

OpenMP

Η εισαγωγή αυτή είναι βασισμένη στο tutorial "Introduction to OpenMP" του Blaise Barney, Lawrence Livermore Laboratory, USA. Μετάφραση και προσαρμογή: Κ.Γ. Μαργαρίτης και Π. Γεωργιόπουλος, Εργαστήριο Παράλληλης Κατανεμημένης Επεξεργασίας, Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας.

Περιεχόμενα

- [Περίληψη](#)
- [Εισαγωγή](#)
 - [Τί είναι το OpenMP?](#)
 - [Ιστορικό](#)
 - [Στόχοι του OpenMP](#)
- [Το Μοντέλο Προγραμματισμού του OpenMP](#)
- [Οδηγίες Directives](#)
 - [Οδηγία PARALLEL](#)
 - [Οδηγίες Διαμοιρασμού Εργασίας](#)
 - [Οδηγία FOR](#)
 - [Οδηγία SECTIONS](#)
 - [Οδηγία SINGLE](#)
 - [Συνδυασμένες Οδηγίες Διαμοιρασμού Εργασίας](#)
 - Οδηγία PARALLEL FOR
 - Οδηγία PARALLEL SECTIONS
 - [Οδηγίες Συγχρονισμού](#)
 - [Οδηγία BARRIER](#)
 - [Οδηγία MASTER](#)
 - [Οδηγία CRITICAL](#)
 - [Οδηγία ATOMIC](#)
 - [Οδηγία FLUSH](#)
 - [Οδηγία ORDERED](#)
 - [Οδηγία THREADPRIVATE](#)
 - [Φράσεις Εμβέλειας Δηλώσεων Δεδομένων](#)
 - [Φράση PRIVATE](#)
 - [Φράση SHARED](#)
 - [Φράση DEFAULT](#)
 - [Φράση FIRSTPRIVATE](#)
 - [Φράση LASTPRIVATE](#)
 - [Φράση COPYIN](#)
 - [Φράση COPYPRIVATE](#)
 - [Φράση REDUCTION](#)
 - [Σύνοψη Φράσεων / Οδηγιών](#)
 - [Κανόνες Σύνδεσης και Ένθεσης Οδηγιών](#)
- [Ρουτίνες Συστήματος Εκτέλεσης](#)
 - [OMP_SET_NUM_THREADS](#)
 - [OMP_GET_NUM_THREADS](#)
 - [OMP_GET_MAX_THREADS](#)
 - [OMP_GET_THREAD_NUM](#)
 - [OMP_GET_THREAD_LIMIT](#)
 - [OMP_GET_NUM_PROCS](#)
 - [OMP_IN_PARALLEL](#)
 - [OMP_SET_DYNAMIC](#)
 - [OMP_GET_DYNAMIC](#)
 - [OMP_SET_NESTED](#)
 - [OMP_GET_NESTED](#)
 - [OMP_INIT_LOCK](#)
 - [OMP_DESTROY_LOCK](#)
 - [OMP_SET_LOCK](#)
 - [OMP_UNSET_LOCK](#)
 - [OMP_TEST_LOCK](#)
 - [OMP_INIT_NEST_LOCK](#)
 - [OMP_DESTROY_NEST_LOCK](#)
 - [OMP_SET_NEST_LOCK](#)
 - [OMP_UNSET_NEST_LOCK](#)
 - [OMP_TEST_NEST_LOCK](#)
 - [OMP_GET_WTIME](#)
 - [OMP_GET_WTICK](#)
- [Μεταβλητές Περιβάλλοντος](#)
- [Ζητήματα Μνήμης και Απόδοσης](#)
- [Αναφορές και Περισσότερες Πληροφορίες](#)
- [Ασκήσεις](#)

Το OpenMP είναι ένα πρότυπο παράλληλου προγραμματισμού, το οποίο δίνει στον χρήστη τη δυνατότητα να αναπτύξει παράλληλα προγράμματα για συστήματα μοιραζόμενης μνήμης, τα οποία είναι ανεξάρτητα από τη συγκεκριμένη αρχιτεκτονική και έχουν μεγάλη ικανότητα κλιμάκωσης. Με το πρότυπο αυτό, ο χρήστης μπορεί να δώσει εντολές και οδηγίες στον μεταγλωττιστή για να ορίσει μέρη του προγράμματος που επιθυμεί να εκτελεστούν παράλληλα, να ορίσει σημεία συγχρονισμού και άλλα. Προς το παρόν το OpenMP μπορεί να χρησιμοποιηθεί στις γλώσσες C/C++ και FORTRAN. Στο έγγραφο αυτό θα δούμε τη βασική αρχιτεκτονική του OpenMP, τις οδηγίες που παρέχει καθώς και κάποιους κώδικες, μέσα από τους οποίους γίνεται άμεσα αντιληπτή η χρησιμότητα του OpenMP.

Εισαγωγή

Τι είναι το OpenMP?

► Το OpenMP είναι:

Το OpenMP είναι μια Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface, API) το οποίο δημιουργήθηκε από κατασκευαστές υλικού και λογισμικού. Τα αρχικά αντιστοιχούν στη φράση "Open Specifications for Multi Processing"]. Το πρότυπο αυτό προσφέρει στον προγραμματιστή ένα σύνολο από οδηγίες, οι οποίες ενσωματώνονται στον κώδικα ενός προγράμματος, και έτσι μπορεί ο μεταγλωττιστής να επιτύχει παραλληλισμό στο πρόγραμμα αυτό, σύμφωνα πάντα με τις οδηγίες και παραμέτρους που έχει θέσει ο προγραμματιστής. Το API αυτό αποτελείται κυρίως από τρία βασικά συστατικά:



- Οδηγίες προς τον μεταγλωττιστή (compiler directives)
- Βιβλιοθήκη με συναρτήσεις (runtime routines)
- Μεταβλητές περιβάλλοντος (environment variables)

Το πρότυπο αυτό έχει ορισθεί για τις γλώσσες προγραμματισμού C/C++ και FORTRAN. Προς το παρόν δεν υπάρχουν σχέδια για ανάπτυξη σε άλλες γλώσσες προγραμματισμού [1]. Οι κώδικες σε OpenMP είναι μεταφέρσιμοι και υπάρχουν υλοποιήσεις για αρκετές πλατφόρμες, μέσα στις οποίες περιλαμβάνονται οι περισσότερες πλατφόρμες UNIX/Linux καθώς και Windows NT και μεταγενέστερα. Ο παραλληλισμός που προσφέρει αφορά συστήματα μοιραζόμενης μνήμης, οπότε δεν μπορεί από μόνο του να εφαρμοστεί σε συστήματα καταμεμημένης μνήμης. Επίσης οι εκάστοτε υλοποιήσεις δεν είναι απαραίτητα υλοποιημένες τέλεια, και προς το παρόν δεν μπορεί να κάνει την καλύτερη χρήση της μοιραζόμενης μνήμης, διότι δεν προσφέρει ακόμα οδηγίες για την τοπικότητα των δεδομένων.

Εισαγωγή

Ιστορικό

Η ιστορία του OpenMP ξεκινάει από τις αρχές της δεκαετίας του '90. Εκείνη τη εποχή, οι εταιρίες συστημάτων μοιραζόμενης μνήμης παρείχαν παρόμοιες επεκτάσεις προγραμμάτων Fortran, βασισμένες σε οδηγίες μεταγλωττιστή. Ο προγραμματιστής έδινε στον μεταφραστή ένα σειρικό πρόγραμμα Fortran με οδηγίες που καθόριζαν ποιοι βρόχοι θα παραλληλοποιούνταν. Στη συνέχεια, ο μεταφραγλωττιστής ήταν υπεύθυνος για την αυτόματη παραλληλοποίηση αυτών των βρόγχων σε συμμετρικούς επεξεργαστές (SMP's). Αν και οι υλοποιήσεις των διάφορων εταιριών ήταν παρόμοιες ως προς τη λειτουργικότητα, ωστόσο, διέφεραν αρκετά. Έτσι, έγινε μια προσπάθεια για να δημιουργηθεί ένα πρότυπο. Το πρώτο πρότυπο που δημιουργήθηκε ήταν το ANSI X3H5 το 1994, αλλά δεν υιοθετήθηκε επειδή εκείνη την εποχή το ενδιαφέρον για αρχιτεκτονικές μοιραζόμενης μνήμης έπεσε, ενώ ανέβηκε το ενδιαφέρον για συστήματα με αρχιτεκτονική καταμεμημένης μνήμης. Το 1997 προτάθηκε το πρότυπο OpenMP συνεχίζοντας από εκεί που είχε μείνει το ANSI X3H5, μιας και τώρα τα συστήματα μοιραζόμενης μνήμης άρχισαν να γίνονται δημοφιλέστερα. Εκείνη τη χρονιά είχε οριστεί μόνο το API για τη Fortran. Το 1998 ορίστηκε το API για τις γλώσσες C/C++, ενώ το 2000 φτιάχτηκε μια δεύτερη έκδοση για τη Fortran και το 2002 ακολούθησε η δεύτερη έκδοση για C/C++. Οι εταιρίες που συμμετέχουν επισήμως στις προδιαγραφές του OpenMP προτύπου είναι:

Εταιρίες Υλικού:

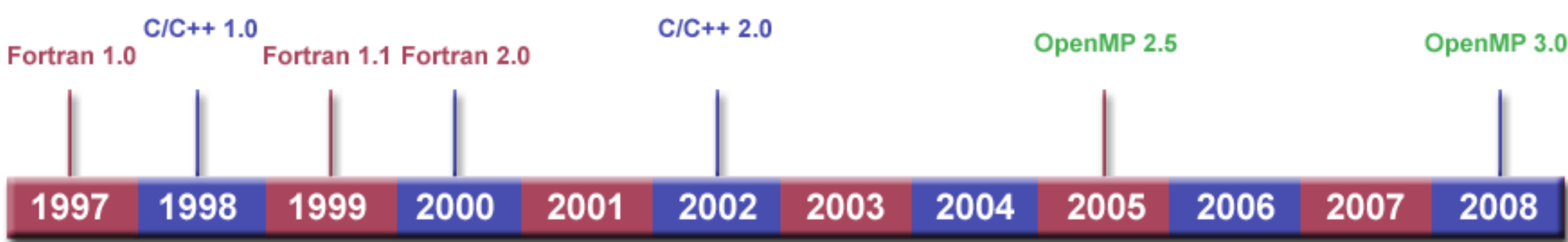
- Compaq / Digital
- Hewlett-Packard Company
- Intel Corporation
- International Business Machines (IBM)
- Kuck & Associates, Inc.
- Silicon Graphics, Inc.
- Sun Microsystems, Inc.
- U.S. Department of Energy ASCI program

Εταιρίες Λογισμικού:

- Absoft Corporation
- Edinburgh Portable Compilers
- GENIAS Software GmBH
- International Business Machines (IBM)
- Myrias Computer Technologies, Inc.
- The Portland Group, Inc. (PGI)

Ημερομηνία	Έκδοση
October 1997	Fortran version 1.0
Late 1998	C/C++ version 1.0
June 2000	Fortran version 2.0
April 2002	C/C++ version 2.0

► Ιστορικό εκδόσεων



Εισαγωγή

Στόχοι του OpenMP

Οι στόχοι του OpenMP είναι να παράσχει ένα πρότυπο που θα ισχύει για αρχιτεκτονικές και πλατφόρμες μοιραζόμενης μνήμης. Ένα πρότυπο που θα είναι μικρό και εύκολα κατανοητό, δηλαδή θα παρέχει ένα απλό και περιορισμένο σύνολο από οδηγίες με το οποίο θα μπορεί να γίνεται σημαντική παραλληλοποίηση. Επίσης, θα πρέπει να είναι εύκολο στη χρήση του. Το OpenMP παρέχει:

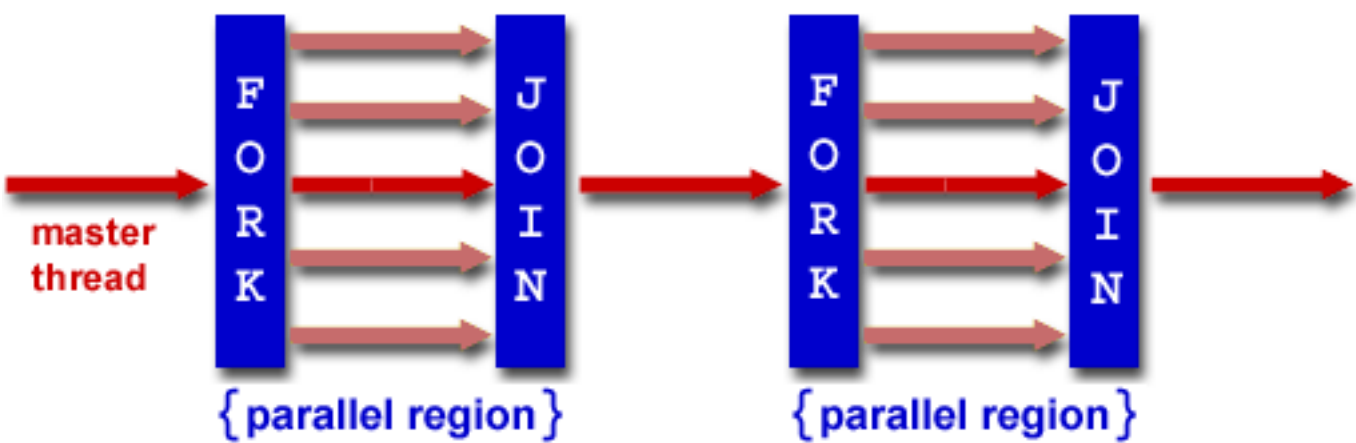
- α. Τη δυνατότητα παραλληλοποίησης ενός προγράμματος σταδιακά (εν αντιθέσει, για παράδειγμα, με το MPI όπου κάθε φορά γίνεται include όλη η βιβλιοθήκη).
- β. Τη δυνατότητα τόσο αδρομερούς (coarse-grain) όσο και λεπτομερούς (fine-grain) παραλληλισμού.

Τέλος, ο κυριότερος στόχος του OpenMP είναι η μεταφερτότητα. Υποστηρίζει Fortran (77, 90 και 95) και C/C++ ενώ έχει υλοποιηθεί για τις πιο πολλές από τις σημαντικές πλατφόρμες όπως Unix/Linux και Windows.

Το Μοντέλο Προγραμματισμού του OpenMP

Το μοντέλο προγραμματισμού του OpenMP είναι βασισμένο στο πολυνηματικό μοντέλο παραλληλισμού. Αρχικά, μία εφαρμογή σε OpenMP ξεκινά με ένα μόνο νήμα, το οποίο ονομάζεται *master thread*. Όταν το πρόγραμμα εισέρχεται σε μία περιοχή που έχει ορίσει ο προγραμματιστής να εκτελεστεί παράλληλα (*παράλληλη περιοχή, parallel region*), τότε δημιουργούνται αρκετά νήματα (fork-join μοντέλο) και το μέρος του κώδικα που βρίσκεται μέσα στη παράλληλη περιοχή εκτελείται παράλληλα. Όταν ολοκληρωθεί ο υπολογισμός της παράλληλης περιοχής όλα τα νήματα τερματίζουν και συνεχίζει μόνο το master thread. Στην εικόνα 1 φαίνεται ο τρόπος με τον οποίο λειτουργεί το μοντέλο αυτό.

Το OpenMP δεν μπορεί να εγγυηθεί ότι σε μία παράλληλη εκτέλεση ενός κώδικα η είσοδος και η έξοδος είναι συγχρονισμένη. Ο συγχρονισμός των νημάτων είναι ευθύνη του προγραμματιστή. Το OpenMP παρέχει αρκετές οδηγίες και συναρτήσεις συγχρονισμού, και πρέπει να χρησιμοποιηθούν σωστά από τον προγραμματιστή.




► Μοντέλος Μνήμης: Συχνό FLUSH;

Το μοντέλο μνήμης του OpenMP είναι ένα χαλαρό μοντέλο μοιραζόμενης μνήμης. Όλα τα νήματα έχουν πρόσβαση στη κύρια μνήμη. Κάθε νήμα μπορεί να έχει μια προσωρινή άποψη (temporary view) της μνήμης. Με αυτόν τον τρόπο, αν και δεν είναι υποχρεωτικό στο μοντέλο μνήμης του OpenMP, μπορεί να ενταμιεύσει (cache) κάποιες μεταβλητές και έτσι να αποφεύγεται η συνεχής προσπέλαση στη μνήμη. Επίσης κάθε νήμα έχει και την ιδιωτική του μνήμη (thread-private memory), όπου τα υπόλοιπα νήματα δεν έχουν πρόσβαση. Επιγραμματικά στο OpenMP ισχύουν τα εξής:

- Σε μια οδηγία παραλληλισμού μπορεί ορίζονται μοιραζόμενες και ιδιωτικές μεταβλητές. Κάθε αναφορά σε μοιραζόμενη μεταβλητή είναι μία αναφορά στην ίδια την μεταβλητή, ενώ μια αναφορά σε ιδιωτική μεταβλητή είναι μια αναφορά σε ένα τοπικό αντίγραφο στην ιδιωτική μνήμη του νήματος.
- Βέβαια για την προσπέλαση των μοιραζόμενων μεταβλητών από διαφορετικά νήματα απαιτείται συγχρονισμός, και όπως είπαμε νωρίτερα, αυτό είναι ευθύνη του προγραμματιστή.
- Είναι πιθανό σε κάποιες υλοποιήσεις η πρόσβαση σε μοιραζόμενες μεταβλητές να είναι ατομική ενέργεια και να μην χρειάζεται παρέμβαση από τον προγραμματιστή.
- Σε περίπτωση εμφωλευμένων οδηγιών παραλληλισμού μία μεταβλητή που είναι ιδιωτική στην εξωτερική παράλληλη περιοχή, μπορεί να είναι μοιραζόμενη στην εσωτερική παράλληλη περιοχή, εκτός αν έχει οριστεί στην εσωτερική οδηγία ως ιδιωτική.

Στη συνέχεια βλέπουμε τη βασική δομή σε έναν κώδικα C/C++ για τον ορισμό της παράλληλης περιοχής.

Παράδειγμα Γενικής Δομής Κώδικα OpenMP

 **C / C++ - General Code Structure**

```
#include <omp.h>

main () {
```



```
int var1, var2, var3;

Serial code
.
.
.

Beginning of parallel section. Fork a team of threads. Specify variable scoping.

#pragma omp parallel private(var1, var2) shared(var3)
{

    Parallel section executed by all threads
    .
    .
    .

    All threads join master thread and disband

}

Resume serial code
.
.
.

}
```

Οδηγίες OpenMP

Στη συνέχεια θα δούμε τις οδηγίες που παρέχει το OpenMP. Θα δούμε τις οδηγίες για τη γλώσσα C/C++. Πολλές από τις παραμέτρους είναι κοινές στις οδηγίες, και αυτές θα εξηγηθούν στο τέλος της ενότητας αυτής. Πριν μελετήσουμε ξεχωριστά τις οδηγίες που παρέχονται από το OpenMP, θα δούμε πρώτα έναν γενικό κανόνα για τον τρόπο που συντάσσονται οι οδηγίες.

► Σύνταξη οδηγιών C / C++:

#pragma omp	όνομα οδηγίας	[φράση (clause), ...]	αλλαγή γραμμής
Απαιτείται σε όλες τις οδηγίες OpenMP C/C++.	Έγκυρο όνομα οδηγίας OpenMP. Εμφανίζεται μετά το pragma omp και πριν τις φράσεις.	Προαιρετικές δηλώσεις. Μπορεί να βρίσκονται σε διαφορετική σειρά και να επαναλαμβάνονται όσες φορές απαιτείται.	Απαιτείται πάντα πριν από το δομημένο block που περικλείει η οδηγία.

► Παράδειγμα:

```
#pragma omp parallel default(shared) private(beta,pi)
```

► Γενικοί κανόνες:

- Είναι case sensitive
- Ακολουθούν τις συμβάσεις της C/C++ για τις οδηγίες προς το μεταγλωττιστή
- Σε κάθε οδηγία υπάρχει μόνο ένα όνομα οδηγίας
- Κάθε οδηγία εφαρμόζεται μόνο στο δομημένο block εντολών που ακολουθεί άμεσα.
- Μεγάλες γραμμές οδηγιών μπορούν να "συνεχιστούν" σε διαδοχικές γραμμές με χρήση backslash ("\") πριν την αλλαγή γραμμής

Οδηγίες OpenMP

Οδηγία PARALLEL

► Στόχος:

Αυτή η οδηγία είναι η βασικότερη από όλες, αφού όπως είδαμε και νωρίτερα αυτή είναι που ορίζει την περιοχή του κώδικα που επιθυμούμε να εκτελεστεί παράλληλα. Παρακάτω φαίνεται ο τρόπος σύνταξής της με όλες τις παραμέτρους.

► Μορφή:

C/C++	<pre>#pragma omp parallel [clause ...] newline if (scalar_expression) private (list) shared (list) default (shared none) firstprivate (list) reduction (operator: list) copyin (list) num_threads (integer-expression) structured_block</pre>
-------	--

Όταν ένα νήμα φτάσει σε μια οδηγία παραλληλης περιοχής, δημιουργεί μια ομάδα από νήματα και το ίδιο γίνεται ο master της

ομάδας. Ο master είναι κι ο ίδιος μέλος της ομάδας. Ο κώδικας της παράλληλης περιοχής αντιγράφεται τόσες φορές όσα τα νήματα και δίδεται σε αυτά για να τον εκτελέσουν. Γενικά, δεν υπάρχει συγχρονισμός μεταξύ των νημάτων. Κάθε νήμα μπορεί να φτάσει σε οποιοδήποτε σημείο μέσα στη παράλληλη περιοχή, σε ακαθόριστη χρονική στιγμή. Στο τέλος της παράλληλης περιοχής υπονοείται ένα φράγμα, πέρα από το οποίο μόνο ο master θα συνεχίσει την εκτέλεση του προγράμματος. Το OpenMP παρέχει τη δυνατότητα δημιουργίας παράλληλης περιοχής μέσα σε μια άλλη παράλληλη περιοχή (τακτική που πρέπει γενικά να αποφεύγεται). Η φωλιασμένη αυτή παράλληλη περιοχή, καταλήγει στη δημιουργία μιας καινούριας ομάδας από νήματα η οποία αποτελείται εξ' ορισμού από ένα νήμα. Ωστόσο, διάφορες υλοποιήσεις μπορούν να επιτρέψουν παραπάνω από ένα νήμα σε μια φωλιασμένη παράλληλη περιοχή.

Ο αριθμός των νημάτων που δημιουργούνται κατά την είσοδο σε μια παράλληλη περιοχή μπορεί να καθοριστεί από τρεις παράγοντες, οι οποίοι κατά σειρά προτεραιότητας είναι:

- α. Η χρήση της συνάρτησης βιβλιοθήκης **omp_set_num_threads()**.
- β. Η τιμή της μεταβλητής περιβάλλοντος **OMP_NUM_THREADS**.
- γ. Η χρήση της φράσης **default** (προκαθορισμένη υλοποίηση).

