

Μία επισκόπηση του OpenMP

PLUG

1 JUNE 2011

Alexandros Papadakis
<papadakis@ceid.upatras.gr>
<http://students.ceid.upatras.gr/~papadakis/>



What is OpenMP ?

OpenMP := ”Open specifications for Multi-Processing”

OpenMP είναι

- Ένα Application Program Interface (API) το οποίο χρησιμοποιείται για παραλληλισμό προγραμμάτων με τη χρήση threads και μοντέλου shared memory
- Αποτελείται από 3 κύρια API components:

Compiler Directives

Environment Variables

Runtime Library Routines

- Portable:

Το API προσδιορίζεται για C/C++ και Fortran

έχει υλοποιηθεί για Linux και Windows NT platforms

Components of OpenMP 2.5

Directives

- ◆ *Parallel region*
- ◆ *Worksharing*
- ◆ *Synchronization*
- ◆ *Data-sharing attributes*
 - ☞ *private*
 - ☞ *firstprivate*
 - ☞ *lastprivate*
 - ☞ *shared*
 - ☞ *reduction*
- ◆ *Orphaning*

Runtime environment

- ◆ *Number of threads*
- ◆ *Thread ID*
- ◆ *Dynamic thread adjustment*
- ◆ *Nested parallelism*
- ◆ *Wallclock timer*
- ◆ *Locking*

Environment variables

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

Advantages of OpenMP

- ❑ Καλή απόδοση (**performance**) και επεκτασιμότητα (**scalability**)
 - Αν το κάνεις σωστά....
- ❑ “Ωριμο” πρότυπο
- ❑ ένα OpenMP πρόγραμμα είναι **portable**
 - Υποστηρίζεται από ένα μεγάλο αριθμό από compilers
- ❑ Χρειάζεται λίγη προγραμματιστική προσπάθεια
- ❑ Επιτρέπει το πρόγραμμα να παραλληλοποιηθεί σταδιακά
- * Το **OpenMP** είναι ιδανική λύση για **multicore** architectures

OpenMP Programming Model

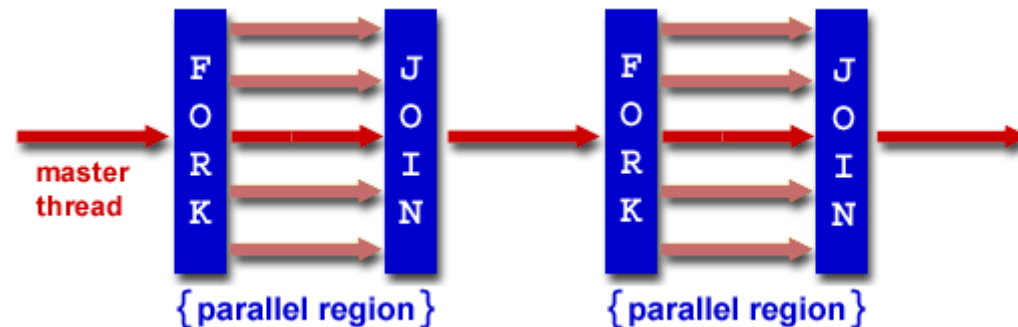
Shared Memory, Thread Based Parallelism:

Το OpenMP είναι βασισμένο στην ύπαρξη πολλαπλών threads όπου επικοινωνούν με το μεντέλο κοινής μνήμης.

Ρητός παραλληλισμός:

Το OpenMP είναι ένα ρητό (όχι αυτόματο) προγραμματιστικό μεντέλο, όπου προσφέρει στον προγραμματιστή τον πλήρη έλεγχο του παραλληλισμού.

Fork - Join Model:



