Παράλληλη Επεξεργασία.

Φλώρος Παναγιώτης	1047137
Κίκιλας Θεόδωρος	1047118
Ζαφειρόπουλος Ανδρέας	1030372

CEID
University of Patras

2017-2018

Σημειώσεις

Στο πλαίσιο τις εργασίας έχουν γίνει μετρήσεις με παραμέτρους:
–n 1000 –r 350
–n 2000 –r 700
–n 3000 –r 1000
–n 4000 –r 1300
–n 5000 –r 1600
Όμως σε αυτή την αναφορά θα χρησιμοποιηθούν οι μετρήσεις των –n 5000 –r 1600 καθώς το να υλοποιηθούν πίνακες και γραφήματα για όλε2 τις παραμέτρους θα έκανε την αναφορά πολύ μεγαλύτερη και κουραστική στην ανάγνωση. Οι αναλυτικές μετρήσεις για όλες τις παραμέτρους βρίσκεται στο αρχείο Measurements.txt το οποίο δημιουργήθηκε από το bash shell script measure.sh. Για την δημιουργία εκτελέσιμων υπάρχει Makefile. Τα αποτελέσματα από το 3ο ερώτημα πρέπει να μετατραπούν στην αρχική αναπαράσταση, περισσότερες πληροφορίες παρακάτω στο Ερώτημα 3.

Ερώτημα 1

Μέρος Α

Στο ακολουθιακό πρόγραμμα στο τέλος κάθε iteration τα δυναμικά τα οποία μόλις υπολογίστηκαν μεταφέρονται ένα προς ένα στον πίνακα u, πράγμα το οποίο απαιτεί χρόνο O(n). Το να βελτιωθεί ο χρόνος σε αυτή τη περίπτωση είναι αρκετά εύκολο αφού η C δίνει την δυνατότητα διαχείρισης δεικτών. Έτσι αλλάζοντας τις διευθύνσεις στις οποίες δείχνουν οι u και uplus μεταξύ τους έχουμε ώς αποτέλεσμα μείωση σε χρόνο O(1). Μπορεί επίσης να μετακινηθεί ο υπολογισμός των omega στο επάνω loop.

Μετρήσεις

n=5000 r=1600	**Ακολουθιακό**	**1a**
Time for calculations	16223.751310	16256.407890
Time for I/O	0.127647	0.159393
Total execution time	16223.878957	16256.567283

Παρατηρείτε ότι ο χρόνος δεν μειώθηκε μάλιστα αυξήθηκε κατά λίγο.

Μέρος Β

Όσον αφορά την αναδιοργάνωση πράξεων υπάρχουν τρία μέρη τα οποία επιδέχονται βελτίωση:

$$uplus[i] = u[i] + dt * (mu - u[i]) \ (1)$$

$$sum+ = sigma[i * n + j] * (u[j] - u[i]) \ (2)$$

$$uplus[i]+ = dt * sum/divide \ (3)$$

Αρχικά

$$(1) => uplus[i] = u[i] + dt * mu - dt * u[i]$$

$$=> uplus[i] = u[i] * (1 - dt) + dt * mu$$

μπορούμε να αποφύγουμε τις επαναλήψεις υπολογισμών των

$$(1 - dt)$$

και

$$dt * mu$$

αφού είναι σταθεροί. Έχοντας

$$uplus[i] = u[i] * oneminusdt + dtmu$$

Η (2) βελτιώνεται περισσότερο από τις υπόλοιπές καθώς ο πίνακας sigma είναι σταθερός. Δίνεται η δυνατότητα να χρησιμοποιηθεί ένας δείκτης, έστω sigmain, ο οποίος κατευθύνεται κατά γραμμές όσο αλλάζει το j. Έτσι μειώνονται κατά λίγο οι πράξεις αφού δεν χρειάζεται να υπολογίζεται το

$$i * n + j$$

κάθε φορά που αλλάζει το j.

$$(2) => sum+ = sigmain[j] * u[j] - sigmain[j] * u[i]$$

Το sigmain αποτελεί ουσιαστικά διάνυσμα γραμμή του sigma ανάλογα το i. Άρα το

$$sigmain[j] * u[j]$$

αποτελεί εσωτερικό γινόμενο (θα χρειαστεί στο επόμενο μέρος). Επίσης, το u[i] για το loop του j αποτελεί σταθερά και το

$$for(j = 0; j < n; j++)\{sigmain[j] * u[i]\}$$

είναι ουσιαστικά

$$\sum_{j=0}^{n}(sigmain[j] * u[i])$$

$$=> u[i] * \sum_{j=0}^{n} sigmain[j]$$

Άρα πριν την εκτέλεση των iterations μπορούμε να υπολογίσουμε ένα διάνυσμα έστω sigmasum μεγέθους n που κάθε στοιχείο του είναι το

