



Κεφάλαιο 5

Προγραμματισμός
συστημάτων κοινόχρηστης
μνήμης με την OpenMP

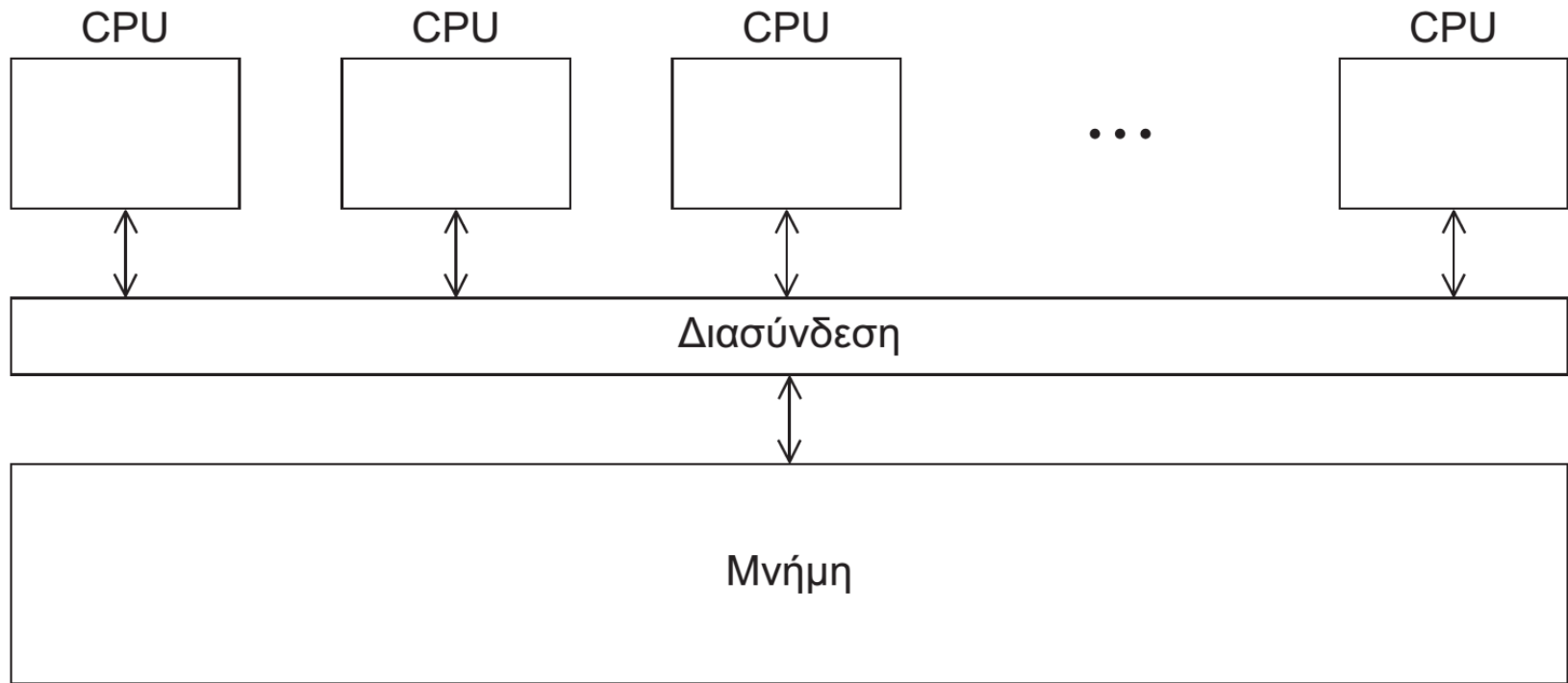
Περίγραμμα κεφαλαίου

- Γραφή προγραμμάτων που χρησιμοποιούν την OpenMP.
- Χρήση της OpenMP για την παραλληλοποίηση σειριακών βρόχων for με μικρές μόνο αλλαγές στον πηγαίο κώδικα.
- Παραλληλία εργασιών.
- Ρητός συγχρονισμός νημάτων.
- Τυπικά προβλήματα στον προγραμματισμό κοινόχρηστης μνήμης.

OpenMP

- Μια API για τον παράλληλο προγραμματισμό κοινόχρηστης μνήμης.
- MP = πολυεπεξεργασία (multiprocessing)
- Σχεδιασμένη για συστήματα στα οποία κάθε νήμα ή διεργασία επιτρέπεται να προσπελάζει ουσιαστικά όλη τη διαθέσιμη μνήμη.
- Το σύστημα θεωρείται ως μια συλλογή πυρήνων (ή CPU) καθένας από τους οποίους έχει πρόσβαση στην κύρια μνήμη.

Σύστημα κοινόχρηστης μνήμης



Οδηγίες pragma

- Ειδικές εντολές προς τον προεπεξεργαστή.
- Συνήθως προστίθενται σε ένα σύστημα για να επιτρέπουν συμπεριφορές που δεν αποτελούν μέρος της βασικής προδιαγραφής της C.
- Οι μεταγλωττιστές που δεν υποστηρίζουν τις οδηγίες pragma απλώς τις αγνοούν.

#pragma

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void Hello(void); /* Συνάρτηση νήματος */

int main(int argc, char* argv[]) {
    /* Λήψη πλήθους νημάτων από τη γραμμή διαταγών */
    int thread_count = strtol(argv[1], NULL, 10);
    # pragma omp parallel num_threads(thread_count)
    Hello();

    return 0;
} /* main */

void Hello(void) {
    int my_rank = omp_get_thread_num();
    int thread_count = omp_get_num_threads();
    printf("Χαιρετισμούς από το νήμα %d από %d\n", my_rank, thread_count);
} /* Hello */
```

