ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Συστήματα Παράλληλης και Κατανεμημένης Επεξεργασίας

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:20 **OpenMP**

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

- Προγραμματισμός συστήματος με διαμοιραζόμενη μνήμη με τη χρήση του OpenMP #1.
- Υβριδικός Προγραμματισμός OpenMP+MPI.

2. Παραδοτέα

(A) 4 ερωτήσεις

(C) 6 ασκήσεις

3. Πληροφορίες Περιβάλλοντος (Άσκηση 1)

(C1) Δημιουργήστε ένα πρόγραμμα με όνομα **c1.c** χρησιμοποιώντας τις ρουτίνες συστήματος που θα εκτυπώνει τα παρακάτω :

Number of processors =

Number of threads =

Max threads =

In parallel =

Dynamic threads enabled =

Nested parallelism supported =

4. Διαμοιρασμός Εργασίας (Άσκηση 2)

(C2) Δημιουργήστε ένα πρόγραμμα **c2.c** όπου αρχικοποιούμε 3 πίνακες 50 θέσεων ως εξής :

```
a[i] = i * 1.5;

b[i] = i + 22.35;

c[i] = d[i] = 0.0;
```

Αρχικά να εμφανίζει τον συνολικό αριθμό των threads που δημιουργούνται. Στη συνέχεια να γίνουν 2 section στα οποία το πρώτο να εκτελεί την πράξη `c[i] = a[i] + b[i]` και εμφανίζει τον πίνακα c και το δεύτερο να εκτελεί την πράξη `d[i] = a[i] * b[i]` και να εμφανίζει τον αντίστοιχο πίνακα. Επίσης να εμφανίζεται ο αριθμός του thread που μπαίνει σε κάθε section.

5. Fix bug (Άσκηση 3)

Το παρακάτω πρόγραμμα εμφανίζει λάθος αποτέλεσμα. Βρείτε το λάθος και διορθώστε το. Επιβεβαιώστε την ορθή λειτουργία (όνομα αρχείου **c3.c**)

(A1) Δώστε το screenshot εκτέλεσης.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 100
float a[N], b[N];
float calc ()
{
int i,tid;
float sum;

tid = omp_get_thread_num();
#pragma omp for reduction(+:sum)
for (i=0; i < N; i++)
      {
      sum = sum + (a[i]*b[i]);
      printf("  tid= %d i=%d\n",tid,i);
      }
}
int main (int argc, char *argv[]) {
int i;
float sum;

for (i=0; i < N; i++)
      a[i] = b[i] = 1.0 * i;
sum = 0.0;

#pragma omp parallel shared(sum)
      calc();

printf("Sum = %f\n",sum);
}
```

6. Fix bug (Άσκηση 4)

Το παρακάτω πρόγραμμα όταν εκτελείται προκαλεί segmentation fault. Βρείτε το λάθος και διορθώστε το. Επιβεβαιώστε την ορθή λειτουργία (όνομα αρχείου **c4.c**)

(A2) Δώστε το screenshot εκτέλεσης.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 1048
int main (int argc, char *argv[])
     {
     int nthreads, tid, i, j;
     double a[N][N];
/* Fork a team of threads with explicit variable scoping */
     #pragma omp parallel shared(nthreads)
private(i,j,tid,a)
          {
 /* Obtain/print thread info */
     tid = omp_get_thread_num();

     if (tid == 0)
          {
          nthreads = omp_get_num_threads();
          printf("Number of threads = %d\n", nthreads);
     }
     printf("Thread %d starting...\n", tid);

/* Each thread works on its own private copy of the array */
     for (i=0; i<N; i++)
          for (j=0; j<N; j++)
          a[i][j] = tid + i + j;

/* For confirmation */
     printf("Thread %d done. Last element= %f\n",tid,a[N-
1][N-1]);
     }  /* All threads join master thread and disband */
}
```

7. Πολλαπλασιασμός πίνακα με διάνυσμα (Άσκηση 5)

Να κατασκευάσετε το πρόγραμμα **c5_serial.c** το οποίο υπολογίζει το γινόμενο ενός πίνακα με ένα διάνυσμα ΣΕΙΡΙΑΚΑ . Ο πίνακας θα είναι

10*N στοιχείων διαστάσεις [10][N] και το διάνυσμα Ν θέσεων. Τοποθετήστε στην πρώτη γραμμή την εντολή `#define N 5`.

