

# ΑΝΑΦΟΡΑ ΓΙΑ ΑΣΚΗΣΗ ΣΤΗΝ ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

**Ονοματεπώνυμο:** Μπάρλος Αριστείδης **ΑΜ:** 235839

**Ονοματεπώνυμο:** Ζήσης Αντρέας Παναγόπουλος **ΑΜ:** 235858

**Ονοματεπώνυμο:** Θανάσης Σπηλιωτακόπουλος **ΑΜ:** 236208

## Μεταγλώττιση και εκτέλεση των αρχείων

Μια καλή ιδέα πρέπει να αντιγράφεται συνεπώς για την εκτέλεση των αρχείων απλά φορτώνονται οι παράμετροι από το command line.

Συνεπώς, δεν χρησιμοποιείται η μεταβλητή περιβάλλοντος OMP\_NUM\_THREADS.

Ακολουθούν παραδείγματα για εκτέλεση ενός αρχείου (π.χ. Static) με μέγιστες βελτιστοποιήσεις:

Για παράδειγμα για *τρέξιμο ενός thread*:

```
gcc -O3 -fopenmp -Wall -Extra -o Static Static.cpp <Παράμετροι Γραμμής Εντολής>  
1
```

*Για το τρέξιμο 2 threads*

```
gcc -O3 -fopenmp -Wall -Extra -o Static Static.cpp <Παράμετροι Γραμμής Εντολής>  
2
```

κ.ο.κ.

Γενικά για να τρέχει το πρόγραμμα μετά την εκτέλεση όπως στο σειριακό πρόγραμμα μπαίνει ως όρισμα από τον χρήστη ο αριθμός των threads:

```
gcc -O3 -fopenmp -Wall -Extra -o Static Static.cpp <Παράμετροι> <Πλήθος  
threads>
```

Αυτό γίνεται για μεγαλύτερη ευκολία στο τρέξιμο, το γράψιμο και την διόρθωση των εντολών από σειριακό σε παράλληλο τρόπο. Κοινώς, ήταν πιο εύκολο να γίνει το παραπάνω από το να δηλώνεται ο αριθμός των threads και να γίνεται export όπως προτείνεται στο Παράρτημα 3.

Επίσης, να τονιστεί πως τα threads σε κάθε πρόγραμμα πρέπει να είναι 1, 2 ή 4. Αν δοθεί κάποιος άλλος αριθμός τότε θα βγει μήνυμα που ειδοποιεί για το όρισμα που παίρνει ο αριθμός. Αυτό συμβαίνει για να οριστεί όσο καλύτερα γίνεται το πρόγραμμα και να μην υπάρχει undefined συμπεριφορά.

Τέλος τα προγράμματα δημιουργήθηκαν και έτρεξαν σε λειτουργικό σύστημα ubuntu 18.04.2 σε VirtualBox με επεξεργαστή i7 και 4 πυρήνες για το λειτουργικό σύστημα εκ των οποίων και οι 4 δίνονται στο guest OS. Μια παρατήρηση είναι πως έμμεσα μπορεί να οριστεί ο αριθμός πυρήνων στο σύστημα αλλάζοντας των αριθμό των πυρήνων που δίνονται στο VirtualBox.

Για την άσκηση χρησιμοποιήθηκαν 4 πυρήνες κάτι που σημαίνει ένα νήμα ανά επεξεργαστή.

Όπως ίσως, είναι εμφανές ως τώρα έχουν γίνει αρκετές αλλαγές στον σειριακό κώδικα. Για τα προγράμματα που δημιουργήθηκαν έγιναν κάποιες αλλαγές για βελτίωση απόδοσης. Όλα όσα έγιναν αναφέρονται παρακάτω ξεχωριστά για κάθε πρόγραμμα.

## APXEIO Static.cpp

Το πρώτο κομμάτι που αφορά την δημιουργία των σύνθετων πολυγραμμών (χρησιμοποιώντας while) που αργότερα θα απλοποιηθούν δεν αλλάζει. Ωστόσο, υπάρχει και το όρισμα `argv[4]` που το τίθεται στην μεταβλητή `numberOfThreads` με την συνάρτηση `atoi` προ του βρόχου `while`.

Στη συνέχεια γίνεται έλεγχος ώστε το νούμερο `int numberOfThreads` να είναι 1,2 ή 4.

Εφόσον το `while` εκτελεστεί ο αριθμός των πολυγραμμών που υπάρχουν στην μεταβλητή `AllPolylines` θα είναι πάντα ίσος με τον αριθμό απλοποιημένων πολυγραμμών σημαίνει πως το `vector SimplifiedAllPolylines` θα έχει ίδιο μήκος με το `vector AllPolylines`.

Συνεπώς, ορίζεται μια μεταβλητή `numberOfLinesInFile`, η οποία μετρά το πλήθος του `vector` που έχει γεμίσει ως τώρα και θα χρησιμοποιηθεί για να προσδιοριστεί η κατανομή φορτίου στατικά αργότερα καθώς επίσης και το μήκος του `SimplifiedAllPolylines` με το `reserve`.

Επίσης, διαιρώντας την παραπάνω μεταβλητή με τον αριθμό νημάτων (που θα είναι 1,2 ή 4) προκύπτει το `workload_number` που είναι ο φόρτος εργασίας για κάθε νήμα. Αν το πλήθος γραμμών διαιρείται ακριβώς με το πλήθος. Αν δεν διαιρείται ακριβώς τότε το υπόλοιπο πρέπει να ανατεθεί με συγκεκριμένο τρόπο για να βγει το αποτέλεσμα σωστά.

Τέλος, χρησιμοποιώντας την μεταβλητή `numberOfLinesInFile`, προσδιορίζεται με `reserve` το μήκος του `vector`. Αυτό είναι σημαντικό από πλευράς απόδοσης γιατί δεν διαστέλλεται ποτέ το `vector SimplifiedAllPolylines` άρα ποτέ δεν χρειάζεται να γίνει υπολογισμός νέου μήκους στο `vector` καθώς επίσης και δεν χρειάζεται να γίνει δέσμευση χώρου αυτού πολλές φορές μέσα στην μνήμη ή κατά την διάρκεια των `push_back`. Επίσης η μνήμη είναι δεδομένο πως δεν θα κατακερματιστεί σε αντίθεση με την περίπτωση που θα γινόντουσαν `reallocations` θέσεις μνήμης (πιθανόν με μη συνεχόμενο τρόπο).

Έχοντας κάνει τα παραπάνω πλέον το πρόγραμμα εισέρχεται σε παράλληλη περιοχή.

Για κάθε s & e καθώς επίσης και total\_time που είναι ιδιωτικά δημιουργείται μια τοπική μεταβλητή που δεν είναι αρχικοποιημένη.

Επίσης, υπάρχει SimplifiedSubVector που είναι το vector για κάθε υποπίνακα και ανήκει από 1 σε κάθε thread. Εκεί θα μουν τα αποτελέσματα της σειριακής εκτέλεσης κάθε νήματος δηλαδή θα εισάγονται τα αποτελέσματα για κάθε απλοποίηση που κάνει κάθε thread. Σκεφτόμενοι λογικά εφόσον έχουμε static παραλληλοποίηση, κάθε SimplifiedSubVector θα είναι το 1/4 του AllSimplifiedPolylines (εκτός και αν έχουμε μονό αριθμό οπότε εκεί το πρόγραμμα όπως τονίζεται και στις διαφάνειες επιλύεται με αρκετούς διακριτούς τρόπους).

Εφόσον το αρχείο που ζητάει η άσκηση είναι συγκεκριμένο και το πλήθος των πολυγραμμών που περιέχει διαιρείται ακριβώς με το 4 έγινε στατική ανάλυση με **2 τρόπους στατικής ανάθεσης** όπως φαίνεται και στον κώδικα:

**A)** Με τον τρόπο που το openmp λειτουργεί και την εντολή

```
#pragma omp for schedule(static,workload_number+1) nowait
```

Το θετικό με αυτή την υλοποίηση είναι ότι σου λύνει τα χέρια και δεν χρειάζεται να ασχοληθεί κανείς με υπόλοιπα αν το σύνολο των γραμμών είναι περιττό. Το μόνο που χρειάζεται προσοχή είναι το chunk που κάθε νήμα θα εκτελέσει να είναι workload\_number + 1 ΑΝ υπάρχει υπόλοιπο(modulo!=0) στην διαίρεση numberOfLinesInFile/ numberOfThreads.

*Προσοχή!*

Σε περίπτωση που δεν γίνει κάτι τέτοιο και απλά μπει workload\_number οι τελευταίες modulo γραμμές θα εκτελεστούν τυχαία από κάποιο thread οδηγώντας σε ανασφαλή αποτελέσματα σε σχέση με το σειριακό πρόγραμμα.

**B)** Αυτή η υλοποίηση ήταν η πρώτη που έκανα και μου αρέσει περισσότερο γιατί παρόλο που θέλει λίγο περισσότερο κώδικα, χρησιμοποιώντας το id κάθε διεργασίας, το πλήθος των γραμμών μπορεί να σπάσει σε ομάδες που ανά 2 είναι ξένες μεταξύ τους δίνοντας έναν έξυπνο τρόπο υπολογισμού.

Αυτή η υλοποίηση όμως, έχει το αρνητικό ότι πρέπει πάντα να ανατίθεται χειροκίνητα από τον προγραμματιστή το υπόλοιπο της διαίρεσης numberOfThreadsInFile / numberOfThreads σε κάποιο νήμα (με κόστος επιπλέον γραμμές κώδικα) και ίσως και ο χρόνος να παίζει κάποιον ρόλο.

Από την άλλη με pragma omp single αποσυμφορούνται και στην συνέχεια το while οι γραμμές υπολογίζονται σε καλύτερο χρόνο από ότι αν υπολογίζονταν με pragma omp for ωστόσο αυτό δεν το έκανα γιατί τότε δεν έχω στατικό τρόπο ανάθεσης αλλά δυναμικό τρόπο ανάθεσης.

Ωστόσο το πρόγραμμα στο νήμα που αναλαμβάνει βαρύτερες σε μέγεθος γραμμές, δεν τρέχει σε καλύτερο χρόνο από το σειριακό πρόγραμμα γιατί οι γραμμές που μετράνε για υπολογισμό είναι οι τελευταίες.

Λόγω της ιδιομορφίας του αρχείου γραμμών που οι πρώτες γραμμές είναι ασήμαντες σε μέγεθος σε σχέση με τις τελευταίες έχουμε αρκετά περίεργα φαινόμενα.

Το πρώτο είναι ότι οι 3 επεξεργαστές από τους 4 λειτουργούν με τάξεις μικρότερους χρόνους από το σειριακό πρόγραμμα. Ο τέταρτος πυρήνας κάνει όμως τόσο χρόνο όσο και ο σειριακός αλγόριθμος παρόλο που δοκίμασα τα πάντα για 1 εβδομάδα.

Αυτό οδηγεί στο συμπέρασμα πως μάλλον οι τελευταίες 4000 είναι όλο το αρχείο ενώ οι άλλες είναι πολύ ελαφρύτερες.

Στην περίπτωση 2 το range δεν μπορεί να τεθεί ως `workload_number+1` γιατί θα οδηγήσει σε `out_of_range` σφάλμα. Μια λύση για αυτό το ζήτημα θα ήταν το try-catch μπλοκ αλλά αυτό θα υποβάθμιζε την ταχύτητα του προγράμματος.

Επίσης, επειδή ζητείται σειριακή ανάθεση το thread 3 κάνει παραπάνω δουλειά κατά `workload_number + 2`.

## ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ

Οι χρόνοι εκτέλεσης δεν είναι ίδιοι αν χρησιμοποιηθεί το `nowait` clause στην περίπτωση Α ενώ στην Β δεν χρειάζεται ούτε `nowait` clause.

Κάθε νήμα ανάλογα με το πλήθος και το μήκος των γραμμών που αναλαμβάνει κάνει και τον ανάλογο χρόνο. Βάσει των παραπάνω επειδή οι τελευταίες επαναλήψεις αποτελούνται από πολύ περισσότερες γραμμές σε σχέση με τις πρώτες, το νήμα 3 για την περίπτωση με τα 4 threads και το νήμα 1 για την περίπτωση με 2 threads δαπανούν περισσότερο χρόνο από τα υπόλοιπα.

Αυτό οδηγεί στο συμπέρασμα πως ο στατικός τρόπος ανάθεσης δεν είναι ικανοποιητικός για αυτό το αρχείο γιατί η «δουλειά δεν είναι ισοκατανεμημένη» αλλά όλη μπροστά για το τελευταίο νήμα.

Για τον υπολογισμό της χρονοβελτίωσης χρησιμοποιήθηκε ο τύπος:

$$\text{speedup} = (\text{Παλιός Σειριακός Χρόνος}) / (\text{Νέος Παράλληλος Χρόνος})$$

Δυστυχώς εδώ δεν υπάρχει σημαντική ή και καθόλου χρονοβελτίωση. Φαίνεται πως οι πρώτες γραμμές είναι παντελώς ασήμαντες σε μέγεθος σε σχέση με τις τελευταίες από πλευράς υπολογισμών αν αφήσω χωρίς βελτιστοποιήσεις των κώδικα και τον αντιγράψω ως έχει.

Επίσης υπάρχει overhead που οφείλεται στην παραλληλοποίηση με `openmp` και έχει σχέση με την μνήμη, πως είναι κατασκευασμένη (δηλαδή αν η αρχιτεκτονική είναι πραγματικά παράλληλη) ή και στους χρόνους που δαπανώνται για να αποφασιστεί πιο νήμα θα είναι το σημαντικό.