$$\sum_{j=0}^{n} sigmain[j]$$

για κάθε γραμμή i και να υπολογίζεται

$$sigmasum[i] * u[i]$$

εκτός του loop του j. Τέλος, στην (3)

$$uplus[i]+ = dt * sum/divide$$

για να μην υπολογίζεται στο loop του i το dt/divide που αποτελεί σταθερός όρος μπορεί να πολλαπλασιαστεί με το sigma και να αποθηκευτεί σε αυτό πριν αρχίσουν να τρέχουν τα iterations.

Μετρήσεις

n=5000 r=1600	**Ακολουθιακό**	**1b**
Time for calculations	16223.751310	14689.618102
Time for I/O	0.127647	0.160124
Total execution time	16223.878957	14689.7782263

Παρατηρείτε ότι ο χρόνος υπολογισμών μειώθηκε αρκετά κατά 9,456

Μέρος C

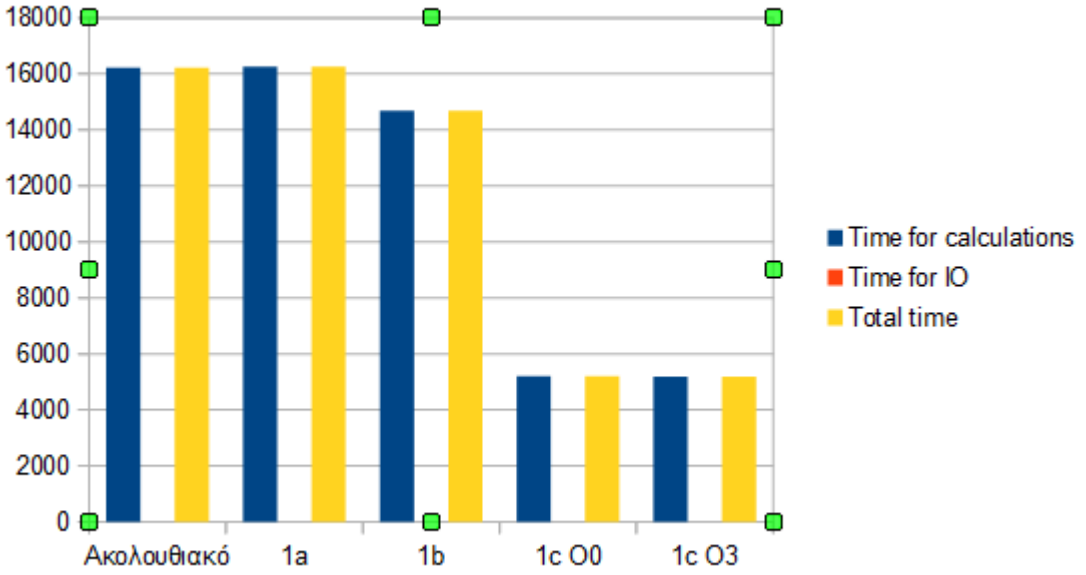
Για αυτήν την υλοποίηση χρησιμοποιήθηκε η cblas σε τρία σημεία στον κώδικα:
1- Πολλαπλασιασμός $sigma[i] * dt/divide$ με την dscal αφού είναι γινόμενο βαθμωτού διπλής ακρίβειας κινητής υποδιαστολής με διάνυσμα μεγέθους n^2.
2-Υπολογισμός του sigmasum με την dasum αφού το sigmasum κρατάει το άθροισμα των στοιχείων των γραμμών του sigma και οι τιμές που παίρνουν τα στοιχεία είναι θετικές εξ΄ ορισμού (dasum κάνει απόλυτο άθροισμα στοιχείων διανύσματος διπλής ακρίβειας).
3-Εσωτερικό γινόμενο $sigmain \cdot u$ με την ddot.

