Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Παράλληλα Συστήματα
Εργασία 2

Ονομ/υμο και ΑΜ

Ονομ/υμο:ΙΩΑΝΝΟΥ ΝΙΚΟΛΑΣ ΑΜ: 1115202200054
Ονομ/υμο: ΑΝΤΡΕΪ-ΑΝΤΡΙΑΝ ΠΡΕΝΤΑ ΑΜ: 1115202000263

Άσκηση 2.1

Πως τρέχουμε το πρόγραμμα

Για να τρέξουμε το πρόγραμμα χρησιμοποιούμε το Makefile που υπάρχει και εκτελούμε την παρακάτω εντολή:

```
./main21 <generations> <size> <method> <threads>
```

Στη θέση του <generations> βάζουμε τον αριθμό των γενών που θα γίνουν. Στη θέση του <size>θέτουμε το μήκος της πλευράς που θα έχει το πλέγμα και θα έχει μέγεθος size * size. Στη θέση του <method> βάζουμε την μέθοδο που θέλουμε να εκτελέσουμε το πρόγραμμα. Πιο συγκεκριμένα, με την τιμή 0 εκτελούμε σειριακά, ενώ με την τιμή 1 παράλληλα. Στη θέση του <threads> βάζουμε τον αριθμό των threads που θα χρησιμοποιήσουμε στην μέθοδο του παραλληλισμού (αν έχουμε). Μπορεί να παραληφθεί αν έχουμε σειριακή μέθοδο. Οι τιμές που θα δοθούν θα πρέπει να είναι μεγαλύτερες του μηδενός, αλλιώς το πρόγραμμα δεν θα εκτελεστεί.

Περιγραφή προβλήματος

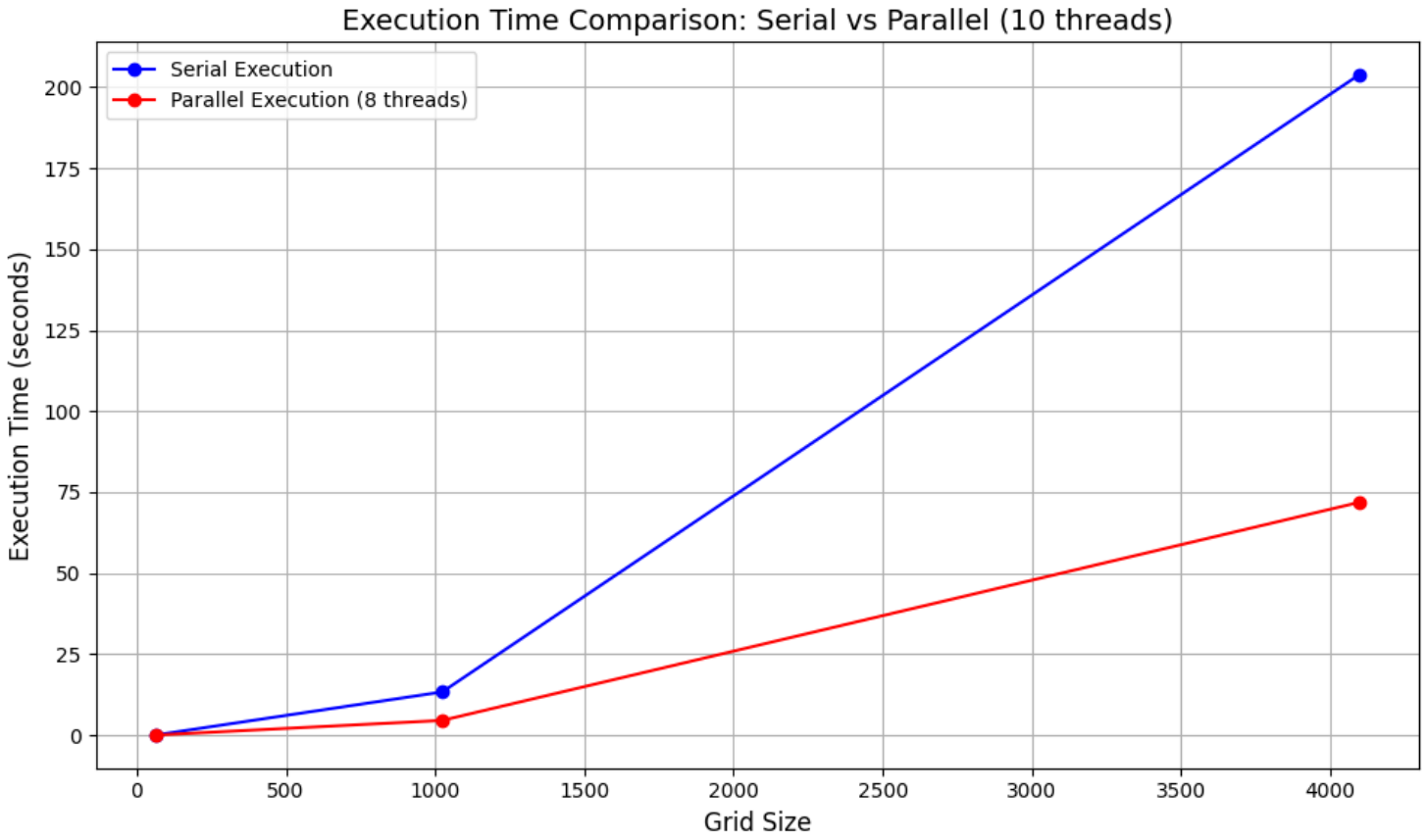
Η άσκηση ζητάει να υλοποιήσουμε το παιχνίδι Game of Life με 2 μεθόδους. Την σειριακή και την παράλληλη μέθοδο,όπου θα γίνει με τη χρήση της βιβλιοθήκης OpenMP.

Τρόπος επίλυσης του προβλήματος

Για να εκτελεστεί το πρόγραμμα πρέπει να δοθούν κάποιες τιμές που θα καθορίσουν την ροή του προγράμματος, δηλαδή το πόσες γενιές θα γίνουν, το μέγεθος του πλέγματος, την μέθοδο που θα χρησιμοποιηθεί και αν έχει επιλεχθεί η μέθοδος του παραλληλισμού, τότε και ο αριθμός των threads. Αρχικά, γίνεται αποθήκευση των δεδομένων που δόθηκαν και ελέγχονται αν είναι αποδεκτές. Μετά, δημιουργείται ένα αρχικό πλέγμα τυχαία με βάση το μέγεθος που δόθηκε στο command line και άλλο ένα το οποίο θα θεωρείται σαν temp. Στη συνέχεια, γίνεται το πέρασμα από την μια γενιά σε μία άλλη με την μέθοδο που επιλέχθηκε μέσω της συνάρτησης void updategen(int **gen, int n, int **newgen, int parallelism, int totalthreads). Μετά από κάθε μεταβίβαση εμφανίζεται η νέα γενιά και τελειώνει η διαδικασία αυτή όταν φτάσουμε στην γενιά που δόθηκε. Στο τέλος, εμφανίζεται ο χρόνος εκτέλεσης του προγράμματος.