```
gcc -g -Wall -fopenmp -o omp_hello omp_hello . c
```

```
./ omp_hello 4
```

εκτέλεση με 4 νήματα

μεταγλώττιση

πιθανά
αποτελέσματα

Χαιρετισμούς από το νήμα 0 από 4
Χαιρετισμούς από το νήμα 1 από 4
Χαιρετισμούς από το νήμα 2 από 4
Χαιρετισμούς από το νήμα 3 από 4

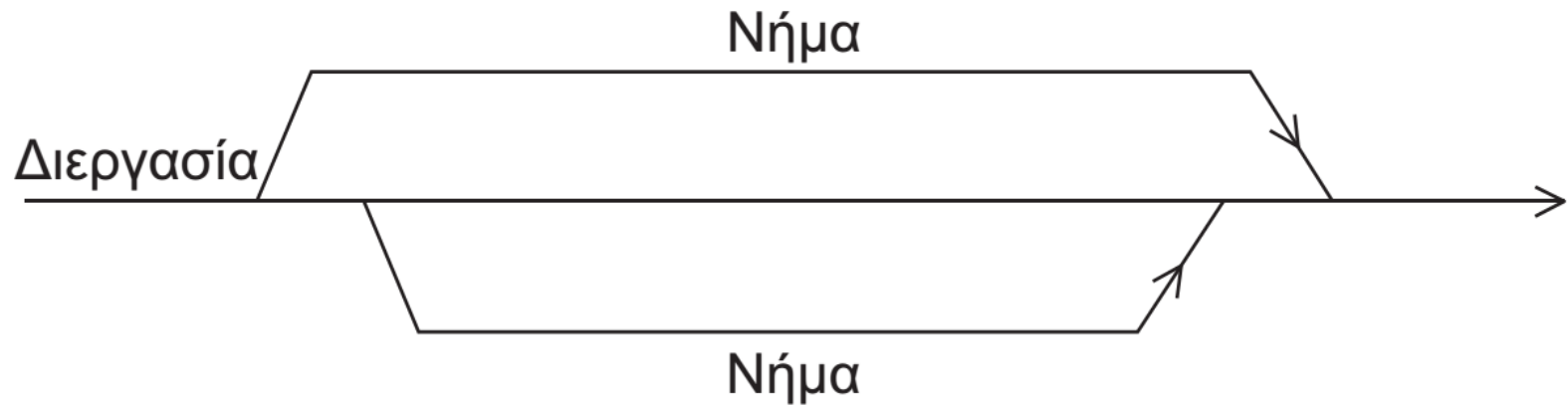
Χαιρετισμούς από το νήμα 3 από 4
Χαιρετισμούς από το νήμα 1 από 4
Χαιρετισμούς από το νήμα 2 από 4
Χαιρετισμούς από το νήμα 0 από 4

Χαιρετισμούς από το νήμα 1 από 4
Χαιρετισμούς από το νήμα 2 από 4
Χαιρετισμούς από το νήμα 0 από 4
Χαιρετισμούς από το νήμα 3 από 4

Οδηγίες `pragma` της OpenMP

- `# pragma omp parallel`
 - Η απλούστερη μορφή της οδηγίας παράλληλης εκτέλεσης `parallel`.
 - Το πλήθος των νημάτων τα οποία θα εκτελέσουν το δομημένο μπλοκ κώδικα που ακολουθεί την οδηγία καθορίζεται από το σύστημα χρόνου εκτέλεσης.

Διεργασία που διακλαδίζεται σε δύο νήματα τα οποία επανενώνονται



Όρος (clause)

- Κείμενο το οποίο τροποποιεί μια οδηγία.
- Στην οδηγία `parallel` μπορούμε να χρησιμοποιούμε τον όρο `num_threads`.
- Επιτρέπει στον προγραμματιστή να καθορίσει το πλήθος των νημάτων που θα εκτελέσουν το μπλοκ κώδικα το οποίο ακολουθεί την οδηγία.

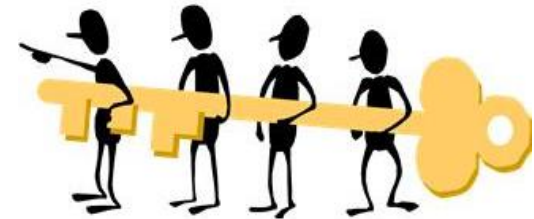
`# pragma omp parallel num_threads (thread_count)`

Πρέπει να επισημάνουμε ότι...

- Το πλήθος των νημάτων που μπορούν να ξεκινήσουν από ένα πρόγραμμα ενδέχεται να περιορίζεται από το σύστημα.
- Το πρότυπο OpenMP δεν εγγυάται ότι η παραπάνω εντολή θα ξεκινήσει πράγματι `thread_count` νήματα.
- Τα περισσότερα σύγχρονα συστήματα μπορούν να ξεκινούν εκατοντάδες ή και χιλιάδες νήματα.
- Αν δεν προσπαθήσουμε να ξεκινήσουμε υπερβολικά πολλά νήματα, σχεδόν πάντα θα παίρνουμε το πλήθος νημάτων που ζητήσαμε.

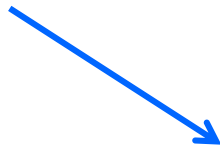
Λίγη ορολογία

- Σύμφωνα με την ορολογία της OpenMP, το σύνολο των νημάτων που εκτελούν το μπλοκ parallel –το αρχικό νήμα και τα νέα νήματα– ονομάζεται **ομάδα** (team), το αρχικό νήμα ονομάζεται **κύριο** (master), και τα πρόσθετα νήματα ονομάζονται **υπηρέτες** (slaves).



Αν ο μεταγλωττιστής δεν υποστηρίζει την OpenMP

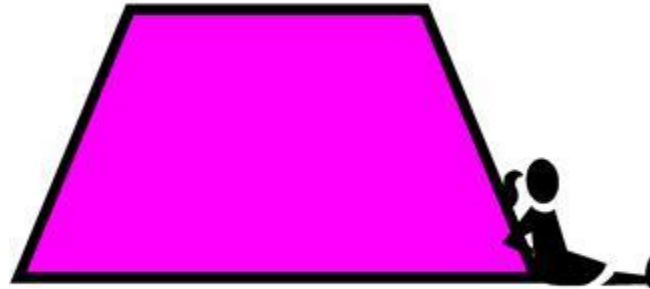
```
# include <omp.h>
```



```
#ifdef _OPENMP  
#  include <omp.h>  
#endif
```

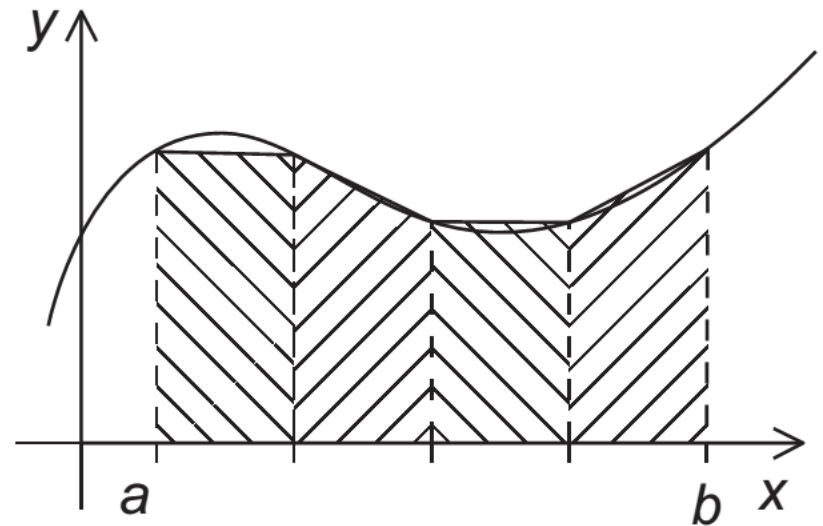
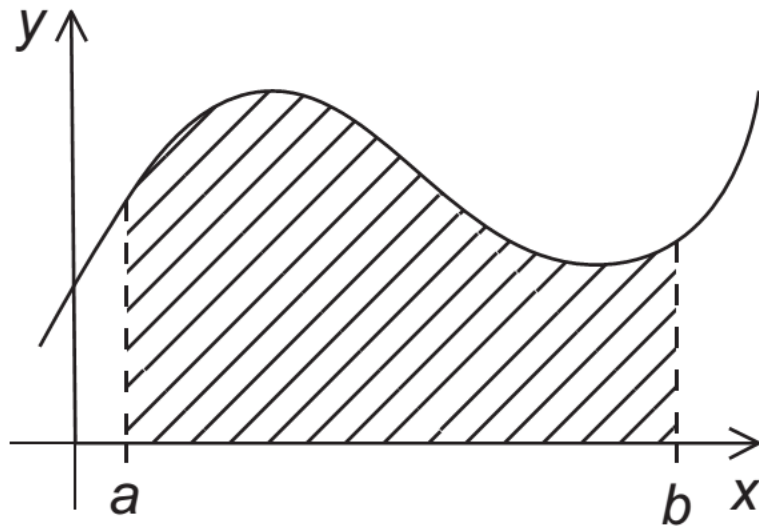
Αν ο μεταγλωττιστής δεν υποστηρίζει την OpenMP