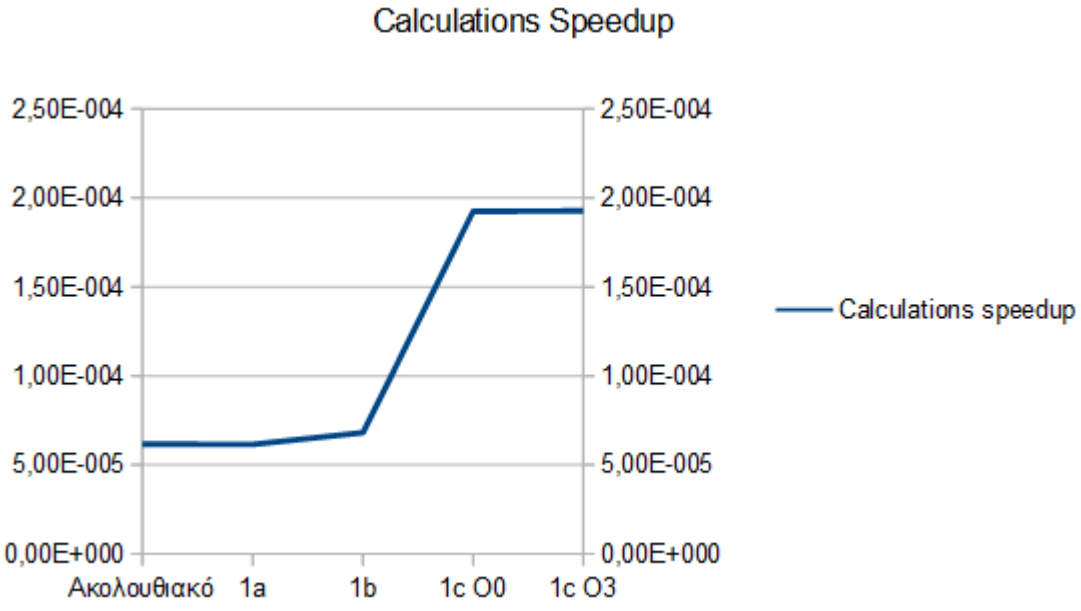
Μετρήσεις

n=5000 r=1600 O0	**Ακολουθιακό**	**1c O0**
Time for calculations	16223.751310	5197.755955
Time for I/O	0.127647	0.163097
Total execution time	16223.878957	5197.919052

n=5000 r=1600 O3	**Ακολουθιακό**	**1c O3**
Time for calculations	16223.751310	5188.279849
Time for I/O	0.127647	0.157700
Total execution time	16223.878957	5188.437549

Παρατηρείτε ότι και με τα δύο optimizations ο χρόνος υπολογισμών είναι κατά πολύ μικρότερος, 67.962% και 68.02% αντίστοιχα, ενώ οι διαφορά μεταξύ των optimizations είναι αρκετά μικρή πράγμα που πιθανότατα αποδίδεται στην υλοποίηση του προηγούμενου μέρους.


18000
16000
14000
12000
10000
8000
6000
4000
2000
0
Ακολουθιακό
1a
1b
1c O0
1c O3
Time for calculations
Time for IO
Total time


Calculations Speedup
2,50E-004
2,00E-004
1,50E-004
1,00E-004
5,00E-005
0,00E+000
Ακολουθιακό
1a
1b
1c O0
1c O3
Calculations speedup

Ερώτημα 2

Μέρος Α

Loop it

Σε αυτό το κομμάτι του ερωτήματος η παραλληλοποίηση δεν είναι δυνατή αφού υπάρχει εξάρτηση δεδομένων π.χ. για να υπολογίσει το iteration 5 το uplus χρειάζεται το u το οποίο είναι το uplus που υπολογίστηκε στο iteration 4 και ούτω καθεξής. Στον κώδικα χρησιμοποιήθηκαν #pragma omp for ordered στο loop του it με ordered structure block το loop του i το οποίο είναι ουσιαστικά ένα ακολουθιακό πρόγραμμα.

Μετρήσεις

n=5000 r=1600 O0 1 Thread	**1b**	**2a it O0 1 Thread**
Time for calculations	14689.618102	14730.431649
Time for I/O	0.160124	0.147156
Total execution time	14689.7782263	14730.578805

n=5000 r=1600 O0 2 Threads	**1b**	**2a it O0 2 Threads**
Time for calculations	14689.618102	14734.176927
Time for I/O	0.160124	0.170111
Total execution time	14689.7782263	14734.347038

n=5000 r=1600 O0 4 Threads	**1b**	**2a it O0 4 Threads**
Time for calculations	14689.618102	14730.592128
Time for I/O	0.160124	0.159160
Total execution time	14689.7782263	14730.751288

Παρατηρείτε ότι ο χρόνος έγινε χειρότερος και οτι το πλήθος Threads δεν βελτιώνει αλλά

αυξάνει το χρόνο πράγμα το οποίο είναι λογικό αφού το πρόγραμμα είναι ακολουθιακό όπως και το 1b στο οποίο είναι βασισμένο αλλά πιο αργό γιατί δημιουργεί και τα Threads.