Compiler Directive Based
Nested Parallelism Support
Dynamic Threads

How do threads interact

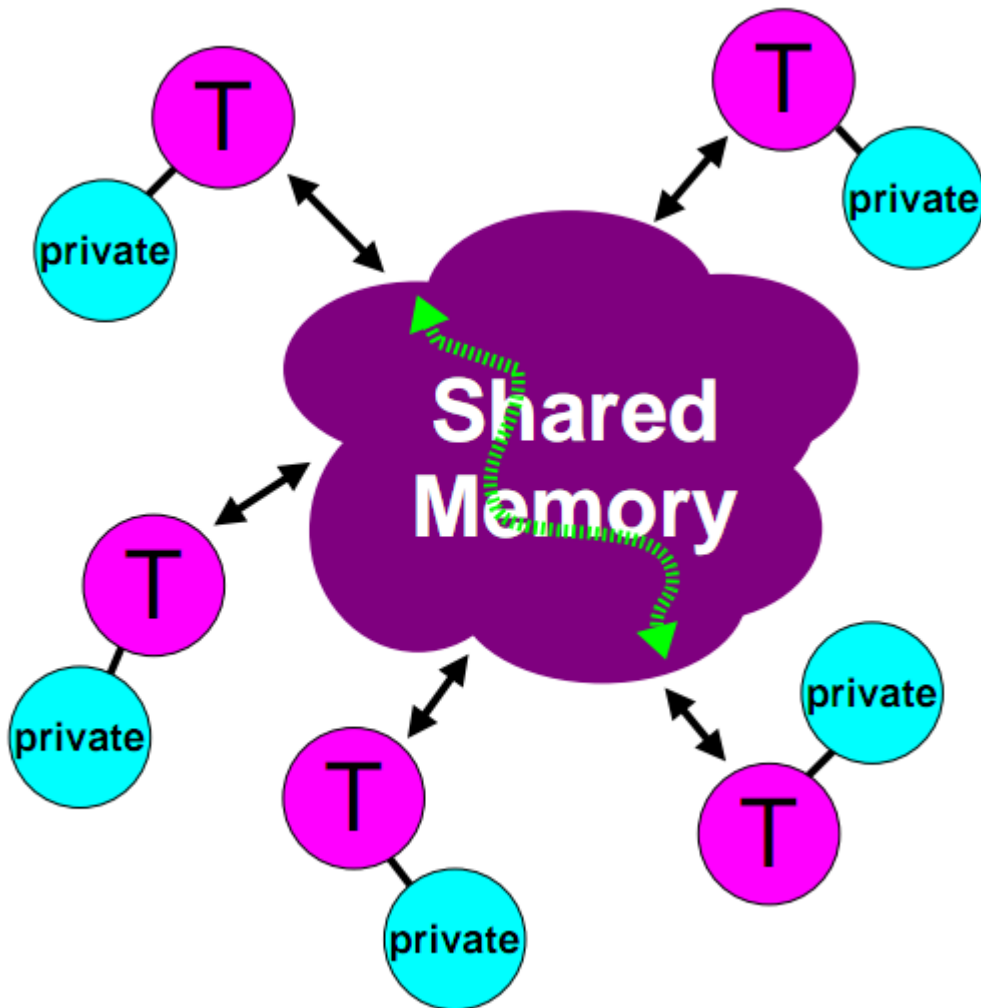
- ❑ Το OpenMP είναι ένα multi-threading, shared address model
 - Τα Threads επικοινωνούν με sharing variables.

- ❑ Αυτόματα το sharing of data προκαλεί race conditions:
 - race condition:
 1. > ./a.out
Output=5
 2. > ./a.out
Output=100

- ❑ Για να ελέξουμε τα race conditions:
 - Χρησιμοποιούμε synchronization για να προστατέψουμε τα “κοινά” δεδομένα.

- ❑ Όμως το synchronization είναι “ακριβό”:
 - Προσπαθούμε να αλλάξουμε πως τα δεδομένα προσπελούνται ώστε να μειώσουμε τη χρήση του Synchronization. { $A[i]=A[i-1]$ vs $A[i]=3*A[i]$ }

OpenMP Memory Model



- ✓ All threads have access to the same, globally shared, memory
- ✓ Data can be shared or private
- ✓ Shared data is accessible by all threads
- ✓ Private data can only be accessed by the thread that owns it
- ✓ Data transfer is transparent to the programmer
- ✓ Synchronization takes place, but it is mostly implicit

Data-Sharing Attributes

□ Αρχικά υπάρχουν 2 βασικοί τύποι:

- **Shared**

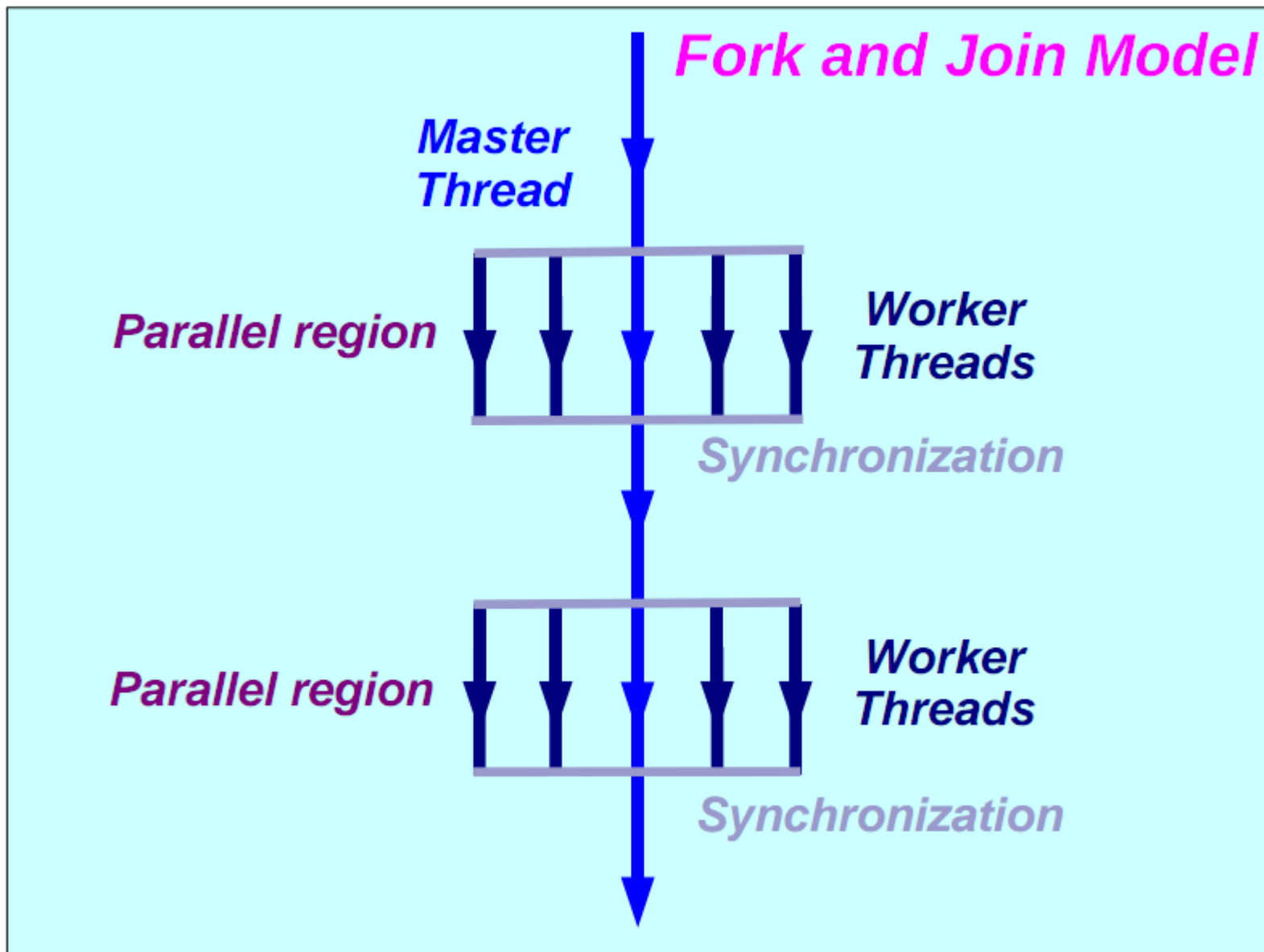
- ✓ υπάρχει μόνο ένα στιγμιότυπο του δεδομένου
- ✓ όλα τα threads μπορούν να διαβάσουν και να γράψουν στο δεδομένο ταυτόχρονα (*race condition*), εκτός αν το προστατέψουμε
- ✓ όλες οι αλλαγές είναι ορατές από όλα τα threads

◆ Αυτό όμως μπορεί να μην συμβαίνει απαραίτητα αμέσως, εκτός αν το αναγκάσουμε (*flush*)

- **Private**

- ✓ κάθε thread έχει ένα αντίγραφο του δεδομένου
- ✓ κανένα άλλο thread δεν μπορεί να έχει πρόσβαση σε αυτό το δεδομένο
- ✓ οι αλλαγές είναι ορατές μόνο στο thread στο οποίο ανήκει το δεδομένο

The OpenMP Execution Model



OpenMP core syntax

- Τα περισσότερα constructs του OpenMP είναι compiler directives.

#pragma omp construct [clause [clause]...]

- παραδειγμα

#pragma omp parallel num_threads(4)

- Τα function prototypes και types βρισκονται:

#include <omp.h>

- Τα περισσότερα OpenMP constructs εφαρμόζονται σ'ένα “structured block”.

- Structured block: ένα block από ένα ή περισσότερα statements, με ένα σημείο εισόδου στην αρχή και ένα σημείο εξόδου στο τέλος.

- Επιτρέπεται να υπάρχει exit() μέσα στο structured block.