```
# ifdef _OPENMP
    int my_rank = omp_get_thread_num ( );
    int thread_count = omp_get_num_threads ( );
# e l s e
    int my_rank = 0;
    int thread_count = 1;
# endif
```



Ο ΚΑΝΟΝΑΣ ΤΟΥ ΤΡΑΠΕΖΙΟΥ

Ο κανόνας του τραπεζίου



Σειριακός αλγόριθμος

```
/* Είσοδος:  $a$ ,  $b$ ,  $n$  */  
h = (b-a)/n;  
approx = (f(a) + f(b))/2.0;  
for (i = 1; i <= n-1; i++) {  
    x_i = a + i*h;  
    approx += f(x_i);  
}  
approx = h*approx;
```

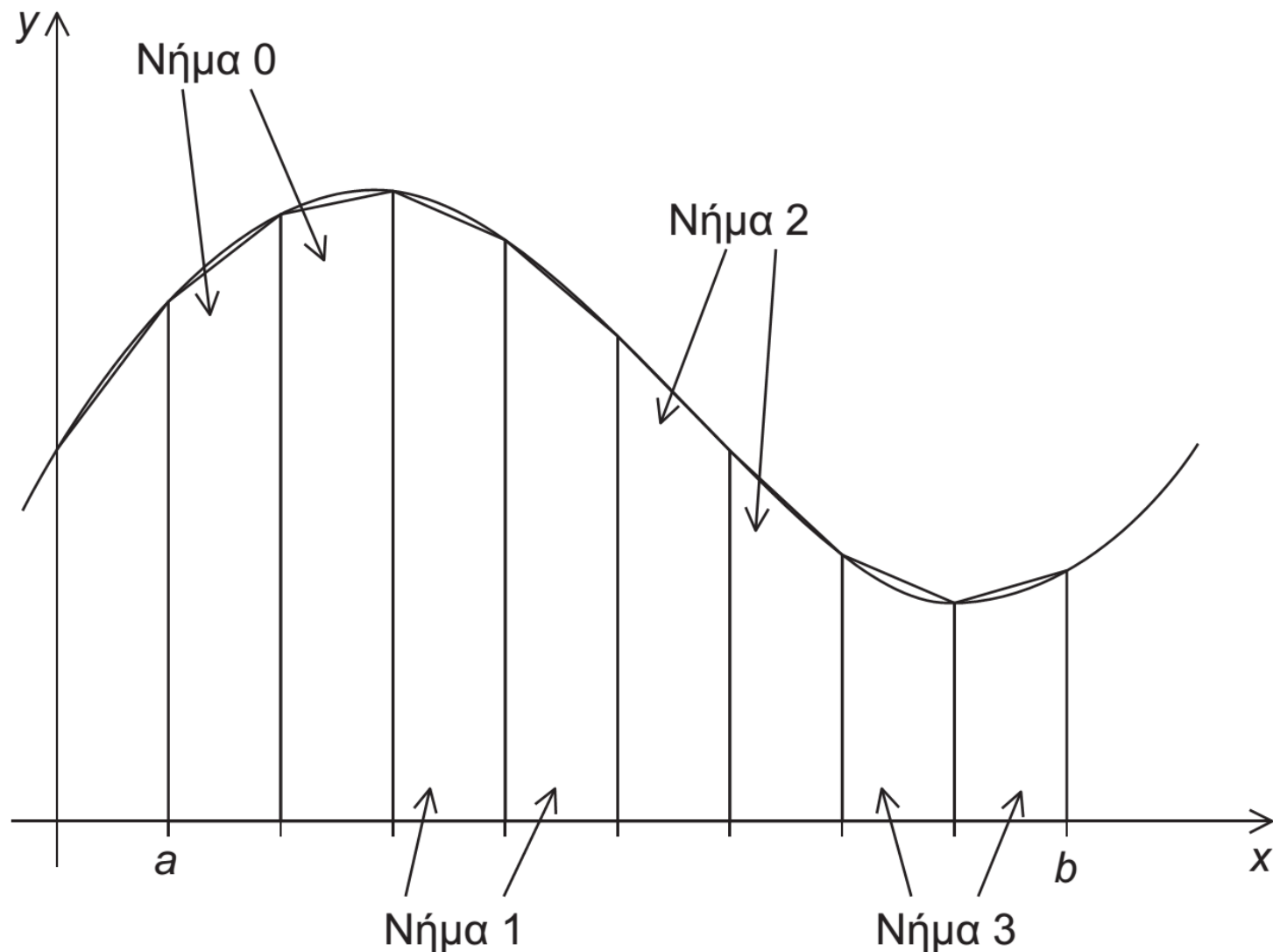
Μια πρώτη έκδοση του προγράμματος με την OpenMP

- 1) Αναγνωρίσαμε δύο τύπους εργασιών:
 - (α) Υπολογισμός των εμβαδών των μεμονωμένων τραπεζίων, και
 - (β) Άθροιση των εμβαδών των τραπεζίων.
- 2) Δεν χρειάζεται επικοινωνία μεταξύ των εργασιών του πρώτου συνόλου, αλλά κάθε μία από αυτές τις εργασίες επικοινωνεί με την εργασία 1(β).

Μια πρώτη έκδοση του προγράμματος με την OpenMP

- 3) Θεωρήσαμε ότι θα υπάρχουν πολύ περισσότερα στοιχειώδη τραπέζια απ' ό,τι πυρήνες.
- Γι' αυτό συναθροίσαμε τις εργασίες αναθέτοντας ένα μπλοκ συνεχόμενων τραπεζίων σε κάθε νήμα (και ένα νήμα σε κάθε πυρήνα).

Ανάθεση τραπεζίων σε νήματα



Χρόνος	Νήμα 0	Νήμα 1
0	φόρτωση <code>global_result = 0</code> σε καταχωρητή	ολοκλήρωση <code>my_result</code>
1	φόρτωση <code>my_result = 1</code> σε καταχωρητή	φόρτωση <code>global_result = 0</code> σε καταχωρητή
2	πρόσθεση <code>my_result</code> στην <code>global_result</code>	φόρτωση <code>my_result = 2</code> σε καταχωρητή
3	αποθήκευση <code>global_result = 1</code>	πρόσθεση <code>my_result</code> στην <code>global_result</code>
4		αποθήκευση <code>global_result = 2</code>

Το αποτέλεσμα θα είναι απρόβλεπτο αν δύο (ή περισσότερα) νήματα επιχειρήσουν να εκτελέσουν ταυτόχρονα την εντολή:

`global_result += my_result ;`



Αμοιβαίος αποκλεισμός