n=5000 r=1600 O3 1 Thread	**1b**	**2a it O3 1 Thread**
Time for calculations	14689.618102	5142.525150
Time for I/O	0.160124	0.164838
Total execution time	14689.7782263	5142.689988

n=5000 r=1600 O3 2 Threads	**1b**	**2a it O3 2 Threads**
Time for calculations	14689.618102	5141.655595
Time for I/O	0.160124	0.148907
Total execution time	14689.7782263	5141.804502

n=5000 r=1600 O3 4 Threads	**1b**	**2a it O3 4 Threads**
Time for calculations	14689.618102	5141.327537
Time for I/O	0.160124	0.170540
Total execution time	14689.7782263	5141.498077

Όσον αφορά το πλήθος των Threads τα αποτελέσματα είναι ίδια με πριν απλά το optimization O3 έχει βελτιώσει κατά πολύ το ακολουθιακό πρόγραμμα.

Loop i

Το κομμάτι αυτό μπορεί να παραλληλοποιηθεί, συγκεκριμένα στο κώδικα προστέθηκε η γραμμή #pragma omp parallel for default(shared) private(j,sum) για να παραλληλοποιηθεί με sum και j private καθώς αν έγραφαν όλα τα Threads σε αυτές θα υπήρχαν προβλήματα π.χ. αν το iteration 4 πρόσθετε κάτι στο sum και αμέσως μετά το iteration 39 χρησιμοποιούσε το sum για να υπολογίσει το uplus του, τα αποτελέσματα θα ήταν λανθασμένα.

Μετρήσεις

n=5000 r=1600 O0 1 Thread	**1b**	**2a i O0 1 Thread**
Time for calculations	14689.618102	14732.864042
Time for I/O	0.160124	0.128840
Total execution time	14689.7782263	14732.992882

Παρατηρείτε ότι με 1 Thread ο χρόνος δεν βελτιώνεται καθώς με 1 Thread το πρόγραμμα είναι ακολουθιακό.

n=5000 r=1600 O0 2 Threads	**1b**	**2a i O0 2 Threads**
Time for calculations	14689.618102	7395.059577
Time for I/O	0.160124	0.131781
Total execution time	14689.7782263	7395.191358

n=5000 r=1600 O0 4 Threads	**1b**	**2a i O0 4 Threads**
Time for calculations	14689.618102	6061.407000
Time for I/O	0.160124	0.145851
Total execution time	14689.7782263	6061.552851

Παρατηρείτε ότι ο χρόνος είναι λιγότερος όσο το πλήθος Threads μεγαλώνει.

n=5000 r=1600 O3 1 Thread	**1b**	**2a i O3 1 Thread**
Time for calculations	14689.618102	5140.896655
Time for I/O	0.160124	0.128886
Total execution time	14689.7782263	5141.025541

n=5000 r=1600 O3 2 Threads	**1b**	**2a i O3 2 Threads**
Time for calculations	14689.618102	2667.623697
Time for I/O	0.160124	0.129549
Total execution time	14689.7782263	2667.753246

n=5000 r=1600 O3 4 Threads	**1b**	**2a i O3 4 Threads**
Time for calculations	14689.618102	2174.475961
Time for I/O	0.160124	0.134036
Total execution time	14689.7782263	2174.609997

Όσον αφορά το πλήθος των Threads τα αποτελέσματα είναι ίδια με πριν απλά το optimization O3 έχει βελτιώσει κατά πολύ το ακολουθιακό πρόγραμμα.

Loop j

Η παραλληλοποίηση αυτού του loop είναι σχετικά εύκολη αφού το μόνο που χρειάζεται να προστεθεί στο κώδικα είναι η γραμμή #pragma omp parallel for reduction(+:sum) default(shared) private(j). Χρησιμοποιούμε το reduction γιατί πρέπει το sum να είναι ιδιωτικό και στο τέλος να έχει ως τιμή το άθροισμα του κάθε sum απο κάθε Thread.

