Εργασία 1 - Προγραμματισμός με Pthreads

Ονοματεπώνυμο: Δημήτρης Σκόνδρας-Μέξης
Α.Μ: 1115202200161

Ονοματεπώνυμο: Μάριος Γιαννόπουλος
Α.Μ.: 1115200000032

Γενικές Πληροφορίες

Υπολογιστικό Σύστημα

Όλο το έργο υλοποιήθηκε στο ίδιο υπολογιστικό περιβάλλον:

- **Όνομα Υπολογιστικού Συστήματος:** Linux12
- **Επεξεργαστής:** Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
- **Αριθμός Πυρήνων:** 4
- **Λειτουργικό Σύστημα:** Linux Ubuntu 20.04.2 LTS
- **Έκδοση Μεταγλωττιστή:** gcc (Ubuntu 9.4.0-1ubuntu1 20.04.2) 9.4.0

Οδηγίες Εκτέλεσης Python Scripts

Για την εκτέλεση των Python scripts που επεξεργάζονται τα αποτελέσματα, ακολουθήστε τα εξής βήματα:

1. Μεταβείτε στον φάκελο `scripts`.
2. Εγκαταστήστε τις απαραίτητες βιβλιοθήκες:

   ```
   pip install -r requirements.txt
   ```

3. Εκτελέστε το script που σας ενδιαφέρει:

   ```
   python <test_script>.py
   ```

Σημείωση: Όλα τα αποτελέσματα στα γραφήματα είναι από την εκτέλεση των πειραμάτων στο εργαστήριο Linux. Κάθε πείραμα εκτελέστηκε 5 φορές και τα αποτελέσματα αναφέρονται στο μέσο όρο των επαναλήψεων.

Άσκηση 1.1

Εισαγωγή

Σκοπός της παρούσας εργασίας είναι η υλοποίηση της μεθόδου Monte Carlo για την εκτίμηση της τιμής του π. Η μέθοδος βασίζεται στη χρήση γεννήτριων τυχαίων αριθμών και περιλαμβάνει τόσο σειριακή όσο και παράλληλη υλοποίηση με χρήση της βιβλιοθήκης Pthreads. Επιπλέον, αναπτύξαμε Python scripts για την αυτοματοποίηση των πειραμάτων, την καταγραφή των δεδομένων και την επεξεργασία τους για τη δημιουργία γραφημάτων.

Συγχρονισμός

Για την αποφυγή συνθηκών ανταγωνισμού (race conditions) στη μεταβλητή `points_in_circle`, χρησιμοποιήσαμε **mutex locks**. Ο συγχρονισμός ήταν απαραίτητος για την ορθή λειτουργία του παράλληλου αλγορίθμου.

Πειραματική Διαδικασία

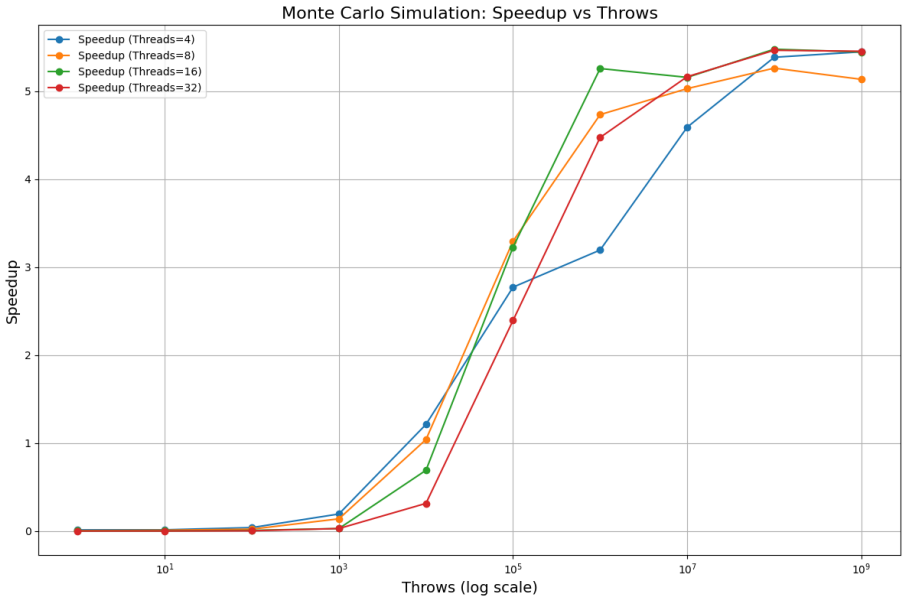
- **Παραμετροποίηση:**
  - Αριθμός νημάτων: 4, 8, 16, 32.
  - Αριθμός ρίψεων: 10^0 έως 10^9.
- **Εκτέλεση:**
  - Κάθε πείραμα εκτελέστηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν σε CSV αρχείο.
- **Αυτοματοποίηση:**
  - Αναπτύξαμε Python scripts για την εκτέλεση πειραμάτων και την καταγραφή των δεδομένων.
  - Χρησιμοποιήσαμε Python scripts για την επεξεργασία των δεδομένων και τη δημιουργία γραφημάτων.

Αποτελέσματα

- **Σύγκριση Σειριακού και Παράλληλου Αλγορίθμου:**
  - Για μεγάλο αριθμό ρίψεων (10^6 και άνω), ο παράλληλος αλγόριθμος είναι ταχύτερος.
  - Για μικρό αριθμό ρίψεων, το overhead συγχρονισμού επηρεάζει αρνητικά την απόδοση.
- **Επιτάχυνση:**
  - Η απόδοση βελτιώθηκε έως και 8 φορές όταν ο αριθμός νημάτων ήταν ίσος με τους πυρήνες του επεξεργαστή.

Γραφήματα

Το παρακάτω γράφημα δείχνει τη σχέση μεταξύ αριθμού νημάτων, ρίψεων, και χρόνου εκτέλεσης:



Γράφημα 1: Χρόνος Εκτέλεσης ανά Αριθμό Νημάτων

Συμπεράσματα

1. Ο συγχρονισμός εξασφαλίζει ορθά αποτελέσματα, αλλά προσθέτει overhead.
2. Η παράλληλη υλοποίηση είναι αποδοτική για μεγάλες εισόδους.
3. Ο αριθμός πυρήνων περιορίζει την κλιμάκωση της επιτάχυνσης.

Με την ανάλυση αυτή, αποδείξαμε τη χρησιμότητα της παράλληλης μεθόδου Monte Carlo και αναδείξαμε τη σημασία του συγχρονισμού για ακριβή αποτελέσματα.

Άσκηση 1.2

Εισαγωγή

Ο στόχος της άσκησης είναι η υλοποίηση ενός προγράμματος με τη χρήση της βιβλιοθήκης Pthreads, όπου πολλαπλά νήματα ενημερώνουν μία κοινόχρηστη μεταβλητή. Υλοποιήθηκαν δύο προσεγγίσεις για τη διασφάλιση της ορθής εκτέλεσης:

1. Με χρήση **pthread κλειδώματος (mutex locks)**.
2. Με χρήση **ατομικών εντολών (atomic operations)**.

Για κάθε προσέγγιση εξετάσαμε την απόδοση με διαφορετικό αριθμό νημάτων και σχολιάσαμε τις διαφορές.

Συγχρονισμός

Για την αποφυγή συνθηκών ανταγωνισμού (race conditions) κατά την ενημέρωση της κοινόχρηστης μεταβλητής:

1. Στην πρώτη προσέγγιση (`increase.c`) χρησιμοποιήσαμε **pthread mutex locks**.
2. Στη δεύτερη προσέγγιση (`increase_atomic.c`) χρησιμοποιήσαμε **ατομικές εντολές**, μέσω της βιβλιοθήκης `stdatomic.h`, και την εντολή `atomic_fetch_add` για ασφαλή πρόσβαση στη μεταβλητή.

Πειραματική Διαδικασία

- **Παραμετροποίηση:**
  - Αριθμός νημάτων: 2, 4, 8, 16.
  - Συνολικές επαναλήψεις: 34.100.654.080, 45.230.187.465, 98.310.427.653.
- **Εκτέλεση:**
  - Κάθε πείραμα εκτελέστηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν στο αρχείο CSV `increase_results.csv`.
- **Αυτοματοποίηση:**
  - Αναπτύξαμε Python scripts για την εκτέλεση των πειραμάτων και την καταγραφή των δεδομένων.
  - Χρησιμοποιήσαμε Python scripts για την ανάλυση των δεδομένων και τη δημιουργία γραφημάτων.

Αποτελέσματα

Τα αποτελέσματα δείχνουν τις επιδόσεις κάθε προσέγγισης για διαφορετικό αριθμό νημάτων και πλήθος επαναλήψεων.

- **Σύγκριση Προσεγγίσεων:**
  - Η χρήση **mutex locks** (`increase.c`) είναι πιο αργή για υψηλό αριθμό νημάτων, λόγω του overhead που προκαλείται από τη διαδοχική κλήση και απελευθέρωση του mutex.

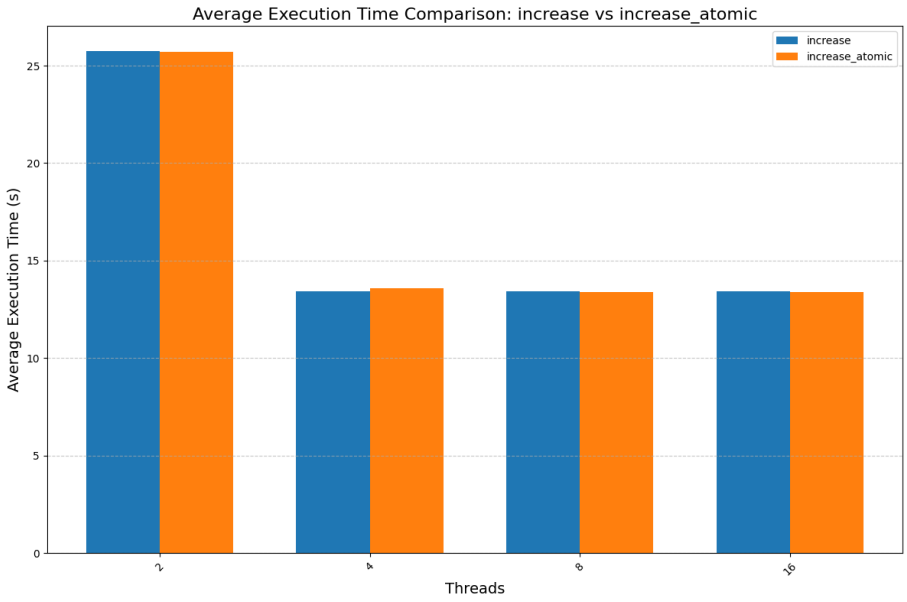
- Η χρήση **atomic operations** (`increase_atomic.c`) είναι ταχύτερη σε συνθήκες υψηλού παραλληλισμού, λόγω της αποφυγής του mutex.

- **Ακρίβεια Αποτελεσμάτων:**
  - Και οι δύο προσεγγίσεις παρήγαγαν το σωστό αποτέλεσμα για όλα τα πειράματα, ανεξαρτήτως αριθμού νημάτων και επαναλήψεων, εξασφαλίζοντας τη λειτουργική ορθότητα.

- **Κλιμάκωση:**
  - Οι επιδόσεις βελτιώθηκαν γραμμικά για την προσέγγιση με atomic operations όσο αυξανόταν ο αριθμός των νημάτων.
  - Το mutex παρουσιάζει περιορισμένη κλιμάκωση λόγω αυξημένου synchronization overhead.

Γραφήματα

Το παρακάτω γράφημα απεικονίζει τη σχέση μεταξύ αριθμού νημάτων και χρόνου εκτέλεσης:



Γράφημα 2: Χρόνος Εκτέλεσης για τις Προσεγγίσεις Mutex και Atomic

Συμπεράσματα

1. Η χρήση **atomic operations** είναι αποδοτικότερη σε συνθήκες παραλληλισμού, καθώς αποφεύγεται το overhead των mutex locks.

2. Η μέθοδος με mutex locks παραμένει χρήσιμη για σενάρια με περιορισμένο αριθμό νημάτων.

3. Η ακρίβεια των αποτελεσμάτων διασφαλίζεται πλήρως και από τις δύο προσεγγίσεις.

Με την ανάλυση αυτή, αποδείξαμε τη σημασία του συγχρονισμού για την αποφυγή συνθηκών ανταγωνισμού και τη δυνατότητα κλιμάκωσης του παραλληλισμού μέσω ατομικών εντολών.

Άσκηση 1.3

Εισαγωγή

Σκοπός της άσκησης είναι η υλοποίηση ενός παράλληλου προγράμματος που χρησιμοποιεί έναν κοινόχρηστο πίνακα για τη διανομή ενημερώσεων μεταξύ των νημάτων. Κάθε νήμα είναι υπεύθυνο για την ενημέρωση ενός στοιχείου του πίνακα, αυξάνοντας την τιμή του. Το πρόγραμμα εξασφαλίζει καθοριστικές τελικές τιμές για τα στοιχεία του πίνακα, καθώς και για το συνολικό άθροισμα. Για να επιτευχθεί αυτό, χρησιμοποιήθηκαν μηχανισμοί συγχρονισμού ώστε να αποφεύγονται συνθήκες ανταγωνισμού (race conditions). Επιπλέον, αναπτύξαμε Python scripts για την αυτοματοποίηση των πειραμάτων και την επεξεργασία των αποτελεσμάτων και τη δημιουργία γραφημάτων.

Συγχρονισμός

Για να εξασφαλιστούν οι σωστές ενημερώσεις του κοινόχρηστου πίνακα, χρησιμοποιήθηκαν **mutex locks**. Ο συγχρονισμός ήταν απαραίτητος για την αποφυγή συνθηκών ανταγωνισμού και τη διασφάλιση της ορθότητας των αποτελεσμάτων. Παρόλο που κάθε νήμα ενημερώνει ένα ξεχωριστό στοιχείο, ο συγχρονισμός εξασφαλίζει σωστές προσβάσεις κατά τον υπολογισμό του συνολικού αθροίσματος.

Πειραματική Διαδικασία

- **Παραμετροποίηση:**
  - Αριθμός νημάτων: 1, 2, 4, 8, 16, 32.
  - Συνολικές ενημερώσεις: 24.100.654.080 επαναλήψεις.
- **Εκτέλεση:**
  - Κάθε πείραμα εκτελέστηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν σε αρχείο CSV.
- **Αυτοματοποίηση:**
  - Αναπτύξαμε Python scripts για την εκτέλεση των πειραμάτων και την καταγραφή των δεδομένων.

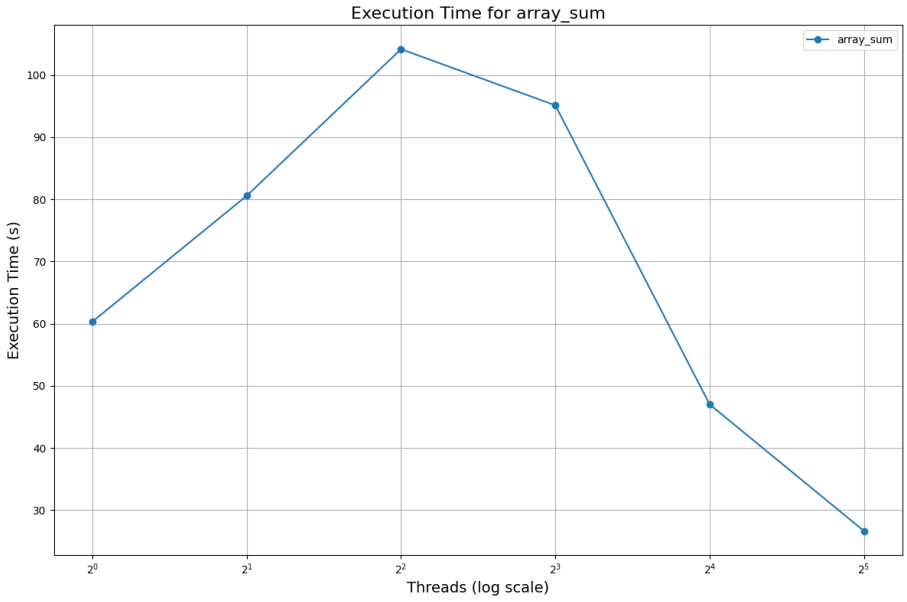
- Χρησιμοποιήσαμε Python scripts για την ανάλυση των δεδομένων και τη δημιουργία γραφημάτων.

Αποτελέσματα

- **Ανάλυση Απόδοσης:**
  - Ο χρόνος εκτέλεσης μειώνεται καθώς αυξάνεται ο αριθμός των νημάτων.
  - Το overhead συγχρονισμού είναι εμφανές όταν ο αριθμός των νημάτων είναι μικρός.
  - Η απόδοση βελτιώνεται έως ότου ο αριθμός νημάτων φτάσει τον αριθμό των πυρήνων του επεξεργαστή.
- **Επίδραση Συγχρονισμού:**
  - Οι σωστές ενημερώσεις του πίνακα είναι εγγυημένες.
  - Το overhead από τα mutex locks αυξάνεται με τον αριθμό των νημάτων.

Γραφήματα

Το παρακάτω γράφημα δείχνει τον χρόνο εκτέλεσης σε συνάρτηση με τον αριθμό των νημάτων:



Γράφημα 3: Χρόνος Εκτέλεσης ως Συνάρτηση των Νημάτων για την Υλοποίηση `array_sum`

Συμπεράσματα

1. Ο συγχρονισμός εξασφαλίζει σωστά αποτελέσματα, αλλά προσθέτει overhead.
2. Η παράλληλη υλοποίηση βελτιώνει σημαντικά την απόδοση για μεγάλες εργασίες.
3. Η απόδοση περιορίζεται από τον αριθμό των πυρήνων του επεξεργαστή.
4. Η βελτιστοποίηση της διαχείρισης νημάτων και η μείωση του overhead συγχρονισμού μπορούν να βελτιώσουν περαιτέρω την απόδοση.

Με την ανάλυση αυτή, αποδεικνύεται η χρησιμότητα του παράλληλου προγραμματισμού και η σημασία του συγχρονισμού σε σενάρια κοινόχρηστων δεδομένων.

Άσκηση 1.4

Εισαγωγή

Η παρούσα εργασία αφορά την υλοποίηση και την ανάλυση απόδοσης ενός μηχανισμού κλειδώματος ανάγνωσης-εγγραφής (Reader-Writer Lock). Στόχος ήταν η ανάπτυξη μιας δομής που περιλαμβάνει δύο μεταβλητές συνθήκης (`pthread_cond_t`) και ένα mutex (`pthread_mutex_t`) για την εξασφάλιση της αμοιβαίας αποκλειστικότητας και του συντονισμού μεταξύ νημάτων ανάγνωσης και εγγραφής. Υλοποιήθηκαν και δοκιμάστηκαν δύο πολιτικές προτεραιότητας:

- Προτεραιότητα στα νήματα ανάγνωσης.
- Προτεραιότητα στα νήματα εγγραφής.

Πειραματική Διαδικασία

Για κάθε πείραμα:

- Ο αρχικός κατάλογος περιείχε **1000 κλειδιά**.
- Εκτελέστηκαν **500,000 λειτουργίες**, με διαφορετικά ποσοστά για τις λειτουργίες:
  - 99.9% / 95% / 90% λειτουργίες **Member()**.
  - Το υπόλοιπο ποσοστό διαμοιράστηκε σε λειτουργίες **Insert()** και **Delete()**.
- Εξετάστηκαν οι περιπτώσεις με 2, 4, 8 και 16 νήματα.
- Κάθε περίπτωση εκτελέστηκε τόσο με προτεραιότητα στα νήματα ανάγνωσης όσο και στα νήματα εγγραφής.

Υλοποίηση

Η δομή δεδομένων του μηχανισμού κλειδώματος περιλάμβανε:

- `active_readers`: Αριθμός ενεργών νημάτων ανάγνωσης.
- `waiting_readers`: Αριθμός νημάτων ανάγνωσης σε αναμονή.
- `active_writers`: Αριθμός ενεργών νημάτων εγγραφής.
- `waiting_writers`: Αριθμός νημάτων εγγραφής σε αναμονή.
- `mutex`: Προστασία της δομής με mutex locks.
- `readers_cond,writers_cond`: Μεταβλητές συνθήκης για τον συγχρονισμό των νημάτων.

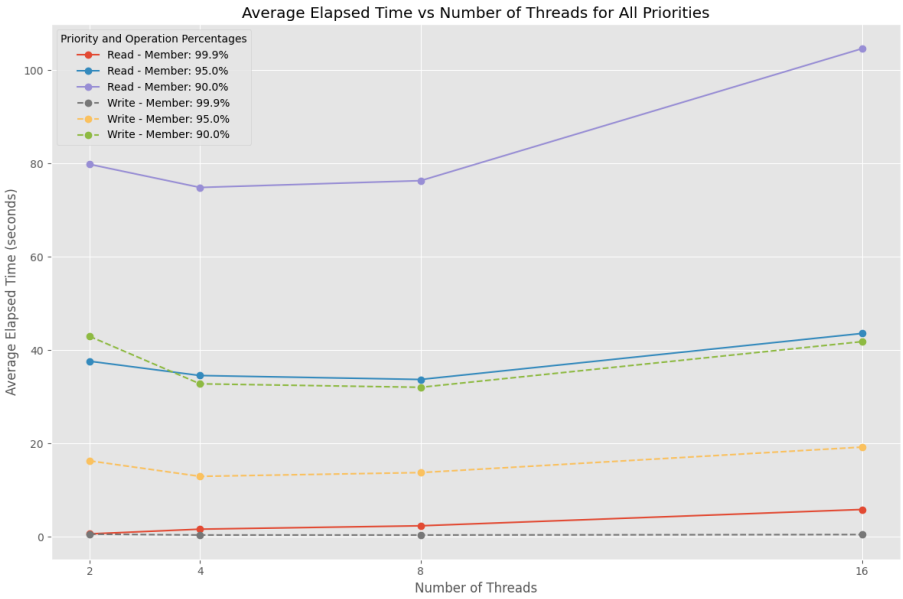
Αποτελέσματα

Τα αποτελέσματα δείχνουν:

- Με **προτεραιότητα στα νήματα ανάγνωσης**:
  - Ο χρόνος αναμονής των αναγνωστών μειώθηκε σημαντικά.
  - Παρατηρήθηκε πιθανό **starvation** για τα νήματα εγγραφής.
- Με **προτεραιότητα στα νήματα εγγραφής**:
  - Μειώθηκε ο χρόνος αναμονής των εγγραφέων.
  - Τα νήματα ανάγνωσης αντιμετώπισαν καθυστερήσεις.

Γραφήματα

Στο παρακάτω γράφημα παρουσιάζεται η σύγκριση των μέσων χρόνων εκτέλεσης για τις διαφορετικές πολιτικές προτεραιότητας και αριθμούς νημάτων:



Γράφημα 4: Μέσος χρόνος εκτέλεσης για λειτουργίες ανάγνωσης και εγγραφής ανά πολιτική προτεραιότητας.

Συμπεράσματα

- Η χρήση κλειδωμάτων ανάγνωσης-εγγραφής εξασφαλίζει αποτελεσματική πρόσβαση σε κοινόχρηστους πόρους σε πολυνηματικά περιβάλλοντα.
- Η επιλογή της πολιτικής προτεραιότητας επηρεάζει άμεσα την απόδοση και την ισορροπία του συστήματος.
- Η προτεραιότητα στα νήματα ανάγνωσης είναι κατάλληλη για περιπτώσεις με περισσότερες λειτουργίες **Member()**, ενώ η προτεραιότητα στα νήματα εγγραφής εξυπηρετεί εφαρμογές με έντονες τροποποιήσεις δεδομένων.

Άσκηση 1.5

Εισαγωγή

Η παρούσα εργασία αφορά την πειραματική ανάλυση και σύγκριση τριών διαφορετικών υλοποιήσεων φράγματος με τη χρήση της βιβλιοθήκης `Pthreads`:

- Υλοποίηση φράγματος με χρήση της `pthread_barrier`.
- Υλοποίηση φράγματος βασισμένη σε κλείδωμα και μεταβλητές συνθήκης (`pthread_mutex` και `pthread_cond`).

- Υλοποίηση φράγματος με αναμονή σε εγρήγορση (*busy waiting*) μέσω της τεχνικής *sense-reversal*.

Κάθε μία από αυτές τις υλοποιήσεις χρησιμοποιήθηκε για τον συγχρονισμό πολλαπλών νημάτων που εκτελούσαν έναν προκαθορισμένο αριθμό επαναλήψεων, όπως περιγράφεται στον ψευδοκώδικα της εκφώνησης.

Πειραματική Διαδικασία

Για την ανάλυση της απόδοσης, πραγματοποιήθηκαν δοκιμές με διαφορετικό αριθμό νημάτων και καταγράφηκαν οι χρόνοι εκτέλεσης. Οι παράμετροι του πειράματος ήταν:

- Αριθμός επαναλήψεων: **5**.
- Αριθμός νημάτων: **2, 4, 8, 16**.
- Κάθε νήμα εκτελούσε μια εργασία που περιλάμβανε προσομοίωση φόρτου και συγχρονισμό στο φράγμα.
- Ο χρόνος εκτέλεσης καταγράφηκε για κάθε περίπτωση.

Υλοποίηση

Οι τρεις υλοποιήσεις περιγράφονται παρακάτω:

Φράγμα με `pthread_barrier`: Η ενσωματωμένη υλοποίηση της `Pthreads` χρησιμοποιήθηκε για την επίτευξη συγχρονισμού, όπως φαίνεται στο πρόγραμμα `barrier_pthread.c`.

Φράγμα με κλείδωμα και μεταβλητές συνθήκης: Η δεύτερη προσέγγιση χρησιμοποιεί ένα `pthread_mutex` για αμοιβαίο αποκλεισμό και μια μεταβλητή συνθήκης `pthread_cond` για τον συγχρονισμό των νημάτων, όπως φαίνεται στο πρόγραμμα `barrier_mutex_cond.c`.

Φράγμα με *sense-reversal*: Η τρίτη υλοποίηση βασίζεται σε αναμονή σε εγρήγορση (*busy waiting*) και αξιοποιεί την τεχνική *sense-reversal*, επιτρέποντας την επαναχρησιμοποίηση του φράγματος, όπως φαίνεται στο πρόγραμμα `barrier_sense_reversal.c`.

Αποτελέσματα

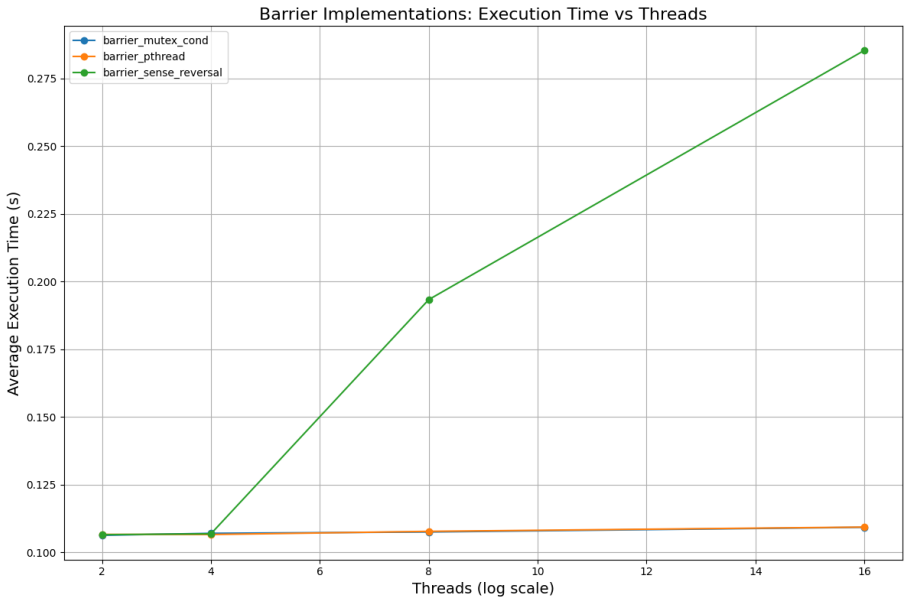
Τα αποτελέσματα δείχνουν ότι:

- Η χρήση της `pthread_barrier` προσφέρει σταθερή απόδοση, με χαμηλούς χρόνους εκτέλεσης, λόγω της ενσωματωμένης βελτιστοποίησης.

- Η υλοποίηση με κλείδωμα και μεταβλητές συνθήκης είχε μεγαλύτερη ευελιξία αλλά παρουσίασε ελαφρώς μεγαλύτερους χρόνους λόγω του μηχανισμού αναστολής (*wait*).
- Η υλοποίηση με *sense-reversal* εμφάνισε αυξημένη κατανάλωση CPU λόγω της αναμονής σε εγρήγορση.

Γραφήματα

Στο παρακάτω γράφημα παρουσιάζεται η σύγκριση των μέσων χρόνων εκτέλεσης για τις τρεις υλοποιήσεις φράγματος με διαφορετικό αριθμό νημάτων:



Γράφημα 5: Μέσοι χρόνοι εκτέλεσης για τις τρεις υλοποιήσεις φράγματος.

Συμπεράσματα

- Η ενσωματωμένη υλοποίηση `pthread_barrier` είναι κατάλληλη για εφαρμογές όπου προτιμάται απλότητα και απόδοση.
- Η υλοποίηση με κλείδωμα και μεταβλητές συνθήκης προσφέρει ευελιξία για πολύπλοκες ανάγκες συγχρονισμού.
- Η υλοποίηση με *sense-reversal* είναι χρήσιμη για συστήματα όπου η αποφυγή αναστολής είναι προτεραιότητα, αλλά αυξάνει τη χρήση CPU.