```
# pragma omp critical  
global_result += my_result ;
```

μόνο ένα νήμα επιτρέπεται να εκτελεί
κάθε φορά το δομημένο μπλοκ κώδικα

```

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void Trap(double a, double b, int n, double* global_result_p);

int main(int argc, char* argv[]) {
    double  global_result = 0.0;
    double  a, b;
    int      n;
    int      thread_count;

    thread_count = strtol(argv[1], NULL, 10);
    printf("Καταχωρίστε τις τιμές a, b, και n\n");
    scanf("%lf %lf %d", &a, &b, &n);
    # pragma omp parallel num_threads(thread_count)
    Trap(a, b, n, &global_result);

    printf("Με n = %d τραπέζια, η εκτίμησή μας\n", n);
    printf("για το ολοκλήρωμα από %f έως %f = %.14e\n",
        a, b, global_result);
    return 0;
} /* main */

```

```

void Trap(double a, double b, int n, double* global_result_p) {
    double h, x, my_result;
    double local_a, local_b;
    int i, local_n;
    int my_rank = omp_get_thread_num();
    int thread_count = omp_get_num_threads();

    h = (b-a)/n;
    local_n = n/thread_count;
    local_a = a + my_rank*local_n*h;
    local_b = local_a + local_n*h;
    my_result = (f(local_a) + f(local_b))/2.0;
    for (i = 1; i <= local_n-1; i++) {
        x = local_a + i*h;
        my_result += f(x);
    }
    my_result = my_result*h;

    # pragma omp critical
    *global_result_p += my_result;
} /* Trap */

```




ΕΜΒΕΛΕΙΑ (SCOPE) ΜΕΤΑΒΛΗΤΩΝ

Εμβέλεια

- Στον σειριακό προγραμματισμό, η εμβέλεια (scope) μιας μεταβλητής αποτελείται από τα τμήματα του προγράμματος στα οποία μπορεί να χρησιμοποιηθεί η μεταβλητή.
- Στην OpenMP, η εμβέλεια μιας μεταβλητής αναφέρεται στο σύνολο των νημάτων που μπορούν να προσπελάσουν τη μεταβλητή σε ένα μπλοκ parallel.

Εμβέλεια στην OpenMP

- Μια μεταβλητή που μπορεί να προσπελαστεί από όλα τα νήματα της ομάδας έχει **κοινόχρηστη** εμβέλεια (shared scope).
- Μια μεταβλητή που μπορεί να προσπελαστεί από ένα μόνο νήμα έχει **ιδιωτική** εμβέλεια (private scope).
- Η προεπιλεγμένη εμβέλεια για τις μεταβλητές που δηλώνονται πριν από ένα μπλοκ parallel είναι η **κοινόχρηστη**.





ΑΝΑΓΩΓΗ

Χρειαζόμαστε αυτή την πιο περίπλοκη παραλλαγή για να αθροίζουμε τα τοπικά αθροίσματα των νημάτων στο καθολικό άθροισμα *global_result*.

```
void Trap(double a, double b, int n, double* global_result_p);
```

Αν και θα προτιμούσαμε την παρακάτω.

```
double Trap(double a, double b, int n);
```



```
global_result = Trap(a, b, n);
```

Αν χρησιμοποιούσαμε όμως το παρακάτω πρωτότυπο, δεν θα χρειαζόμασταν κρίσιμο τμήμα!

```
double Local_trap(double a, double b, int n);
```

Αν τακτοποιήσουμε τον κώδικα όπως εδώ...


```
global_result = 0.0;
# pragma omp parallel num_threads(thread_count)
{
#     pragma omp critical
    global_result += Local_trap(double a, double b, int n);
}
```

... αναγκάζουμε τα νήματα να εκτελούνται σειριακά.

Μπορούμε να αποφύγουμε αυτό το πρόβλημα δηλώνοντας μια ιδιωτική μεταβλητή μέσα στο μπλοκ `parallel` και μεταφέροντας το κρίσιμο τμήμα μετά την κλήση της συνάρτησης.

```
global_result = 0.0;
# pragma omp parallel num_threads(thread_count)
{
    double my_result = 0.0    /* ιδιωτική */

    my_result += Local_trap(double a, double b, int n);
# pragma omp critical
    global_result += my_result;
}
```



Κάτι καλύτερο
θα μπορούμε
να κάνουμε.

Ούτε κι
εμένα.

Δεν μ'
αρέσει.

Τελεστές αναγωγής

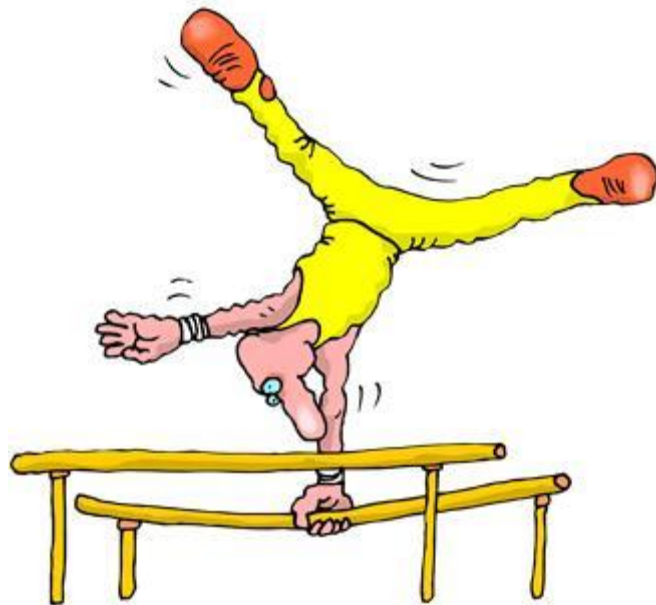
- **Τελεστής αναγωγής** (reduction operator) είναι μια διμελής πράξη (όπως η πρόσθεση ή ο πολλαπλασιασμός).
- **Αναγωγή** είναι ένας υπολογισμός που εφαρμόζει επανειλημμένα τον ίδιο τελεστή αναγωγής σε μια ακολουθία τελεστών για να προκύψει ένα μοναδικό αποτέλεσμα.
- Όλα τα ενδιάμεσα αποτελέσματα της πράξης πρέπει να αποθηκεύονται στην ίδια μεταβλητή: τη μεταβλητή αναγωγής.

Αρκεί να προσθέσουμε έναν όρο reduction στην οδηγία parallel.

reduction(<τελεστής>: <λίστα μεταβλητών>)

→ +, *, -, &, |, ^, &&, ||

```
global_result = 0.0;  
# pragma omp parallel num_threads(thread_count) \  
    reduction(+: global_result)  
global_result += Local_trap(double a, double b, int n);
```

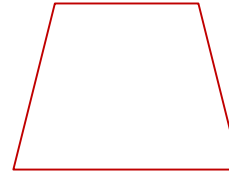


Η ΟΔΗΓΙΑ «PARALLEL FOR»

Parallel for

- Προκαλεί τη διακλάδωση μιας ομάδας νημάτων για την εκτέλεση του επόμενου δομημένου μπλοκ κώδικα.
- Όμως, το δομημένο μπλοκ που ακολουθεί την οδηγία parallel for πρέπει να είναι ένας βρόχος for
- Επίσης, με την οδηγία parallel for το σύστημα παραλληλοποιεί τον βρόχο μοιράζοντας τις επαναλήψεις του μεταξύ των νημάτων.