Αποτελέσματα

Παρακάτω δίνεται ένα γράφημα στο οποίο περιγράφεται ο χρόνος εκτέλεσης του προγράμματος με τη χρήση των 2 μεθόδων για διάφορες τιμές του μεγέθους του πλέγματος. Θεωρούμε ότι δεν γίνεται η εμφάνιση του πλέγματος από τη μια γενιά σε μια άλλη, έχουμε 1000 γενιές και στην μέθοδο του παραλληλισμού χρησιμοποιούμε 8 threads. Επίσης, τα αποτελέσματα είναι ο μέσος όρος 4 εκτελέσεων για κάθε περίπτωση.



Το πρόγραμμα εκτελέστηκε στον επεξεργαστή AMD Ryzen 5 7520U with Radeon Graphics (2.80 GHz, 4 Cores, 8 Threads), σε WSL Ubuntu 2.0 με έκδοση μεταγλωττιστή gcc (Ubuntu 9.4.0-1ubuntu1 20.04.1) 9.4.0.

Σχολιασμός Αποτελεσμάτων

Παρατηρούμε ότι η μέθοδος του παραλληλισμού είναι πολύ καλύτερη από τη σειριακή μέθοδο. Αυτό συμβαίνει επειδή, κατά την εξέλιξη από μια γενιά σε μια άλλη, χρησιμοποιούμε τα δηλωμένα threads και τα εκμεταλλευόμαστε. Ο τρόπος με τον οποίο τα εκμεταλλευόμαστε είναι αναθέτοντας κάθε γραμμή (ή μπλοκ γραμμών) σε ένα thread, το οποίο κάνει τις κατάλληλες αλλαγές. Οι αλλαγές γίνονται παράλληλα, με αποτέλεσμα να επιτυγχάνουμε την μεταβίβαση σε πιο σύντομο χρονικό διάστημα σε αντίθεση με τη σειριακή μέθοδο.

Άσκηση 2.2

Περιγραφή Προβλήματος

Η άσκηση ζητάει να παραλληλοποιηθεί η αντικατάσταση που ακολουθεί μετά την απαλοιφή Gauss . Συγκεκριμένα, ζητείται να παραλληλοποιηθεί ο κατά σειρά και κατά γραμμή αλγόριθμος της αντικατάστασης προς τα πίσω, που χρησιμοποιούνται για την εύρεση του διανύσματος x.

Ορισμός του Προβλήματος

Ο αλγόριθμος της αντικατάστασης πίσω είναι μια διαδικασία για την επίλυση ενός συστήματος γραμμικών εξισώσεων με άνω τριγωνική μήτρα. Ο αλγόριθμος αυτός είναι σειριακός και πρέπει να παραλληλοποιηθεί ώστε να εκμεταλλευτεί τη δυνατότητα των σύγχρονων υπολογιστών.

Περιγραφή Λύσης

Η λύση αποτελείται από τα εξής αρχεία:

Αρχεία Προγράμματος

- `back_subst.c`: το κύριο πρόγραμμα
- `back_subst_modl.c`: το μοδυλε
- `back_subst.h`: το ηεαδερ

Συλλογή Ορισμάτων

Το πρόγραμμα καλείται με τα εξής ορίσματα:

$$./back\_subst < size >< method >< algorithm >< thread\_count >$$

όπου:

- `size`: αριθμός γραμμών και στηλών του επαυξημένου πίνακα A
- `method`: σειριακός ή παράλληλος αλγόριθμος (δέχεται 'serial' ή 'parallel')
- `algorithm`: αντικατάσταση κατά γραμμή ή στήλη (δέχεται 'row' ή 'column')
- `thread_count`: αριθμός νημάτων

