ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2018/2019

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ

Η ομάδα της εργασίας:

Όνομα	***Α.Μ.***	***Έτος***
Παύλος Βραχνής	***236010***	***5ο***
Παρασκευόπουλος Βασίλης	***236179***	***5ο***
Γεωργόπουλος Ζαχαρίας	***235735***	***5ο***

ΕΡΩΤΗΜΑ 1 - STATIC:

Οι παρακάτω χρόνοι εκτέλεσης, οι οποίοι παρουσιάζονται στους παρακάτω πίνακες, πραγματοποιήθηκαν με βάση το static scheduling. Δήλωση για το chunk δεν έχει γίνει, καθώς θεωρήσαμε ότι είναι καλύτερο να γίνει “αυτόματα” από το πρότυπο OpenMP, έτσι οι επαναλήψεις χωρίζονται (και δίνονται στα threads) σε κομμάτια που είναι κατά προσέγγιση ίσα σε μέγεθος.
Επίσης, παρατηρούμε ότι ο χρόνος εκτέλεσης των νημάτων σχεδόν ταυτίζεται με το Συνολικό Χρόνο Εκτέλεσης του Προγράμματος μας. Αυτό το φαινόμενο προκαλείται για δύο λόγους, είτε γιατί το μεγαλύτερο μέρος του κώδικα εκτελείται σειριακά είτε γιατί υπάρχει πολύ μεγάλος συγχρονισμός με αποτέλεσμα να μην αντικατοπτρίζεται στην πραγματική εργασία που κάνουν τα νήματα.
Συνολικά, παρατηρούμε ότι ο σειριακός χρόνος εκτέλεσης είναι οριακά μικρότερος απ’ότι ο χρόνος εκτέλεσης με στατικό τρόπο ανάθεσης, το οποίο είναι λογικό αφού η ανάθεση των πολυγραμμών στα νήματα γίνεται στατικα με αποτέλεσμα τα νήματα που τελείωσαν με το κομμάτι τους να περιμένουν σε ένα barrier να τελειώσουν και τα υπόλοιπα νήματα, λόγω του Round-Robin τρόπου ανάθεσης.

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)
0.1	20.138363	20.138140
0.01	37.209991	37.209672
0.001	55.959234	55.958837
0.0001	85.020187	85.019941

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)
0.1	9.973071	9.972835
0.01	16.859193	16.858860
0.001	24.654681	24.654414

0.0001	31.901642	31.9012276

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)
0.1	20.350756	20.350303	20.350231
0.01	35.896719	35.896258	35.895011
0.001	56.688298	56.687980	56.687903
0.0001	84.227017	84.226576	84.225839

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)
0.1	9.378695	9.378304	9.378291
0.01	16.720769	16.719877	16.718231
0.001	24.212023	24.211493	24.211455
0.0001	31.534489	31.534141	31.532478

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)	RunTime Thread 2 (sec)	RunTime Thread 3 (sec)
0.1	20.549319	20.549327	20.549323	20.549319	20.549229
0.01	37.580193	37.575931	37.575926	37.57968	37.575881
0.001	56.112467	56.111916	56.108216	56.107171	56.107128
0.0001	79.979883	79.979487	79.972238	79.972093	79.971988

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)	RunTime Thread 2 (sec)	RunTime Thread 3 (sec)
0.1	8.545044	8.544469	8.540523	8.540498	8.540356
0.01	15.287701	15.287172	15.286447	15.286372	15.286261
0.001	23.174236	23.169013	23.169008	23.168998	23.169005
0.0001	29.169005	28.907960	28.907244	28.907160	28.907047

ΕΡΩΤΗΜΑ 2 - Dynamic:

