ΜΥΕ023-Παράλληλα Συστήματα και Προγραμματισμός

Σετ Ασκήσεων #1

Χρήστος Καραγιαννίδης
ΑΜ : 4375
Email: cs04375@uoi.gr

Όνομα υπολογιστή	opti3060ws10
Επεξεργαστής	Intel i3-8300
Πλήθος πυρήνων	4
Μεταφραστής	gcc v7.5.0

Άσκηση 1

Το προβλημα

Στην ασκηση αυτη ζητείται να παραλληλοποιηθεί η ευρεση του πλήθους των πρωτων αριθμων και του μεγαλύτερου πρωτου αριθμου δεδομένου του ανω ορίου Ν.

Μέθοδος παραλληλοποίησης

Χρησιμοποιηθηκε το σειριακό προγραμμα απο την ιστοσελίδα του μαθήματος. Για την παραλληλοποίηση της μοναδικής for προστέθηκαν οι οδηγίες :

#pragma omp parallel private(num,quotient,divisor,remainder) reduction(max:lastprime) num_threads(4)

#pragma omp for schedule (dynamic,5)

Η μεταβλητή i του for loop ειναι απο default private και ετσι πρέπει για να εχει καθε νήμα που θα κανει ενα μερος της εκτελεσης τα δικά του iterations να κανει (πχ. 1^{ο} νημα 0-25000, 2^{ο} 25001-50000 κ.ο.κ).Οι μεταβλητες num,quotient,divisor και remainder ειναι private γιατι πρεπει καθε νημα να υπολογιζει τα δικα του αποτελέσματα ανεξαρτητα απο τα υπόλοιπα

και αφου το num εξαρταται απο το i και τα quotient , remainder εξαρτώνται απο το num αρα τα num, quotient, remainder δεν μπορουν να ειναι shared αφου το i εχει εντελώς διαφορετικο ευρος τιμων για καθε νημα. Το divisor δεν εξαρταται απο καποια αλλη μεταβλητη ομως αυξανεται κατα 2 οσο remainder && (bitwise and) divisor <= quotient οποτε θελουμε και αυτο να ειναι private γιατι αν ειναι shared ενα νήμα μπορει να εχει divisor=602 και ενα αλλο divisor=1 , αν οποιοδήποτε απο τα 2 διαβασει την μεταβλήτη divisor με την τιμή που έθεσε το αλλο θα γινει λαθος πράξη μεσα στην do while (και στον υπολογισμο της συνθηκης της do while) . Το reduction(max:lastprime) ειναι εκεί ωστε στο τελος της εκτέλεσης ολων των νηματων να κρατήσουμε απο τις NUM_THREADS private μεταβλητες lastprime (γινεται αυτοματα private η lastprime οταν μπει στην reduction) την μεγαλυτερη απο αυτές που θα ειναι ο αριθμός που στο τέλος θα τυπωσουμε . Αν ξεραμε οτι τα νηματα θα εκτελεστουν παραλληλα αλλα θα τελειώσει τελευταίο το νήμα με το μεγαλυτερο ανω οριο εύρους τιμών (πχ. Για for(int i=0;i<N;i++) [75000 – N]) , αρα το νήμα που θα εχει τον μεγαλύτερο lastprime δεν θα χρειαζόταν να βαλουμε την reduction ομως δεν το ξερουμε και δεν θελουμε η lastprime να εχει την τιμη του νήματος που εκτελέστηκε τελευταίο.

Πειραματικά αποτελέσματα - μετρήσεις

Το πρόγραμμα εκτελέστηκε στο σύστημα που αναφέρεται στην εισαγωγη και η χρονομέτρηση εγινε με τη συναρτηση gettimeofday της βιβλιοθήκης <sys/time.h>.
Χρησιμοποιήθηκαν απο 1 εως 4 νήματα.
Κάθε πειραμα εκτελεστηκε 4 φορες και υπολογιστηκαν οι μεσοι οροι.
Η χρονομέτρηση ξεκιναει αμέσως πριν την κλήση της καθε συνάρτησης και σταματάει αμέσως μετά.