Μετρήσεις

n=5000 r=1600 O0 1 Thread	**1b**	**2a j O0 1 Thread**
Time for calculations	14689.618102	15367.205261
Time for I/O	0.160124	0.172593
Total execution time	14689.7782263	15367.377854

Παρατηρείτε ότι με 1 Thread ο χρόνος δεν βελτιώνεται καθώς με 1 Thread το πρόγραμμα είναι ακολουθιακό μάλιστα είναι λίγο χειρότερος.

n=5000 r=1600 O0 2 Threads	**1b**	**2a j O0 2 Threads**
Time for calculations	14689.618102	8334.892684
Time for I/O	0.160124	0.158220
Total execution time	14689.7782263	8335.050904

n=5000 r=1600 O0 4 Threads	**1b**	**2a j O0 4 Threads**
Time for calculations	14689.618102	6531.386999
Time for I/O	0.160124	0.133117
Total execution time	14689.7782263	6531.520116

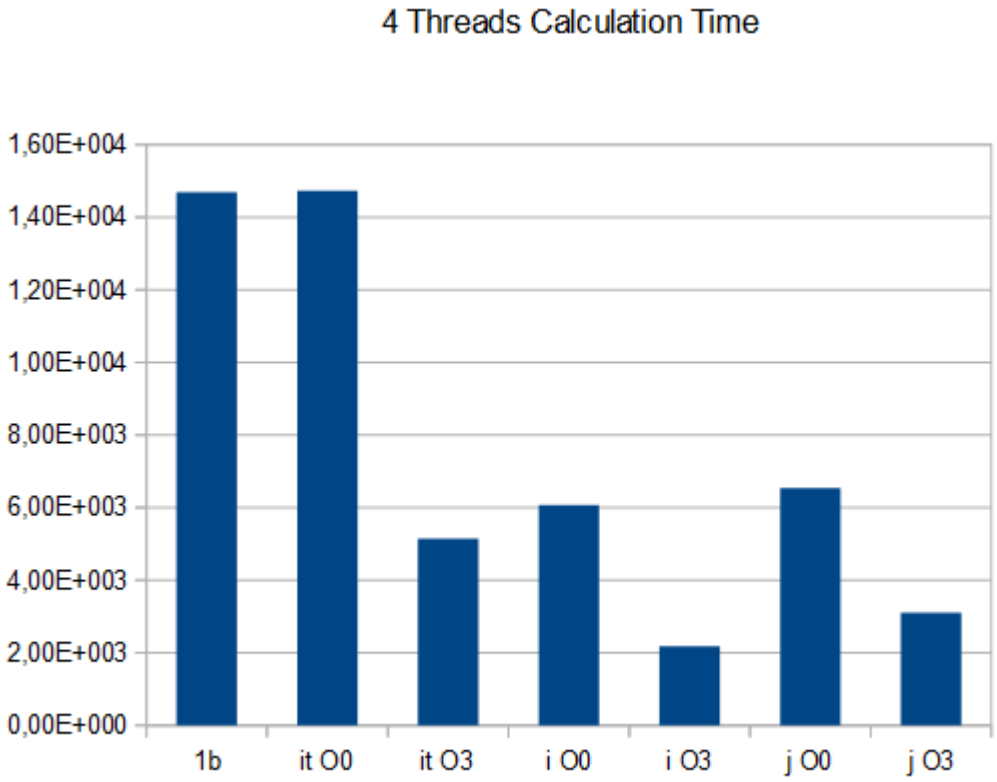
Παρατηρείτε ότι ο χρόνος είναι λιγότερος όσο το πλήθος των Threads μεγαλώνει.

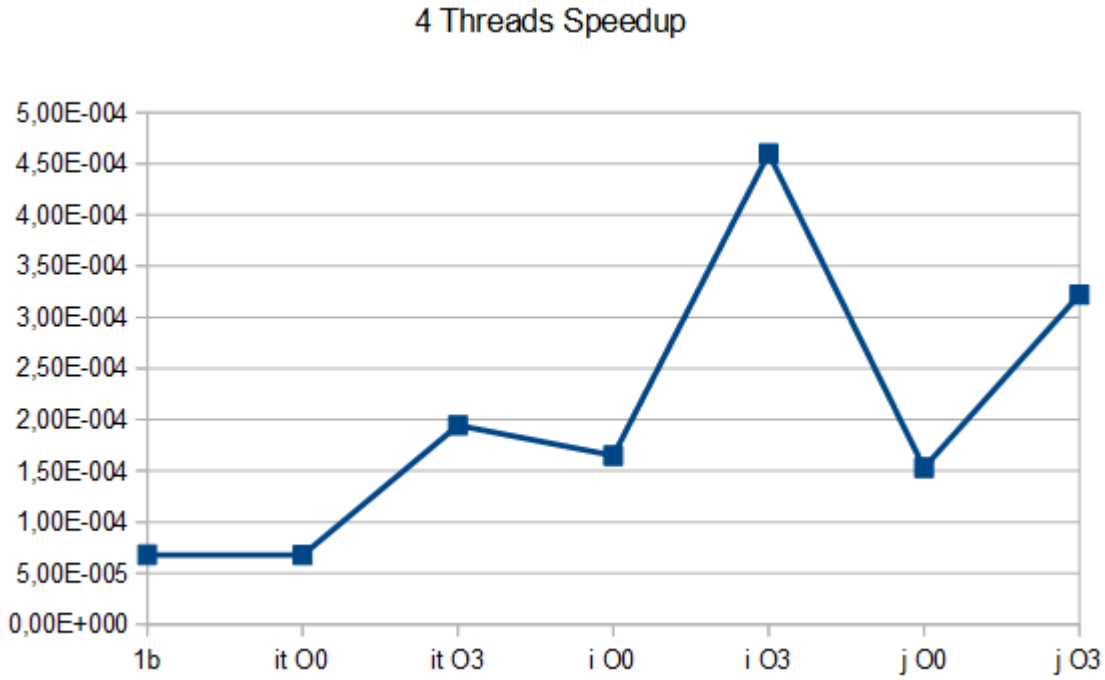
n=5000 r=1600 O3 1 Thread	**1b**	**2a j O3 1 Thread**
Time for calculations	14689.618102	5720.457899
Time for I/O	0.160124	0.161886
Total execution time	14689.7782263	5720.619785