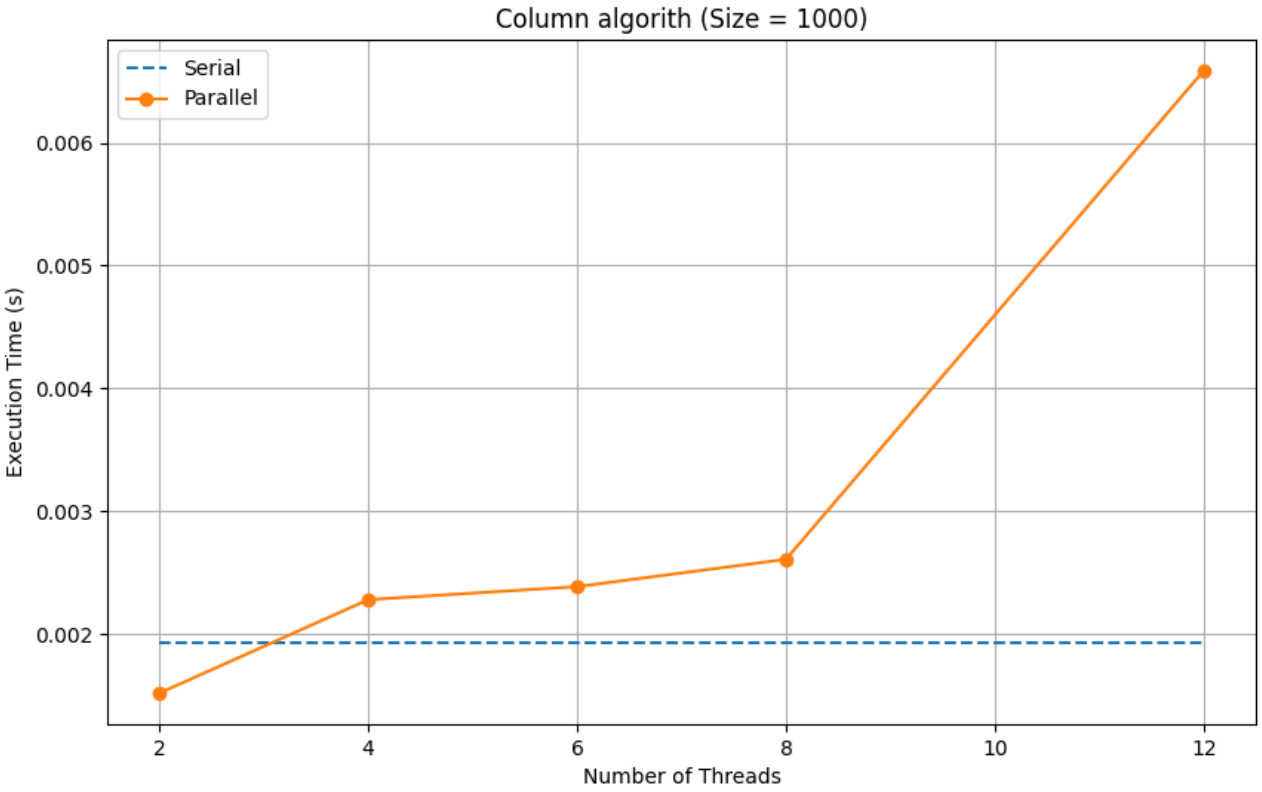
Προεπεξεργασία Πίνακα

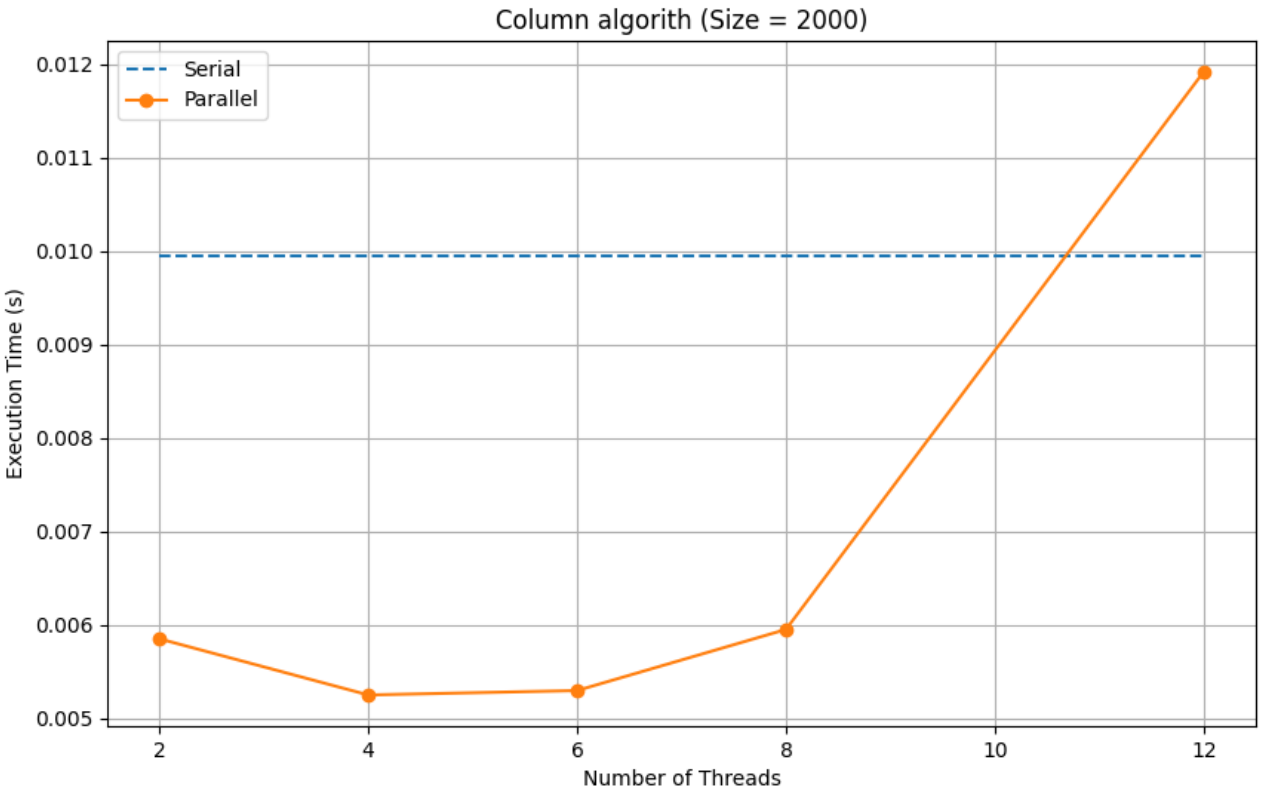
Πριν την αντικατάσταση κατά γραμμή/στήλη, γίνεται προεπεξεργασία:

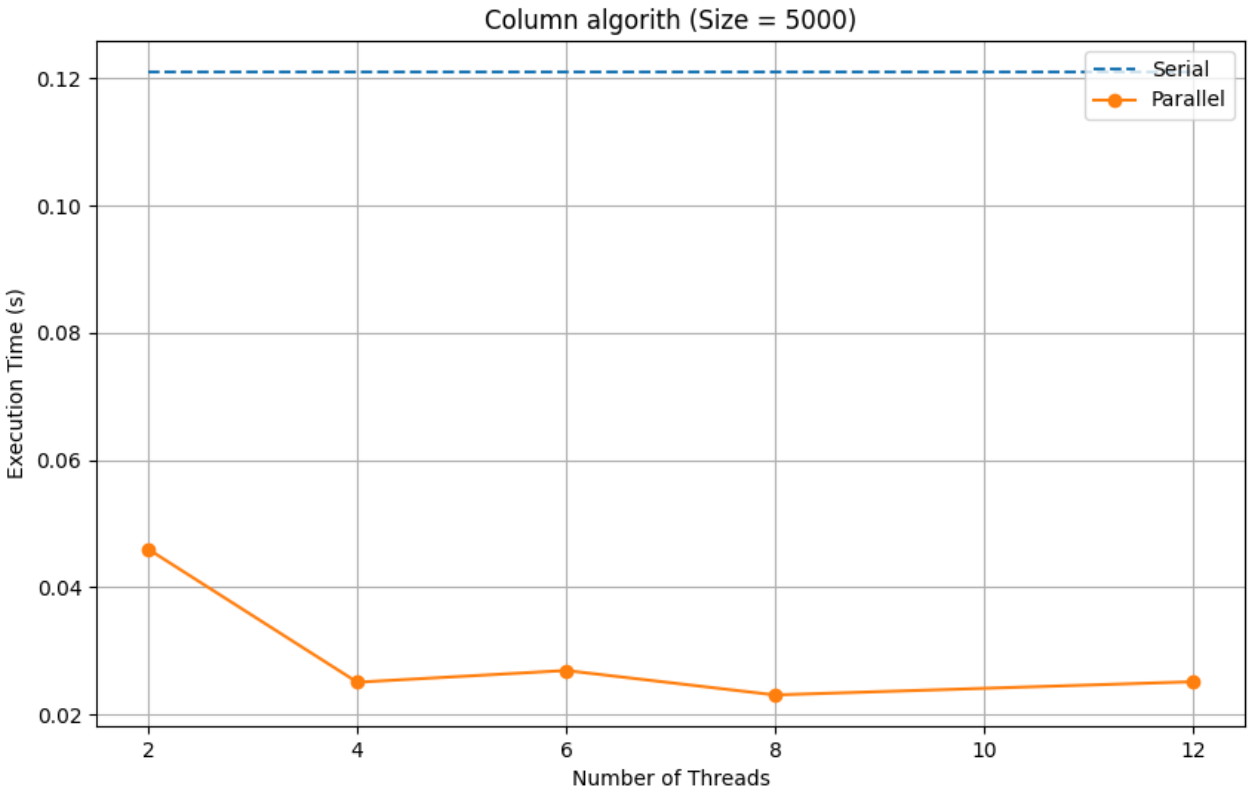
1. Δέσμευση δυναμικά ενός επαυξημένου πίνακα A
2. Παραγωγή τυχαίων τιμών για τον πίνακα A καλώντας τη συνάρτηση `generate_matrix()`
3. Εφαρμογή της συνάρτησης `gauss_elimination()`, η οποία μετατρέπει τον πίνακα A σε άνω τριγωνικό

