**	**930**	**511380,4668**
1	**1200000**	**16**	**10000**	**1**	**930**	**975622,1321**

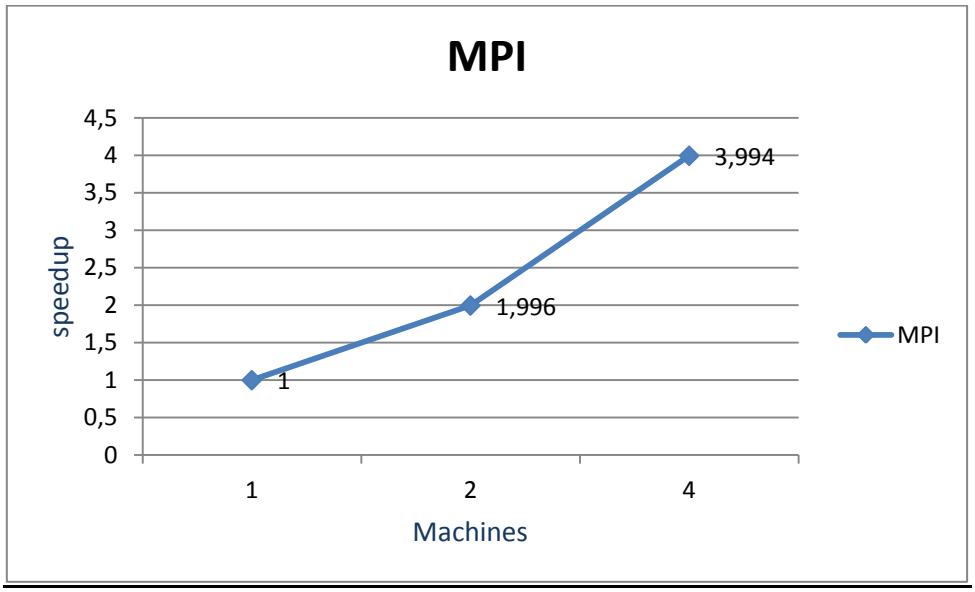
Πίνακας 2.4



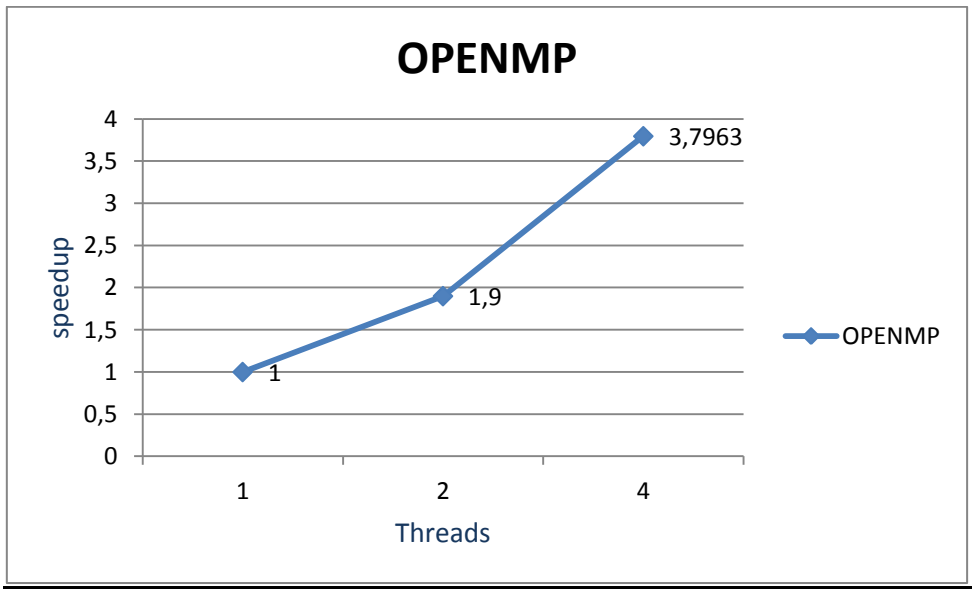
Γράφημα 2.4

7. Συμπεράσματα

Random initialization :



