Εισαγωγή στο OpenMP

Γιώργος Γκούμας
Αναπ. Καθηγητής ΕΜΠ
goumas@cslab.ece.ntua.gr

EURO Greece

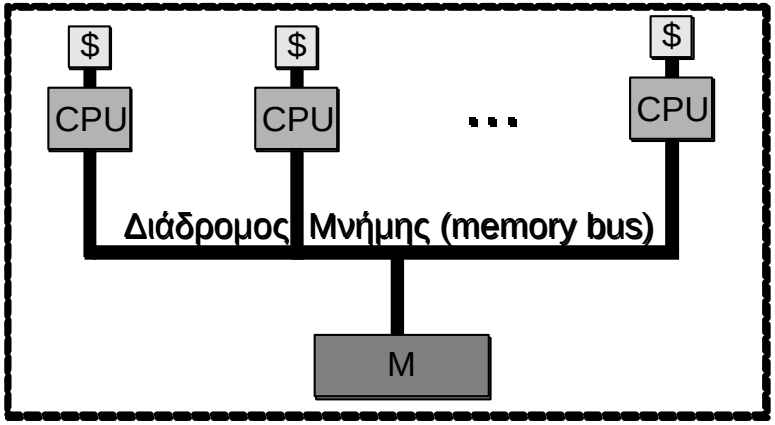
Παράλληλες αρχιτεκτονικές: Οργάνωση μνήμης

- Κοινής μνήμης (shared memory)
  - UMA (Uniform memory access): Χρόνος προσπέλασης ανεξάρτητος του επεξεργαστή και της θέσης μνήμης
  - NUMA (Non-uniform memory access): Χρόνος προσπέλασης εξαρτάται από τον επεξεργαστή και τη θέση μνήμης
  - cc-NUMA (cache-coherent NUMA): NUMA με συνάφεια κρυφής μνήμης
- Κατανεμημένης μνήμης (distributed memory)
- Υβριδική

Αρχιτεκτονική κοινής (μοιραζόμενης) μνήμης

- Οι επεξεργαστές έχουν κοινή μνήμη
- Η πρόσβαση σε όλα τα δεδομένα γίνονται με εντολές ανάγνωσης και εγγραφής στη μνήμη (load / store)
- Τυπικά τα συστήματα διαθέτουν ατομικές εντολές που διευκολύνουν το συγχρονισμό (TAS, CAS)
- Κάθε επεξεργαστής διαθέτει τοπική ιεραρχία κρυφών μνημών
- Απαιτείται υλοποίηση πρωτοκόλλου συνάφειας μνήμης για να διατηρηθεί η συνάφεια των δεδομένων στην κρυφή μνήμη.
- Συνήθως η διασύνδεση γίνεται μέσω διαδρόμου μνήμης (memory bus)
  - Αλλά και πιο εξελιγμένα δίκτυα διασύνδεσης

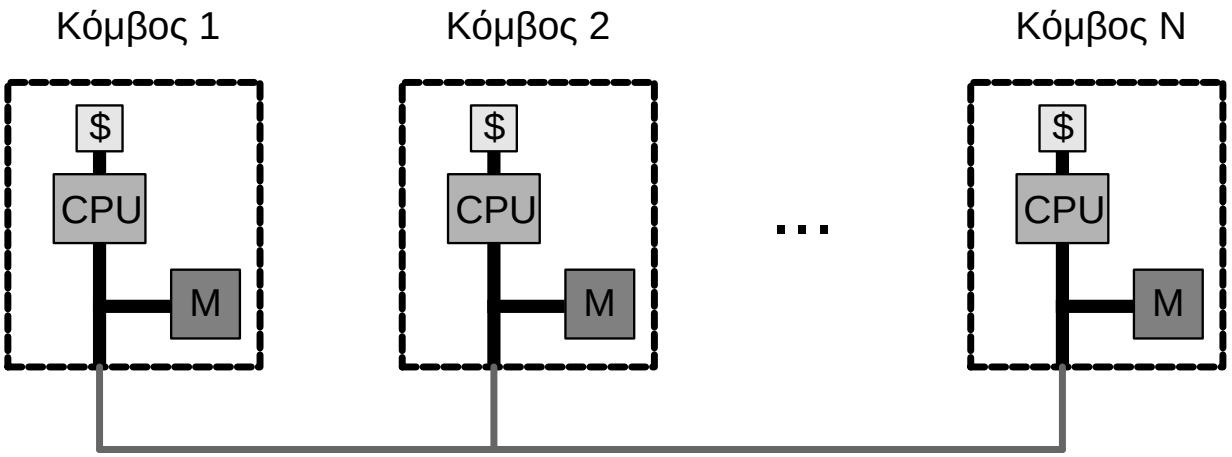
- Ομοιόμορφη ή μη-ομοιόμορφη προσπέλαση στη μνήμη (Uniform Memory Access – UMA, Non-uniform Memory Access – NUMA)
- Η κοινή μνήμη διευκολύνει τον παράλληλο προγραμματισμό (αλλά μπορεί να δημιουργήσει σοβαρά προβλήματα λόγω race conditions!)
- Δύσκολα κλιμακώσιμη αρχιτεκτονική – τυπικά μέχρι λίγες δεκάδες κόμβους (δεν κλιμακώνει το δίκτυο διασύνδεσης)