Οι παρακάτω χρόνοι εκτέλεσης, οι οποίοι παρουσιάζονται στους παρακάτω πίνακες, πραγματοποιήθηκαν με βάση το dynamic scheduling. Δήλωση για το chunk δεν έχει γίνει, καθώς θεωρήσαμε ότι είναι καλύτερο να γίνει “αυτόματα” από το πρότυπο OpenMP, έτσι οι επαναλήψεις χωρίζονται (και δίνονται στα threads) σε κομμάτια που είναι κατά προσέγγιση ίσα σε μέγεθος.
Οι παράμετροι των οδηγιών αυτών πρέπει να καθοριστούν πειραματικά ώστε να δώσουν τον καλύτερο δυνατό χρόνο εκτέλεσης της εφαρμογής είναι το μέγεθος του chunk και συγκεκριμένα η εντολή:
#pragma omp for ordered schedule(dynamic, chunk)

Παρατηρούμε ότι οι χρόνοι σε αυτό το ερώτημα σε σύγκριση με το πρώτο ερώτημα, είναι ότι μειώνονται σε μεγάλο βαθμό και ειδικά όσο αυξάνουμε τον αριθμό των νημάτων. Αυτό είναι λογικό, αφού το dynamic scheduling λειτουργεί με βάση την λογική “First come, First served”. Έτσι, μόλις ένα νήμα τελειώσει το κομμάτι του τότε αυτό “λαμβάνει” το επόμενο κομμάτι προς επεξεργασία ανεξάρτητα με το τι κάνουν τα υπόλοιπα νήματα.

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)
0.1	18.806942	18.806701
0.01	34.688690	34.688393
0.001	54.978584	54.978166
0.0001	81.380521	81.380238

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)
0.1	8.491683	8.491458
0.01	15.720780	15.720435
0.001	23.240806	23.240502
0.0001	28.246275	28.245940

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)
0.1	10.226631	10.226137	10.226251
0.01	19.159732	19.159235	19.159356
0.001	28.357081	28.356475	28.356596
0.0001	42.151466	42.150957	42.150792

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)
0.1	4.686209	4.685682	4.685806
0.01	8.736486	8.735854	8.735984
0.001	11.669068	11.668693	11.668416
0.0001	16.126170	16.125597	16.125767

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)	RunTime Thread 2 (sec)	RunTime Thread 3 (sec)
0.1	5.989479	5.988443	5.988327	5.988443	5.988448
0.01	10.627128	10.626459	10.620493	10.620372	10.620489
0.001	20.640186	20.631704	20.639758	20.631852	20.631829
0.0001	29.061244	29.056975	29.057002	29.057051	29.060852

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)	RunTime Thread 0 (sec)	RunTime Thread 1 (sec)	RunTime Thread 2 (sec)	RunTime Thread 3 (sec)
0.1	2.896696	2.890312	2.890180	2.890307	2.890294
0.01	5.816591	5.816122	5.814089	5.814204	5.814183
0.001	6.787567	6.786916	6.786941	6.787070	6.787046
0.0001	9.710493	9.706330	9.706325	9.706390	9.706296

ΕΡΩΤΗΜΑ 3 - Task1:

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	21.508543
0.01	38.813699
0.001	56.686366
0.0001	94.051617

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	8.811769
0.01	15.717283
0.001	22.516891
0.0001	34.127470

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	11.228403
0.01	20.732769
0.001	32.817742
0.0001	52.476818

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	5.061998
0.01	9.164993
0.001	14.699550
0.0001	20.891631

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	6.264545
0.01	11.461919
0.001	20.623065
0.0001	29.155565

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	3.085362
0.01	5.612065
0.001	8.051554
0.0001	11.085990

ΕΡΩΤΗΜΑ 4 - Task2:

Παρατηρούμε ότι οι χρόνοι εκτέλεσης της εφαρμογής με την εκτέλεση αυτή παρουσιάζει χρόνος αυξημένους απ'ότι με την εκτέλεση του Ερωτήματος 3. Αυτό είναι λογικό, αφού κάποιο από τα νήματα που ανέλαβε να εκτελέσει ένα από τα δύο έργα, μπορεί για κάποιο λόγο να αναστείλει την λειτουργία του, με αποτέλεσμα το έργο να περιμένει και πάλι το συγκεκριμένο νήμα να συνεχίσει την εκτέλεση του. Αυτό μπορεί να αποφευχθεί εφόσον ορίσουμε το clause "untied", έτσι ώστε οποιαδήποτε από τα άλλα νήματα να μπορεί να συνεχίσει την εκτέλεση του συγκεκριμένου έργου. Στο Ερώτημα 3 έχουμε μόνο ένα έργο άρα είναι λογικό οι χρόνοι στο Ερώτημα 4 να είναι αυξημένοι, αφού το φαινόμενο αυτό μπορεί να συμβεί σε δύο έργα τώρα.

