Ε-85: Ειδικά Θέματα Λογισμικού
Προγραμματισμός Συστημάτων Υψηλών Επιδόσεων

Χειμερινό Εξάμηνο 2009-10

«Προγραμματισμός με το μοντέλο OpenMP»

Παναγιώτης Χατζηδούκας
(Π.Δ. 407/80)

Ορολογία

- Μια ομάδα νημάτων στο OpenMP αποτελείται από το κύριο νήμα (master) και τους εργάτες (workers)
- Ένα παράλληλο τμήμα είναι ένα κομμάτι κώδικα που εκτελείται από όλα τα νήματα ταυτόχρονα
  - Το κύριο νήμα έχει πάντοτε αριθμό αναγνωριστικού = 0
  - Ο ορισμός του μεγέθους της ομάδας νημάτων, εφόσον επιτρέπεται, πραγματοποιείται πριν την εισαγωγή σε μια παράλληλη περιοχή
  - Οι παράλληλες περιοχές μπορεί να είναι εμφωλευμένες, αλλά η υποστήριξη της εκτέλεσης εσωτερικών επιπέδων εξαρτάται από την υλοποίηση
- Μια δομή διαμοίρασης έργου μοιράζει την εκτέλεση του κώδικα που περιέχεται στο παράλληλο τμήμα μεταξύ των μελών της ομάδας νημάτων
  - Επομένως, χωρίζει κατάλληλα τη δουλειά

Παράδειγμα 1

```
#pragma omp parallel shared(n,x,y) private(i) num_threads(4)
{
   #pragma omp for
   for (i=0; i<n; i++)
         x[i] += y[i];
} /*-- End of parallel region --*/
```


Παράδειγμα 2

```
void mxv_row(int m,int n,double *a,double *b,double *c) {
   int i, j;
   double sum;

   #pragma omp parallel for private(i, j, sum) shared(m, n, a, b, c)
   for (i=0; i<m; i++) {
         sum = 0.0;
         for (j=0; j<n; j++)
               sum += b[i*n+j]*c[j];
         a[i] = sum;
   } /*-- End of parallel for --*/
}
```


Παράδειγμα 3

```
#pragma omp parallel shared(n,a,b,c,d) private(i)
{
   #pragma omp for  nowait
   for (i=0; i<n-1; i++)
         b[i] = (a[i] + a[i+1])/2;
   #pragma omp for  nowait
   for (i=0; i<n; i++)
         d[i] = 1.0/c[i];

} /*-- End of parallel region --*/
```

(implied barrier)

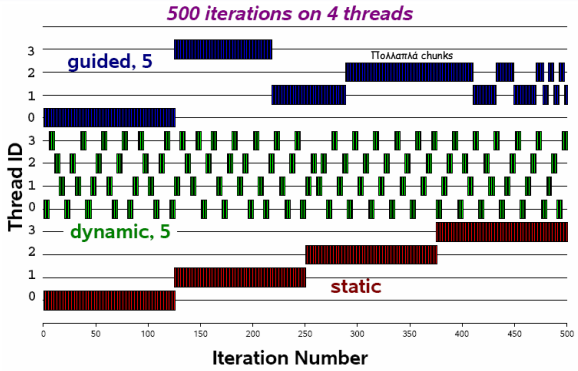
Εξισορρόπηση Εργασίας

- Αποτελεί βασικό παράγοντα απόδοσης
- Για συνηθισμένες λειτουργίες, π.χ. πρόσθεση διανυσμάτων, η εξισορρόπηση εργασίας δεν αποτελείται ζήτημα
- Για λιγότερο ομαλές λειτουργίες πρέπει να δοθεί ιδιαίτερη σημασία στην διαμοίραση της εργασίας μεταξύ των νημάτων
- Παράδειγμα μη ομαλών (irregular) λειτουργιών:
  - Αντιμετάθεση ενός πίνακα
  - Πολλαπλασιασμός τριγωνικών πινάκων
  - Παράλληλες αναζητήσεις σε μία διασυνδεδεμένη λίστα
- Για τέτοιες περιπτώσεις, η δομή διαμοίραση for χρησιμοποιείται με την οδηγία schedule που καθορίζει διάφορους αλγορίθμους δρομολόγησης των επαναλήψεων