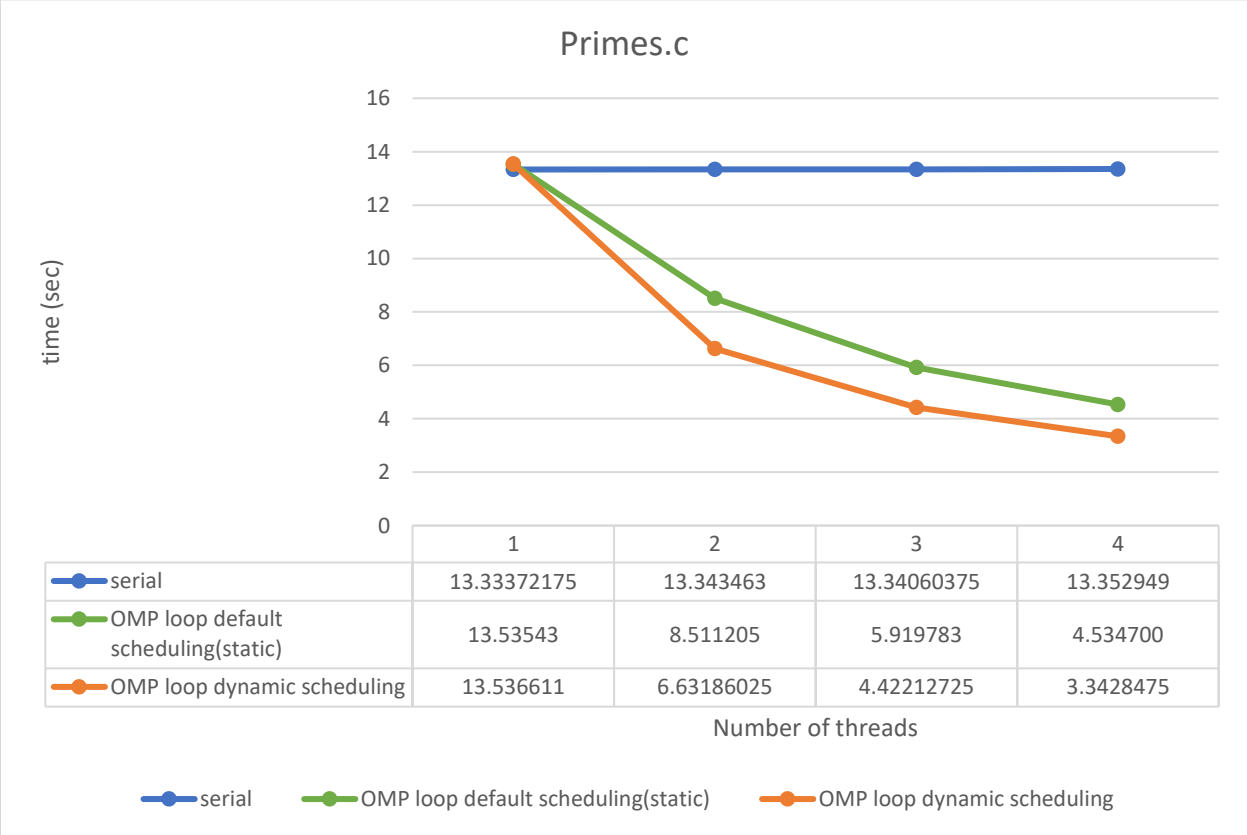
1 thread		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	Serial	13.330755	13.330564	13.332152	13.330890
	Omp loops	13.536357	13.534838	13.534655	13.535862

2 threads		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	Serial	13.336902	13.331979	13.332439	13.331793
	Omp loops	8.645563	8.465713	8.466796	8.466747

3 threads		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	Serial	13.337331	13.335599	13.336308	13.334561
	Omp loops	5.919096	5.918935	5.919413	5.921686

4 threads		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	Serial	13.331287	13.332746	13.331147	13.330888
	Omp loops	4.534906	4.534997	4.534671	4.534224

Με βαση τους παραπανω πίνακες βγήκαν οι μεσοι όροι και η παρακάτω γραφική παράσταση.