n=5000 r=1600 O3 2 Threads	**1b**	**2a j O3 2 Threads**
Time for calculations	14689.618102	3602.732337
Time for I/O	0.160124	0.133094
Total execution time	14689.7782263	3602.865431

n=5000 r=1600 O3 4 Threads	**1b**	**2a j O3 4 Threads**
Time for calculations	14689.618102	3104.891877
Time for I/O	0.160124	0.133519
Total execution time	14689.7782263	3105.025396

Όσον αφορά το πλήθος των Threads τα αποτελέσματα είναι ίδια με πριν απλά το optimization O3 έχει βελτιώσει κατά πολύ το ακολουθιακό πρόγραμμα.


4 Threads Calculation Time
1,60E+004
1,40E+004
1,20E+004
1,00E+004
8,00E+003
6,00E+003
4,00E+003
2,00E+003
0,00E+000
1b
it O0
it O3
i O0
i O3
j O0
j O3


4 Threads Speedup
5,00E-004
4,50E-004
4,00E-004
3,50E-004
3,00E-004
2,50E-004
2,00E-004
1,50E-004
1,00E-004
5,00E-005
0,00E+000
1b
it O0
it O3
i O0
i O3
j O0
j O3

Μέρος Β

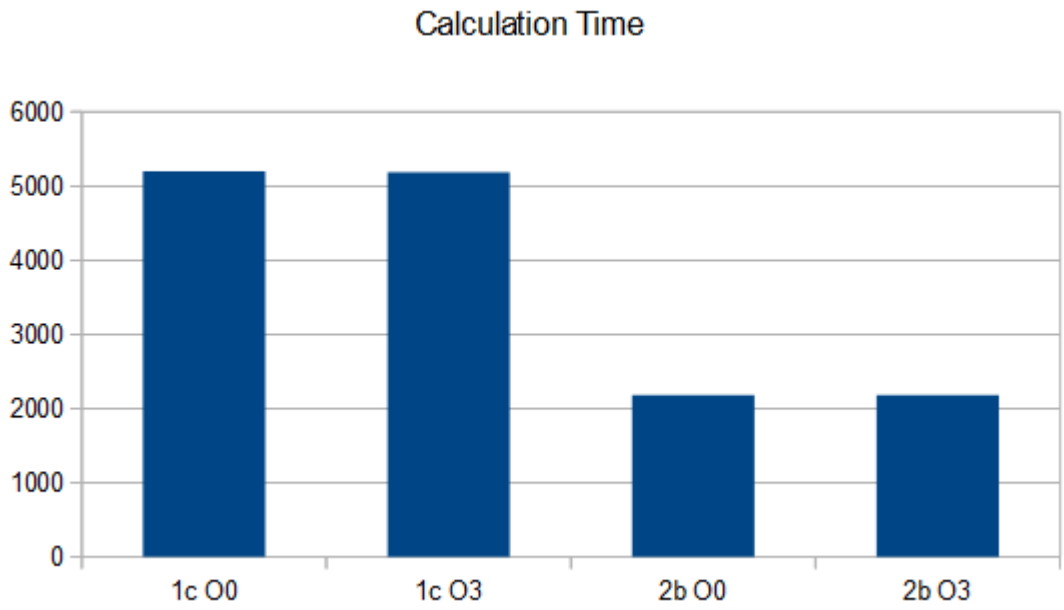
Σε αυτό το μέρος έγιναν οι ίδιες αλλαγές όσον αφορά την cblas στο ερώτημα 1 μέρος c και προστέθηκαν #pragma omp parallel for στους αρχικούς υπολογισμούς των sigma και sigmasum.