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	22.941936
0.01	38.567942
0.001	64.523179
0.0001	99.823249

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 1:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	8.956860
0.01	16.719685
0.001	27.723130
0.0001	33.384373

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	12.004602
0.01	22.007117
0.001	34.825176
0.0001	56.073206

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 2:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	5.337035
0.01	8.976220
0.001	15.160796
0.0001	22.788563

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	7.101936
0.01	13.649421
0.001	20.007988
0.0001	36.916767

Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και αριθμό Threads ίσο με 4:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	2.880059
0.01	5.226579
0.001	10.205099
0.0001	16.912998

Διαγράμματα Χρονοβελτίωσης:

Η χρονοβελτίωση (speedup) αποτελεί βασικό μέτρο σύγκρισης της απόδοσης και ορίζεται ως εξής: **speedup = sequential time / parallel time**

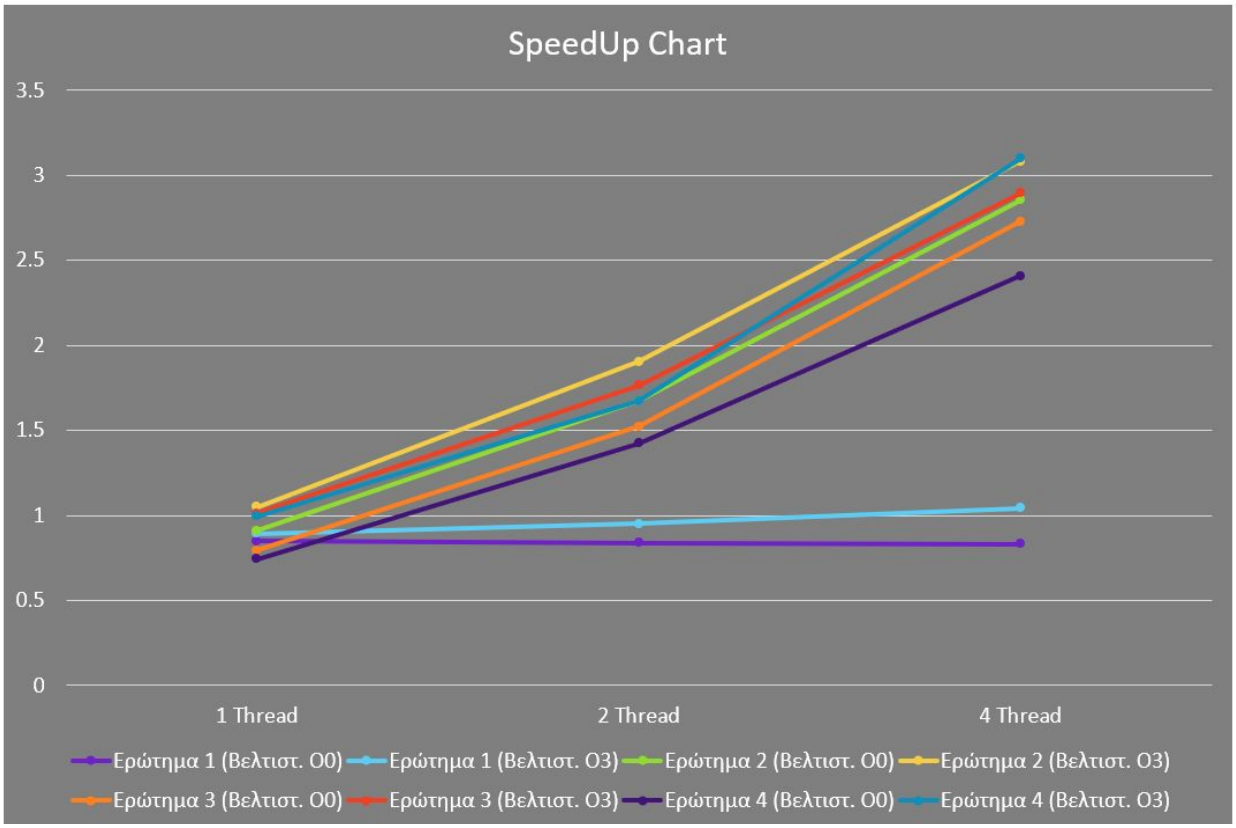
Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O0 και σειριακή εκτέλεση του προγράμματος:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	17.093721
0.01	33.668720
0.001	51.969106
0.0001	76.688206

