ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

ΗΡΥ 418 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΠΑΡΑΛΛΗΛΩΝ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2017-18

Άσκηση 1 : Χρήση OpenMP και pthreads

Εαρινό εξάμηνο 2017-18

ΟΜΑΔΑ: LAB41835573

ΣΦΥΡΗΣ ΝΕΚΤΑΡΙΟΣ (2013030058)

ΖΑΧΑΡΙΑ ΝΕΟΦΥΤΟΣ (2014030210)

ΠΕΡΙΓΡΑΦΗ

Στην άσκηση αυτή μας ζητήθηκε να χρησιμοποιήσουμε το OpenMP Application Protocol Interface (API) και ένα υποσύνολο των συναρτήσεων του POSIX threads standard (γνωστό και ως pthreads) για να υπολογίσουμε τις αποστάσεις Hamming μεταξύ δύο συνόλων συμβολοσειρών.

ΣΕΙΡΙΑΚΟΣ ΚΩΔΙΚΑΣ

Αρχικά γράψαμε τον αρχικό κώδικα σε C (omp_0.c), ο οποίος δέχεται τις εξής εισόδους:

- m : αριθμός των συμβολοσειρών του συνόλου A
- n : αριθμός των συμβολοσειρών του συνόλου B
- l : μέγεθος κάθε συμβολοσειράς

Η λειτουργία του κώδικα αυτού είναι να δημιουργεί και να αρχικοποιεί τις m και n συμβολοσειρές με τυχαίους χαρακτήρες και ακολούθως να υπολογίζει όλες τις αποστάσεις Hamming αποθηκεύοντας τα αποτελέσματα σε ένα πίνακα m*n (hamming_array). Στο τέλος θα τυπώνει στην οθόνη το συνολικό άθροισμα όλων των m*n αποστάσεων (sum_hamming).

Έτσι , για δίαφορες τιμές του m , n και l υπολογίσαμε τον χρόνο εκτέλεσης του σειριακού κώδικα.

ΧΡΗΣΗ OPENMP

Μελετώντας τον αρχικό κώδικα βρήκαμε σημεία όπου μπορούμε να εκμεταλευτούμε παραλληλισμό με διαφορετικό granularity, και κατά συνέπεια διαφορετικό computation-to-communication ratio. Έτσι καταλήξαμε στο να έχουμε 3 διαφορετικά task, όπου με την βοήθεια του OpenMP επιταγχύναμε τον υπολογισμό του hamming distance. Για την openmp έπρεπε να γίνει η χρήση της βιβλιοθήκης <omp.h>.

TASK1

Υλοποιήσαμε το task1 έτσι ώστε σε ένα ζευγάρι απο συμβολοσειρές (ένα απο σύνολο A και ένα απο B), υπολογίζουν το hamming distance όσα threads έχουμε στην διάθεση μας (π.χ. l=100 και αριμός το threads=4 τότε κάθε thread αναλαμβάνει να συγκρίνει 25 χαρακτήρες απο τη συμβολοσειρά του συνολού A και 25 χαρακτήρες απο τη συμβολοσειρά του B). Αυτό γίνεται για όλα τα ζευγάρια των δυο συνόλων.