Thread Creation: Parallel Regions

- ❑ Στο OpenMP δημιουργούμε threads με το `parallel` construct.
- ❑ Για παράδειγμα, θέλουμε να δημιουργήσουμε 4 threads:

Each thread executes a copy of the code within the structured block

```
double A[1000];  
omp_set_num_threads(4);  
#pragma omp parallel  
{  
    int ID = omp_get_thread_num();  
    pooh(ID,A);  
}
```

Runtime function to request a certain number of threads

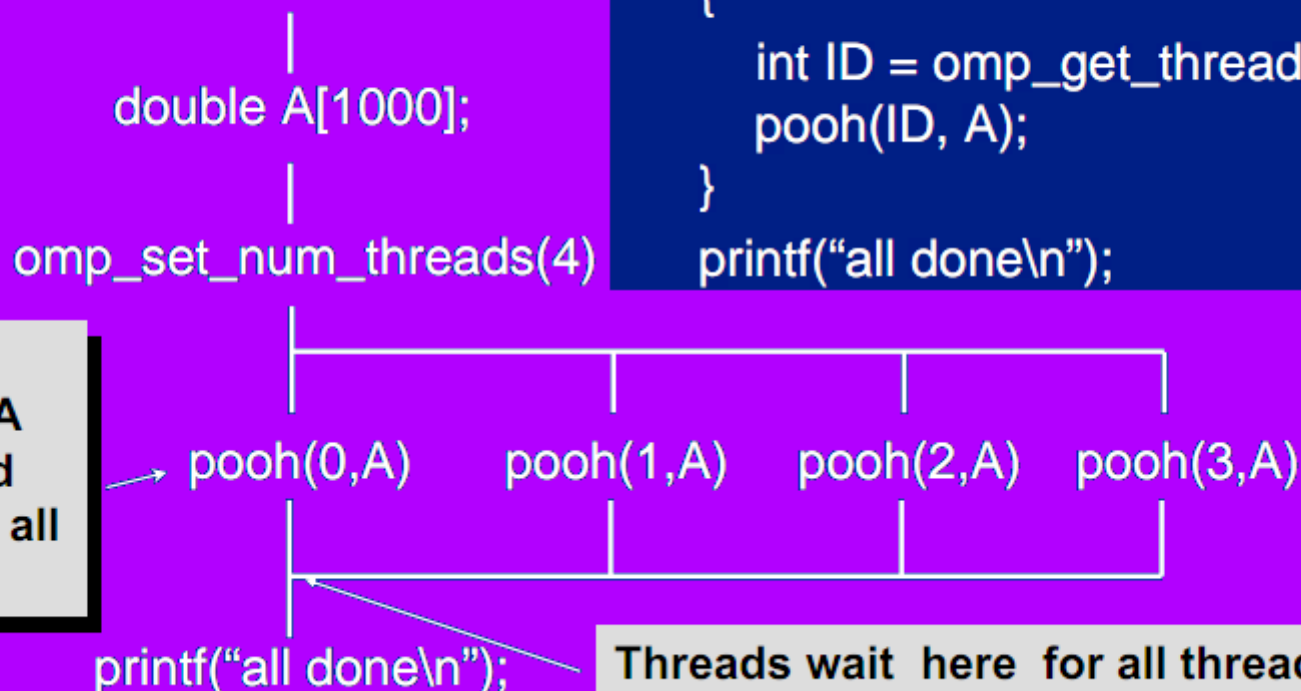
Runtime function returning a thread ID

- ❑ Κάθε thread καλεί την `pooh(ID,A)` για $ID = 0$ μέχρι 3

Parallel Regions-Example

Each thread executes the same code redundantly.

```
double A[1000];  
omp_set_num_threads(4);  
#pragma omp parallel  
{  
    int ID = omp_get_thread_num();  
    pooh(ID, A);  
}  
printf("all done\n");
```



A single copy of A is shared between all threads.

Threads wait here for all threads to finish before proceeding (i.e. a *barrier*)

SPMD vs. worksharing

□ Ένα **parallel** construct από μόνο του δημιουργεί ένα SPMD ή “Single Program Multiple Data” πρόγραμμα...
δηλαδή, κάθε **thread** εκτελεί τον ίδιο κώδικα.

□ Πως μπορούμε να χωρίσουμε το κώδικα μεταξύ των threads?

