ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάθημα Προπτυχιακών Σπουδών:

Παράλληλος Υπολογισμός

Ακαδημαϊκό Έτος: 2022-2023

Εξάμηνο: 6ο

Εργασία

Ομάδα Εργασίας:

Δημήτρης Στυλιανού Π20004

Κωνσταντίνος Λοϊζίδης Π20007

Αποστόλης Σιαμπάνης Π20173

Περιεχόμενα

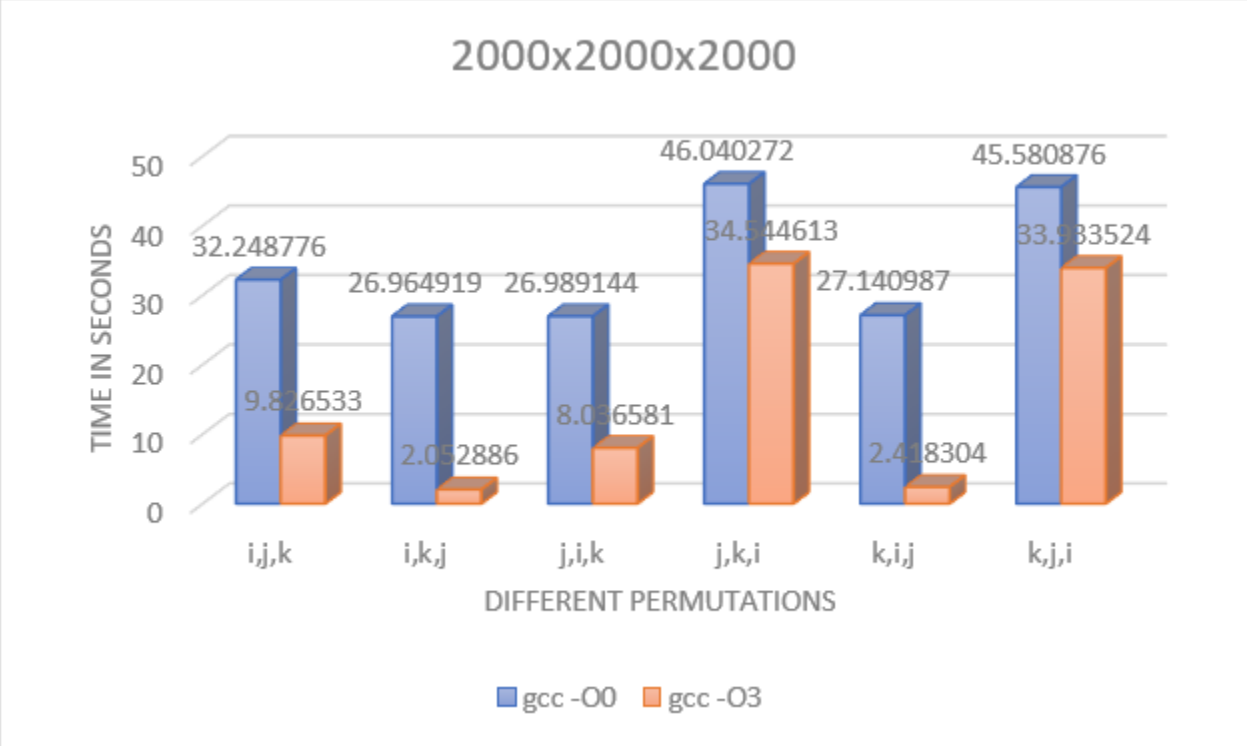
Ερώτημα 1 3
Παρατηρήσεις 5
Ερώτημα 2 6
Posix Threads 6
Παρατηρήσεις 8
Vectorize O3 9
OpenMP 11
Παρατηρήσεις 13
Ερώτημα 3 14
Μέγεθος μητρώων N = K = M = 4000 14
Dynamic Schedule 14
Guided Schedule 15
Static Schedule 16
Μέγεθος μητρώων N = 200, K = 8000, M = 40000 18
Dynamic Schedule 18
Guided Schedule 19
Static Schedule 20
Μέγεθος μητρώων N = 8000, K = 200, M = 40000 21
Dynamic Schedule 21
Guided Schedule 22
Static Schedule 22
Παρατηρήσεις 24

Για τον πηγαίο κώδικα της εργασίας πατήστε εδώ.

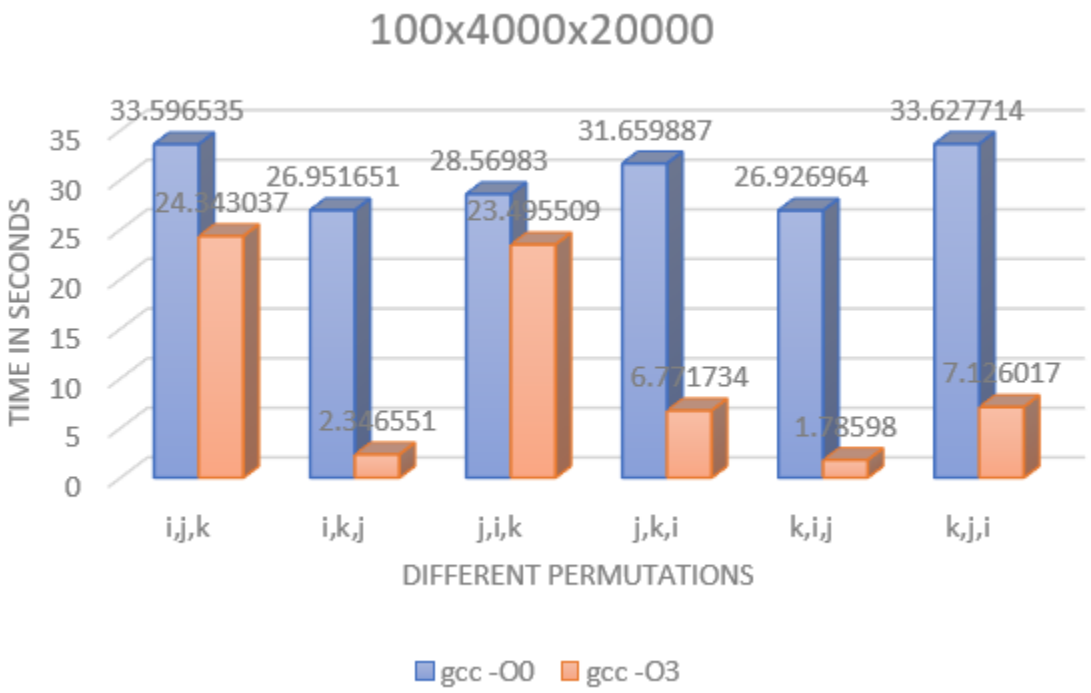
Ερώτημα 1

Εκτελώντας τον ακολουθιακό κώδικα με τις 6 πιθανές μεταθέσεις, με τη μεταγλώττιση του προγράμματος χωρίς βελτιστοποίηση (gcc -O0) και με μέγιστη βελτιστοποίηση (gcc -O3), παρατηρήσαμε τους παρακάτω χρόνους:

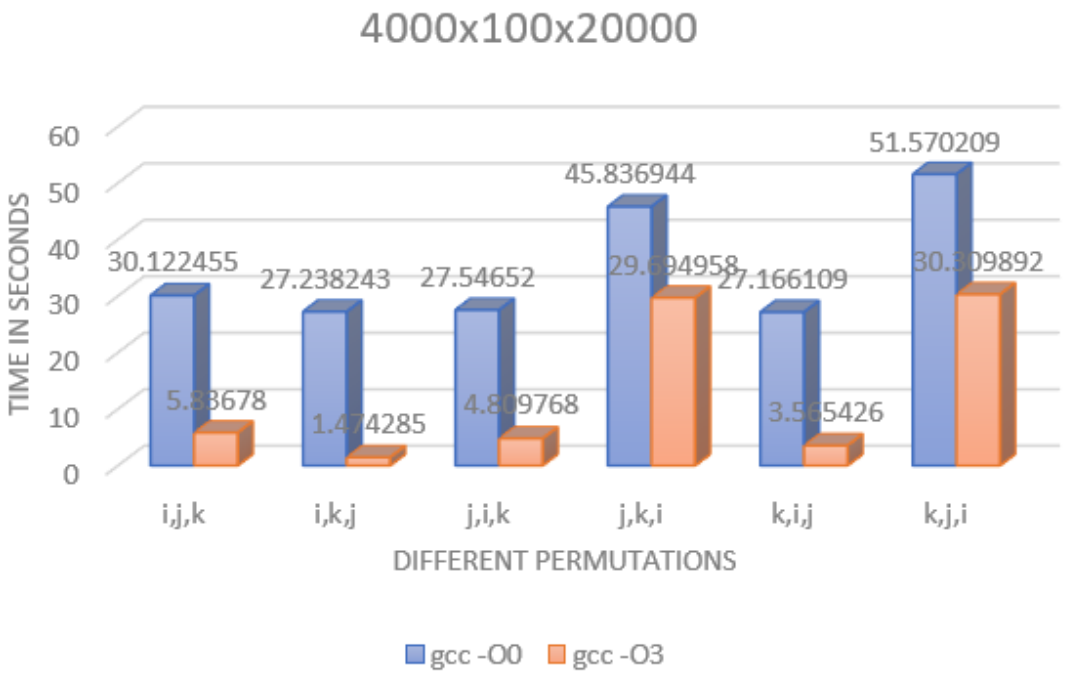
1. Με μέγεθος μητρώων N = K = M = 2000



2. Με μέγεθος μητρώων N = 100, K = 4000, M = 20000



3. Με μέγεθος μητρώων N = 4000, K = 100, M = 20000



Παρατηρήσεις:

- Γενικότερα, παρατηρήσαμε ότι η δεύτερη μετάθεση (i,k,j) είναι αποδοτικότερη εφόσον “εκμεταλλεύεται” με καλύτερο τρόπο την κρυφή μνήμη (cache memory). Αυτό γίνεται διότι όταν φορτώνεται ένα στοιχείο στην κρυφή μνήμη, φορτώνονται και κάποια επόμενα στοιχεία του πίνακα (ανάλογα την χωρητικότητα της κρυφής μνήμης).