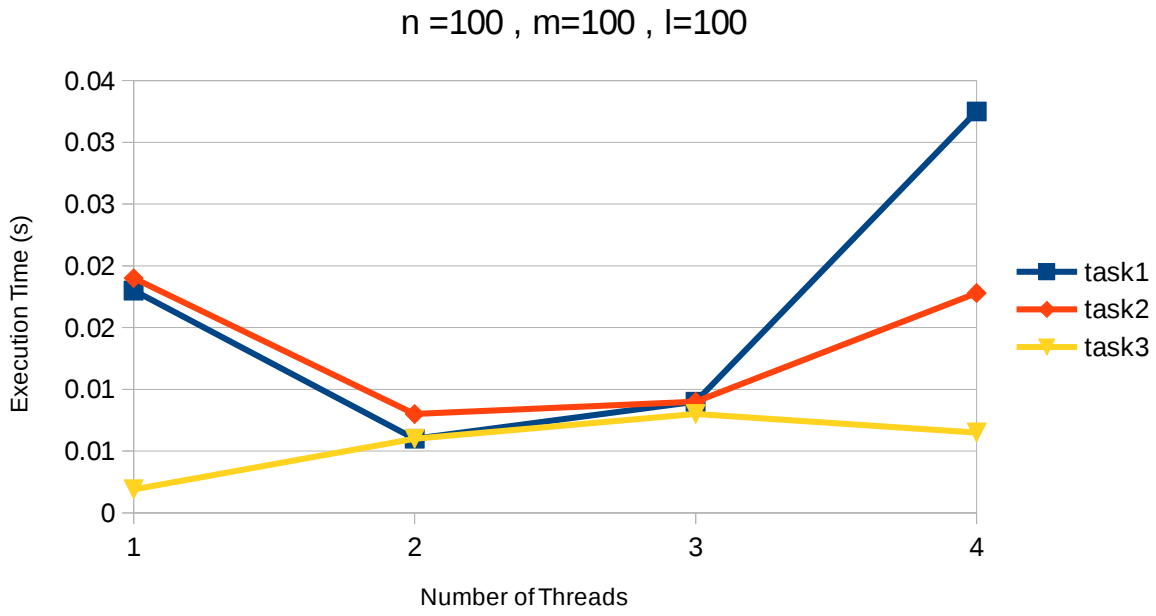
TASK2

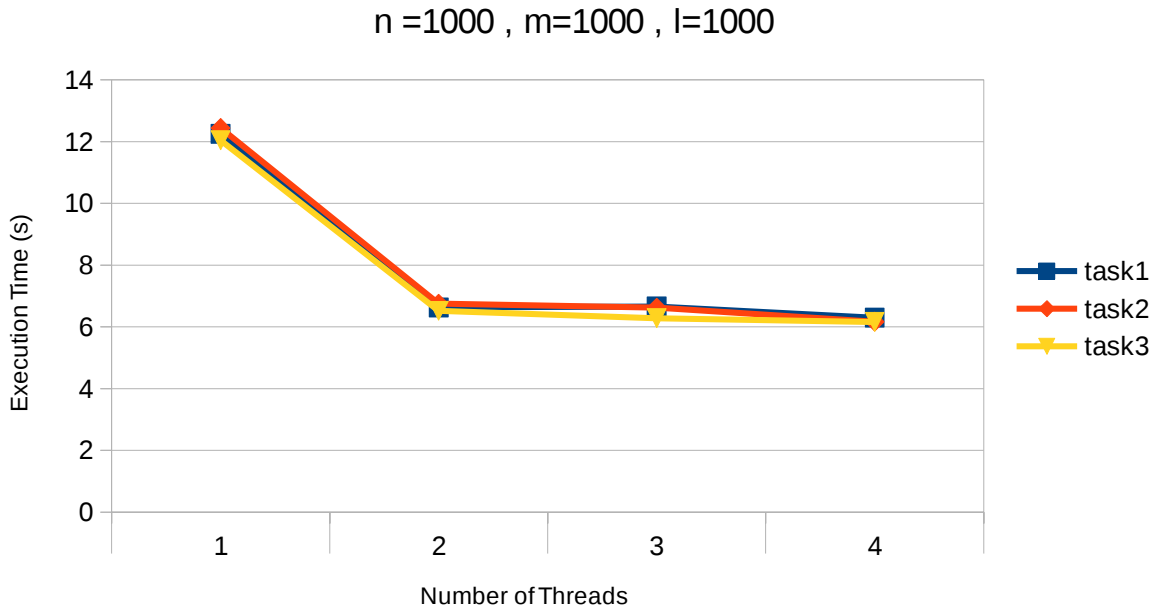
Ακολούθως το task2 τροποποιήθηκε έτσι ώστε σε ένα ζευγάρι απο συμβολοσειρές (ένα απο σύνολο A και ένα απο B), το hamming distance να το υπολογίζει ένα thread .(π.χ. l=100 και αριμός το threads=4 τότε κάθε thread αναλαμβάνει να συγκρίνει 100 χαρακτήρες απο τη συμβολοσειρά του συνολού A και 100 χαρακτήρες απο τη συμβολοσειρά του B). Αυτό γίνεται για όλα τα ζευγάρια των δυο συνόλων.

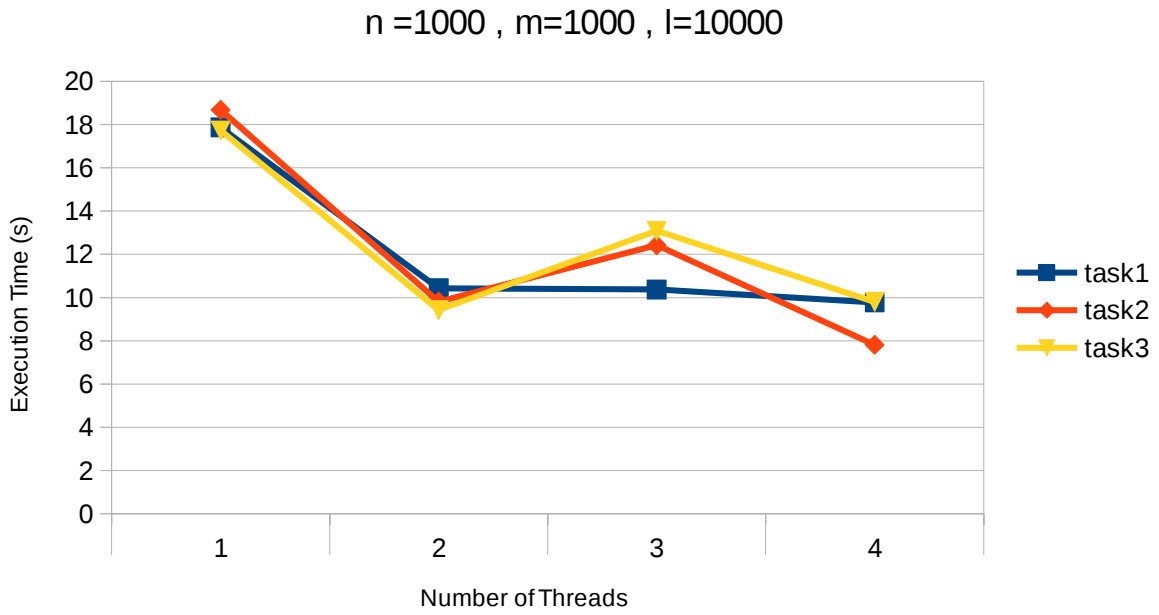
TASK3

Στη συνέχεια το task3 τροποποιήθηκε και αυτό έτσι ώστε , ένα thread να συγκρίνει όλες τις συμβολοσειρές του συνόλου A με μια συμβολοσειρά του συνόλου B .(π.χ. m=100 και n=1000, l=100 και αριμός το threads=4 τότε κάθε thread αναλαμβάνει να συγκρίνει 100 συμβολοσειρές(των 100 χαρακτήρων) με μια συμβολοσειρά του συνολού B. Αυτό γίνεται για όλους τους συνδυασμούς.

Πιο κάτω βλέπουμε τους χρόνους εκτέλεσης για διάφορα m , n , l του σειριακού κώδικα (omp_0.c), του task1 (omp_1.c), task2 (omp_2.c) και του task3 (omp_3.c).







Για την περίπτωση όπου n , m, l = 100 και number of threads είναι 2 έχουμε speedup 135% για το 1ο task, 161% για το δεύτερο και 147% για το τρίτο.
Για την περίπτωση όπου n , m, l = 1000 και number of threads είναι 3 έχουμε speedup 160% για το 1ο task, 159% για το δεύτερο και 159% για το τρίτο.

Για την περίπτωση όπου n = 1000, m = 1000, l = 10000 και number of threads είναι 4 έχουμε speedup 1197% για το 1ο task, 1190% για το δεύτερο και 1228% για το τρίτο.

Παρατηρούμε αρχικά ότι για μικρά m,n,l δεν έχει και μεγάλη διαφορά στο χρόνο εκτέλεσης αλλά όταν αυξηθούν σημαντικά τότε παρατηρούμε μεγάλη διαφορά στην βελτισοποιήσση του χρόνου εκτέλεσης.

Για τον υπολογισμό των χρόνων χρησιμοποιήθηκε η συνάρτηση gettime(), και για να την καλέσουμε κάναμε import την βιβλιοθήκη <sys/time.h>.
Για να δημιουργήσουμε τα εκτελέσιμα : **gcc -fopenmp name.c -o execname**

ΧΡΗΣΗ PTHREADS

Στη συνέχεια αντικαταστήσαμε τον OpenMP κώδικα με pthreads (με κάποιες προσαρμογές ώστε να συγχρονίζονται τα threads) και επαναλάβαμε τις ίδιες μετρήσεις.

Για τα pthreads έπρεπε να γίνει η χρήση της βιβλιοθήκης <pthread.h>.

TASK1

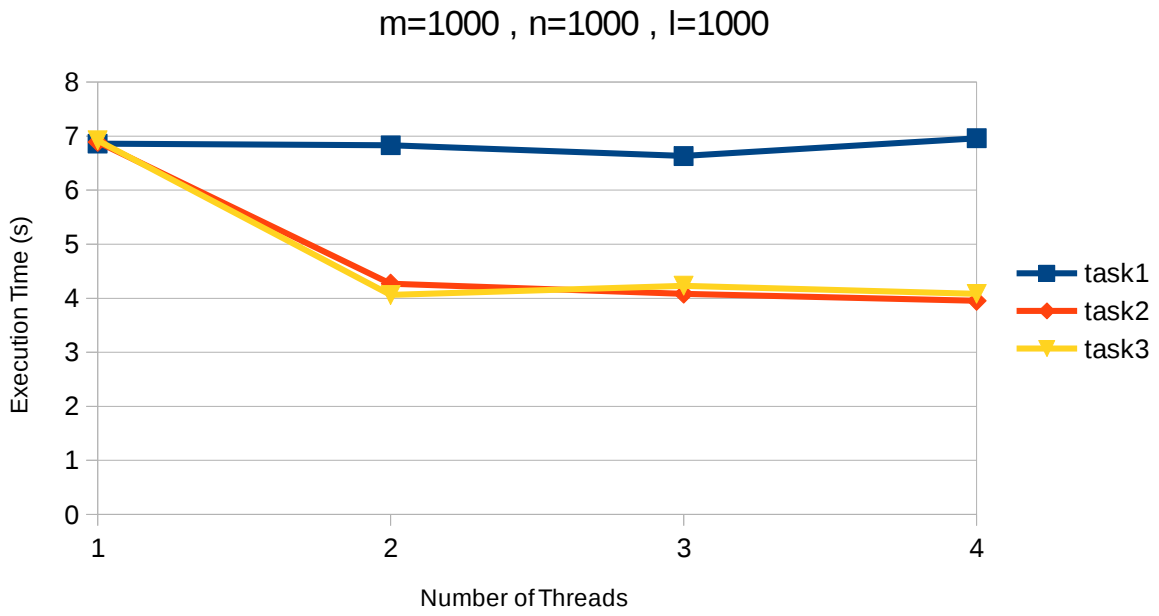
Όπως και πριν για αυτό το task, κάθε thread το αντιστοιχίσαμε για την εκτέλεση ενός μέρους μιας συμβολοσειράς. Για να το πετύχουμε αυτό ορίσαμε ένα start και ένα end σημείο, τα οποία όριζαν τον αριθμό των συμβόλων για τα οποία το κάθε thread θα ήταν υπεύθυνο και έτσι γίνονταν οι ανάλογες συγκρίσεις. Τα απαραίτητα δεδομένα, όπως ο αριθμός μεγέθους των συνόλων και της συμβολοσειράς κρατούνται σε μία δομή – struct και έτσι μπορούμε να έχουμε πρόσβαση σε αυτά από οποιδήποτε μέρος του κώδικα. Επίσης για να εξασφαλίσουμε ότι κάποιο thread δεν θα πανογράψει στην μεταβλητή που κρατάμε για το άθροισμα του hamming distance, χρησιμοποιούμε mutex_lock και unlock στο κατάλληλο σημείο. Από εκεί και πέρα κάθε φορά που δημιουργούμε ένα thread από τα όσα μας ζητήθηκαν, εκείνο πηγαίνει και εκτελεί το κομμάτι του task που πρέπει να εκτελέσει.

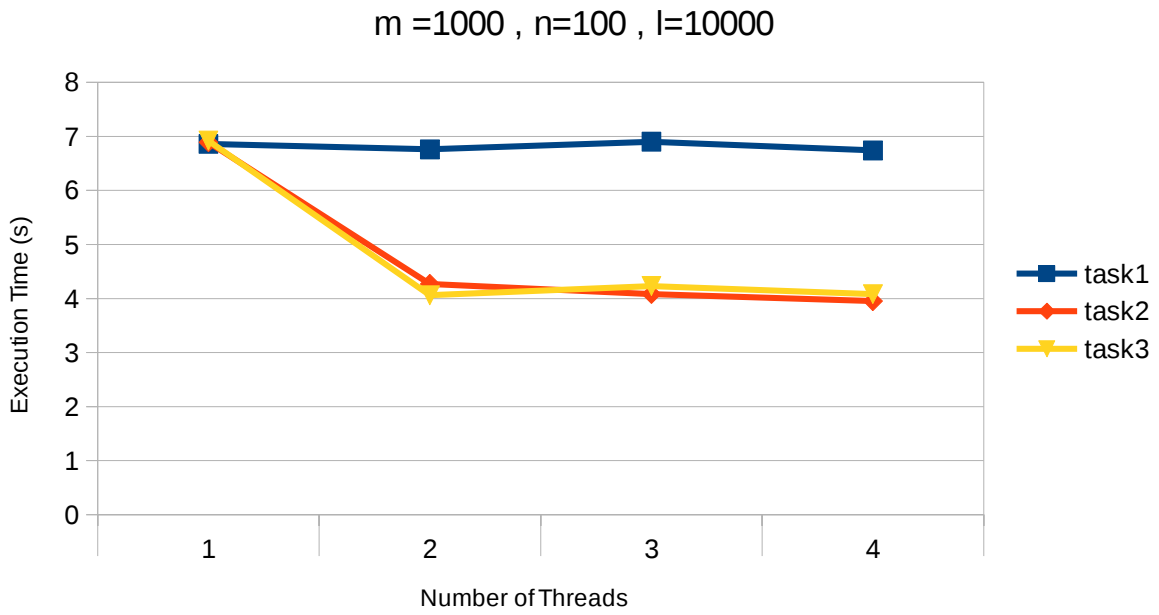
TASK2

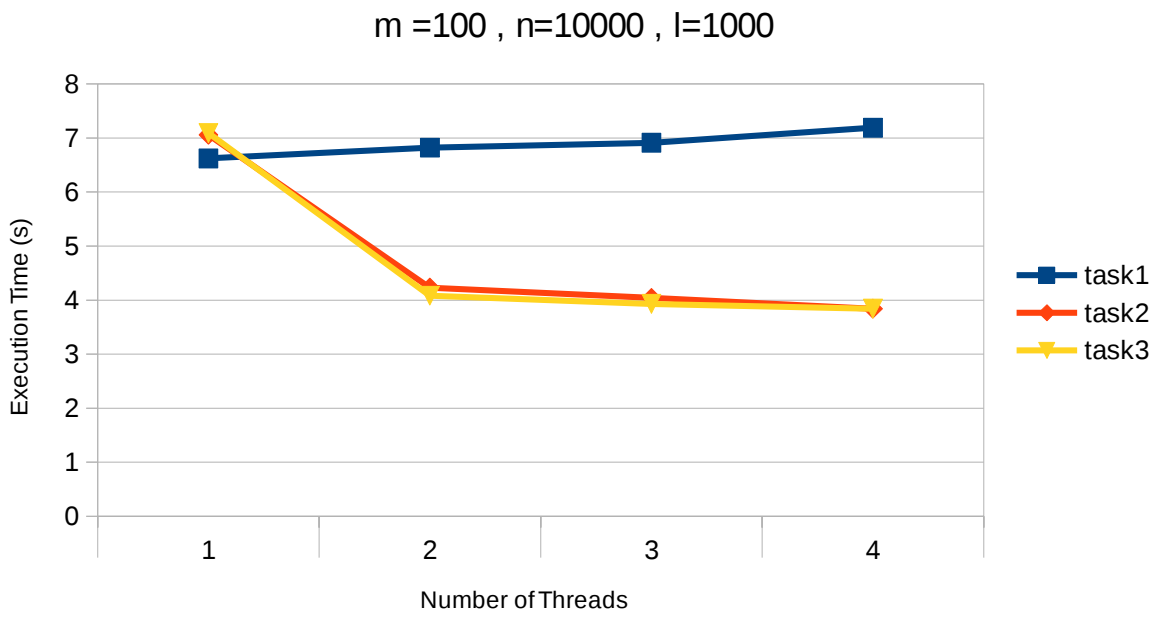
Αυτό που αλλάζει με το προηγούμενο task, είναι ότι κάθε thread τώρα το αντιστοιχούμε στη σύγκριση ακριβώς μίας συμβολοσειράς με μία άλλη. Για να το πετύχουμε αυτό ορίσαμε ένα start και ένα end σημείο, τα οποία όριζαν τον αριθμό των συμβολοσειρών του δεύτερου πίνακα για τα οποία το κάθε thread θα ήταν υπεύθυνο να κάνει τις συγκρίσεις με τις συμβολοσειρές του πρώτου.

TASK3

Για αυτό το task κάθε thread ενός συνόλου, το αντιστοιχίσαμε σε όλες τις συμβολοσειρές του άλλου συνόλου. Για να το πετύχουμε αυτό ορίσαμε ένα start και ένα end σημείο, τα οποία όριζαν τον αριθμό των συμβολοσειρών του πρώτου συνόλου για τα οποία το κάθε thread θα ήταν υπεύθυνο να κάνει τις συγκρίσεις με τις συμβολοσειρές του δευτέρου.







Για την περίπτωση όπου n , m, l = 1000 και number of threads είναι 2 έχουμε speedup 132% για το 1ο task, 223% για το δεύτερο και 213% για το τρίτο.
Για την περίπτωση όπου n = 1000, m = 100, l = 10000 και number of threads είναι 2 έχουμε speedup 135% για το 1ο task, 252% για το δεύτερο και 243% για το τρίτο.
Για την περίπτωση όπου n = 100, m = 10000, l = 1000 και number of threads είναι 2 έχουμε speedup 123% για το 1ο task, 245% για το δεύτερο και 238% για το τρίτο.

Παρατηρούμε γενικά ότι ο χρόνος εκτέλεσης των task σε σχέση με τον σειριακό κώδικα είναι πολύ μικρότερος και προφανώς όσο αυξάνουμε τα μεγέθη των συνόλων αυτή η διαφορά γίνεται όλο και πιο εμφανής. Μεγάλη σημασία έχει ότι σε αντίθεση με την χρήση της openmp, στα pthreads καθορίζουμε εμείς τον τρόπο λειτουργίας και συμπεριφοράς του κάθε thread το οποίο μπορούμε να εκμεταλλευτούμε και σε μεγάλου μεγέθους και πολυπλοκοτητας προβλήματα για ακόμα καλύτερο speedup.

Για να δημιουργήσουμε τα εκτελέσιμα : **gcc -pthread name.c -o execname**

Συνολικά για να μπορέσουμε να τρέξουμε τα αρχεία για openmp και pthreads εκτελούμε σε κάθε έναν φάκελο του πρότζεκτ που τα περιέχει, το εκτελέσιμο του αρχείου **bash** (πχ ./bash).