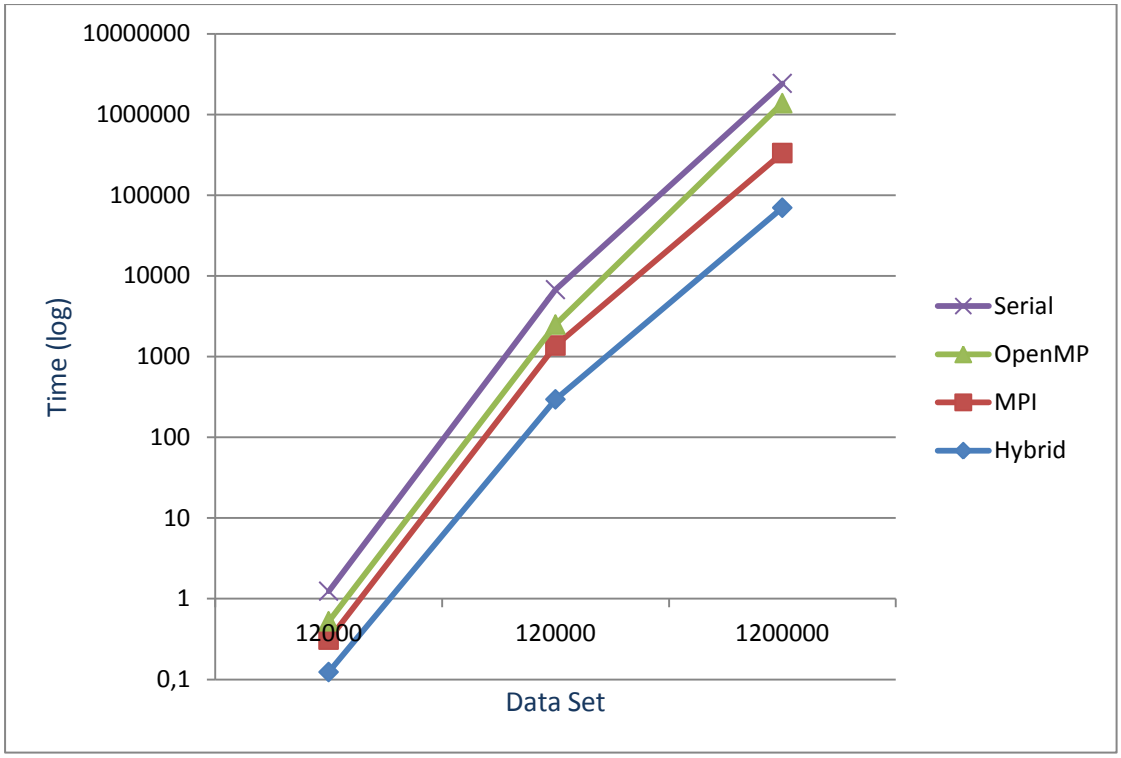
Γράφημα 3.1



Γράφημα 3.2

Στα γραφήματα *3.1* και *3.2* παρουσιάζεται το speedup για το τελευταίο dataset τόσο της MPI υλοποίησης όσο και της OpenMP. Το speedup είναι σχεδόν γραμμικό και στις δύο περιπτώσεις με μία μικρή υπεροχή της MPI υλοποίησης σε σχέση με την OpenMP. Αυτό μας οδηγεί στο συμπέρασμα ότι στον παρών αλγόριθμο, το κόστος συγχρονισμού στο εσωτερικό των κόμβων είναι υψηλότερο σε

σχέση με το κόστος επικοινωνίας μεταξύ των clusters. Βέβαια πρέπει να σημειωθεί ότι αν "στρεσαριστεί" το δίκτυο π.χ. αυξηθεί ο αριθμός των centroids η επίδοση του MPI θα πέσει σχέση με του OpenMP.Ακόμη στα συγκεκριμένα πειράματα τα μηχανήματα που σχηματίζουν τα clusters δεν έχουν μεγάλο latency επικοινωνίας δεδομένου ότι βρίσκονται πολύ κοντά. Έτσι δοκιμάζοντας μια άλλη τοπολογία ίσως να υπάρχει χειρότερη επίδοση το MPI.