Οδηγία schedule

- Χρήση
  - schedule (static | dynamic | guided [, chunk])
  - schedule (runtime)
- static [,chunk]
  - Διαμοιράζει τις επαναλήψεις, που έχουν χωριστεί σε τμήματα μεγέθους "chunk«, μεταξύ των νημάτων με κυκλικό τρόπο
  - Αν δεν ορίζεται το "chunk", αυτό ορίζεται κατά προσέγγιση ίσο με N/P και κάθε νήμα εκτελεί ένα τμήμα επαναλήψεων
- dynamic [,chunk]
  - Διαχωρίζει τις επαναλήψεις σε τμήματα μεγέθους "chunk“
  - Κάθε νήμα μόλις τελειώσει ένα τμήμα, παίρνει δυναμικά το επόμενο
- guided [,chunk]
  - Παρόμοια με dynamic, αλλά το μέγεθος του τμήματος μειώνεται εκθετικά
- runtime
  - Ο αλγόριθμος δρομολόγησης καθορίζεται κατά τον χρόνο εκτέλεσης ελέγχοντας τη μεταβλητή περιβάλλοντος OMP_SCHEDULE

Παράδειγμα



Συγχρονισμός

- Το OpenMP ορίζει τις ακόλουθες εντολές για την υποστήριξη συγχρονισμού:
  - atomic
  - critical section
  - barrier
  - flush
  - ordered
  - single (Στην πραγματικότητα είναι δομή διαμοίρασης έργου που περιλαμβάνει συγχρονισμό)
  - master (Στην ουσία δεν είναι εντολή συγχρονισμού)

Συγχρονισμός – critical

- Μόνο ένα νήμα μπορεί κάθε στιγμή να μπει στο critical section

```
#pragma omp parallel for private(b) shared(res)
for (i =0; i<niters; i++) {
        b = doit(i);
  #pragma omp critical
  {
        consume(b, &res);
  }
}
```


Συγχρονισμός – atomic

- Το atomic είναι ειδική περίπτωση του critical section που μπορεί να χρησιμοποιηθεί για ορισμένες απλές εντολές.
- Εφαρμόζεται μόνο για την ανανέωση μιας θέσης μνήμης (την ανανέωση του x στο ακόλουθο παράδειγμα)

```
#pragma omp parallel private(b)
{
        b = doit(i);
  #pragma omp atomic
        x = x + b;
}
```


Συγχρονισμός – barriers

- Barrier: Κάθε νήμα περιμένει να φτάσουν όλα τα υπόλοιπα νήματα πριν προχωρήσει.

```
#pragma omp parallel shared (A, B, C) private(id)
{
  id=omp_get_thread_num();
  A[id] = big_calc1(id);
  #pragma omp barrier
  #pragma omp for
  for(i=0;i<N;i++){
        C[i]=big_calc3(I,A);
  }     /* υπονοούμενο barrier */
  #pragma omp for nowait
  for(i=0;i<N;i++){
        B[i]=big_calc2(C, i);
  }     /* χωρίς barrier */
  A[id] = big_calc3(id);
} /* υπονοούμενο barrier */
```

Συγχρονισμός – ordered

- Η εντολή ordered επιβάλει την ακολουθιακή σειρά εκτέλεσης για ένα τμήμα

```
#pragma omp parallel private (res)
{
  #pragma omp for ordered
  for (i=0;i<N;i++){
        res = do_work(i);
  #pragma ordered
        printf("res = %d\n", res);
  }
}
```


Συγχρονισμός – master

- Η εντολή master χαρακτηρίζει ένα δομημένο block το οποίο εκτελείται μόνο από το νήμα - αρχηγό. Τα υπόλοιπα νήματα το αγνοούν (δεν υπάρχουν υπονοούμενα barriers ή flushes).

```
#pragma omp parallel private (tmp)
{
  do_many_things();
  #pragma omp master
  {
        exchange_boundaries();
  }
  #pragma barrier
  do_many_other_things();
}
```


Συγχρονισμός – single

- Η εντολή single χαρακτηρίζει ένα block κώδικα το οποίο εκτελείται από ένα μόνο νήμα.
- Υπονοούνται barrier και flush στο τέλος του single block

```
#pragma omp parallel private (tmp)
{
  do_many_things();
  #pragma omp single
  {
        exchange_boundaries();
  }
  do_many_other_things();
}
```


Συγχρονισμός – flush

- Η εντολή flush δηλώνει ένα σημείο στο οποίο το νήμα επιχειρεί να δημιουργήσει συνεπή εικόνα της μνήμης.
  - Όλες οι πράξεις μνήμης (αναγνώσεις και εγγραφές) που ορίζονται πριν από αυτό το σημείο πρέπει να ολοκληρωθούν.
  - Όλες οι πράξεις μνήμης (αναγνώσεις και εγγραφές) που ορίζονται μετά από αυτό το σημείο πρέπει να εκτελεστούν μετά το flush
  - Οι μεταβλητές σε registers ή write buffers πρέπει να εγγραφούν στη μνήμη
- Τα ορίσματα του flush ορίζουν ποιες μεταβλητές θα γίνουν flush. Αν δεν υπάρχουν ορίσματα όλες οι ορατές στο νήμα μεταβλητές γίνονται flush.