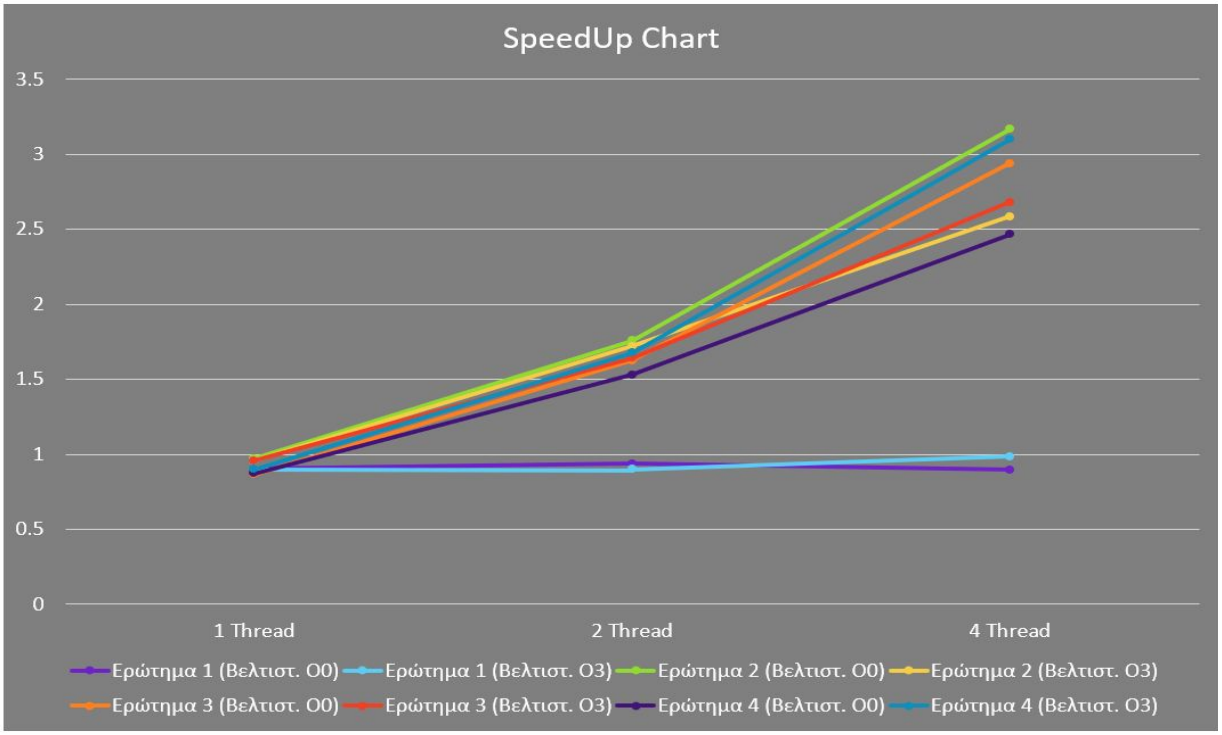
Τα παρακάτω αποτελέσματα που εμφανίζονται στον πίνακα είναι με χρήση της βελτιστοποίησης -O3 και σειριακή εκτέλεση του προγράμματος:

epsilon	Συνολικός Χρόνος Εκτέλεσης (sec)
0.1	8.927392
0.01	15.038672
0.001	22.288811
0.0001	29.423967