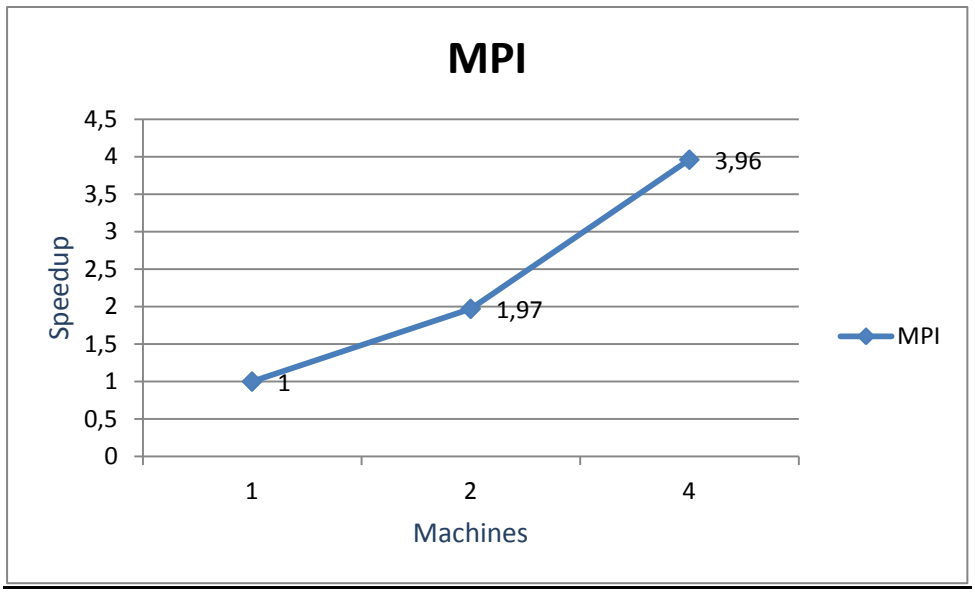


Γράφημα 3.3

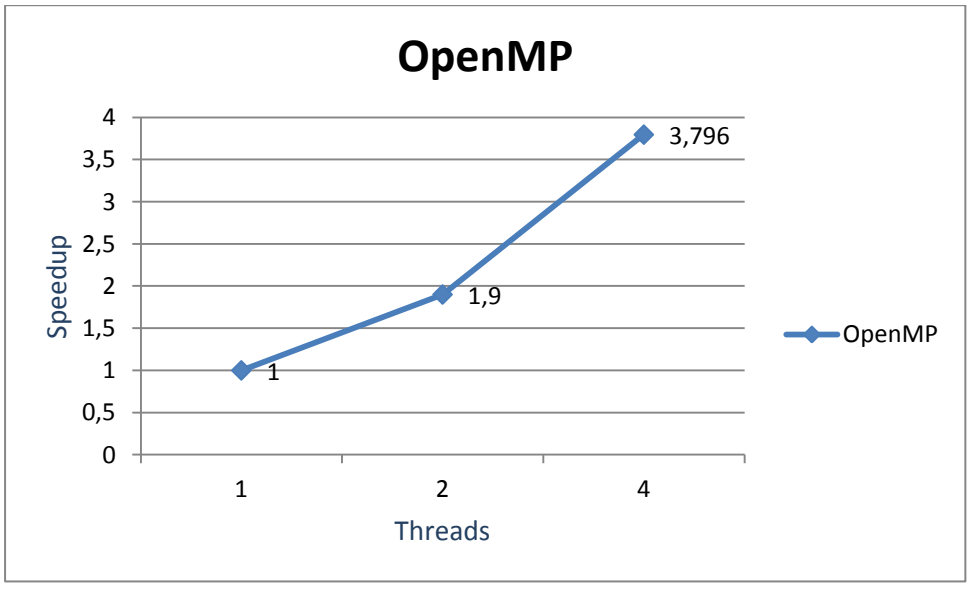
Στο παραπάνω γράφημα παρατίθενται συγκεντρωτικά οι μετρήσεις για τις τέσσερις διαφορετικές υλοποιήσεις (serial, pure OpenMP, pure MPI, Hybrid) για τα διαφορετικά data set.

