ΜΥΕ023–Παράλληλα Συστήματα και Προγραμματισμός

Σετ Ασκήσεων #2

Ιωάννης Πρίφτη

ΑΜ: 3321

E-mail: cs03321@uoi.gr

Το 1ο σετ ασκήσεων αφορά στον παράλληλο προγραμματισμό μέσω του προτύπου OpenMP. Ζητείται η παραλληλοποίηση ενός σειριακού προγράμματος που αφορά τον πολλαπλασιασμό τετραγωνικών πινάκων στην γλώσσα προγραμματισμού C.

Όλες οι μετρήσεις έγιναν στο παρακάτω σύστημα:

Όνομα υπολογιστή	opti3060ws06
Επεξεργαστής	Intel core i3-8300
Πλήθος πυρήνων	4
Μεταφραστής	gcc v7.5.0

Άσκηση 1

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο πολλαπλασιασμός τετραγωνικών πινάκων με την χρήση του προτύπου OpenMP. Αυτό πρέπει να γίνει παραλληλοποιώντας έναν από τους 3 βρόχους κάθε φορά με χρήση 4 νημάτων. Τέλος ζητείται να δοκιμαστούν διαφορετικές μέθοδοι διαμοίρασης των επαναλήψεων (schedules).

Μέθοδος Παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα, όπως επίσης και πίνακες A,B,C μεγέθους 2048χ2048 από την ιστοσελίδα του μαθήματος.

Για την παραλληλοποίηση του πρώτου βρόχου απλά προστέθηκε η οδηγία: `#pragma omp parallel for num_threads(4) default(none) shared(A,B,C) private(i,j,k ) reduction(+:sum) schedule(static)` πριν τον πρώτο βρόχο του i. Αρχικά ορίζουμε την τιμή του πλήθους των νημάτων ίση με 4 ώστε να γίνει η εκτέλεση του προγράμματος από 4 νήματα. Η οδηγία `default(none)` τοποθετήθηκε ώστε να μπορούμε να ορίσουμε με απόλυτη ελευθερία και αυστηρά τις ιδιότητες των μεταβλητών όσον αφορά την χρήση τους από τα νήματα. Οι μεταβλητές i, j και k πρέπει να είναι ιδιωτικές ώστε κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζει τα υπόλοιπα ενώ οι πίνακες A, B και C είναι κοινόχρηστες. Όσον αφορά την μεταβλητή sum είναι και αυτή ιδιωτική για το κάθε νήμα με την διαφορά ότι στο τέλος του βρόχου του k τα αποτελέσματα συνδυάζονται και εκχωρούνται στην αντίστοιχη θέση του πίνακα C. Δεν απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επηρεάζει διαφορετικά στοιχεία του C. Τέλος έγινε διαμοίραση των επαναλήψεων στατικά σε αυτήν την περίπτωση και δυναμικά σε δεύτερη φάση.

Παρόμοια, για την παραλληλοποίηση του δεύτερου βρόχου απλά προστέθηκε η οδηγία:

```
#pragma omp parallel for num_threads(4) default(none) shared(A,B,C,i) private(j,k) reduction(+:sum) schedule(static)
```

πριν τον δεύτερο βρόχο του j. Όπως και πριν οι μεταβλητές A,B,C και επιπλέον i παραμένουν κοινόχρηστες όπως και j,k ιδιωτικές για τους λόγους που προαναφέρθηκαν. Όσον αφορά την μεταβλητή sum είναι και αυτή ιδιωτική για το κάθε νήμα με την διαφορά ότι στο τέλος του βρόχου του k τα αποτελέσματα συνδυάζονται και εκχωρούνται στην αντίστοιχη θέση του πίνακα C. Δεν απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επηρεάζει διαφορετικά στοιχεία του C. Τέλος έγινε διαμοίραση των επαναλήψεων στατικά σε αυτήν την περίπτωση και δυναμικά σε δεύτερη φάση.