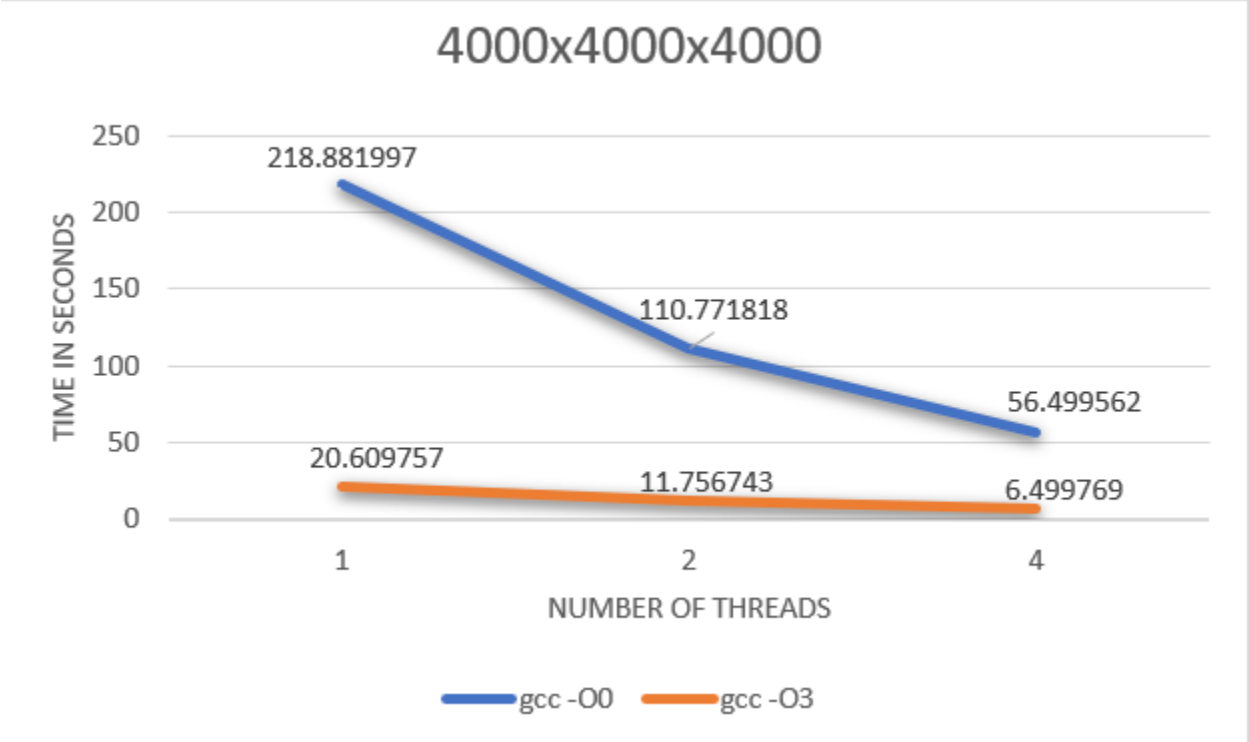
  Για παράδειγμα, αν φορτωθεί στην κρυφή μνήμη (μεγέθους 64 bytes) ο ακέραιος N[i][j] (μεγέθους 4 bytes), στη κρυφή μνήμη θα φορτωθούν ακόμη 64/4 - 1 = 15 ακέραιοι αριθμοί (N[i][j+1] μέχρι N[i][j+15]).

Ερώτημα 2

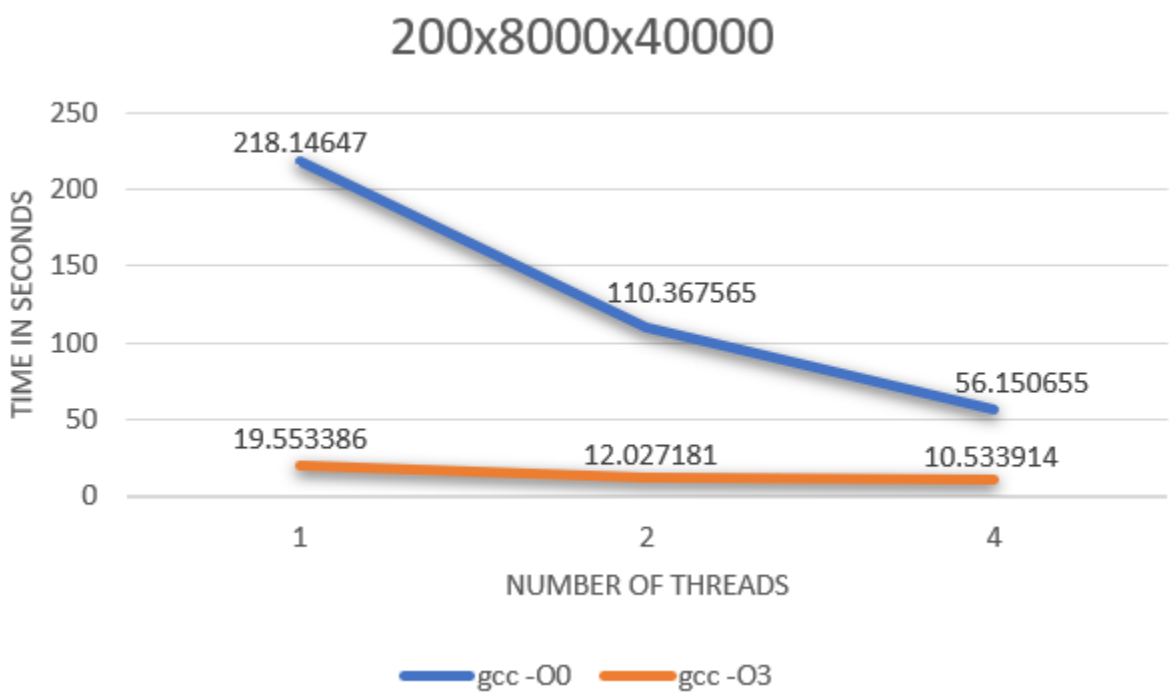
Posix Threads

Εκτελώντας τον παράλληλο κώδικα (posix threads) χρησιμοποιώντας διαφορετικό πλήθος νημάτων (threads) κάθε φορά, με τη μεταγλώττιση του προγράμματος χωρίς βελτιστοποίηση (gcc -O0) και με μέγιστη βελτιστοποίηση (gcc -O3), παρατηρήσαμε τους παρακάτω χρόνους:

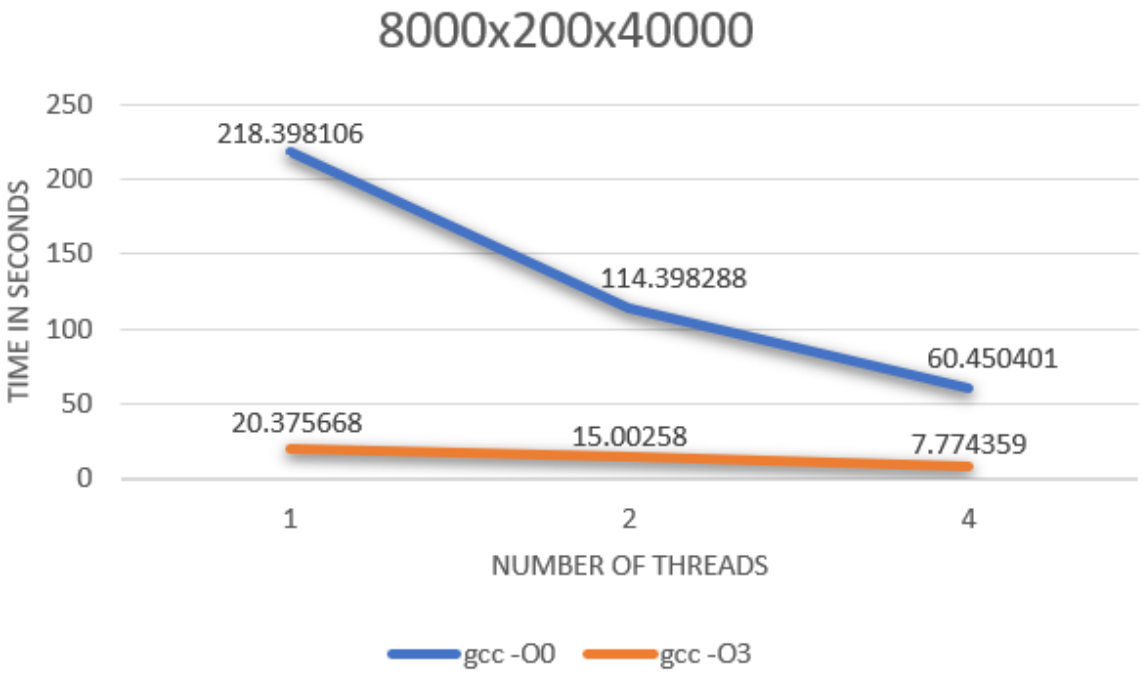
1. Με μέγεθος μητρώων N = K = M = 4000



2. Με μέγεθος μητρώων N = 200, K = 8000, M = 40000



3. Με μέγεθος μητρώων N = 8000, K = 200, M = 40000



Παρατηρήσεις:

- Παρατηρούμε με τη μεταγλώττιση του προγράμματος χωρίς βελτιστοποίηση (gcc -O0), ότι μόλις αυξήσουμε τα νήματα από 1 σε 2 και ακολούθως σε 4, ο χρόνος εκτέλεσης μειώνεται αισθητά (σχεδόν 2 φορές γρηγορότερα).

