Παράλληλη Επεξεργασία

Εαρινό Εξάμηνο 2022-23
«OpenMP - III: χρονοπρογραμματισμός βρόχων»

Παναγιώτης Χατζηδούκας, Ευστράτιος Γαλλόπουλος

Περίληψη

- Χρονοπρογραμματισμός βρόχων στο OpenMP
- Διαθέσιμες πολιτικές και χαρακτηριστικά τους
- Σύνοψη - Επιλογή κατάλληλης πολιτικής
- Επιμέρους ερωτήματα
  - Επιβαρύνσεις χρονοπρογραμματισμού βρόχων
  - Τρόπος υλοποίησης πολιτικών
  - Συσχέτιση χρονοπρογραμματισμού και υλικού

Εντολή parallel for

- Δημιουργεί μία ομάδα νημάτων που εκτελεί το δομημένο τμήμα κώδικα που ακολουθεί
- Ο κώδικας που ακολουθεί την εντολή είναι ένας βρόχος
- Ο μεταγλωττιστής και το σύστημα χρόνου εκτέλεσης του μοντέλου OpenMP παραλληλοποιεί το βρόχο μοιράζοντας τις επαναλήψεις του βρόχου μεταξύ των νημάτων

```
#pragma omp parallel for
   for (int i = 0; i < n; i++)
     a[i] = b[i] + c[i];
```

Εντολή parallel for

- Ακολουθιακός κώδικας

```
sum = 0.0;
for (int i = 0; i <= n; i++)
    sum += f(i);
```

- Παράλληλοποίηση με ομάδα νημάτων και δομή διαμοίρασης εργασίας for

```
sum = 0.0;
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i <= n; i++)
    sum += f(i);
```

- Εξ' ορισμού πολιτική διαμοίρασης επαναλήψεων

```
sum = 0.0;
#pragma omp parallel for reduction(+:sum) schedule(static)
for (int i = 0; i <= n; i++)
    sum += f(i);
```

Συνάρτηση με μεταβλητό χρόνο εκτέλεσης

- Έστω η παρακάτω συνάρτηση:

```
double f(long i) {
   long j;
   long start = i*(i+1)/2;
   long finish = start + i;
   double return_val = 0.0;

   for (j = start; j <= finish; j++) {
      return_val += sin(j);
   }
   return return_val;
}  /* f */
```

- f(**i**): η μαθηματική συνάρτηση sin() καλείται **i** φορές.
  - Ο χρόνος εκτέλεσης της f(2*i) είναι περίπου διπλάσιος από τον χρόνο που απαιτείται για την εκτέλεση της f(i).

Χρονομετρήσεις

- Μέγεθος προβλήματος: n = 20000
- Ακολουθιακός κώδικας (χωρίς OpenMP)
  - χρόνος εκτέλεσης = 3.82 seconds
- Παράλληλος κώδικας
  - Εξ' ορισμού πολιτική διαμοίρασης επαναλήψεων (static)
  - χρόνος εκτέλεσης (2 νήματα) = 2.96 seconds

```
    sum = 0.0;
#pragma omp parallel for reduction(+:sum)
    for (int i = 0; i <= n; i++)
        sum += f(i);
```

- Χρονοβελτίωση = 1.29 (με μέγιστο το 2) :(

Πιθανή Λύση

- Κυκλική διαμοίραση των επαναλήψεων

Thread	Iterations
0	$0, n/t, 2n/t, \ldots$
1	$1, n/t+1, 2n/t+1, \ldots$
⋮	⋮
$t-1$	$t-1, n/t+t-1, 2n/t+t-1, \ldots$

- Υποστήριξη από το OpenMP

```
   sum = 0.0;
#pragma omp parallel for reduction(+:sum) schedule(static,1)
    for (int i = 0; i <= n; i++)
        sum += f(i);
```

- Απόδοση με 2 νήματα (n = 20000)
  - χρόνος εκτέλεσης 1.98 seconds
  - χρονοβελτίωση = 1.93 :)

schedule(type, chunksize)