EURO Greece

Αρχιτεκτονική κατανεμημένης μνήμης

- Κάθε επεξεργαστής έχει τη δική του τοπική μνήμη και ιεραρχία τοπικών μνημών
- Διασυνδέεται με τους υπόλοιπους επεξεργαστές μέσω δικτύου διασύνδεσης
- Η πρόσβαση σε δεδομένα που βρίσκονται σε απομακρυσμένους κόμβους γίνεται ρητά μέσω κλήσεων επικοινωνίας, ανταλλαγής μηνυμάτων (send / receive) ή μέσω συνεννόησης και των δύο πλευρών για πρόσβαση στην απομακρυσμένη μνήμη (παρεμβαίνει το ΛΣ)
- Η κατανεμημένη μνήμη δυσκολεύει τον προγραμματισμό γιατί ο προγραμματιστής απαιτείται να σχεδιάσει και να υλοποιήσει την πρόσβαση σε διακριτές μνήμες (κατακερματισμένος – fragmented προγραμματισμός)
- Η αρχιτεκτονική κλιμακώνει σε χιλιάδες υπολογιστικούς κόμβους



Προγραμματισμός σε μοιραζόμενη μνήμη

```
void thread1(int *shared_var)
{
  int i;
  for (i=0; i<LOOPS; i++)
     *shared_var += 7;
}
```

```
void thread2(int *shared_var)
{
  int i;
  for (i=0; i<LOOPS; i++)
     *shared_var -= 7;
}
```

```
int main()
{
   int shared_var = 13;
   CREATE THREAD(thread1,shared_var)
   CREATE THREAD(thread2,shared_var)
   WAIT THREADS(thread1,thread2)
   printf("shared_var=%d\n", shared_var);
   return 0;
}
```

Προγραμματισμός σε μοιραζόμενη μνήμη

```
void thread1(int *shared_var)
{
  int i;
  for (i=0; i<LOOPS; i++)
     *shared_var += 7;
}
```

```
void thread2(int *shared_var)
{
  int i;
  for (i=0; i<LOOPS; i++)
     *shared_var -= 7;
}
```

```
kkourt@twin3:~/src/tests$ for i in $(seq 10); do ./test1 ; done
 shared_var=353891
 shared_var=360660
 shared_var=362683
 shared_var=360688
 shared_var=356124
 shared_var=321159
 shared_var=357552
 shared_var=-458739
 shared_var=355585
 shared_var=361871
```

EURO Greece

Προγραμματισμός σε μοιραζόμενη μνήμη: Posix Threads

- Χαμηλού επιπέδου διεπαφή για το χειρισμό νημάτων
- Παραδείγματα Συναρτήσεων:
  - `pthread create()`
  - `pthread mutex flock,unlockg()`
  - `pthread cond fwait,signalg()`
  - `pthread barrier wait()`
- Δεν μπορεί εύκολα να χρησιμοποιηθεί για την παραλληλοποίηση σειριακών εφαρμογών
- Δεν είναι αρκετά απλή για να χρησιμοποιηθεί από επιστήμονες, που θέλουν να παραλληλοποιήσουν τις εφαρμογές τους
- Σε πολλές περιπτώσεις απαιτείται κάτι πιο απλό και εύχρηστο, ακόμα και αν χρειάζεται να θυσιαστεί ένα σημαντικό τμήμα του ελέγχου πάνω στην εκτέλεση

OpenMP

- Πρότυπο για προγραμματισμό σε μοιραζόμενη μνήμη
- Ορίζει συγκεκριμένη διεπαφή (API) και όχι υλοποίηση
  - οδηγίες σε μεταγλωττιστή (compiler directives)
  - βιβλιοθήκη χρόνου εκτέλεσης (run-time library)
  - μεταβλητές συστήματος (environment variables)
- Ο παραλληλισμός δηλώνεται ρητά (explicitly) από τον προγραμματιστή
- Γλώσσες: C/C++, Fortran
- MP = MultiProcessor
- Τα προγράμματα του OpenMP:
  - Μπορούν να μεταφραστούν από μεταγλωττιστή που δεν το υποστηρίζει.
  - Μπορούν να εκτελεστούν σειριακά.
- Εφαρμόζεται κυρίως σε εφαρμογές με μεγάλους πίνακες

Ένα πρώτο παράδειγμα

```
// Άθροισμα διανυσμάτων
for (i = 0; i < n; i++)  // Ανεξάρτητες επαναλήψεις στο βρόχο
   c[i] = a[i] + b[i];
```

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

```
% gcc -fopenmp source.c
% setenv OMP_NUM_THREADS 4
% ./a.out
```

Μοντέλο εκτέλεσης

- Η εκτέλεση ξεκινά από 1 **initial thread**
- Όταν το initial thread συναντήσει μία *παράλληλη περιοχή*
  - Δημιουργείται μία ομάδα νημάτων (**team of threads**) που περιλαμβάνει το initial thread (τώρα λέγεται **master thread**) και 0 ή περισσότερα άλλα threads
  - Κάθε νήμα αναλαμβάνει την εκτέλεση του μπλοκ εντολών που περιλαμβάνει η παράλληλη περιοχή (υπάρχουν κατάλληλες οδηγίες που διαφοροποιούν την εκτέλεση των νημάτων, βλ. συνέχεια)
- Στο τέλος της παράλληλης περιοχής τα νήματα συγχρονίζονται
- Οι παράλληλες περιοχές μπορεί να είναι φωλιασμένες (nested)
  - Αν η υλοποίηση το υποστηρίζει η φωλιασμένη ομάδα νημάτων μπορεί να περιέχει περισσότερα του ενός thread