- ε αυτό ονομάζεται **worksharing**
 - ✓loop construct
 - ✓section construct
 - ✓single construct

Work-sharing Constructs

```
#pragma omp for  
{  
    ....  
}
```

```
#pragma omp sections  
{  
    ....  
}
```

```
#pragma omp single  
{  
    ....  
}
```

- ☞ Η δουλειά κατανέμεται (distributed) μεταξύ των threads
- ☞ Πρέπει να είναι ενθυλακωμένα μέσα σ'ένα parallel region
- ☞ Δεν υπάρχει “υπονοούμενο” barrier στην είσοδο; αλλά υπάρχει “υπονοούμενο” barrier στην έξοδο (εκτός αν χρησιμοποιήσουμε ένα nowait clause)
- ☞ Ένα work-sharing construct δεν δημιουργεί καινούρια threads

The omp for directive

- Οι επαναλήψεις του loop κατανέμονται στα threads

```
#pragma omp for [clause[,] clause] ...]  
  <original for-loop>
```

Clauses που υποστηρίζει:

- ✓private
- ✓firstprivate
- ✓lastprivate
- ✓reduction
- ✓ordered
- ✓schedule
- ✓nowait

The schedule clause

❑ Το schedule clause επηρεάζει το πως τα loops αντιστοιχίζονται στα threads

✓ `schedule(static [,chunk])`

-Οι επαναλήψεις μοιράζονται εξ αρχής σε τμήματα μεγέθους N/chunk ή $N/\#\text{threads}$ αν δεν οριστεί το chunk

✓ `schedule(dynamic[,chunk])`

- Τα τμήματα ανατίθενται δυναμικά κατά την εκτέλεση. Αν δεν οριστεί το chunk, το default είναι 1

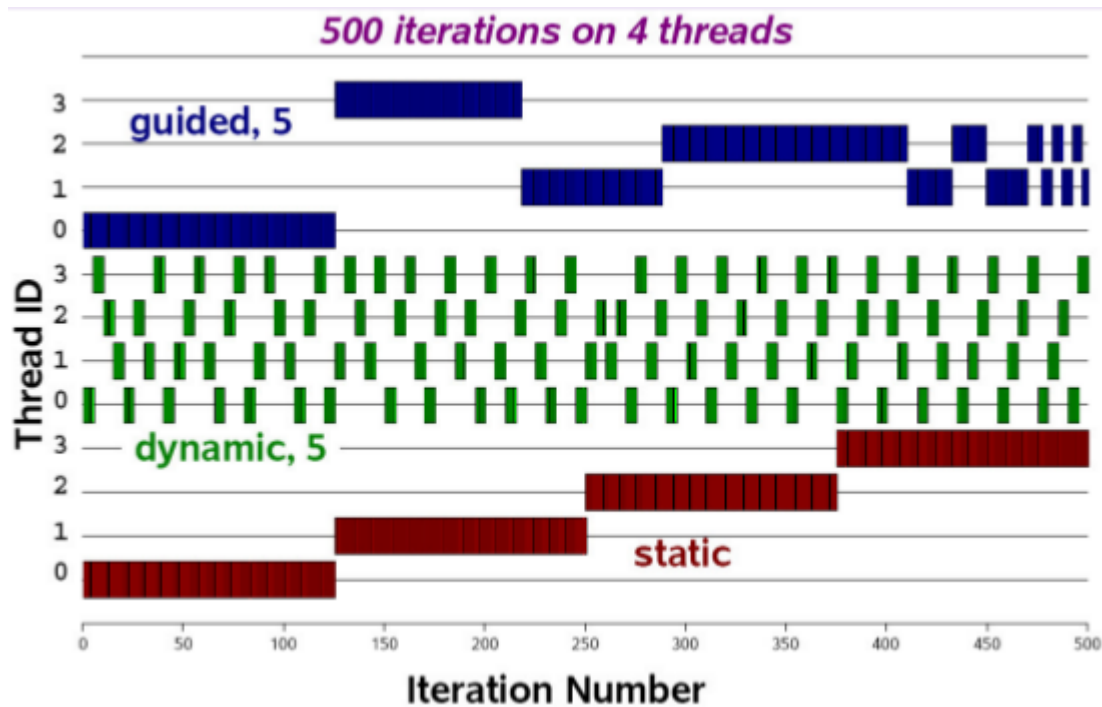
✓ `schedule(guided[,chunk])`

-Δυναμική ανάθεση τμημάτων εκθετικά μειούμενου μεγέθους. Μέγεθος πρώτου $N/\#\text{threads}$. Default chunk=1

✓ `schedule(runtime)`

-Η επιλογή της πολιτικής γίνεται κατά το χρόνο εκτέλεσης βάσει της τιμής της μεταβλητής περιβάλλοντος `OMP_SCHEDULE`

Example - the schedule clause



The loop worksharing Construct

- To loop worksharing construct μοιράζει τις επαναλήψεις στα threads

```
#pragma omp parallel
{
#pragma omp for
    for (l=0;l<N;l++){
        NEAT_STUFF(l);
    }
}
```

The variable `l` is made “private” to each thread by default. You could do this explicitly with a “private(`l`)” clause

A motivating example

Sequential code

```
for(i=0;i<N;i++) { a[i] = a[i] + b[i];}
```

OpenMP parallel region

```
#pragma omp parallel
{
    int id, i, Nthrds, istart, iend;
    id = omp_get_thread_num();
    Nthrds = omp_get_num_threads();
    istart = id * N / Nthrds;
    iend = (id+1) * N / Nthrds;
    if (id == Nthrds-1) iend = N;
    for(i=istart;i<iend;i++) { a[i] = a[i] + b[i];}
}
```

OpenMP parallel region and a worksharing for construct

```
#pragma omp parallel
#pragma omp for
    for(i=0;i<N;i++) { a[i] = a[i] + b[i];}
```

Combined parallel/for Construct

- OpenMP shortcut: βάζοντας το “parallel” και το “for” directive στην ίδια γραμμή

```
double res[MAX]; int i;
#pragma omp parallel
{
    #pragma omp for
    for (i=0;i< MAX; i++) {
        res[i] = huge();
    }
}
```

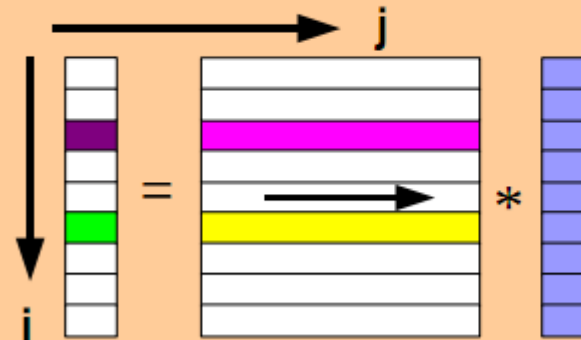
```
double res[MAX]; int i;
#pragma omp parallel for
for (i=0;i< MAX; i++) {
    res[i] = huge();
}
```

These are equivalent

Example - Matrix times vector

```
#pragma omp parallel for private(i,j,sum)
```

```
for (i=0; i<m; i++)  
{  
    sum = 0.0;  
    for (j=0; j<n; j++)  
        sum += b[i][j]*c[j];  
    a[i] = sum;  
}
```



TID=0

TID=1

```
for (i=0,1,2,3,4)
```

i = 0

$sum = \sum b[i=0][j]*c[j]$

$a[0] = sum$

i = 1

$sum = \sum b[i=1][j]*c[j]$

$a[1] = sum$

```
for (i=5,6,7,8,9)
```

i = 5

$sum = \sum b[i=5][j]*c[j]$

$a[5] = sum$

i = 6

$sum = \sum b[i=6][j]*c[j]$

$a[6] = sum$

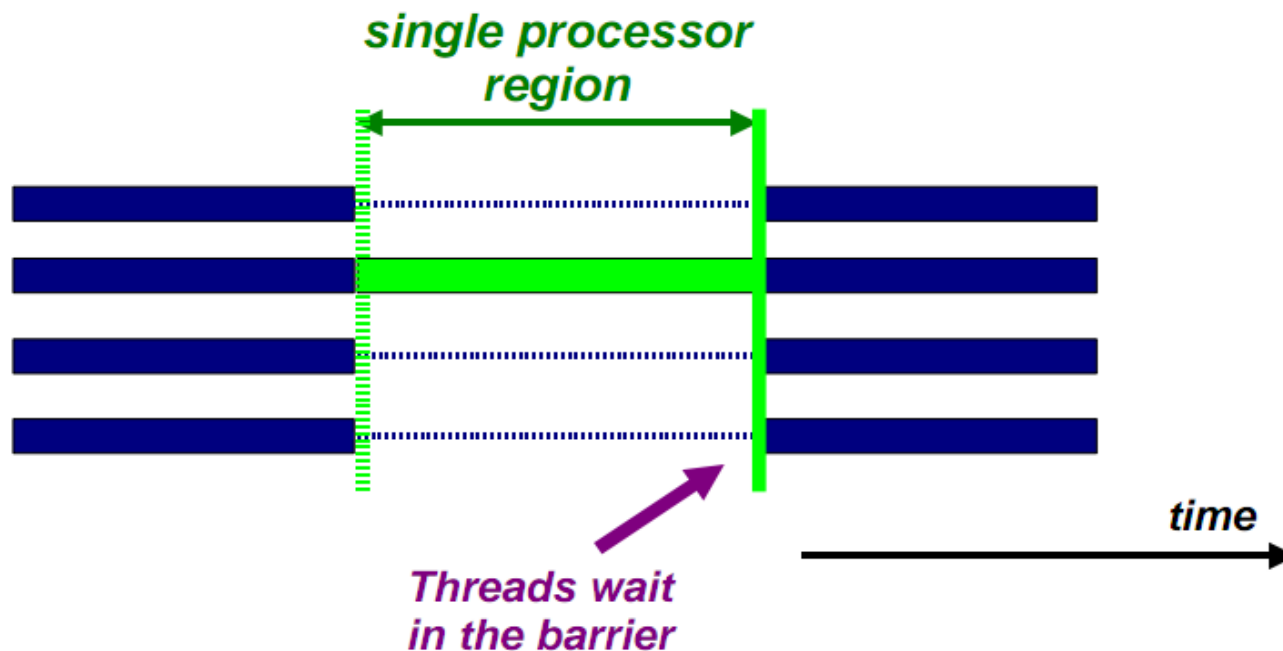
Single worksharing Construct

- ❑ Το single construct υποδηλώνει ότι ένα block κώδικα θα εκτελεστεί μόνο από ένα thread (όχι απαραίτητα το master thread).
- ❑ Ένα barrier “υπονοείται” στο τέλος του single block (μπορεί να αφαιρεθεί με ένα nowait clause).

```
#pragma omp single [private][firstprivate] \  
                    [nowait]  
{  
    <code-block>  
}
```

Single Construct - Example

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



Sections worksharing Constructs

- Όταν θέλουμε μεμονομένα blocks κώδικα να κατανέμονται ένα για κάθε thread

```
#pragma omp sections [clause(s)]  
{  
  #pragma omp section  
    <code block1>  
  #pragma omp section  
    <code block2>  
  #pragma omp section  
    :  
}
```

Clauses που υποστηρίζει:

- ✓private
- ✓firstprivate
- ✓lastprivate
- ✓reduction
- ✓nowait

Sections Construct - Example

```
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
            X_calculation();
        #pragma omp section
            y_calculation();
        #pragma omp section
            z_calculation();
    }
}
```

By default, there is a barrier at the end of the “omp sections”. Use the “nowait” clause to turn off the barrier.

Synchronization

□ Το synchronization χρησιμοποιείται για να επιβάλει περιορισμούς και να προστατέψει την πρόσβαση στα shared data

- High level synchronization

- ✓ critical
- ✓ atomic
- ✓ barrier
- ✓ ordered

- Low level synchronization

- ✓ flush

Critical and Atomic constructs

- ❑ Critical: Όλα τα threads εκτελούν τον κώδικα, αλλά μόνο ένα κάθε φορά (όχι ταυτόχρονα):

```
#pragma omp critical [(name)]  
{<code-block>}
```

There is no implied barrier on entry or exit !

- ❑ Atomic: Αξιοποιεί μηχανισμούς αμοιβαίου αποκλεισμού που βρίσκονται διαθέσιμοι σε επίπεδο υλικού

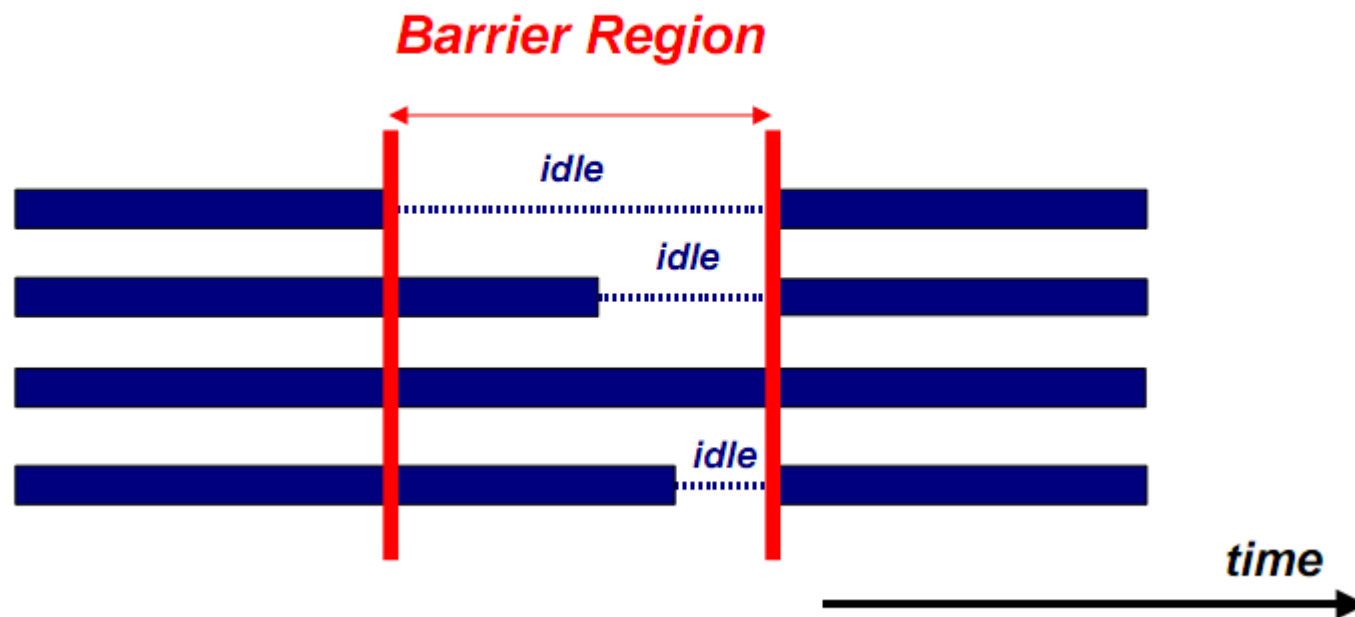
```
#pragma omp atomic  
<statement>
```

This is a lightweight, special form of a critical section

Barrier Construct

- Κάθε thread περιμένει ώσπου όλα τα threads φθάσουν.

```
#pragma omp barrier
```



About OpenMP clauses

- ❑ Όπως είδαμε πολλές OpenMP directives υποστηρίζουν clauses
- ❑ Τα clauses χρησιμοποιούνται για να προσδιορίσουν επιπρόσθετες πληροφορίες
- ❑ Για παράδειγμα, το `private(a)` είναι ένα clause για την `for` directive:
 - `#pragma omp for private(a)`
- ❑ Για κάθε directive χρησιμοποιούνται συγκεκριμένα clauses

The if/{first,last} private clauses

□ if (scalar expression)

- ✓ Εκτελείται παράλληλα μόνο αν `expression == true`
- ✓ αλλιώς, εκτελείται σειριακά

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

□ firstprivate (list)

- ✓ όλες οι μεταβλητές στη λίστα αρχικοποιούνται με τη τιμή που είχαν πριν μπουν στο `parallel construct`

□ lastprivate (list)

- ✓ ανάθεση της τελικής τιμής όπως αυτή προβλέπεται από την ακολουθιακή εκτέλεση

Example private variables

```
main()
{
    A = 10;

    #pragma omp parallel
    {
        #pragma omp for private(i) firstprivate(A) lastprivate(B) ...
        for (i=0; i<n; i++)
        {
            ....
            B = A + i;          /*-- A undefined, unless declared
                               firstprivate --*/
            ....
        }

        C = B;                /*-- B undefined, unless declared
                               lastprivate --*/
    } /*-- End of OpenMP parallel region --*/
}
```

The reduction clause

reduction (operator : list)

- ❑ Χρησιμοποιείται μέσα σένα parallel ή work-sharing construct:
 - Δημιουργείται ένα τοπικό αντίγραφο για κάθε thread και αρχικοποιείται ανάλογα με τον operator (0 για “+”).
 - Ο τελεστής εφαρμόζεται στο τοπικό αντίγραφο
 - Τα τοπικά αντίγραφα συνοψίζονται στην κοινή μεταβλητή.

- ❑ Οι μεταβλητές μέσα στη “list” πρέπει να είναι shared

Example reduction clause

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

int main(int argc, char** argv)
{
    int N = atoi(argv[1]);
    int sum = 0;
    int * A = malloc(N * sizeof(int));

    my_init(A);

    #pragma omp parallel for reduction(+ : sum) schedule(static)
        for (i=0; i < N; i++) {
            sum += A[i];
        }

    printf("Total sum = %d\n", sum);
}
```

Compiling

□ Σε περιβάλλον όπου παρέχεται ο GNU C compiler (π.χ. GNU C compiler > 4.2 version) συνήθως απαιτούνται :

- ✓ να συμπεριληφθεί το header file <omp.h>
- ✓ να συμπεριληφθεί στον compiler η σημαση -fopenmp

□ `gcc -fopenmp -o executable source.c`



References & Links

<http://openmp.org/mp-documents/ntu-vanderpas.pdf>

<http://openmp.org/mp-documents/omp-hands-on-SC08.pdf>

<http://parallel.hpclab.ceid.upatras.gr/front-6.pdf>

<http://www.openmp.org/mp-documents/spec30.pdf>

<http://www.ceid.upatras.gr/>

<http://patras-lug.gr/>

<http://www.p-space.gr/>

<http://openmp.org/>

The End!

Q & A