```
h = (b-a)/n;  
approx = (f(a) + f(b))/2.0;  
for (i = 1; i <= n-1; i++)  
    approx += f(a + i*h);  
approx = h*approx;
```



```
h = (b-a)/n;  
approx = (f(a) + f(b))/2.0;  
# pragma omp parallel for num_threads(thread count) \  
    reduction(+: approx)  
for (i = 1; i <= n-1; i++)  
    approx += f(a + i*h);  
approx = h*approx;
```

Έγκυρες μορφές παραλληλοποιήσιμων βρόχων for

```

for (μετρητής = αρχή ; μετρητής <= τέλος ; μετρητής += βήμα
    μετρητής < τέλος
    μετρητής -= βήμα
    μετρητής = μετρητής + βήμα
    μετρητής = βήμα + μετρητής
    μετρητής = μετρητής - βήμα
)

```

Περιορισμοί

- Η μεταβλητή μετρητής πρέπει να έχει τύπο ακεραίου ή δείκτη (π.χ. δεν μπορεί να είναι τύπου κινητής υποδιαστολής –float).
- Οι παραστάσεις αρχή, τέλος, και βήμα πρέπει να ανήκουν σε κάποιον συμβατό τύπο. Για παράδειγμα, αν η μετρητής είναι δείκτης, τότε η βήμα πρέπει να είναι ακεραίου τύπου.

Περιορισμοί

- Οι παραστάσεις αρχή, τέλος, και βήμα δεν πρέπει να μεταβάλλονται κατά την εκτέλεση του βρόχου.
- Κατά την εκτέλεση του βρόχου, η μεταβλητή μετρητής επιτρέπεται να τροποποιείται μόνο από την «παράσταση βηματικής αύξησης» της εντολής **for**.

Εξαρτήσεις δεδομένων

```
fibonacci[0] = fibonacci[1] = 1;  
for (i = 2; i < n; i++)  
    fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
```

έχουμε 2 νήματα

```
fibonacci[0] = fibonacci[1] = 1;  
# pragma omp parallel for num_threads(2)  
for (i = 2; i < n; i++)  
    fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
```

1 1 2 3 5 8 13 21 34 55

αυτό είναι το σωστό

1 1 2 3 5 8 0 0 0 0

κάποιες φορές όμως
παίρνουμε αυτό

Τι συνέβη;



1. Οι μεταγλωττιστές OpenMP δεν ελέγχουν για εξαρτήσεις μεταξύ των επαναλήψεων ενός βρόχου όταν τον παραλληλοποιούμε με την οδηγία `parallel for`.
2. Όταν τα αποτελέσματα μίας ή περισσότερων επαναλήψεων ενός βρόχου εξαρτώνται από άλλες επαναλήψεις, τότε, γενικά, ο βρόχος δεν μπορεί να παραλληλοποιηθεί σωστά από την OpenMP.

Αριθμητική προσέγγιση του π

$$\pi = 4 \left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right] = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

```
double factor = 1.0;
double sum = 0.0;
for (k = 0; k < n; k++) {
    sum += factor/(2*k+1);
    factor = -factor;
}
pi_approx = 4.0*sum;
```


1η λύση με την OpenMP

εξάρτηση μέσω βρόχου

```
double factor = 1.0;
double sum = 0.0;
# pragma omp parallel for num_threads(thread_count) \
    reduction(+:sum)
for (k = 0; k < n; k++) {
    sum += factor/(2*k+1);
    factor = -factor;
}
pi_approx = 4.0*sum;
```

2η λύση με την OpenMP

```
double sum = 0.0;
# pragma omp parallel for num_threads(thread_count) \
    reduction(+:sum) private(factor)
for (k = 0; k < n; k++) {
    if (k % 2 == 0)
        factor = 1.0;
    else
        factor = -1.0;
    sum += factor/(2*k+1);
}
```



Διασφαλίζει ότι η factor
έχει ιδιωτική εμβέλεια.

Ο όρος default

- Επιτρέπει στον προγραμματιστή να ορίζει ρητά την εμβέλεια κάθε μεταβλητής σε ένα μπλοκ.

default(none)

- Όταν χρησιμοποιούμε αυτόν τον όρο, ο μεταγλωττιστής απαιτεί να καθορίσουμε εμείς την εμβέλεια κάθε μεταβλητής που χρησιμοποιείται στο μπλοκ και έχει δηλωθεί έξω από αυτό.

Ο όρος default

```
double sum = 0.0;
# pragma omp parallel for num_threads(thread_count) \
    default(none) reduction(+:sum) private(k, factor) \
    shared(n)
for (k = 0; k < n; k++) {
    if (k % 2 == 0)
        factor = 1.0;
    else
        factor = -1.0;
    sum += factor/(2*k+1);
}
```



ΠΕΡΙΣΣΟΤΕΡΑ ΓΙΑ ΤΟΥΣ ΒΡΟΧΟΥΣ ΣΤΗΝ ΟΡΕΝΜΡ: ΤΑΞΙΝΟΜΗΣΗ

Ταξινόμηση φυσαλίδας (bubble sort)

```
for (list_length = n; list_length >= 2; list_length--)  
    for (i = 0; i < list_length-1; i++)  
        if (a[i] > a[i+1]) {  
            tmp = a[i];  
            a[i] = a[i+1];  
            a[i+1] = tmp;  
        }
```



Σειριακή ταξινόμηση με μετάθεση περιττού-αρτίου

```
for (phase = 0; phase < n; phase++)  
    if (phase % 2 == 0) { /* Άρτια φάση */  
        for (i = 1; i < n; i += 2)  
            if (a[i-1] > a[i]) Swap(&a[i-1], &a[i]);  
    } else { /* Περιττή φάση */  
        for (i = 1; i < n-1; i += 2)  
            if (a[i] > a[i+1]) Swap(&a[i], &a[i+1]);  
    }
```

Σειριακή ταξινόμηση με μετάθεση περιττού-αρτίου

Φάση	Αριθμοδείκτης						
	0		1		2		3
0	9	\leftrightarrow	7		8	\leftrightarrow	6
	7		9		6		8
1	7		9	\leftrightarrow	6		8
	7		6		9		8
2	7	\leftrightarrow	6		9	\leftrightarrow	8
	6		7		8		9
3	6		7	\leftrightarrow	8		9
	6		7		8		9

1η υλοποίηση της ταξινόμησης με μετάθεση περιττού-αρτίου στην OpenMP