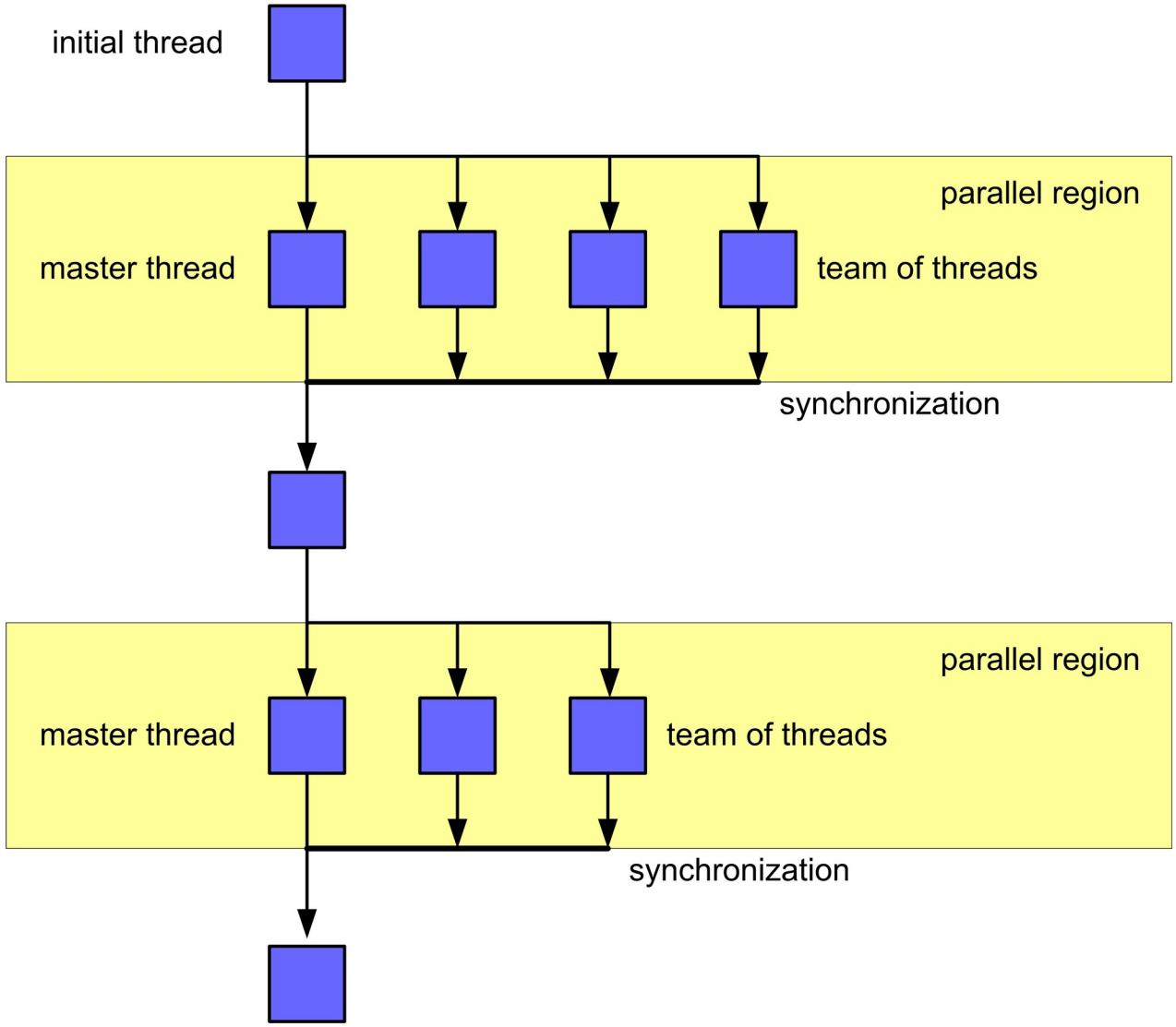
Μοντέλο εκτέλεσης

```
#include <omp.h>

   main(){
  …
  #pragma omp parallel num_threads(4)
  {
    …
  }
  …
  omp_set_num_threads(3);
  #pragma omp parallel
  {
    …
  }
  …
}
```

