Παράλληλη Επεξεργασία

Βλάσης Παναγιώτης Παναγιώτου ΑΜ: 1067517

Σοφία Λαμπροπούλου ΑΜ: 1072606

Κωνσταντίνος Παρασκευόπουλος ΑΜ: 1072608

Δανάη Χαλούλου ΑΜ: 1072596

# Εισαγωγή

Για την υλοποίηση της εργασίας χρησιμοποιήθηκε υπολογιστής με 4 πυρήνες, 8 νήματα και 2,8 GHz συχνότητα. Ωστόσο επειδή η υλοποίηση των προγραμμάτων έγινε σε virtual machine χρησιμοποιήθηκαν μόνο δύο πυρήνες και 4 νήματα. Εξαίτιας του virtual machine υπήρχε περιορισμός στην χρήση της RAM στα 4 GB.

Παρατηρώντας τον σειριακό κώδικα που μας δόθηκε, η καθυστέρηση στον υπολογισμό οφείλεται στα μεγάλα for loops για όλα τα trials στην συνάρτηση main. Οπότε σε αυτό το σημείο θα γίνει το μεγαλύτερο μέρος της παραλληλοποίησης. Επίσης η υλοποίηση χρησιμοποιεί και γεννήτρια τυχαίων αριθμών η οποία εκ φύσεως της προκαλεί καθυστερήσεις στον υπολογισμό, η οποία μάλλον ωστόσο μας δίνει την δυνατότητα σε ορισμένα σημεία να μην μας ενδιαφέρει η σειρά εκτέλεσης.

**Εξήγηση σειριακού κώδικα**

Η συνάρτηση double f(double \*x, int n) είναι ουσιαστικά η σύναρτηση f(x) (rosenbrock) που δίνεται στην εκφώνηση, και υπολογίζει ένα σημείο Χ (το σημειο Χ αναπαρισταται απο ενα n-διαστατο διανυσμα) στο οποίο η μη-γραμμική συνάρτηση f(x) έχει ένα τοπικό ελάχιστο.Η συνάρτηση f(x) έχει καθολικό ελάχιστο στο σημείο (1,1,...,1) (n-διαστατο σημειο), όπου παίρνει την τιμή 0.Για την ευρεση ενος τοπικου ελαχιστο πραγματοποιειται ευθύγραμμη αναζήτηση από το τρέχον σημείο προς συγκεκριμένες διευθύνσεις. Εφόσον βρεθεί κάποιο νέο σημείο με καλύτερη συναρτησιακή τιμή τότε η μέθοδος αντικαθιστά το τρέχον σημείο με το νέο. Διαφορετικά, προσαρμόζει τις παραμέτρους της ευθύγραμμης αναζήτησης και επαναλαμβάνει τη διαδικασία στο ίδιο σημείο.Η συνάρτηση int hooke(int nvars, double startpt[MAXVARS], double endpt[MAXVARS], double rho, double epsilon, int itermax) κανει αυτη την ευθυγραμμη αναζητηση.

Στην αρχη ομως του προγραμματος , μεσω της υπορουτινας/συναρτησης double best\_nearby() γινεται ενας αρχικος τυχαιοκρατικος υπολογισμος ενος τυχαιου τοπικου ελαχιστου Χ.

Η συναρτηση int hooke() ξεκιναει απο αυτο το αρχικο τυχαιο τοπικο ελαχιστο και κανει ευθυγραμμη αναζητηση προς συγκεκριμενες διευθυνσεις που καθοριζονται μεσα στη συναρτηση int hooke.

# Αναλυτική τεκμηρίωση παράλληλων υλοποιήσεων και τυχόν βελτιστοποιήσεων

OpenMP

Αρχικά χρησιμοποιούμε την βιβλιοθήκη omp.h.

Η βελτιστοποίηση έγινε για τον κύριο βρόχο for ο οποίος προκαλεί και την μεγαλύτερη καθυστέρηση. Επίσης χρησιμοποιούμε τις μεταβλητές fx, jj, I , οι οποίες είναι private για κάθε νήμα. Και τέλος την μεταβλητή num\_of\_threads η οποία θα ορίζει πόσα νήματα θα δημιουργηθούν κάθε φορά. Εκτός αυτού του βρόχου μπορούσε να γίνει παραλληλοποίηση και της for που υπολογίζει με τυχαιοκρατικό τρόπο την συνάρτηση Rosenbrock με πεδίο αναζήτησης το [-5, 5). Ακόμη δημιουργήσαμε ένα critical section, ώστε να μας επιστρέφονται τα τελικά αποτελέσματα με ασφάλεια και να μην υπάρχει πιθανότητα δυο νήματα να γράψουν στην ίδια μεταβλητή όπως η best\_trial. Τέλος για να βρούμε τον πραγματικό αριθμό των πράξεων πολλαπλασιάζουμε την μεταβλητή funevals με τον συνολικό αριθμό των νημάτων num\_of\_threads, αυτή η τακτική χρησιμοποιείται σε όλα τα πρωτόκολλα παραλληλοποίησης.