```
for (phase = 0; phase < n; phase++) {
    if (phase % 2 == 0)
#       pragma omp parallel for num_threads(thread_count) \
        default(none) shared(a, n) private(i, tmp)
        for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
                tmp = a[i-1];
                a[i-1] = a[i];
                a[i] = tmp;
            }
        }
    else
#       pragma omp parallel for num_threads(thread_count) \
        default(none) shared(a, n) private(i, tmp)
        for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
                tmp = a[i+1];
                a[i+1] = a[i];
                a[i] = tmp;
            }
        }
}
```

2η υλοποίηση της ταξινόμησης με μετάθεση περιττού-αρτίου στην OpenMP

```
# pragma omp parallel num_threads(thread_count) \  
    default(none) shared(a, n) private(i, tmp, phase)  
    for (phase = 0; phase < n; phase++) {  
        if (phase % 2 == 0)  
#            pragma omp for  
            for (i = 1; i < n; i += 2) {  
                if (a[i-1] > a[i]) {  
                    tmp = a[i-1];  
                    a[i-1] = a[i];  
                    a[i] = tmp;  
                }  
            }  
        else  
#            pragma omp for  
            for (i = 1; i < n-1; i += 2) {  
                if (a[i] > a[i+1]) {  
                    tmp = a[i+1];  
                    a[i+1] = a[i];  
                    a[i] = tmp;  
                }  
            }  
    }  
}
```

Ταξινόμηση με μετάθεση περιττού-αρτίου,
με χρήση δύο οδηγιών `parallel for` και δύο οδηγιών `for`
(οι χρόνοι σε δευτερόλεπτα)

thread_count	1	2	3	4
Δύο οδηγίες <code>parallel for</code>	0,770	0,453	0,358	0,305
Δύο οδηγίες <code>for</code>	0,732	0,376	0,294	0,239





ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΒΡΟΧΩΝ

Θέλουμε να
παραλληλοποιήσουμε
αυτόν τον βρόχο.

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

Νήμα	Επαναλήψεις
0	0, t, 2t, ...
1	1, t + 1, 2t + 1, ...
⋮	⋮
t - 1	t - 1, t + t - 1, 2t + t - 1, ...

Ανάθεση εργασίας
με κυκλική διαμέριση.


```
double f(int i) {  
    int j, start = i*(i+1)/2, finish = start + i;  
    double return_val = 0.0;  
  
    for (j = start; j <= finish; j++) {  
        return_val += sin(j);  
    }  
    return return_val;  
} /* f */
```

Ο ορισμός μας για τη συνάρτηση f .

Αποτελέσματα

- Η $f(i)$ καλεί της συνάρτηση \sin , για τον υπολογισμό ενός ημιτόνου, i φορές.
- Θεωρούμε ότι ο χρόνος που χρειάζεται για την εκτέλεση της $f(2i)$ είναι περίπου ο διπλάσιος από αυτόν που χρειάζεται για την εκτέλεση της $f(i)$.
- $n = 10.000$
 - ένα νήμα
 - χρόνος εκτέλεσης = 3,67 δευτερόλεπτα.

Αποτελέσματα

- $n = 10.000$
 - δύο νήματα
 - προεπιλεγμένη αντιστοίχιση
 - χρόνος εκτέλεσης = 2,76 δευτερόλεπτα
 - επιτάχυνση = 1,33
- $n = 10.000$
 - δύο νήματα
 - κυκλική αντιστοίχιση
 - χρόνος εκτέλεσης = 1,84 δ/λεπτα
 - επιτάχυνση = 1,99



Ο όρος Schedule

- Προεπιλεγμένο χρονοδιάγραμμα:

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

- Κυκλικό χρονοδιάγραμμα:

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

schedule (type , chunksize)

- Ο τύπος (type) μπορεί να είναι:
 - **static** (στατικός): οι επαναλήψεις ανατίθενται στα νήματα πριν εκτελεστεί ο βρόχος.
 - **dynamic** (δυναμικός) ή **guided** (καθοδηγούμενος): οι επαναλήψεις ανατίθενται στα νήματα καθώς ο βρόχος εκτελείται.
 - **auto**: το χρονοδιάγραμμα καθορίζεται από τον μεταγλωττιστή ή και το σύστημα χρόνου εκτέλεσης.
 - **runtime**: το χρονοδιάγραμμα καθορίζεται κατά τον χρόνο εκτέλεσης.
- Το chunksize (μέγεθος μεριδίου) είναι ένας θετικός ακέραιος.

Χρονοδιαγράμματα τύπου static

δώδεκα επαναλήψεις, 0, 1, . . . , 11, και τρία νήματα

`schedule(static, 1)`

Νήμα 0: 0, 3, 6, 9

Νήμα 1: 1, 4, 7, 10

Νήμα 2: 2, 5, 8, 11

Χρονοδιαγράμματα τύπου static

δώδεκα επαναλήψεις, 0, 1, . . . , 11, και τρία νήματα

```
schedule(static, 2)
```

Νήμα 0: 0, 1, 6, 7

Νήμα 1: 2, 3, 8, 9

Νήμα 2: 4, 5, 10, 11

Χρονοδιαγράμματα τύπου static

δώδεκα επαναλήψεις, 0, 1, . . . , 11, και τρία νήματα