## **ΒΕΛΤΙΩΣΕΙΣ ΣΤΟΝ ΚΩΔΙΚΑ**

Οι βελτιστοποιήσεις-παρεμβάσεις για καλύτερη απόδοση στον κώδικα που έγιναν είναι αρκετές με βασικές τις εξής:

A) Όλες οι row() συναρτήσεις αντικαταστάθηκαν με πιο γρήγορες είτε την sqrt είτε με απλές πράξεις για τον υπολογισμό της PerpendicularDistance .

B) Αντί να γίνονται πολλές πράξεις μέσα στον loop βρόχο γίνεται μόνο μια - η εκτέλεση της συνάρτησης RamerDouglasPecker όπου δέχεται μεταβλητές με reference και φτιάχνει στον πίνακα SimplifiedAllPolylines όσα χρειάζονται στην θέση που πρέπει.

Για το πείραμα χρησιμοποιήθηκαν 2 επεξεργαστές και τα νήματα στην περίπτωση 1 έχουν workload\_number ενώ στην 2 το υπόλοιπο είναι 0 (άρα δεν εκτελείται ποτέ ή if από κάτω). Αυτό μπορεί να συνεπάγεται κάποιο overhead στην περίπτωση που διαβάζεται και η συνθήκη αλλά δεν είναι σημαντικό.

### **Εν κατακλείδι:**

Με βελτιστοποιήσεις στον κώδικα τα αποτελέσματα είναι πολύ καλύτερα από τον κώδικα που δίνεται. Αυτό συμβαίνει γιατί η row συνάρτηση καθυστερεί μέχρι και 20 φορές περισσότερο από την sqrt και γιατί η πράξη  $x*x$  είναι πολύ γρηγορότερη από την πράξη row(x,2).

Επίσης, επέλεξα να βελτιστοποιήσω τον κώδικα δίνοντας θέσεις των βασικών πινάκων ως απευθείας ορίσματα γλιτώνοντας πράξεις ανάθεσης ή καθαρισμού των vectors.

Επίσης, να διευκρινιστεί πως και ο τρόπος A και ο Τρόπος B είναι ανεξάρτητοι μεταξύ τους καθώς δεν γίνονται ενθέσεις στον πίνακα παρά μόνο αντικαταστάσεις στοιχείων επομένως θα μπορούσαν να τρέξουν και παρέα.

Απλά στον B Τρόπο συνήθως ο χρόνος εκτέλεσης είναι ΣΥΝΗΘΩΣ χαμηλότερος ίσως γιατί ο SimplifiedAllPolylines έχει ήδη σωστές τιμές, επομένως γίνεται αντικατάσταση της ίδιας τιμής άρα δεν υπάρχουν χρόνοι υπολογισμού νέων μεγεθών.

Για τα πειράματα που έγιναν, κάθε πρόγραμμα έτρεξε 3 φορές και ο μέσος όρος των αποτελεσμάτων είναι το αποτέλεσμα στην αναφορά (για πιο αξιόπιστα αποτελέσματα). Στον Στατικό τρόπο έτρεξαν οι δύο διαφορετικοί αλγόριθμοι κάθε ένας ξεχωριστά και υπάρχουν και οι 2 στην αναφορά.

**Οι πίνακες με τους χρόνους καθώς και τα διαγράμματα χρονοβελτίωσης, βρίσκονται στο τέλος της παρουσίασης των αρχείων και της προγραμματιστικής λογικής που ακολουθήθηκε, για την επίλυση της άσκησης.**

**APXEIO Dynamic.cpp**

Όπως ίσως έγινε σαφές και από το Static πρόγραμμα πλέον δεν υπάρχει εξάρτηση από δεδομένα. Συνεπώς, ο κώδικας δεν άλλαξε ιδιαίτερα εκτός από την λέξη dynamic που αντικατέστησε την static.

Για κομμάτι επεξεργάσιμων γραμμών δόθηκαν διάφορες τιμές και τελικά κρίθηκε προτιμητέο να παραδοθεί η τιμή που δίνει τον καλύτερο χρόνο. Αυτές οι τιμές πρέπει να ελεγχθούν πειραματικά και να καθορίσουν ένα κομμάτι μνήμης αρκετά μεγάλο ώστε κάθε πυρήνας να παίρνει ίσα κατανεμημένο φορτίο δουλειάς αλλά ταυτόχρονα το overhead επικοινωνίας επεξεργαστών να ελαχιστοποιείται.

Εδώ αξιοποιήθηκαν οι εντολές του openmp και κυρίως η εντολή:

**#pragma omp for schedule(dynamic,mem\_chunk) nowait**

Όπου mem\_chunk είναι το πλήθος γραμμών που κάθε νήμα αναλαμβάνει να εκτελέσει σειριακά και μόλις το πλήθος γραμμών τελειώσει, το νήμα να φύγει από το έμμεσο φράγμα (implicit barrier) της for και να τερματίσει εκτυπώνοντας τον χρόνο του.

Αυτή η εντολή τοποθετείται φυσικά εντός της openmp εντολής **#pragma omp parallel num\_threads(number\_of\_threads)**

Όπου number\_of\_threads είναι ο integer αριθμός των νημάτων, num\_threads() είναι εντολή που προσδιορίζει πόσα threads θα πάρει η parallel εντολή.

*Προσοχή!*

Στην άσκηση δεν παίζει ρόλο αλλά το πόσα νήματα ζητάμε και πόσα παίρνουμε στην πραγματικότητα, μπορεί να μην ταυτίζονται ποσοτικά και πιο συγκεκριμένα τα threads που λαμβάνει το πρόγραμμα να είναι πιο λίγα από αυτά που ζητούται.

Εδώ βέβαια δεν υπάρχει τέτοιο πρόβλημα γιατί ένα νήμα ανά επεξεργαστή είναι το ελάχιστο.

Επίσης, εφόσον στον κώδικα που εκτελείται δεν υπάρχει ανάγκη για συγχρονισμό καλό είναι να προσδιοριστεί πειραματικά μόνο το πόσο ακριβώς μεγάλο πρέπει να είναι το κομμάτι που επεξεργάζεται ένα νήμα για να έχουμε τους μικρότερους δυνατούς χρόνους οι οποίοι θα είναι ίσα κατανεμημένοι στα διάφορα νήματα.

Διαλέγοντας ως κομμάτι το **mem\_chunk=1** παρατηρείται η καλύτερη ισοκατανομή του φορτίου μεταξύ των νημάτων και οι πιο μικρές διαφορές στους χρόνους τους.

Ιδανικά θα θέλαμε όσο το δυνατόν μεγαλύτερο κομμάτι για να τελειώνει το μέγεθος των γραμμών γρηγορότερα και να λαμβάνονται λιγότερες αποφάσεις για αναθέσεις σε κάθε thread στους χρόνους εκτέλεσης. Ωστόσο η μορφή του αρχείου που δίνεται δεν επιτρέπει κάτι καλύτερο.

## **ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ**

Πλέον η δουλειά είναι σχεδόν ίσα κατανεμημένη, συνεπώς, κάθε πυρήνας αναλαμβάνει ανάλογο φορτίο ενώ στην στατική ανάθεση αυτό δεν συμβαίνει.

Αν πάρουμε το ΑΘΡΟΙΣΜΑ όλων των χρόνων από τα δυναμικά νήματα παρόλο που προκύπτει μια σχετική ισοκατανομή στους χρόνους, ο χρόνος για δυναμική εκτέλεση είναι μεγαλύτερος του αντίστοιχου στατικού τρόπου ανάθεσης.

Αυτό το φαινόμενο υφίσταται εξαιτίας του χρόνου που δαπανάται για τον συγχρονισμό των νημάτων κατά των χρόνο εκτέλεσης.

Ωστόσο το πρόβλημα που το thread 3 αναλαμβάνει πολλαπλάσια δουλειά από τα υπόλοιπα νήματα δεν υπάρχει πλέον και πλέον παρατηρείται και χρονικά μεγαλύτερη ταχύτητα εκτέλεσης του προγράμματος παρά τον χρόνο που δαπανάται για την επικοινωνία ενώ για 1 νήμα ο χρόνος εκτέλεσης δεν αλλάζει σημαντικά.

## ΣΥΜΠΕΡΑΣΜΑΤΑ

Οι χρόνοι εκτέλεσης επειδή το αρχείο δεν έχει ίσα κατανεμημένο φόρτο εργασίας είναι καλύτεροι από τον στατικό τρόπο ανάθεσης, σε αυτό το αρχείο, ωστόσο αν αθροίσουμε τους χρόνους εκτέλεσης θα βγαίνουν παραπάνω από τον στατικό τρόπο ανάθεσης εξαιτίας του χρόνου που δαπανάται για επικοινωνία.

## APXEIO Task1.cpp

Η παραλληλοποίηση με χρήση tasks σε αυτή την περίπτωση σπάει σε 2 κομμάτια που κάθε κομμάτι έχει τις δικές του εντολές. Το ένα κομμάτι εντολών βρίσκεται στο Main Calculation ενώ το άλλο κομμάτι εντός του RamerDouglasPecker εντός της if(dmax>epsilon).

Στο κομμάτι που είναι στην main χρησιμοποιείται αρχικά η εντολή:

**#pragma omp parallel num\_threads(number\_of\_threads)**

Που ζητά έναν αριθμό νημάτων ίσο με τον αριθμό 1,2 ή 4 που θέλουμε να δούμε πως τρέχει.

Ένθετα είναι μια εντολή:

**#pragma omp single**

Η οποία δίνει σε ένα τυχαίο νήμα την εκτέλεση της παρακάτω ένθετης εντός single εντολής:

**#pragma omp taskloop untied num\_tasks(100)**

Η εντολή αυτή χωρίζει τον κώδικα του for που ακολουθεί σε tasks όπου ισχύει πως κάθε task περιλαμβάνει μια επανάληψη του βρόχου. Θα μπορούσε να περιλαμβάνει περισσότερες εκμεταλλεζόμενοι καλύτερα την παραλληλία με την συνθήκη grainsize(πλήθος επαναλήψεων) αλλά λόγω της ιδιομορφίας του αρχείου που δεν περιλαμβάνει ισοσταθμισμένο φόρτο εργασίας αυτό δεν μπορεί να οδηγήσει σε καλύτερα αποτελέσματα.

Το num\_tasks ορίζει έναν μέγιστο αριθμό tasks που μπορούν να δημιουργηθούν.

Το untied υπάρχει για να αποφευχθεί φαινόμενο starvation όπου όλα τα threads έχουν τελειώσει τους υπολογισμούς τους και περιμένουν το thread δημιουργό tasks να

δώσει περαιτέρω εργασίες ενώ το νήμα δημιουργός είναι απασχολημένο κάνοντας υπολογισμούς σε ένα βρόχο. Επίσης, αυξάνεται η ταχύτητα εκτέλεσης.

Στο untied τα tasks-επαναλήψεις που δημιουργούνται είναι ανεξάρτητα του νήματος που τα δημιούργησε και μπορούν να εκτελεστούν από οποιοδήποτε νήμα, δηλαδή στην συγκεκριμένη περίπτωση αποφεύγεται το παραπάνω φαινόμενο.

Τέλος, να σημειωθεί πως η επανάληψη εκτελείται αντίστροφα γιατί το polylines\_large αρχείο που δίνεται έχει τους μεγάλους χρονικά υπολογισμούς στο τέλος του.

Εκτελώντας τον βρόχο ανάποδα όλα τα νήματα πιάνουν κατευθείαν δουλειά και καθώς πηγαίνουμε από τις πιο χρονοβόρες γραμμές στις λιγότερο χρονοβόρες τα νήματα μπορούν να απασχολούνται «τελειώνοντας» τους πιο βαρείς υπολογισμούς από το να περιμένουν προς το τέλος του προγράμματος χρησιμοποιώντας περισσότερη παράλληλη εκτέλεση υπολογισμών. Αυτό ίσως και να μην επηρεάζει καθόλου το πρόγραμμα καθώς δεν εξετάστηκαν και οι 2 περιπτώσεις.

Τώρα όσον αφορά το κομμάτι που βρίσκεται εντός του RamerDouglasPecker, πάρθηκαν κάποιες πρωτοβουλίες έπειτα από την παραδοχή πως δεν μας νοιάζει ποιο task θα γεμίσει τον πίνακα αρκεί ο πίνακας να γεμίσει σωστά.

Συνεπώς, έθεσα μαζί με τον υπολογισμό της πρώτης υπό-γραμμής να γεμίζει και ο πίνακας. Στην συνέχεια καθώς και το νήμα που ασχολείται με τον υπολογισμό της τελευταίας πολυγραμμής περιμένουν όλους τους απογόνους-tasks να τερματίσουν και μετά γίνεται εισαγωγή και της δεύτερης απλοποιημένης πολυγραμμής στον ήδη υπάρχον πίνακα.

Πιο συγκεκριμένα οι εντολές που χρησιμοποιήθηκαν είναι:

**#pragma omp taskgroup**

Όπου ότι ενθέτεται μέσα στα πλαίσια επιτρέπει αναδρομική κλήση και εκτέλεση.

Αν και κάνουν την ίδια δουλειά με την taskwait, εδώ η taskwait δεν θα μπορούσε να χρησιμοποιηθεί.