Για την παραλληλοποίηση του τρίτου βρόχου, όμως χρειάστηκε να γίνει μια αλλαγή. Αφαιρέθηκε η αρχικοποίηση της μεταβλητής sum(sum = 0) έξω από τον βρόχο διότι κάθε νήμα την μηδένιζε πριν την ολοκλήρωση του βρόχου του k και έτσι δεν γινόταν σωστά ο υπολογισμός. Επιπλέον προστέθηκε η οδηγία:

```
#pragma omp parallel for num_threads(4) default(none) shared(A,B,C,i,j) private(k) reduction(+:sum) schedule(static)
```

πριν τον βρόχο του k. Όπως και πριν οι μεταβλητές A,B,C,i και επιπλέον j παραμένουν κοινόχρηστες όπως και η k ιδιωτική για τους λόγους που προαναφέρθηκαν. Όσον αφορά την μεταβλητή sum είναι και αυτή ιδιωτική για το κάθε νήμα με την διαφορά ότι στο τέλος του βρόχου του k τα αποτελέσματα συνδυάζονται και εκχωρούνται στην αντίστοιχη θέση του πίνακα C. Δεν απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επηρεάζει διαφορετικά στοιχεία του C. Τέλος έγινε διαμοίραση των επαναλήψεων στατικά σε αυτήν την περίπτωση και δυναμικά σε δεύτερη φάση.

Πειραματικά αποτελέσματα – μετρήσεις

Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέραμε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime() καλώντας την πριν την αρχή του βρόχου i(start=omp_get_wtime(); //get start time)και στο τέλος του (end=omp_get_wtime(); //get end time) για όλες τις παραπάνω μεθόδους παραλληλοποίησης και εκτυπώνοντας την διαφορά των χρόνων end και start πριν την ολοκλήρωση της συνάρτησης main.

Χρησιμοποιήθηκαν 4 νήματα και το πρόγραμμα εκτελέστηκε από 4 φορές για κάθε μέθοδο παραλληλοποίησης(schedule(static), schedule(dynamic)) και για κάθε βρόχο που παραλληλοποιήθηκε, άρα συνολικά 24 φορές.

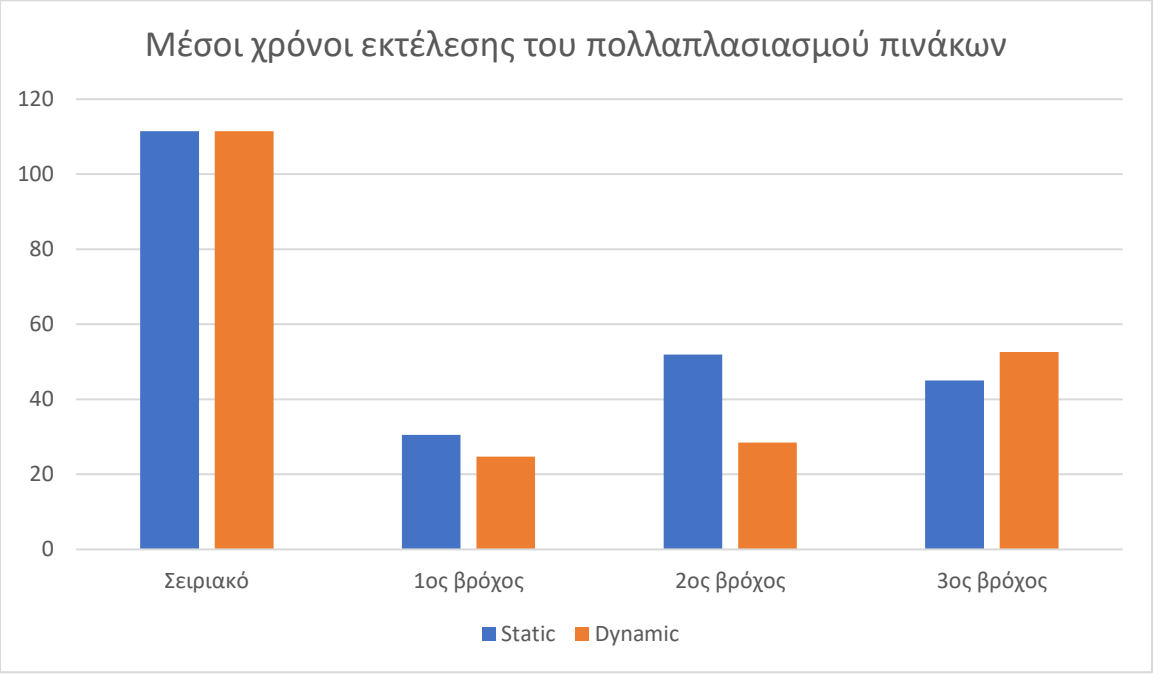
Στους παρακάτω πίνακες φαίνονται οι χρόνοι για κάθε περίπτωση καθώς και οι μέσοι όροι τους (οι χρόνοι είναι σε sec).