- Πώς οι επαναλήψεις του βρόχου μοιράζονται μεταξύ των νημάτων της παράλληλης περιοχής
- Επιλογές (*type*)
  - **static**: η ανάθεση των επαναλήψεων στα νήματα γίνεται (αποφασίζεται) πριν την εκτέλεση του βρόχου.
  - **dynamic** ή **guided**: η ανάθεση των επαναλήψεων αποφασίζεται κατά την εκτέλεση του βρόχου.
  - **runtime**: η πολιτική καθορίζεται κατά την εκτέλεση του προγράμματος.
  - **auto**: ο μεταγλωττιστής και/ή το σύστημα χρόνου εκτέλεσης καθορίζει την πολιτική
- Η μεταβλητή *chunksize* είναι ένας θετικός ακέραιος

Στατική Πολιτική (static, chunksize)

- Οι επαναλήψεις χωρίζονται σε τμήματα μεγέθους *chunksize* και τα τμήματα μοιράζονται κυκλικά στα νήματα
- Αν το *chunksize* δεν ορίζεται, είναι ίσο με *N/P* (#επαναλήψεων/#νημάτων) και κάθε νήμα εκτελεί ένα τμήμα συνολικά

```
#pragma omp parallel for num_threads(3) schedule(static,1)
    for (int i = 0; i <= 11; i++)
        A[i] = do_work(i);
```

$$\begin{aligned} \text{Thread } 0: &\quad 0,3,6,9 \\ \text{Thread } 1: &\quad 1,4,7,10 \\ \text{Thread } 2: &\quad 2,5,8,11 \end{aligned}$$

Παραδείγματα Στατικής Πολιτικής

```
#pragma omp parallel for num_threads(3) schedule(static,2)
    for (int i = 0; i <= 11; i++)
        A[i] = do_work(i);
```

$$\begin{aligned} \text{Thread } 0: &\quad 0,1,6,7 \\ \text{Thread } 1: &\quad 2,3,8,9 \\ \text{Thread } 2: &\quad 4,5,10,11 \end{aligned}$$

```
#pragma omp parallel for num_threads(3) schedule(static,4)
    for (int i = 0; i <= 11; i++)
        A[i] = do_work(i);
```

$$\begin{aligned} \text{Thread } 0: &\quad 0,1,2,3 \\ \text{Thread } 1: &\quad 4,5,6,7 \\ \text{Thread } 2: &\quad 8,9,10,11 \end{aligned}$$

12/3 = 4, επομένως schedule(static) = schedule(static,4)

Δυναμική Πολιτική (dynamic)

```
#pragma omp parallel for num_threads(3) schedule(dynamic,4)
    for (int i = 0; i <= 11; i++)
        A[i] = do_work(i);
```

- Οι επαναλήψεις χωρίζονται ξανά σε συνεχόμενα τμήματα μεγέθους *chunksize*.
- Κάθε νήμα εκτελεί ένα τμήμα και μόλις τελειώσει ζητάει το επόμενο διαθέσιμο από το σύστημα χρόνου εκτέλεσης.
  - Μέχρι όλες οι επαναλήψεις να έχουν εκτελεστεί.
- Αν το *chunksize* δεν ορίζεται, είναι ίσο με 1
- Πιθανές αναθέσεις για *chunksize=4*

Thread 0: 0, 1, 2, 3
Thread 1: 4, 5, 6, 7
Thread 2: 8,9,10,11

Thread 0: 8,9,10,11
Thread 1: 4, 5, 6, 7
Thread 2: 0, 1, 2, 3

Πολιτική guided

```
#pragma omp parallel for num_threads(3) schedule(guided)
    for (int i = 0; i <= 11; i++)
        A[i] = do_work(i);
```

- Παρόμοια με τη δυναμική πολιτική, με το κάθε αδρανές νήμα να ζητά το αμέσως επόμενο διαθέσιμο τμήμα επαναλήψεων.
- Το μέγεθος του τμήματος μειώνεται κάθε φορά
  - chunksize = round(remaining iterations / #threads)
- Αν το *chunksize* δεν έχει οριστεί, το μέγεθος των τμημάτων μειώνεται μέχρι την τιμή 1.
- Διαφορετικά μειώνεται μέχρι την τιμή *chunksize*
  - το τελευταίο τμήμα μπορεί να είναι μικρότερο από *chunksize*.

