ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Συστήματα Παράλληλης και Κατανεμημένης Επεξεργασίας

Ενότητα: Μέτρηση χρόνου τοίχου κατά POSIX

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.

Περιεχόμενα

1. Μέτρηση χρόνου τοίχου κατά POSIX

Κατά τη διαδικασία βελτιστοποίησης μιας εφαρμογής, είναι πολύ σημαντικό ο προγραμματιστής να μπορεί να μετράει το χρόνο εκτέλεσης πριν και μετά τη βελτιστοποίηση, ώστε να διαπιστώσει κατά πόσο οι τροποποιήσεις που έχει κάνει έχουν επιδράσει θετικά ή αρνητικά στις επιδόσεις. Υπάρχουν διάφοροι τρόποι μέτρησης χρόνου. Ο καλύτερος τρόπος μέτρησης, είναι η μέτρηση του πραγματικού χρόνου. Για να μετρηθεί ο πραγματικός χρόνος *(χρόνος ρολογιού, wall time)* που απαιτήθηκε χρησιμοποιείται η POSIX συνάρτηση `clock_gettime()` .

Για να χρησιμοποιηθεί αυτή η συνάρτηση πρέπει αρχικά να γίνει:
`#include <time.h>` στη συνέχεια να δηλωθούν οι μεταβλητές χρόνου αρχικής και τελικής τιμής με `struct timespec start, end;` να τοποθετηθεί μια φορά η κλήση της συνάρτησης μέτρησης πριν από το κρίσιμο τμήμα του προγράμματος με `clock_gettime (CLOCK_MONOTONIC, &start);` και μια φορά η κλήση της συνάρτησης μετά το κρίσιμο τμήμα `clock_gettime(CLOCK_MONOTONIC, &end);` Αυτό θα έχει ως αποτέλεσμα να καταγράφει η τιμή των δευτερολέπτων και νανο-δευτερολέπτων στις αντίστοιχες μεταβλητές.

Για να χρησιμοποιηθούν οι τιμές, θα πρέπει να γίνει η κατάλληλη αφαίρεση και η εκτύπωση των αποτελεσμάτων.

```
const int DAS_NANO_SECONDS_IN_SEC = 1000000000;
long timeElapsed_s = end.tv_sec - start.tv_sec;
long timeElapsed_n = end.tv_nsec - start.tv_nsec;
//If we have a negative number in timeElapsed_n , borrow a
carry from seconds
if ( timeElapsed_n < 0 ) {timeElapsed_n =
DAS_NANO_SECONDS_IN_SEC + timeElapsed_n; timeElapsed_s--;}
printf("Time: %ld.%09ld secs \n",timeElapsed_s,timeElapsed_n);
```

Συνήθως, εκτελείται το πρόγραμμα αρκετές φορές *(π.χ. 5-10 φορές)* και σημειώνεται η μέση τιμή. Αυτό γίνεται, γιατί η εργασία σε πολυ-προγραμματιστικά περιβάλλοντα μερικές φορές, εισάγει μη προβλεπόμενες καθυστερήσεις.

Η μέτρηση του χρόνου γίνεται στην κρίσιμη συνάρτηση και όχι σε όλο τον κώδικα. Για παράδειγμα στον παρακάτω ψευδοκώδικα ανίχνευσης κίνησης:

```
main(){
read_sequence();
xxx_motion_estimation();
motion_compensation();
snr();
}
```

θα πρέπει να τοποθετηθούν οι συναρτήσεις μέτρησης χρόνου πριν την κρίσιμη συνάρτηση που είναι η `xxx_motion_estimation();` ως εξής:

```
main(){
read_sequence();

clock_gettime(CLOCK_MONOTONIC, &start);
xxx_motion_estimation();
clock_gettime(CLOCK_MONOTONIC, &end);
...
motion_compensation();
snr();
}
```

Θα πρέπει να σημειωθεί ότι σε περίπτωση που χρησιμοποιούνται πολλαπλά νήματα ή πολλαπλές διεργασίες, τότε η μέτρηση του χρόνου θα πρέπει να γίνει από ένα μόνο νήμα.

Για παράδειγμα, στη περίπτωση που χρησιμοποιείται **openmp**, τότε θα πρέπει να προσδιορίσουμε ότι μόνο ένα νήμα θα εκτελέσει τη μέτρηση, είτε με το `#pragma omp master` είτε με τη συνάρτηση `omp_get_thread_num()`, όπως φαίνεται στα παρακάτω παραδείγματα.

```
(α τρόπος)
main(){
read_sequence();
#pragma omp parallel
{
#pragma omp master
clock_gettime(CLOCK_MONOTONIC, &start);
xxx_motion_estimation();
#pragma omp master
clock_gettime(CLOCK_MONOTONIC, &end);
...
}
```

```
(β τρόπος)
#pragma omp parallel
{
if(omp_get_thread_num()==0){clock_gettime(CLOCK_MONOTONIC,
&start); }
xxx_motion_estimation();
if(omp_get_thread_num()==0){clock_gettime(CLOCK_MONOTONIC,
&end); }
...
}
```