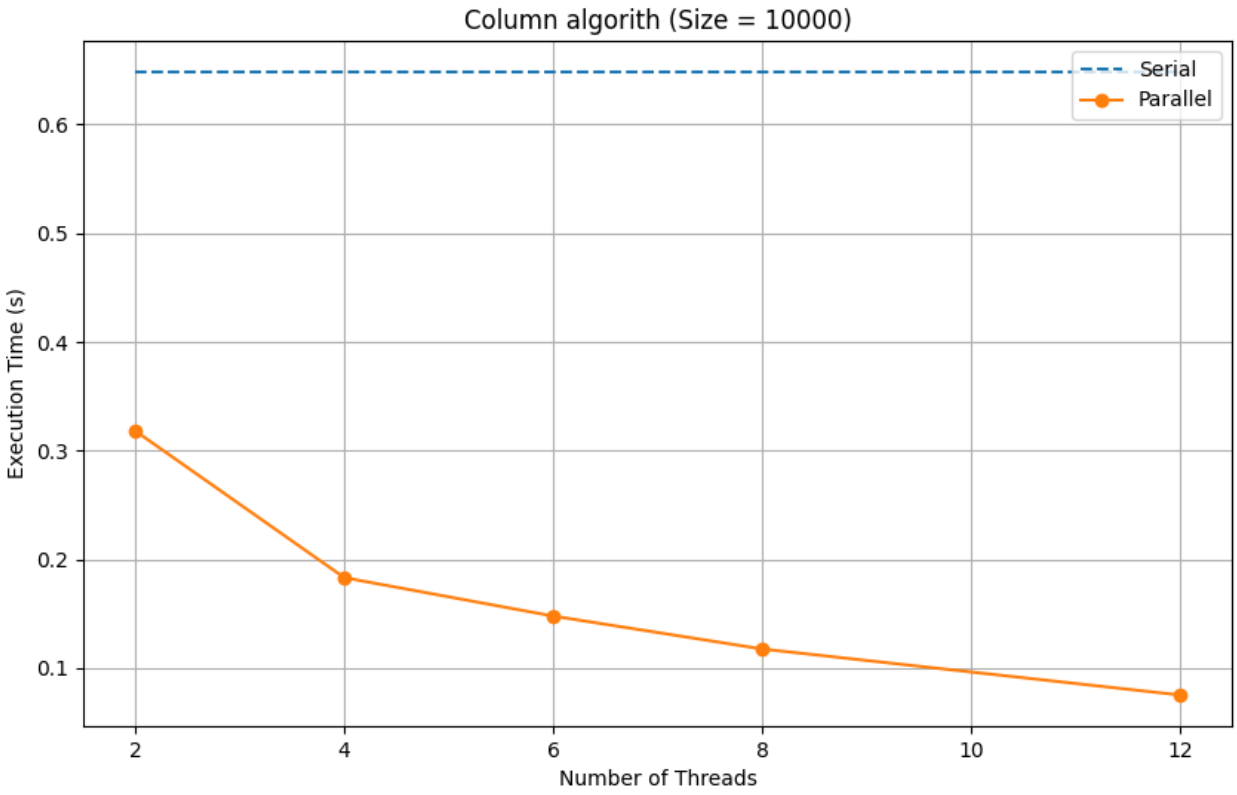
Αποτελέσματα

"Κατά στήλη" Αλγόριθμος

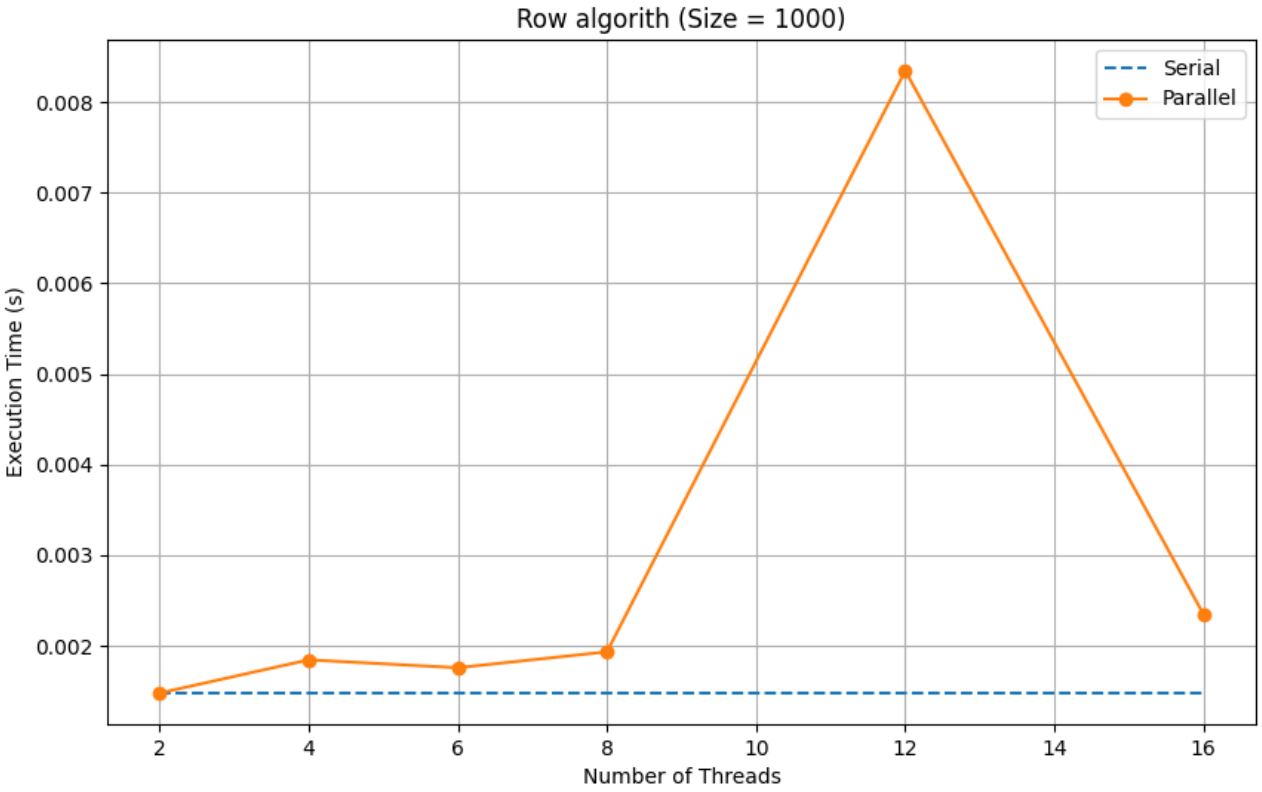


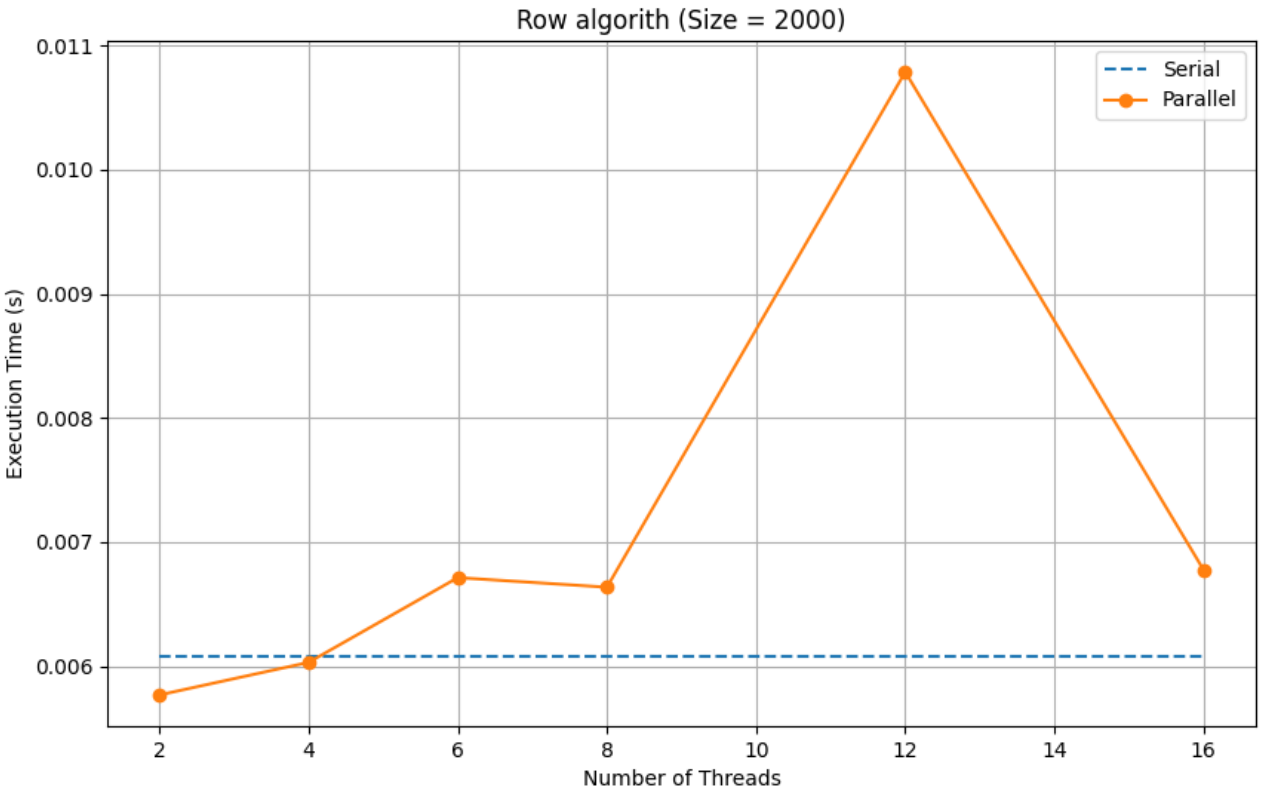

Column algorith (Size = 2000)
Serial
Parallel
Execution Time (s)
0.012
0.011