Η διαφορά των 2 είναι πως η taskgroup περιμένει να εκτελεστεί κάθε απόγονος διαδικασία ενώ η taskwait περιμένει να εκτελεστεί μόνο το παιδί της διαδικασίας.

Τέλος, να επισημανθεί ότι στον κώδικα έγινε προσπάθεια κάποιο task να είναι όσο το δυνατόν πιο απαιτητικό και χρονοβόρο καθώς αυτό είναι το καλύτερο από πλευράς απόδοσης. Λίγα και “βαριά” tasks παρά πολλά και μικρά λόγω overhead δημιουργίας και διαχείρισης των tasks.

Μια δεύτερη εντολή είναι η

**#pragma omp task untied mergeable final(endRDP<30) shared(out)**

Όπου το final(endRDP<30) υπάρχει για να υποδηλώνει ότι αν ο πίνακας έχει λιγότερο μήκος από 30 τότε θεωρείται μικρός ο υπολογισμός και δεν δημιουργείται



νέο task. Αυτό γίνεται για να περικοπεί το overhead δημιουργίας και εισαγωγής-εξαγωγής από την ουρά που μπαίνουν τα tasks για εκτέλεση καθώς ο υπολογισμός θεωρείται (και είναι) αρκετά μικρός. Ωστόσο, είναι μια τιμή που την πήρα με πειράματα και η καλύτερη τιμή μπορεί να είναι κάποια άλλη τιμή.

Το shared(out) χρησιμεύει για σωστή εξαγωγή αποτελεσμάτων καθώς κάθε εντολή από το πιο εξωτερικό block κώδικα στο task block δεν παίρνει τα αποτελέσματα της επεξεργασίας αλλά τα αποτελέσματα που υπήρχαν πριν την επεξεργασία.

Το mergeable σημαίνει πως το περιβάλλον δεδομένων είναι κοινό με του πατέρα που δημιουργεί το task. Αυτό γίνεται για να αποφευχθεί η αναδρομική κλήση και δημιουργία πολλών μεταβλητών πολλές φορές χωρίς να χρειάζεται. Θα μπορούσε και να μην χρησιμοποιηθεί καθόλου. Επίσης, υπάρχει untied άρα ακόμα και αν το νήμα την κάνει suspend κάποιο άλλο νήμα θα μπορεί να την εκτελέσει.

Τα παραπάνω χρησιμοποιήθηκαν γιατί λόγω αποτελεσμάτων δίνεται καλύτερη απόδοση. Επίσης, κρίσιμος παράγοντας για ταχύτητα ήταν και η αναδιοργάνωση των εντολών εντός του taskgroup. Πιο συγκεκριμένα πρώτα ξυπνάει ένα task το νήμα δημιουργός και στην συνέχεια συνεχίζει την εκτέλεση των πιο κάτω εντολών. Έτσι επιτυγχάνεται καλύτερη παραλληλία από το να εκτελεί το νήμα-δημιουργός την αναδρομική συνάρτηση RamerDouglasPecker και στην συνέχεια να φτιάχνει task και να γίνεται suspend μέχρι να τελειώσει και το task.

Τέλος σε σχέση με το Dynamic είναι μεγαλύτερος ο χρόνος γιατί το νήμα δημιουργός tasks επιλύει μόνος του το πρόβλημα αν  $endRDP < 30$  ενώ όλα τα υπόλοιπα tasks περιμένουν.

## APXEIO Task2.cpp

Στο αρχείο task2 υπήρξε μια απορία σε σχέση με την εκφώνηση. Υπήρξε απορία αν η άσκηση ζητούσε να δημιουργείται από ένα task ΓΙΑ ΚΑΘΕ ΜΙΑ εργασία ή θα δημιουργούνταν ένα task ΚΑΙ ΤΙΣ 2 ΕΡΓΑΣΙΕΣ ΜΑΖΙ. Αποφασίστηκε πως αυτό που εννοούσε η άσκηση ήταν πως θα δημιουργείται ένα task για την απλοποίηση μιας πολυγραμμής και ένα task για την απλοποίηση της άλλης πολυγραμμής.

Το θέμα που προέκυψε είναι πως τα tasks και ο χρόνος που παίρνει για να δημιουργηθούν και να τρέξουν είναι πολύ μεγάλος λόγω του overhead που προκύπτει από την δημιουργία και διαχείριση των 2 tasks αντί μιας που υπήρχε στην προηγούμενη άσκηση από τα νήματα που διαχειρίζονται την εκτέλεση του κώδικα καθώς επίσης και την αποθήκευση ή εξαγωγή από την ουρά εργασιών 'καθώς πλέον δημιουργείται δουλειά για 2 άλλα νήματα εκτός του δημιουργού.

Επομένως καθώς το όντως έχει καλό scalability ο χρόνος που κάνει να εκτελεστεί είναι λίγος παραπάνω από τον χρόνο που κάνει για να εκτελεστεί σειριακά ακόμα και με cutoff. Η λύση στο πρόβλημα αυτό είναι να μειωθεί ο αριθμός των εργασιών όσο το δυνατόν είναι δυνατόν και φυσικά να χρησιμοποιηθεί ένα κατώφλι (κάτι που έγινε) και παρουσιάζει καλύτερους χρόνους.

Το συμπέρασμα από αυτό το τελευταίο πείραμα είναι ότι

A) Οι εργασίες θέλουν κάποιον χρόνο για δημιουργία και αποθήκευση-εξαγωγή από την ουρά εργασιών (task que) κάτι που προσθέτει overhead συνεπώς, δεν είναι πανάκεια και από κάποια στιγμή και μετά υπάρχει υποβάθμιση στο πρόγραμμα και στον χρόνο εκτέλεσης.

B) Επομένως, σημαντικό είναι πως οι εργασίες λόγω του παραπάνω να είναι βαριές γιατί κοστίζουν σε χρόνο και ενέργεια.

Στο πρόγραμμα έθεσα έναν μέγιστο αριθμό εργασιών που μπορεί κάθε στιγμή να φτιαχτούν και το πρόγραμμα δουλεύει αρκετά καλά.

Ωστόσο, αυτό δουλεύει καλύτερα στο σύστημα το δικό μου με 4 επεξεργαστές. Σε κάποιο άλλο σύστημα το όριο μπορεί να ήταν διαφορετικό ανάλογα με την επεξεργαστική ισχύ και το πλήθος των επεξεργαστών.

Επειδή το πρόβλημα είναι αναδρομικό χρήσιμο στάθηκε τόσο το mergeable όσο και το final που χρησιμοποιήθηκε για να περικόπτεται η δημιουργία ενός πλήθους εργασιών για τις οποίες το overhead ξεπερνά την ανάγκη για παραλληλοποίηση.

Οι εντολές που χρησιμοποιήθηκαν όπως και στην προηγούμενη περίπτωση ήταν σε 2 διαφορετικά μέρη.

Στην MAIN συνάρτηση όπου χρησιμοποιήθηκε το μπλοκ κώδικα:

```
#pragma omp parallel num_threads(number_of_threads)
{
    #pragma omp single
    {
        #pragma omp taskloop untied mergeable num_tasks(100)
        for (it1 = numberOfLinesInFile-1; it1 >= 0; --it1)
        {
            RamerDouglasPeucker(AllPolylines[it1], epsilon,
SimplifiedAllPolylines[it1]);
        }
    }
}
```

Αυτό το κομμάτι όπως και πριν ζητά έναν αριθμό νημάτων που επιλέγουμε εμείς.

Ένα από αυτά τα νήματα (όποιο προλάβει) γίνεται νήμα-δημιουργός αποστολών.

Στην συνέχεια για παραλληλία υπάρχει η εντολή με το taskloop που ορίζει πως μέχρι 100 tasks μπορούν να υπάρχουν.

Στην συνέχεια εκτελείται η for με ανάποδη σειρά έτσι ώστε τα νήματα να μην περιμένουν αφού υπάρχει ανισορροπία στην δουλειά που απαιτείται να γίνει.

Τέλος εκτελείται ο αλγόριθμος.

Το άλλο μπλόκ εντολών `openmp` είναι η εξής ακολουθία εντολών η οποία έγινε προσπάθεια να κρατηθεί με την αρχική σειρά αλλά με όσο το δυνατόν καλύτερη απόδοση και φυσικά με γνώμονα να βγαίνουν σωστά τα αποτελέσματα:

```
#pragma omp taskgroup
{
    #pragma omp task untied mergeable final(endRDP<30) shared(out)
    {
        vector<Point> firstLine(pointList.begin(), pointList.begin() + index + 1);
        RamerDouglasPeucker(firstLine, epsilon, recResults1);
        out.assign(recResults1.begin(), recResults1.end() - 1);
    }
    #pragma omp task untied final(endRDP<30) shared(recResults2)
    {
        vector<Point> lastLine(pointList.begin() + index, pointList.end());
        RamerDouglasPeucker(lastLine, epsilon, recResults2);
    }
}
out.insert(out.end(), recResults2.begin(), recResults2.end());
```

Εδώ λόγω αναδρομικής κλήσης για συγχρονισμό των εργασιών επιλέχθηκε το `taskgroup` και όχι το `taskwait` καθώς το `taskgroup` θα συγχρονίζει τις αποστολές και όλους τους απογόνους των αποστολών ενώ το `taskwait` συγχρονίζει μόνο τις τωρινές αποστολές και τα παιδιά τους.

Στην συνέχεια, η γραμμή δημιουργεί ένα `task` το οποίο δεν συνδέεται με το νήμα που ξεκίνησε να την εκτελεί αλλά μπορεί να εκτελεστεί από οποιοδήποτε νήμα ακόμα και αν γίνει `suspend` για κάποιο χρονικό διάστημα.

Το ίδιο συμβαίνει και με τις παρακάτω γραμμές.

Οι εντολές που ενθέτονται εντός των `tasks` είναι η αρχικοποίηση με το μισά σωστά αποτελέσματα του `shared` πίνακα `out`, καθώς επίσης και με τον υπολογισμό των αποτελεσμάτων για σωστή εισαγωγή που βρίσκεται εκτός του `taskgroup`.

Οι μεταβλητές που είναι `shared` χρησιμοποιούνται για να περαστούν αποτελέσματα εκτός των `tasks`. Αυτή η υλοποίηση είναι η καλύτερη σκέψη που έγινε καθώς αν υπήρχε μόνο ο αλγόριθμος `RamerDouglasPecker` θα σπαταλιούνταν περαιτέρω χρόνος και επεξεργαστική ισχύς για να αντιγραφούν οι `vectors` ως `firstprivate vectors` εντός του `task` (την εξ'ορισμού δηλαδή μέθοδο εισαγωγής δεδομένων αν ο

προγραμματιστής δεν δηλώσει τι θέλει να είναι κάθε μεταβλητή που θα χρησιμοποιήσει η task). Αυτό έγινε επίσης για να μπορεί να εξαχθεί αποτέλεσμα που θα χρησιμοποιηθεί εκτός του task και αυτό αφορά τον μισό out πίνακα που θα βγει από το task και τον recResult2 που θα περιλαμβάνει το δεύτερη μισή απλοποιημένη πολυγραμμή. Τέλος, να τονιστεί πως αν δεν υπήρχε περικοπή tasks οι χρόνοι που θα έβγαιναν θα ήταν απαίσιои τόσο σε αυτή όσο και στην υλοποίηση του task1.

Επίσης, είναι δεδομένο πως εξαιτίας της δημιουργίας νέου task για απλοποίηση πολυγραμμών ο χρόνος που δίνει αυτή η υλοποίηση θα είναι πάντα μεγαλύτερος από τον χρόνο προηγουμένως στο task1 λόγω παραπάνω overhead εξαιτίας των επιπλέον tasks. Τέλος, ο χρόνος για την περίπτωση  $\epsilon=0,1$  αν και γραμμικός για  $\text{numberOfThreads}=2$  είναι μεγαλύτερος από τον χρόνο σειριακής εκτέλεσης.

## ΠΙΝΑΚΕΣ ΛΗΨΗΣ ΣΤΙΓΜΙΟΤΥΠΩΝ

### Περίπτωση 1: -00

Ο πρώτος πίνακας μετράει για τους αλγορίθμους χωρίς διανυσματοποιήσεις αλλά με βελτιστοποιημένους κώδικες. Χρησιμοποιήθηκαν 4 επεξεργαστές όταν έτρεξαν τα προγράμματα για κάθε περίπτωση ακόμα και στον σειριακό αλγόριθμο.

Παρατίθεται ωστόσο και ο σειριακός κώδικας χωρίς χρονοβελτιώσεις.

Ο σειριακός τρέχει με ένα νήμα αλλά ελάχιστα πιο γρήγορα από τον αντίστοιχα παραλληλοποιημένο κώδικα με ένα νήμα γιατί δεν σπαταλάται καθόλου χρόνος για να αποφασιστεί πιο νήμα θα τρέξει τον αλγόριθμο καθώς επίσης και να ερμηνευτούν άλλες εντολές του openmp που πιθανόν να υπάρχουν.

Επίσης, είναι σημαντικό να τονιστεί πως όσο το  $\epsilon$  ψιλον αλλάζει και γίνεται μικρότερο τόσο πιο καλός είναι ο παράλληλος αλγόριθμος. Αυτό γίνεται γιατί υπάρχει αύξηση των απλοποιήσεων που εκτελούνται λόγω  $\epsilon$ .

Επίσης κατά τον δυναμικό τρόπο ανάθεσης έγιναν οι μετρήσεις (τουλάχιστον 3) και κάθε φορά πάρθηκε ο μέσος όρος της μεγαλύτερης εκτέλεσης καθώς γίνεται η υπόθεση πως αυτή κλείνει την εκτέλεση του προγράμματος.

