ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Συστήματα Παράλληλης και Κατανεμημένης Επεξεργασίας

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:11

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Περιεχόμενα

1. Σκοπός της άσκησης

- Συγχρονισμός POSIX νημάτων με αμοιβαίο αποκλεισμό.
- Νήματα POSIX και OpenMPI.
- Δημιουργία Makefile.

2. Παραδοτέα

(A) 2 ερωτήσεις

(C) 3 ασκήσεις

1 bonus

3. Αμοιβαίος αποκλεισμός

Θα κατασκευάσουμε ένα πρόγραμμα το οποίο θα χρησιμοποιεί νήματα για τον υπολογισμό ενός γινόμενου διανυσμάτων, χρησιμοποιώντας αμοιβαίο αποκλεισμό με mutex.

1. Δημιουργήστε το αρχείο c1.c
2. Τα δεδομένα θα είναι global για να είναι διαθέσιμα σε όλα τα νήματα. Κάθε νήμα θα εργάζεται πάνω σε άλλο κομμάτι των δεδομένων. Το κυρίως ***(main)*** θα περιμένει όλα τα νήματα να ολοκληρώσουν το έργο τους και στη συνέχεια θα εκτυπώνει τα αποτελέσματα. Ορίστε ένα struct:

```
typedef struct
 {
     Double    *a;
     Double    *b;
     Double    sum;
     Int  veclen;
 } DOTDATA;
```

3. Ορίστε με `#define NUMTHRDS 4`
4. Ορίστε με `#define VECLEN 100`
5. Ορίστε μια μεταβλητή `dotstr` που είναι τύπου DOTDATA
6. Ορίστε `pthread_t` σε πίνακα όσα είναι τα NUMTHRDS
7. Ορίστε μια μεταβλητή τύπου `pthread_mutex_t` με όνομα `mutexsum`
8. Στο κυρίως πρόγραμμα:
    1. ορίστε τους δείκτες προς διπλής ακρίβειας πίνακες `*a` , `*b`
    2. Να κάνετε δέσμευση χώρου για τους πίνακες a,b

```
/* Assign storage and initialize values */
a = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));
b = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));
```

3. Να αρχικοποιήσετε τους πίνακες `a` και `b` από `i=0` έως `i<VECLEN*NUMTHRDS` με `a[i]=1.0` , `b[i]=2*a[i]`
4. τοποθετήστε στο κοινό `struct` που είναι global τις παραπάνω τιμές:
   ```
   dotstr.veclen = VECLEN;
   dotstr.a = a;
   dotstr.b = b;
   dotstr.sum=0;
   ```
5. Αρχικοποιήσετε το mutex με τη συνάρτηση `pthread_mutex_init()`
6. Ορίστε όπως στο προηγούμενο εργαστήριο το attribute για να είναι `JOINABLE`

(A1) Τι σημαίνει για το νήμα, η παράμετρος αυτή;

7. Δημιουργήστε ένα βρόχο for για τον αριθμό των νημάτων (`t=0 όσο t<αριθμό threads`) που έχουμε, οποίος θα δημιουργεί με το `pthread_create()` τόσα νήματα όσα έχουμε ορίσει στο define. Όλα τα νήματα θα εκτελούν τη συνάρτηση `compute_multiplication()` με μια παράμετρο που είναι το `t`. Θα χρησιμοποιείται το attribute που έχουμε ορίσει ως `JOINABLE`. Προτείνεται να χρησιμοποιήσετε την τεχνική που έχει παρουσιαστεί σε προηγούμενη εργαστήριο, κατά την οποία δε στέλνετε τη διεύθυνση της παραμέτρου t, αλλά τη διεύθυνση του t κελιού του πίνακα που φέρει τη συγκεκριμένη τιμή. Δηλαδή, αν έχετε ορίσει
   `int *taskids[NUMTHRDS];` τότε μέσα στο loop, πριν ακριβώς από την κλήση της pthread_create() θα έχετε:
   ```
   taskids[t] = (int *) malloc(sizeof(int));
   *taskids[t] = t;
   ```
8. Μετά το βρόχο θα υπάρχει η εντολή `pthread_attr_destroy()` για την απομάκρυνση του attribute.
9. Στη συνέχεια θα υπάρχει ένας βρόχος for για τον αριθμό των νημάτων όπου θα εκτελείται για κάθε thread το `pthread_join()`
10. Έξω από αυτό το βρόχο θα εκτυπώνεται το αποτέλεσμα `dotstr.sum` ως πραγματικός αριθμός και θα γίνεται η εκκαθάριση της μνήμης των a, b (συνάρτηση `free()`).