	1η εκτέλεση		2η εκτέλεση		3η εκτέλεση		4η εκτέλεση	
Βρόχος που παραλληλοποιήθηκε	static	dynamic	static	dynamic	static	dynamic	static	dynamic
1ος (του i)	24.063705	24.842311	35.574371	24.108897	37.620450	24.944762	24.783103	24.833604
2ος (του j)	53.492970	30.959487	45.962755	31.896065	54.076793	25.455713	54.227577	25.613918
3ος (του k)	37.694035	54.031375	38.768339	53.956403	49.975160	52.954249	53.784943	49.392811

	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση
Σειριακό	109,528320	111,682731	112,186703	112,372746

Μέσοι όροι

Βρόχος που παραλληλοποιήθηκε	static	dynamic
1ος (του i)	30.51040725	24.68239350
2ος (του j)	51.94002375	28.48129575
3ος (του k)	45.05561925	52.58340950
Σειριακό	111.442625	



Σχόλια

Με βάση τα παραπάνω αποτελέσματα παρατηρούμε όπως και αναμενόταν ότι το σειριακό πρόγραμμα είναι με διαφορά το πιο αργό σε σχέση με τα υπόλοιπα που παραλληλοποιήθηκαν και δοκιμάστηκαν με πίνακες μεγέθους 2048χ2048.

Παρατηρούμε επίσης ότι οι υπόλοιπες περιπτώσεις να είναι πολύ πιο γρήγορες από το σειριακό, περίπου έως και δύο φορές ταχύτερες.

Όσον αφορά τις δύο πρώτες περιπτώσεις που υλοποιήθηκαν, δηλαδή στην μία περίπτωση παραλληλοποιήθηκε ο πρώτος βρόχος με 4 νήματα και πολιτική static και έπειτα dynamic και σε δεύτερη φάση ο δεύτερος βρόχος αντίστοιχα. Τα αποτελέσματα ήταν τα αναμενόμενα, δηλαδή κατά την πολιτική static τα νήματα σπάνε το μπλοκ σε ισομεγέθη κομμάτια και τα υλοποιούν ανεξαρτήτως φόρτου που μπορεί να έχει το σύστημα εκείνη την στιγμή με πιθανότητα κάποια νήματα να τελειώνουν πιο γρήγορα την εκτέλεση του δικού τους κομματιού με συνέπεια να περιμένουν και να μην συμβάλλουν στον υπολογισμό των υπόλοιπων όπως στην πολιτική dynamic που δυναμικά τα νήματα εκτελούν διαφορετικά κομμάτια ανάλογα με τον φόρτο που έχει εκείνη την στιγμή το σύστημα. Έτσι στην περίπτωση της dynamic πολιτικής έχουμε μια μικρή βελτίωση στον χρόνο εκτέλεσης που είναι και αναμενόμενο για τον λόγο που προαναφέρθηκε.

Στην τρίτη υλοποίηση, δηλαδή στην παραλληλοποίηση του τρίτου βρόχου τα αποτελέσματα διαφέρουν λιγάκι σε σχέση με πριν, με τους χρόνος εκτέλεσης να είναι σχετικά αυξημένη σε σχέση με πριν με την διαφορά όμως στην static πολιτική να έχουμε λίγο ταχύτερη εκτέλεση σε σχέση με την πολιτική dynamic. Αυτό συμβαίνει διότι αν παρατηρήσουμε σε κάθε μια και από τις προηγούμενες περιπτώσεις όσο πηγαίνουμε πιο μέσα στους βρόχους που παραλληλοποιούμε(πρώτα τον πρώτο, μετά τον δεύτερο και τέλος τον τρίτο) το σύστημα επιβαρύνεται διότι πρέπει να εκτελούνται πολλαπλάσια κομμάτια ανάλογα με την πολιτική(dynamic ή static) διότι για παράδειγμα στην περίπτωση της static πολιτικής στην παραλληλοποίηση του τρίτου βρόχου ας υποθέσουμε ότι διαμοιράζονται οι επαναλήψεις σε N/P μπλοκ, αυτό σημαίνει ότι ο βρόχος κ θα ‘σπάσει’ σε N/P κομμάτια που θα εκτελεστούν από συγκεκριμένο νήμα το καθένα, όμως αφού ολοκληρωθεί η εκτέλεση του βρόχου αυτό θα επέλθει για επόμενο j(από τον προηγούμενο βρόχο) η εκτέλεση του ξανά και αυτό γίνεται πολύ πιο συχνά στον βρόχο k, λιγότερο στον j και τέλος πολύ λιγότερο στον i. Άρα όσο παραλληλοποιούμε εσωτερικότερους βρόχους τα νήματα θα υλοποιούν όλο και περισσότερα μπλοκ, το οποίο σημαίνει ότι θα υπάρχουν όλο και περισσότερες εναλλαγές και αναμονές μεταξύ του. Εκεί οφείλονται και οι αυξημένοι χρόνοι, με την παραλληλοποίηση του πρώτου βρόχου να έχουμε τους πιο γρήγορους χρόνους και με την παραλληλοποίηση του τρίτου τους πιο αργούς, φυσικά πάντα πιο γρήγορα από το σειριακό.