### Υπο-περίπτωση 0,1

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	16,588909	-	-
Σειριακό RDP (με βελτιώσεις)	4,447981	-	-
Static1(pragmа-for)	4,634694	4,709227	4,346648
Static2(manually)	4,559677	4,59138	4,335014
Dynamic	4,578644	2,383793	1,576631
Task1	6,063132	3,953315	3,274458
Task2	7,684978	5,189318	3,980419

### Υπο-περίπτωση 0,01

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP	30,228621	-	-
Σειριακό RDP (με βελτιώσεις)	9,265661	-	-
Static1(pragma-for)	9,339424	9,187917	8,673083
Static2(manually)	9,198690	9,103887	8,653654
Dynamic	9,097272	4,843205	3,133437
Task1	12,440672	7,236401	5,469120
Task2	16,002669	9,071965	7,119844

### Υπο-περίπτωση 0,001

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	46,940106	-	-
Σειριακό RDP (με βελτιώσεις)	18,315252	-	-
Static1(pragma-for)	18,545448	18,550356	16,948339
Static2(manually)	18,512917	17,998176	16,737559
Dynamic	18,244306	9,625924	6,324976
Task1	23,985495	14,245906	9,938602
Task2	32,483732	16,890357	12,765851

### Υπο-περίπτωση 0,0001

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	72,738838	-	-
Σειριακό RDP (με βελτιώσεις)	36,848791	-	-
Static1(pragma-for)	36,854948	35,859416	33,543426
Static2(manually)	37,053645	35,733271	34,156669
Dynamic	36,359496	19,601888	12,290414
Task1	45,824826	25,633780	18,450659
Task2	55,523545	31,109505	22,777146

### Περίπτωση 1: RDP -03

Χρησιμοποιήθηκαν 4 επεξεργαστές για κάθε περίπτωση που σημαίνει ένα νήμα/επεξεργαστή.

Ως γενικό συμπέρασμα προκύπτει ότι με τις διανυσματοποιήσεις οι χρόνοι είναι πολύ μικρότεροι από όταν το αρχείο μεταγλωττιζόταν με -O0.

Επίσης, οι χρόνοι παρουσιάζουν βελτίωση, όσο περισσότεροι επεξεργαστές ασχολούνται τόσο από πλευράς ενέργειας όσο και από πλευράς χρόνου. Ο αλγόριθμος Task2 είναι χειρότερος από πλευράς χρόνου από τον Task1 και σε κάποιες περιπτώσεις το overhead είναι μεγαλύτερο από τον υπολογισμό αφού δεν υπάρχει speedup.

### Υπο-περίπτωση 0,1

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	7,994614	-	-
Σειριακό RDP (με βελτιώσεις)	0,556048	-	-
Static1(pragma-for)	0,604621	0,558734	0,539514
Static2(manually)	0,589633	0,575602	0,586500
Dynamic	0,558013	0,304866	0,272879
Task1	0,674186	0,410247	0,393879
Task2	0,664348	0,443595	0,446526

### Υπο-περίπτωση 0,01

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	14,226300	-	-
Σειριακό RDP (με βελτιώσεις)	1,150196	-	-
Static1(pragma-for)	1,183080	1,144822	1,125490
Static2(manually)	1,153859	1,149551	1,108126
Dynamic	1,160626	0,620207	0,510911
Task1	1,405105	0,879469	0,742603
Task2	1,517664	0,883916	0,871541

### Υπο-περίπτωση 0,001

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	20,324560	-	-
Σειριακό RDP (με βελτιώσεις)	2,396585	-	-
Static1(pragma-for)	2,384240	2,340367	2,171788
Static2(manually)	2,389459	2,338939	2,200872
Dynamic	2,380264	1,281621	0,953988
Task1	3,143994	1,916092	1,633533
Task2	3,564230	2,331834	1,975373

### Υπο-περίπτωση 0,0001

	1 νήμα	2 νήματα	4 νήματα
Σειριακό RDP (χωρίς βελτιώσεις)	26,321580	-	-
Σειριακό RDP (με βελτιώσεις)	4,852152	-	-
Static1(pragma-for)	4,771988	4,655919	4,522034
Static2(manually)	4,774386	4,628766	4,383505
Dynamic	4,769070	2,550397	1,869482
Task1	6,578166	4,108594	3,265884
Task2	7,491719	4,803308	3,779168

### ΠΑΡΑΤΗΡΗΣΕΙΣ

Στις Static και Dynamic περιπτώσεις ανάθεσης εμφανίζεται ο μεγαλύτερος χρόνος για την εκτέλεση του προγράμματος που προκύπτει από τον μέσο όρο τουλάχιστον 3 περιπτώσεων. Ο μικρός/μικρότερος χρόνος δεν λαμβάνεται υπόψιν για την άσκηση καθώς ζητείται ο μέγιστος χρόνος που χρειάζεται για την εκτέλεση του προγράμματος. Επίσης υπάρχουν περιπτώσεις όπου ο χρόνος έβγαине πολύ μεγαλύτερος και (πιο σπάνια) πολύ μικρότερος σε κάποια μέτρηση πιθανώς από κάποιο bottleneck. Σε αυτή την περίπτωση ο μέσος όρος που δίνεται προκύπτει από παραπάνω μετρήσεις.

Για τους χρόνους του task1 παρατηρείται το φαινόμενο ο χρόνος για το 1 νήμα να είναι μεγαλύτερος από τον χρόνο για το σειριακό τρόπο εκτέλεσης. Αυτό συμβαίνει γιατί στον χρόνο σειριακής εκτέλεσης προστίθεται και ο χρόνος δημιουργίας και

εισαγωγής-εξαγωγής στην ουρά των tasks ενώ για τον χρόνο στο task2 το overhead από την δημιουργία tasks με 2 πυρήνες-νήματα δεν υπερκαλύπτεται από την παράλληλη εκτέλεση και είναι πιο αργό από την σειριακή εκτέλεση. Ίσως αυτό να σημαίνει πως θα μπορούσε να γίνει και κάτι καλύτερο όσον αφορά τον υπολογισμό ορίων περικοπής αλλά το πρόβλημα είναι κυρίως πως το overhead είναι αρκετά σημαντικό ως προς τον χρόνο εκτέλεσης. Υπάρχει όμως καλό scalability όπου αυτό είναι δυνατόν, ειδικά εφόσον το σύστημα που έγιναν οι μετρήσεις είναι τετραπύρηνο.

## ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΟΒΕΛΤΙΩΣΗΣ

Εδώ υπάρχουν όλα τα διαγράμματα χρονοβελτίωσης καθώς επίσης και πίνακες που δείχνουν τους πίνακες χρονοβελτίωσης που χρησιμοποιήθηκαν..

Κάθε περίπτωση και υπο-περίπτωση τίθεται σε σύγκριση με τους χρόνους για να ελεγχθεί η χρονοβελτίωση. Ο αριθμός των διαγραμμάτων είναι ίσος με τον αριθμό των πινάκων και εκείνα τα δεδομένα χρησιμοποιούνται και στα διαγράμματα.

Τέλος, για διευκόλυνση το static με #pragma ονομάζεται Static1 ενώ το προγραμματισμένο με το id Static2.

### Περίπτωση 1: -00

Epsilon=0,1

	Static1	Static2	Dynamic	Task1	Task2
1 νήμα	0,959714	0,975504	0,971462	0,733611	0,578789
2 νήματα	0,944525	0,968767	1,865926	1,125127	0,857141
4 νήματα	1,023313	1,026059	2,821194	1,358387	1,117465

Epsilon=0,01

	Static1	Static2	Dynamic	Task1	Task2
1 νήμα	0,9921019	1,007280	1,018509	0,744787	0,579007
2 νήματα	1,008462	1,01777	1,913125	1,280424	1,021351
4 νήματα	1,068323	1,070722	2,957028	1,694178	1,301385

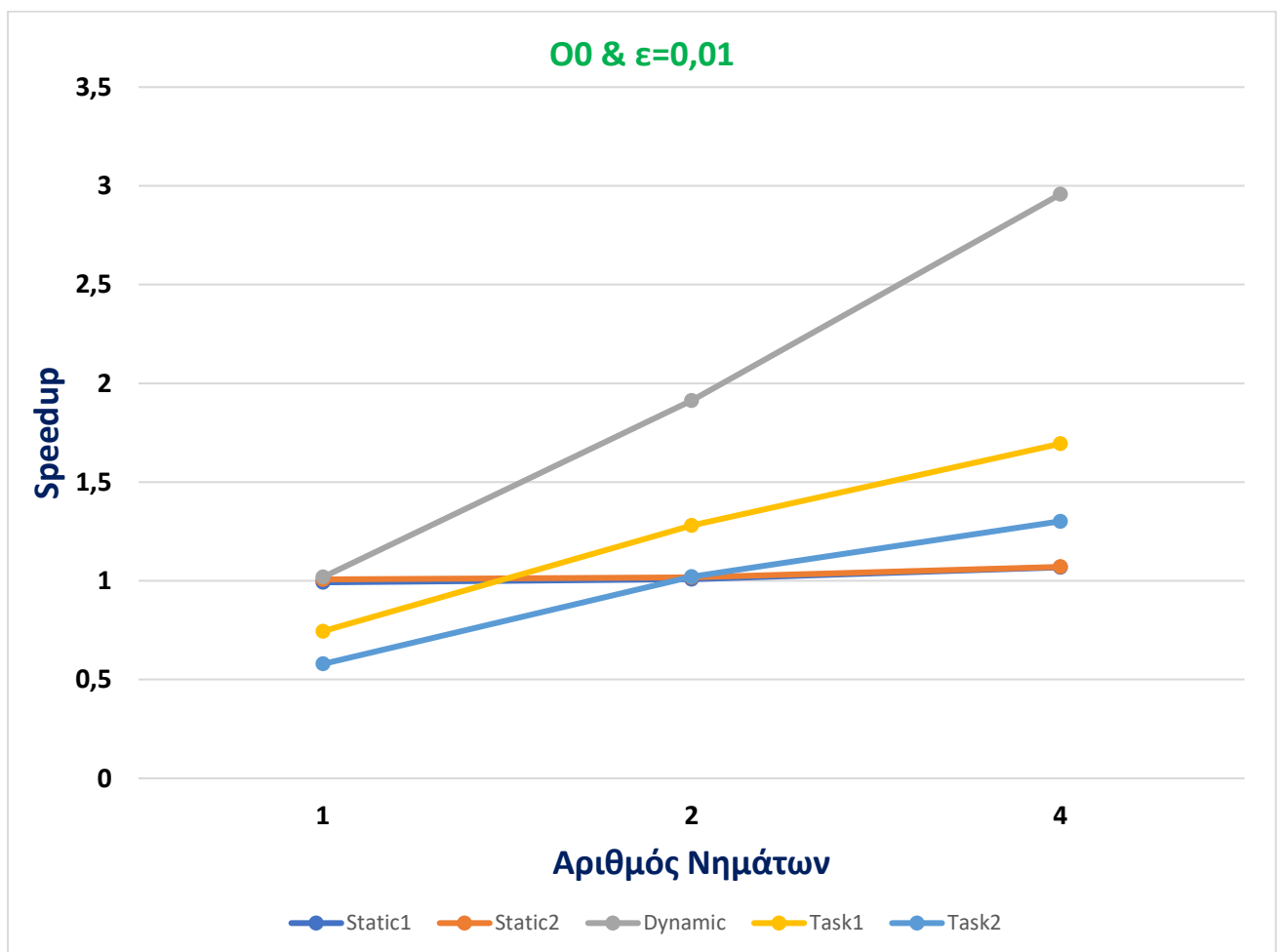
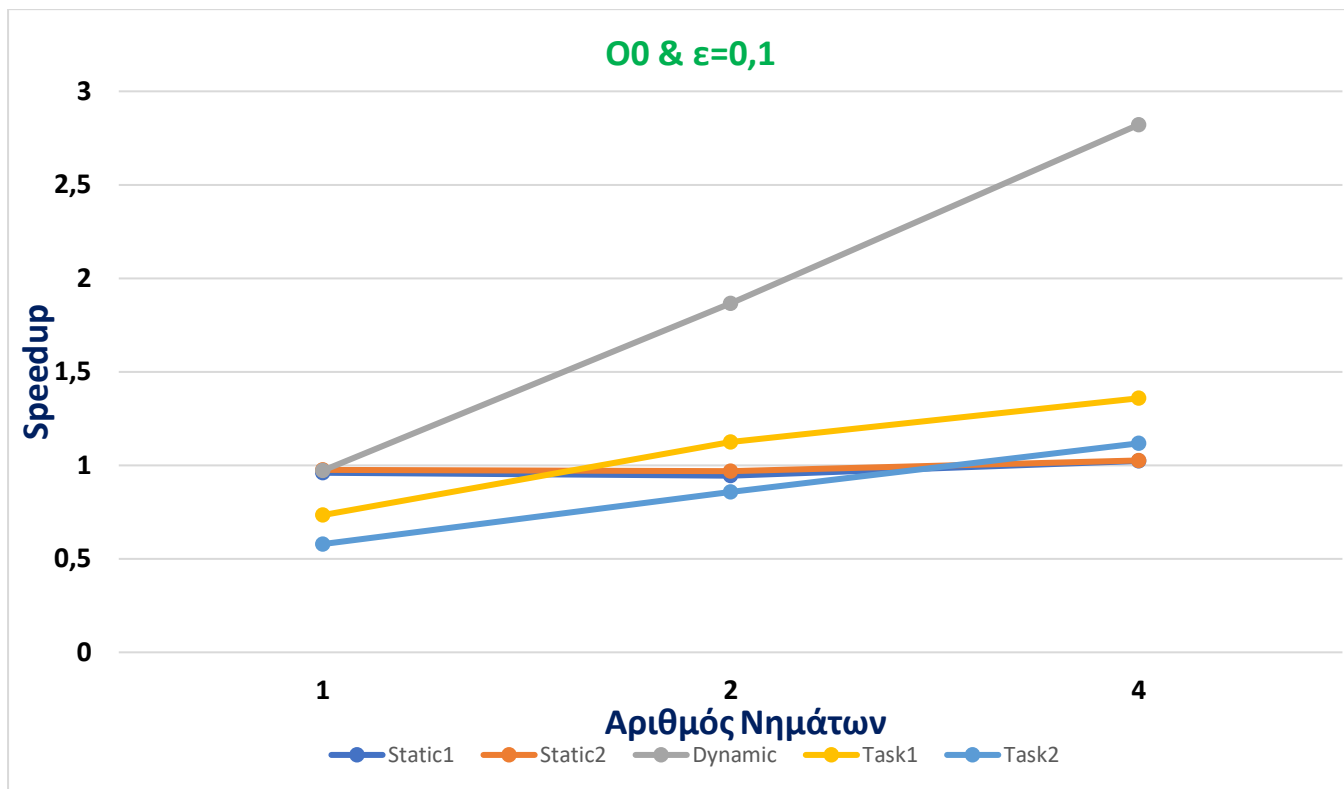
Epsilon=0,001