Επίσης, υπάρχει η δυνατότητα δυναμικής προσαρμογής του αριθμού των νημάτων για μία συγκεκριμένη παράλληλη περιοχή. Αυτό μπορεί να επιτευχθεί με τους εξής δύο τρόπους:

- α. Η χρήση της συνάρτησης βιβλιοθήκης **omp_set_dynamic()**.
- β. Η τιμή της μεταβλητής περιβάλλοντος **OMP_DYNAMIC**.

Η αρίθμηση των νημάτων ξεκινάει από τον αριθμό 0, οποίος δίνεται στον master κάθε ομάδας και φτάνει στον αριθμό N-1 όπου N ο αριθμός των νημάτων. Η ταυτότητα κάθε νήματος μπορεί να βρεθεί χρησιμοποιώντας τη συνάρτηση βιβλιοθήκης **omp_get_thread_num()**, ενώ ο συνολικός αριθμός των νημάτων μπορεί να βρεθεί με τη χρήση της συνάρτησης βιβλιοθήκης **omp_get_num_threads()**. Αν και τα νήματα εκτελούν το ίδιο τμήμα κώδικα, ωστόσο, θα θέλαμε να εκτελέσουν διαφορετικά μονοπάτια αυτού του κώδικα. Αυτό επιτυγχάνεται με το να δίνουμε διαφορετικό κομμάτι κώδικα σε κάθε νήμα (για παράδειγμα, με τη χρήση if-else, μοντέλο παραλληλισμού SPMD).

Στον κώδικα που ακολουθεί φαίνεται ένα απλό παράδειγμα. Αν υποθέσουμε ότι θα δημιουργηθούν πέντε νήματα, τότε στην έξοδο θα είναι το μήνυμα με το "Hello World" πέντε φορές αλλά το μήνυμα για το πλήθος των νημάτων θα το τυπώσει μόνο το master thread, και αυτό γιατί το id του είναι το 0.

Παράδειγμα: Παράλληλη Περιοχή

- Απλό πρόγραμμα "Hello World"
 - Κάθε νήμα εκτελεί το κώδικα που βρίσκεται στη παράλληλη περιοχή (έλεγχος εκτέλεσης με if)
 - Συναρτήσεις βιβλιοθήκης δίνουν το συνολικό αριθμό και το id κάθε νήματος



C / C++ - Parallel Region Example

```
#include <omp.h>

main () {

int nthreads, tid;

/* Fork a team of threads with each thread having a private tid variable */
#pragma omp parallel private(tid)
{

/* Obtain and print thread id */
tid = omp_get_thread_num();
printf("Hello World from thread = %d\n", tid);

/* Only master thread does this */
if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n", nthreads);
}

} /* All threads join master thread and terminate */

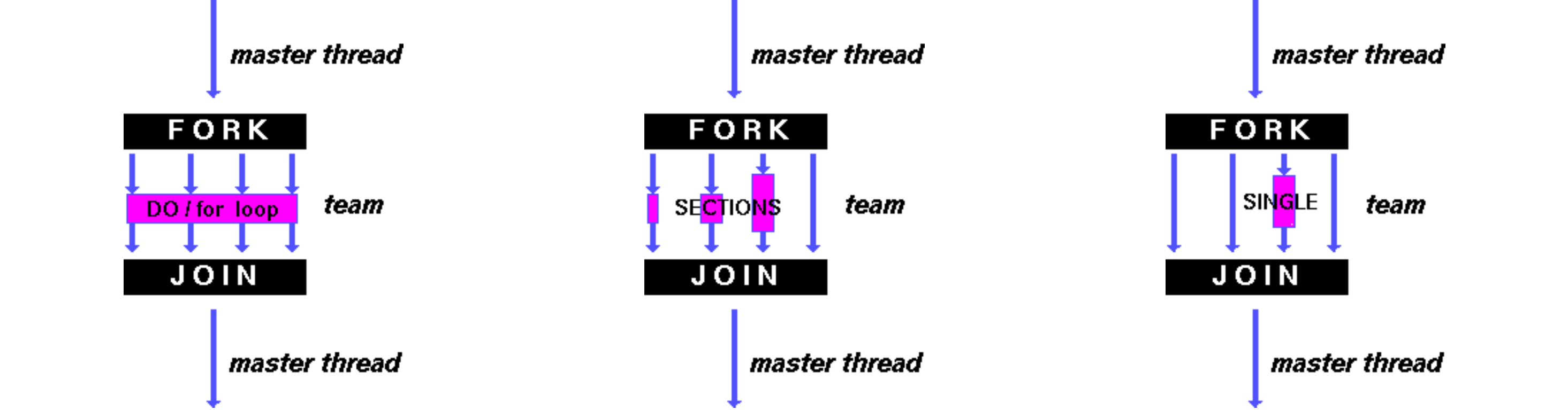
}
```

Οδηγίες OpenMP

Οδηγίες Διαμοιρασμού Εργασίας

Μια οδηγία διαμοιρασμού εργασίας κατανέμει την εκτέλεση του κώδικα που περικλείεται στην παράλληλη περιοχή μεταξύ των νημάτων της περιοχής. Οι οδηγίες διαμοιρασμού εργασίας δεν δημιουργούν καινούρια νήματα και δεν υπάρχει κάποιος συγχρονισμός κατά την είσοδο σε μια τέτοια οδηγία, υπάρχει όμως συγχρονισμός κατά την έξοδο (εισάγεται φράγμα). Υπάρχουν τρεις τύποι οδηγιών διαμοιρασμού εργασίας:

- | | | |
|--|--|---|
| FOR - Διαμοιράζει τις επαναλήψεις ενός βρόχου for στα νήματα της τρέχουσας παράλληλης περιοχής. Αντιστοιχεί στο μοντέλο Παραλληλισμού Δεδομένων (Data Parallelism). | SECTIONS - Διαμοιράζει την εργασία σε διακριτά δομικά blocks. Κάθε block εκτελείται από ένα νήμα. Αντιστοιχεί στο μοντέλο Συναρτησιακού Παραλληλισμού (Functional Parallelism). | SINGLE - Σειριοποιεί ένα τμήμα του κώδικα ώστε να εκτελεστεί υποχρεωτικά από ένα νήμα. |
|--|--|---|



► Περιορισμοί:

- Για να εκτελεσθεί παράλληλα μια οδηγία διαμοιρασμού εργασίας, θα πρέπει να εσωκλείεται σε μια παράλληλη περιοχή.
- Οι περιοχές διαμοιρασμού εργασίας θα πρέπει να διαμοιράζουν τα δεδομένα (ή την εργασία) σε όλα τα νήματα ή σε ένα.
- Διαδοχικές περιοχές διαμοιρασμού εργασίας θα πρέπει να προσπελαύνονται από τα μέλη μιας ομάδας με την ίδια σειρά.

Οδηγίες OpenMP

Οδηγίες Διαμοιρασμού Εργασίας
Οδηγία FOR

► Σκοπός:

Χρησιμοποιώντας την οδηγία αυτή μπορούμε να επιτύχουμε παραλληλισμό δεδομένων. Όλα τα νήματα θα εκτελέσουν το ίδιο τμήμα κώδικα, αλλά σε διαφορετικά δεδομένα. Αυτό είναι ιδιαίτερα χρήσιμο όταν γίνονται πράξεις με διανύσματα ή πίνακες. Προϋποθέτει ότι βρισκόμαστε μέσα σε παράλληλη περιοχή. Ακολουθεί η σύνταξη της οδηγίας:

► Μορφή:

C/C++	<pre>#pragma omp for [clause ...] newline schedule (type [,chunk]) ordered private (list) firstprivate (list) lastprivate (list) shared (list) reduction (operator: list) collapse (n) nowait for_loop</pre>
-------	---

► Φράσεις:

Αυτή η οδηγία χρησιμοποιείται μόνο για τον παραλληλισμό μιας εντολής for. Η φράση schedule ορίζει τον τρόπο με τον οποίο θα γίνει ο διαμοιρασμός των επαναλήψεων ανάμεσα στα νήματα. Το πρώτο όρισμα της φράσης μπορεί να είναι:

- **static:** Οι επαναλήψεις του βρόχου χωρίζονται σε μέρη μεγέθους chunk και μοιράζονται στατικά στα νήματα (πριν αρχίσει η εκτέλεση κάθε νήμα γνωρίζει πόσες και ποιές επαναλήψεις θα εκτελέσει). Αν δεν οριστεί το chunk, τότε γίνεται ισομεγέθους διαμοιρασμός, αν αυτό βέβαια είναι εφικτό.
- **dynamic:** Με αυτήν την παράμετρο οι επαναλήψεις χωρίζονται σε μέρη μεγέθους chunk. Αν δεν οριστεί το μέγεθος, κάθε μέρος έχει μία επανάληψη του κώδικα. Κάθε φορά που ένα νήμα ολοκληρώνει την εκτέλεση του μέρους που του έχει ανατεθεί, του ανατίθεται δυναμικά ένα άλλο μέρος.
- **guided:** Το μέγεθος κάθε μέρους μειώνεται εκθετικά, καθώς γίνονται οι αναθέσεις των κομματιών του χώρου επανάληψης. Η τιμή chunk ορίζει το μικρότερο κομμάτι στο οποίο μπορεί να σπάσει. Η προκαθορισμένη τιμή είναι 1.
- **runtime:** Η απόφαση σχεδιασμού αναβάλλεται μέχρι να αρχίσει να εκτελείται το πρόγραμμα. Με αυτήν την παράμετρο δεν επιτρέπεται να οριστεί το chunk.

Η φράση ordered εξηγείται παρακάτω. Τέλος, η φράση nowait, αν υπάρχει, αναιρεί το φράγμα που υπάρχει, όπως είπαμε νωρίτερα, στο τέλος της οδηγίαςFOR.

Για να εκτελεστεί σωστά η οδηγία αυτή, πρέπει να ισχύουν κάποιες συνθήκες για την εντολή for της C/C++. Η μεταβλητή βήματος πρέπει να αυξάνεται με μια σταθερή τιμή, και να μη προκύπτει ως τιμή μιας παράστασης που περιέχει μεταβλητές. Το ίδιο ισχύει για τη συνθήκη τερματισμού: πρέπει η τιμή που συγκρίνεται σε κάθε επανάληψη να είναι επίσης σταθερή. Γενικά ο βρόχος πρέπει να τερματίζει μετά από ένα πεπερασμένο αριθμό επαναλήψεων. Στην συνέχεια βλέπουμε ένα απλό παράδειγμα για πρόσθεση δύο διανυσμάτων. Η εντολή for είναι σε σωστή μορφή και οι επαναλήψεις χωρίζονται ανά 100. Η ανάθεση των τμημάτων γίνεται δυναμικά. Τέλος, τα νήματα δεν θα περιμένουν μετά την ολοκλήρωση των υπολογισμών, καθώς υπάρχει η φράση nowait.

Παράδειγμα: Οδηγία FOR

- Απλό πρόγραμμα πρόσθεσης ανυσμάτων
 - Οι πίνακες A, B, C, και η μεταβλητή N διαμοιράζονται σε όλα τα νήματα.

- Η μεταβλητή i είναι ιδιωτική σε κάθε νήμα: κάθε νήμα έχει δικό του αντίγραφο.
- Οι επαναλήψεις κατανέμονται δυναμικά σε τμήματα μεγέθους CHUNK.
- Τα νήματα δεν θα συγχρονιστούν στο τέλος της επανάληψης (NOWAIT).