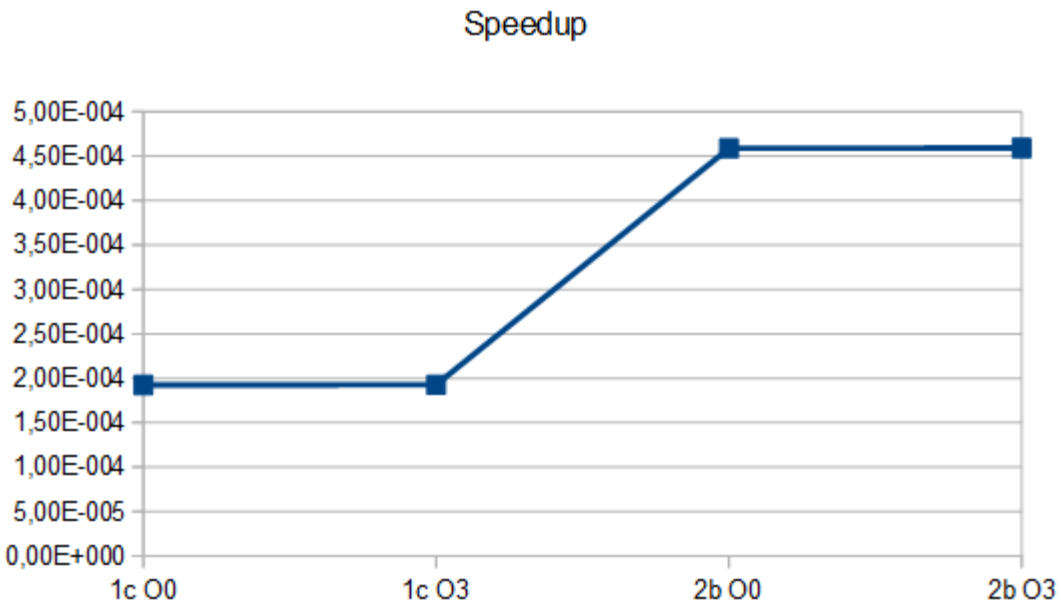
Μετρήσεις

n=5000 r=1600 O0	**1c O0**	**2b O0**
Time for calculations	5197.755955	2179.646815
Time for I/O	0.163097	0.130501
Total execution time	5197.919052	2179.777316

n=5000 r=1600 O3	**1c O3**	**2b O3**
Time for calculations	5188.279849	2177.946638
Time for I/O	0.157700	0.130503
Total execution time	5188.437549	2178.077141

Οι μετρήσεις για τα 2b έγιναν με 4 threads.



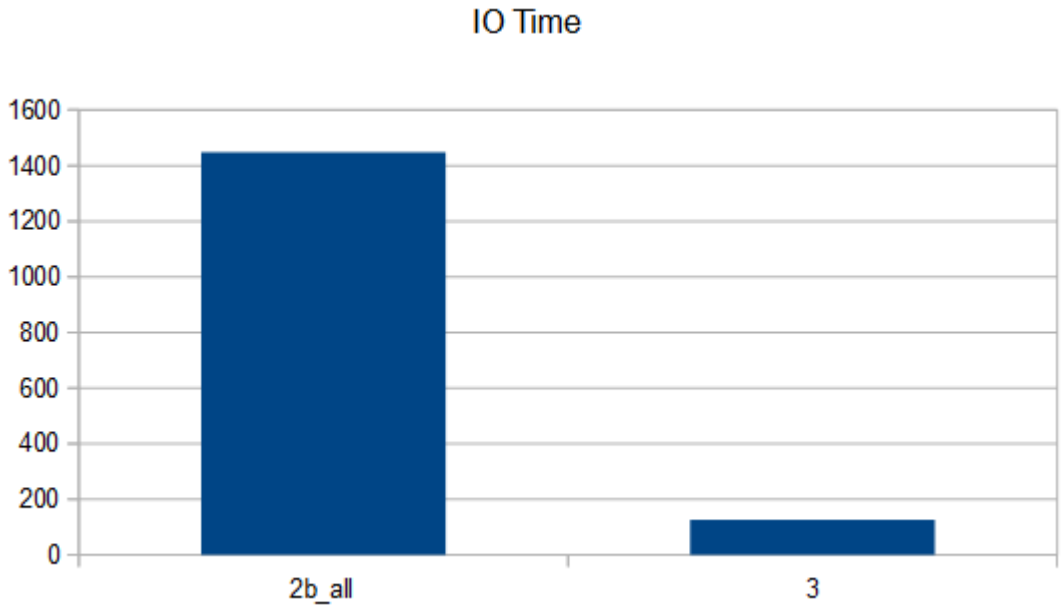

Speedup
5,00E-004
4,50E-004
4,00E-004
3,50E-004
3,00E-004
2,50E-004
2,00E-004
1,50E-004
1,00E-004
5,00E-005
0,00E+000
1c O0
1c O3
2b O0
2b O3

