1ο Σετ Ασκήσεων Παράλληλα Συστήματα και Προγραμματισμός

Μάθημα: Παράλληλα Συστήματα και Προγραμματισμός(ΜΥΕ023)
Διδάσκων: Δημακόπουλος Βασίλειος
Ονοματεπώνυμο: Καζακίδης Θεοχάρης
ΑΜ: 4679
Εξάμηνο: Εαρινό
Ακαδημαϊκό Έτος: 2022-2023

Το πρώτο σετ ασκήσεων αφορά στο παραλληλισμό δύο εφαρμογών με χρήση του προγραμματιστικού μοντέλου OpenMP. Ζητείται η παραλληλοποίηση προγράμματος για την εύρεση πρώτων αριθμών καθώς και για θόλωση εικόνας (Gaussian Blur). Τέλος ζητείται συνοπτική παρουσίαση της λειτουργίας taskloop του OpenMP.

Όλες οι μετρήσεις που θα παρουσιαστούν έγιναν στο παρακάτω σύστημα:

Όνομα υπολογιστή	dl380ws06
Επεξεργαστής	Intel(R) Xeon(R) CPU L5520 @ 2.27GHz
Πλήθος πυρήνων	4
Μεταφραστής	gcc version 11.2.0

Άσκηση 1

→Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο αλγόριμθος εύρεσης πρώτων αριθμών. Πρέπει να συμπληρωθεί κατάλληλα η συνάρτηση openmp_primes() και να δοκιμαστούν εναλλακτικοί τρόποι διαμοιρασμού των επαλανήψεων (scheduling).

→ Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος. Για την παραλληλοποίηση προστέθηκε η οδηγία
#pragma omp parallel for shared(i) private(num, divisor, quotient, remainder) reduction(+: count) num_threads(4) schedule(static)

Θέτουμε shared την μεταβλητή i καθώς θέλουμε όλα τα νήματα να γνωρίζουν την τιμή της. Οι μεταβλητές num, divisor, quotient, remainder πρέπει να ορισθούν ως ιδιωτικές για κάθε νήμα. Ακόμα με την εντολή reduction(+: count) συγκεντρώνουμε στην μεταβλητή count τον συνολικό αριθμό των πρώτων αριθμών που βρήκε το πρόγραμμά μας.Χρησιμοποιούμε το schedule(static) όπου διαμοιράζει τις επαναλήψεις, που έχουν χωριστεί σε τμήματα μεγέθους "chunk«,μεταξύ των νημάτων με κυκλικό τρόπο

#pragma omp parallel for shared(i) private(num, divisor, quotient, remainder) reduction(+: count) num_threads(4) schedule(dynamic)

Χρησιμοποιούμε το schedule(dynamic) όπου διαχωρίζει τις επαναλήψεις σε τμήματα μεγέθους "chunk, αλλά το μέγεθος του τμήματος μειώνεται εκθετικά

#pragma omp parallel for shared(i) private(num, divisor, quotient, remainder) reduction(+: count) num_threads(4) schedule(quided)

Χρησιμοποιούμε το schedule(quided) όπου διαχωρίζει τις επαναλήψεις σε τμήματα μεγέθους "chunk αλλά το μέγεθος του τμήματος μειώνεται εκθετικά

Ακόμη χρησιμοποιήθηκε η παρακάτω εντολή για προστασία της κρίσιμης περιοχής αφού θέλουμε να εξασφαλίσουμε αμοιβαίο αποκλεισμό στην εντολή (lastprime = num;) καθώς αλληλοεπιδρούμε με global μεταβλητή:

#pragma omp critical

→Πειράματα

Όλα τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέρω στην αρχή της αναφοράς και συγκεκριμένα κάθε πείραμα εκτελέστηκε 4 φορές και στην συνέχεια υπολογίστηκαν οι μέσοι όροι των χρόνων(δεν έλαβα υπόψη τον χρόνο ανάγνωσης αρχείων).Η χρονομέτρηση εγινε μεσω της συνάρτησης omp_get_wtime();

Σειριακό	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μ.Ο. Εκτέλεσης
STATIC	21.338295	21.303755	21.661350	21.264037	21.3918592
DYNAMIC	14.469651	14.356198	14.365510	14.350689	14.385512
GUIDED	14.711575	15.265012	15.830573	14.897475	15.17615875