```
schedule(static, 4)
```

Νήμα 0: 0, 1, 2, 3

Νήμα 1: 4, 5, 6, 7

Νήμα 2: 8, 9, 10, 11

Χρονοδ/ματα τύπου dynamic

- Οι επαναλήψεις χωρίζονται πάλι σε μερίδια μεγέθους **chunksize** διαδοχικών επαναλήψεων.
- Κάθε νήμα εκτελεί ένα τέτοιο μερίδιο και, αφού το ολοκληρώσει, ζητάει να του ανατεθεί άλλο ένα από το σύστημα χρόνου εκτέλεσης.
- Η διαδικασία συνεχίζεται μέχρι να ολοκληρωθούν όλες οι επαναλήψεις.
- Το **chunksize** μπορεί να παραλειφθεί. Όταν παραλείπεται, θεωρείται ότι έχει την τιμή 1.

Χρονοδ/ματα τύπου guided

- Κάθε νήμα εκτελεί ένα μερίδιο επαναλήψεων και, όταν το ολοκληρώσει, ζητάει να του δοθεί κι άλλο.
- Ωστόσο, καθώς ολοκληρώνονται τα μερίδια στα χρονοδιαγράμματα τύπου guided, το μέγεθος των νέων μεριδίων ελαττώνεται.
- Αν δεν καθοριστεί το **chunksize**, το μέγεθος των μεριδίων ελαττώνεται σταδιακά μέχρι το 1.
- Αν καθοριστεί το **chunksize**, τότε το μέγεθος των μεριδίων ελαττώνεται σταδιακά μέχρι την τιμή **chunksize**, με μόνη εξαίρεση το τελευταίο μερίδιο, το οποίο επιτρέπεται να είναι μικρότερο.

Νήμα	Μερίδιο	Μέγεθος μεριδίου	Απομένουσες επαναλήψεις
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

Ανάθεση των επαναλήψεων 1–9999 στο πρόγραμμα για τον κανόνα του τραπεζίου, με χρονοδ/μα τύπου guided και 2 νήματα.

Χρονοδ/ματα τύπου runtime

- Το σύστημα βασίζεται στη μεταβλητή περιβάλλοντος `OMP_SCHEDULE` για να προσδιορίσει τον τρόπο χρονοπρογραμματισμού του βρόχου κατά τον χρόνο εκτέλεσης.
- Η μεταβλητή περιβάλλοντος `OMP_SCHEDULE` μπορεί να πάρει οποιαδήποτε από τις τιμές για στατικά, δυναμικά, και καθοδηγούμενα χρονοδιαγράμματα.



ΠΑΡΑΓΩΓΟΙ ΚΑΙ ΚΑΤΑΝΑΛΩΤΕΣ

- Μπορούμε να τις θεωρήσουμε ως αφηρημένη αναπαράσταση μιας σειράς πελατών οι οποίοι περιμένουν να πληρώσουν για τα προϊόντα που αγόρασαν στο ταμείο ενός σούπερ μάρκετ.
- Μια δομή δεδομένων χρήσιμη για πολλές πολυνηματικές εφαρμογές.
- Π.χ., έστω ότι έχουμε πολλά νήματα-«παραγωγούς» και πολλά νήματα-«καταναλωτές».
 - Τα νήματα-παραγωγοί θα μπορούσαν να «παράγουν» αιτήματα για τη λήψη δεδομένων από έναν διακομιστή.
 - Τα νήματα-καταναλωτές θα «καταναλώνουν» τα αιτήματα βρίσκοντας ή παράγοντας τα απαιτούμενα δεδομένα.

Μεταβίβαση μηνυμάτων

- Αν κάθε νήμα διέθετε μια κοινόχρηστη ουρά μηνυμάτων, τότε όποιο νήμα ήθελε να «στείλει μήνυμα» σε ένα άλλο νήμα, θα μπορούσε απλώς να εισάγει το μήνυμά του στο τέλος της ουράς του νήματος προορισμού.
- Αντίστοιχα, ένα νήμα θα λάμβανε ένα μήνυμα εξάγοντάς το από την αρχή της ουράς μηνυμάτων του.

Μεταβίβαση μηνυμάτων

```
for (sent_msgs = 0; sent_msgs < send_max; sent_msgs ++ ) {  
    Send_msg();      /* Αποστολή μηνύματος */  
    Try_receive(); /* Απόπειρα λήψης μηνύματος */  
}  
while (!Done())      /* Όσο κάποια νήματα στέλνουν ακόμη... */  
    Try_receive(); /* ...απόπειρα λήψης μηνύματος */
```


Αποστολή μηνυμάτων

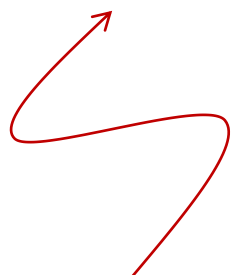
```
mesg = random();  
dest = random() % thread_count;  
# pragma omp critical  
Enqueue(queue, dest, my_rank, mesg);
```

Λήψη μηνυμάτων

```
if (queue_size == 0) return;  
else if (queue_size == 1)  
#    pragma omp critical  
    Dequeue(queue, &src, &mesg);  
else  
    Dequeue(queue, &src, &mesg);  
Print_message(src, mesg);
```

Αναγνώριση τερματισμού

```
queue_size = enqueued - dequeued;  
if (queue_size == 0 && done_sending == thread_count)  
    return TRUE;  
else  
    return FALSE;
```



κάθε νήμα αυξάνει την τιμή αυτής της μεταβλητής όταν ολοκληρώνει τον δικό του βρόχο for

Εκκίνηση (1)

- Όταν το πρόγραμμα ξεκινάει την εκτέλεσή του, ένα μοναδικό νήμα, το κύριο νήμα, θα διαβάσει τα ορίσματα της γραμμής διαταγών και θα δεσμεύει μνήμη για μια συστοιχία ουρών μηνυμάτων η οποία θα περιλαμβάνει από μία ουρά για κάθε νήμα.
- Η συστοιχία πρέπει να είναι κοινόχρηστη μεταξύ των νημάτων ώστε κάθε νήμα να μπορεί να στέλνει μηνύματα σε οποιοδήποτε άλλο, επομένως, οποιοδήποτε νήμα θα έχει τη δυνατότητα να εισάγει μηνύματα σε οποιαδήποτε από τις ουρές.

Εκκίνηση (2)

- Ένα ή περισσότερα νήματα ενδέχεται να ολοκληρώσουν τη δέσμευση μνήμης για τις ουρές τους πριν από τα άλλα νήματα
- Χρειαζόμαστε ένα ρητό φράγμα ώστε όποτε ένα νήμα φτάνει στο φράγμα να μπλοκάρεται μέχρι να φτάσουν στο φράγμα και τα υπόλοιπα νήματα της ομάδας.
- Μόνο αφού όλα τα νήματα της ομάδας φτάσουν στο φράγμα θα μπορούν, στη συνέχεια, να συνεχίσουν την εκτέλεσή τους.

```
# pragma omp barrier
```

Η οδηγία `atomic` (1)

- Σε αντίθεση με την οδηγία `critical`, προστατεύει μόνο κρίσιμα τμήματα που αποτελούνται από μία μοναδική εντολή ανάθεσης τιμής της C.

```
# pragma omp atomic
```

- Η εντολή επιτρέπεται να έχει μια από τις παρακάτω μορφές:

```
x <τελ>= <παράσταση>;
```

```
x++;
```

```
++x;
```

```
x--;
```

```
--x;
```

Η οδηγία atomic (2)

- <τελ> μπορεί να είναι ένας από τους διμελείς τελεστές `+, *, -, /, &, ^, |, <<, ή >>`
- Πολλοί επεξεργαστές παρέχουν μια ειδική εντολή (γλώσσας μηχανής) φόρτωσης-τροποποίησης-αποθήκευσης (load-modify-store)
- Ένα κρίσιμο τμήμα που εκτελεί μόνο μια φόρτωση-τροποποίηση-αποθήκευση μπορεί να προστατευτεί πολύ πιο αποδοτικά μέσω αυτής της ειδικής εντολής παρά από τις δομές που χρησιμοποιούνται για την προστασία πιο γενικών κρίσιμων τμημάτων.

Κρίσιμα τμήματα

- Η OpenMP μας παρέχει τη δυνατότητα να προσθέσουμε έναν όνομα στην οδηγία `critical`

`# pragma omp critical (όνομα)`

- Όταν χρησιμοποιούμε αυτή την επιλογή, δύο μπλοκ που προστατεύονται με οδηγίες `critical` οι οποίες έχουν διαφορετικά ονόματα μπορούν να εκτελούνται ταυτόχρονα.
- Τα ονόματα, όμως, καθορίζονται κατά τη μεταγλώττιση, και εμείς θέλουμε ένα διαφορετικό κρίσιμο τμήμα για την ουρά κάθε νήματος.

Κλειδώματα

- Ένα κλείδωμα αποτελείται από μια δομή δεδομένων και συναρτήσεις που επιτρέπουν στους προγραμματιστές να επιβάλλουν ρητά τον αμοιβαίο αποκλεισμό κατά την προσπέλαση ενός κρίσιμου τμήματος.



Κλειδώματα

/ Εκτέλεση από ένα νήμα */*

Ανάθεση αρχικών τιμών στη δομή δεδομένων κλειδώματος

. . .

/ Εκτέλεση από πολλά νήματα */*

Απόπειρα κλειδώματος ή ενεργοποίησης δομής δεδομένων κλειδώματος;

Κρίσιμο τμήμα;

Ξεκλείδωμα ή απενεργοποίηση δομής δεδομένων κλειδώματος;

. . .

/ Εκτέλεση από ένα νήμα */*

Καταστροφή δομής δεδομένων κλειδώματος;

Χρήση κλειδωμάτων στο πρόγ/μα μεταβίβασης μηνυμάτων