Υπονοούμενος Συγχρονισμός

- Barriers υπονοούνται μετά τις ακόλουθες εντολές OpenMP:
  - end parallel
  - end do (except when nowait is used)
  - end sections (except when nowait is used)
  - end critical
  - end single (except when nowait is used)
- Flush υπονοείται μετά τις ακόλουθες εντολές OpenMP:
  - barrier
  - critical, end critical
  - end do
  - end parallel
  - end sections
  - end single
  - ordered, end ordered

Συναρτήσεις Βιβλιοθήκης

- Συναρτήσεις locks
  - omp_init_lock(), omp_set_lock(), omp_unset_lock(), omp_test_lock()
- Συναρτήσεις περιβάλλοντος χρόνου εκτέλεσης:
  - Αλλαγή/Έλεγχος του αριθμού των νημάτων
    - omp_set_num_threads(), omp_get_num_threads(), omp_get_thread_num(), omp_get_max_threads()
  - Ενεργοποίηση/απενεργοποίηση εμφωλευμένου παραλληλισμού και dynamic mode
    - omp_set_nested(), omp_set_dynamic(),
    - omp_get_nested(), omp_get_dynamic()
  - Έλεγχος εκτέλεσης σε παράλληλο τμήμα
    - omp_in_parallel()
  - Αριθμός επεξεργαστών στο σύστημα
    - omp_num_procs()

Προστασία Πόρων με locks

```
   omp_lock_t lck;
   omp_init_lock(&lck);
#pragma omp parallel private (tmp)
{
   id = omp_get_thread_num();
   tmp = do_lots_of_work(id);
   omp_set_lock(&lck);
   printf("%d%d", id, tmp);
   omp_unset_lock(&lck);
}
```


Συναρτήσεις βιβλιοθήκης

- Για να ελεγχθεί ο αριθμός των νημάτων που εκτελούν ένα πρόγραμμα αρχικά απενεργοποιείται το dynamic mode και κατόπιν καθορίζεται ο αριθμός των νημάτων.

```
#include <omp.h>
void main()
{
      omp_set_dynamic(0);
      omp_set_num_threads(4);
      #pragma omp parallel
      {
            int id=omp_get_thread_num();
            do_lots_of_stuff(id);
      }
}
```


Μεταβλητές Περιβάλλοντος

- Έλεγχος scheduling των loops εφόσον έχει καθοριστεί "omp for schedule(RUNTIME)".
  - OMP_SCHEDULE "schedule[, chunk_size]"
- Καθορισμός του default αριθμού νημάτων.
  - OMP_NUM_THREADS int_literal
- Ενεργοποίηση/απενεργοποίηση dynamic mode
  - OMP_DYNAMIC TRUE || FALSE
- Ενεργοποίηση/απενεργοποίηση παράλληλης εκτέλεσης εμφωλευμένων παράλληλων τμημάτων
  - OMP_NESTED TRUE || FALSE

Χρήση

- Κώδικας OpenMP

```
#include <omp.h>
#include <stdio.h>
void main() {
   #pragma omp parallel
   {
     printf("Hello world from thread %d of %d\n",
            omp_get_thread_num(), omp_get_num_threads();
   }
}
```

- Παραγωγή εκτελέσιμου αρχείου για OMPi, GNU GCC 4.2, Intel compiler, Sun Studio compiler)

```
$ ompicc -o hello hello.c
$ gcc -fopenmp -o hello hello.c
$ icc -openmp -o hello hello.c
$ suncc -xopenmp=parallel -o hello hello.c
```


Χρήση

- Εκτέλεση

```
$ export OMP_NUM_THREADS=4
$ ./hello
Hello world from thread 0 of 4
Hello world from thread 2 of 4
Hello world from thread 1 of 4
Hello world from thread 3 of 4
$ export OMP_NUM_THREADS=1
$ ./hello
Hello world from thread 0 of 1
```


Βιβλιογραφία – Links

- OpenMP consortium
  - http://www.openmp.org
- OpenMP community
  - http://www.compunity.org
- Intel Compilers
  - http://developer.intel.com/software/products/compilers
- The OMPi Compiler
  - http://www.cs.uoi.gr/~ompi
- OpenMP Application Program Interface, Version 2.5, 2005
  - Διαθέσιμο και στη σελίδα του μαθήματος