EURO Greece

Μοντέλο εκτέλεσης

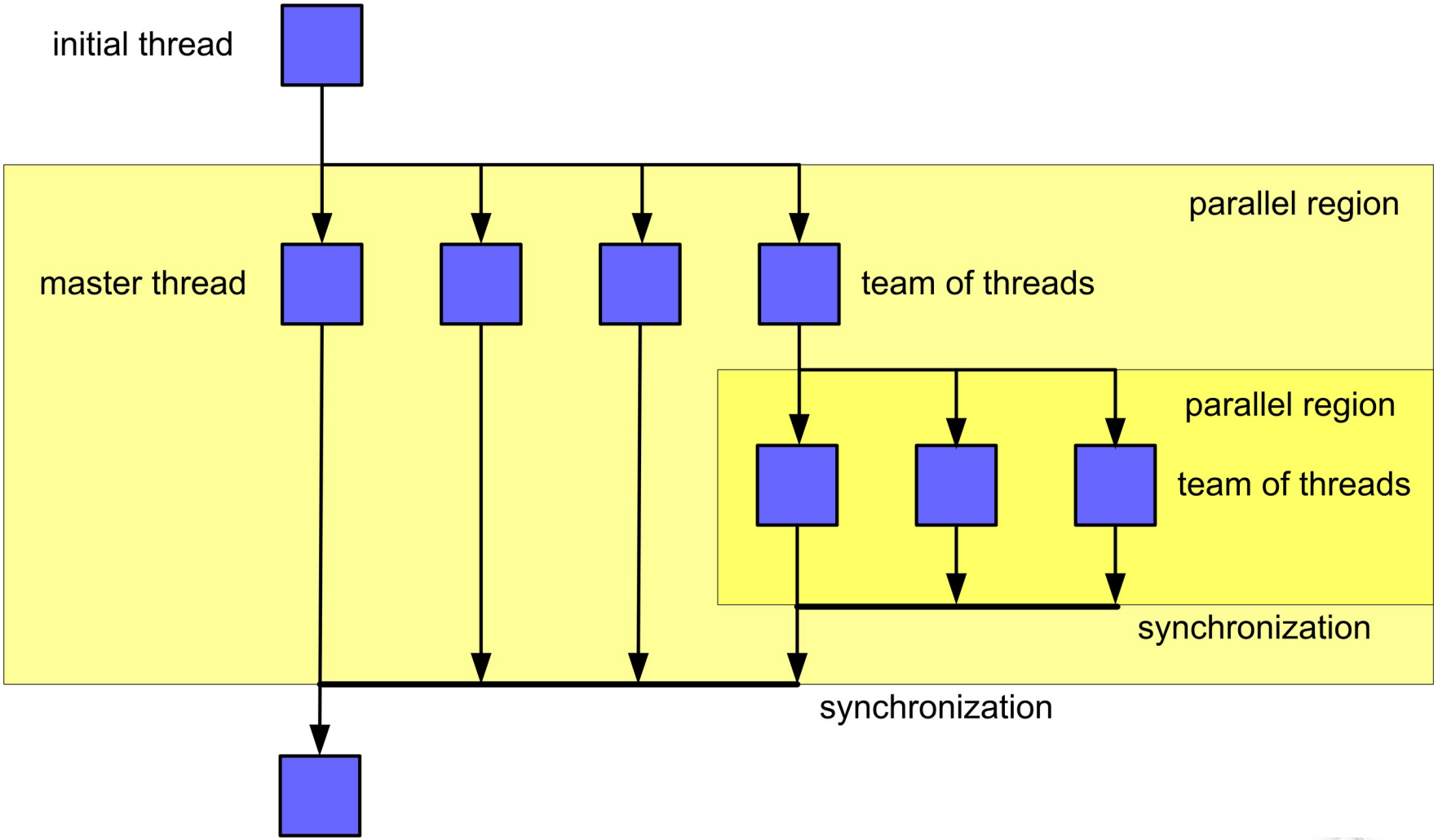


Μοντέλο εκτέλεσης: φωλιασμένες παράλληλες περιοχές

```
#include <omp.h>

   main(){
  …
  #pragma omp parallel num_threads(4)
  {
    …
    if (omp_get_thread_num()== 3){
       …
       omp_set_num_threads(3);
       #pragma omp parallel
       {
          …
       }
    }
  }
  …
}
```

Μοντέλο εκτέλεσης: φωλιασμένες παράλληλες περιοχές



OpenMP components

Directives

- Parallel regions
- Work sharing
- Synchronization
- Tasks
- Data-sharing attributes
  - private
  - firstprivate
  - lastprivate
  - shared
  - reduction

Environment variables

- Number of threads
- Scheduling type
- Dynamic thread adjustment
- Nested parallelism

Runtime environment

- Number of threads
- Thread ID
- Dynamic thread adjustment
- Nested parallelism
- Timers
- API for locking

Βασικές έννοιες

- Παράλληλη Περιοχή (Parallel Region)
  - Κώδικας που εκτελείται από πολλαπλά νήματα
- Κατανομή Εργασίας (Work Sharing)
  - Η διαδικασία κατά την οποία κατανέμεται η εργασία στα νήματα μια παράλληλης περιοχής
- Οδηγία Μεταγλωττιστή (Compiler Directive)
  - Η διεπαφή για την χρήση του OpenMP σε προγράμματα.
  - Για την C:
  - `#pragma omp <directive> <clauses>`
- Construct:

```
#pragma omp ...
<C statement>
```

Παράλληλες περιοχές

```
#pragma omp parallel [clause[ [, ]clause] ...] new-line
  structured-block
```

- where clause is one of the following:
  - if (scalar-expression)
  - num_threads (integer-expression)
  - default (shared | none)
  - private (list)
  - firstprivate (list)
  - shared (list)
  - copyin (list)
  - reduction (operator: list)
- Ο αριθμός των νημάτων καθορίζεται:
  - Από το num_threads clause
  - Με τη χρήση της omp set_num_threads()
  - Με τη μεταβλητή περιβάλλοντος OMP_NUM_THREADS (Χρόνος Εκτέλεσης)
- Υπονοείται barrier στο τέλος της περιοχής
- Το barrier υπονοεί flush