Σχόλια

Παρατηρούμε οτι η σειριακη εκτέλεση (για UPTO = 10000000 στην 6η γραμμη του primes.c) ειναι σταθερα στα 13.3 sec οπως επίσης και η παράλληλη για αριθμό νημάτων = 1. Ωστοσο οσο αυξάνονται τα νηματα η παράλληλη εκτελεση μειώνει τον χρονο της και τείνει να

πλησιάσει το 1/NUM_THREADS το οποίο θα ήταν ιδανικό . Μπορούμε ομως να διακρίνουμε αλλη μια εκτέλεση η οποια ειναι σχεδόν ιδια με την προήγούμενη παραλληλη με την μονη τους διαφορα να ειναι οτι η μια ειναι με default scheduling ενω ή άλλη ειναι με dynamic στο οποιο παρατηρηθηκε μια αρκετα σημαντική βελτίωση στους χρόνους καθως πλέον δεν ήμασταν με static που ειναι το default αρα δεν δημιουργήθηκαν 4 ισες υποδιαιρεσεις του for loop και καθε νήμα πήρε μία. Με το dynamic δημιουργήθηκαν μικρότερα chunks και οποιο νήμα ήταν ελευθερο αναλάμβανε ενα chunk ετσι δεν υπήρχε καποιο νήμα που να περιμένει τα άλλα να τελειώσουν και γι αυτο και ειδαμε σημαντική βελτίωση στους χρόνους. Το static scheduling ακομα και όταν δώθηκε συγκεκριμένο chunk size (5 και 10) ήταν πιο αργό καθως το νήμα 1 επρεπε να περιμένει να τελειωσει την εκτέλεση του το νήμα 4 προκειμενου να αναλάβει αλλο chunk. Το guided ανεξαρτήτως του chunk size ηταν ελαφρώς πιο αργο (δεκατα του δευτερολέπτου) αν οχι παρομοιο χρονικά με το dynamic. Οπότε στο τελός επιλέχθηκε το dynamic.

Άσκηση 2

Το προβλημα

Στην ασκηση αυτη ζητείται να παραλληλοποιηθεί η συναρτηση θόλωσης εικόνας με ακτίνα r=8 για την εικόνα 1500.bmp με 2 τροπους α) με omp loops β) με omp tasks.

Μέθοδος παραλληλοποίησης

Χρησιμοποιηθηκε το σειριακό προγραμμα απο την ιστοσελίδα του μαθήματος.

Omp loops

Για την παραλληλοποίηση της πρώτης for προστέθηκαν οι οδηγίες :

#pragma omp parallel private(weightSum,redSum,greenSum,blueSum,row,col,j) num_threads(4)

#pragma omp for

Η μεταβλητή i του for loop ειναι απο default private και ετσι πρέπει για να εχει καθε νήμα που θα κανει ενα μερος της εκτελεσης τα δικά του iterations να κανει (πχ. 1^{ο} νημα 0-25000, 2^{ο} 25001-50000 κ.ο.κ).Για τον ιδιο λόγο πρεπει να ειναι private και οι row, col, j και αυτες απο default ειναι shared γιατι δεν ειναι μερος της for που μοιραζεται στα νήματα. Οι μεταβλητες weightSum,redSum,greenSum και blueSum ειναι private γιατι πρεπει καθε νημα να υπολογιζει τα δικα του αποτελέσματα για τις rgb τιμες του καθε pixel ανεξαρτητα απο τα υπόλοιπα.Η περιοχη εξω απο την 3^{η} for loop (γραμμες 277-286) θα ηταν critical περιοχή αν 2

νήματα επηρέαζαν την τιμή ενος pixel ομως αφου στον πινακα red green και blue καθε φορα πηγαινουμε στην θεση i*width+j ακομα και τα width και j του ενος νηματος να τυχαίνει να ειναι ιδια σε μια χρονικη στιγμη δεν θα «προσπελάσουν» και τα 2 νηματα το ιδιο pixel αφου θα εχουν σιγουρα διαφορετικο i . Επομένως δεν ειναι απαραίτητο να μπει critical εκει και το μονο που θα προσφέρει ειναι επιπλέον καθυστέρηση.