	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μ.Ο. Εκτέλεσης
STATIC					
4 Νήματα	8.108114	7.552742	7.564705	7.505177	7.6826845
3 Νήματα	9.783340	9.795223	9.828652	9.825006	9.808.055
2 Νήματα	13.769744	13.860510	13.781722	13.848812	13.815197
1 Νήματα	21.986435	22.436109	21.840854	22.165421	22.10720405
DYNAMIC					
4 Νήματα	5.152471	4.746100	4.858279	4.642083	4.849733
3 Νήματα	6.029040	6.046071	5.459528	5.689987	5.806157
2 Νήματα	8.202084	8.240126	8. 015231	8.240126	8.227445
1 Νήματα	14.581676	14.462463	14.470556	14.456773	14.492864
GUIDED					
4 Νήματα	5.918067	5.669290	5.319107	5.838728	5.686.298
3 Νήματα	6.032468	7.500743	6.178831	6.012536	6.431.145
2 Νήματα	8.256694	8.558141	8.195986	8.439851	8.362.668
1 Νήματα	15.551101	14.465130	14.679406	15.302071	14.999427

→Παρατήρηση

Από την άλλη μεριά για το static παρατήρησα ότι όσο αύξησα τον αριθμό νημάτων όπου ζητήσατε από 1 έως 4 τόσο μειωνόταν ο χρόνος εκτελέσεις. Αντίστοιχα και για το dynamic και το guided ισχύει ότι και για το static ότι όσο παραπάνω νήματα είχα στο πρόγραμμα τόσο λιγότερο χρόνο είχα σαν μεσο ορο στην εκτέλεση. Από τις 3 πολιτικές διαμοιραμού αυτή που έφερε τα καλύτερα αποτελέσματα ήταν η dynamic και συγκεκριμένα όταν χρησιμοποίησα 4 νήματα ο μέσος όρος ήταν κοντά στο 5 ενώ στη guided ήταν κοντά στο 6 και ο χειρότερος από τους 3 ήταν στη static που ήταν κοντά στο 8. Άρα με βάση τα πειράματα τα οποία εκτέλεσαν καλύτερος αλγόριθμος για την άσκηση την παραπάνω ήταν η dynamic

Τα scheduling εδωσαν όπως περιμέναμε καλύτερο χρονο από το σειριακό, όμως παρατηρούμε ότι την καλύτερη επίδοση έδωσε το dynamic όπου είναι αναμενόμενο γιατί η εύρεση πρώτων αριθμών δεν έχει ομαλό φόρτο. Δηλαδή στις πρώτες επαναλήψεις που θα βρει πχ το 2 ή το 5 ή το 7 κάνει λιγότερες πράξεις από μετά που θα βρει κάποιο μεγαλύτερο πρώτο αριθμό. Αυτό επαληθεύεται και απ το σχήμα στην σελίδα 33

Άσκηση 2

→Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθούν τα loop(s) μέσα στη gaussian_blur_omp_loops() και να συμπληρωθεί κατάλληλα η συνάρτηση gaussian_blur_omp_loops() . Ακόμη πρέπει να συμπληρωθεί κατάλληλα η συνάρτηση gaussian_blur_omp_tasks() και να χρησιμοποηθεί tasks για παραλληλοποίηση

→ Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε με βάση τις σημειώσεις του μαθήματος

Υλοποίησα μέσα στη συνάρτηση gaussian_blur_omp_loops():

→1ο loop: Παρόμοια με τον την 1η άσκηση χρησιμοποιώ την ίδια εντολή απλά παραμετροποιημένη για την 2η άσκηση δηλαδή η εντολή γίνεται

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

Δηλαδή για την κοινόχρηστη μεταβλητή i(shared)πρέπει τα νήματα να ξέρουν σε ποια θέση του loop βρίσκονται και με ιδιωτικές(private) ώστε να μην επηρεάζονται το κάθε

νήμα με το άλλο μεταβλητές j,row,col,redSum,greenSum, blueSum, weightSum με αριθμό νημάτων π.χ. 4 .Επαναληπτικά τρέχω το πρόγραμμα για κάθε νήμα από 1 ως 4 από 4 φορές και παίρνω μέσο όρο. Όταν τελειώσω παγώνω την εντολή και ενεργοποιώ την επόμενη εσωτερική.