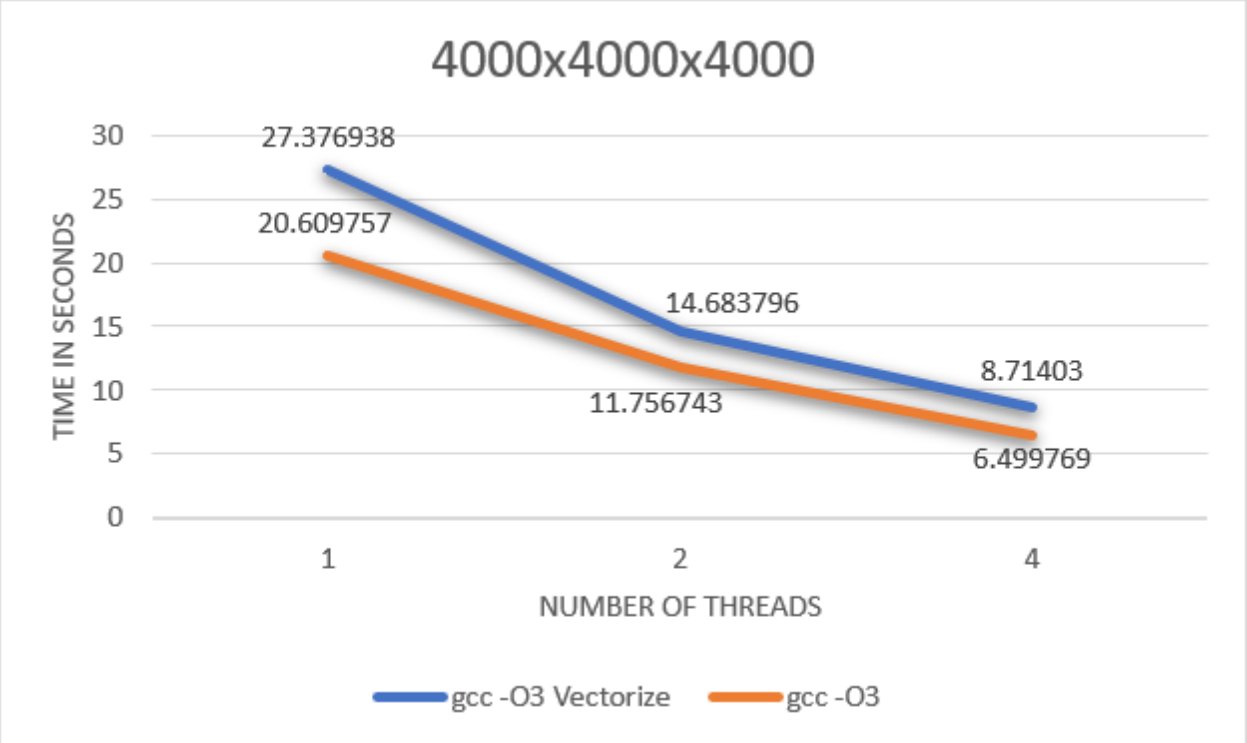
- Παρομοίως, μεταγλωτίζοντας το πρόγραμμα με βελτιστοποίηση (gcc -O3), βλέπουμε ότι ο χρόνος εκτέλεσης μειώνεται αισθητά, σε σύγκριση με τον χρόνο εκτέλεσης του προγράμματος χωρίς βελτιστοποίηση. Επιπρόσθετα, αυξάνοντας τα νήματα στο γράφημα 1, η επίδοση του προγράμματος εξακολουθεί να είναι σχεδόν 2 φορές γρηγορότερη, γεγονός που δεν παρατηρείται στα άλλα 2 γραφήματα.

- Κατά την προσπάθεια παραλληλοποίησης του προγράμματος βάσει του βρόχου k, παρατηρήσαμε ότι κρίνεται απαραίτητη η χρήση αμοιβαίου αποκλεισμού (mutex). Με αυτό το τρόπο εξασφαλίζουμε ότι πολλαπλά νήματα δεν θα προσπελάσουν την ίδια διεύθυνση μνήμης.

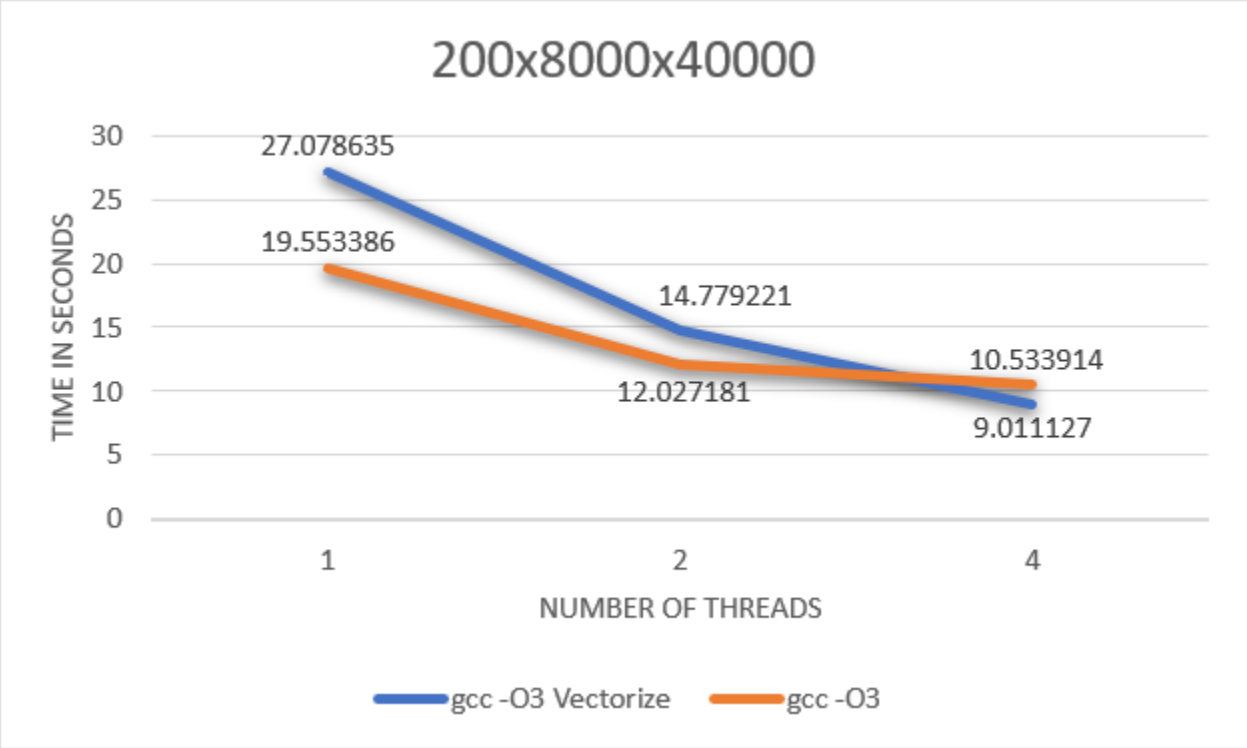
Vectorize O3

Εκτελώντας τον παράλληλο κώδικα (posix threads) χρησιμοποιώντας διαφορετικό πλήθος νημάτων κάθε φορά, με τη μεταγλώττιση του προγράμματος με βελτιστοποίηση (gcc -O3 -fno-tree-loop-vectorize -fno-tree-slp-vectorize), παρατηρήσαμε τις παρακάτω διαφορές στους χρόνους:

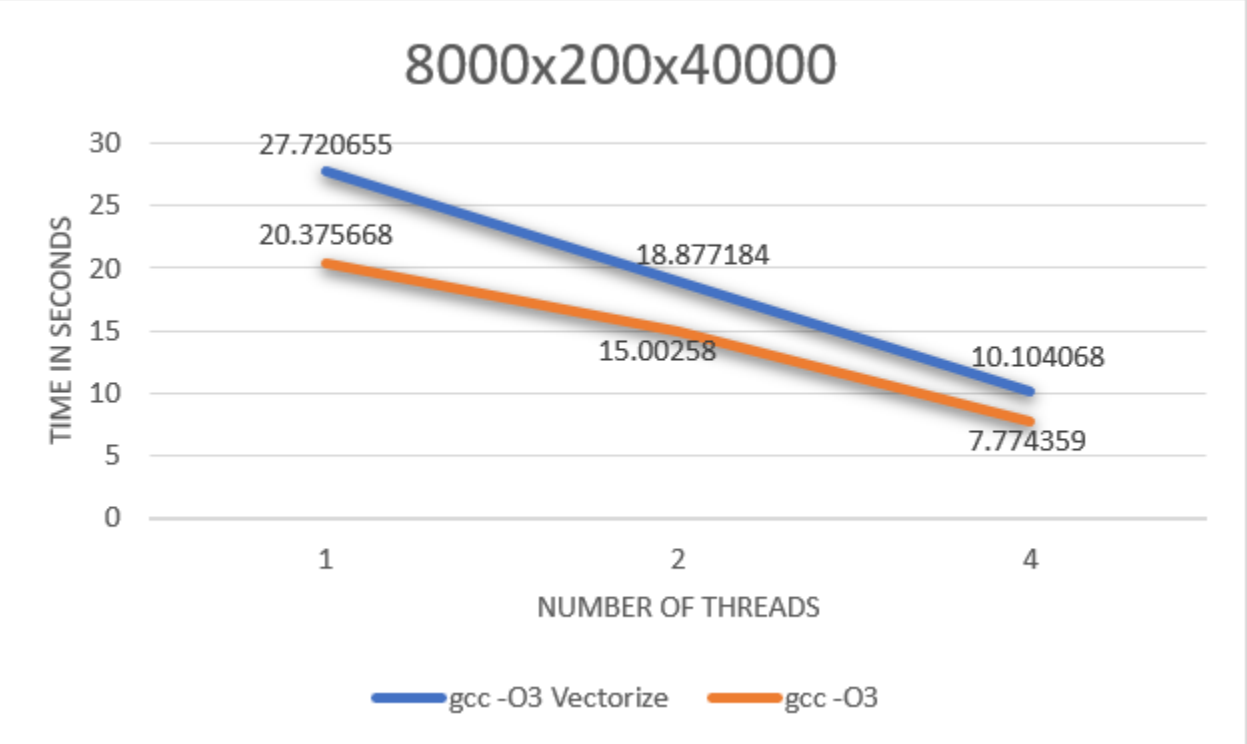
1. Με μέγεθος μητρώων N = K = M = 4000



2. Με μέγεθος μητρώων N = 200, K = 8000, M = 40000



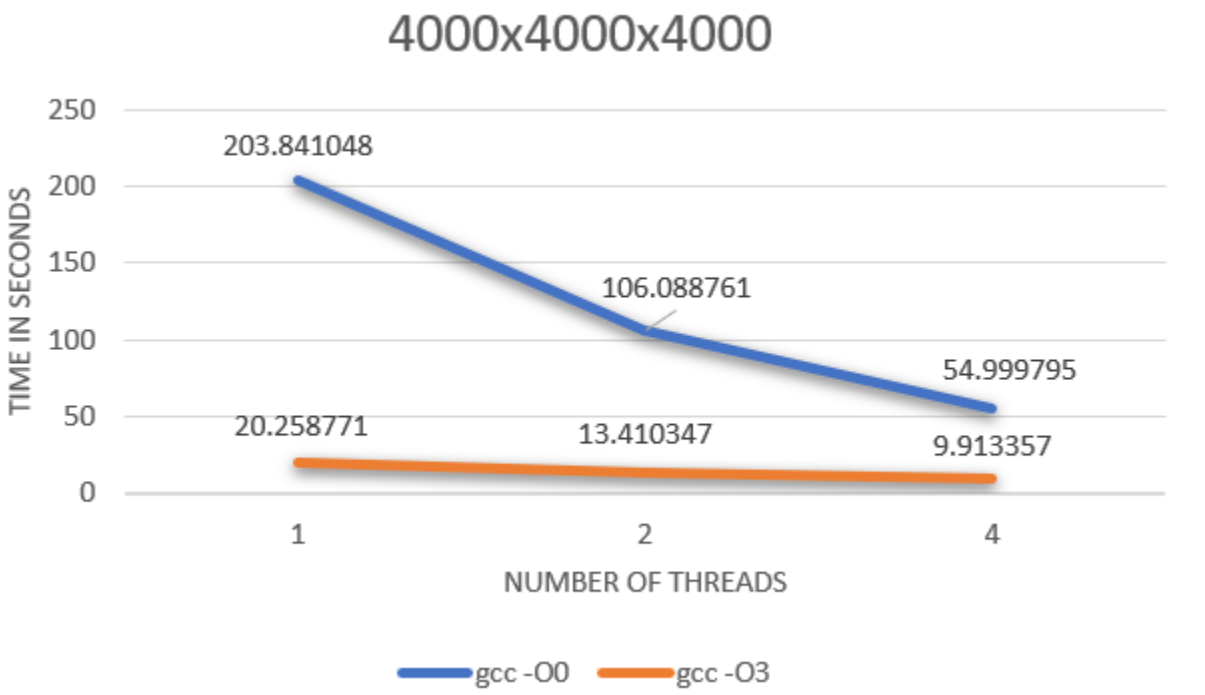
3. Με μέγεθος μητρώων N = 8000, K = 200, M = 40000