 **C / C++ - for Directive Example**

```
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000

main ()
{

int i, chunk;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
    a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,chunk) private(i)
{

    #pragma omp for schedule(dynamic,chunk) nowait
    for (i=0; i < N; i++)
        c[i] = a[i] + b[i];

} /* end of parallel section */

}
```

Οδηγίες OpenMP

Οδηγίες Διαμοιρασμού Εργασίας

Οδηγία SECTIONS

► Σκοπός:

Η οδηγία sections είναι μια μη επαναληπτική περιοχή διαμοιρασμού εργασίας. Καθορίζει ότι τα εσωκλειώμενα τμήματα κώδικα θα διαμοιραστούν μεταξύ των νημάτων της ομάδας. Μια οδηγία sections μπορεί να περιέχει περισσότερες από μία, ανεξάρτητες, οδηγίες section. Κάθε τμήμα εκτελείται μια φορά από ένα νήμα της ομάδας, ενώ διαφορετικά τμήματα εκτελούνται από διαφορετικά νήματα. Η σύνταξη μιας οδηγίας sections φαίνεται παρακάτω:

► Μορφή:

C/C++	<pre>#pragma omp sections [<i>clause ...</i>] <i>newline</i> <i>private (list)</i> <i>firstprivate (list)</i> <i>lastprivate (list)</i> <i>reduction (operator: list)</i> <i>nowait</i> { #pragma omp section <i>newline</i> <i>structured_block</i> #pragma omp section <i>newline</i> <i>structured_block</i> }</pre>

Στο τέλος κάθε οδηγίας section υπονοείται κάποιο φράγμα που θα βοηθήσει στο συγχρονισμό των νημάτων, εκτός κι αν χρησιμοποιηθεί η φράση nowait οπότε τα νήματα δεν περιμένουν για συγχρονισμό μετά το πέρας της εργασίας τους.

Οι περιορισμοί που θα πρέπει να ισχύουν στις οδηγίες sections είναι οι εξής:

- όπως και στις οδηγίες for, απαγορεύεται μια διακλάδωση να εισέρχεται σε ένα block ή να καταλήγει έξω από αυτό.
- οι οδηγίες section πρέπει να λαμβάνουν χώρα μέσα σε μια οδηγία sections.
- μόνο μία συνθήκη nowait μπορεί να περιέχεται σε μια οδηγία sections

Παράδειγμα : Οδηγία SECTIONS

- Απλό πρόγραμμα που δείχνει διαφορετικά blocks να εκτελούνται από διαφορετικά νήματα. Ένα νήμα υλοποιεί πρόσθεση και το άλλο πολλαπλασιασμό στοιχείων ανυσμάτων.
 - Τα δεδομένα είναι διαμοιραζόμενα ή ιδιωτικά, όπως και στο προηγούμενο παράδειγμα
 - Όταν κάποιο νήμα τελειώσει με το block επαναλήψεων που του αντιστοιχεί, προχωρά χωρίς να περιμένει να τελειώσει και το δεύτερο νήμα (nowait).

--

```
#include <omp.h>
#define N      1000

main ()
{

int i;
float a[N], b[N], c[N], d[N];

/* Some initializations */
for (i=0; i < N; i++)
    a[i] = i * 1.5;
    b[i] = i + 22.35;

#pragma omp parallel shared(a,b,c,d) private(i)
{

    #pragma omp sections nowait
    {

        #pragma omp section
        for (i=0; i < N; i++)
            c[i] = a[i] + b[i];

        #pragma omp section
        for (i=0; i < N; i++)
            d[i] = a[i] * b[i];

    } /* end of sections */

} /* end of parallel section */

}
```

Oδηγίες OpenMP

Oδηγίες Διαμοιρασμού Εργασίας

Oδηγία SINGLE

► Σκοπός:

Η οδηγία single καθορίζει ότι ο κώδικας που εσωκλείεται από αυτή, θα εκτελεστεί μόνο από ένα νήμα. Το νήμα που θα φτάσει πρώτο στην single οδηγία, αυτό θα εκτελέσει τον κώδικα. Η οδηγία αυτή, είναι χρήσιμη όταν έχουμε τμήματα από κώδικα που δεν εξαρτώνται αποκλειστικά από τα νήματα όπως για παράδειγμα λειτουργίες I/O. Η σύνταξη μιας οδηγίας single φαίνεται παρακάτω:

► Μορφή:

C/C++	<pre>#pragma omp single [clause ...] newline private (list) firstprivate (list) nowait structured_block</pre>
-------	--

Τα νήματα που δεν εκτελούν την οδηγία, περιμένουν στο τέλος του εσωκλειώμενου κώδικα, εκτός αν χρησιμοποιηθεί η συνθήκη nowait οπότε τα νήματα δεν θα περιμένουν για να συγχρονιστούν.

► Περιορισμοί:

Οι περιορισμοί που ισχύουν είναι ότι απαγορεύεται μια διακλάδωση να εισέρχεται σε ένα block οδηγίας single ή να καταλήγει έξω από αυτό και ότι μόνο μία συνθήκη nowait μπορεί να χρησιμοποιηθεί σε κάθε οδηγία single.

Oδηγίες OpenMP

Συνδυασμένες Οδηγίες Διαμοιρασμού Εργασίας

Οι Συνδυασμένες Οδηγίες Διαμοιρασμού Εργασίας είναι συντομεύσεις για να καθορίσουμε μια οδηγία παράλληλης περιοχής η οποία περιέχει μόνο μία οδηγία διαμοιρασμού εργασίας (ένα παράλληλο for ή παράλληλα τμήματα, όπως τα δύο προηγούμενα παραδείγματα). Σημασιολογικά, οι συνδυασμένες παράλληλες οδηγίες διαμοιρασμού εργασίας, είναι ισοδύναμες, με τη οδηγία μιας παράλληλης περιοχής, αμέσως ακολουθούμενης από μία οδηγία διαμοιρασμού εργασίας. Επιτρέπονται όλες οι υπαρκτές φράσεις για μια παράλληλη περιοχή και για την σχετική οδηγία διαμοιρασμού εργασίας, εκτός από τη nowait αφού, έτσι κι αλλιώς, στο τέλος κάθε παράλληλης περιοχής υπονοείται ένα φράγμα για το συγχρονισμό των νημάτων. Αντίστοιχα με τις οδηγίες διαμοιρασμού εργασίας, έτσι κι εδώ, υπάρχουν οι εξής τύποι οδηγιών:

- α. Οδηγία parallel for.
- β. Οδηγία parallel sections
- γ. Η οδηγία single δεν θα είχε νόημα αφού μιλάμε πάντα για παράλληλες περιοχές διαμοιρασμού εργασίας.

Στην συνέχεια ακολουθεί δύο παραδείγματα κώδικα με τις οδηγίες parallel for και parallel sctions αντίστοιχα. Ο χαρακτήρας '\n' επιτρέπει να γραφτεί η οδηγία σε παραπάνω από μία γραμμές χωρίς να λαμβάνεται ο χαρακτήρας αλλαγής γραμμής, που είναι υποχρεωτικός για

Παράδειγμα: Οδηγία parallel for

 **C / C++ - parallel for Directive Example**

```
#include <omp.h>
#define N      1000
#define CHUNKSIZE  100


main ()  {

int i, chunk;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
    a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

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

• Παράδειγμα: Οδηγία parallel sections

 **C / C++ - parallel sections Directive Example**

```
#include <omp.h>
#define N      1000

main ()
{

int i;
float a[N], b[N], c[N], d[N];

/* Some initializations */
for (i=0; i < N; i++)
    a[i] = i * 1.5;
    b[i] = i + 22.35;

#pragma omp parallel sections nowait\
    shared(a,b,c,d) private(i)
{

    #pragma omp section
    for (i=0; i < N; i++)
        c[i] = a[i] + b[i];

    #pragma omp section
    for (i=0; i < N; i++)
        d[i] = a[i] * b[i];

} /* end of parallel sections */

}
```

Οδηγίες OpenMP

Οδηγίες Συγχρονισμού

Μέχρι τώρα είδαμε κάποιες οδηγίες οι οποίες εισάγουν παραλληλισμό στον κώδικα με διάφορους τρόπους. Βέβαια, για να είναι ο κώδικας σωστός πρέπει να είμαστε σίγουροι ότι δεν θα υπάρξουν προβλήματα αφού υπάρχουν δεδομένα τα οποία είναι κοινά σε όλα τα νήματα. Απαιτείται συγχρονισμός μεταξύ των νημάτων, έτσι ώστε το αποτέλεσμα του προγράμματος είναι το το αναμενόμενο. Θα δούμε τις οδηγίες που το OpenMP προσφέρει για το σκοπό αυτό, και στο τέλος της ενότητας υπάρχει ένας πίνακας με τη σύνταξη όλων των οδηγιών που θα περιγράψουμε στη συνέχεια.

- Έστω ένα απλό παράδειγμα όπου δύο νήματα προσπαθούν ταυτόχρονα να αυξήσουν μια διαμοιραζόμενη μεταβλητή x (έστω οτι η τιμή της x αρχικά είναι 0):

THREAD 1: increment(x) { x = x + 1; } THREAD 1: 10 LOAD A, (x address) 20 ADD A, 1 30 STORE A, (x address)	THREAD 2: increment(x) { x = x + 1; } THREAD 2: 10 LOAD A, (x address) 20 ADD A, 1 30 STORE A, (x address)
---	---

- Μια πιθανή σειρά εκτέλεσης:
 1. Thread 1 loads the value of x into register A.
 2. Thread 2 loads the value of x into register A.
 3. Thread 1 adds 1 to register A
 4. Thread 2 adds 1 to register A
 5. Thread 1 stores register A at location x
 6. Thread 2 stores register A at location x
- Η τελική τιμή του x θα είναι 1, όχι 2 όπως θα έπρεπε.
- Για να αποφύγουμε τέτοιες καταστάσεις, η αύξηση του x από τα δύο νήματα πρέπει να συγχρονιστεί.
 - Το OpenMP παρέχει αρκετές Οδηγίες Συγχρονισμού που ελέγχουν την εκτέλεση των εντολών ενός νήματος σε σχέση με άλλα νήματα.

Oδηγίες OpenMP

Oδηγίες Συγχρονισμού

Oδηγία BARRIER

► Purpose:

Η οδηγία barrier συγχρονίζει όλα τα νήματα της ομάδας απαιτώντας από κάθε νήμα να σταματήσει, προσωρινά την εκτέλεσή του, στο σημείο όπου υπάρχει η οδηγία barrier οδηγία, μέχρις ότου όλα τα νήματα φτάσουν σε αυτό το σημείο. Στη συνέχεια, όλα τα νήματα ξεκινούν παράλληλα, από εκείνο το σημείο, την εκτέλεσή του κώδικα που ακολουθεί. Η σύνταξη της οδηγίας barrier είναι η παρακάτω:

► Μορφή:

C/C++	<code>#pragma omp barrier newline</code>
-------	--

► Περιορισμοί:

Προκειμένου να εφαρμοστεί σωστά η οδηγία barrier πρέπει να ισχύουν οι ακόλουθοι περιορισμοί:

- α. Κάθε οδηγία barrier σε μια παράλληλη περιοχή, πρέπει να αντιμετωπιστεί από όλα τα νήματα ή από κανένα.
- β. Η σειρά με την οποία τα νήματα προσπελαίνουν τις οδηγίες διαμοιρασμού εργασίας και τις οδηγίες barrier σε μια παράλληλη περιοχή, πρέπει να είναι ίδια για κάθε νήμα της ομάδας.

Oδηγίες OpenMP

Oδηγίες Συγχρονισμού

Oδηγία MASTER

► Σκοπός:

Η οδηγία αυτή ορίζει ένα τμήμα κώδικα το οποίο θα εκτελεστεί από το mater thread μόνο. Δεν υπάρχει κάποιο φράγμα στο τέλος του για τα υπόλοιπα νήματα, τα οποία απλά προσπερνούν την οδηγία αυτή.

► Μορφή:


C/C++	<code>#pragma omp master newline structured_block</code>
-------	---

► Περιορισμοί:

- Δεν επιτρέπονται διακλαδώσεις έξω απο το μπλοκ της master.

Παράδειγμα: Οδηγίες master και barrier

Στο παράδειγμα που ακολουθεί η εκτύπωση γίνεται μόνο από το master thread. Η οδηγία barrier αναγκάζει τα υπολοιπα νήματα να περιμένουν την εκτύπωση ώστε όλα μαζί να ξεκινήσουν το δεύτερο τμήμα των υπολογισμών.

 C / C++ - master and barrirer Directives Example

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

int main( )
{
    int a[5], i;

    #pragma omp parallel
    {
```

```
// Perform some computation.
#pragma omp for
for (i = 0; i < 5; i++)
    a[i] = i * i;

// Print intermediate results.
#pragma omp master
for (i = 0; i < 5; i++)
    printf_s("a[%d] = %d\n", i, a[i]);

// Wait.
#pragma omp barrier

// Continue with the computation.
#pragma omp for
for (i = 0; i < 5; i++)
    a[i] += i;
}
```