Άσκηση 2

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο πολλαπλασιασμός τετραγωνικών πινάκων με την χρήση του προτύπου OpenMP(tasks). Αυτό πρέπει να γίνει χρησιμοποιώντας τα tasks που μας προσφέρει το OpenMP με χρήση 4 νημάτων. Τέλος ζητείται να δοκιμαστούν διάφορα μεγέθη για τα μπλοκ τα οποία και πρέπει να τα δίνουμε στο πρώτα μέσω της συνάρτησης main.

Μέθοδος Παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα, όπως επίσης και πίνακες A,B,C μεγέθους 2048χ2048 από την ιστοσελίδα του μαθήματος.

Αρχικά όπως και στην προηγούμενη άσκηση η χρονομέτρηση γίνεται με τον ίδιο ακριβώς τρόπο.

Στην συνέχεια προστέθηκε η γραμμή:
`#pragma omp parallel num_threads(4) default(none) private(i,j) shared(A,B,C,S) reduction(+:sum)` στην ίδια ακριβώς θέση και για τους ίδιους ακριβώς λόγους όπως αι στην περίπτωση της παραλληλοποίησης του πρώτου βρόχου στην προηγούμενη άσκηση.

Υπάρχουν όμως δύο διαφορές. Αρχικά παρατηρούμε ότι δεν υπάρχει η οδηγία for, αυτό συμβαίνει για τον λόγο ότι δεν θέλουμε να διαμοιράσει η OpenMP τα threads και τις επαναλήψεις αυτόματα διότι τότε δεν θα μπορούσαμε να πετύχουμε τον στόχο της άσκησης ο οποίος είναι να ‘σπάσουμε’ τον υπολογισμό σε υποπίνακες με tasks. Στη συνέχεια έχει προστεθεί η οδηγία: `#pragma omp single nowait` ώστε ένα νήμα την φορά να εκτελεί και επιπλέον τα υπόλοιπα μπορούν να συνεχίσουν χωρίς να περιμένουν να τελειώσει την εκτέλεση του το πρώτο νήμα.

Όσον αφορά τους βρόχους I και j έχει προστεθεί στο βήμα η τιμή του S ώστε να πετύχουμε τον υπολογισμό των επόμενων κάθε φορά υποπινάκων.

```
for (i = 0; i < N; i+=S){
      for (j = 0; j < N; j+=S){
```

Στη συνέχει έχει προστεθεί η οδηγία:
`#pragma omp task default(none) firstprivate(i,j) private(ii,jj,k,sum) shared(A,B,C,S)` που σημαίνει ότι οι μεταβλητές i και j και είναι ιδιωτικές στα νήματα που θα υλοποιήσουν το ερχόμενο μπλοκ με την διαφορά ότι θα αρχικοποιηθούν με τις τιμές που είχαν οι μεταβλητές προηγουμένως. Οι μεταβλητές ii,jj,k,sum είναι ιδιωτικές για κάθε νήμα ώστε να μπορεί το κάθε νήμα να εκτελέσει τον παρακάτω κώδικα χωρίς να επηρεάσει την εκτέλεση των υπολοίπων νημάτων αντίστοιχα και έτσι να προκύψει το ζητούμενο αποτέλεσμα. Επιπλέον ορίζουμε ως task το ερχόμενο κομμάτι κώδικα.

Στη συνέχεια έχουν προστεθεί στον κώδικα δύο επιπλέον βρόχοι οι οποίοι είναι υπεύθυνοι για να διατρέξουν τον κάθε υποπίνακα μεγέθους S από εκεί που έχουν ‘πει’ οι παραπάνω βρόχοι i και j, γι’αυτό και βλέπουμε ii=i και jj=j αντίστοιχα στους δύο βρόχους.

```
for(int ii=i; ii<S+i; ii++){
      for(int jj=j; jj<S+j; jj++){
```

Τέλος, ο βρόχος του k έχει παραμένει ως έχει.