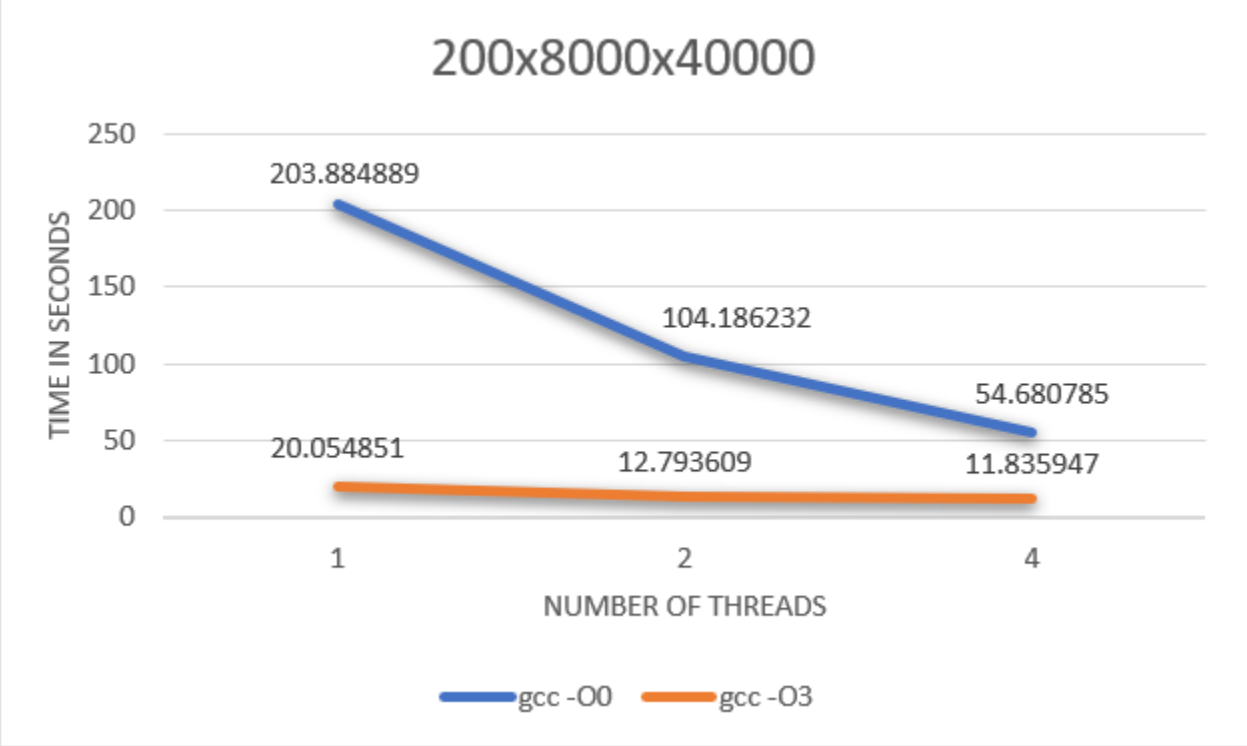
OpenMP

Εκτελώντας τον παράλληλο κώδικα (OpenMP) χρησιμοποιώντας διαφορετικό πλήθος νημάτων κάθε φορά, με τη μεταγλώττιση του προγράμματος χωρίς βελτιστοποίηση (gcc -O0) και με μέγιστη βελτιστοποίηση (gcc -O3), παρατηρήσαμε τους παρακάτω χρόνους:

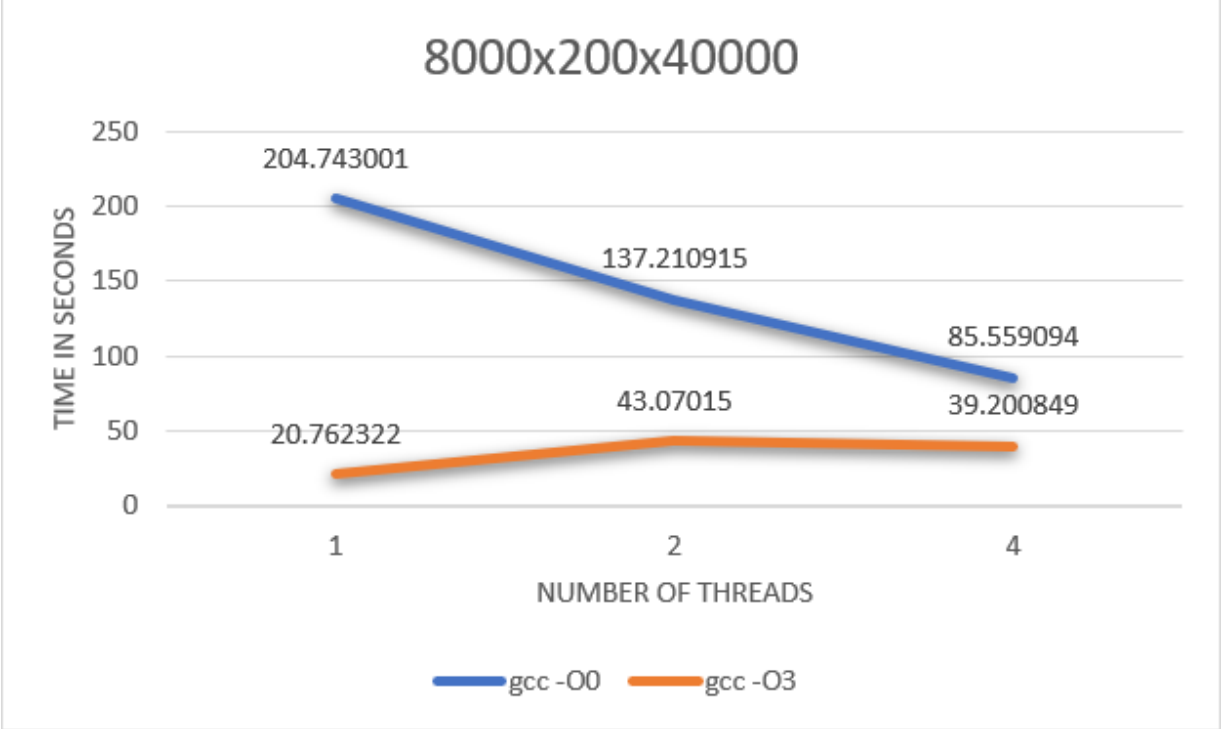
1. Με μέγεθος μητρώων N = K = M = 4000



2. Με μέγεθος μητρώων N = 200, K = 8000, M = 40000



3. Με μέγεθος μητρώων N = 8000, K = 200, M = 40000



Παρατηρήσεις:

- Παρατηρούμε ,πως η περίπτωση με μέγεθος μητρώων N = 8000, K = 200, M = 40000 είναι η μόνη που οι χρόνοι εκτέλεσης της αυξάνονται με την χρήση νημάτων.

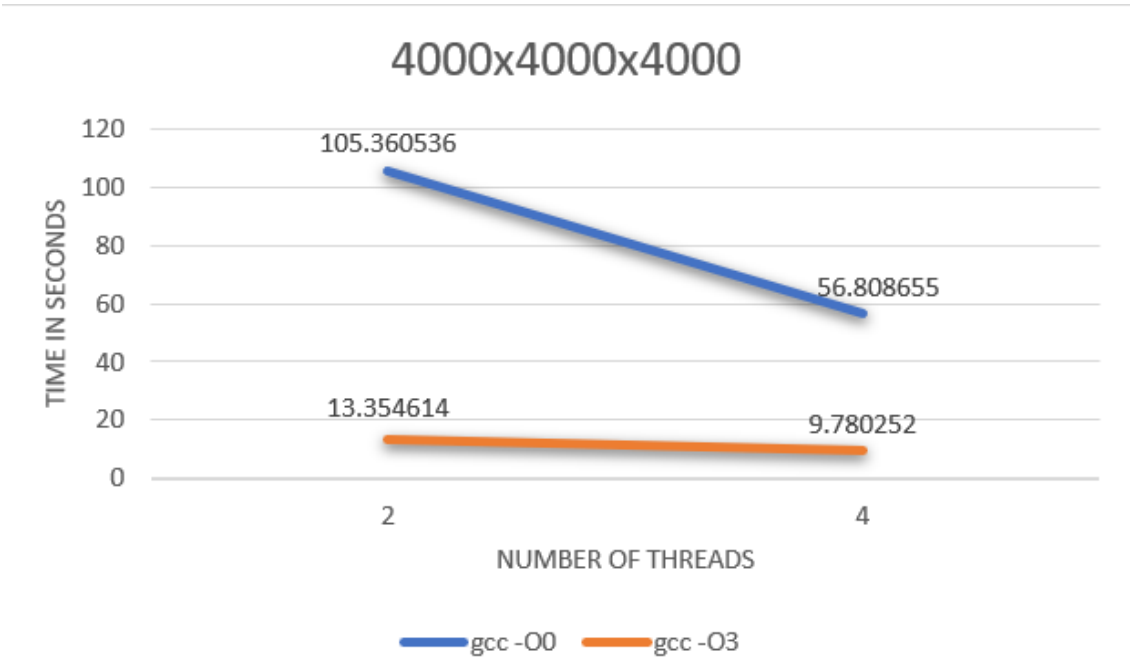
Ερώτημα 3

Εκτελώντας τον παράλληλο κώδικα (OpenMP) χρησιμοποιώντας διαφορετικό πλήθος νημάτων (threads) κάθε φορά, με τη μεταγλώττιση του προγράμματος χωρίς βελτιστοποίηση (gcc -O0) και με μέγιστη βελτιστοποίηση (gcc -O3), με την χρήση διαφορετικών προσεγγίσεων για τον διαμοιρασμό των επαναλήψεων των βρόγχων σε νήματα, παρατηρήσαμε τους παρακάτω χρόνους ανά μέγεθος μητρώων:

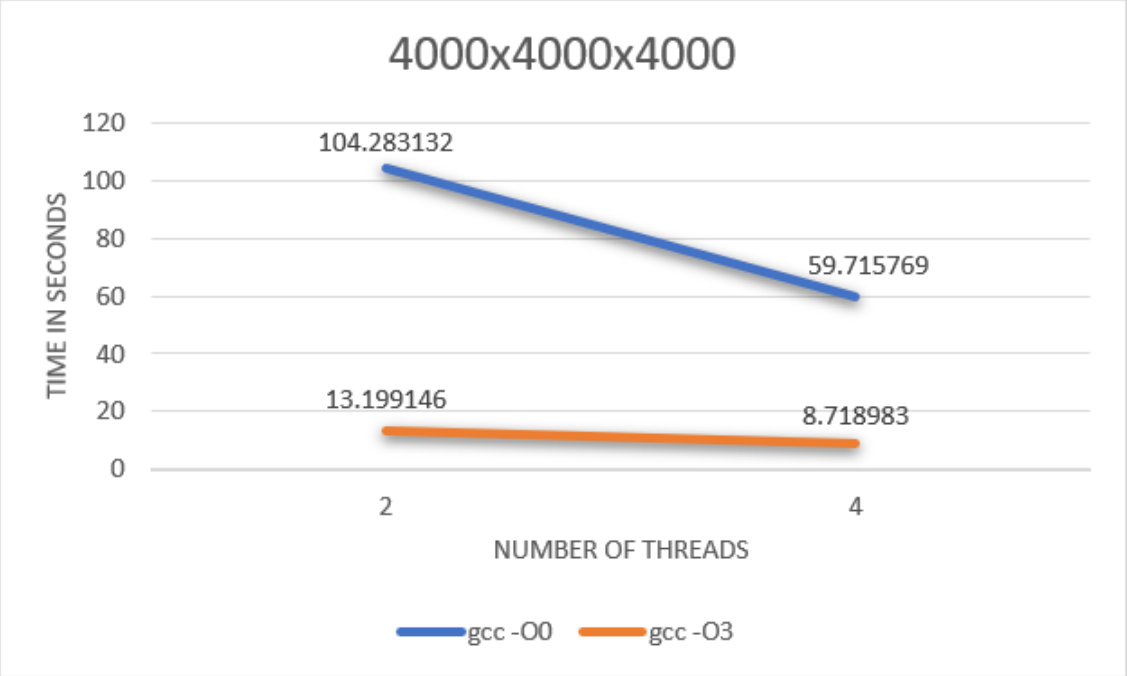
Μέγεθος μητρώων N = K = M = 4000

Dynamic Schedule

Με chunk size = 4

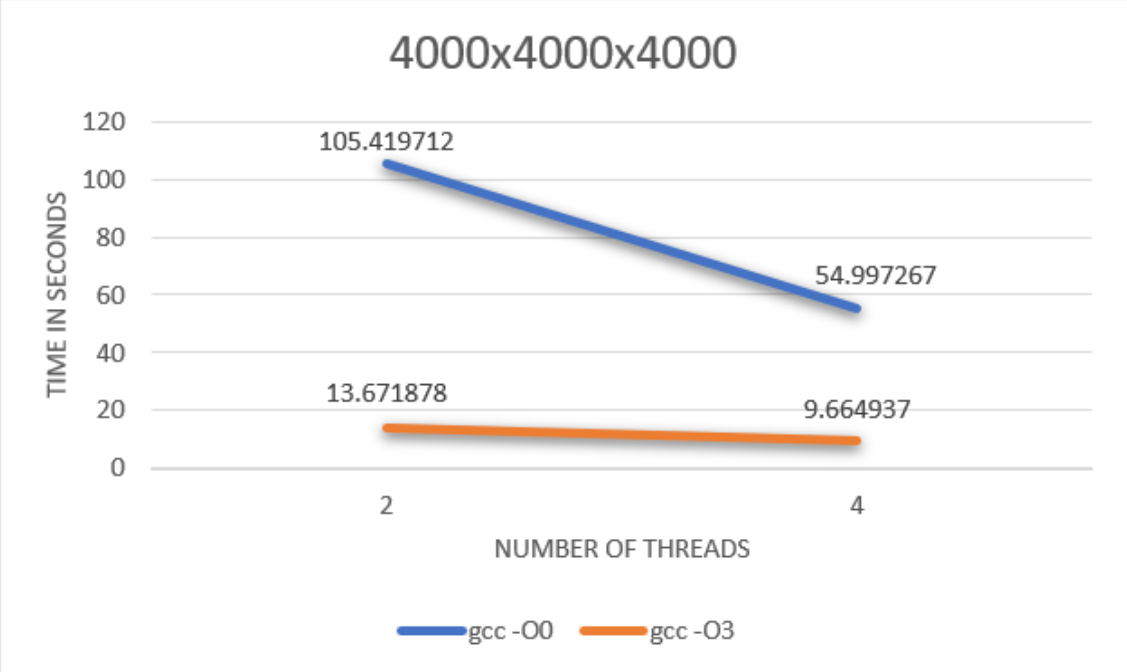


Με chunk size = 100

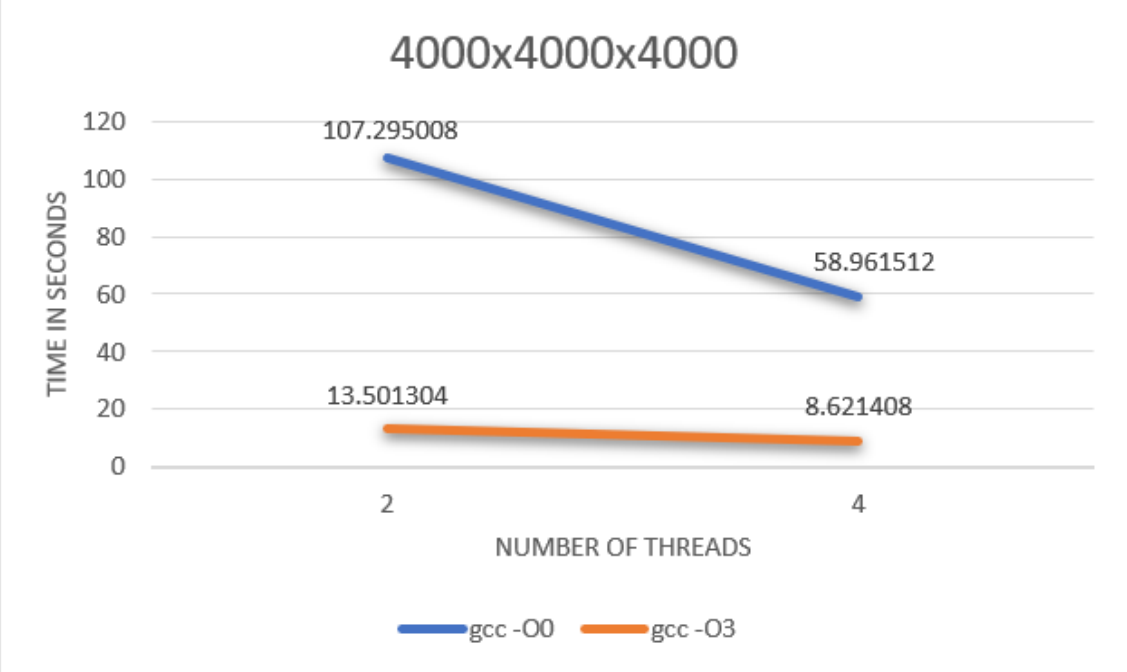


Guided Schedule

Με chunk size = 4