Οδηγίες OpenMP

Οδηγίες Συγχρονισμού
Οδηγία CRITICAL

► Σκοπός:

Η οδηγία critical καθορίζει μια περιοχή η οποία πρέπει να εκτελεστεί μόνο από ένα νήμα κάθε φορά. Κυρίως χρησιμοποιείται για να ορίσουμε μια κρίσιμη περιοχή (critical region). Σε μια κρίσιμη περιοχή, μόνο μια διεργασία μπορεί να γράψει ή να διαβάσει μια μοιραζόμενη μεταβλητή διασφαλίζοντας έτσι την ακεραιότητα αυτής της μεταβλητής. Η σύνταξη της οδηγίας critical είναι η παρακάτω:

► Μορφή:

C/C++	<pre>#pragma omp critical [name] newline structured_block</pre>
-------	---

► Σημειώσεις:

Το name είναι ένα αναγνωριστικό όνομα της κρίσιμης περιοχής που καθορίζει η οδηγία critical (κάθε κρίσιμη περιοχή πρέπει να έχει μοναδικό αναγνωριστικό όνομα). Οι κρίσιμες περιοχές που δεν έχουν όνομα θεωρείται ότι έχουν την ίδια ταυτότητα, ενώ διαφορετικές critical οδηγίες που έχουν το ίδιο όνομα, θεωρείται ότι αναφέρονται στην ίδια κρίσιμη περιοχή. Αν ένα νήμα εκτελεί μια κρίσιμη περιοχή και ένα άλλο νήμα φτάσει σε αυτή την κρίσιμη περιοχή και προσπαθήσει να την εκτελέσει, τότε το δεύτερο νήμα θα αναγκαστεί να περιμένει μέχρι το πρώτο νήμα να φύγει από την κρίσιμη περιοχή.

► Περιορισμοί:

- Απαγορεύονται οι διακλαδώσεις μέσα ή έξω απο ένα μπλοκ CRITICAL.

Παράδειγμα: Οδηγία CRITICAL

- Όλα τα νήματα της παράλληλης περιοχής θα προσπαθήσουν να εκτελεστούν παράλληλα, όμως η οδηγία CRITICAL που εσωκλείει την αύξηση του x, επιτρέπει την εκτέλεση ενός νήματος κάθε φορά

 C / C++ - critical Directive Example

```
#include <omp.h>

main()
{

    int x;
    x = 0;

    #pragma omp parallel shared(x)
    {

        #pragma omp critical
        x = x + 1;

    } /* end of parallel section */

}
```

Οδηγίες OpenMP

Οδηγίες Συγχρονισμού

Οδηγία ATOMIC

► Σκοπός:

Η οδηγία atomic καθορίζει ότι η συγκεκριμένη θέση μνήμης πρέπει να ενημερώνεται ατομικά (atomic action), μην επιτρέποντας πολλά νήματα να ενημερώσουν ταυτόχρονα τη συγκεκριμένη θέση μνήμης. Έτσι απαγορεύει σε οποιοδήποτε νήμα να διακόψει κάποιο άλλο νήμα που βρίσκεται στη διαδικασία προσπέλασης ή αλλαγής της τιμής μιας μεταβλητής μοιραζόμενης μνήμης. Η σύνταξη της οδηγίας atomic είναι η παρακάτω:

► Μορφή:


C/C++	<pre>#pragma omp atomic newline statement_expression</pre>
-------	--

► Περιορισμοί:

- Αυτή η οδηγία μπορεί να εφαρμοστεί μόνο σε μία απλή εντολή C/C++ που ακολουθεί άμεσα.
- Η εντολή C/C++ έχει πολύ συγκεκριμένη σύνταξη(πχ δε μπορεί να είναι κλήση σε συνάρτηση).

Παράδειγμα: οδηγία ATOMIC

- Όλα τα νήματα της παράλληλης περιοχής θα αυξήσουν με τη σειρά το μετρητή count, ώστε τελικά ο count θα έχει τιμή ίση με τον αριθμό των νημάτων.

 C / C++ - atomic Directive Example

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

int main() {
    int count = 0;
    {
        #pragma omp atomic
        count++;
    }
    printf_s("Number of threads: %d\n", count);
}
```

Οδηγίες OpenMP

Οδηγίες Συγχρονισμού Οδηγία FLUSH

► Σκοπός:

Η οδηγία flush καθορίζει ένα σημείο συγχρονισμού στο οποίο η υλοποίηση του κώδικα πρέπει να παρέχει ένα συνεπές στιγμιότυπο της μνήμης. Σε αυτό το σημείο η τρέχουσα τιμή μιας μοιραζόμενης μεταβλητής εγγράφεται άμεσα στη μνήμη από τη cache (write back). Η σύνταξη της οδηγίας flush είναι η παρακάτω:

► Μορφή:

C/C++	<pre>#pragma omp flush (var1, var2, ...) newline</pre>
-------	---

► Σημειώσεις:

Οι var1, var2, ...είναι μια λίστα από μοιραζόμενες μεταβλητές οι οποίες θα εγγραφούν άμεσα στη μνήμη προκειμένου να αποφύγουμε να εγγράψουμε όλες τις μοιραζόμενες μεταβλητές. Αν κάποιο από τα var1, var2, ...είναι δείκτης, τότε εγγράφεται ο δείκτης και όχι το αντικείμενο στο οποίο δείχνει.

- Αν δεν υπάρχει αυτή η λίστα, η οποία είναι προαιρετική, τότε εγγράφονται όλες οι μοιραζόμενες μεταβλητές στη μνήμη.
- Οι υλοποιήσεις των προγραμμάτων πρέπει να εγγυώνται ότι οποιεσδήποτε αλλαγές σε μεταβλητές ορατές από τα νήματα, θα είναι ορατές στα νήματα μετά από αυτό το σημείο.
- Η οδηγία flush υπονοείται για τις ακόλουθες οδηγίες (εκτός αν υπάρχει στον κώδικα η συνθήκη nowait):

C / C++
<pre>barrier parallel - στην είσοδο και έξοδο critical - στην είσοδο και στην έξοδο ordered - στην είσοδο και στην έξοδο for - στην έξοδο sections - στην έξοδο single - στην έξοδο</pre>

Oδηγίες Συγχρονισμού

Oδηγία ORDERED

► Σκοπός:

Η οδηγία `ordered` καθορίζει ότι οι επαναλήψεις του εσωκλειώμενου βρόγχου θα εκτελεστούν με τη σειρά όπως θα εκτελούνταν σε ένα σειριακό υπολογιστή. Η σύνταξη της οδηγίας `ordered` είναι η παρακάτω:

► Μορφή:

C/C++	<pre>#pragma omp for ordered [<i>clauses...</i>] (<i>loop region</i>) #pragma omp ordered <i>newline</i> <i>structured_block</i> (<i>endo of loop region</i>)</pre>
-------	---

Όταν ένα νήμα το οποίο πρόκειται να εκτελέσει την πρώτη επανάληψη του βρόγχου, συναντήσει μια οδηγία `ordered`, τότε προχωρά στην εκτέλεση χωρίς να περιμένει. Αντίθετα, όταν νήματα που εκτελούν επόμενες επαναλήψεις, συναντήσουν την ίδια οδηγία `ordered`, τότε περιμένουν στην αρχή της `ordered` περιοχής μέχρι να τελειώσουν από αυτή την περιοχή όλα τα νήματα που εκτελούν προηγούμενες επαναλήψεις (έτσι, μόνο ένα νήμα επιτρέπεται να είναι σε μια `ordered` περιοχή κάθε φορά).


► Περιορισμοί:

Οι περιορισμοί που πρέπει να ισχύουν σε μια οδηγία `ordered` είναι οι εξής:

- α. Μια οδηγία `ordered` μπορεί να εμφανίζεται μόνο σε μια οδηγία `for` ή `parallel for`.
- β. Δεν επιτρέπεται η διακλάδωση εκτός `ordered for` βρόχου.
- γ. Μια επανάληψη ενός βρόχου δεν πρέπει να εκτελεί την ίδια `ordered` οδηγία παραπάνω από μια φορά και δεν πρέπει να εκτελεί πάνω από μία `ordered` οδηγίες.

Παράδειγμα: οδηγία ORDERED

- Και στα δύο `for` οι εκτυπώσεις γίνονται με την σωστή σειρά. Στη δεύτερη περίπτωση απαιτείται διπλή χρήση της οδηγίας `ordered` επειδή δημιουργούνται πολλά παράλληλα νήματα όπου το καθένα έχει δική του τιμή για το `iter`.

 C / C++ - ordered Directive Example

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

static float a[1000], b[1000], c[1000];

void test(int first, int last)
{
    int i;
    #pragma omp for schedule(static) ordered
    for (i = first; i <= last; ++i) {
        // Do something here.
        if (i % 2)
            printf("test() iteration %d\n", i);
    }
}

void test2(int iter)
{
    #pragma omp ordered
    printf("test2() iteration %d\n", iter);
}

int main( )
{
    int i;
    #pragma omp parallel
    {
        test(1, 8);
        #pragma omp for ordered
        for (i = 0 ; i < 5 ; i++)
            test2(i);
    }
}
```

Oδηγία THREADPRIVATE

► Σκοπός:

Η οδηγία threadprivate καθορίζει ότι καθολικά αντικείμενα (ή μεταβλητές) μπορούν να γίνουν προσωρινά ιδιωτικά για κάποιο νήμα. Με αυτό τον τρόπο, μπορούμε να ορίσουμε καθολικά αντικείμενα, αλλά να μετατρέψουμε την εμβέλειά τους και να τα κάνουμε τοπικά για κάποιο νήμα. Οι μεταβλητές για τις οποίες ισχύει η οδηγία threadprivate συνεχίζουν να είναι ιδιωτικές, για κάθε νήμα, ακόμα και σε διαφορετικές παράλληλες περιοχές. Η σύνταξη της threadprivate οδηγίας είναι η παρακάτω:


► Μορφή:

C/C++

#pragma omp threadprivate (list)

► Notes:

- Η οδηγία πρέπει να εμφανίζεται αμέσως μετά τις δηλώσεις των καθολικών μεταβλητών (άρα πριν τη main()). Μετά την οδηγία, η συγκεκριμένη μεταβλητή που έχει γίνει threadprivate, αντιγράφεται σε κάθε νήμα και κάθε νήμα κρατάει το δικό του αντίγραφο, έτσι ώστε δεδομένα τα οποία εγγράφονται από ένα νήμα να μην είναι ορατά στα υπόλοιπα. Για παράδειγμα:

 **C/C++ - threadprivate Directive Example**

```
#include <omp.h>

int  a, b, i, tid;
float x;

#pragma omp threadprivate(a, x)

main () {

/* Explicitly turn off dynamic threads */
omp_set_dynamic(0);

printf("1st Parallel Region:\n");
#pragma omp parallel private(b,tid)
{
tid = omp_get_thread_num();
a = tid;
b = tid;
x = 1.1 * tid +1.0;
printf("Thread %d:  a,b,x= %d %d %f\n",tid,a,b,x);
} /* end of parallel section */

printf("*****\n");
printf("Master thread doing serial work here\n");
printf("*****\n");

printf("2nd Parallel Region:\n");
#pragma omp parallel private(tid)
{
tid = omp_get_thread_num();
printf("Thread %d:  a,b,x= %d %d %f\n",tid,a,b,x);
} /* end of parallel section */

}
```

Output:

1st Parallel Region:
Thread 0: a,b,x= 0 0 1.000000
Thread 2: a,b,x= 2 2 3.200000
Thread 3: a,b,x= 3 3 4.300000
Thread 1: a,b,x= 1 1 2.100000

Master thread doing serial work here

2nd Parallel Region:
Thread 0: a,b,x= 0 0 1.000000
Thread 3: a,b,x= 3 0 4.300000
Thread 1: a,b,x= 1 0 2.100000
Thread 2: a,b,x= 2 0 3.200000

- Κατά την πρώτη είσοδο σε μια παράλληλη περιοχή, οι τιμές των threadprivate μεταβλητών θεωρούνται ακαθόριστες, εκτός κι αν έχει χρησιμοποιηθεί στην parallel (section ή for) οδηγία, η φράση copyin.
- Οι μεταβλητές THREADPRIVATE διαφέρουν απο τις ιδιωτικές, PRIVATE, μεταβλητές (που θα δούμε στη συνέχεια) επειδή διατηρούνται μεταξύ διαφορετικών παράλληλων περιοχών (parallel sections) του κώδικα.

► Περιορισμοί:

Για να συνεχίσουν οι μεταβλητές να είναι ιδιωτικές για όλες τις παράλληλες περιοχές, θα πρέπει να ισχύουν οι παρακάτω συνθήκες:

- α. Δεν υπάρχει παράλληλη περιοχή μέσα σε μια άλλη παράλληλη περιοχή.
- β. Ο αριθμός των νημάτων στις παράλληλες περιοχές πρέπει να είναι ο ίδιος.
- γ. Ο δυναμικός σχηματισμός νημάτων πρέπει να είναι “turned off” στη πρώτη παράλληλη περιοχή και να παραμένει ο ίδιος στις υπόλοιπες παράλληλες περιοχές

Φράσεις Εμβέλειας Δηλώσεων Δεδομένων

- Λέγονται και Φράσεις **Διαμοιρασμού Δεδομένων**
- Στο προγραμματισμό συστημάτων μοιραζόμενης μνήμης (όπως με το OpenMP) είναι σημαντική η κατανόηση και σωστή χρήση της εμβέλειας των δηλώσεων δεδομένων.
- Εφ' όσον το OpenMP υλοποιεί μοντέλο προγραμματισμού συστήματα μοιραζόμενης μνήμης, οι περισσότερες μεταβλητές είναι εξ' ορισμού μοιραζόμενες.
- Οι καθολικές μεταβλητές είναι γενικά μοιραζόμενες.
 - Εξαιρούνται ο μετρητές βρόχων for που είναι εξ' ορισμού ιδιωτικές
 - Επίσης ιδιωτικές είναι οι τοπικές μεταβλητές και τα ορίσματα συναρτήσεων που καλούνται μέσα σε παράλληλες περιοχές
- Οι Φράσεις Εμβέλειας Δηλώσεων Δεδομένων του OpenMP χρησιμοποιούνται για τον ρητό καθορισμό της εμβέλειας και του διαμοιρασμού των δεδομένων. Είναι οι εξής:
 - PRIVATE
 - FIRSTPRIVATE
 - LASTPRIVATE
 - SHARED
 - DEFAULT
 - REDUCTION
 - COPYIN
- Χρησιμοποιούνται σε συνδυασμό με διάφορες οδηγίες (PARALLEL, FOR, και SECTIONS).
- Παρέχουν τη δυνατότητα ελέγχου της εμβέλειας και του διαμοιρασμού των εμπλεκόμενων μεταβλητών:
 - Ορίζουν ποιές μεταβλητές της ακολουθιακής περιοχής μεταφέρουν τις τιμές τους προς/από τη παράλληλη περιοχή και με ποιό τρόπο.
 - Ορίζουν ποιές μεταβλητές θα είναι ορατές σε όλα τα νήματα και ποιές θα ανήκουν σε συγκεκριμένο νήμα.
- Οι Φράσεις έχουν ισχύ μόνο στα όρια των οδηγιών που συνοδεύουν.

Φράση PRIVATE

► Σκοπός:

Με την φράση αυτή μπορούμε να ορίσουμε μεταβλητές ιδιωτικές για κάθε νήμα. Αυτό σημαίνει ότι κάθε νήμα έχει ένα δικό της αντίγραφο της μεταβλητής στην ιδιωτική της μνήμη. Όλες οι αναφορές στις private μεταβλητές αλλάζουν και μετατρέπονται σε αναφορές στα αντίγραφα των μεταβλητών.Τα αντίγραφα των μεταβλητών δεν αρχικοποιούνται στη δημιουργία των νημάτων, εκτός και αν χρησιμοποιηθεί η φράση FIRSTPRIVATE.

► Μορφή:

C/C++	private (<i>list</i>)
-------	-------------------------

► Σημειώσεις:

- Σύγκριση μεταξύ της PRIVATE και THREADPRIVATE:

	PRIVATE	THREADPRIVATE
Δεδομένα	C/C++: μεταβλητή	C/C++: μεταβλητή
Που Δηλώνεται	Στην αρχή της παράλληλης περιοχής	Στην αρχή του προγράμματος, μαζί με τις καθολικές μεταβλητές
Διατήρηση τιμής?	Οχι	Ναι
Εμβέλεια	Τοπική μεταβλητή ή παράμετρος συνάρτησης	Ουσιαστικά καθολική μεταβλητή
Αρχικοποίηση	Με FIRSTPRIVATE	Με COPYIN

Φράση SHARED

► Σκοπός:

Σε αντίθεση με την προηγούμενη φράση, η shared ορίζει μεταβλητές ως διαμοιραζόμενες σε όλα τα νήματα. Αυτό σημαίνει ότι αν ένα νήμα αλλάξει την τιμή μιας τέτοιας μεταβλητής, η αλλαγή θα είναι ορατή σε όλα τα νήματα. Τα νήματα δεν έχουν τοπικό αντίγραφο της μεταβλητής αυτής, χρησιμοποιούν την θέση της στη διαμοιραζόμενη μνήμη. Ο προγραμματιστής είναι υπεύθυνος για να φροντίσει να μην δημιουργήσουν προβλήματα ταυτόχρονες αναγνώσεις και εγγραφές της ίδιας μεταβλητής από πολλά νήματα. Οι shared μεταβλητές διατηρούν την τιμή τους κατά την έξοδο από παράλληλη περιοχή.

► Μορφή:

C/C++	shared (<i>list</i>)
-------	------------------------

Φράση DEFAULT

► Σκοπός:

Η φράση αυτή χρησιμοποιείται για να ορίσει ένα προκαθορισμένο τύπο πρόσβασης για όλες τις μεταβλητές σε μια παράλληλη περιοχή περιοχές του κώδικα. Για την C/C++ δεν υπάρχει η δυνατότητα για να οριστούν private μεταβλητές μέσω της default αλλά μπορεί κάποιες υλοποιήσεις να προσφέρουν τη δυνατότητα αυτή. Περιορισμένη χρήση.

► Μορφή:

C/C++	default (shared none)
-------	-------------------------

Φράση FIRSTPRIVATE Clause

► Σκοπός:

Συνδυάζει τη λειτουργία της φράσης PRIVATE με τη δυνατότητα αρχικοποίησης των μεταβήτών κατά την είσοδο στη παράλληλη περιοχή.

► Μορφή:

C/C++	firstprivate (<i>list</i>)
-------	------------------------------

► Σημειώσεις:

- Τα αντίγραφα της μεταβλητής FIRSTPRIVATE αρχικοποιούνται στη τιμή που είχε η μεταβλητή πριν την είσοδο στη παράλληλη περιοχή.

Φράση LASTPRIVATE

► Σκοπός:

Ακριβώς το αντίθετο με την FIRSTPRIVATE κανει η δήλωση LASTPRIVATE. Στην περίπτωση αυτή μια μεταβλητή LASTPRIVATE διατηρεί, κατά την έξοδο απο τη παράλληλη περιοχή, την τελευταία τιμή που απέκτησε κάποιο από τα αντίγραφά της.

► Μορφή:

C/C++	lastprivate (<i>list</i>)
-------	-----------------------------

► Σημείωση:

- Η τιμή που θα δοθεί θα είναι η τιμή που πήρε η μεταβλητή στην εκτέλεση της τελευταίας (με λεξικογραφική έννοια) παράλληλης εκτέλεσης. Αυτό ισχύει τόσο σε οδηγίες parallel for, αλλά και σε οδηγίες parallel sections, δηλαδή αυτή θα κρατηθεί η τιμή από το τελευταίο for loop ή το τελευταίο section.

Φράση COPYIN

► Σκοπός:

Με την παράμετρο αυτή μπορούμε να δώσουμε αρχική τιμή σε threadprivate μεταβλητές. Όταν τα νήματα μπαίνουν σε μία παράλληλη περιοχή, οι threadprivate μεταβλητές δεν έχουν κάποια αρχική τιμή. Με την χρήση της παραμέτρου αυτής, μiai threadprivate μεταβλητή αρχικοποιείται στην τιμή που έχει η μεταβλητή στο master thread.

► Μορφή:

C/C++	copyin (<i>list</i>)
-------	------------------------

Φράση COPYPRIVATE

► Σκοπός:


- Η φράση COPYPRIVATE χρησιμοποιείται για την διανομή τιμών μεταβλητής από ένα νήμα σε όλα τα αντίγραφα της μεταβλητής στα υπόλοιπα νήματα.
- Σχετίζεται με την οδηγία SINGLE.

► Μορφή:

C/C++	copyprivate (<i>list</i>)
-------	-----------------------------

► **Παράδειγμα: χρήση φράσεων Διαμοιρασμού Δεδομένων:**

Στο παράδειγμα γίνεται χρήση διαφόρων φράσεων.