→2ο loop:

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

Παραπλήσια υλοποίηση απλά τώρα είναι κοινόχρηστη και η j μαζί με την i καθώς πρέπει και αυτή να ξέρει σε ποια θέση του loop βρίσκονται τα νήματα ενώ private είναι row,col,redSum,greenSum, blueSum, weightSum και αριθμός νημάτων εδώ 4 νήματα . Επαναληπτικά τρέχω το πρόγραμμα για κάθε νήμα από 1 ως 4 από 4 φορές και παίρνω μέσο όρο. Όταν τελειώσω παγώνω την εντολή και ενεργοποιώ την επόμενη εσωτερική.

→3ο loop:

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

Παραπλήσια υλοποίηση απλά τώρα είναι κοινόχρηστη και η row μαζί με την i,j καθώς πρέπει και αυτή να ξέρει σε ποια θέση του loop βρίσκονται τα νήματα ενώ private είναιcol,redSum,greenSum, blueSum, weightSum και αριθμός νημάτων εδώ 4 νήματα . Επαναληπτικά τρέχω το πρόγραμμα για κάθε νήμα από 1 ως 4 από 4 φορές και παίρνω μέσο όρο. Όταν τελειώσω παγώνω την εντολή και ενεργοποιώ την επόμενη εσωτερική.

→4ο loop:

#pragma omp parallel for num_threads(4) reduction(+:redSum,greenSum, blueSum, weightSum)

Πρέπει να συγκεντρώσω όλες τις τιμές των νημάτων και να πάρω το ολικό αποτέλεσμα. Αυτό θα το επιτύχω με το reduction όπου θα ενώσει όλες τις τιμές από τα νήματα από την κάθε δουλειά που κάνουν με αριθμό νημάτων εδώ 4. Επαναληπτικά τρέχω το πρόγραμμα για κάθε νήμα από 1 ως 4 από 4 φορές και παίρνω μέσο όρο

Υλοποίησα μέσα στη συνάρτηση gaussian_blur_omp_loops():

Δεδομένου ότι μια γραμμή είναι ένα task από εκφώνηση .

#pragma omp parallel num_threads(4)

Αριθμός νημάτων 4 και επαναληπτικά τρέχω το πρόγραμμα για κάθε νήμα από 1 ως 4 από 4 φορές και παίρνω μέσο όρο

#pragma omp single

Όποιο νήμα συναντήσει το single να μπει κάθε φορά ένα νήμα στο loop

Πριν την for που αντιστοιχεί στην row βάζω την εντολή

#pragmaomp task firstprivate(i,j,row,col,weightSum,redSum,greenSum,blueSum)

Χρησιμοποιώ firstprivate γιατί θα πρέπει να είναι ιδιωτικές μεταβλητές για κάθε νήμα και με αρχικοποίηση συμφώνα με τις πρωταρχικές τιμές των μεταβλητών

#pragma omp taskwait

Barrier που μπλοκάρει τα tasks όπου έχουν δημιουργηθεί εώς ότου ολοκληρωθεί όλο το σύνολο των tasks

→Τι προστέθηκε επιπλέον

Ακόμη προστέθηκαν μεταβλητές που αρχικοποιηθηκαν στην αρχή του προγράμματος(serial_start=0, serial_end=0, parallel_start=0, parallel_end=0, tasks_start=0, tasks_end=0;) και χρησιμοποιούνται για τη χρονομέτρηση μέσω της omp_get_wtime στο σειριακό και omp πρόγραμμα(tasks_start=omp_get_wtime(); tasks_end=omp_get_wtime();

Αντίστοιχα για parallel κα task.Για εκτύπωση των χρόνων:

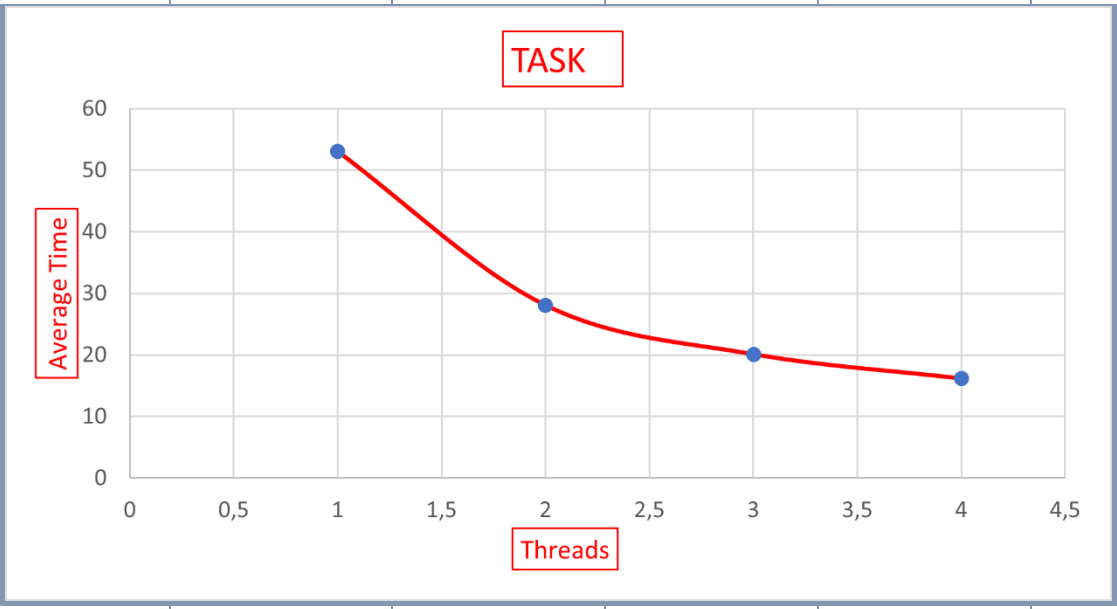
```
printf("Total execution time (sequential): %lf\n", (serial_end-serial_start));//υπολογισμός σειριακού χρόνου
        printf("Total execution time (omp loops): %lf\n", (parallel_end-parallel_start)); //υπολογισμός omp χρόνου
        printf("Total execution time (omp tasks): %lf\n", (tasks_end-tasks_start)); //υπολογισμός task χρόνου
```

→Πειράματα

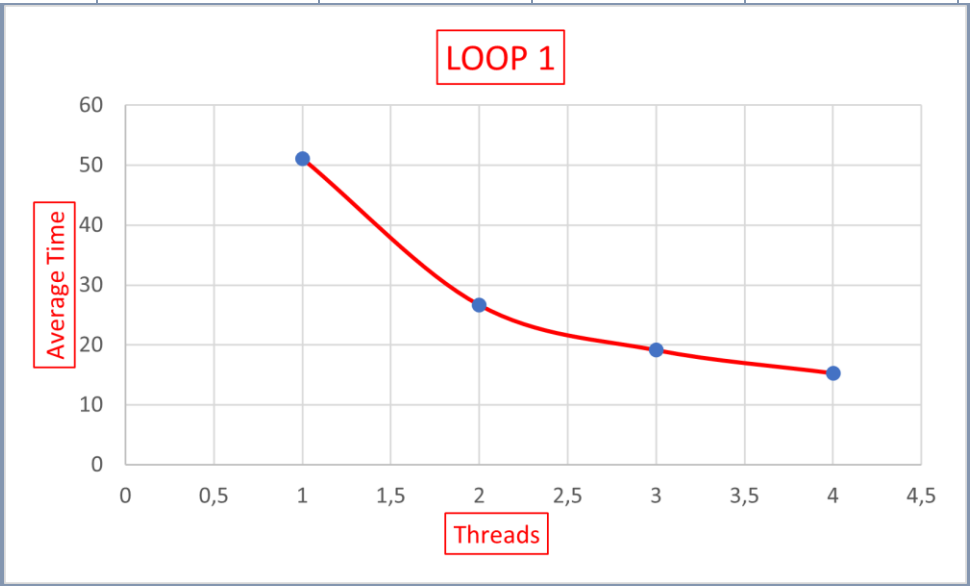
Χρησιμοποίησα την εντολή gcc gaussian-blur.c -fopenmp -lm για να κάνει compile το πρόγραμμα και έπειτα την ./a.out(radius)(image) για να τρέξει. Συγκεκριμένα στο radius είχα την τιμή 8 και στο image την εικόνα 1500.bmp όπως ζητήσατε στην εκφώνηση. Επαναληπτικά τρέχω το πρόγραμμα για κάθε νήμα από 1 ως 4 από 4 φορές και παίρνω μέσο όρο

	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μ.Ο. Εκτέλεσης
SEQUENTIAL 1					
	50.32161	50.924687	50.763690	50.452043	50.6155075
TASKS					
4 Νήματα	16.113496	16.232527	16.146003	16.201390	16.173354
3 Νήματα	20.141335	20.097802	20.065011	20.176040	20.120047