Polychronopoulos, Constantine D., and David J. Kuck. "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers." *IEEE transactions on computers* 100.12 (1987): 1425-1439.

Πολιτική guided

Thread	Chunk	Size of Chunk	Remaining Iterations
0	1 – 5000	5000	4999
1	5001 – 7500	2500	2499
1	7501 – 8750	1250	1249
1	8751 – 9375	625	624
0	9376 – 9687	312	312
1	9688 – 9843	156	156
0	9844 – 9921	78	78
1	9922 – 9960	39	39
1	9961 – 9980	20	19
1	9981 – 9990	10	9
1	9991 – 9995	5	4
0	9996 – 9997	2	2
1	9998 – 9998	1	1
0	9999 – 9999	1	0

Παράδειγμα ανάθεσης επαναλήψεων (1-9999)
σε δύο νήματα σύμφωνα με την πολιτική guided

chunksize = round(remaining iterations / #threads)

Πολιτική runtime

```
#pragma omp parallel for num_threads(3) schedule(runtime)
    for (int i = 0; i <= 11; i++)
        A[i] = do_work(i);
```

- Το σύστημα χρόνου εκτέλεσης διαβάζει τη μεταβλητή περιβάλλοντος `OMP_SCHEDULE` για να καθορίσει την πολιτική κατά το χρόνο εκτέλεσης του προγράμματος.
- Η μεταβλητή `OMP_SCHEDULE` μπορεί να πάρει τιμές σύμφωνες με τις πολιτικές static, dynamic και guided.
- Παραδείγματα

```
$ export OMP_SCHEDULE="static,1"
$ export OMP_SCHEDULE="dynamic,4"
$ export OMP_SCHEDULE="guided"
```

Σύνοψη - Επιλογή Πολιτικής

- Το OpenMP μοιράζει αυτόματα τις επαναλήψεις των βρόχων
- Εξ' ορισμού οι επαναλήψεις χωρίζονται σε τμήματα ίσου μεγέθους, με ένα τμήμα για κάθε νήμα
- Επειδή η παραπάνω πολιτική δεν είναι πάντα η καλύτερη, το OpenMP προσφέρει μία ποικιλία επιλογών
- Αν οι επαναλήψεις απαιτούν τον ίδιο περίπου χρόνο, η στατική πολιτική είναι η καλύτερη *και έχει μικρότερη επιβάρυνση*

Σύνοψη - Επιλογή Πολιτικής

- Αν ο χρόνος κάθε επανάληψης διαφέρει σημαντικά, η δυναμική πολιτική (dynamic) είναι πιο αποδοτική
- Ο ρητός ορισμός του μεγέθους των τμήματων (*chunksize*) ή η χρήση της πολιτικής guided αποτελούν επιπλέον επιλογές
- Η επιλογή της καλύτερης πολιτικής εξαρτάται σημαντικά από την κατανόηση της συμπεριφοράς του βρόχου

Ερωτήματα

1. Επιβαρύνσεις χρονοπρογραμματισμού βρόχων
2. Τρόπος υλοποίησης πολιτικών
3. Συσχέτιση χρονοπρογραμματισμού επαναλήψεων και υλικού

Στόχος - κίνητρο: "Δεν πρέπει να μας αρκεί να γνωρίζουμε τη χρήση του OpenMP. Είναι σημαντικό να γνωρίζουμε πώς λειτουργεί και υλοποιείται εσωτερικά"