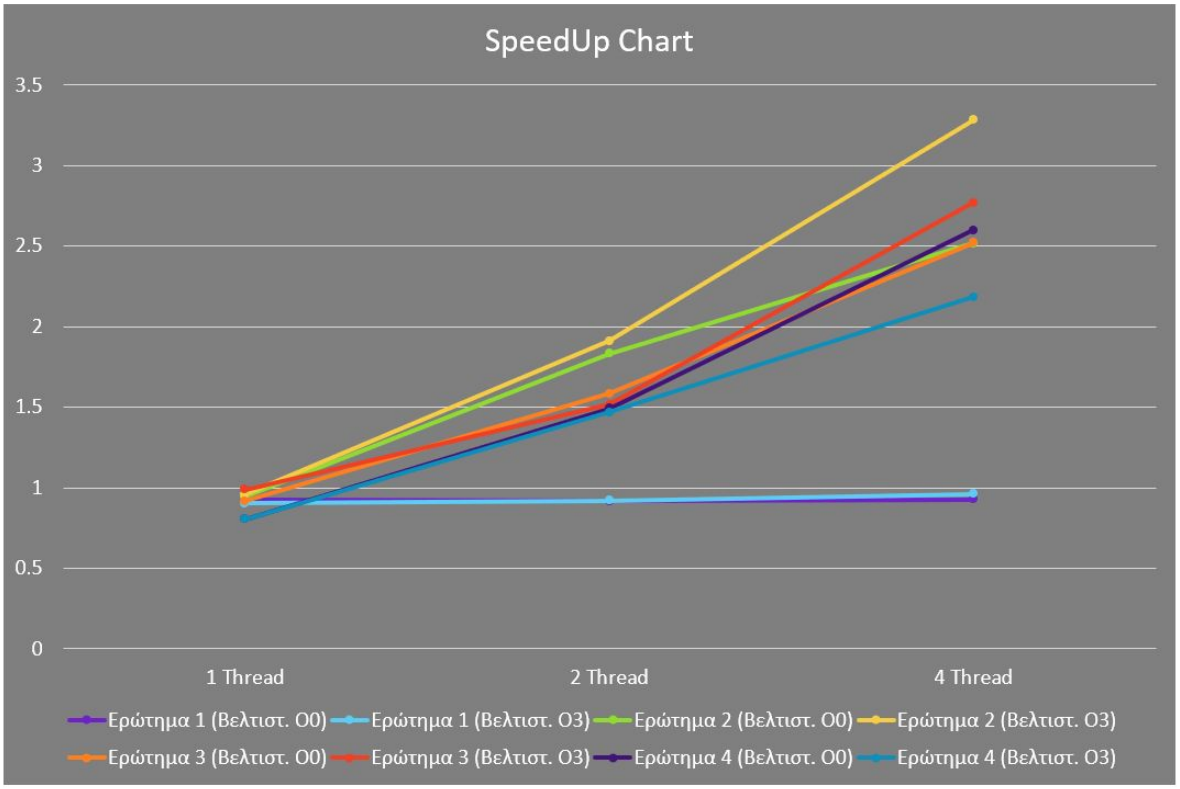
Έστω ότι επιλέγουμε την περίπτωση που η μεταβλητή epsilon είναι ίση με 0.1, τότε έχουμε τα παρακάτω διαγράμματα χρονοβελτίωσης για όλα τα ερώτημα.



Έστω ότι επιλέγουμε την περίπτωση που η μεταβλητή epsilon είναι ίση με 0.01, τότε έχουμε τα παρακάτω διαγράμματα χρονοβελτίωσης για όλα τα ερώτημα.



Έστω ότι επιλέγουμε την περίπτωση που η μεταβλητή epsilon είναι ίση με 0.001, τότε έχουμε τα παρακάτω διαγράμματα χρονοβελτίωσης για όλα τα ερώτημα.



Έστω ότι επιλέγουμε την περίπτωση που η μεταβλητή epsilon είναι ίση με 0.0001, τότε έχουμε τα παρακάτω διαγράμματα χρονοβελτίωσης για όλα τα ερώτημα.