Hello world

```
#include <omp.h>
#include <stdio.h>
int main()
{
#pragma omp parallel
  printf("Hello world! (thread_id: %d)\n", omp_get_thread_num());
  return 0;
}
```

Κατανομή εργασίας

- **directives**:
  - `#pragma omp for`
  - `#pragma omp sections`
  - `#pragma omp single`
- Τα directives για την κατανομή εργασίας, περιέχονται σε μία παράλληλη περιοχή.
- Δεν δημιουργούνται νέα νήματα
- Δεν υπονοείται barrier στην είσοδο

#pragma omp for

```
#pragma omp for [schedule(…)] [nowait]
   for-loop
```

- Κατανέμει επαναλήψεις εντολής ***for*** σε ομάδα νημάτων
- Εντολή for σε κανονική μορφή (canonical form)
- schedule: καθορίζει τρόπο κατανομής επαναλήψεων
  - static[,chunk]: round-robin στατική κατανομή
  - dynamic[,chunk]: δυναμική κατανομή σε ανενεργά νήματα
  - guided[,chunk]: δυναμική κατανομή με εκθετική μείωση
  - runtime: κατανομή καθορίζεται σε χρόνο εκτέλεσης
- nowait: αποτρέπει συγχρονισμό κατά την έξοδο

Παράδειγμα

```
for(i=1;i<n;i++)
    b[i]=(a[i]+a[i-1])/2.0;

#pragma omp parallel
  {
    #pragma omp for
      for(i=1;i<n;i++)
        b[i]=(a[i]+a[i-1])/2.0;
  }
```

#pragma omp sections

```
#pragma omp sections [nowait]
{
  #pragma omp section
    structured-block
  #pragma omp section
    structured-block
}
```

- Ορίζει μία η περισσότερες ανεξάρτητες περιοχές section που κατανέμονται μεταξύ των νημάτων
- Κάθε περιοχή section ανατίθεται σε διαφορετικό νήμα
- nowait: αποτρέπει συγχρονισμό κατά την έξοδο

#pragma omp single

```
#pragma omp single [nowait]
   structured-block
```

- Ορίζει τμήμα κώδικα που εκτελείται από μόνο ένα νήμα της ομάδας
- nowait: αποτρέπει συγχρονισμό κατά την έξοδο

Παράδειγμα – progress report

```
#pragma omp parallel
{
  #pragma omp single
    printf(“Beginning work1.\n”);
  work1();
  #pragma omp single
    printf(“Finished work1.\n”);
  #pragma omp single nowait
    printf(“Finished work1, beginning work2.\n”);
  work2();
}
```

Συντομεύσεις

```
#pragma omp parallel
#pragma omp for
for (...)
```



```
#pragma omp parallel for
for (...)
```

```
#pragma omp parallel
#pragma omp sections
```



```
#pragma omp parallel sections
```

Constructs συγχρονισμού

- `#pragma omp barrier:`

  Συγχρονισμός νημάτων
- `#pragma omp master:`

  Κώδικας που εκτελείται μόνο από το κύριο νήμα
- `#pragma omp critical:`

  Κώδικας που δεν εκτελείται παράλληλα
- `#pragma omp atomic:`

  Ατομική λειτουργία σε θέση μνήμης (++,–,+=,...)
- `#pragma omp flush:`

  Επιβολή συνεπούς εικόνας των μοιραζόμενων αντικειμένων
- `#pragma omp ordered:`

  Επιβολή σειριακής εκτέλεσης structured block

#pragma omp barrier / master

```
#pragma omp parallel
{
   #pragma omp barrier
   #pragma omp master
      gettimeofday(start,(struct timezone*)NULL);
   work();
   #pragma omp barrier
   #pragma omp master
   {
      gettimeofday(finish,(struct timezone*)NULL);
      print_stats(start,finish);
   }
}
```

#pragma omp critical

#pragma omp critical *[(name)] new-line*
structured-block

```
#pragma omp parallel shared(x, y) private(x next, y next)
{
   #pragma omp critical (xaxis)
      x_next=dequeue(x);
   work(x_next);
   #pragma omp critical (yaxis)
      y_next=dequeue(y);
   work(y_next);
}
```

#pragma omp atomic

#pragma omp atomic new-line
  expression-stmt

```
#pragma omp parallel for shared(x, y, index, n)
{
   for(i=0;i<n;i++)
      #pragma omp atomic
         x[index[i]] += work1(i);
      y[i]+=work2(i);
}
```