 **C / C++ - data sharing Clauses Example**

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

#define NUM_THREADS 4
#define SLEEP_THREAD 1
#define NUM_LOOPS 2

enum Types {
    ThreadPrivate,
    Private,
    FirstPrivate,
    LastPrivate,
    Shared,
    MAX_TYPES
};

int nSave[NUM_THREADS][MAX_TYPES][NUM_LOOPS] = {{0}};
int nThreadPrivate;

#pragma omp threadprivate(nThreadPrivate)
#pragma warning(disable:4700)

int main() {
    int nPrivate = NUM_THREADS;
    int nFirstPrivate = NUM_THREADS;
    int nLastPrivate = NUM_THREADS;
    int nShared = NUM_THREADS;
    int nRet = 0;
    int i;
    int j;
    int nLoop = 0;

    nThreadPrivate = NUM_THREADS;
    printf("These are the variables before entry "
           "into the parallel region.\n");
    printf("nThreadPrivate = %d\n", nThreadPrivate);
    printf("      nPrivate = %d\n", nPrivate);
    printf(" nFirstPrivate = %d\n", nFirstPrivate);
    printf("  nLastPrivate = %d\n", nLastPrivate);
    printf("      nShared = %d\n\n", nShared);
    omp_set_num_threads(NUM_THREADS);

    #pragma omp parallel copyin(nThreadPrivate) private(nPrivate) shared(nShared) firstprivate(nFirstPrivate)
    {
        #pragma omp for schedule(static) lastprivate(nLastPrivate)
        for (i = 0 ; i < NUM_THREADS ; ++i) {
            for (j = 0 ; j < NUM_LOOPS ; ++j) {
                int nThread = omp_get_thread_num();

                if (nThread == SLEEP_THREAD)
                    system ("sleep 1");
                nSave[nThread][ThreadPrivate][j] = nThreadPrivate;
                nSave[nThread][Private][j] = nPrivate;
                nSave[nThread][Shared][j] = nShared;
                nSave[nThread][FirstPrivate][j] = nFirstPrivate;
                nSave[nThread][LastPrivate][j] = nLastPrivate;
                nThreadPrivate = nThread;
                nPrivate = nThread;
                nShared = nThread;
                nLastPrivate = nThread;
                --nFirstPrivate;
            }
        }

        for (i = 0 ; i < NUM_LOOPS ; ++i) {
            for (j = 0 ; j < NUM_THREADS ; ++j) {
                printf("These are the variables at entry of loop %d of thread %d.\n", i + 1, j);
                printf("nThreadPrivate = %d\n", nSave[j][ThreadPrivate][i]);
                printf("      nPrivate = %d\n", nSave[j][Private][i]);
                printf(" nFirstPrivate = %d\n", nSave[j][FirstPrivate][i]);
                printf("  nLastPrivate = %d\n", nSave[j][LastPrivate][i]);
                printf("      nShared = %d\n\n", nSave[j][Shared][i]);
            }
        }

        printf("These are the variables after exit from the parallel region.\n");
        printf("nThreadPrivate = %d (The last value in the master thread)\n", nThreadPrivate);
        printf("      nPrivate = %d (The value prior to entering parallel region)\n", nPrivate);
        printf(" nFirstPrivate = %d (The value prior to entering parallel region)\n", nFirstPrivate);
        printf("  nLastPrivate = %d (The value from the last iteration of the loop)\n", nLastPrivate);
        printf("      nShared = %d (The value assigned, from the delayed thread, %d)\n\n", nShared, SLEEP_THREAD);
    }
}
```

Φράση REDUCTION

► **Σκοπός:**


Η φράση αυτή εκτελεί μία πράξη αναγωγής σε κάποιες μοιραζόμενες μεταβλητές . Όλες οι μεταβλητές που βρίσκονται σε μία παράλληλη περιοχή και υπάρχουν στη λίστα της φράσης reduction, αντιγράφονται σε τοπικά αντίγραφα, ένα για κάθε νήμα. Με την ολοκλήρωση των επαναλήψεων, εφαρμόζεται η πράξη που ορίζεται στο πεδίο operator και το τελικό αποτέλεσμα αποθηκεύεται στην αρχική θέση τους.

► **Μορφή:**

C/C++	reduction (<i>operator: list</i>)
-------	-------------------------------------

► **Παράδειγμα: REDUCTION - Εσωτερικό Γινόμενο Ανυσμάτων:**

Στην συνέχεια φαίνεται ένα απλό παράδειγμα, όπου κάθε νήμα κάνει κάποιους υπολογισμούς και το μερικό αποτέλεσμα μπαίνει την μοιραζόμενη μεταβλητή result. Επειδή όμως η result είναι στη λίστα της reduction, έχουν δημιουργηθεί τοπικά αντίγραφα και στην ολοκλήρωση των υπολογισμών τα μερικά αποτελέσματα προστίθενται στην result του master thread.

 **C / C++ - reduction Clause Example**

```
#include <omp.h>

main ()  {

int    i, n, chunk;
float a[100], b[100], result;

/* Some initializations */
n = 100;
chunk = 10;
result = 0.0;
for (i=0; i < n; i++)
    {
        a[i] = i * 1.0;
        b[i] = i * 2.0;
    }

#pragma omp parallel for      \
    default(shared) private(i) \
    schedule(static,chunk)    \
    reduction(+:result)

    for (i=0; i < n; i++)
        result = result + (a[i] * b[i]);

printf("Final result= %f\n",result);