Omp tasks

Σημειωση : Η συγκεκριμενη συναρτηση παρόλο που τρέχει καλα χρονικά και κανει blur το μεγαλυτερο μέρος της εικονας εμφανίζει random μαυρες γραμμες οι οποιες πιστευω οτι ειναι επειδη ενα task/νημα προλαβαίνει να μηδενίσει τα redSum,greenSum,blueSum,weightSum και ενα αλλο task παει και κανει assign τις μηδενικες τιμες σε μια ολοκληρη γραμμη αρα ολα εκεινα τα pixel εχουν (r,g,b) = (0,0,0) και συνεπως το χρωμα μαυρο. Δοκιμασα να βαλω critical αρκετες περιοχές ομως δεν κατάφερα να κανω τις γραμμες να εξαφανιστούν.

Για την παραλληλοποίηση προστέθηκαν οι οδηγίες :

#pragma omp parallel num_threads(4) εξω απο τη πρωτη for

#pragma omp task firstprivate(row,col,redSum,greenSum,blueSum,weightSum) εξω απο την 3η for

Οι μεταβλητες weightSum,redSum,greenSum και blueSum ειναι private γιατι πρεπει καθε νημα να υπολογιζει τα δικα του αποτελέσματα για τις rgb τιμες του καθε pixel ανεξαρτητα απο τα υπόλοιπα. Η μεταβλητη col πρεπει να ειναι private για να εχει το καθε task τον δικο του αριθμο επαναλήψεων για εκεινο το loop.

Πειραματικά αποτελέσματα - μετρήσεις

Το πρόγραμμα εκτελέστηκε στο σύστημα που αναφέρεται στην εισαγωγη και η χρονομέτρηση εγινε με τη συναρτηση timeit() που δόθηκε η οποία χρησιμοποιεί gettimeofday της βιβλιοθήκης <sys/time.h>.
Χρησιμοποιήθηκαν απο 1 εως 4 νήματα.
Κάθε πειραμα εκτελεστηκε 4 φορες και υπολογιστηκαν οι μεσοι οροι.
Η χρονομέτρηση ξεκιναει αμέσως πριν την κλήση της καθε συνάρτησης και σταματάει αμέσως μετά.

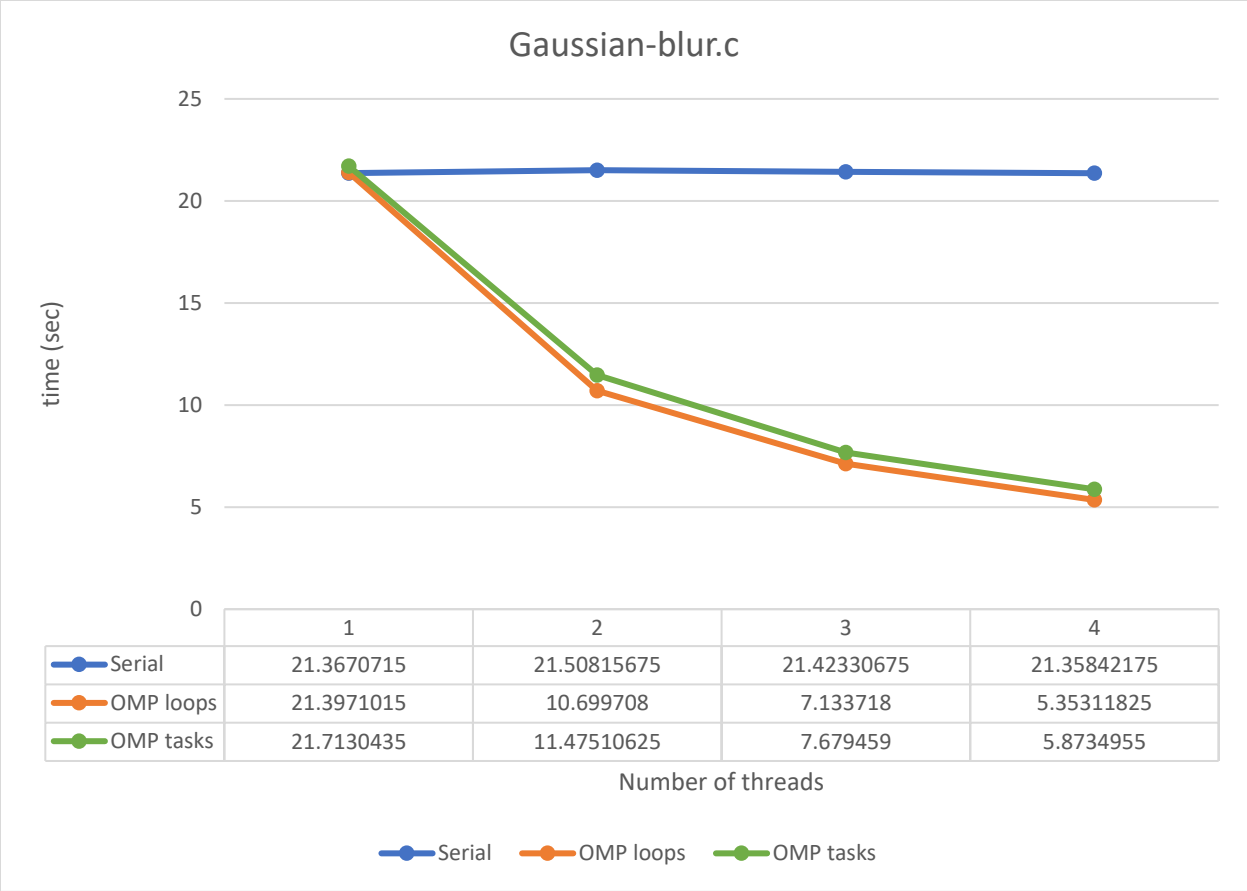
1 thread		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	serial	21.48412	21.48069	21.32603	21.35396
	loop	21.58044	21.54791	21.36794	21.38646
	task	21.73623	21.74619	21.63684	21.61163