0.010
0.009
0.008
0.007
0.006
0.005
2
4
6
8
10
12
Number of Threads

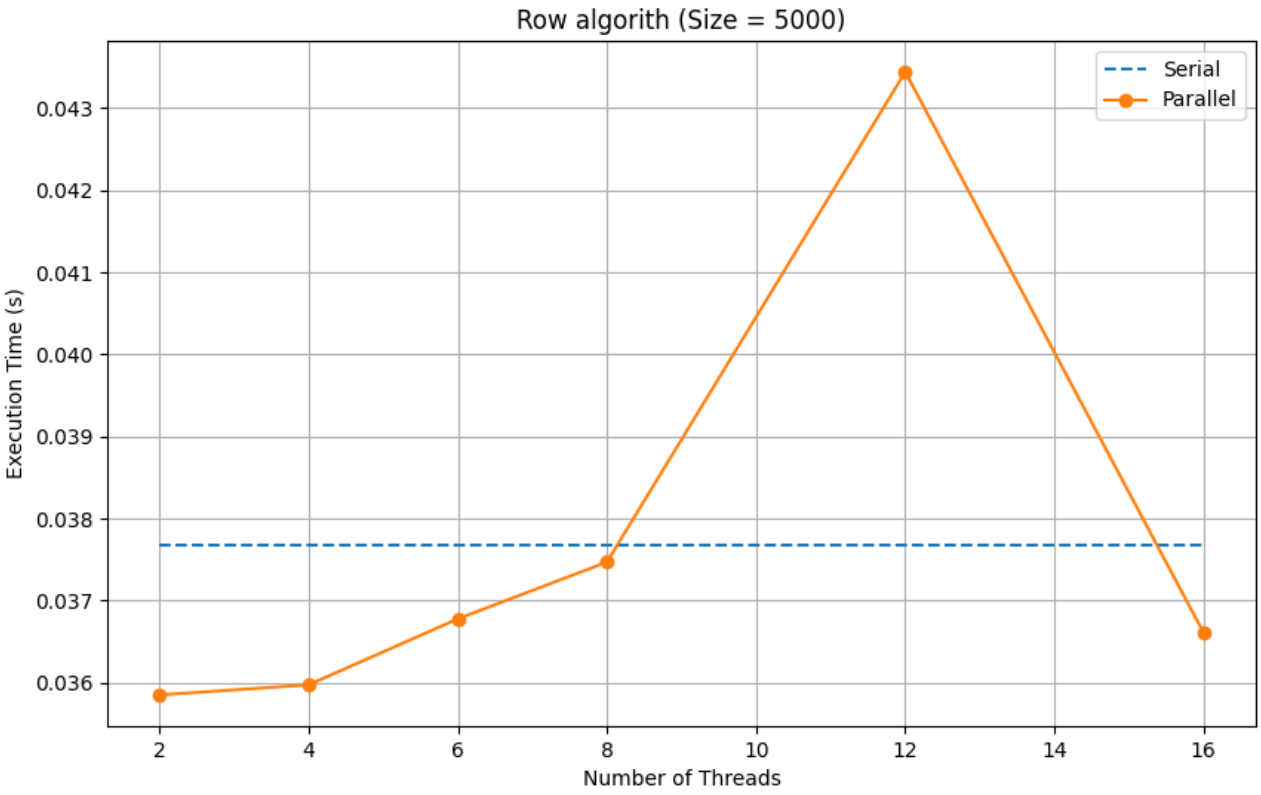

Column algorith (Size = 5000)
Serial
Parallel
Execution Time (s)
0.12
0.10
0.08
0.06
0.04
0.02
2
4
6
8
10
12
Number of Threads