Εάν θεωρήσουμε κάθετες νοητές γραμμές στο γράφημα, παρατηρούμε τόσο την σαφή υπεροχή της Hybrid υλοποίησης όσο και την κλιμάκωση που υπάρχει μεταξύ των dataset. Δηλαδή ο χρόνος εκτέλεσης της Hybrid σε κάθε περίπτωση είναι μικρότερος, τόσο των παράλληλων εκδοχών όσο και της σειριακής. Επιπρόσθετα όσο κλιμακώνουν τα dataset, η διαφορά μεταξύ των χρόνων εκτέλεσης μειώνεται αντίστοιχα.

KKZ initialization :

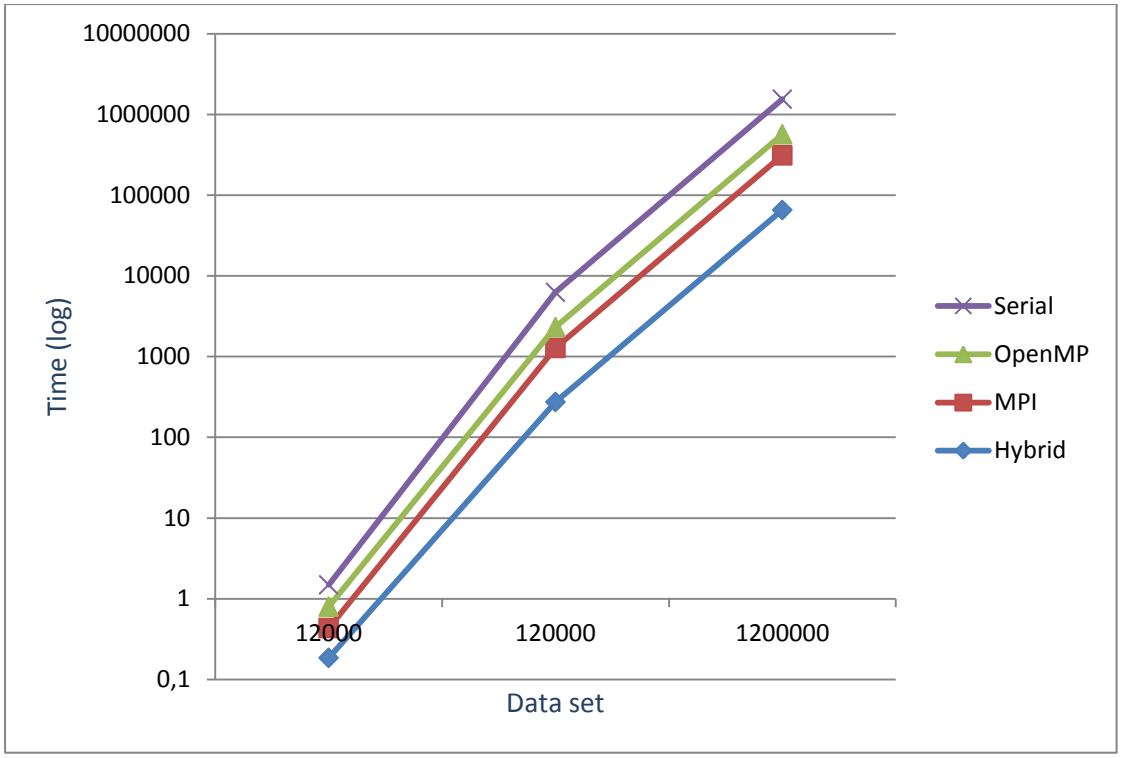


Γράφημα 3.4



Γράφημα 3.5

Στα γραφήματα *3.4* και *3.5* παρουσιάζεται το speedup για το τελευταίο dataset τόσο της MPI υλοποίησης όσο και της OpenMP. Τα συμπεράσματα είναι ακριβώς ίδια με την Random αρχικοποίησης καθώς η λογική του αλγορίθμου δεν αλλάζει.



Γράφημα 3.6

Ομοίως και για τον KKZ παρατίθενται συγκεντρωτικά οι μετρήσεις για τις τέσσερις διαφορετικές υλοποιήσεις (serial, pure OpenMP, pure MPI, Hybrid) για τα διαφορετικά data set.