#pragma omp ordered

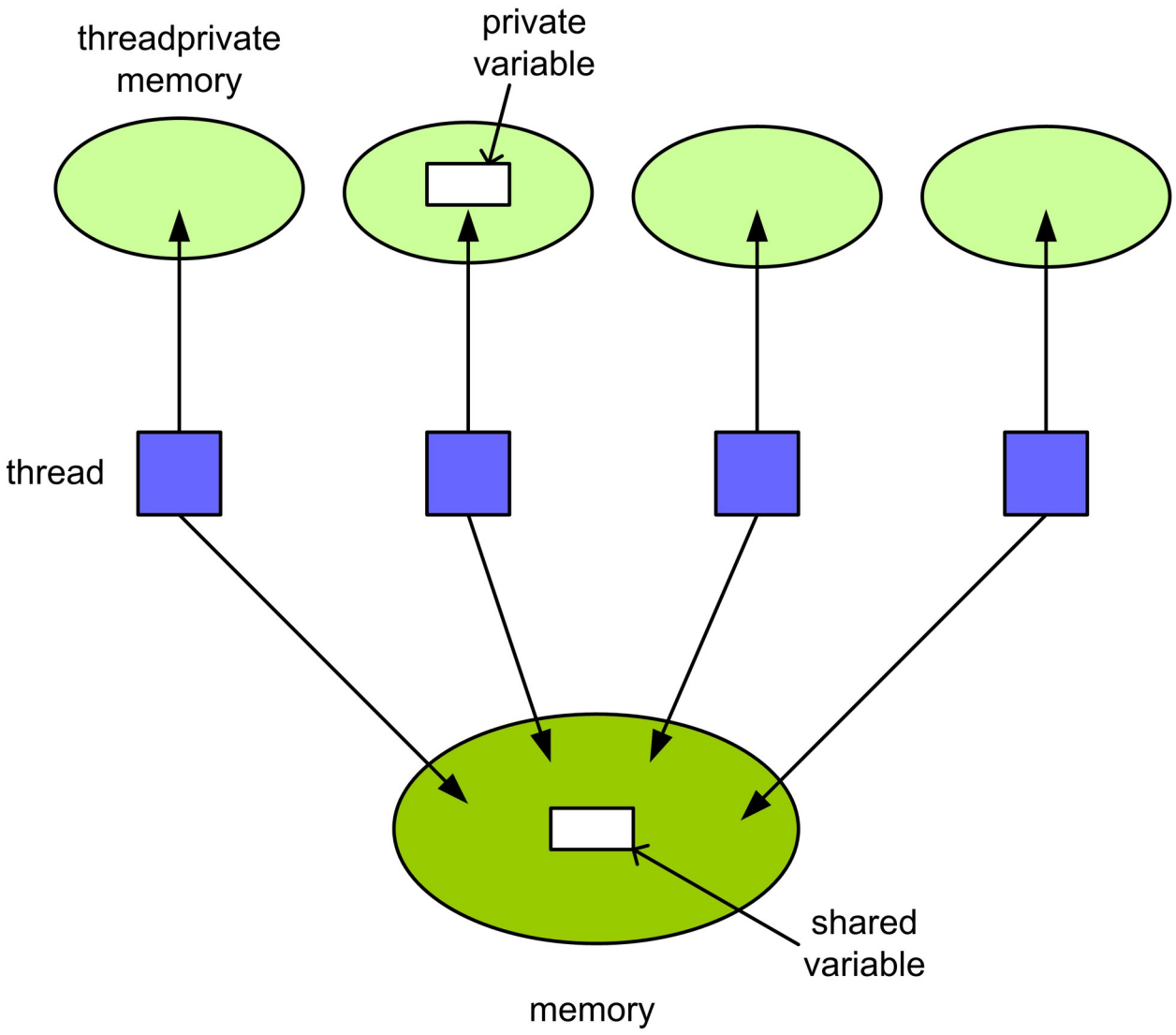
#pragma omp ordered *new-line*
structured-block

```
#pragma omp parallel
{
   #pragma omp for ordered
      for(i=0;i<N;i++){
         a[i]=compute(i);
         #pragma omp ordered
         printf("a[%d]=%d\n",i,a[i]);
      }
}
```

Μοντέλο δεδομένων

- Σε μία παράλληλη περιοχή, υπάρχουν δύο ήδη μεταβλητών, **shared** και **private**
- Οι αλλαγές στα αντικείμενα που βρίσκονται στην κοινή μνήμη (shared) δεν γίνονται απαραίτητα αντιληπτές στο σύνολο των νημάτων
- Κάθε νήμα έχει μία τοπική εικόνα των δεδομένων
- Η λειτουργία **flush** επιβάλει συνέπεια ανάμεσα στις τοπικές εικόνες και στην κεντρική μνήμη

Μοντέλο δεδομένων



Περιβάλλον δεδομένων

- private (*variable-list*)
  - Ανάθεση νέου αντικειμένου για κάθε νήμα
  - Το πρότυπο αντικείμενο έχει απροσδιόριστη τιμή κατά την είσοδο και έξοδο στο construct, και δεν πρέπει να τροποποιείται
- firstprivate (*variable-list*)
  - Σαν private, κάθε νέο αντικείμενο **αρχικοποιείται** (εισέρχεται στην παράλληλη περιοχή) με την τιμή του προτύπου ακριβώς πριν την έναρξη της παράλληλης περιοχής
- lastprivate (*variable-list*)
  - Σαν private, το πρότυπο αντικείμενο εξέρχεται από την παράλληλη περιοχή με την τιμή που κατέχει το thread που εκτέλεσε την τελευταία επανάληψη (σε parallel loop) ή το τελευταίο section (σε parallel sections)