(A2) Για ποιο λόγο κάνουμε εκκαθάριση της μνήμης; Τι θα συμβεί αν δεν το κάνουμε; Να τεκμηριώσετε την απάντησή σας με αναφορά σε κάποια πηγή.

11. Καταστρέψετε το mutex , αφού δε το χρειάζεστε.
12. Κάντε `exit()`

13. (**C1** η συνάρτηση) Η συνάρτηση `compute_multiplication()` που θα κάνει τον πολλαπλασιασμό θα αποτελείται από τα εξής στοιχεία:

    a. Τα στοιχεία που θα επεξεργαστεί το κάθε νήμα θα εξαρτώνται από την παράμετρο που έχουμε δώσει, και συγκεκριμένα θα ξεκινάνε από `t*dotstr.veclen` έως
    `t*dotstr.veclen + dotstr.veclen` .
    Η μεταβλητή t πρέπει να δηλωθεί ως ακέραιος τύπος (int).

    b. Θα δημιουργήσετε ένα βρόχο που θα κάνει για όλα τα `i` που ανήκουν μέσα στο εύρος που αντιστοιχεί το `sum=sum+a[i]*b[i]`.

    c. Προκειμένου να τοποθετήσουμε την τιμή sum στο struct που είναι κοινό για όλα θα πρέπει να το **κλειδώσουμε**, να γράψουμε και στη συνέχεια να το **ξεκλειδώσουμε**. Κλειδώστε το mutex με τη συνάρτηση `pthread_mutex_lock`

    d. Ενημερώστε το άθροισμα με `dotstr.sum += sum`

    e. Ξεκλειδώστε το sum για να είναι διαθέσιμο για να γράψει κάποιο άλλο νήμα με την κλήση της συνάρτησης:
    `pthread_mutex_unlock()`

    f. Εκτυπώστε ένα ενημερωτικό μήνυμα: `"Είμαι το threadID pthread_self() %u και υπολόγισα από το διάστημα έναρξής %d έως λήξης %d, το άθροισμα sum %f , και η νέα global τιμή είναι dotstr.sum %f"` . Αυτό το μήνυμα βοηθάει στην περίπτωση που δε λειτουργεί σωστά το πρόγραμμα, κάτι που οφείλεται συνήθως σε μη σωστή αρχικοποίηση των παραπάνω μεταβλητών. Όταν ολοκληρωθεί το πρόγραμμα, μπορείτε να κάνετε comment την παραπάνω γραμμή *(αλλά να την αφήσετε στον κώδικά σας)*.

    g. Το νήμα θα σταματήσει την εκτέλεση με την κλήση της `pthread_exit((void*) 0)`.

    h. Κάντε compile και επιβεβαιώστε την ορθή λειτουργία του παραπάνω προγράμματος. Αν το αποτέλεσμα είναι 800 και δεν εμφανίζονται warnings/errors τότε το πρόγραμμα λειτουργεί ορθά.

(C2) Να παραδώσετε το ανωτέρω αρχείο.

4. Δημιουργία Makefile (1)

Ακολουθώντας τις οδηγίες που εμφανίζονται στη σελίδα:

http://mrbook.org/tutorials/make/

κατασκευάστε ένα **Makefile** το οποίο θα χρησιμοποιείται για να κάνει compile το ανωτέρω αρχείο. Δηλαδή, αν ο χρήστης δώσει την εντολή **`make`** θα γίνεται compile και αν ο χρήστης δώσει **`make clean`** θα διαγράφονται τα αρχεία που έχουν δημιουργηθεί. Τοποθετήστε επίσης την ετικέτα execute, ώστε αν ο χρήστης δώσει **`make execute`** να εκτελείται η εφαρμογή.