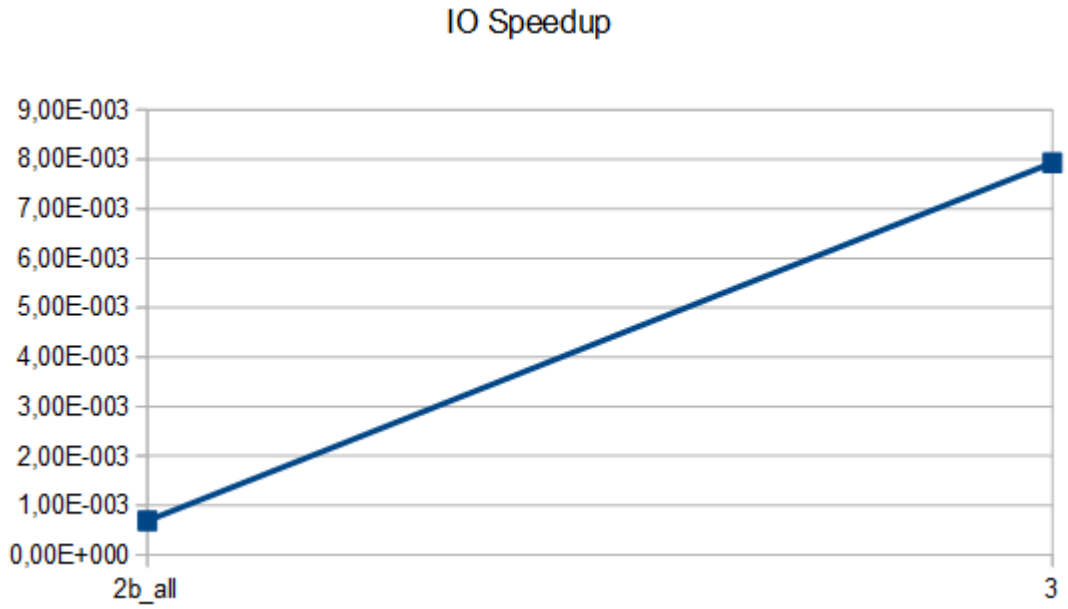
Ερώτημα 3

Στο τελευταίο ερώτημα χρησιμοποιήθηκαν δύο buffers για να αποθηκεύουν προσωρινά τα u και omega ώστε να μειωθεί το πλήθος τον system calls στο λειτουργικό σύστημα για εγγραφή στο δίσκο. Συγκεκριμένα επιλέχθηκε το πρόγραμμα να γράφει στο δίσκο όσες φορές έγραφε και χωρίς το flag -DALL_RESULTS. Επίσης, μετατράπηκε η αναπαράσταση των αποτελεσμάτων σε δυαδική μορφή. Με αυτό το τρόπο μειώνεται κατά πολύ και ο χρόνος εγγραφής στο δίσκο αλλά και ο χώρος αποθήκευσης στο δίσκο. Για να μετατραπούν τα αρχεία εξόδου στην αρχική του μορφή υπάρχει ο κώδικας converter.c. Αυτό το πρόγραμμα δέχεται τρεις παραμέτρους αρχείο εισόδου, αρχείο εξόδου και την τιμή του n που χρησιμοποιήθηκε αντίστοιχα. (Προσοχή! Οι παράμετροι πρέπει να δοθούν με αυτή τη σειρά). Για τις χρονικές συγκρίσεις δημιουργήθηκε ένα εκτελέσιμο του ερωτήματος 2b που έγινε compile με το flag -DALL_RESULTS.

Μετρήσεις

n=5000 r=1600 O0	**2b_all**	**3**
Time for calculations	2210.524448	2189.588569
Time for I/O	1448.254361	126.081989
Total execution time	3658.778809	2315.670558


IO Time
1600
1400
1200
1000
800
600
400
200
0
2b_all
3


IO Speedup
9,00E-003
8,00E-003
7,00E-003
6,00E-003
5,00E-003
4,00E-003
3,00E-003
2,00E-003
1,00E-003
0,00E+000
2b_all
3