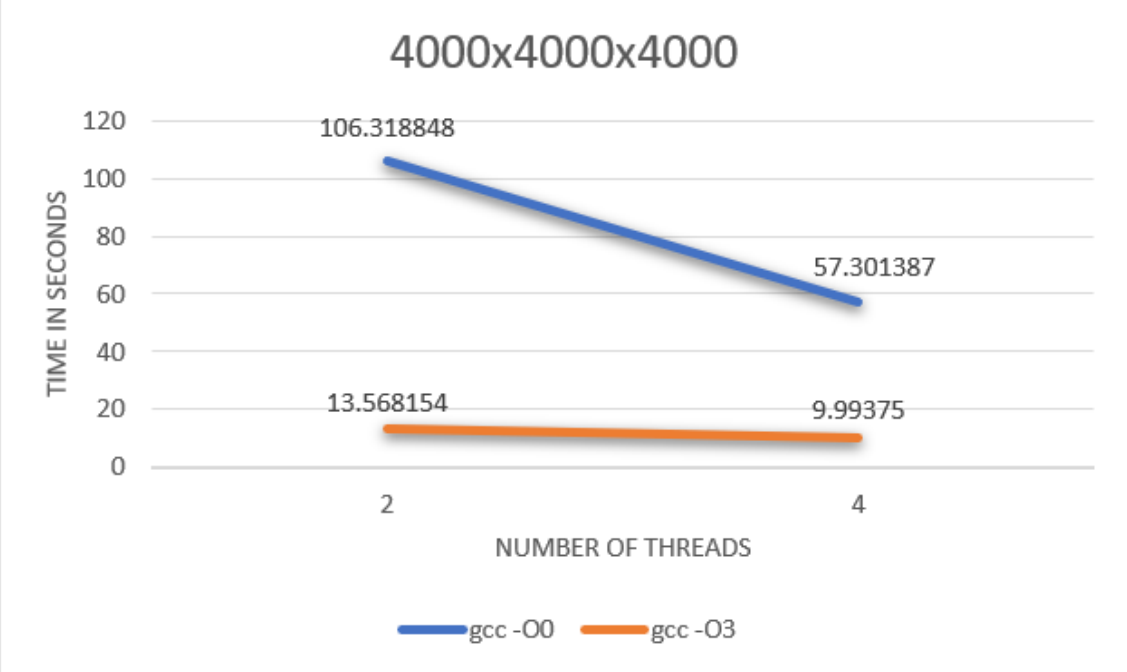


Με chunk size = 100

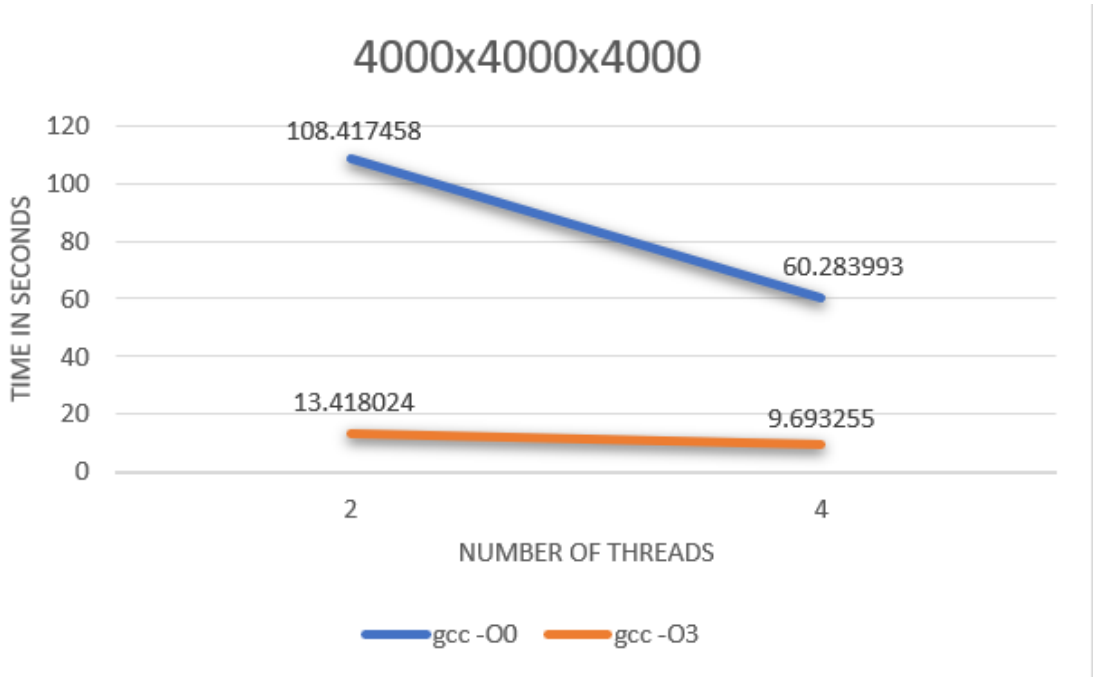


Static Schedule

Με chunk size = 4



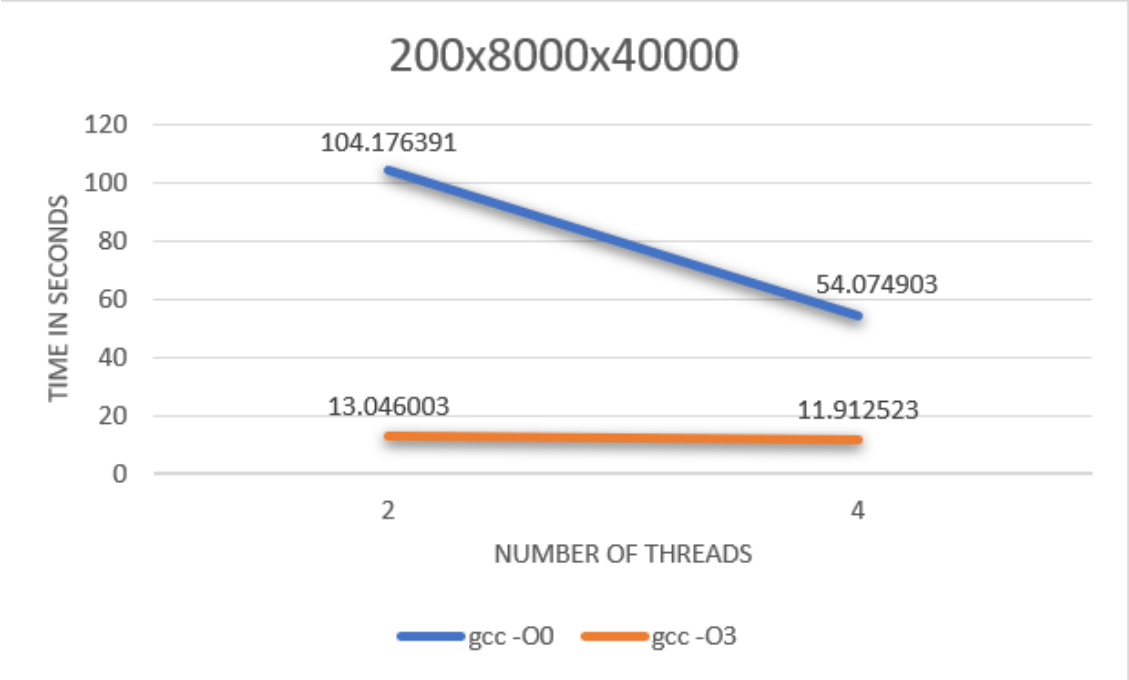
Mε chunk size = 100



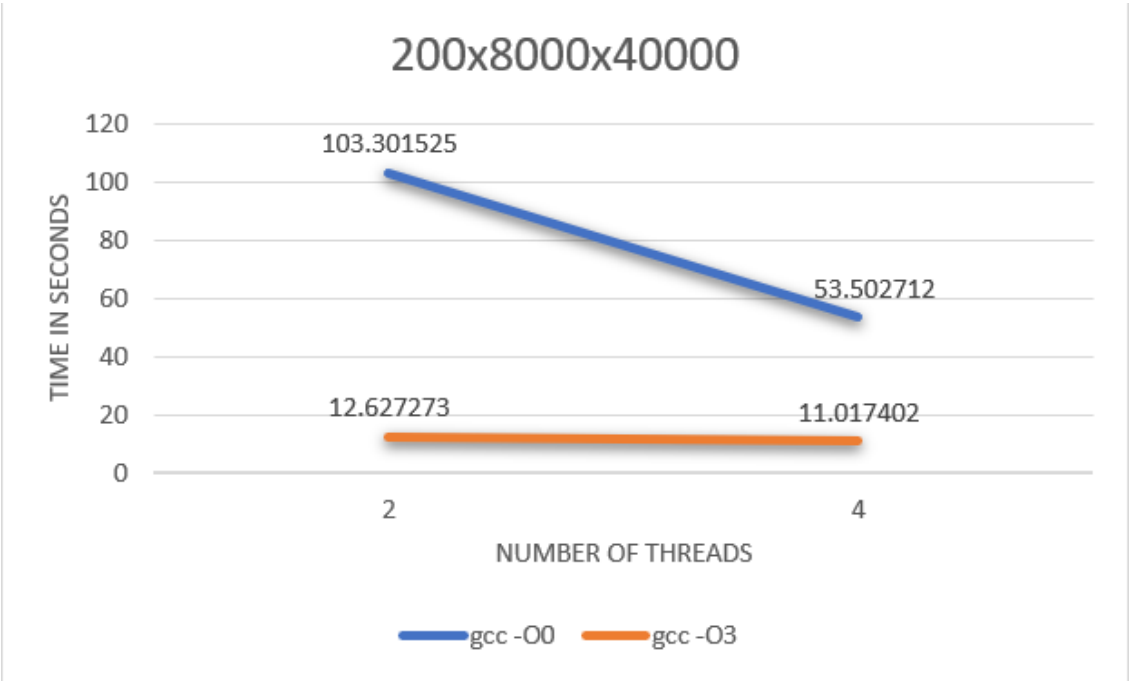
Μέγεθος μητρώων N = 200, K = 8000, M = 40000