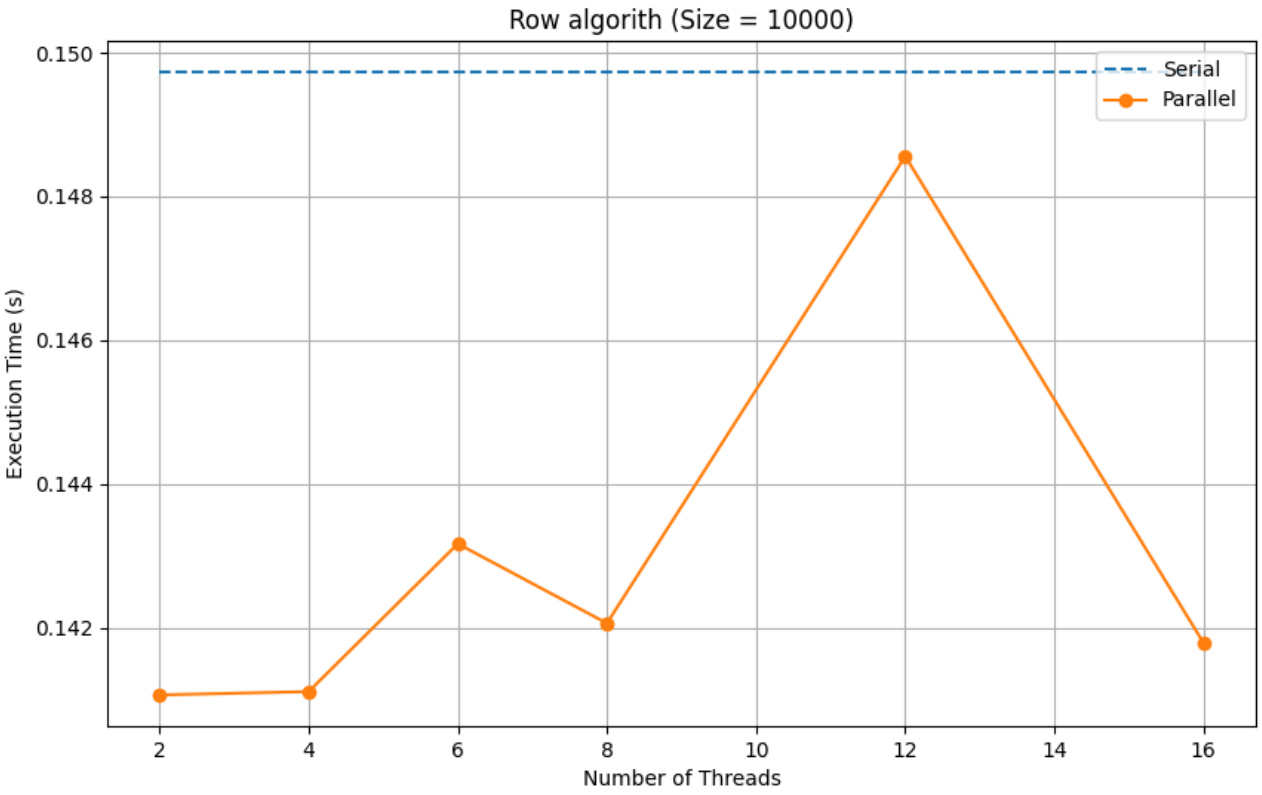

Column algorith (Size = 10000)
Serial
Parallel
Execution Time (s)
0.6
0.5
0.4
0.3
0.2
0.1
2
4
6
8
10
12
Number of Threads