**Γενικά η παραλληλοποίηση**

Text

Description automatically generated

**Παραλληλοποίηση βρόχου**



**Critical Section**

Text

Description automatically generated

**Εύρεση συνολικού αριθμού υπολογισμών συνάρτησης**



OpenMP Tasks

Αρχικά χρησιμοποιούμε την βιβλιοθήκη omp.h.

Η βελτιστοποίηση έγινε για τον κύριο βρόχο for ο οποίος προκαλεί και την μεγαλύτερη καθυστέρηση. Επίσης χρησιμοποιούμε τις μεταβλητές fx, jj, I , οι οποίες είναι private για κάθε νήμα. Και τέλος την μεταβλητή num\_of\_threads η οποία θα ορίζει πόσα νήματα θα δημιουργηθούν κάθε φορά. Εκτός αυτού του βρόχου μπορούσε να γίνει παραλληλοποίηση και της for που υπολογίζει με τυχαιοκρατικό τρόπο την συνάρτηση Rosenbrock με πεδίο αναζήτησης το [-5, 5). Εντός του κύριου βρόχου for γράψαμε ένα block κώδικα που μπορεί να χρησιμοποιηθεί από οποιοδήποτε νήμα δίχως να περιμένει να τελειώσουν τα υπόλοιπα, το οποίο μας βοηθάει σε αυτήν την περίπτωση διότι βρισκόμαστε εντός ενός for loop περιορισμένου αριθμού επαναλήψεων. Ακόμη, εντός του κύριου βρόχου for, δημιουργήσαμε μια περιοχή δυαδικού σημαφόρου (mutex area) για τις μεταβλητές που μας δίνουν το τελικό αποτέλεσμα ώστε να εξασφαλιστεί η άρτια εκτέλεση του προγράμματος.

**Γενικά η παραλληλοποίηση**

Text

Description automatically generated

**Έναρξη παραλληλισμού και δημιουργία νημάτων** **μέσω της ακέραιας παραμέτρου number\_of\_threads**

Text

Description automatically generated

**Έναρξη block κώδικα που δύναται να γίνει από οποιοδήποτε νήμα δίχως να αναμένει να τελειώσουν και τα άλλα**



**Χρήση της συνάρτησης/μεθοδου firstprivate ώστε να ορίσουμε τις μεταβλητές που δεν επιθυμούμε να διαμοιράζονται μεταξύ των νημάτων και έναρξη tasks ώστε να πραγματοποιηθεί υπολογισμός της συνάρτησης**



**Critical section**

A screenshot of a video game

Description automatically generated with medium confidence

**Εύρεση συνολικού αριθμού υπολογισμών συνάρτησης**



MPI

Αρχικά χρησιμοποιούμε την βιβλιοθήκη mpi.h.

Η βελτιστοποίηση έγινε για τον κύριο βρόχο for ο οποίος προκαλεί και την μεγαλύτερη καθυστέρηση. Ορίσαμε κατάλληλες μεταβλητές για την εποπτεία του rank και του size της διεργασίας, για την πηγή και το tag των μηνυμάτων, καθώς και για τον προορισμό και την κατάσταση των μηνυμάτων. Έπειτα, μέσω της δήλωσης 4 κατάλληλων εντολών εξασφαλίσαμε την άρτια διαχείριση και αλληλεπίδραση των διεργασιών. Τέλος, εντός του βρόχου for πραγματοποιούμε χωρισμό του αριθμού των trials σε κάθε διεργασία, και εντός αυτού του for loop υπάρχουν ακόμα 2 εμφωλευμένα for loops, δομές if-else και κατάλληλες εντολές MPI\_Send και MPI\_Recv για τον υπολογισμό από κάθε διεργασία των καλύτερων τιμών, την αποστολή των δεδομένων των workers-processes στις master-processes και την αποδοχή των δεδομένων της κάθε διεργασίας με σειριακό τρόπο.

**Γενικά η παραλληλοποίηση**

Text

Description automatically generated

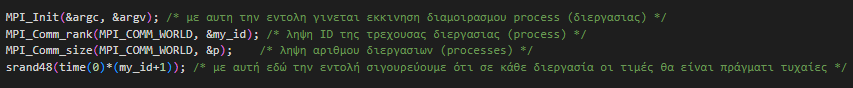
Text

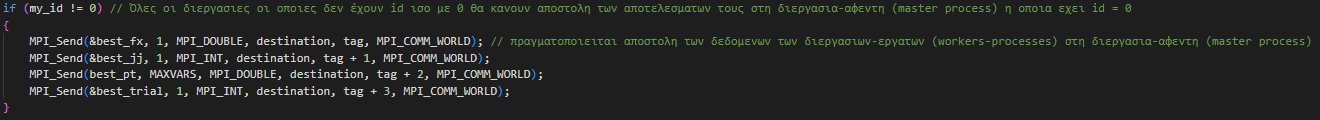
Description automatically generated

**Μεταβλητές και εντολές για την εκτέλεση του ΜΡΙ**

Text

Description automatically generated





Text

Description automatically generated

**Εύρεση συνολικού αριθμού υπολογισμών συνάρτησης**



OpenMP + MPI hybrid

Συνδυάσαμε τις 2 παραπάνω υλοποιήσεις, ουσιαστικά σπάμε το μεγάλο loop σε p κομμάτια ενώ χρησιμοποιούμε και το openmp ώστε να παραλληλοποιήσουμε τον βρόχο όπως εξηγήσαμε παραπάνω προσέχοντας βέβαια και το critical section στο οποίο χρησιμοποιούμε mutex προκειμένου να εξασφαλίσουμε ότι δε θα γράψουν 2 νήματα ταυτόχρονα στην ίδια μεταβλητή. Έξω από το loop χρησιμοποιώντας τη συνάρτηση MPI\_Send στέλνουμε τα αποτελέσματα στον “master” πυρήνα ο οποίος θα λαμβάνει μέσω της εντολής MPI\_Recv. Τέλος, κλείνουμε το MPI με την εντολή MPI\_Finalize.

**Γενικά η παραλληλοποίηση**

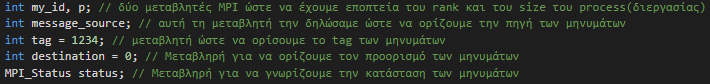
Text

Description automatically generated

Text

Description automatically generated

**Μεταβλητές και εντολές για την εκτέλεση του OpenMP + MPI**





**Έναρξη παραλληλισμού OpemMP , δημιουργία νημάτων με την παράμετρο num\_threads(int) και χρήση της μεθόδου/συνάρτησης firstprivate για να κάνουμε ορισμό των μεταβλητών που δεν επιθυμούμε να γίνεται διαμοίραση μεταξύ των νημάτων (threads)**

Άνοιγμα φωτογραφίας

**Παραλληλοποίηση βρόχου**

Άνοιγμα φωτογραφίας

**Critical Section**

Άνοιγμα φωτογραφίας

**Εύρεση συνολικού αριθμού υπολογισμών συνάρτησης**

Άνοιγμα φωτογραφίας

# Αποτελέσματα

Εκτελώντας τις παραπάνω υλοποιήσεις σε 3 διαφορετικούς αριθμούς τμημάτων λαμβάνουμε τα εξής αποτελέσματα:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **4 Threads** | **6 Threads** | **8 Threads** |
| **OpenMP** | 598,866 | 413,422 | 400,153 |
| **OpenMP Tasks** | 485,654 | 409,211 | 361,715 |
| **MPI** | 522,024 | 479,714 | 413,86 |
| **MPI + OpenMP** | 460,756 | 360,937 | 360,867 |

*Στον κατακόρυφο άξονα, φαίνεται ο χρόνος εκτέλεσης σε δευτερόλεπτα ενώ στον οριζόντιο το νήμα της υλοποίησης.*

# Συμπεράσματα

Αρχικά, παρατηρούμε ότι για 4 νήματα το MPI+ΟpenMP είναι ο πιο γρήγορος τρόπος παραλληλοποίησης το οποίο είναι λογικό αφού χρησιμοποιεί 2 είδη παραλληλοποιήσεων. Στη συνέχεια το OpenMP Tasks που είναι πιο γρήγορο από το MPI και το OpenMP καθώς παραλληλοποιεί χρησιμοποιώντας tasks που είναι μια αρκετά πιο ελαφριά μορφή διεργασίας. Στη συνέχεια, εκτελώντας στα 6 νήματα παρατηρούμε ότι αυτή η κατάταξη συνεχίζει ως ήταν ωστόσο το MPI αυτή τη φορά είναι πιο αργό από το OpenMP και είναι ουσιαστικά ο πιο αργός τρόπος παραλληλοποίησης. Τέλος, στα 8 νήματα παρατηρούμε ότι τα 4 είδη παραλληλοποίησης έχουν πολύ μικρές διαφορές με τη σειρά να είναι MPI+OpenMP το γρηγορότερο, στη συνέχεια ακολουθούν OpenMP Tasks (με διαφορά μόλις 1 δευτερόλεπτο), OpenMP και τελευταίο το MPI.

*\*\*Οι παράξενοι χρόνοι εκτέλεσης του MPI* *ίσως οφείλονται στο ότι χρησιμοποιήσαμε διαφορετικό υπολογιστή για την εκτέλεσή του.*

Γενικότερα οι αναμενόμενες παρατηρήσεις θεωρούμε πως είναι MPI+OpenMP να είναι πάντα το γρηγορότερο, MPI και OpenMP Tasks να ακολουθούν με αυτή τη σειρά και τελευταίο το OpenMP.

|  |  |  |
| --- | --- | --- |
| **Speedup 4** | **Speedup 6** | **Speedup 8** |
| 2,306526 | 3,341138 | 3,45193 |
| 2,844206 | 3,37552 | 3,818752 |
| 2,646047 | 2,879424 | 3,337602 |
| 2,997899 | 3,826984 | 3,827726 |

Τα αντίστοιχα speedup είναι τα παραπάνω και υπολογίζονται από τον τύπο t1/tp όπου t1 ο χρόνος εκτέλεσης της διαδικασίας σειριακά (1381,300 sec) και tp ο αντίστοιχος χρόνος εκτέλεσης για κάθε αριθμό νημάτων.