Παράδειγμα 1

```
void a1(int n, float *a, float *b)
{
    int i;
#pragma omp parallel for
    for (i=1; i<n; i++) /* i is private by default */
        b[i] = (a[i] + a[i-1]) / 2.0;
}
```


Παράδειγμα 2

```
#include <stdio.h>
#include <omp.h>
int main()
{
    int x = 2;
    #pragma omp parallel num_threads(2) shared(x)
    {
        if (omp_get_thread_num() == 0) {
            x = 5;
        } else {
            printf("1: Thread# %d: x = %d\n", omp_get_thread_num(),x );
        }
        #pragma omp barrier

        if (omp_get_thread_num() == 0) {
            printf("2: Thread# %d: x = %d\n", omp_get_thread_num(),x );
        } else {
            printf("3: Thread# %d: x = %d\n", omp_get_thread_num(),x );
        }
    }
    return 0;
}
```


Παράδειγμα 3

```
#include <stdio.h>
int main()
{
    # ifdef _OPENMP
    printf("Compiled by an OpenMP-compliant implementation.\n");
    # endif
    return 0;
}
```


Παράδειγμα 4

```
#include <omp.h>

void sub(float *x, int npoints) {
    int iam, nt, ipoints, istart;
#pragma omp parallel private(iam,nt,ipoints,istart)
 {
    iam = omp_get_thread_num();
    nt = omp_get_num_threads();
    ipoints = npoints / nt; /* size of partition */
    istart = iam * ipoints; /* starting array index */
    if (iam == nt-1) /* last thread may do more */
    ipoints = npoints - istart;
    subdomain(x, istart, ipoints);
 }
}

void main() {
    float array[10000];
    sub(array, 10000);
}
```

```
void subdomain(float *x, int istart,
                int ipoints)
{
   int i;
   for (i = 0; i < ipoints; i++)
          x[istart+i] = 123.456;
}
```


Παράδειγμα 5

```
#include <omp.h>
int main()
{
    omp_set_dynamic(1);
#pragma omp parallel num_threads(10)
    {
        /* do work here – at most 10 threads */
    }
    return 0;
}
```


Παράδειγμα 6

```
#include <math.h>
void a8(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma omp parallel
    {
        #pragma omp for nowait
        for (i=1; i<n; i++)
            b[i] = (a[i] + a[i-1]) / 2.0;
        #pragma omp for nowait
        for (i=0; i<m; i++)
            y[i] = sqrt(z[i]);
    }
}
```

Παράδειγμα 7

```
void XAXIS();
void YAXIS();
void ZAXIS();

void a9()
{
    #pragma omp parallel sections
    {
        #pragma omp section
            XAXIS();
        #pragma omp section
            YAXIS();
        #pragma omp section
            ZAXIS();
    }
}
```


Παράδειγμα 8

```
#include <stdio.h>
void work1() {}
void work2() {}

void a10()
{
    #pragma omp parallel
    {
        #pragma omp single
            printf("Beginning work1.\n");

        work1();
        #pragma omp single
            printf("Finishing work1.\n");

        #pragma omp single nowait
            printf("Finished work1 and beginning work2.\n");
        work2();
    }
}
```


Παράδειγμα 9

```
int dequeue(float *a);
void work(int i, float *a);

void a13(float *x, float *y)
{
    int ix_next, iy_next;
    #pragma omp parallel shared(x, y) private(ix_next, iy_next)
    {
        #pragma omp critical (xaxis)
            ix_next = dequeue(x);
        work(ix_next, x);

        #pragma omp critical (yaxis)
            iy_next = dequeue(y);
        work(iy_next, y);
    }
}
```


Παράδειγμα 10

```
int counter = 0;
#pragma omp threadprivate(counter)

int increment_counter()
{
    counter++;
    return(counter);
}

int increment_counter_2()
{
    static int counter_2 = 0;
    #pragma omp threadprivate(counter_2)
    counter_2++;
    return(counter_2);
}
```


Παράδειγμα 11

```
void work(int i, int j) {}

void good_nesting(int n)
{
    int i, j;
    #pragma omp parallel default(shared)
    {
        #pragma omp for
        for (i=0; i<n; i++) {
            #pragma omp parallel shared(i, n)
            {
                #pragma omp for
                for (j=0; j < n; j++)
                    work(i, j);
            }
        }
    }
}
```