Και πάλι, παρατηρούμε τόσο την σαφή υπεροχή της Hybrid υλοποίησης όσο και την κλιμάκωση που υπάρχει μεταξύ των dataset.

Σύγκριση Random Initialization με KKZ Initialization:

Machines	Data Points	Dimension	Centroids	Threads	Random Iter	Random Time	KKZ Iter	KKZ Time
4	120000	16	1024	4	**395**	295,333141	**370**	274,259329
4	120000	16	1024	2	**395**	567,409306	**370**	531,497309
4	120000	16	1024	1	**395**	1074,647152	**370**	1006,631509
2	120000	16	1024	4	**395**	572,110406	**330**	477,024125
2	120000	16	1024	2	**395**	1121,669505	**330**	937,0909789
2	120000	16	1024	1	**395**	2126,563475	**330**	1776,62265
1	120000	16	1024	4	**395**	1128,838505	**366**	1040,770364
1	120000	16	1024	2	**395**	2226,06437	**366**	2062,631796
1	120000	16	1024	1	**395**	4250,252535	**366**	3938,208678
4	1200000	16	10000	4	**999**	69869,023594	**937**	65532,80736
4	1200000	16	10000	2	**999**	138176,8818	**937**	129600,5125
4	1200000	16	10000	1	**999**	262330,4804	**937**	246049,7058
2	1200000	16	10000	4	**999**	138229,3753	**902**	124807,688
2	1200000	16	10000	2	**999**	275509,7255	**902**	248757,8759
2	1200000	16	10000	1	**999**	523845,825	**902**	472981,712
1	1200000	16	10000	4	**999**	276057,2764	**930**	256990,2573
1	1200000	16	10000	2	**999**	549321,5985	**930**	511380,4668
1	1200000	16	10000	1	**999**	1048007,267	**930**	975622,1321

Παρατηρούμε ότι αν και η εκτέλεση του KKZ παίρνει σαφώς μεγαλύτερο χρόνο από την τυχαία αρχικοποίηση οδηγεί σε καλύτερη τοποθέτηση και άρα λιγότερες επαναλήψεις τον k-means. Αυτό έχει ως συνέπεια λιγότερο χρόνο εκτέλεσης. Ακόμη σαν μελλοντική βελτίωση θα μπορούσε να παραλληλοποιηθεί ο KKZ ούτος ώστε να μειωθεί ο χρόνος εκτέλεσης του.

8. Βιβλιογραφία

[1] Katsavounidis, I., Kuo, C., Zhang, Z., 1994. A new initialization technique for generalised lloyd iteration. EEE Signal Processing Letters 1 (10), 144_146.

[2] Rolf Rabenseifner, Georg Hager, Gabriele Jost, "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," pdp, pp.427-436, 2009 Parallel, Distributed and Network-based Processing, 2009.

[3] Rabenseifner, Hager, Jost, "Hybrid MPI & OpenMP Parallel Programming", Summary from Tutorial M02 at SC10, November 15, 2010, New Orleans, LA, USA.

[4] S. Mohanavalli, S. M. Jaisakthi, and C. Aravindan, Strategies for Parallelizing KMeans Data Clustering Algorithm, In: Proceedings of International Conference on Advances in Information Technology and Mobile Communications, CCIS Vol. 147, Part 3, Springer-Verlag, pp. 427 - 430, 2011.

[5] A Tutorial on Clustering Algorithms,
Available at: http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/index.html

[6] Wikipedia, The free encyclopedia, "K-means clustering". Available at: http://en.wikipedia.org/wiki/K-means_clustering

[7] Wei-keng Liao, Electrical Engineering and Computer Science Department, Northwestern University, "Parallel K-Means Data Clustering". Available at: http://users.eecs.northwestern.edu/~wkliao/Kmeans/

[8] Tan, Steinbach, Kumar+ Ghosh, "The k-means algorithm",[Notes]. Available at: http://www.cs.uvm.edu/~xwu/kdd/Slides/Kmeans-ICDM06.pdf

[9] Princeton University, Princeton Engineering, "Clustering Algorithms K-means", [Class Notes]. Available at: http://www.cs.princeton.edu/courses/archive/spr08/cos435/Class_notes/clustering2_toPost.pdf