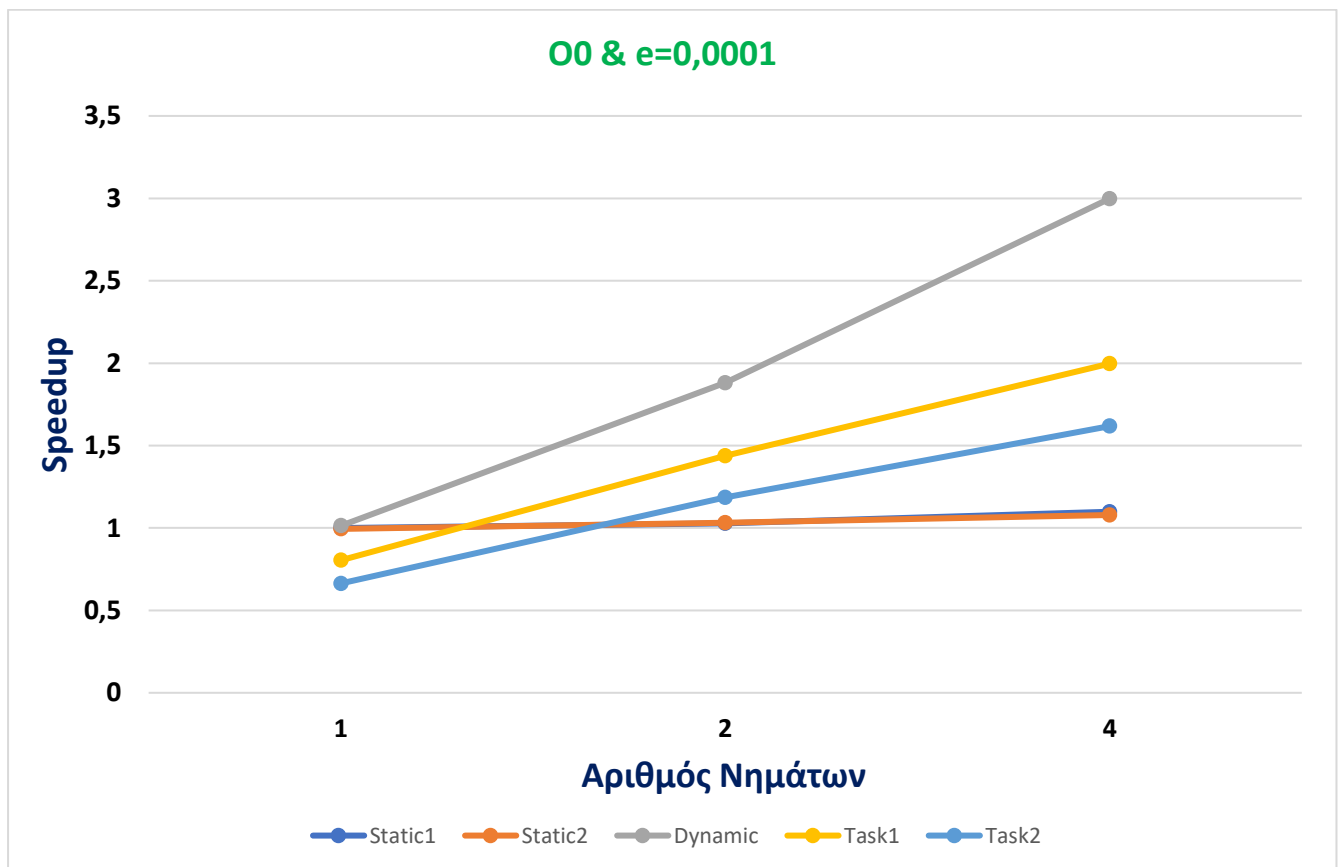
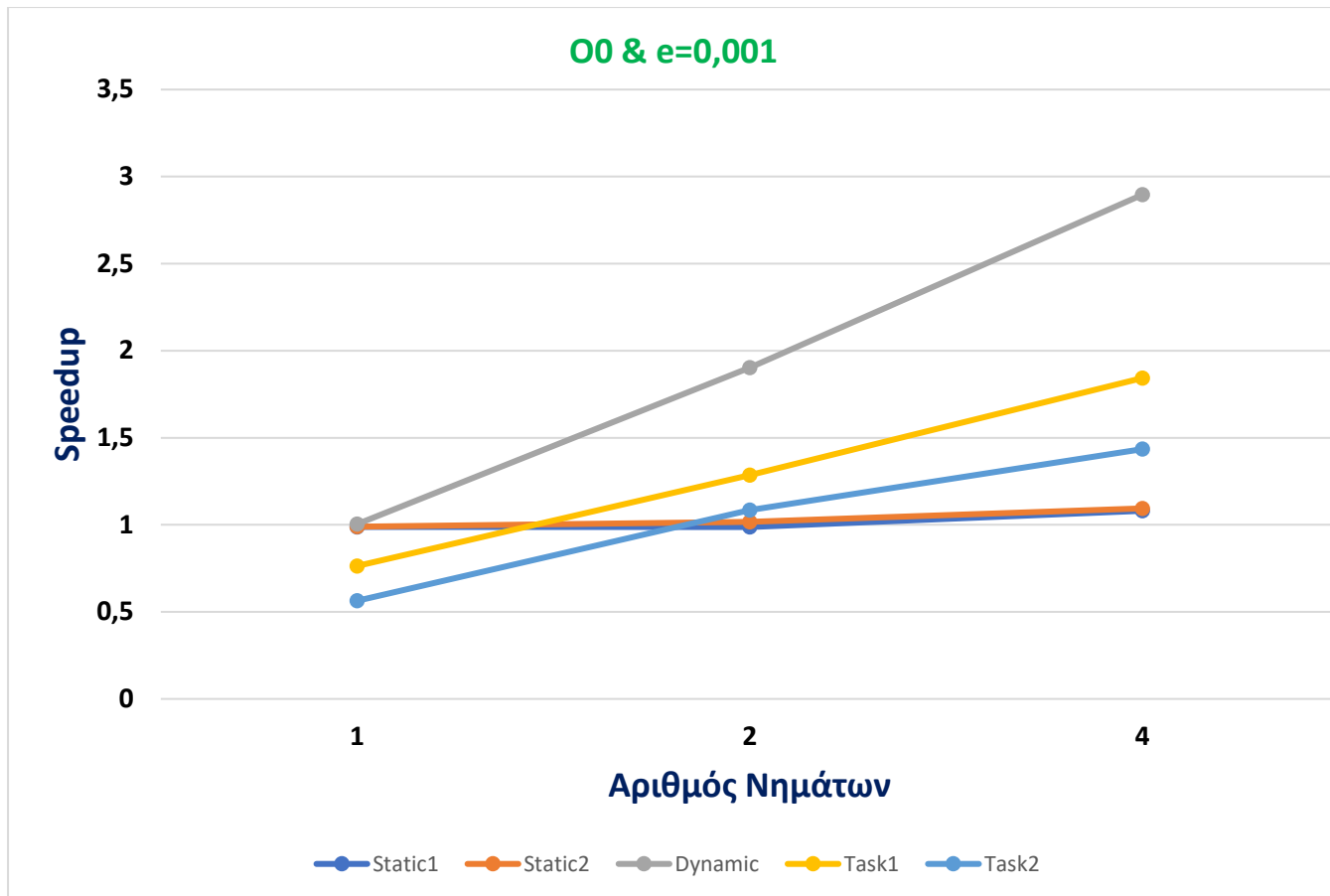
	Static1	Static2	Dynamic	Task1	Task2
1 νήμα	0,987587	0,989323	1,003888	0,763596	0,563828
2 νήματα	0,987326	1,017617	1,9027	1,28565	1,084361
4 νήματα	1,080651	1,094260	2,895703	1,84284	1,434706

Epsilon=0,0001

	Static1	Static2	Dynamic	Task1	Task2
1 νήμα	0,999833	0,994471	1,013457	0,804123	0,663660
2 νήματα	1,027590	1,031218	1,879859	1,437509	1,184486
4 νήματα	1,098540	1,078816	2,998173	1,997153	1,617796







### Περίπτωση 1: -03

Epsilon = 0,1

	Static1	Static2	Dynamic	Task1	Task2
1	0,919663	0,943040	0,996479	0,824769	0,836983
2	0,995192	0,966028	1,823910	1,355398	1,253504
4	1,030646	0,948078	2,037709	1,411723	1,245276

Epsilon = 0,01

	Static1	Static2	Dynamic	Task1	Task2
1	0,972205	0,996825	0,991014	0,818584	0,757873
2	1,004694	1,000561	1,854536	1,307830	1,301250
4	1,021951	1,037964	2,251264	1,548871	1,319726

Epsilon = 0,001

	Static1	Static2	Dynamic	Task1	Task2
1	1,005177	1,002982	1,006856	0,762274	0,672399
2	1,024021	1,024646	1,869963	1,250767	1,027768
4	1,103508	1,088925	2,512175	1,467117	1,213231

Epsilon = 0,0001

	Static1	Static2	Dynamic	Task1	Task2
1	1,016799	1,016288	1,017421	0,737615	0,647669
2	1,042147	1,048260	1,902509	1,180976	1,010169
4	1,073002	1,106911	2,595453	1,485709	1,283920

### Παρατηρήσεις

Στην συγκεκριμένη περίπτωση όπως και πριν χρησιμοποιήθηκαν 4 πυρήνες για κάθε περίπτωση.