*** **Προσοχή1:** Το αρχείο δεν έχει κατάληξη. Πρέπει να είναι απλώς `Makefile` και όχι Makefile.txt , Makefile.c, Makefile.kati .

*** **Προσοχή2:** Μέσα στο αρχείο Makefile δεν τοποθετούμε απόλυτες διαδρομές, αλλά σχετικές ως προς τον κατάλογο που βρίσκεται το Makefile, π.χ. δε θα έχουμε $CC /opt/intel/myprogram.c αλλά $CC myprogram.c

5. OpenMPI + Threads
(Προαιρετικά, +75% έξτρα βαθμός BONUS)

Τροποποιήστε το ανωτέρω κομμάτι κώδικα ώστε να χρησιμοποιεί ταυτόχρονα και threads και OpenMPI για εκτέλεση σε πολλαπλούς υπολογιστές και κάθε υπολογιστής να χρησιμοποιεί πολλαπλά threads.

- Δημιουργήστε το αρχείο **threads+mpi.c** με αντιγραφή του παραπάνω αρχείου.
- Τοποθετήστε τις συναρτήσεις αρχικοποίησης OpenMPI
- Κάθε κόμβος θα εκτελεί 2 threads. Θα έχουμε 5 MPI κόμβους.
- Να στέλνεται σε κάθε MPI κόμβο ο αριθμός των στοιχείων που θα επεξεργαστεί.
- Ο κάθε κόμβος να διαμοιράζει στα αντίστοιχα νήματα που έχει το συνολικό αριθμό των υπολογισμών που πρέπει να κάνει. Θα χρησιμοποιούνται mutex για την ασφαλή ενημέρωση της κοινής τιμής.
- Θα εκτελείται η MPI_REDUCE με MPI_SUM για να αθροίσει όλα τα μερικά αθροίσματα των κόμβων.
- Θα εκτυπώνεται το τελικό αποτέλεσμα από ένα κόμβο.
- Κάντε compile και επιβεβαιώστε την ορθή λειτουργία.

Το compile θα γίνει με το mpiCC με την παράμετρο –lpthread

(C3) Δημιουργία Makefile για το threads+mpi.c

Ακολουθώντας τις οδηγίες που εμφανίζονται στη σελίδα:

http://mrbook.org/tutorials/make/

κατασκευάστε ένα **Makefile** το οποίο θα χρησιμοποιείται για να κάνει compile το αρχείο threads-05.c. Δηλαδή, αν ο χρήστης δώσει την εντολή make θα γίνεται compile, αν ο χρήστης δώσει make clean θα διαγράφονται τα αρχεία που έχουν δημιουργηθεί. Τοποθετήστε επίσης την ετικέτα execute, ώστε αν ο χρήστης δώσει make execute να εκτελείται με MPI η εφαρμογή στο MTL cluster ή σε περίπτωση που δεν έχετε πρόσβαση σε αυτό (και ύστερα από έγκριση του διδάσκοντα), στην τοπική συστοιχία που έχετε δηλώσει στο αρχείο uowm-hosts *(το αρχείο με τα ονόματα των κόμβων σας)*.

Κατά τον προγραμματισμό με τα Posix threads, πολλές φορές απαιτείται ο αμοιβαίος αποκλεισμός των νημάτων. Αυτό γίνεται με τα mutex. Καλείστε να διαβάσετε τις σχετικές σελίδες βοήθειας των εντολών αρχικοποίησης, χρήσης, και καταστροφής των pthread_mutex. Βρείτε τις σελίδες με την εντολή man –k pthread_mutex . Τέλος, διαβάστε τη σελίδα βοήθειας για την ακύρωση ενός νήματος με την εντολή man pthread_cancel .