"Κατά γραμμή" Αλγόριθμος

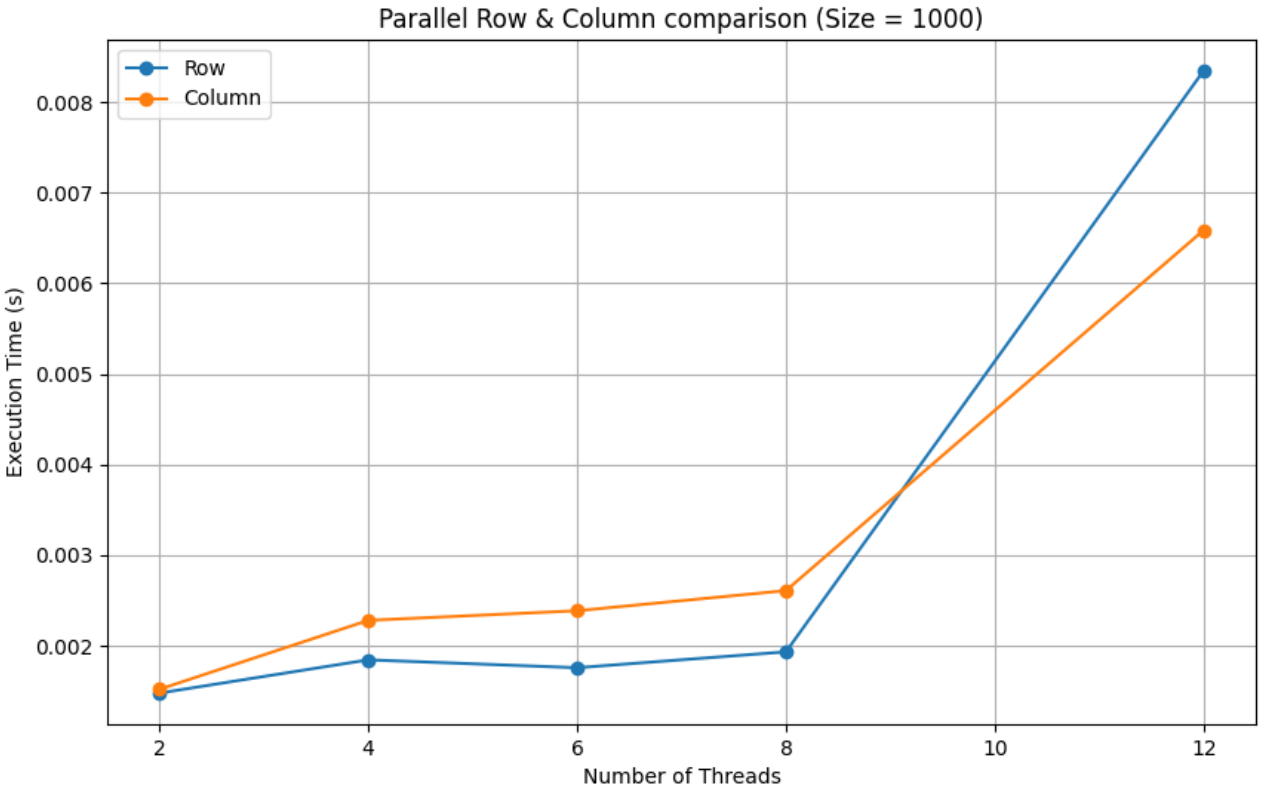


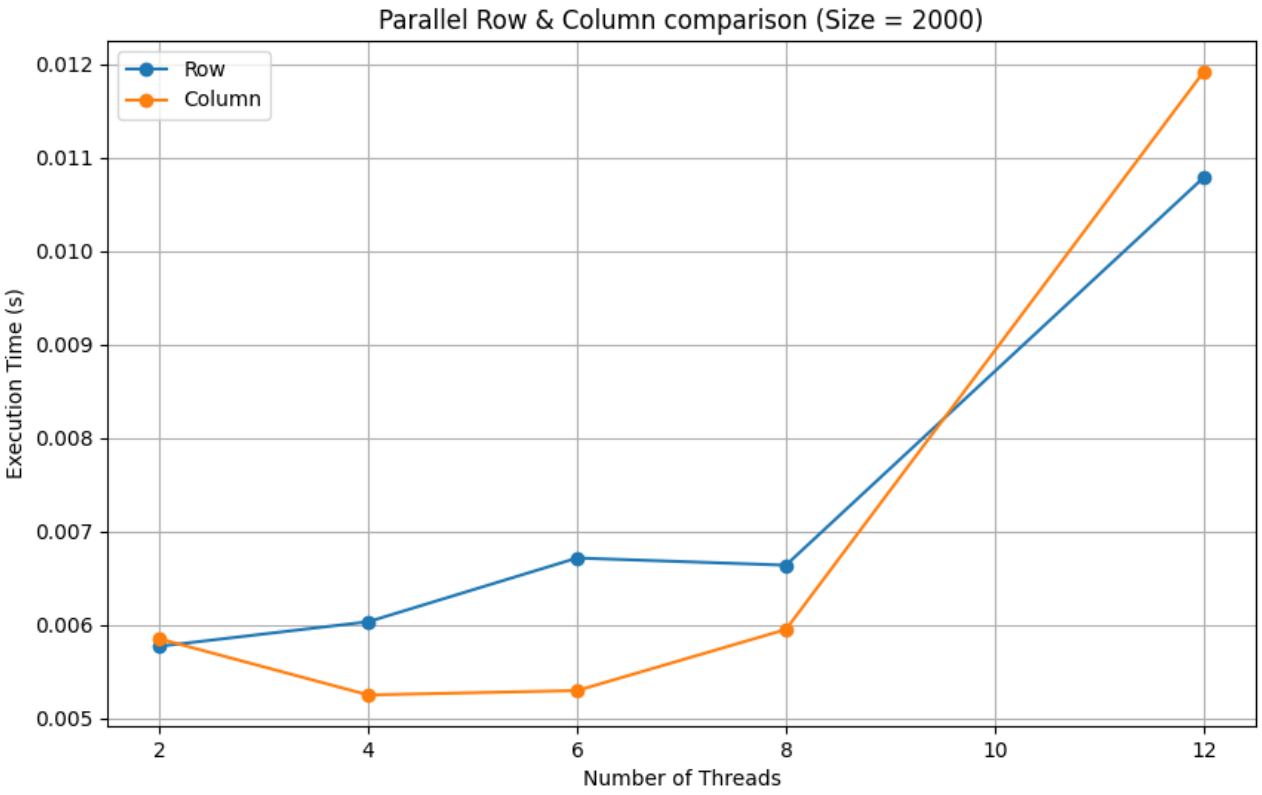

Row algorith (Size = 2000)
Serial
Parallel
Execution Time (s)
0.011
0.010
0.009
0.008
0.007
0.006
2
4
6
8
10
12
14
16
Number of Threads

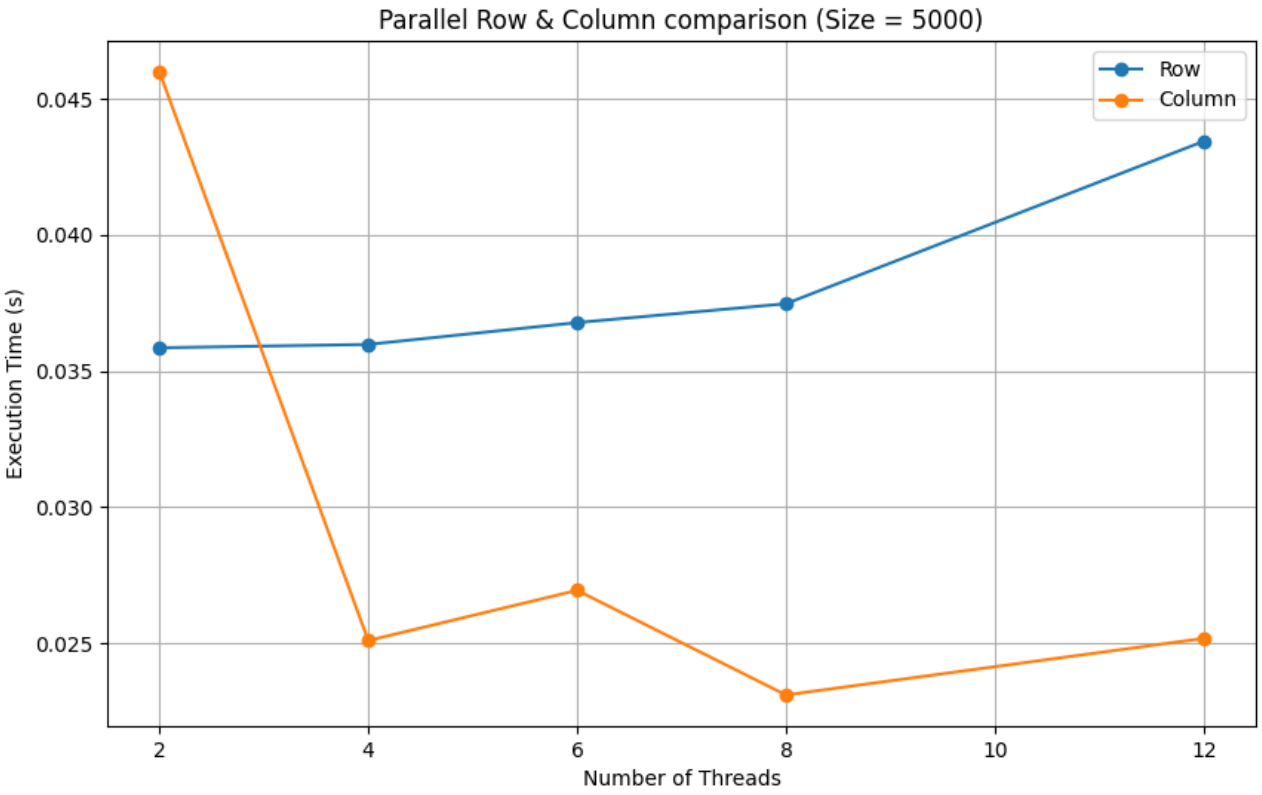

Row algorith (Size = 5000)
Serial
Parallel
Execution Time (s)
0.043
0.042
0.041
0.040
0.039
0.038
0.037
0.036
2
4
6
8
10
12
14
16
Number of Threads