Dynamic Schedule

Με chunk size = 4

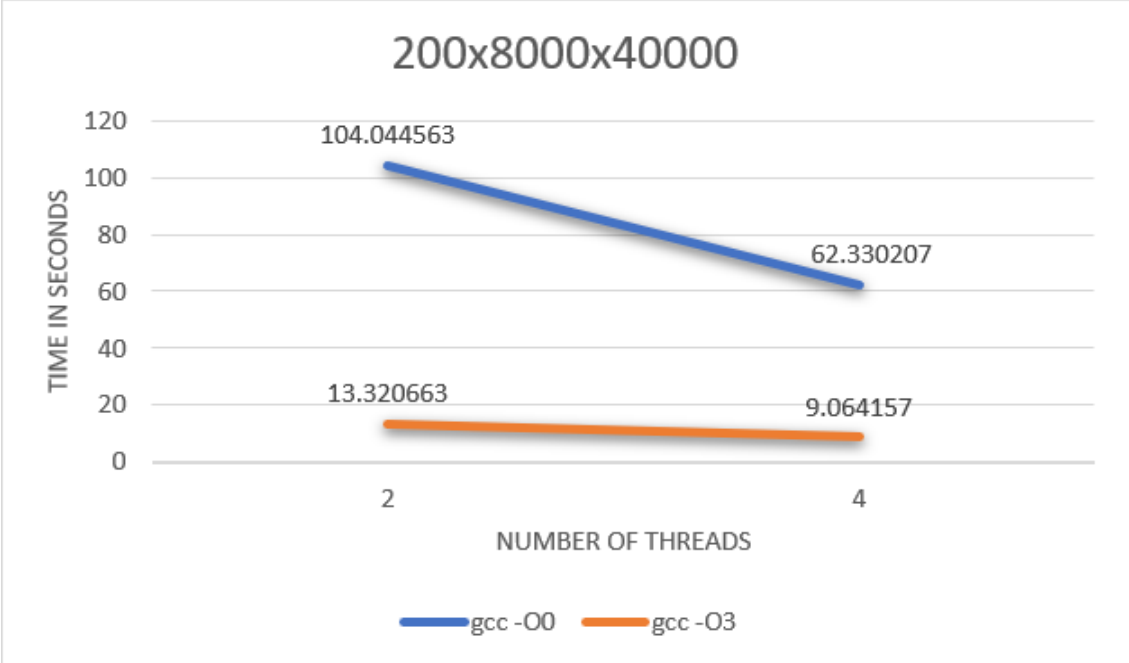


Με chunk size = 100

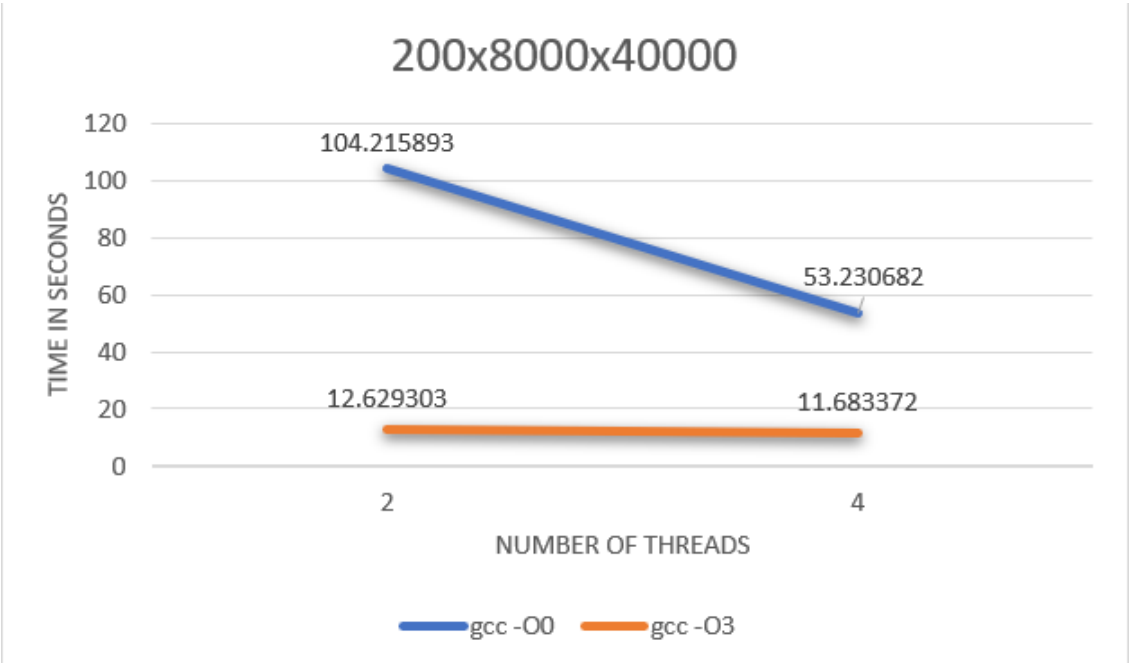


Guided Schedule

Με chunk size = 4

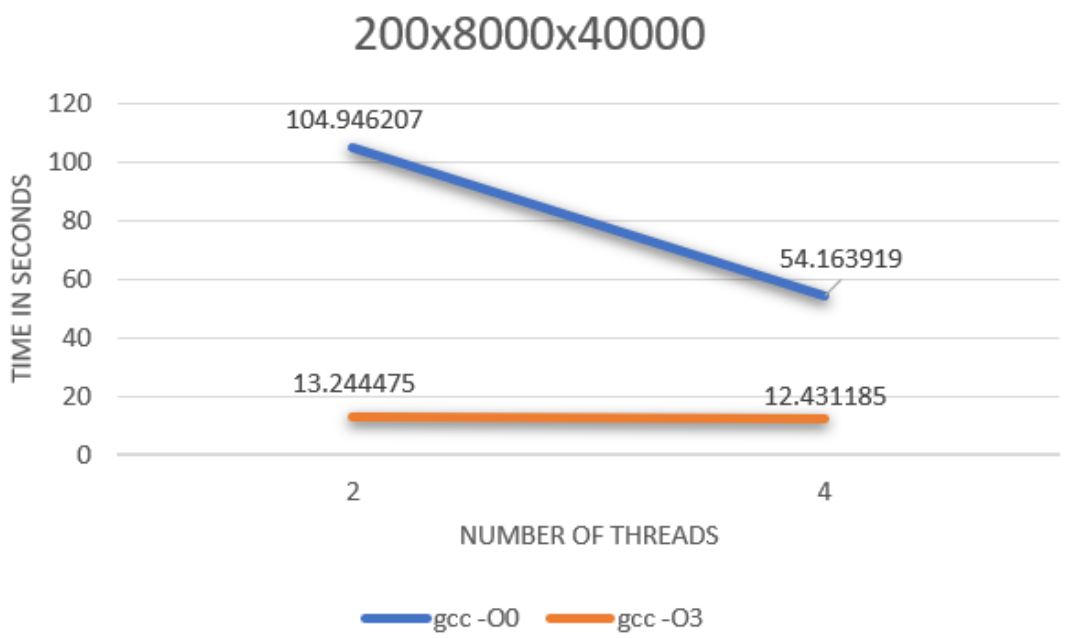


Με chunk size = 100

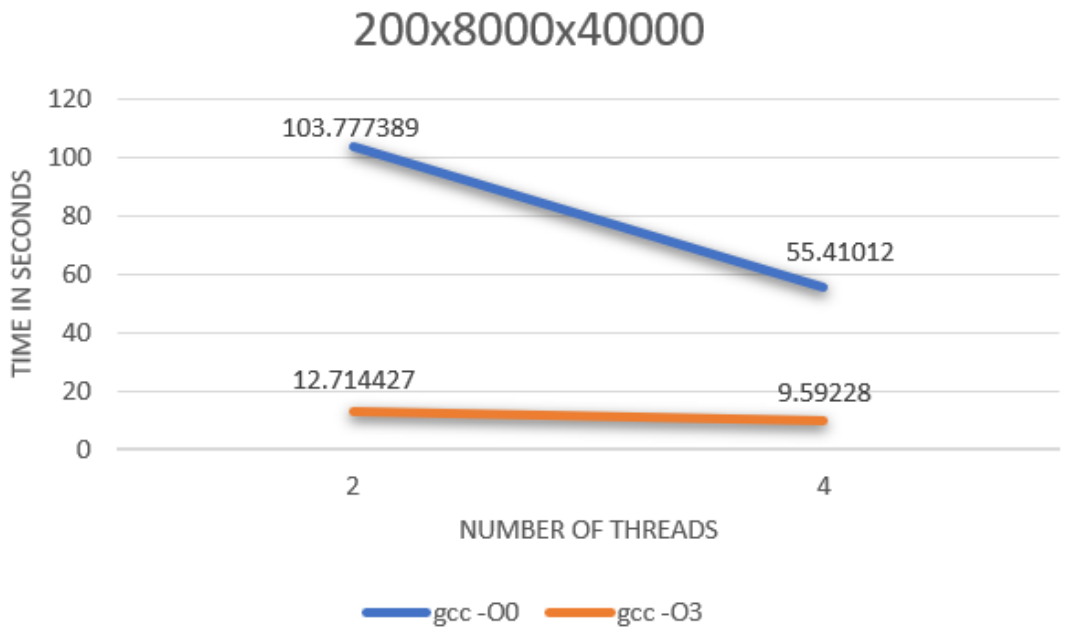


Static Schedule

Με chunk size = 4



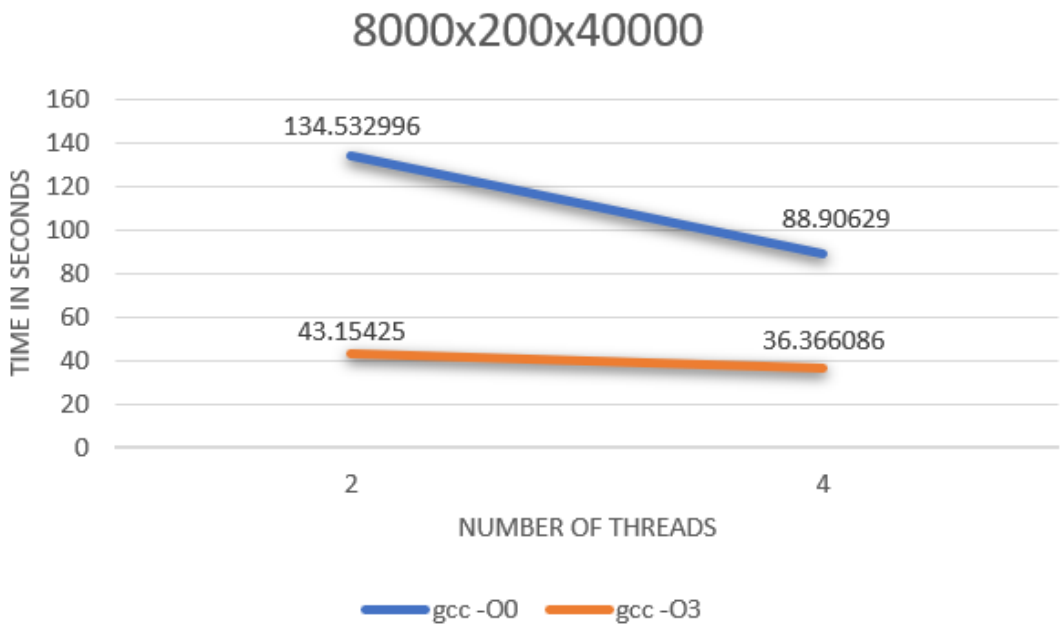
Με chunk size = 100



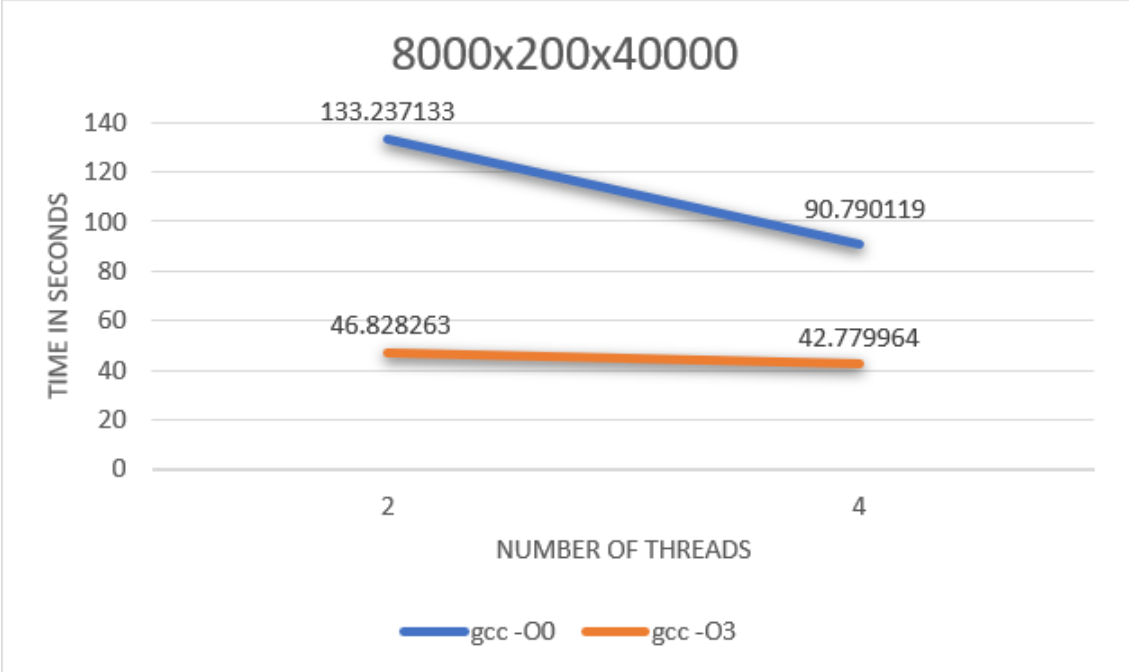
Μέγεθος μητρώων N = 8000, K = 200, M = 40000