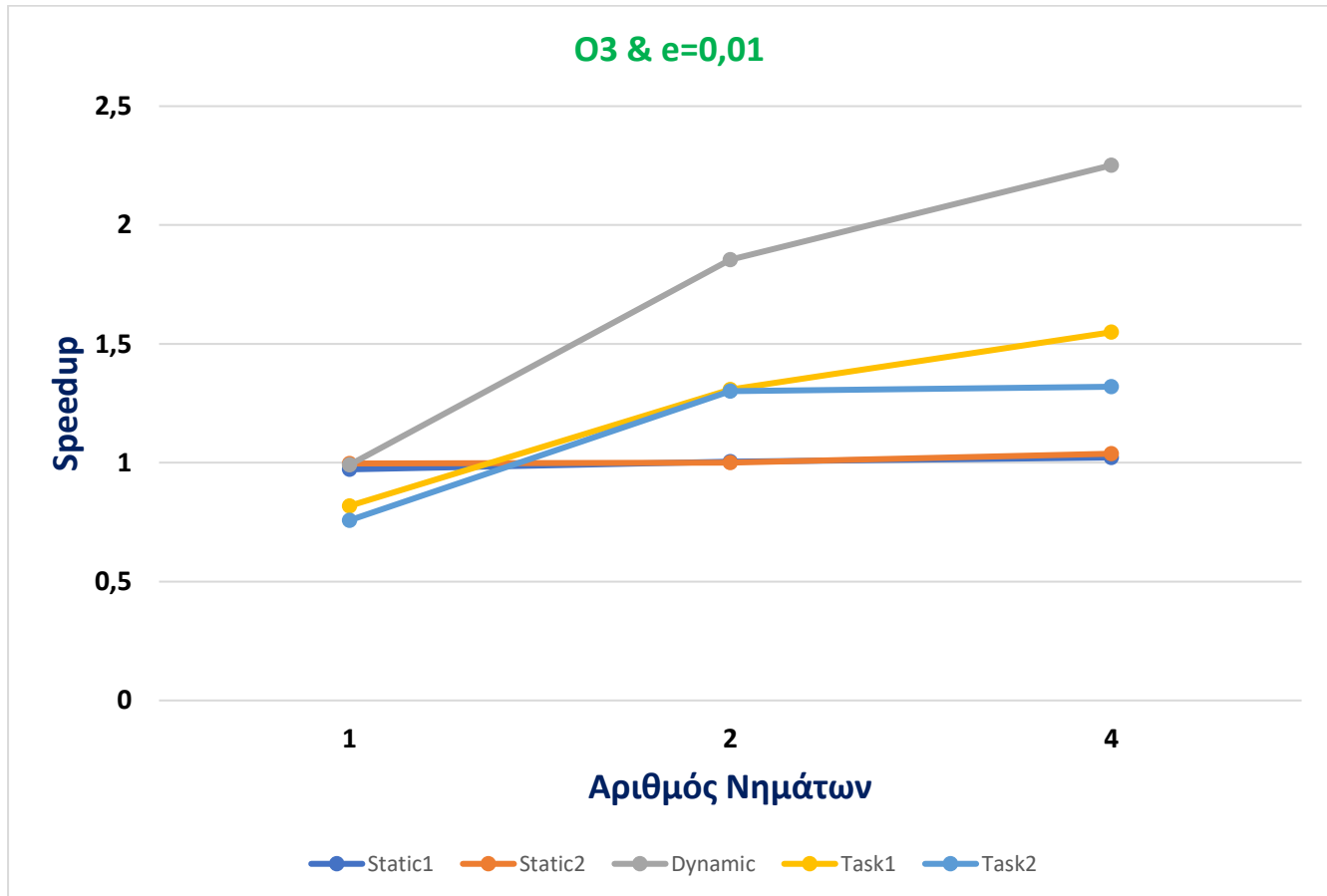
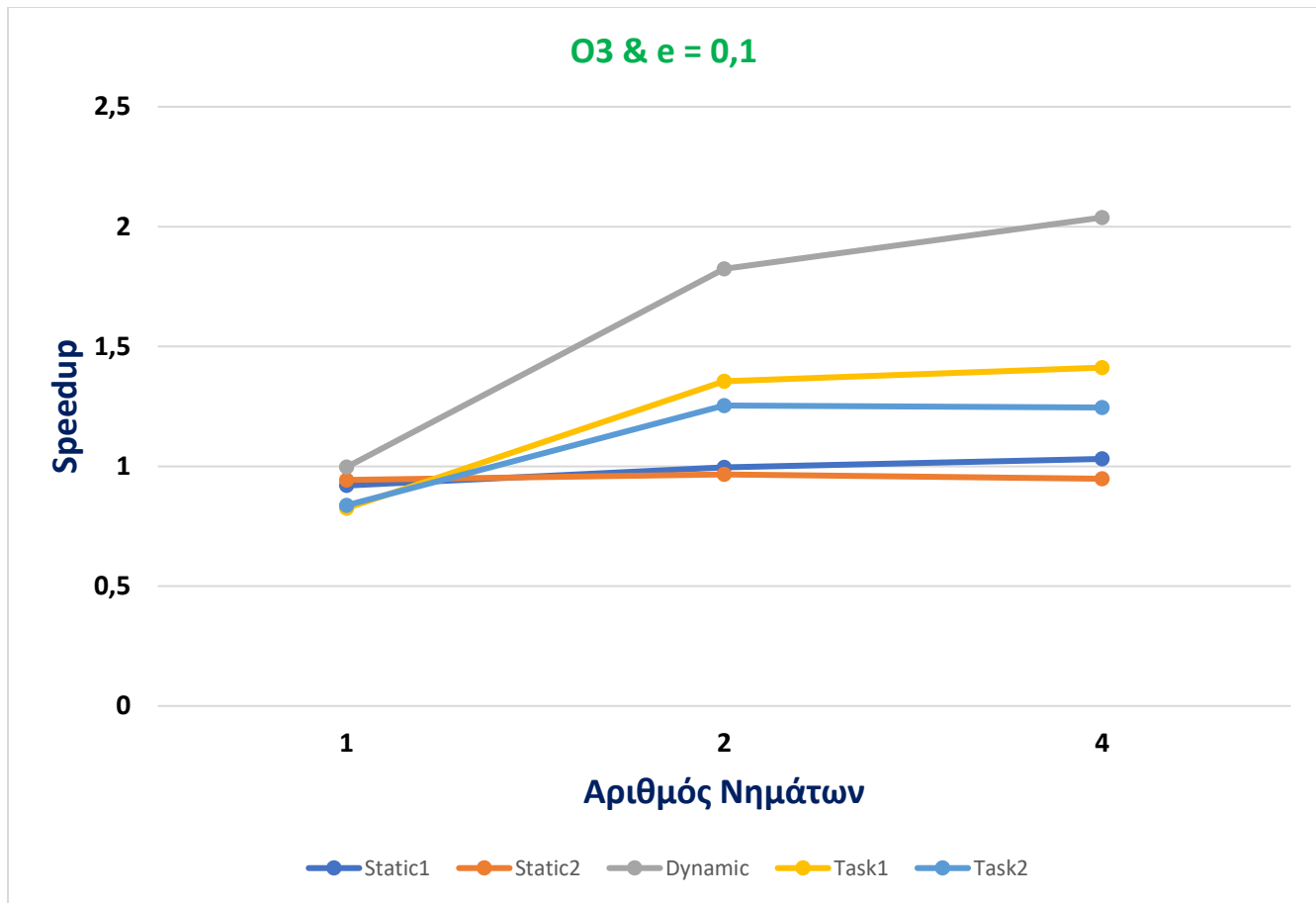
Οι χρόνοι με διανυσματοποιήσεις είναι πολύ καλύτεροι.

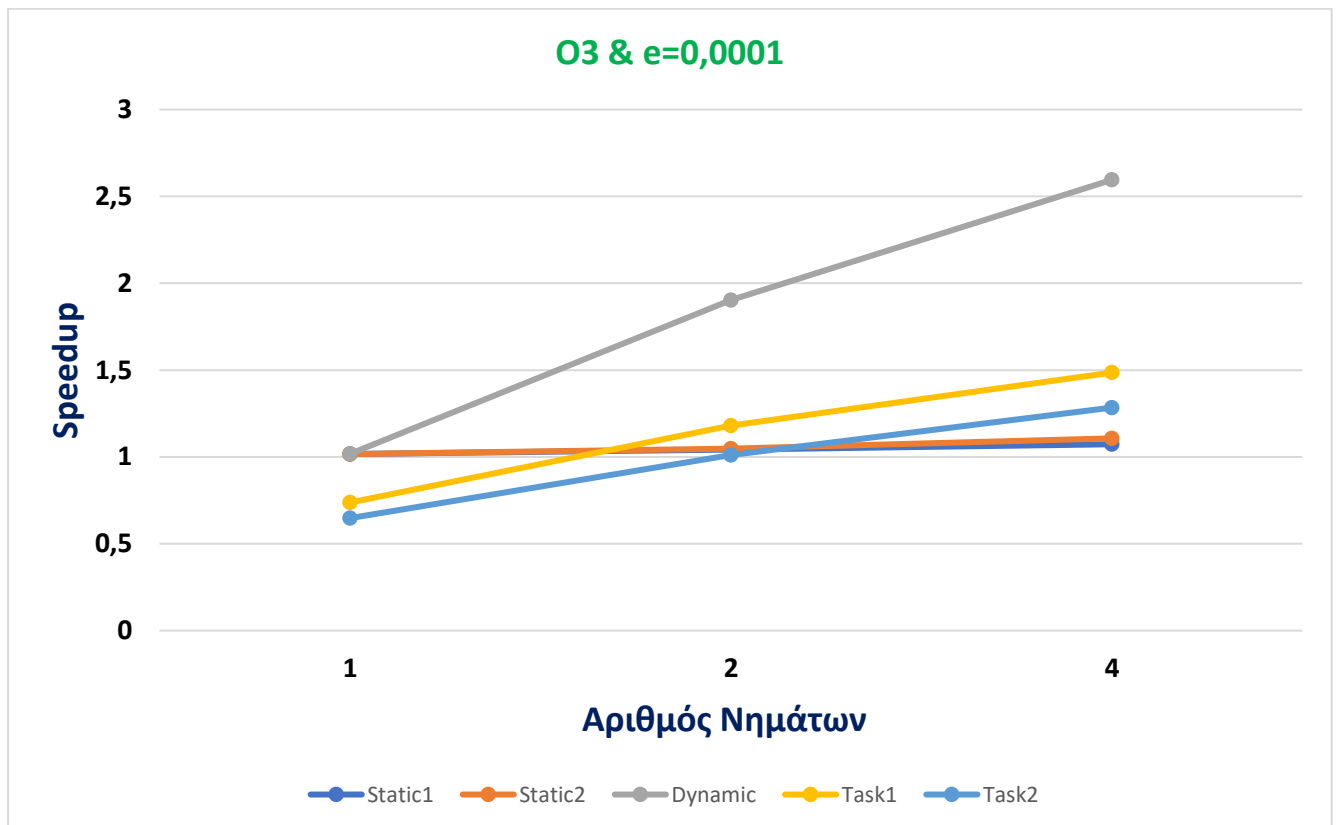
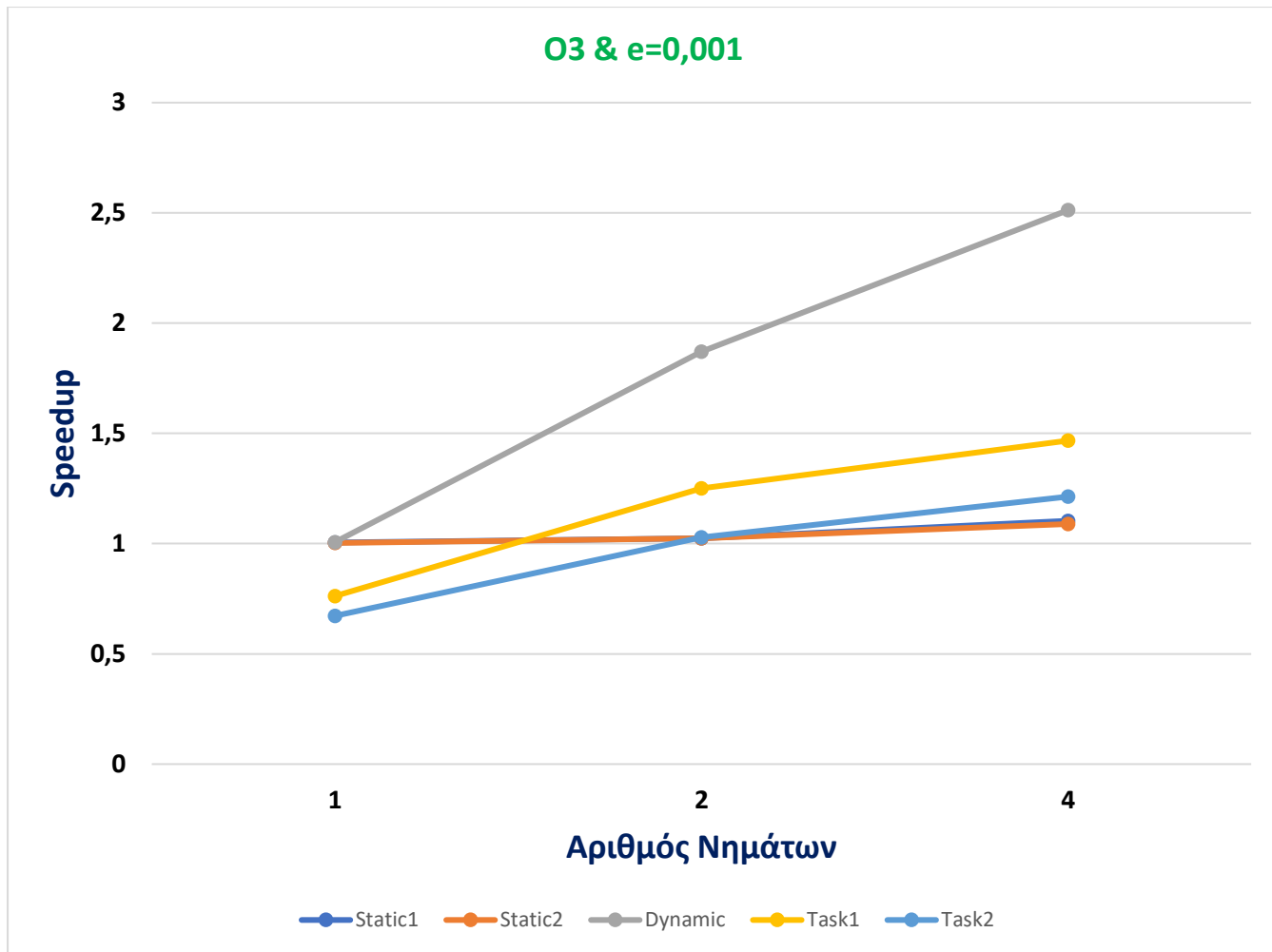
Ο τύπος που χρησιμοποιήθηκε σε όλες τις περιπτώσεις για να βρεθεί η χρονοβελτίωση (speedup) ήταν ο εξής:

Speedup = Χρόνος σειριακής Εκτέλεσης / Μέτρηση για συγκεκριμένη περίπτωση

Συνεπώς, μπορεί όποιος θέλει κάνοντας την διαίρεση από τα αποτελέσματα που έχουν δοθεί να διαπιστώσει την ορθότητα των αποτελεσμάτων.