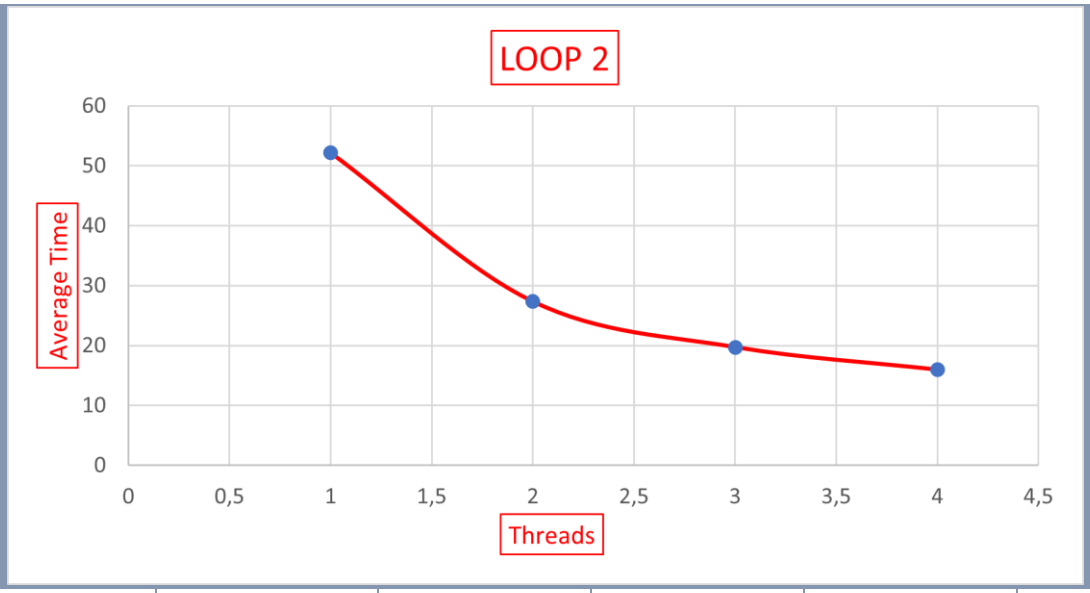
2 Νήματα	28.021667	27.989540	28.058572	28.242660	28.078109
1 Νήματα	53.085721	53.146756	53.077588	53.020215	53.082570



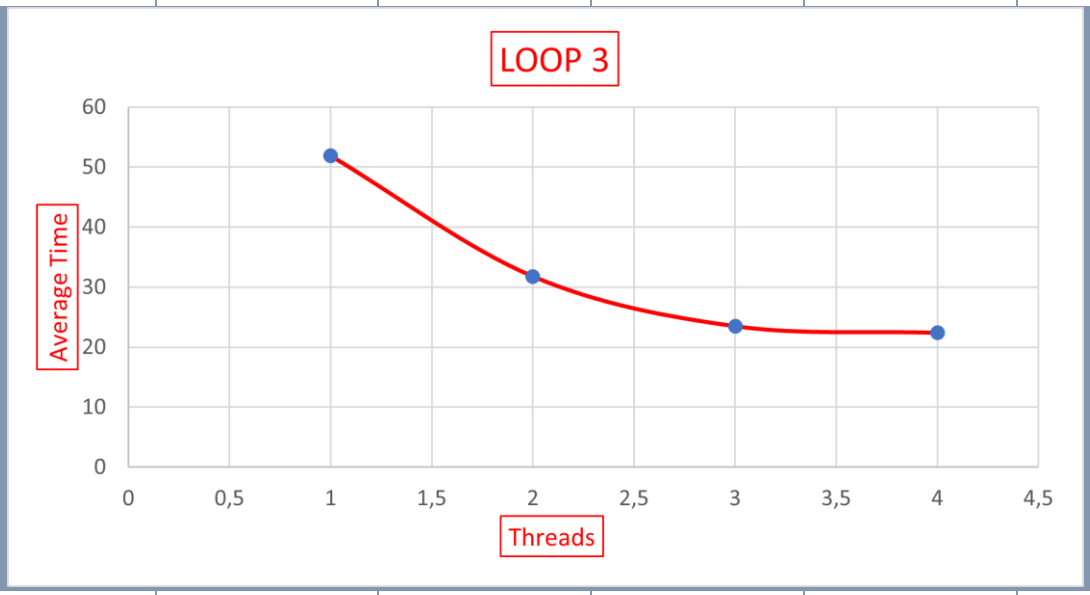
LOOP 1					
4 Νήματα	15.196088	15.241643	15.255384	15.302136	15.248812
3 Νήματα	19.062696	19.174074	19.151668	19.037925	19.106590
2 Νήματα	26.629156	26.545237	26.656632	26.701174	26.633049
1 Νήματα	51.119686	51.084226	51.231659	51.054042	51.122403



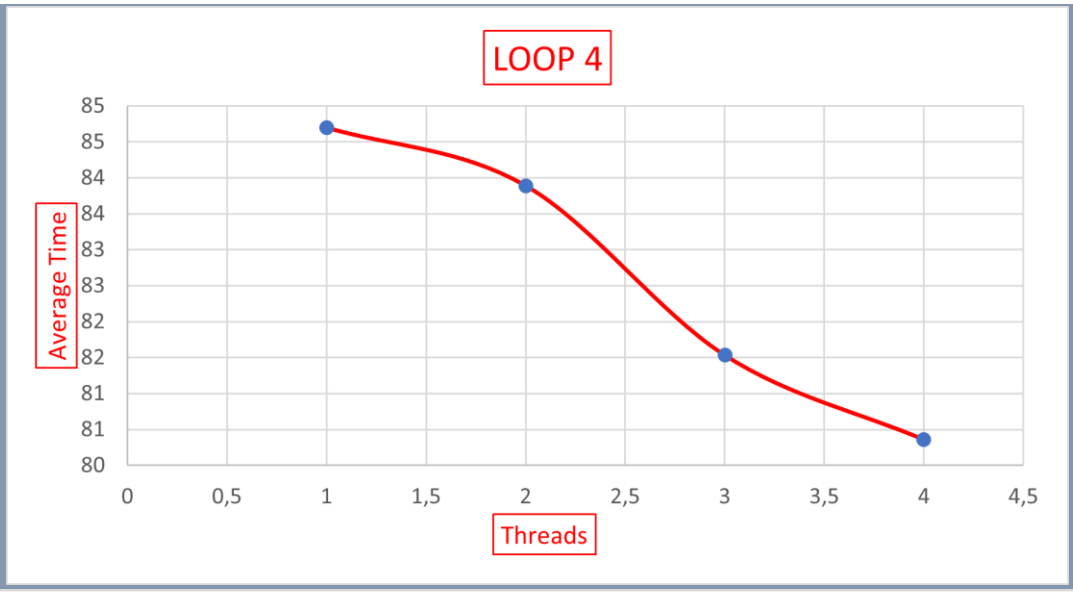
LOOP 2					
4 Νήματα	16.060008	15.943543	15.837482	16.002450	15.960.871
3 Νήματα	19.781785	19.670557	19.827036	19.617068	19.724.112
2 Νήματα	26.938903	27.329105	27.400745	27.640071	27.327.206
1 Νήματα	52.449828	51.80089	52.366489	51.982728	52.149.984



LOOP 3					
4 Νήματα	22.688524	22.536474	22.344805	21.890973	22.365.194
3 Νήματα	23.758151	23.289705	23.132192	23.664272	23.461.080
2 Νήματα	31.940937	32.039257	31.447891	31.675849	31.775.984
1 Νήματα	52.281223	51.926549	52.296697	51.172419	51.919.222



LOOP 4					
4 Νήματα	80.314103	80.404441	80. 313062	79. 672581	80.359.272
3 Νήματα	81.239258	81.888163	81.239461	81.763450	81.532.583
2 Νήματα	83.940937	83.819840	83.852139	83.940937	83.888.463
1 Νήματα	85.131028	84.436928	84.886406	84.327408	84.695.443



→**Παρατήρηση**

Παρατηρούμε ότι το καλύτερο χρόνο δίνει η παραλληλοποιήση του loop1 βρόγχου ανάμεσα στους χρόνους όλων των βρόχων και χειρότερο χρόνο ο loop4.

Ακόμη παρατηρούμε ότι μεταξύ του loop1 και του task καλύτερο χρόνο βλέπουμε στο loop1.. Άρα στο συγκεκριμένο πρόβλημα η χρήση task δεν μας προσφέρει κατι περισσότερο από την παραλληλοποίηση του κατάλληλου βρόγχου δηλαδή του loop1.

Άσκηση 3

→Τι είναι η taskloop?