Dynamic Schedule

Με chunk size = 4

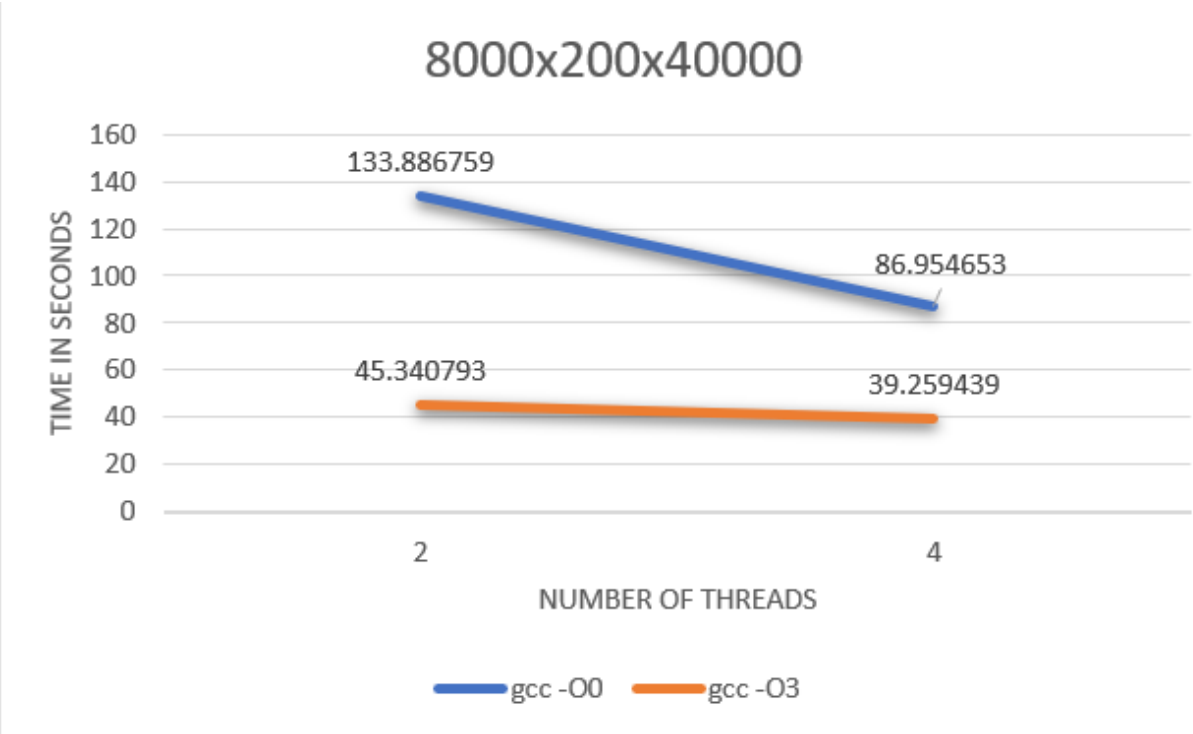


Με chunk size = 100

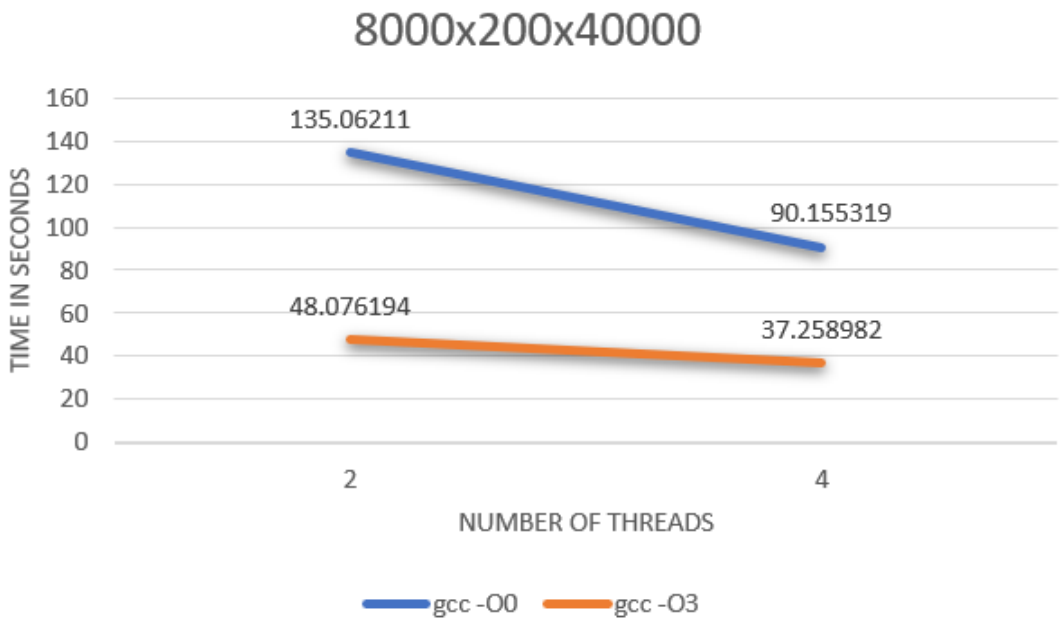


Guided Schedule

Με chunk size = 4

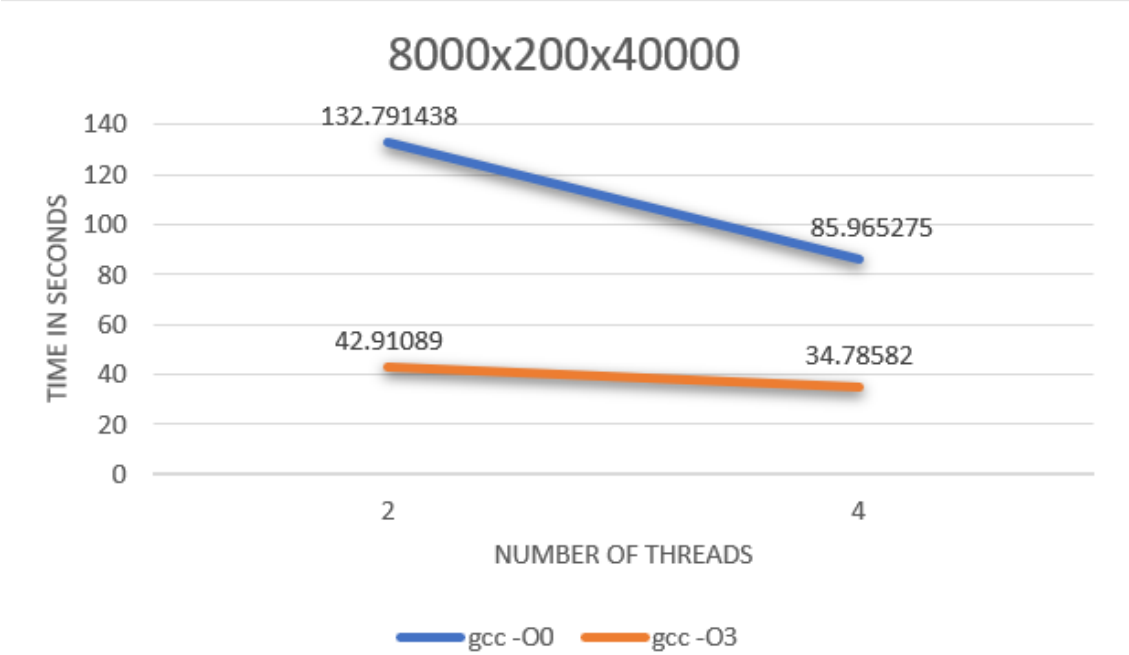


Με chunk size = 100

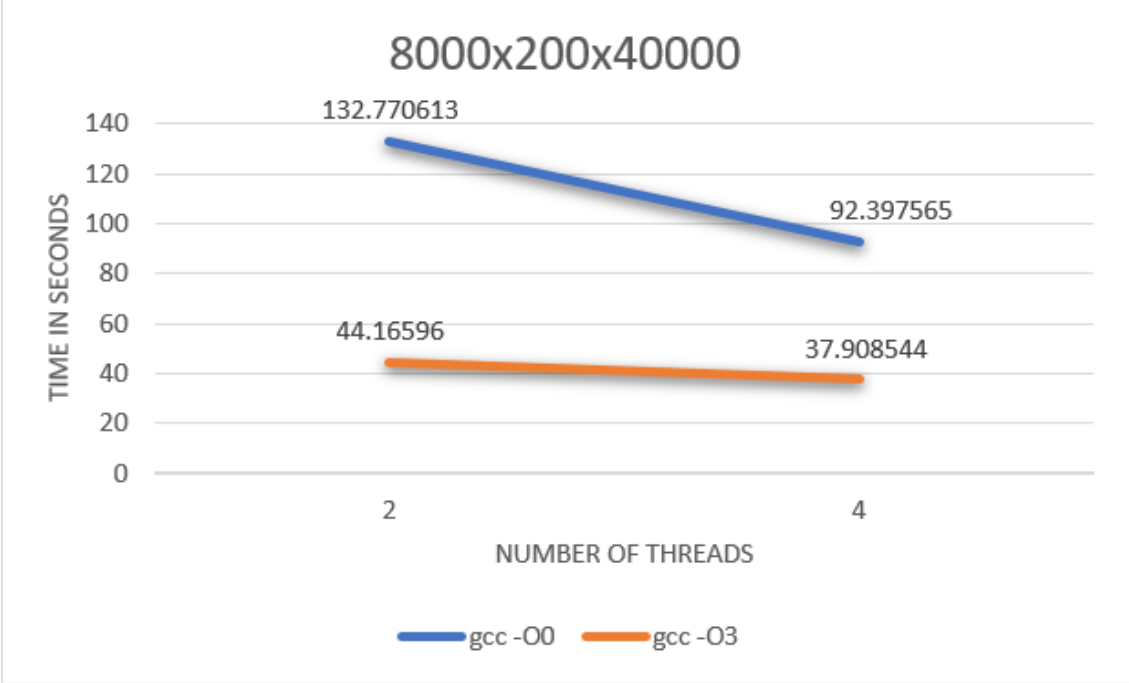


Static Schedule

Με chunk size = 4



Με chunk size = 100



Παρατηρήσεις:

- Παρατηρούμε ότι μεταξύ των διαφορετικών προσεγγίσεων που δοκιμάσαμε (dynamic, guided και static) δεν υπάρχουν σημαντικές διαφορές στους χρόνους εκτέλεσης.

- Γίνεται αντιληπτό πως οι χρόνοι στις υλοποιήσεις με chunk size 100, είναι ελάχιστα μικρότεροι σε σχέση με τις αντίστοιχες υλοποιήσεις που διαθέτουν chunk size 4.

- Στην περίπτωση με μέγεθος μητρώων N = 8000, K = 200, M = 40000 οι χρόνοι εκτέλεσης είναι αισθητά μεγαλύτεροι, σε σύγκριση με τις υπόλοιπες διαστάσεις μητρώων.