Όπως θα

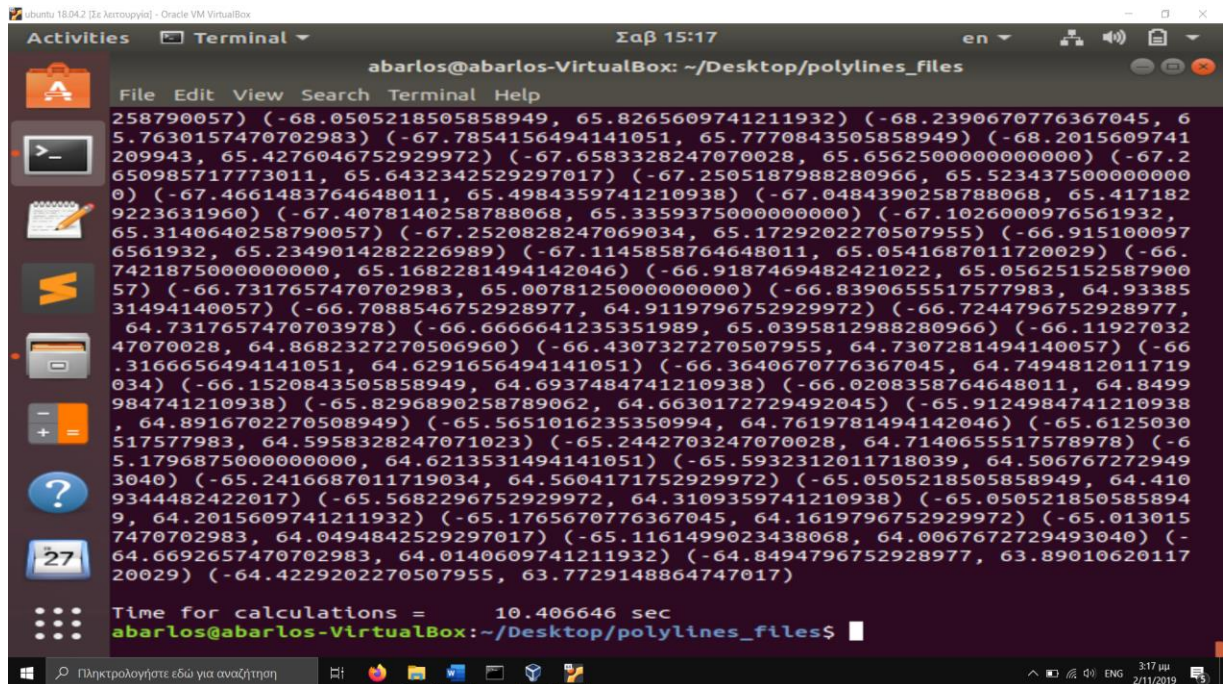




## SCREENSHOTS

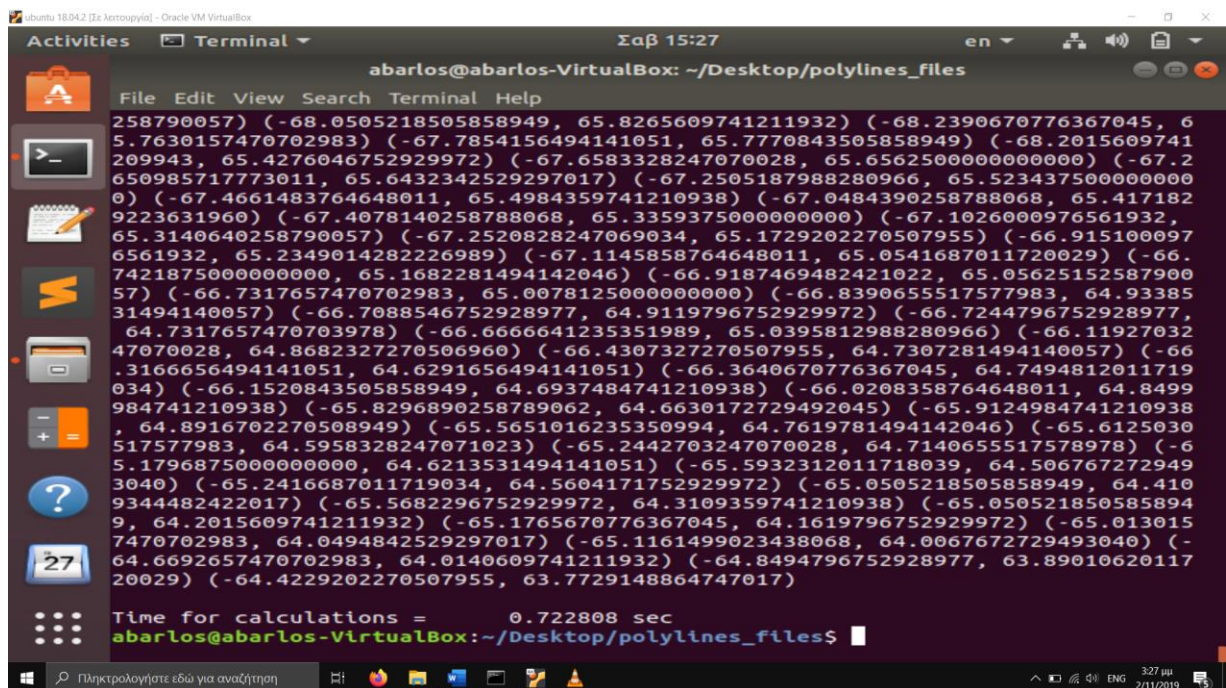
Αν και δεν ζητήθηκαν παρατίθενται screenshots για την σωστή λειτουργία των προγραμμάτων στην περίπτωση:  $\varepsilon = 0,1$  και αρχείο polylines\_large. Αφού τραβήχτηκαν οι φωτογραφίες για να τις ελεγχθούν τα αποτελέσματα, μπήκαν για να βοηθήσουν και εσάς στην επιβεβαίωση ότι όλα πάνε σωστά.!

### notChangedRDP



The screenshot shows a terminal window titled 'abarloso@abarloso-VirtualBox: ~/Desktop/polylines\_files'. The terminal displays a long list of coordinates in parentheses, separated by spaces. At the bottom, it shows 'Time for calculations = 10.406646 sec' and the prompt 'abarloso@abarloso-VirtualBox:~/Desktop/polylines\_files\$'.

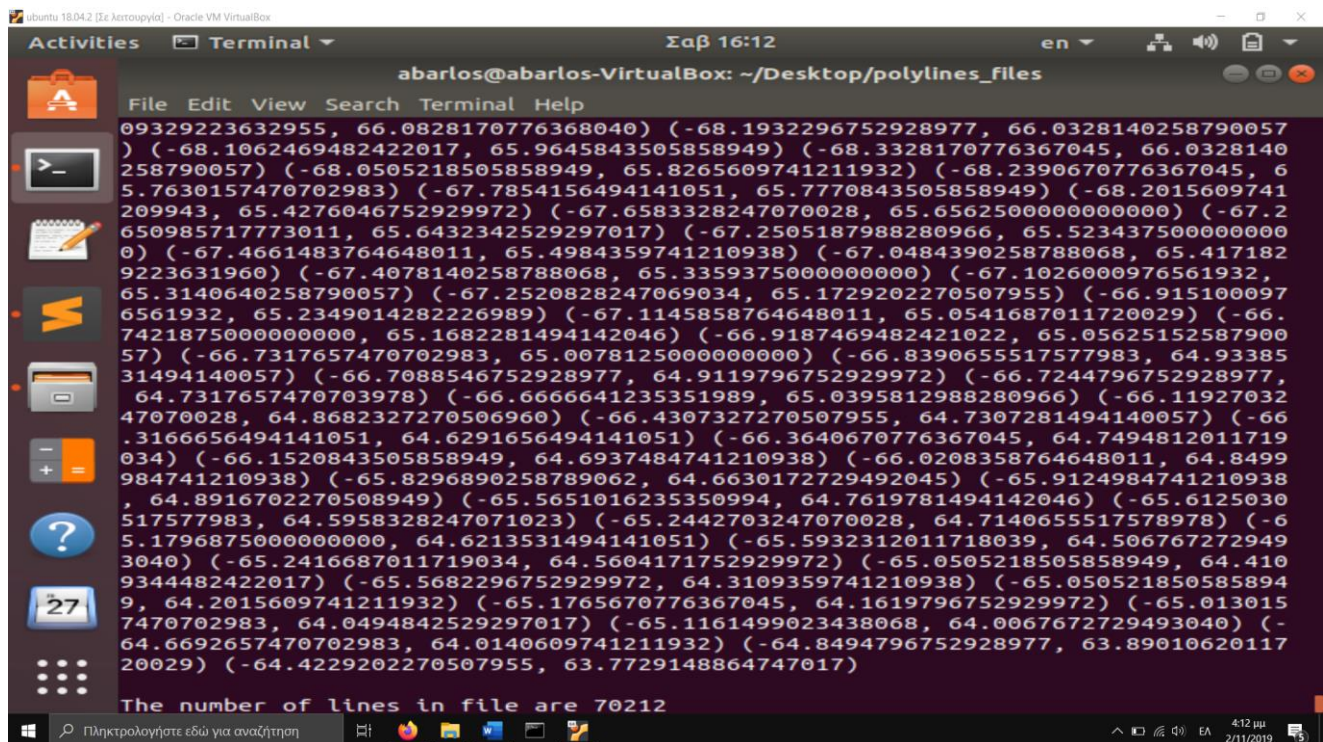
### RDP



The screenshot shows a terminal window titled 'abarloso@abarloso-VirtualBox: ~/Desktop/polylines\_files'. The terminal displays a long list of coordinates in parentheses, separated by spaces. At the bottom, it shows 'Time for calculations = 0.722808 sec' and the prompt 'abarloso@abarloso-VirtualBox:~/Desktop/polylines\_files\$'.



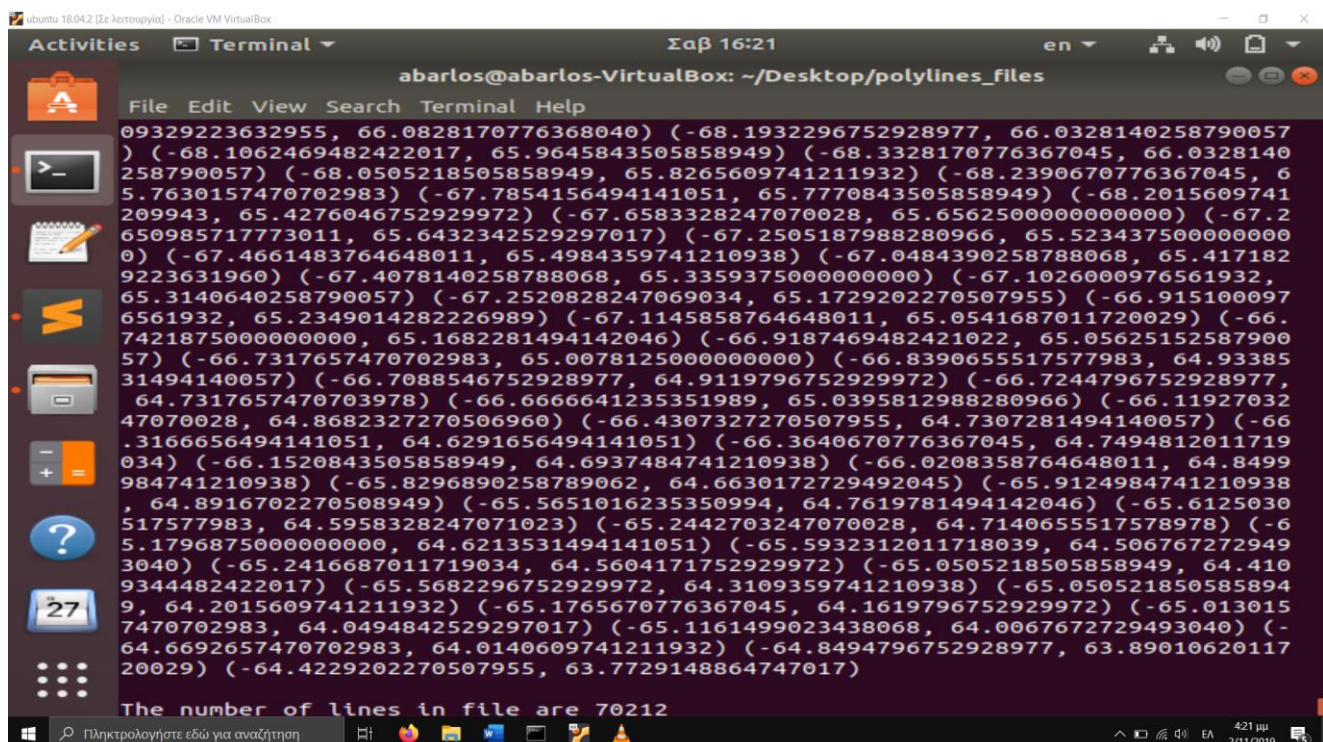
## Static pragma omp for



```
abarlos@abarlos-VirtualBox: ~/Desktop/polylines_files
File Edit View Search Terminal Help
09329223632955, 66.0828170776368040) (-68.1932296752928977, 66.0328140258790057
) (-68.1062469482422017, 65.9645843505858949) (-68.3328170776367045, 66.0328140
258790057) (-68.0505218505858949, 65.8265609741211932) (-68.2390670776367045, 6
5.7630157470702983) (-67.7854156494141051, 65.7770843505858949) (-68.2015609741
209943, 65.4276046752929972) (-67.6583328247070028, 65.6562500000000000) (-67.2
650985717773011, 65.6432342529297017) (-67.2505187988280966, 65.523437500000000
0) (-67.4661483764648011, 65.4984359741210938) (-67.0484390258788068, 65.417182
9223631960) (-67.4078140258788068, 65.3359375000000000) (-67.1026000976561932,
65.3140640258790057) (-67.2520828247069034, 65.1729202270507955) (-66.915100097
6561932, 65.2349014282226989) (-67.1145858764648011, 65.0541687011720029) (-66.
7421875000000000, 65.1682281494142046) (-66.9187469482421022, 65.05625152587900
57) (-66.7317657470702983, 65.0078125000000000) (-66.8390655517577983, 64.93385
31494140057) (-66.7088546752928977, 64.9119796752929972) (-66.7244796752928977,
64.7317657470703978) (-66.6666641235351989, 65.0395812988280966) (-66.11927032
47070028, 64.8682327270506960) (-66.4307327270507955, 64.7307281494140057) (-66
.3166656494141051, 64.6291656494141051) (-66.3640670776367045, 64.7494812011719
034) (-66.1520843505858949, 64.6937484741210938) (-66.0208358764648011, 64.8499
984741210938) (-65.8296890258789062, 64.6630172729492045) (-65.9124984741210938
, 64.8916702270508949) (-65.5651016235350994, 64.7619781494142046) (-65.6125030
517577983, 64.5958328247071023) (-65.2442703247070028, 64.7140655517578978) (-6
5.1796875000000000, 64.6213531494141051) (-65.5932312011718039, 64.506767272949
3040) (-65.2416687011719034, 64.5604171752929972) (-65.0505218505858949, 64.410
9344482422017) (-65.5682296752929972, 64.3109359741210938) (-65.050521850585894
9, 64.2015609741211932) (-65.1765670776367045, 64.1619796752929972) (-65.013015
7470702983, 64.0494842529297017) (-65.1161499023438068, 64.0067672729493040) (-
64.6692657470702983, 64.0140609741211932) (-64.8494796752928977, 63.89010620117
20029) (-64.4229202270507955, 63.7729148864747017)

The number of lines in file are 70212
```