```
# pragma omp critical  
/* q_p = msg_queues[dest] */  
Enqueue(q_p, my_rank, mesg);
```

```
/* q_p = msg_queues[dest] */  
omp_set_lock(&q_p->lock);  
Enqueue(q_p, my_rank, mesg);  
omp_unset_lock(&q_p->lock);
```

Χρήση κλειδωμάτων στο πρόγ/μα μεταβίβασης μηνυμάτων

```
# pragma omp critical  
/* q_p = msg_queues[my_rank] */  
Dequeue(q_p, &src, &mesg);
```

```
/* q_p = msg_queues[my_rank] */  
omp_set_lock(&q_p->lock);  
Dequeue(q_p, &src, &mesg);  
omp_unset_lock(&q_p->lock);
```

Κάποιοι περιορισμοί

1. Μην αναμιγνύετε διαφορετικές τεχνικές αμοιβαίου αποκλεισμού σε ένα κρίσιμο τμήμα.
2. Δεν υπάρχει καμία εγγύηση δικαιοσύνης στις δομές αμοιβαίου αποκλεισμού.
3. Η «ένθεση» δομών αμοιβαίου αποκλεισμού μπορεί να γίνει επικίνδυνη.

Πολλαπλός μήτρας με διάνυσμα

$$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots + a_{i,n-1}x_{n-1}$$

a_{00}	a_{01}	\cdots	$a_{0,n-1}$		y_0
a_{10}	a_{11}	\cdots	$a_{1,n-1}$		y_1
\vdots	\vdots		\vdots		\vdots
a_{i0}	a_{i1}	\cdots	$a_{i,n-1}$	x_0	$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots a_{i,n-1}x_{n-1}$
\vdots	\vdots		\vdots	x_1	\vdots
$a_{m-1,0}$	$a_{m-1,1}$	\cdots	$a_{m-1,n-1}$	\vdots	\vdots
				x_{n-1}	y_{m-1}

```

for (i = 0; i < m; i++) {
    y[i] = 0.0;
    for (j = 0; j < n; j++)
        y[i] += A[i][j]*x[j];
}
    
```

Πολλαπλ/μός μήτρας με διάνυσμα

```
# pragma omp parallel for num_threads(thread_count) \
    default(none) private(i, j) shared(A, x, y, m, n)
for (i = 0; i < m; i++) {
    y[i] = 0.0;
    for (j = 0; j < n; j++)
        y[i] += A[i][j]*x[j];
}
```

Χρόνοι εκτέλεσης και
αποδοτικότητες προγράμματος
πολλ/μού μήτρας-διανύσματος (οι
χρόνοι σε δευτερόλεπτα)

Νήματα	Διαστάσεις μήτρας					
	8.000.000 × 8		8000 × 8000		8 × 8.000.000	
	Χρόνος	Αποδ.	Χρόνος	Αποδ.	Χρόνος	Αποδ.
1	0,322	1,000	0,264	1,000	0,333	1,000
2	0,219	0,735	0,189	0,698	0,300	0,555
4	0,141	0,571	0,119	0,555	0,303	0,275

Ασφάλεια νημάτων

```
void Tokenize(
    char*   lines[]      /* είσοδος/έξοδος */,
    int     line_count   /* είσοδος      */,
    int     thread_count /* είσοδος      */) {
    int my_rank, i, j;
    char *my_token;

    # pragma omp parallel num_threads(thread_count) \
        default(none) private(my_rank, i, j, my_token)
        shared(lines, line_count)
    {
        my_rank = omp_get_thread_num();
    # pragma omp for schedule(static, 1)
        for (i = 0; i < line_count; i++) {
            printf("Νήμα %d > γραμμή %d = %s", my_rank, i, lines[i]);
            j = 0;
            my_token = strtok(lines[i], " \t\n");
            while ( my_token != NULL ) {
                printf("Νήμα %d > λεκτική μονάδα %d = %s\n", my_rank, j, my_token);
                my_token = strtok(NULL, " \t\n");
                j++;
            }
        } /* for i */
    } /* omp parallel */

} /* Tokenize */
```


Συμπερασματικά σχόλια (1)

- Η OpenMP είναι ένα καθιερωμένο πρότυπο για τον προγραμματισμό συστημάτων κοινόχρηστης μνήμης.
- Χρησιμοποιεί ειδικές συναρτήσεις και οδηγίες προς τον προεπεξεργαστή που ονομάζονται οδηγίες pragma.
- Τα προγράμματα που βασίζονται στην OpenMP εκκινούν πολλά νήματα (threads) αντί για πολλές διεργασίες (processes).
- Πολλές οδηγίες της OpenMP μπορούν να τροποποιηθούν με τη χρήση όρων (clauses).

Συμπερασματικά σχόλια (2)

- Σημαντικό πρόβλημα στην ανάπτυξη προγ/των κοινόχρηστης μνήμης είναι οι πιθανές συνθήκες ανταγωνισμού (race conditions).
- Η OpenMP παρέχει διάφορους μηχανισμούς για τη διασφάλιση του αμοιβαίου αποκλεισμού (mutual exclusion) στην προσπέλαση των κρίσιμων τμημάτων.
 - Οδηγίες critical
 - Επώνυμες οδηγίες critical
 - Οδηγίες atomic
 - Απλά κλειδώματα

Συμπερασματικά σχόλια (3)

- Εξ ορισμού, τα περισσότερα συστήματα εφαρμόζουν διαμέριση κατά μπλοκ (block partitioning) για την κατανομή των επαναλήψεων ενός παραλληλοποιημένου βρόχου for σε νήματα.
- Η OpenMP παρέχει μια ποικιλία επιλογών χρονοπρογραμματισμού (scheduling options).
- Στην OpenMP, εμβέλεια (scope) μιας μεταβλητής είναι το σύνολο των νημάτων από τα οποία μπορεί να προσπελαστεί η μεταβλητή.

Συμπερασματικά σχόλια (4)

- Αναγωγή (reduction) είναι ένας υπολογισμός που εφαρμόζει επανειλημμένα τον ίδιο τελεστή αναγωγής σε μια ακολουθία τελεστών μέχρι να καταλήξουμε σε ένα μοναδικό αποτέλεσμα.