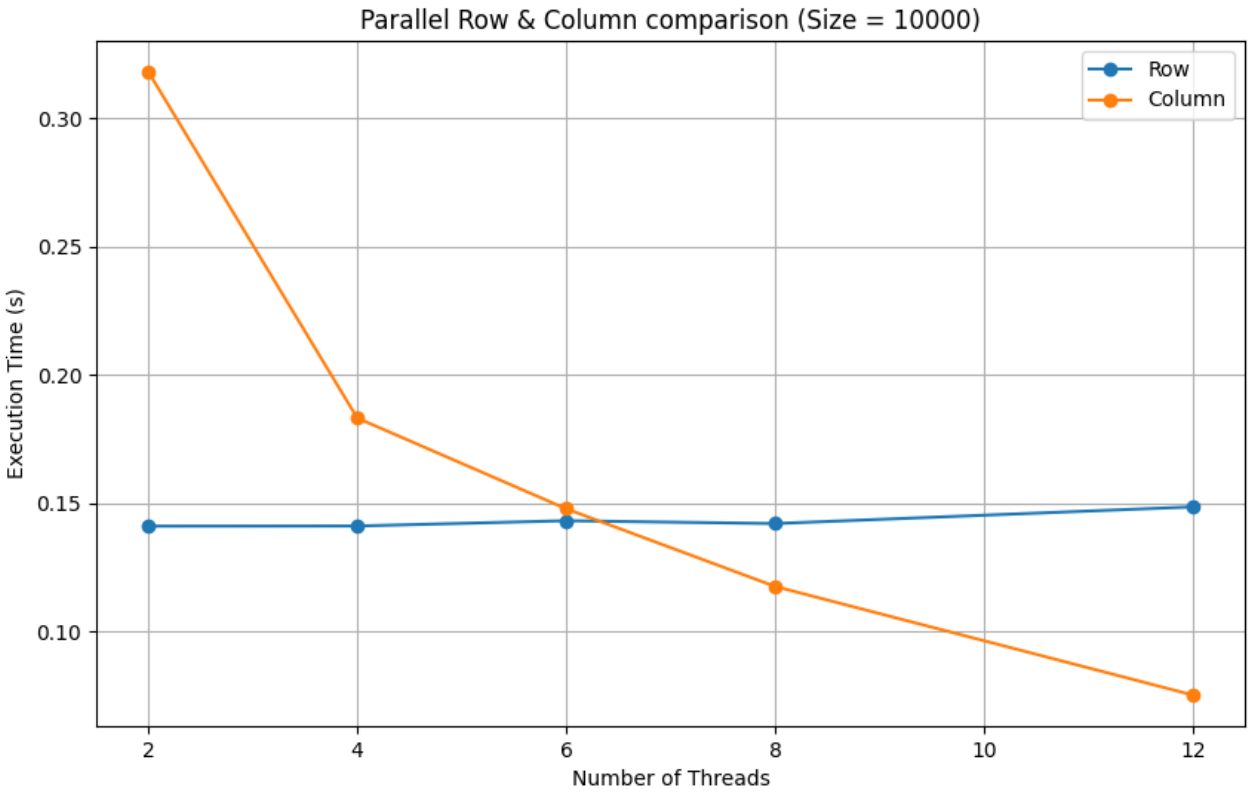

Row algorith (Size = 10000)
Serial
Parallel
Execution Time (s)
0.150
0.148
0.146
0.144
0.142
2
4
6
8
10
12
14
16
Number of Threads

"Κατά γραμμή" εναντίον "Κατά στήλη"




Parallel Row & Column comparison (Size = 2000)
Row
Column
Execution Time (s)
0.012
0.011
0.010
0.009
0.008
0.007
0.006
0.005
2
4
6
8
10
12
Number of Threads


Parallel Row & Column comparison (Size = 5000)
Row
Column
Execution Time (s)
0.045
0.040
0.035
0.030
0.025
2
4
6
8
10
12
Number of Threads


Parallel Row & Column comparison (Size = 10000)
Row
Column
Execution Time (s)
0.30
0.25
0.20
0.15
0.10
2
4
6
8
10
12
Number of Threads

Αντικατάσταση "Κατά Γραμμή"

Η αντικατάσταση κατά γραμμή δεν μπορεί να παραλληλοποιηθεί αποδοτικά, διότι για τον υπολογισμό του x_i χρειάζονται τα προηγούμενα $x_{i+1}, x_{i+2}, \ldots, x_n$. Επομένως, εάν θέλαμε να το προσεγγίσουμε με παραλληλισμό, θα χρειαζόμασταν να υπολογίζουμε τα x_i με αμοιβαίο αποκλεισμό, πράγμα που προκαλεί overhead και οδηγεί σε χειρότερη επίδοση.

Αντικατάσταση "Κατά Στήλη"

Η αντικατάσταση κατά στήλη μπορεί να παραλληλοποιηθεί αποδοτικά. Για κάθε στήλη:

- Αρχικά, το x_i της στήλης αυτής υπολογίζεται εύκολα ως $b_i/A[i,i]$ (η πράξη αυτή γίνεται από ένα thread για να αποφευχθεί το race condition).
- Τα υπόλοιπα $n-1$ x που βρίσκονται στις υπόλοιπες γραμμές μπορούν να ανανεωθούν παράλληλα με βάση το νέο x_i.

Ο όρος `#pragma omp parallel` περικλείει όλη τη συνάρτηση, ώστε τα threads να δημιουργούνται μόνο μία φορά.

Σχολιασμός Αποτελεσμάτων

Η αντικατάσταση κατά γραμμή είναι σχετικά πιο γρήγορη από αυτήν κατά στήλη, επειδή στην cache αποθηκεύονται τα διπλανά στοιχεία στην ίδια σειρά.

Για την αντικατάσταση κατά στήλη, όταν το μέγεθος είναι μικρό, η παράλληλη έκδοση εμφανίζει καλύτερη επίδοση για λιγότερα threads. Για πολλά threads (μέχρι όσα υποστηρίζει η cpu), ο σειριακός είναι πιο αποδοτικός. Όσο μεγαλώνει το μέγεθος, βελτιώνεται όλο και περισσότερος ο χρόνος εκτέλεσης του παράλληλου σε σύγκριση με του σειριακού. Στην περίπτωση αυτή τα πολλά threads βελτιώνουν σημαντικά τον χρόνο.

Για την αντικατάσταση κατά γραμμή, ο παράλληλος αλγόριθμος εμγανίζει ελάχιστη βελτίωση για μεγάλο μέγεθος, με βέλτιστο για λίγα threads. Αυτό ισχύει γιατί κάθε loop εκτελείται από ένα νήμα μόνο (λόγω της εξάρτησης μεταξύ των x_i), με λιγότερα threads θα υπάρχει λιγότερο overhead.

Συγκρίνοντας τον παράλληλο αλγόριθμο κατά γραμμή και στήλη, παρατηρούμε ότι για αρκετά μικρά μεγέθη ο κατά γραμμή είναι ελάχιστα πιο γρήγορος. Όσο αυξάνεται το μέγεθος, η απόδοση του κατά γραμμή αρχίζει να σταθεροποιήτε μεταξύ νημάτων και ο κατά στήλη αρχίζει να εμφανίζει καλύτερες αποδόσεις για περισσότερα νήματα.

Σύγκριση με Σειριακό Αλγόριθμο

Η αντικατάσταση κατά γραμμή είναι σχετικά πιο γρήγορη από αυτήν κατά στήλη, επειδή στην cache αποθηκεύονται τα διπλανά στοιχεία στην ίδια σειρά.

Επίδοση Παράλληλου Αλγορίθμου

Για μικρό μέγεθος του πίνακα, ο παράλληλος αλγόριθμος είναι πιο αποδοτικός με λιγότερα threads. Για μεγάλο μέγεθος πίνακα, η επίδοση του παράλληλου αλγορίθμου μειώνεται.

Βέλτιστο Schedule Clause

Η παραλληλοποίηση του αλγορίθμου αντικατάστασης κατά στήλη είναι πιο αποδοτική όταν δεν ορίζεται κάποιο `schedule` clause, καθώς το χρονοδιάγραμμα των νημάτων είναι καλύτερα κατανεμημένο χωρίς εξωτερικούς περιορισμούς.