2 threads		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	serial	21.31626	21.58522	21.34197	21.32658
	loop	10.77119	10.76733	10.76821	10.7694
	task	11.47343	11.52256	11.46987	11.48276

3 threads		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	serial	21.47393	21.33118	21.32139	21.33652
	loop	7.221964	7.200094	7.184858	7.18158
	task	7.683101	7.711657	7.653792	7.7184

4 threads		1η επανάληψη	2η επανάληψη	3η επανάληψη	4η επανάληψη
	serial	21.32239	21.42412	21.39045	21.35829
	loop	5.387757	5.387511	5.39125	5.389776
	task	5.827571	5.836976	5.834502	5.851125

Με βαση τους παραπανω πίνακες βγήκαν οι μεσοι όροι και η παρακάτω γραφική παράσταση.



Σχόλια

Παρατηρούμε οτι η σειριακη εκτέλεση (για r=8 και αρχειο εισόδου 1500.bmp) ειναι σταθερα στα 21.4 sec οπως επίσης και η παράλληλη με loops αλλα και η παράλληλη με tasks για αριθμό νημάτων = 1 . Ωστοσο οσο αυξάνονται τα νηματα οι παράλληλες εκτελέσεις μειώνουν τον χρονο τους και τείνουν να πλησιάσουν το 1/NUM_THREADS το οποίο θα ήταν ιδανικό . Φαινεται επίσης η omp loops εκτελεση να ειναι ελαφρώς πιο γρήγορη απο την tasks και αυτο γιατι η tasks δημιουργεί το task αλλα δεν το εκτελεί κατευθειαν , αφήνει να το εκτελέσει καποιο αλλο νημα καποια αλλη στιγμή.

Άσκηση 3

Περιγραφή

Η taskloop ορίζεται ως ***#pragma omp taskloop [clause[[,]clause]…] newline*** . Όταν ένα νήμα συναντάει την εντολή taskloop τότε δημιουργούνται τοσα tasks οσα και τα νήματα που εχουν οριστει στην #pragma omp parallel. Κάθε ενα απο τα tasks που δημιουργήθηκαν εκτελεί ολες τις επαναλήψεις του loop που ακολουθει. Τα tasks αφου δημιουργηθουν θα τα αναλαβει καποιο διαθεσιμο νημα καποια στιγμη ομως δεν ειναι σιγουρο οτι καθε νημα θα παρει και απο ενα οποτε δεν ειναι και σιγουρο οτι οι NUM_THREADS επαναληψεις θα εκτελεστούν παράλληλα (αυτο εχει ως αποτελεσμα χειροτερο χρονο απο τη σειριακη εκτελεση) . Τα δεδομένα που θα πάρει το κάθε task εξαρτώνται από τα clauses και τον τύπο διαμοιρασμού των δεδομένων για την κάθε μεταβλητή(πχ firstprivate(x,y) , shared(x,y) etc..) , ωστοσο ως default οι μεταβλητες ειναι shared. Η σειρά με την οποία δημιουργούνται τα tasks για το loop δεν είναι συγκεκριμένη και αφού και η σειρά εκτέλεσης τους δεν είναι προκαθορισμένη , η εντολή taskloop δεν ενδείκνυται για loops στα οποία η σειρά εκτέλεσης είναι απαραίτητο να τηρηθεί.