## Static manually



```
abarlos@abarlos-VirtualBox: ~/Desktop/polylines_files
File Edit View Search Terminal Help
09329223632955, 66.0828170776368040) (-68.1932296752928977, 66.0328140258790057
) (-68.1062469482422017, 65.9645843505858949) (-68.3328170776367045, 66.0328140
258790057) (-68.0505218505858949, 65.8265609741211932) (-68.2390670776367045, 6
5.7630157470702983) (-67.7854156494141051, 65.7770843505858949) (-68.2015609741
209943, 65.4276046752929972) (-67.6583328247070028, 65.6562500000000000) (-67.2
650985717773011, 65.6432342529297017) (-67.2505187988280966, 65.523437500000000
0) (-67.4661483764648011, 65.4984359741210938) (-67.0484390258788068, 65.417182
9223631960) (-67.4078140258788068, 65.3359375000000000) (-67.1026000976561932,
65.3140640258790057) (-67.2520828247069034, 65.1729202270507955) (-66.915100097
6561932, 65.2349014282226989) (-67.1145858764648011, 65.0541687011720029) (-66.
7421875000000000, 65.1682281494142046) (-66.9187469482421022, 65.05625152587900
57) (-66.7317657470702983, 65.0078125000000000) (-66.8390655517577983, 64.93385
31494140057) (-66.7088546752928977, 64.9119796752929972) (-66.7244796752928977,
64.7317657470703978) (-66.6666641235351989, 65.0395812988280966) (-66.11927032
47070028, 64.8682327270506960) (-66.4307327270507955, 64.7307281494140057) (-66
.3166656494141051, 64.6291656494141051) (-66.3640670776367045, 64.7494812011719
034) (-66.1520843505858949, 64.6937484741210938) (-66.0208358764648011, 64.8499
984741210938) (-65.8296890258789062, 64.6630172729492045) (-65.9124984741210938
, 64.8916702270508949) (-65.5651016235350994, 64.7619781494142046) (-65.6125030
517577983, 64.5958328247071023) (-65.2442703247070028, 64.7140655517578978) (-6
5.1796875000000000, 64.6213531494141051) (-65.5932312011718039, 64.506767272949
3040) (-65.2416687011719034, 64.5604171752929972) (-65.0505218505858949, 64.410
9344482422017) (-65.5682296752929972, 64.3109359741210938) (-65.050521850585894
9, 64.2015609741211932) (-65.1765670776367045, 64.1619796752929972) (-65.013015
7470702983, 64.0494842529297017) (-65.1161499023438068, 64.0067672729493040) (-
64.6692657470702983, 64.0140609741211932) (-64.8494796752928977, 63.89010620117
20029) (-64.4229202270507955, 63.7729148864747017)

The number of lines in file are 70212
```



## Dynamic

The screenshot shows a terminal window titled "abarlo@abarlo-VirtualBox: ~/Desktop/polylines\_files". The terminal displays a long list of coordinates in the format (x y z). The list is truncated on both ends. At the bottom, it states "The number of lines in file are 70212". The window's title bar includes "Ubuntu 18.04.2 (Σε λειτουργία) - Oracle VM VirtualBox", "Σαβ 16:30", and "en". The bottom status bar shows system icons and the date "2/11/2019".

```
09329223632955, 66.0828170776368040) (-68.1932296752928977, 66.0328140258790057
) (-68.1062469482422017, 65.9645843505858949) (-68.3328170776367045, 66.0328140
258790057) (-68.0505218505858949, 65.8265609741211932) (-68.2390670776367045, 6
5.7630157470702983) (-67.7854156494141051, 65.7770843505858949) (-68.2015609741
209943, 65.4276046752929972) (-67.6583328247070028, 65.6562500000000000) (-67.2
650985717773011, 65.6432342529297017) (-67.2505187988280966, 65.523437500000000
0) (-67.4661483764648011, 65.4984359741210938) (-67.0484390258788068, 65.417182
9223631960) (-67.4078140258788068, 65.3359375000000000) (-67.1026000976561932,
65.3140640258790057) (-67.2520828247069034, 65.1729202270507955) (-66.915100097
6561932, 65.2349014282226989) (-67.1145858764648011, 65.0541687011720029) (-66.
7421875000000000, 65.1682281494142046) (-66.9187469482421022, 65.05625152587900
57) (-66.7317657470702983, 65.0078125000000000) (-66.8390655517577983, 64.93385
31494140057) (-66.7088546752928977, 64.9119796752929972) (-66.7244796752928977,
64.7317657470703978) (-66.6666641235351989, 65.0395812988280966) (-66.11927032
47070028, 64.8682327270506960) (-66.4307327270507955, 64.7307281494140057) (-66
.3166656494141051, 64.6291656494141051) (-66.3640670776367045, 64.7494812011719
034) (-66.1520843505858949, 64.6937484741210938) (-66.0208358764648011, 64.8499
984741210938) (-65.8296890258789062, 64.6630172729492045) (-65.9124984741210938
, 64.8916702270508949) (-65.5651016235350994, 64.7619781494142046) (-65.6125030
517577983, 64.5958328247071023) (-65.2442703247070028, 64.7140655517578978) (-6
5.1796875000000000, 64.6213531494141051) (-65.5932312011718039, 64.506767272949
3040) (-65.2416687011719034, 64.5604171752929972) (-65.0505218505858949, 64.410
9344482422017) (-65.5682296752929972, 64.3109359741210938) (-65.050521850585894
9, 64.2015609741211932) (-65.1765670776367045, 64.1619796752929972) (-65.013015
7470702983, 64.0494842529297017) (-65.1161499023438068, 64.0067672729493040) (-
64.6692657470702983, 64.0140609741211932) (-64.8494796752928977, 63.89010620117
20029) (-64.4229202270507955, 63.7729148864747017)

The number of lines in file are 70212
```

## Task1

The screenshot shows the same terminal window as above, displaying the same list of coordinates. At the bottom, it shows the time taken for calculations: "Time for calculations = 3.279579 sec". The prompt is "abarlo@abarlo-VirtualBox:~/Desktop/polylines\_files\$". The window's title bar includes "Ubuntu 18.04.2 (Σε λειτουργία) - Oracle VM VirtualBox", "Κυρ 14:44", and "en". The bottom status bar shows system icons and the date "3/11/2019".

```
258790057) (-68.0505218505858949, 65.8265609741211932) (-68.2390670776367045, 6
5.7630157470702983) (-67.7854156494141051, 65.7770843505858949) (-68.2015609741
209943, 65.4276046752929972) (-67.6583328247070028, 65.6562500000000000) (-67.2
650985717773011, 65.6432342529297017) (-67.2505187988280966, 65.523437500000000
0) (-67.4661483764648011, 65.4984359741210938) (-67.0484390258788068, 65.417182
9223631960) (-67.4078140258788068, 65.3359375000000000) (-67.1026000976561932,
65.3140640258790057) (-67.2520828247069034, 65.1729202270507955) (-66.915100097
6561932, 65.2349014282226989) (-67.1145858764648011, 65.0541687011720029) (-66.
7421875000000000, 65.1682281494142046) (-66.9187469482421022, 65.05625152587900
57) (-66.7317657470702983, 65.0078125000000000) (-66.8390655517577983, 64.93385
31494140057) (-66.7088546752928977, 64.9119796752929972) (-66.7244796752928977,
64.7317657470703978) (-66.6666641235351989, 65.0395812988280966) (-66.11927032
47070028, 64.8682327270506960) (-66.4307327270507955, 64.7307281494140057) (-66
.3166656494141051, 64.6291656494141051) (-66.3640670776367045, 64.7494812011719
034) (-66.1520843505858949, 64.6937484741210938) (-66.0208358764648011, 64.8499
984741210938) (-65.8296890258789062, 64.6630172729492045) (-65.9124984741210938
, 64.8916702270508949) (-65.5651016235350994, 64.7619781494142046) (-65.6125030
517577983, 64.5958328247071023) (-65.2442703247070028, 64.7140655517578978) (-6
5.1796875000000000, 64.6213531494141051) (-65.5932312011718039, 64.506767272949
3040) (-65.2416687011719034, 64.5604171752929972) (-65.0505218505858949, 64.410
9344482422017) (-65.5682296752929972, 64.3109359741210938) (-65.050521850585894
9, 64.2015609741211932) (-65.1765670776367045, 64.1619796752929972) (-65.013015
7470702983, 64.0494842529297017) (-65.1161499023438068, 64.0067672729493040) (-
64.6692657470702983, 64.0140609741211932) (-64.8494796752928977, 63.89010620117
20029) (-64.4229202270507955, 63.7729148864747017)

Time for calculations = 3.279579 sec
abarlo@abarlo-VirtualBox:~/Desktop/polylines_files$
```



## Task2

The screenshot shows a terminal window titled "abarlo@abarlo-VirtualBox: ~/Desktop/polylines\_files". The terminal displays a large list of coordinates in parentheses, separated by commas. The coordinates are in the format (x, y), where x and y are decimal values. The list is as follows:

```
(-68.0505218505858949, 65.8265609741211932) (-68.2390670776367045, 65.7630157470702983) (-67.7854156494141051, 65.7770843505858949) (-68.2015609741209943, 65.4276046752929972) (-67.6583328247070028, 65.6562500000000000) (-67.2650985717773011, 65.6432342529297017) (-67.2505187988280966, 65.5234375000000000) (-67.4661483764648011, 65.4984359741210938) (-67.0484390258788068, 65.4171829223631960) (-67.4078140258788068, 65.3359375000000000) (-67.1026000976561932, 65.3140640258790057) (-67.2520828247069034, 65.1729202270507955) (-66.9151000976561932, 65.2349014282226989) (-67.1145858764648011, 65.0541687011720029) (-66.7421875000000000, 65.1682281494142046) (-66.9187469482421022, 65.0562515258790057) (-66.7317657470702983, 65.0078125000000000) (-66.8390655517577983, 64.9338531494140057) (-66.7088546752928977, 64.9119796752929972) (-66.7244796752928977, 64.7317657470703978) (-66.6666641235351989, 65.0395812988280966) (-66.1192703247070028, 64.8682327270506960) (-66.4307327270507955, 64.7307281494140057) (-66.3166656494141051, 64.6291656494141051) (-66.3640670776367045, 64.7494812011719034) (-66.1520843505858949, 64.6937484741210938) (-66.0208358764648011, 64.8499984741210938) (-65.8296890258789062, 64.6630172729492045) (-65.9124984741210938, 64.8916702270508949) (-65.5651016235350994, 64.7619781494142046) (-65.6125030517577983, 64.5958328247071023) (-65.2442703247070028, 64.7140655517578978) (-65.1796875000000000, 64.6213531494141051) (-65.5932312011718039, 64.5067672729493040) (-65.2416687011719034, 64.5604171752929972) (-65.0505218505858949, 64.4109344482422017) (-65.5682296752929972, 64.3109359741210938) (-65.0505218505858949, 64.2015609741211932) (-65.1765670776367045, 64.1619796752929972) (-65.0130157470702983, 64.0494842529297017) (-65.1161499023438068, 64.0067672729493040) (-64.6692657470702983, 64.0140609741211932) (-64.8494796752928977, 63.8901062011720029) (-64.4229202270507955, 63.7729148864747017)
```

Below the list of coordinates, the terminal shows the time for calculations:

```
Time for calculations = 0.445535 sec
```

The prompt is `abarlo@abarlo-VirtualBox:~/Desktop/polylines_files$`.