Περιβάλλον δεδομένων

- shared (*variable list*)
  - Μοιραζόμενη μεταβλητή για όλα τα νήματα της ομάδας
- reduction (*op: variable-list*)
  - Αναφέρεται σε εντολές τις μορφής x = x op expr, όπου op ένας από τους *, -, &, ^, |, &&, ||
  - Κάθε μεταβλητή το πολύ σε μια reduction clause
  - Για κάθε μεταβλητή δημιουργείται αντίστοιχη τοπική μεταβλητή σε κάθε νήμα και αρχικοποιείται ανάλογα με τελεστή op
- default (shared | none)
  - shared: ισοδύναμο με τον ορισμό κάθε μεταβλητής που δεν υπάρχει σε κανέναν περιβάλλον (shared, private, reduction, κλπ) , σαν shared
  - none: αν μία μεταβλητή δεν έχει ενταχθεί σε κάποιο περιβάλλον τότε ο compiler «χτυπάει» λάθος

Παράδειγμα

```
#pragma omp parallel for reduction(+:sum)
   for(i=1;i<n;i++)
      sum = sum + a(i);
```

Βιβλιοθήκη χρόνου εκτέλεσης

- Περιβάλλον εκτέλεσης
  - omp_set_num_threads
  - omp_get_thread_num
  - omp_set_dynamic
- Συγχρονισμός με κλειδώματα
  - omp_init_lock
  - omp_set_lock / omp_test_lock
  - omp_unset_lock
  - nested
- Χρονομέτρηση
  - omp_get_wtime
  - omp_get_wtick

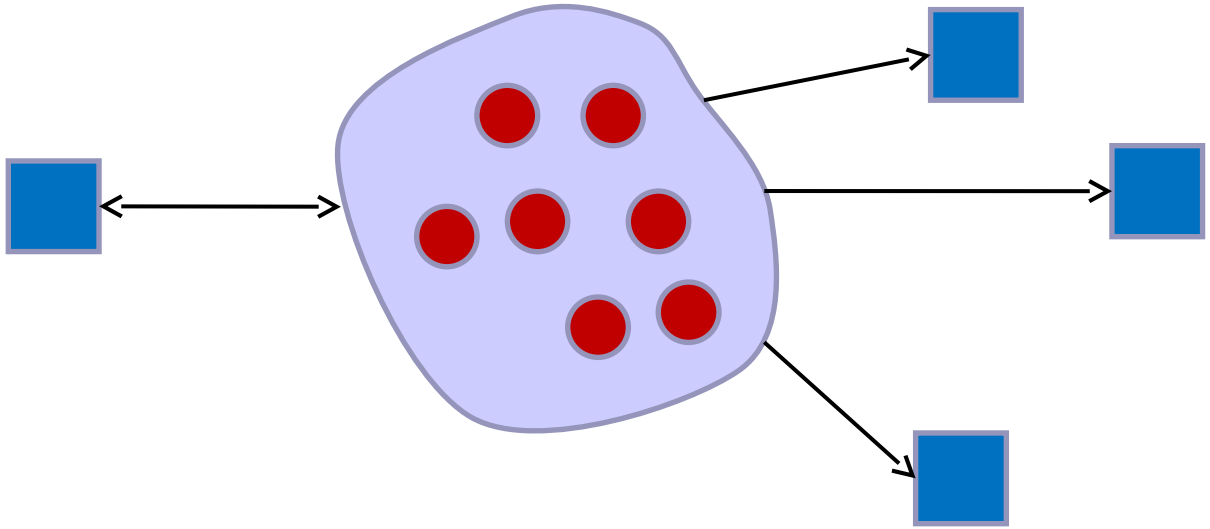
Μεταβλητές συστήματος

- Δρομολόγηση
  - export OMP_SCHEDULE=“static”
  - export OMP_SCHEDULE=“static,100”
  - setenv OMP_SCHEDULE “dynamic,20”
  - setenv OMP_SCHEDULE “guided,50”
- Δυναμική πολυνηματική εκτέλεση
  - export OMP_DYNAMIC=TRUE
  - setenv OMP_DYNAMIC FALSE
- Πλήθος νημάτων
  - export OMP_NUM_THREADS=2

OpenMP tasks

- Η παραλληλοποίηση με χρήση tasks ξεκίνησε να υποστηρίζεται από το OpenMP στο τελευταίο πρότυπο (OpenMP 3.0) – May 2008
- Παρέχει τη δυνατότητα παραλληλοποίησης για εφαρμογές που παράγουν δουλειά δυναμικά
- Παρέχει ένα ευέλικτο μοντέλο για μη κανονικό (irregular) παραλληλισμό
- Ευκαιρίες για παραλληλισμό σε:
  - While loops
  - Recursive structures

Η λογική των OpenMP tasks



thread

task

#pragma omp task

```
#pragma omp task [clause [[,]clause] ...]
  structured-block
```

όπου clause:

if(*scalar-expression***)**
untied
default(shared | none)
private(*list***)**
firstprivate(*list***)**
shared(*list***)**

- Το thread που συναντά ένα `#pragma omp task` directive δημιουργεί ένα task με τον κώδικα που περιέχει το `structured-block` και το βάζει σε ένα task pool
- Το thread μπορεί να εκτελέσει ή όχι ένα task που συναντά

Παράδειγμα

```
void process_list_items(node * head)
{
   #pragma omp parallel
   {
      #pragma omp single
      {
         node *p = head;
         while (p) {
            #pragma omp task
               process(p);
            p = p->next;
         }
      }
   }
}
```