Αν υπαρχει ενα atomic στη δομη της taskloop τοτε η πρόσβαση στις συγκεκριμένες θεσεις μνήμης (που περιέχονται στην #pragma omp atomic) επιτρέπεται απο ενα νήμα/task τη φορά. Λειτουργεί οπως η critical δηλαδη δεν αφήνει σε αλλο νήμα/task να εκτελέσει εκεινο το κομματι κωδικα αν δεν ειναι το μονο που το εκτελει εκεινη την στιγμή.

Άμα υπάρχει ένα reduction στη δομή της taskloop τότε κατά το τέλος της εκτέλεσης των tasks θα χρησιμοποιηθεί ο operator που δόθηκε για να κάνει τη πράξη αναμεσά στα αντικείμενα που ανήκουν στην λίστα των tasks που ανήκουν στο ίδιο taskgroup. Η δομή taskloop εκτελείται σαν κάθε task που δημιουργήθηκε να ορίστηκε με μια δομή task στην οποία υπήρχε ένα in_reduction clause με τον ίδιο operator και αντικείμενα λίστας. Επομένως τα tasks που παράχθηκαν είναι μέλη του reduction που ορίστηκε από το task_reduction clause που εφαρμόστηκε στη δομή taskgroup (η οποία ουσιαστικά δημιουργείται όταν καλούμε την taskloop).

Άμα υπάρχει ένα in_reduction clause στη δομή του taskloop τότε η συμπεριφορά θα είναι σαν κάθε task που δημιουργήθηκε , ορίστηκε από μια δομή task στην οποία υπήρχε ένα in_reduction clause με τον ίδιο operator και αντικείμενα στην λίστα. Συνεπώς τα tasks που παράχθηκαν είναι μέλη ενός reduction που ορίστηκε προηγουμένως.

Άμα υπάρχει ένα grainsize clause στη δομή του taskloop τότε ο αριθμός του κάθε μέρους των επαναλήψεων που θα ανατεθεί σε κάθε task θα είναι ίσος ή μεγαλύτερος του αριθμού που θα

δοθεί και λιγότερο από 2 φορές την τιμή που θα δοθεί. Η παράμετρος που θα δοθεί με την grainsize πρέπει να είναι ένας θετικός ακέραιος αριθμός (ή έκφραση που αποτιμάται σε θετικό ακέραιο). Αν προσδιορίζεται ο αριθμός num_tasks τότε η taskloop δημιουργεί τόσα tasks όσο είναι το ελάχιστο της έκφρασης της num_tasks και των αριθμών των επαναλήψεων. Κάθε task πρέπει να έχει τουλάχιστον 1 επανάληψη/εκτέλεση. Η παράμετρος num_tasks πρέπει και αυτή να είναι ένας θετικός ακέραιος αριθμός. Αν δεν υπάρχει ούτε η num_tasks ούτε η grainsize τότε ο αριθμός των tasks που θα δημιουργηθούν και ο αριθμός των επαναλήψεων που θα αναλάβει το καθένα είναι προκαθορισμένο.

Αν παραπάνω από 1 loops είναι συνδεδεμένο με την δομή taskloop τότε ο αριθμός των φορών που οποιοσδήποτε κώδικας που επεμβαίνει αναμεσά σε δυο συσχετισμένα loops θα εκτελεστεί, είναι απροσδιόριστο αλλά θα γίνει τουλάχιστον μια φορά ανά επανάληψη του loop που περιέχει τον κώδικα αυτόν.