1. Επιβαρύνσεις Χρονοπρογραμματισμού

- Αρχικός σειριακός κώδικας

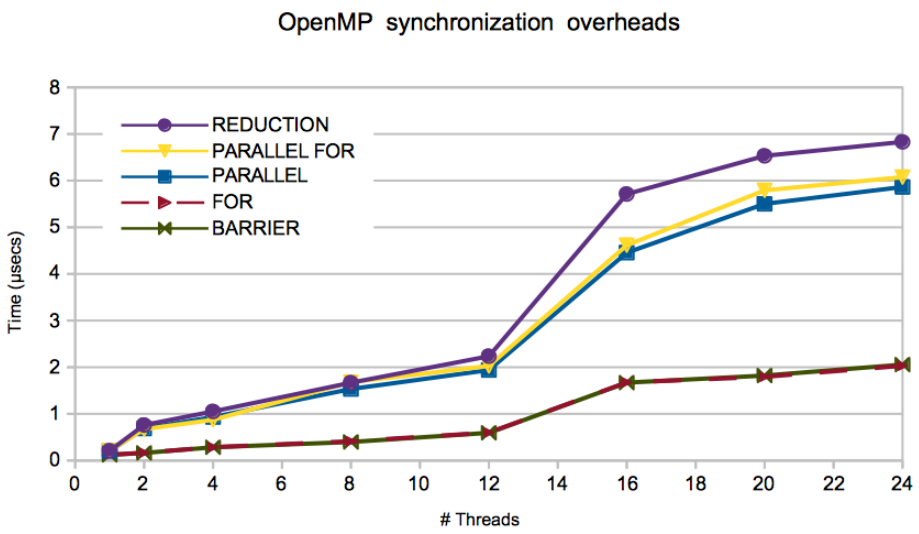
```
sum = 0.0;
for (int i = 0; i <= n; i++)
    sum += f(i);
```

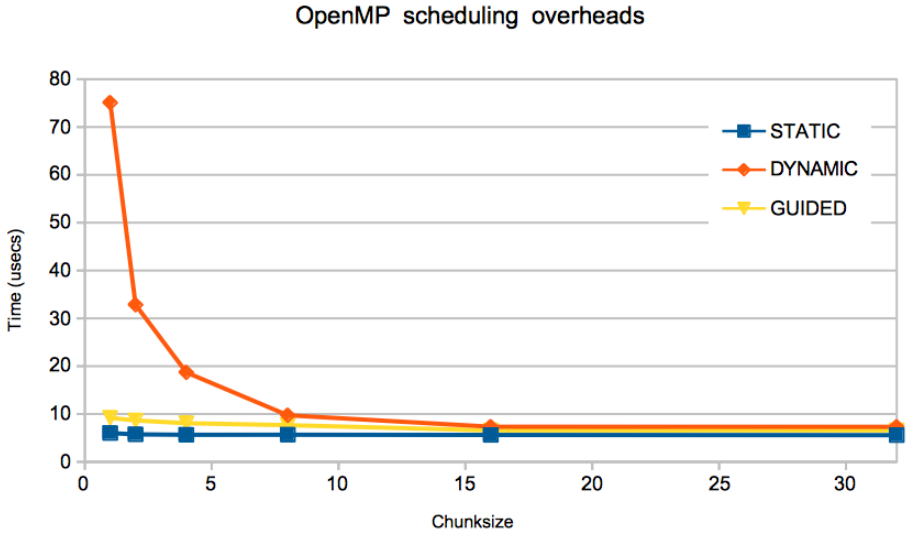
- Για n = 20000, χρόνος εκτέλεσης = 3.82 seconds
- Μέσος χρόνος ανά επανάληψη = 0.191 ms = 191 μs

- Πόσο είναι το κόστος του χρονοπρογραμματισμού και του OpenMP γενικότερα;
- Τα μετροπρογράμματα EPCC παρέχουν μετρήσεις των επιβαρύνσεων για όλες τις εντολές του OpenMP
  - συγχρονισμός, χρονοπρογραμματισμός βρόχων, μοντέλο εργασιών (tasking)
  - κώδικας: https://www.epcc.ed.ac.uk/

Τυπικές επιβαρύνσεις OpenMP

- Πολυεπεξεργαστικό σύστημα με 24 πυρήνες





- Οι επιβαρύνσεις είναι της τάξης των microseconds
- Τυπικό laptop με 2 νήματα
  - συνολική επιβάρυνση ~15 μs ανεξαρτήτου πολιτικής

2. Υλοποίηση Δυναμικής Πολιτικής

- Μας δίνεται ο παρακάτω κώδικας

```
#pragma omp parallel for schedule(dynamic,1)
    for (int i = 0; i < N; i++)
        A[i] = do_work(i);
```

- Πως μπορεί να υλοποιηθεί χωρίς τη χρήση οποιαδήποτε δομής διαμοίρασης του OpenMP?

Υλοποίηση Δυναμικής Πολιτικής

```
#pragma omp parallel for schedule(dynamic,1)
    for (int i = 0; i < N; i++)
        A[i] = do_work(i);
```

Μία πιθανή λύση

```
    int gi = 0;     // loop-index
#pragma omp parallel
    {
        int i;     // private value of the loop-index
        while (1)
        {
#pragma omp critical
            {
                i = gi++;  // get the next chunk (of size 1)
            }

            if (i >= N) break;     // necessary check
            A[i] = work(i);         // actual work
        }
    }
```

Γενική υλοποίηση της schedule(dynamic,chunksize)?

Υλοποίηση Δυναμικής Πολιτικής

- Κώδικας παραγόμενος από τον μεταφραστή OMPi

```
static void * _thrFunc1_(void * __me)
{
    struct __shvt__ {
        int (* N);
    };
    struct __shvt__ * _shvars = (struct __shvt__ *) ort_get_shared_vars(__me);
    int (* N) = _shvars->N;

    /* (l30) #pragma omp parallel  -- body moved below */
# 30 "injected_code"
    {
        /* #pragma omp for schedule(dynamic, 1) */
        int i;
        struct _ort_gdopt_ gdopt_;
        int niters_ = 0, iter_ = -1, fiter_, liter_ = -2;

        ort_entering_for(0, 0, &gdopt_);
        niters_ = ort_num_iters(1, (long) ((*N) - (0)), (long) 1, (int *) 0);
        while (ort_get_dynamic_chunk(niters_, 1, &fiter_, &liter_, (int *) 0, &gdopt_))
        {
            for (iter_ = fiter_, i = 0 + fiter_ * 1; iter_ < liter_; iter_++, i += 1)
# 33 "for.c"
                A[i] = work(i);
        }
    }
    ort_taskwait(2);
    return ((void *) 0);
}
```

3. Χρονοπρογραμματισμός και Υλικό

- Μας δίνεται ο παρακάτω κώδικας

```
#pragma omp parallel for schedule(static,1)
    for (int i = 0; i < N; i++)
        A[i] = do_tiny_work(i);
```

- Ερώτημα: πότε και γιατί η χρήση της πολιτικής (static,1) δεν είναι καλή ιδέα;

Χρονοπρογραμματισμός και Υλικό

```
#pragma omp parallel for schedule(static,1)
    for (int i = 0; i < N; i++)
        A[i] = do_tiny_work(i);
```

- Η απάντηση βρίσκεται στη σωστή διαχείριση της (κρυφής) μνήμης, που πρέπει να λαμβάνεται υπόψη πέρα από
  - τον κώδικα του χρήστη,
  - το περιβάλλον προγραμματισμού και εκτέλεσης του OpenMP,
  - το λειτουργικό σύστημα,
  - και τους επεξεργαστικούς πυρήνες.

Αναφορές

- OpenMP Specifications & Quick Reference Card
  - www.openmp.org
- An Introduction to Parallel Programming, Peter Pacheco
  - https://www.cs.usfca.edu/~peter/ipp/
- Parallel Programming in MPI and OpenMP, Victor Eijkhout
  - http://pages.tacc.utexas.edu/~eijkhout/pcse/html/index.html
  - https://bitbucket.org/VictorEijkhout/parallel-computing-book/src