Η taskloop χρησιμοποιείται για τον παραλληλισμό βρόχων με δυναμικό χρονοπρογραμματισμό εργασιών στις διαθέσιμες μονάδες επεξεργασίας. Είναι παρόμοια με την οδηγία "task" του OpenMP, αλλά παρέχει μεγαλύτερη ευελιξία όσον αφορά τον προγραμματισμό εργασιών. Η οδηγία taskloop δημιουργεί ένα σύνολο εργασιών που μπορούν να εκτελεστούν παράλληλα και τις αναθέτει δυναμικά στις διαθέσιμες μονάδες νήματος. Αυτό έχει ως αποτέλεσμα καλύτερα αποτελέσματα σε μικρότερο χρόνο
Πρέπει να καθορίστει ο βρόχος που θα παραλληλιστεί με την εντολή "#pragma omp taskloop".

Το συντακτικό του ορίζεται ως εξής:

```
#pragma omp taskloop [clause[[,] clause] ...]
 for (init-expression; test-condition; incr-expression)
statement
```

Μέσω της taskloop κάθε επανάληψη ενός loop αντιστοιχεί σε μια εργασία , καθεμία από τις οποίες μπορεί να εκτελεστεί ανεξάρτητα. Το σύστημα εκτέλεσης OpenMP είναι υπεύθυνο για τον προγραμματισμό των εργασιών

Στην OpenMP στην εντολή #pragma omp parallel", οι clauses μπορούν να χρησιμοποιηθούν για να καθορίσουν ποιες μεταβλητές χρησιμοποιούνται από κοινού(global) από τα νήματα, ποιες μεταβλητές είναι ιδιωτικές για κάθε νήμα και πόσα νήματα θα χρησιμοποιηθούν.Ομοίως, στην εντολή "#pragma omp taskloop", οι clauses μπορούν να χρησιμοποιηθούν για να καθορίσουν τον αριθμό των εργασιών, τον ελάχιστο αριθμό επαναλήψεων ανά εργασία και τις κοινές και ιδιωτικές μεταβλητές.

Παρακάτω παρατίθενται μερικά clauses της taskloop όπου τροποποιούν ουσιαστικά μια εντολή παραλληλοποιησης για μεγαλύτερη ευελιξία στην χρήση :

num_tasks(nt)→ Καθορίζει τον αριθμό των δουλειών στις οποίες χωρίζονται οι επαναλήψεις του loop,π.χ. η #pragma omp taskloop num_tasks(2) δημιουργεί 2 task που μπορούν να εκτελούν παράλληλα τις επαναλήψεις του loop.

grainsize(g)→ Δείχνει τον ελάχιστο αριθμό επαναλήψεων ανά δουλειά, π.χ η #pragma omp taskloop num_tasks(2) grainsize(10) δημιουργεί 2 task, καθεμία με >=10

επαναλήψεις, δηλαδή μας διασφαλίζει ότι θα γίνουν τουλάχιστον οι g επαναλήψεις που ορίσαμε και πάνω .

shared(list): Αφήνει τις κοινές μεταβλητές στο loop να έχουν πρόσβαση σε όλες τις δουλειές, π.χ. #pragma omp taskloop num_tasks(2) shared(i, j, s) καθορίζει ότι οι μεταβλητές i, j και s είναι κοινές για όλες τις δουλειές.

→Ένα μικρό παράδειγμα εφαρμογής της taskloop:

```
#include <stdio.h>
#include <omp.h>

int main()
{
int i, sum = 0;

#pragma omp parallel
#pragma omp taskloop private(i)
reduction(+:sum)
for (i = 0; i < 100; i++) {
sum += i;
}

printf("sum = %d\n", sum);
return 0;
}
```

Είναι ένα απλό loop που αθροίζει τους αριθμούς από το 0 έως το 99. Χρησιμοποιούμε την οδηγία taskloop για να παραλληλοποιήσουμε το loop, με κάθε επανάληψη του loop να εκτελείται ως ξεχωριστή εργασία. Καθορίζουμε επίσης το reduction ως σύνολο αθροίσματος όλων των εργασιών.

Πηγές: https://www.openmp.org/spec-html/5.0/openmpsu47.html
https://www.ibm.com/docs/de/xl-c-and-cpp-linux/16.1.0?topic=parallelization-pragma-omp-taskloop
https://blog.rwth-aachen.de/itc-events/files/2021/02/09-openmp-CT-taskloop.pdf