Το taskloop έχει επαναλήψεις που είναι αριθμημένες από το 0 έως το N-1 οπού N ο αριθμός επαναλήψεων. Ο υπολογισμός των επαναλήψεων γίνεται πριν την είσοδο στο εξωτερικό loop. Αν η εκτέλεση κάποιου συ σχετιζόμενου loop αλλάξει οποιαδήποτε από τις τιμές που χρησιμοποιούνται για τον υπολογισμό οποιουδήποτε μετρητή επαναλήψεων τότε είναι απροσδιόριστη η συμπεριφορά.

Αν υπάρχει ένα if clause στη δομή του taskloop και αν η έκφραση μέσα στην if αποτιμηθεί σε false τότε τα tasks φτιάχνονται και καλούνται χωρίς καθυστέρηση.

Αν υπάρχει ένα final clause και η έκφραση μέσα στο final αποτιμηθεί σε true τα tasks που θα δημιουργηθούν θα είναι final tasks.

Αν υπάρχει ένα untied clause τα tasks που θα δημιουργηθούν θα είναι untied tasks.

Αν υπάρχει ένα mergeable clause τα tasks που θα δημιουργηθούν θα είναι mergeable tasks.

Αν υπάρχει ένα priority clause στη δομή του taskloop τα tasks που παράγονται χρησιμοποιούν ως τιμή προτεραιότητας την τιμή που δίνεται σαν να είχε προσδιοριστεί για κάθε task ξεχωριστά. Αν δεν δοθεί τιμή προτεραιότητας τότε τα tasks έχουν default τιμή προτεραιότητας 0.

Πρόγραμμα

Το πρόγραμμα πάνω στο οποίο θα εφαρμόσουμε την **#pragma omp taskloop** είναι μια παραλλαγή του προγράμματος matmul.c το οποιο βρίσκεται στην σελιδα ενισχυτικης διδασκαλιας του μαθήματος στο lab 3. Η παραλλαγή αυτού του προγράμματος ονομάζεται mymatmul.c . Το προγραμμα υπολογιζει για καθε i,j το γινομενο A[i][j] *B[i][j] και το κανει assign στην θεση i,j ενός πίνακα C. Προστέθηκε η συναρτηση **zero_arrays()** η οποία μηδενιζει τα περιεγχομενα του πινακα C αμεσως μετα την σειριακη εκτέλεση ωστε να ειναι κενος κατα

την παράλληλη και τροποποιήθηκε η συνάρτηση **print_results()** και πλεον κραταει την τιμη του στοιχειου 0,0 του πινακα C σε μια μεταβλητη first και ελεγχει ολα τα στοιχεια του πινακα . Αν βρει καποιο στοιχειο που να ειναι διαφορετικό απο τη μεταβλητη first κανει print οτι εγινε λαθος στους υπολογισμους (Ξέρουμε οτι ολα τα στοιχεια του C πρεπει να εχουν την ιδια τιμη αφου ολα πολλαπλασιάστηκαν με τους ιδιους αριθμους).

Στην συναρτηση **matmul_OMP_taskloop()** προστέθηκαν οι :

#pragma omp parallel num_threads(4)

#pragma omp taskloop private(i,j,k) shared(A,B,C)

Εξω απο την 1η for . Θελουμε καθε taskloop να εχει τους δικους του μετρήτες για τα iterations αλλα να διαβάζουν τους ιδιους πινακες A,B και να γραφουν στον ιδιο πινακα C οποτε οι i,j,k ειναι private και οι A,B,C ειναι shared . Επιπλέον επρεπε να βάλουμε την
#pragma omp atomic
κατα την εγγραφή στον πίνακα C γιατι αν δεν μπεί τοτε δύο task μπορεί να διαβασουν την τιμη του C για i,j εστω C[i][j] = 2 , να πάει το task1 να αυξήσει το 2 κατα 16 και αρα να γινει 18 και οταν παει το task2 να το αυξησει και αυτο κατα 16 δεν εχει διαβασει την ενημερμένη τιμη του C[i][j] και συνεπως κανει την προσθεση 2+16 και αυτο αντι για την πραξη 18+16 που επρεπε να κανει. Ετσι χανονται πραξεις και οι τιμές των στοιχειων του πινακα C ειναι απροσδιόριστες. Με την **#pragma omp atomic** εξασφαλίζουμε οτι μονο ενα νήμα ανα πασα χρονική στιγμη θα διαβασει και θα γραψει στον πινακα C. Παρατηρουμε λοιπον οτι κατα την εκτέλεση του προγραμματος ανάλογα με τον αριθμό νημάτων που δινεται τα περιεγχομενα του πινακα C ειναι συμφωνα με τον τυπο : C_taskloop[i][j] = NUM_THREADS * C_serial[i][j] επειδη οπως αναφέρθηκε στην περιγραφη το taskloop φτιαχνει τοσα tasks οσα ειναι και τα νηματα και καθε task εκτελει ΟΛΑ τα iterations.