Πειραματικά αποτελέσματα – μετρήσεις

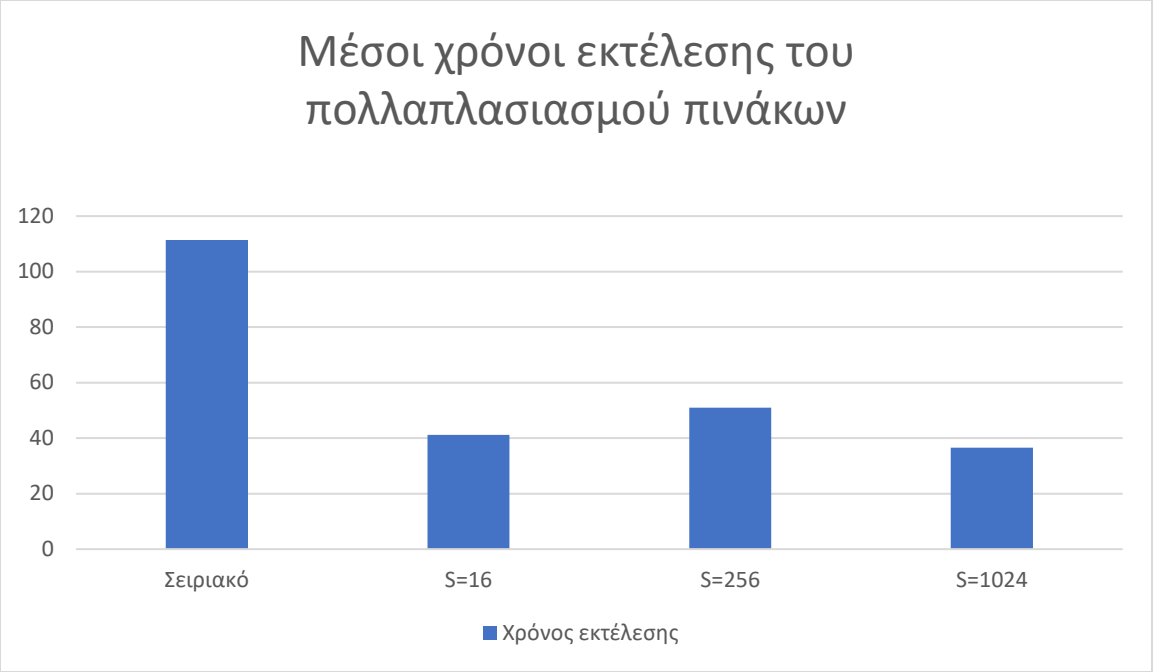
Οι μετρήσεις όπως και στην προηγούμενη άσκηση έγιναν κρατώντας τον χρόνο έναρξης και λήξης του βρόχου i με τις γραμμές `start=omp_get_wtime(); //get start time` στην αρχή και `end=omp_get_wtime(); //get end time` στο τέλος αντίστοιχα.

Το πρόγραμμα εκτελέστηκε από 4 φορές για S=16,256,1024 και έγινε οι καταγραφή των χρόνων και των μέσων όρων τους στους παρακάτω πίνακες(Όλοι οι χρόνοι είναι σε sec.).

S	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση
16	38.86892	40.114973	45.854929	39.811433
256	54.682861	54.446733	52.134774	42.784065
1024	35.746005	37.210199	37.580422	35.564744
Σειριακά	109.528320	111.682731	112.186703	112.372746

Μέσοι όροι

S	
16	41.16256375
256	51.01210825
1024	36.52534250
Σειριακά	111.442625



Σχόλια

Παρατηρούμε ότι το σειριακό είναι έως και δύο φορές πιο αργό από τις υπόλοιπες υλοποιήσεις που χρησιμοποιούν tasks. Για S=16 και για S=256 οι χρόνοι είναι λογικοί όπως επίσης για S=256 έχουμε ελαφρώς πιο αργή εκτέλεση σε σχέση για S=16. Όσον αφορά την περίπτωση για S=1024 περίμενα ότι θα έχει ελαφρώς πιο αργή εκτέλεση από την περίπτωση για S=256, δεν ξέρω να το εξηγήσω. Αυτό που μπορώ να πω με σιγουριά είναι ότι το πρόγραμμα δουλεύει σωστά και βγάζει τον πίνακα C όπως πρέπει.

Άσκηση 2

Δεν πρόλαβα να την κάνω.