- Το πρόγραμμα θα ζητάει από το χρήση να πληκτρολογήσει Ν ακέραιες τιμές για το διάνυσμα.
- Να δημιουργήσετε τη συνάρτηση initialize_array_from_file() στην οποία θα τοποθετούνται αρχικές τιμές στον πίνακα, οι οποίες θα διαβάζονται από ένα αρχείο με όνομα array.dat
- Θα υπάρχει η κλήση της συνάρτησης print_results() η οποία θα εκτυπώνει τον πίνακα, το διάνυσμα και το τελικό αποτέλεσμα.

(A3) Χρονομετρήστε το σειριακό πρόγραμμα.

(C5) Αντιγράψτε το προηγούμενο αρχείο, στο αρχείο **c5.c** και τροποποιήστε το ώστε να γίνει παράλληλο.

(A4) Χρονομετρήστε το παράλληλο πρόγραμμα

8. MPI + OpenMP (Άσκηση 6)

Η χρήση στο ίδιο πρόγραμμα διαφορετικών μοντέλων παράλληλου προγραμματισμού με τέτοιο τρόπο ώστε να επωφεληθούμε από τα πλεονεκτήματα και να απαλύνουμε τα μειονεκτήματα των διαφορετικών μοντέλων. Εδώ αναφερόμαστε στο μοντέλο πολυνηματικού προγραμματισμού που εφαρμόζεται συνήθως σε συστήματα μοιραζόμενης μνήμης *(OpenMP)* και στο μοντέλο μεταβίβασης μηνυμάτων που συνήθως έχει εφαρμογή σε συστήματα κατανεμημένης μνήμης *(MPI)*.

Η εισαγωγή του MPI σε εφαρμογές OpenMP επιτρέπει την εκτέλεσή τους σε συστοιχίες από Symmetric Multi Processores (SMPs) ή/και multicore υπολογιστές.

- Η εισαγωγή του OpenMP σε εφαρμογές MPI βοηθά στην αποδοτική χρήση των SMP ή/και multicore υπολογιστών μιας συστοιχίας. Είναι γεγονός ότι το MPI αναγνωρίζει τους πυρήνες ως διαφορετικούς επεξεργαστές που επικοινωνούν μέσω ουρών μηνυμάτων, αλλά μπορεί να διαχειριστεί μόνο στατικές διεργασίες και όχι δυναμικά μεταβαλλόμενα νήματα.
- Η εξ' αρχής σχεδίαση ενός προγράμματος με χρήση MPI και OpenMP βοηθά στη βελτίωση της απόδοσης και της κλιμάκωσης της εφαρμογής.
- Το MPI αναλαμβάνει τη διανομή, συλλογή και επικοινωνία δεδομένων μεταξύ των υπολογιστών της συστοιχίας.

- Το OpenMP αναλαμβάνει το διαμοιρασμό της εργασίας και των δεδομένων σε κάθε ένα υπολογιστή της συστοιχίας.
- Έχουμε δύο επίπεδα παραλληλισμού:
  - κατανομή εργασίας/δεδομένων στους πολυπύρηνους/SMP υπολογιστές της συστοιχίας.
  - κατανομή της εργασίας/δεδομένων στους πυρήνες/CPUs κάθε υπολογιστή της συστοιχίας.

9. MPI

- Είναι το de facto πρότυπο προγραμματισμού συστημάτων κατανεμημένης μνήμης με μεταβίβαση μηνυμάτων.
- Διαθέτει υλοποιήσεις σε πολλές γλώσσες, αρχιτεκτονικές και τοπολογίες.
- Χρησιμοποιεί το μοντέλο ανάπτυξης κώδικα Single Programm Multiple Data *(SPMD)* που διευκολύνει το προγραμματισμό.
- Η επικοινωνία είναι ρητή και μέσω αυτής επιτυγχάνεται και ο συγχρονισμός μεταξύ υπολογιστών.
- Δημιουργεί μια διεργασία ανά επεξεργαστή, η οποία 'ζει' καθ' όλη τη διάρκεια της εκτέλεσης και διαχειρίζεται τα δεδομένα.

10. OpenMP

- Είναι το de facto πρότυπο προγραμματισμού μοιραζόμενης μνήμης με χρήση νημάτων.
- Υποστηρίζεται από όλους τους κύριους μεταγλωττιστές C/C++, Fortan *(υπάρχει και σε Java).*
- Χρησιμοποιεί compiler directives που διευκολύνουν το προγραμματισμό.
- Η επικοινωνία και ο συγχρονισμός επιτυγχάνονται έμμεσα, μέσω της πρόσβασης στη μοιραζόμενη μνήμη.,
- Δημιουργεί μια διεργασία *(Master Thread)*, η οποία 'ζει' καθ' όλη τη διάρκεια της εκτέλεσης και δημιουργεί / τερματίζει νήματα τα οποία προσπελαύνουν κοινά δεδομένα.

Παράδειγμα

```
#include <omp.h>
#include "mpi.h"
#include <stdio.h>
#define _NUM_THREADS 4

/*  each MPI process spawns a distinct OpenMP master thread
*  so limit the number of MPI processes to one per node
*/
int main (int argc, char *argv[]) {
int p,my_rank,c;
/* set number of threads to spawn */
omp_set_num_threads(_NUM_THREADS);

/* initialize MPI stuff */
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD,&p);
MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);

/* the following is OpenMP code executed by each MPI process
 * in practice it should do something useful
 */

#pragma omp parallel reduction(+:c)
{
  c = omp_get_num_threads();
}

/* expect a number to get printed for each MPI process */
printf("%d\n",c);

/* finalize MPI */
MPI_Finalize();
return 0;
}
```

Μεταγλώττιση

```
$ mpicc -o hello -fopenmp hello.c
$ mpirun -n 5 ./hello
```

- Το MPI αναλαμβάνει την επικοινωνία και το συγχρονισμό, ενώ το OpenMP τους υπολογισμούς.
- Σημεία που χρειάζονται προσοχή:
  - Ο λόγος υπολογισμού προς επικοινωνία πρέπει να είναι όσο το δυνατό υψηλός
  - Το OpenMP πρέπει να κάνει όσο το δυνατό μεγαλύτερη χρήση των επεξεργασιών ανά υπολογιστή.
  - Το MPI είναι πιο αποδοτικό όταν εκτελεί σχετικά συγκεντρωμένη και 'βαριά' επικοινωνία.

(C6) Να υπολογιστεί το π σε αρχείο με όνομα **c6.c** χρησιμοποιώντας **MPI+openMP**.

Δίνονται παρακάτω οι οδηγίες

- Το MPI δημιουργεί μια διεργασία ανά υπολογιστή. Συνολικά δημιουργούνται numprocs διεργασίες
- Η διεργασία αυτή λειτουργεί και ως Master OpenMP Thread για κάθε υπολογιστή.
- Κάθε Master Thread δημιουργεί OMP_NUM_THREADS νήματα
- Οι διεργασίες αριθμούνται από το myrank και τα νήματα από το ζεύγος myrank, mythread_id
- Η επικοινωνία και ο συγχρονισμός ελέγχονται από το Master Thread *(ή τη διεργασία MPI)* κάθε υπολογιστή.
- Εναλλακτικά η επικοινωνία θα μπορούσε να αποδοθεί στα νήματα, Ο υπολογισμό έτσι είναι πιο ευέλικτος και ασύγχρονος αλλά ο συγχρονισμός πιο δύσκολος.