Πειραματικά αποτελέσματα - μετρήσεις

Το πρόγραμμα εκτελέστηκε στο σύστημα που αναφέρεται στην εισαγωγη και η χρονομέτρηση εγινε με τη συναρτηση timeit() που δόθηκε η οποία χρησιμοποιεί gettimeofday της βιβλιοθήκης <sys/time.h>.
Χρησιμοποιήθηκαν απο 1 εως 4 νήματα.
Κάθε πειραμα εκτελεστηκε 4 φορες και υπολογιστηκαν οι μεσοι οροι.
Η χρονομέτρηση ξεκιναει αμέσως πριν την κλήση της καθε συνάρτησης και σταματάει αμέσως μετά.

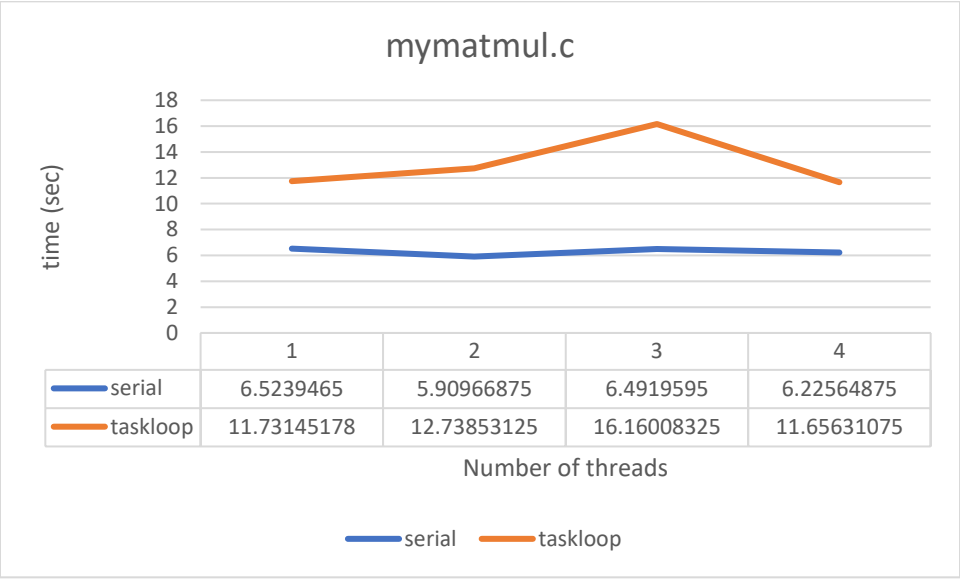
1 thread					
	serial	5.821902	6.631458	6.839663	6.802763
	taskloop	11.26391	11.71129	11.95495	11.99566

2 threads					
	serial	5.826289	5.920279	5.900056	5.992051
	taskloop	11.71896	16.48032	11.37226	11.3826

3threads					
	serial	7.533244	5.824177	6.788591	5.821826
	taskloop	19.40101	13.47543	19.71186	12.05204

4threads					
	serial	5.891451	6.679466	5.824205	6.507473
	taskloop	11.57878	11.90442	11.37038	11.77167

Με βαση τους παραπανω πίνακες βγήκαν οι μεσοι όροι και η παρακάτω γραφική παράσταση.



Σχόλια

Παρατηρούμε οτι το γεγονός πως κάθε νημα κανει ολα τα iterations αποτυπώνεται και στους χρονους εκτέλεσης καθως η εκτέλεση του taskloop ειναι αρκετα μεγαλύτερη του serial. Παρατηρηθηκαν επίσης μεγάλες διακυμάνσεις στα αποτελέσματα.