Συγχρονισμός tasks

`#pragma omp taskwait`

το τρέχον task σταματά την εκτέλεσή του μέχρι όλα τα tasks που έχουν δημιουργηθεί μέχρι στιγμής από το τρέχον (παιδιά) να ολοκληρώσουν την εκτέλεσή τους

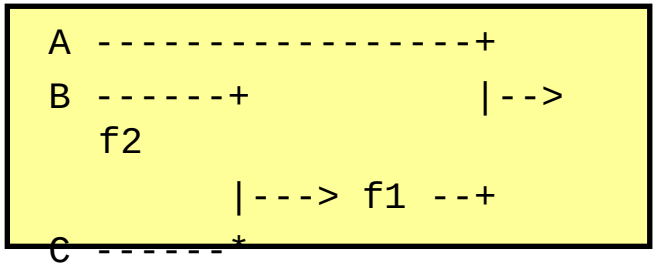
Ισχύει μόνο για τα άμεσα παιδιά (π.χ. όχι για τα εγγόνια)

(**Σημείωση:** βλέπε taskgroups στο OpenMP 4.0 για την αναμονή και άλλων απογόνων)

Μοντέλο εκτέλεσης

- Προσοχή στις έννοιες **δημιουργία** / **εκτέλεση** task!
- Κάθε **task** μπορεί να εκτελεστεί από ένα από τα **threads** της ομάδας που το δημιούργησε
- Κάθε thread της ομάδας δημιουργεί ένα αρχικό (implicit) task
- Άρα κάθε λειτουργία σχετική με tasks έχει νόημα μόνο σε παράλληλες περιοχές
- Όταν ξεκινήσει η εκτέλεση ενός task by default είναι προσδεμένο (tied) με ένα thread
  - Αυτό μπορεί να αλλάξει (βλ. untied)
- Ένα task αναστέλλει τη λειτουργία του όταν υποχρεωθεί να **εκτελέσει** ένα άλλο task (βλ. If (0)) ή αν συναντήσει ένα taskwait

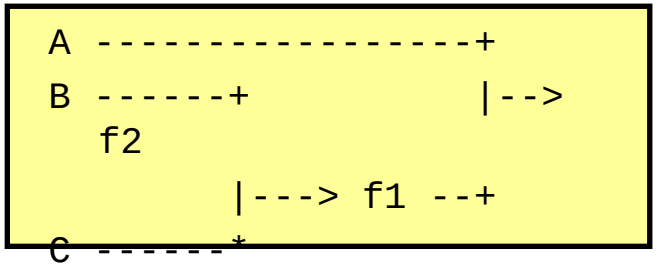
Παράδειγμα



Παράδειγμα

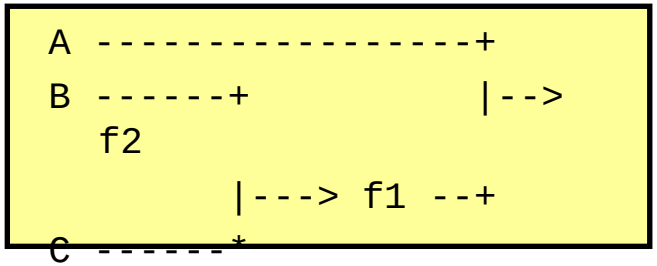
```
void foo ()
{
   int a, b, c, x, y;
   #pragma omp parallel
   {
      #pragma omp single //serial creation of tasks
      {
         #pragma omp task shared (a)
            a = A();
         #pragma omp task shared (b, c, x)
         {
            #pragma omp task shared (b)
               b = B();
            #pragma omp task shared (c)
               c = C();
            #pragma omp taskwait
            #pragma omp task
               x = f1 (b, c);
         }
         #pragma omp taskwait
         #pragma omp task
            y = f2 (a, x);
      }
   }
}
```

Σωστό;



Παράδειγμα

```
void foo ()
{
   int a, b, c, x, y;
   #pragma omp parallel
   {
      #pragma omp single //serial creation of tasks
      {
         #pragma omp task shared (a)
            a = A();
         #pragma omp task shared if (0) (b, c, x)
         {
            #pragma omp task shared (b)
               b = B();
            c = C();
            #pragma omp taskwait
         }
         x = f1 (b, c);
         #pragma omp taskwait
         y = f2 (a, x);
      }
   }
}
```



OpenMP 4.0

- July 2013
- Υποστήριξη για vectorization
  - #pragma omp simd
- Υποστήριξη για επιταχυντές
  - #pragma omp declare target
  - #pragma omp target data
- Ισχυρότερη υποστήριξη για task graphs
  - taskgroups
  - task dependencies
- Επιτρέπει τον ορισμό reduction function από το χρήστη
- …

http://www.openmp.org
http://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf

OpenMP 4.5

- November 2015
- Υποστήριξη παραλληλοποίησης loop με tasks
  - #pragma omp taskloop
- Υποστήριξη “DOACROSS” παραλληλισμού (υπάρχουν εξαρτήσεις ανάμεσα στα iterations ενός loop)
  - Π.χ. #pragma omp ordered depend (sink: i - 1, j) depend (sink: i, j - 1)

http://www.openmp.org

http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf