Πολυτεχνείο Κρήτης

ΤΜΉΜΑ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΏΝ ΚΑΙ ΜΗΧΑΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ

Αρχιτεκτονική Παράλληλων και Κατανεμημένων Υπολογιστών (**ΗΡΥ418)**

Αναφορά 1ης Άσκησης

OPENMP & POSIX THREADS

Ον/μο	**Α.Μ**
Ζησκας Χρήστος	**2014030191**
Μποκαλίδης Αναστασιος	**2014030069**

IMPLEMENTATIONS

SERIAL

Το πρόγραμμα λειτουργεί για συγκεκριμένο αριθμό ορισμάτων από την main . Αριθμός ορισμάτων πέρα από το προκαθορισμένο οδηγεί σε αποτυχία εκτέλεσης. Τα ορίσματα αναγνωρίζονται ανεξαρτήτου θέσης καταχώρησης. Αρχικοποιούνται οι δείκτες απόδοσης και οι χρόνοι.

Οι δείκτες των string αρχικοποιούνται δυναμικά στην main και λαμβάνονται ως παράμετροι στις συναρτήσεις. Ο πίνακας με τα score δημιουργείται δυναμικά στη main και το μέγεθος που δεσμεύεται είναι ανάλογο του γινομένου (m+1)*(n+1) .

όπου

m,n : ο συνολικός αριθμός των χαρακτήρων που λαμβάνει κάθε string μετά την ανάγνωση του.

Οι συναρτήσεις που διαβάζουν από το dataset διευθύνουν τους pointer των string ώστε να δείχνουν στο ζητούμενο set χαρακτήρων. Η ανάγνωση του κάθε χαρακτήρα από το dataset γίνεται σειριακά απορρίπτοντας χαρακτήρες newline tab και γενικότερα μη αποδεκτούς χαρακτήρες για τη ζητούμενη άσκηση. Η τακτική ανάγνωση είναι αποτελεσματική για οποιοδήποτε τύπο αρχείου dataset

Η δυναμική δέσμευση μνήμης γίνεται τόσο όσο υποδεικνύουν τα size του αρχείου. Τα malloc γίνονται για να είναι εύκολα διαχειρίσιμη η μνήμη αλλά και διότι σε ορισμένα αρχείο το μέγεθος των strings είναι τόσο μεγάλο ώστε να υπερφορτώνεται η στοίβα. Θα μπορούσε επιπλέον να χρησιμοποιηθεί η εντολή setrlimit() για να λυθεί εξίσου το πρόβλημα. Πιο αποδοτική η λύση της δυναμικής κατάτμησης.

Τα στοιχεία της απόδοσης του προγράμματος παρουσιάζονται στο τερματικό κατά το πέρας του προγράμματος. Χρονομετρείται κάθε κομμάτι που αφορά τα στοιχεία πριν και μετά το τέλος κάθε βήματος προς απόδοση – (execution ,calculation ,traceback κλπ) .

Η υλοποίηση ενδείκνυται για οποιοδήποτε αρχείο dataset

Functions:

terminal1,terminal2: Εύρεση της παραμέτρου που έχει εισαχθεί από το terminal ώστε να ληφθεί η μεταβλητή που θα χρησιμοποιηθεί για την υλοποίηση του πίνακα score και των περαιτέρω διαδικασιών που αφορούν το αρχείο εξόδου. Η αντιστοίχιση επιτυγχάνεται με την ταυτοποίηση αλφαριθμητικών. Η terminal1 εξάγει τις αριθμητικές μεταβλητές (match,mismatch,gap).
Η terminal2 εξάγει τις αλφαριθμητικές μεταβλητές(path,id).
Για εσφαλμένο αριθμό ορισμάτων η για εσφαλμένα στοιχεία ,το πρόγραμμα τερματίζει.
Δέχεται ως ορίσματα τις μεταβλητές τις main. Αναγνωρίζουν τα στοιχεία ανεξάρτητα τις θέσεις που λαμβάνονται από την main

ReadVariables: Διαβάζει τα στοιχεία που αφορούν τα μεγέθη των string που θα μελετηθούν στο αρχείο(pair size, Qmin, Qmax,Dmax). Δέχεται ως ορίσματα τους δείκτες των μεταβλητών που τα απαιτούμενα μεγέθη των string. Καλείται μια φορά κατά την εκτέλεση του προγράμματος για να διαβάσει τις μεταβλητές του dataset

readQ,readD: Χρησιμοποιείται κάθε συνάρτηση για ανάγνωση του stirng Q και D αντίστοιχα. Οι συναρτήσεις λειτουργούν ως εξής:
Loop ώστε να αναγνωσθεί το σύνολο των χαρακτήρων που υποδεικνύει το Qmax σειριακά. Χαρακτήρες newline και tab αγνοούνται χωρίς να λαμβάνονται υπόψη από το δείκτη που χρησιμοποιείται ως διάβασμα.
Ο δείκτης αυξάνεται μόνο όταν θεωρεί αποδεκτό χαρακτήρα (αριθμητικό ,αλφαριθμητικό). Δέχεται ως ορίσματα δείκτη Q και D αντίστοιχα για τις συναρτήσεις, καθώς και τον αριθμό των χαρακτήρων που προτείνει το αρχείο για διάβασμα.

calculation: Δέχεται ως ορίσματα τους δείκτες των string αλλά και το δείκτη για το score καθώς και τις θέσεις των string προς επεξεργασία. .Ο δείκτης για το score αρχικοποιείται στη main και τα κελιά του, τροποποιούνται στη συνάρτηση υπολογισμού. Η επεξεργασία των κελιών θα γίνει ως εξης : Κάθε χαρακτήρας του string Q αναλύεται με κάθε χαρακτήρα του string D . Ο πίνακας score γεμίζει σύμφωνα με τον αλγόριθμο Smith-waterman.

Ο αριθμός των κλήσεων της συνάρτησης ορίζεται από το γινόμενο των στοιχείων(Δυο for iterations , το ένα *εμφωλευμένο* στο άλλο). Η τιμές των κελιών της πρώτης γραμμής και της πρώτης στήλης είναι αρχικοποιημένες με μηδενικά. Οπότε ο υπολογισμός ξεκινάει από το κελί της πρώτης γραμμής και της πρώτης στήλης και συνεχίζει στα επόμενα . Για τον υπολογισμό χρησιμοποιούνται τα άνω γειτονικά κελιά . Σε κάθε κλήση πραγματοποιείται έλεγχος για αναγνώριση της μέγιστης τιμή που υπάρχει στον πίνακα . Αν η τιμή ,που εξάγεται από το αποτέλεσμα του υπολογισμού, είναι μεγαλύτερη από την μέγιστη τιμή που έχει διατηρηθεί από προηγούμενες κλήσεις , τότε η προσφάτως καταχωρημένη τιμή γίνεται η μέγιστη τιμή που διατηρεί ο πίνακας.
Με το πέρας της διαδικασίας του υπολογισμό του score συνεχίζει η διαδικασία του traceback. Σαρώνονται τα στοιχεία του πίνακα και για τα κελιά που ισούνται με την μέγιστη τιμή καλείται η συνάρτηση οπισθοδρόμησης

Traceback: Όμοια διαδικασία με τον υπολογισμό του score ως προς την διαχείριση των κελιών . Για το εξεταζόμενο κελί , αναγνωρίζονται οι χαρακτήρες που αντιστοιχούν στα string . Αν είναι όμοια ακολουθείται το διαγώνιο κελί
Αλλιώς η διαδρομή συνεχίζει ελέγχοντας το κελί με τη μέγιστη τιμή . Για ισότητα μεταξύ των κελιών αλλά για διαφορετικούς χαρακτήρες ακολουθείται το κελί που βρίσκεται στην κατακόρυφο. Η διαδικασία συνεχίζει μέχρι ο αλγόριθμος να καταλήξει σε κελί με μηδενική τιμή. Σε αυτό το σημείο γίνεται και η εγγραφή των στοιχείων που αφορούν το ζευγάρι – Match, Start,Stop καθώς και τα string που προέκυψαν ως αποτέλεσμα του traceback- στο αρχείο.
Το μήκος των string είναι ίδιο και για τα δυο string χαρακτήρων.

OPEN MP

Η λογική που έχει τηρηθεί αφορά την βελτίωση της απόδοσης που σχετίζεται με το σειριακό πρόγραμμα . Για την υλοποίηση σε παραλληλία OpenMP λαμβάνεται υπόψη η έννοια του parallelism. Parallelism αφορά την πραγματοποίηση ενεργών tasks την ίδια στιγμή .Το σειριακό πρόγραμμα τροποποιείται για το σκοπό αυτό. Η κατασκευή τους υλοποιείται με directive #pragma omp parallel για multiple threads . Απαιτείται ως επιπλέον είσοδος και ο αριθμός των threads. Στον σειριακό κώδικα προστίθενται τα κατάλληλα directives ώστε να μετατραπεί το κομμάτι υπολογισμού του score matrix σε παράλληλο. Έχει προηγηθεί *παραλληλοποίηση* και του traceback αυτού του τμήματος αλλά παρατηρήθηκε ότι η απόδοση του δεν βελτιώνεται σε σταθερή συχνότητα . Επιπλέον ένα μέρος του αρχείου γράφεται με στην συνθήκη του βήματος traceback ώστε να εξοικονομείται ανεκμετάλλευτος χρόνος.
Οπότε δεν ενδείκνυται η *παραλληλοποίηση* του κομματιού αυτού.

Και στις δυο υλοποιήσεις χρησιμοποιείται το #pragma omp parallel για την εκτέλεση των thread και εκτελούνται μόνο μέσα σε αυτό το κομμάτι του κώδικα. Εκτός των bracket διατηρείται μόνο το master thread.
Οι global μεταβλητές και οι μεταβλητές της main μοιράζονται μεταξύ των threads και βρίσκονται στο heap. Οι μεταβλητές που δημιουργούνται στο παράλληλο region αποθηκεύονται στη στοίβα και είναι private. Η στοίβα είναι private

Directive που έχουν χρησιμοποιηθεί:
#pragma omp parallel shared() num_threads()
#pragma omp barrier
#pragma omp for nowait
#pragma omp sections
1) Δημιουργείται parallel region που θα εκτελεστεί από τον αριθμό των threads που έχουν ζητηθεί από τον compiler
2)Αποτελεί φράγμα εκτέλεσης στον κώδικα καθώς τα threads αναμένουν σε αυτό το σημείο μέχρι να φτάσουν όλα τα threads που εκτελούν τον κώδικα.
3)Χρησιμοποιείται το directive #pragma omp for για να γίνει share το task μεταξύ των threads. Έτσι οι επαναλήψεις γίνονται split μεταξύ των threads.
4) Χρησιμοποιείται *nowait όταν* δεν υπάρχει λόγος να συγχρονιστούν τα threads στο τέλος των loops.

5) Χρησιμοποιείται section . Δημιουργείται τμήμα κώδικα το οποίο θα γίνει από ένα thread και κάθε thread θα κάνει διαφορετικό section . Είναι αδιάφορο πιο thread α υλοποιήσει την δουλεία .για τον διαχωρισμό των loop στα threads

Fine Grane

Χρησιμοποιείται ως shared οι pointers των string Q,D,scoreTable. Για κάθε loop τα thread θα διαχειρίζονται από κοινού τα string καθώς και τον πίνακα του score και στο scope τους βρίσκονται οι μεταβλητές για match,missmatch,gap. Το parallel region θα εκτελεστεί από αριθμό threads 0 μέχρι num_threads που έχει ζητηθεί απο τον compiler . Η maxValue είναι global μεταβλητή για αυτό είναι *διαχειρίσιμη* από κάθε thread. Ξεκινάει το παράλληλο region πάνω στο υπολογισμό του scoring matrix για την υλοποίηση με μέγεθος task ενός κελιού. Το directive *#pragma omp parallel διατηρεί για κάθε thread στη stack μεταβλητη που διαχειρίζεται τη θέση στο string Q* . Υλοποιείται for μέχρι το πλήθος του Qc. Χρησιμοποιείται το barrier έτσι ώστε τα threads να φτάσουν στο σημείο συγχρονισμένα και για τον υπολογισμό των κελιών να έχουν προηγηθεί οι υπολογισμοί κελιών της ίδια γραμμής που είναι χρήσιμα για τα επόμενα κελιά. Χρησιμοποιείται #pragma omp for nowait ώστε καθε thread να εκτελεί την συνάρτηση calculation για τον υπολογισμό του κελιού που αφορά το threads και να μην περιμένουν συγχρονισμό. Όταν τελειώσουν οι υπολογισμοί του παράλληλου προγράμματος διατηρείται μόνο το master thread το οποίο συνεχίζει την εκτέλεση της διαδικασίας του traceback. Το καθένα παίρνει περίπου γραμμές*στήλες προς threads επαναλήψεις. Η συγκεκριμένη υλοποίηση πάσχει απο load imbalance λόγο του ότι μπορεί να υπάρχουν strings που διαφέρουν αισθητά τα μήκη τους.

Couarse Grane

Αποτελεί την υλοποίηση με μέγεθος task ένα ζεύγος χαρακτήρων. Χρησιμοποιείται παραλληλοποίηση του σειριακό κώδικα σύμφωνα τον αριθμό των ζευγαριών που υπάρχουν στο αρχείο. Χρησιμοποιείται το section ώστε κάθε ένα απο τα threads να χωρίσουν το loop σε τμήματα και κάθε thread να υλοποιήσει το δικό του τμήμα από τα ζευγάρια. Τα τμήματα του calculation του score και του traceback συνδυάζονται

σε νέα συνάρτηση η οποία επιδέχεται παραλληλοποίηση. Κάθε thread αποθηκεύει στη στοίβα μεταβλητή που αφορά ποιο ζευγάρι θα εκτελέσει. Όταν τελειώσουν οι υπολογισμοί του παράλληλου προγράμματος διατηρείται μόνο το master thread το οποίο συνεχίζει την εκτέλεση της διαδικασίας του traceback. Δημιουργείται load balance καθώς αν το φορτίο δεν κατανέμεται εξίσου στον αριθμό των threads, το thread που θα εκτελεστεί τελευταίο θα αναλάβει το μικρότερο φόρτο σε σύγκριση με τα υπόλοιπα που θα έχουν ομοιόμορφη κατανομή του task

PTHREAD

Για τις υλοποιήσεις σύμφωνα με το σχέδιο των Pthreads χρειάζεται ως επιπλέον είσοδος ο αριθμός των threads,

Η παραλληλοποίηση το πρόγραμμά γίνεται ως εξής:

1).Δημιουργείται ένας συγκεκριμένος αριθμός από threads, και το φόρτο γίνεται divide του σε κάθε νήματος ανάλογα με τον αριθμό threads που παράγονται.

2)Αναθέτεται το task ως προς τον αριθμό των thread

3)Το νήμα της Main αδρανοποιείται σε ένα condition variable, έτσι ώστε να μην προχωρήσει αναμένοντας κάποιο awake.

4)Τα νήματα τελειώνουν το task και γίνονται awake στη main, η οποία θα ενώσει τα νήματα και θα δημιουργήσει νέα.

5)Τα task ολοκληρώνονται στη main, θα τυπώνονται οι χρόνοι και οι αποδόσεις της υλοποίησης.

Για την υλοποίηση των pthread έχει τροποποιηθεί το serial πρόγραμμα ώστε να ανταποκρίνεται στη δημιουργία των thread. Ουσιαστική όλο το process αποδίδεται από συνάρτηση.

Fine Grained

Το task αφορά κελί. Στο for loop το master thread στέλνει με την pthread_create() τα νέα νήματα στη συνάρτηση που γίνονται οι υπολογισμοί. Στη συνέχεια πάει και το αρχικό thread στη συνάρτηση για την εκτέλεση και αφού τελειώσει περιμένει τα υπόλοιπα με την pthread_join(). Έχει χωριστεί το φορτίο που λαμβάνει το κάθε thread

σε κάθε loop iteration και για αυτό το λόγο έχει γίνει και τροποποίηση στη συνάρτηση του υπολογισμού . Στην υλοποίηση με μέγεθος task ένα χαρακτήρα, το κάθε thread δεν υπολογίζει ποιες συγκρίσεις θα κάνει. Θα πάρει μέρος στον υπολογισμό όλων των κελιών, ξεκινώντας από το ψηφίο στην θέση που αντιστοιχεί το threads με βήμα ίσο με το πλήθος των νημάτων. Τα κελιά υπολογίζονται από παραπάνω από ένα thread και οι πράξεις γίνονται ατομικά με αποτέλεσμα να αυξηθεί η επικοινωνία. Δημιουργείται load imbalance αν ο αριθμός των ψηφίων δεν διαιρείται απόλυτα με τον αριθμό των νημάτων . Σε εκείνη την περίπτωση το τελευταίο thread θα αναλάβει φόρτο ίσο με το υπόλοιπο της δίαίρεσης.

Couarse Grained

Ακολουθείται όμοια τακτική με την τακτική του coarse grain αλλα για διαφορετικό μέγεθος task. Για την υλοποίηση με μέγεθος task ένα ζευγάρι, στο κάθε νήμα κατανέμεται ολόκληρη η διαδικασία της του calculation και traceback για κάποιο ζευγάρι. Του . Τα tasks μοιράζονται διαιρώντας το πλήθος τους με τον αριθμό των νημάτων και στρογγυλοποιώντας προς τα πάνω. Επειδή το μέγεθος του task είναι μεγάλο, δεν είναι βέλτιστο να πάρει αυτά που περισσεύουν το τελευταίο νήμα. Υπάρχει μεγάλη πιθανότητα load balance διότι αν το φορτίο δεν κατανέμεται ομοιόμορφα , το thread που θα εκτελεστεί τελευταίο θα έχει επεξεργασθεί τα λιγότερα pairs .

Αξιολόγηση αποτελεσμάτων

Η αξιολόγηση του σειριακού προγράμματος αποτυπώνεται στο γράφημα που αφορά τον χρόνο εκτέλεσης στα διάφορα set δεδομένων. Η αύξηση του χρόνο εκτέλεσης οφείλεται στην ανάλογη αύξηση των ζευγαριών των string καθώς και στο μέγεθος τους.

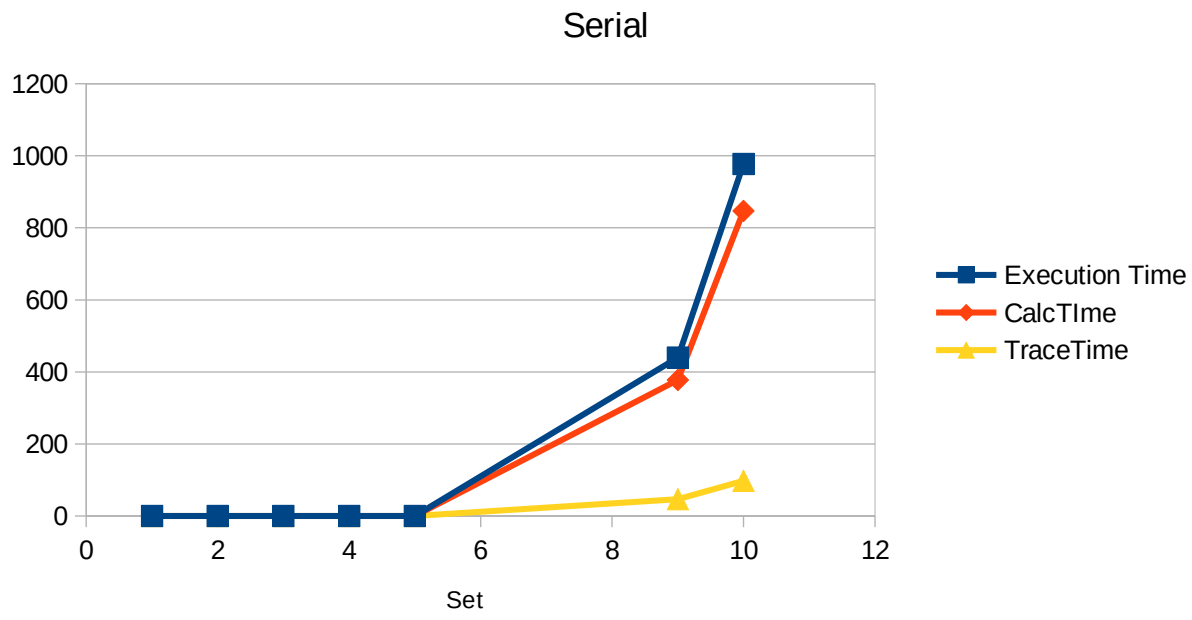
Το Calculation time αποτυπώνει την χρονική διάρκεια εκτέλεσης του προγράμματος. Είναι μεγάλο εξαιτίας του φόρτου των δεδομένων ζευγαριών . Ο συνολικός χρόνος του κάθε dataset εξαρτάται απο: StringQ characters(i)* String D characters(i)* pairNum

όπου i το ι οστό ζευγάρι

το οποίο είναι τεράστιο αποτέλεσμα για τα συγκεκριμένα set

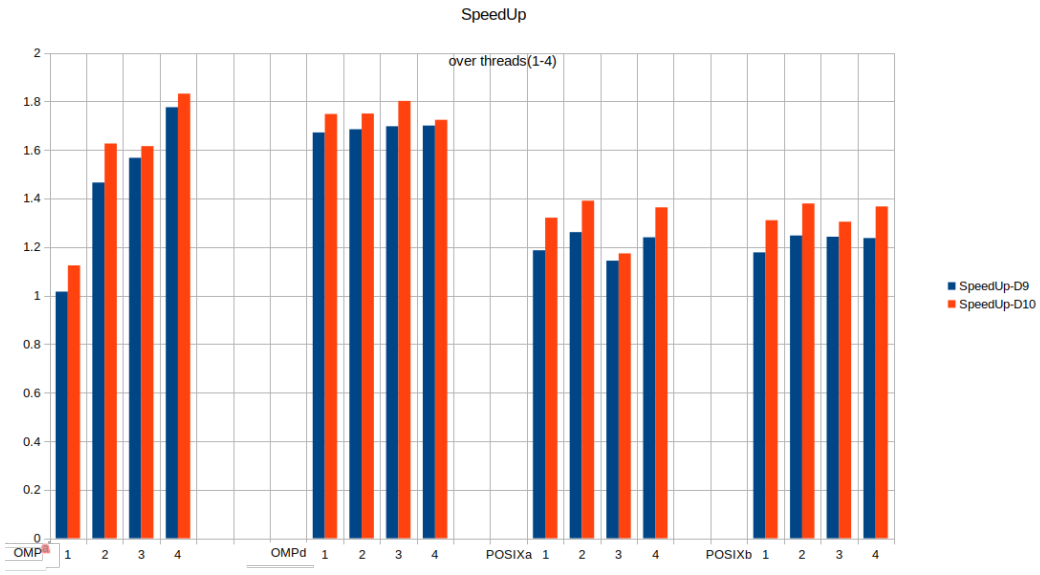
Το Traceback time είναι μικρότερο χρονικά από το calculation καθώς traceback υφίστανται τα κελιά με τιμή ίση με τη μέγιστη τιμή του κελιού του πίνακα. Συγκριτικά πολύ μικρότερος αριθμός κελιών που επιδίδονται στη διαδικασία του trace σε αντίθεση με το calculation.

Στο διάγραμμα επιβεβαιώνεται και η ιδέα της μη παραλληλοποίηση του κώδικα traceback καθώς αποτελεί μικρο ποσοστό φόρτου της συνολικής διεργασίας.



Για την αξιολόγηση αποτελεσμάτων λαμβάνεται η εξίσωση του speedup ως εξής

$$SpeedUp_{exec\ time} = \frac{ExecutionTime_{serial}}{ExecutionTime_{parallell}}$$



Παρατήρηση ότι λαμβάνοντας υπόψη τις εκτελέσεις των προγραμμάτων, οι παράλληλες υλοποιήσεις για το διαφορετικό αριθμό thread έχουν βελτιωμένο χρόνο εκτέλεσης. Για τη fine grained υλοποίηση του OMP επισημαίνεται η αυξανόμενη βελτίωση του χρόνου και η ανοδική βελτίωση του δείκτη speedup. Αυτό οφείλεται στην παραλληλοποίηση του task μεταξύ των thread. Όσο αυξάνεται ο αριθμός των thread υπάρχει βελτιωμένη έκδοση του προγράμματος. Για αριθμό threads=8 έχει σημειωθεί ότι η βελτίωση είναι μηδαμινή και θεωρείται το border της βελτίωσης.

Για τις υλοποιήσεις του POSIX παρατηρείται σταθερή βελτίωση σε σχέση με το σειριακό η οποία δεν βελτιώνεται για τον αριθμό των threads λόγω της ανάγκης επικοινωνίας μεταξύ τους.

Επίσης για 1 thread οι παράλληλοι αλγόριθμοι είναι αργοί εξίσου με το serial επειδή πληρώνουν το overhead για την δημιουργία των νημάτων κλπ χωρίς να μπορούν να

εκμεταλλευτούν την παραλληλία. Γενικά τα αποτελέσματα είναι τα αναμενόμενα.

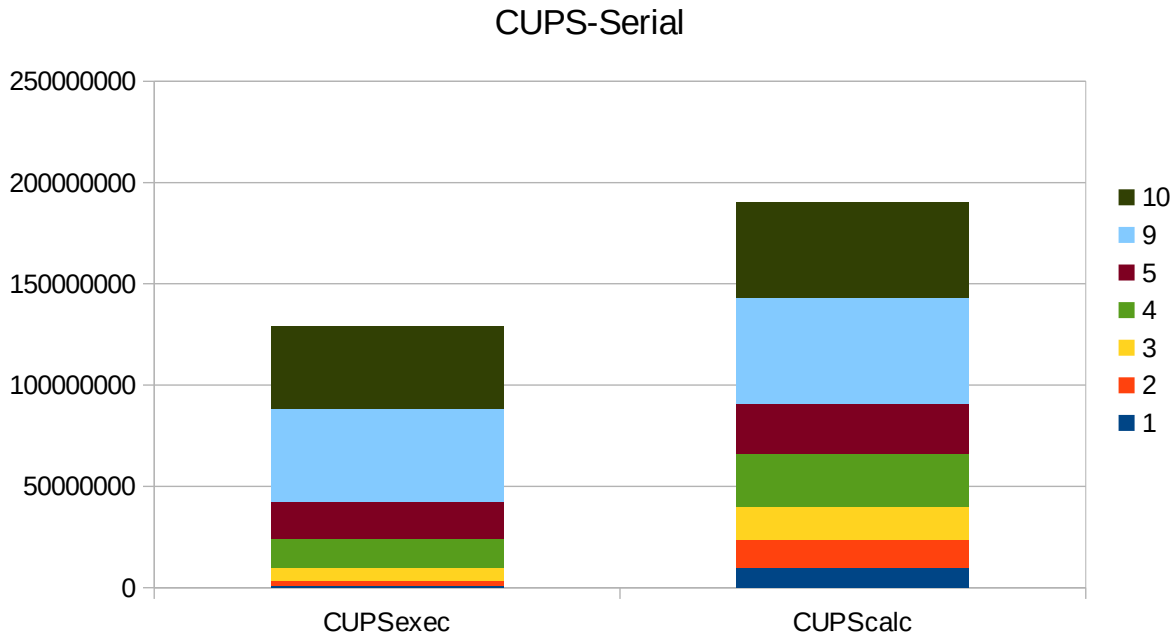
Όσο αφορά τον δείκτη Cell Updates Per Second έχει ορισθεί ακόλουθο δυο ποσοτήτων , του execution time και του calculation time

ως

$$CUPS_{execexec\ time} = \frac{Cells}{ExecutionTime_{parallell}}$$

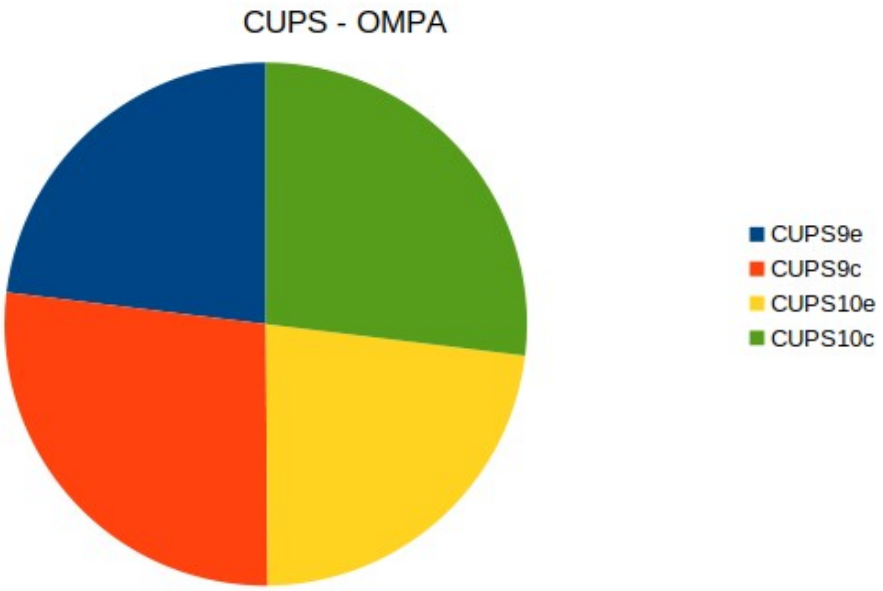
$$CUPS_{calcexec\ time} = \frac{Cells}{CalculationTime_{parallell}}$$

Καθώς ο όγκος το δεδομένων μεγαλώνει από αριθμό σε αριθμό dataset αναμένεται ο αριθμός των κελιών να αυξάνεται από αρχείο σε αρχείο. Ταυτόχρονα η αύξηση του execution time και του calculation αυξάνεται αναλόγως και επομένως διατηρεί σε σταθερά πλαίσια τον δείκτη CUPS για τους δυο χρόνους. Λαμβάνοντας υπόψη ότι το execution time είναι μεγαλύτερο του calculation time για τις διάφορες υλοποιήσεις ο δείκτης CUPS για το execution διατηρεί σταθερή διαφορά απέναντι στο CUPS για το calculation. Αυτό αποτυπώνεται και στο serial program.



Η διαφορά του δείκτη CUPScalc ως προς το δείκτη CUPSexec επιβεβαιώνεται.

Αναλύοντας το CUPS για την υλοποίηση του OMP



Το συμπέρασμα που εξάγεται απο το γράφημα επιβεβαιώνει την αρχική ιδέα . Οι δείκτες CUPS για το calculation υπερισχύουν του execution καθώς ακόμη και με τη βελτίωση των χρόνων ,το execution θα είναι πάντα μεγαλύτερο σε σχέση με το χρόνο του calculation . Οπότε το CUPScal θα συνεχίζει να είναι μεγαλύτερο.

Όμοια συμπεράσματα αποδεικνύονται και για τις υπόλοιπες υλοποιήσεις

(βλέπε γραφήματα excel).

Γενικά από τα γραφήματα συμπεραίνεται ότι οι παράλληλοι αλγόριθμοι γίνονται πιο αποτελεσματικοί όσο ο όγκος των δεδομένων μεγαλώνει. Επίσης η απόδοση τους εξαρτάται από τη δομή των δεδομένων. Σύμφωνα με τις μετρήσεις μας, την καλύτερη επίδοση έχουν οι αλγόριθμοι για fine granularity, ανεξαρτήτως αν είναι υλοποιημένοι με OpenMP ή Pthreads. Έχουν το καλύτερο computation to communication ratio καθώς η μόνη επικοινωνία που χρειάζεται είναι η διαχείριση των θέσεων που λαμβάνουν για τα κελιά που θα υπολογίσουν, που έχουν τα νήματα και το task είναι αρκετά μικρό ώστε να μην δημιουργείται load imbalance.

Συμπεράσματα και παρατηρήσεις.

Το OpenMP API γίνεται κατανοητό και γρήγορο στην εκμάθηση, στην χρήση και την συντήρηση του κώδικα. Ο μεθοδολογία γραφής του είναι πολύ ξεκάθαρη με σημαντικό μειονέκτημα ότι δεν υπάρχουν ιδιαίτερα περιθώρια για βελτιστοποιήσεις λόγω περιορισμών. Η high level υλοποίηση δεν επιτρέπει στον συγγραφέα του κώδικα να αντιληφθεί πως συμπεριφέρεται στο background η υλοποίηση του. Για το λόγο αυτό μπορούν να υπάρξουν δυσκολίες για τις οποίες δεν ευθύνεται άμεσα ο κώδικας . Αντίθετα τα Pthreads δεν έχουν την ίδια αυτοματοποίηση. Είναι πιο χρονοβόρα στη δημιουργία του κώδικα και στο συγχρονισμό καθώς είναι ευθύνη του δημιουργού πως θα διασφαλίσει την επικοινωνία μεταξύ των thread λόγω χάρη η ασφάλεια και η ελευθερία του έχει το thread αναλαμβάνοντας τη λειτουργία του mutex. Τα πλεονεκτήματά τους είναι η δυνατότητα αλλαγής και βελτίωσης λεπτομερειών, καθώς και γνωστή η ακριβής λειτουργικότητα του κώδικα.

Σκοπός του OpenMp είναι η υλοποίηση παραλληλισμού με σχετικά εύκολα βήματα ,ελάχιστες εντολές και μικρή πολυπλοκότητα. Οι εσωτερικοί έλεγχοι γίνονται από το σύστημα οπότε δεν υπάρχει μεγάλη ευχέρεια αλλαγή από τον χρήστη. Το αρνητικό είναι ότι ίσως δεν κάνει την βέλτιστη χρήση της κοινής μνήμης