}
```

► **Περιορισμοί:**

- Οι REDUCTION μεταβλητές πρέπει να είναι δηλωμένες ως SHARED, πρέπει να είναι βαθμωτές και δέν μπορεί να είναι πίνακες ή δομές.
- Οι πράξεις αναγωγής μπορεί να μη λειτουργούν παρόμοια σε πραγματικούς αριθμούς.
- Οι φράσεις REDUCTION μπορεί να έχουν μία απο τις ακόλουθες μορφές:

C / C++
<div><i>x = x op expr</i> <i>x = expr op x</i> (εκτός από αφαίρεση) <i>x binop = expr</i> <i>x++</i> <i>++x</i> <i>x--</i> <i>--x</i></div>
<div><i>x</i> είναι βαθμωτή μεταβλητή στη λίστα <i>expr</i> είναι βαθμωτή έκφραση που δεν αναφέρεται στο <i>x</i> <i>op</i> δεν είναι υπερφορτωμένος (overloaded), και είναι ένας από τους +, *, -, /, &, ^, , &&, <i>binop</i> δεν είναι υπερφορτωμένος (overloaded), και είναι ένας από τους +, *, -, /, &, ^, </div>

Οδηγίες OpenMP

Σύνοψη Φράσεων / Οδηγιών

- Ο πίνακας που ακολουθεί συνοψίζει ποιές Φράσεις είναι δεκτές σε ποιές Οδηγίες OpenMP.

Φράση	Οδηγία					
	PARALLEL	FOR	SECTIONS	SINGLE	PARALLEL FOR	PARALLEL SECTIONS
IF	<div><div></div></div>				<div><div></div></div>	<div><div></div></div>
PRIVATE	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
SHARED	<div><div></div></div>	<div><div></div></div>			<div><div></div></div>	<div><div></div></div>

DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
COPYPRIVATE				●		
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

- Οι παρακάτω Οδηγίες OpenMP δεν δέχονται Φράσεις:
 - MASTER
 - CRITICAL
 - BARRIER
 - ATOMIC
 - FLUSH
 - ORDERED
 - THREADPRIVATE
- Υπάρχουν διαφοροποιήσεις σε διάφορες υποποιήσεις του OpenMP.

Οδηγίες OpenMP

Κανόνες Σύνδεσης και Ένθεσης

Στη συνέχεια θα περιγράψουμε μερικούς κανόνες πάνω σε οδηγίες που περιέχονται μέσα σε άλλες παράλληλες οδηγίες. Οι κανόνες αυτοί είναι οι εξής:

▶ Σύνδεση Οδηγιών:

- Οι οδηγίες for, sections, single, master και barrier συνδέονται με τη οδηγία parallel που τις περιέχει. Αν δεν υπάρχει τέτοια τότε δε έχουν κανένα αποτέλεσμα.
- Η οδηγία ordered συνδέεται με την οδηγία for που τη περιέχει.
- Η οδηγία atomic επιβάλλει αποκλειστική πρόσβαση σε όλα τα νήματα που τη προσπελαύνουν, όχι μόνο τα τρέχοντα.
- Η οδηγία critical επιβάλλει αποκλειστική πρόσβαση σε όλα τα νήματα που τη προσπελαύνουν, όχι μόνο τα τρέχοντα.
- Μια οδηγία δε μπορεί να συνδεθεί με οδηγία εκτός της παράλληλης περιοχής που τη περιέχει.

• ▶ Ένθεση Οδηγιών:

- Μια οδηγία parallel μέσα σε μια άλλη, λογικά ορίζει μια νέα ομάδα νημάτων η οποία απολείται μόνο από το τρέχον νήμα, εκτός αν είναι ενεργοποιημένη η λειτουργία nested parallelism.
- Οι οδηγίες for, sections και single που βρίσκονται μέσα στην ίδια παράλληλη περιοχή δεν επιτρέπεται να περιέχουν η μία την άλλη τους.
- Οι οδηγίες for, sections και single δεν επιτρέπονται μέσα στις οδηγίες (περιοχές) critical, ordered, master.
- Οδηγίες critical με το ίδιο όνομα δεν επιτρέπεται να βρίσκονται η μία μέσα στην άλλη.
- Οδηγίες barrier δεν επιτρέπονται μέσα στις οδηγίες critical, ordered, master, for, sections, single.
- Οδηγίες master δεν επιτρέπονται μέσα στις οδηγίες for, sections, single.
- Οδηγίες ordered δεν επιτρέπονται μέσα σε οδηγίες critical.
- Κάθε οδηγία η οποία επιτρέπεται όταν εκτελείται δυναμικά μέσα σε μια παράλληλη περιοχή, επιτρέπεται κι όταν εκτελείται έξω από μια παράλληλη περιοχή. Όταν εκτελείται έξω από μια παράλληλη περιοχή, τότε η οδηγία εκτελείται σε μια ομάδα νημάτων αποτελούμενη μόνο από το master thread.

Ρουτίνες Συστήματος Εκτέλεσης

▶ Επισκόπηση:

- Το πρότυπο OpenMP ορίζει ένα API με κλήσεις σε ρουτίνες του συστήματος εκτέλεσης για διάφορες λειτουργίες:
 - Εύρεση / Καθορισμός διαθέσιμων νημάτων / επεξεργαστών
 - Κλειδώματα (semaphores)
 - Μέτρηση χρόνου εκτέλεσης
 - Δυναμική προσαρμογή αριθμού νημάτων κλπ
- Για τις συναρτήσεις Κλειδωμάτων:
 - Η μεταβλητή κλειδώματος πρέπει να έχει το κατάλληλο τύπο omp_lock_t και να προσπελαστεί μόνο μέσα από τις κατάλληλες βιβλιοθήκες.

- Υλοποίηση:

- Ανάλογα με την υλοποίηση, ορισμένες κλήσεις συναρτήσεων μπορεί να μην επιφέρουν αποτέλεσμα, αν δεν υποστηρίζονται από το σύστημα εκτέλεσης.

OMP_SET_NUM_THREADS

► Σκοπός:

Η συνάρτηση αυτή ορίζει το πλήθος των νημάτων που θα δημιουργηθούν σε μία παράλληλη περιοχή (θετικός ακέραιος). Πρέπει να κληθεί σε μέρη του κώδικα που εκτελούνται σειριακά (από το master thread). Η συνάρτηση αυτή υπερέχει της μεταβλητής περιβάλλοντος OMP_NUM_THREADS. Επίσης, αν η δυνατότητα για δυναμική δημιουργία νημάτων είναι ενεργοποιημένη, η συνάρτηση δεν ορίζει τον αριθμό των νημάτων που θα δημιουργηθούν αλλά το μέγιστο πλήθος που μπορούν να δημιουργηθούν.

► Μορφή:

C/C++	<pre>#include <omp.h> void omp_set_num_threads(int num_threads)</pre>
-------	---

OMP_GET_NUM_THREADS

► Σκοπός:

Η συνάρτηση αυτή επιστρέφει το πλήθος των νημάτων που εκτελούνται εκείνη τη στιγμή. Σε μία παράλληλη περιοχή επιστρέφει τον αριθμό των νημάτων που δημιουργήθηκαν για την περιοχή αυτή, ενώ σε μία ακολουθιακή περιοχή επιστρέφει την τιμή 1.

► Μορφή:

C/C++	<pre>#include <omp.h> int omp_get_num_threads(void)</pre>
-------	---

OMP_GET_MAX_THREADS

► Σκοπός:

Η συνάρτηση αυτή επιστρέφει την μέγιστη τιμή που η προηγούμενη συνάρτηση (OMP_GET_NUM_THREADS) μπορεί να επιστρέψει. Πρακτικά επιστρέφει την τιμή της μεταβλητής περιβάλλοντος OMP_NUM_THREADS ή την τιμή του τέθηκε από τη συνάρτηση omp_set_num_threads().

C/C++	<pre>#include <omp.h> int omp_get_max_threads(void)</pre>
-------	---

OMP_GET_THREAD_NUM

► Σκοπός:

Η συνάρτηση αυτή επιστρέφει τον αριθμό του νήματος που την καλεί. Το master thread έχει την τιμή 0. Αν κληθεί σε μια ακολουθιακή περιοχή ή σε μια εμφωλευμένη παράλληλη περιοχή θα επιστρέψει την τιμή 0.

► Μορφή:

C/C++	<pre>#include <omp.h> int omp_get_thread_num(void)</pre>
-------	--

OMP_GET_THREAD_LIMIT

► Σκοπός:

Είναι καινούρια στο OpenMP 3.0. Επιστρέφει τον μέγιστο αριθμό των νημάτων που είναι διαθέσιμα στο πρόγραμμα.

► Μορφή:

C/C++	<pre>#include <omp.h> int omp_get_thread_limit (void)</pre>
-------	---

► Σημειώσεις:

- Δείτε επίσης και τη μεταβλητή περιβάλλοντος OMP_THREAD_LIMIT.

OMP_GET_NUM_PROCS

► Σκοπός:

Με την κλήση της συνάρτησης αυτής μπορούμε να μάθουμε το πλήθος των επεξεργαστών που είναι διαθέσιμοι στο πρόγραμμα.

► Μορφή:

C/C++	<pre>#include <omp.h> int omp_get_num_procs(void)</pre>
-------	---

OMP_IN_PARALLEL

► Σκοπός:

Η συνάρτηση αυτή ορίζει αν τη στιγμή της κλήσης της βρισκόμαστε σε παράλληλη ή σειριακή περιοχή. Επιστρέφει την τιμή 0 αν είμαστε σε σειριακή περιοχή, η μια μη μηδενική τιμή αν είμαστε σε parallel περιοχή.

► Μορφή:

C/C++	<pre>#include <omp.h> int omp_in_parallel(void)</pre>
-------	---

OMP_SET_DYNAMIC

► Σκοπός:

Με τη συνάρτηση αυτή μπορούμε να ενεργοποιήσουμε τη δυνατότητα για δυναμικό ορισμό του πλήθους των νημάτων για κάθε παράλληλη περιοχή. Άν η συνάρτηση επιστρέψει μή μηδενική τιμή τότε σημαίνει οτι η ενεργοποιηση αυτού του μηχανισμού έγινε με επιτυχία, ενώ σε αντίθετη περίπτωση ο μηχανισμός δεν έχει ενεργοποιηθεί. Η συνάρτηση αυτή έχει προτεραιότητα από την OMP_DYNAMIC μεταβλητή περιβάλλοντος. Η κλήση της πρέπει να γίνει απο το σειριακό τμήμα του προγράμματος. Η προκαθορισμένη κατάσταση εξαρτάται απο την υλοποίηση του OpenMP.

► Μορφή:

C/C++	<pre>#include <omp.h> void omp_set_dynamic(int dynamic_threads)</pre>
-------	---

OMP_GET_DYNAMIC

► Μορφή:

Με τη συνάρτηση αυτή μπορούμε να δούμε αν η δυνατότητα για δυναμικό ορισμό του αριθμού των νημάτων είναι ενεργοποιημένη.

► Σκοπός:

C/C++	<pre>#include <omp.h> int omp_get_dynamic(void)</pre>
-------	---

OMP_SET_NESTED

► Σκοπός:

Με τη συνάρτηση αυτή μπορούμε να ενεργοποιήσουμε και να απενεργοποιήσουμε τη δυνατότητα για ένθετο παραλληλισμό. Εξ ορισμού αυτή η δυνατότητα είναι απενεργοποιημένη. Η τιμή nested = 0 σημαίνει απενεργοποίηση, ενώ μια μη-μηδενική τιμή σημαίνει ενεργοποίηση. Η κλήση της συνάρτησης υπερισχύει της τιμής της μεταβλητής περιβάλλοντος OMP_NESTED.

► Μορφή:

C/C++	<pre>#include <omp.h> void omp_set_nested(int nested)</pre>
-------	---

OMP_GET_NESTED

► Σκοπός:

- Εξετάζει αν επιτρέπεται ένθετος παραλληλισμός ή όχι. Η επιστροφή μη-μηδενικής τιμής σημαίνει οτι επιτρέπεται.

► Μορφή:

--

C/C++	<pre>#include <omp.h> int omp_get_nested (void)</pre>
-------	---

OMP_INIT_LOCK

► Σκοπός:

- Αρχικοποίηση μεταβλητής κλειδώματος. Αρχικά η μεταβλητή είναι ξεκλειδωτη.

► Μορφή:

C/C++	<pre>#include <omp.h> void omp_init_lock(omp_lock_t *lock) void omp_init_nest_lock(omp_nest_lock_t *lock)</pre>
-------	---

OMP_DESTROY_LOCK

► Σκοπός:

- Διαγραφή μεταβλητής κλειδώματος. Η μεταβλητή πρέπει να έχει αρχικοποιηθεί.

► Format:

C/C++	<pre>#include <omp.h> void omp_destroy_lock(omp_lock_t *lock) void omp_destroy_nest__lock(omp_nest_lock_t *lock)</pre>
-------	--

OMP_SET_LOCK

► Σκοπός

- Αναγκάζει το νήμα που την εκτελεί να περιμένει μέχρι το κλείδωμα να ελευθερωθεί. Τότε το νήμα καταλαμβάνει το κλείδωμα και συνεχίζει την εκτέλεσή του. Το κλείδωμα πρέπει να έχει αρχικοποιηθεί.

► Format:

C/C++	<pre>#include <omp.h> void omp_set_lock(omp_lock_t *lock) void omp_set_nest__lock(omp_nest_lock_t *lock)</pre>
-------	--

OMP_UNSET_LOCK

► Σκοπός:

- Αυτή η ρουτίνα απελευθερώνει ένα κλείδωμα. Το κλείδωμα πρέπει να έχει αρχικοποιηθεί.

► Μορφή:

C/C++	<pre>#include <omp.h> void omp_unset_lock(omp_lock_t *lock) void omp_unset_nest__lock(omp_nest_lock_t *lock)</pre>
-------	--

OMP_TEST_LOCK

► Σκοπός:

- Το νήμα που εκτελεί αυτή τη ρουτίνα ελέγχει τη κατάσταση ενός κλειδώματος, χωρίς όμως να μπει σε κατάσταση αναμονής αν το κλείδωμα δεν είναι διαθέσιμο. Επιστρέφει 0 αν το κλείδωμα είναι ελεύθερο, αλλιώς επιστρέφει μη-μηδενική τιμή. Το κλείδωμα πρέπει να έχει αρχικοποιηθεί.

► Μορφή:

C/C++	<pre>#include <omp.h> int omp_test_lock(omp_lock_t *lock) int omp_test_nest__lock(omp_nest_lock_t *lock)</pre>
-------	--

OMP_GET_WTIME

► Σκοπός

- Ρουτίνα για τη μέτρηση χρόνου εκτέλεσης (wall clock όχι CPU)
- Επιστρέφει μια τιμή διπλής ακρίβειας. Συνήθως χρησιμοποιείται σε ζεύγη ώστε ο χρόνος που πέρασε για την εκτέλεση ενός

- τιμήματος κώδικα υπολογίζεται ως διαφορά δύο μετρήσεων.
- Η μέτρηση είναι συνεπής για το ίδιο νήμα, επομένως δεν έχει νόημα η αφαίρεση δύο τιμών που έχουν μετρηθεί σε διαφορετικά νήματα.

► **Μορφή:**

C/C++	#include <omp.h> double omp_get_wtime(void)
-------	--

OMP_GET_WTICK

► **Σκοπός:**

- Επιστρέφει μια τιμή διπλής ακρίβειας, που είναι ο χρόνος μεταξύ δύο διαδοχικών κτύπων ρολογιού.

► **Μορφή:**

C/C++	#include <omp.h> double omp_get_wtick(void)
-------	--

Μεταβλητές Περιβάλλοντος

- Το OpenMP παρέχει μια σειρά μεταβλητών περιβάλλοντος που βοηθούν στην εκτέλεση του παράλληλου κώδικα.
- Τα ονόματα των μεταβλητών γράφονται πάντα σε κεφαλαία.

OMP_SCHEDULE

Αφορά μόνο στις οδηγίες `for`, `parallel for` που έχουν ορίσει τη φράση `schedule` ίση με `RUNTIME`. Η τιμή αυτής της μεταβλητής καθορίζει το τρόπο εκτέλεσης των επαναλήψεων. Για παράδειγμα:

```
setenv OMP_SCHEDULE "guided, 4"  
setenv OMP_SCHEDULE "dynamic"
```

OMP_NUM_THREADS

Ορίζει τον μέγιστο επιτρεπόμενο αριθμό νημάτων. Για παράδειγμα:

```
setenv OMP_NUM_THREADS 8
```

OMP_DYNAMIC

Επιτρέπει ή απαγορεύει τη δυναμική προσαρμογή του διαθέσιμου αριθμού νημάτων στις παράλληλες περιοχές. Λαμβάνει τις τιμές `TRUE` και `FALSE`. Για παράδειγμα:

```
setenv OMP_DYNAMIC TRUE
```

OMP_NESTED

Επιτρέπει ή απαγορεύει ένθετο παραλληλισμό, αν το επιτρέπει η υλοποίηση. Λαμβάνει τις τιμές `TRUE` και `FALSE`. Για παράδειγμα:

```
setenv OMP_NESTED TRUE
```

OMP_STACKSIZE

Νέο στο OpenMP 3.0. Ελέγχει το μέγεθος στοίβας των (non-master) νημάτων. Παραδείγματα:

```
setenv OMP_STACKSIZE 2000500B  
setenv OMP_STACKSIZE "3000 k "  
setenv OMP_STACKSIZE 10M  
setenv OMP_STACKSIZE " 10 M "  
setenv OMP_STACKSIZE "20 m "  
setenv OMP_STACKSIZE " 1G"  
setenv OMP_STACKSIZE 20000
```

OMP_WAIT_POLICY

Νέο στο OpenMP 3.0. Ορίζει αν η αναμονή των νημάτων στην οδηγία `wait` θα είναι ενεργός (busy waiting) ή όχι. Λαμβάνει τιμές `ACTIVE` και `PASSIVE`. Εξαρτάται από την υλοποίηση. Παραδείγματα:

```
setenv OMP_WAIT_POLICY ACTIVE  
setenv OMP_WAIT_POLICY active  
setenv OMP_WAIT_POLICY PASSIVE  
setenv OMP_WAIT_POLICY passive
```

OMP_THREAD_LIMIT

Νέο στο OpenMP 3.0. Ορίζει το μέγιστο αριθμό νημάτων που μπορεί να χρησιμοποιήσει ένα πρόγραμμα OpenMP. Άρα `OMP_THREAD_LIMIT >= OMP_NUM_THREADS`. Παράδειγμα:

```
setenv OMP_THREAD_LIMIT 8
```

► Μέγεθος Στοίβας:

- Το πρότυπο OpenMP δεν ορίζει το μέγεθος στοίβας, επομένως κάθε υλοποίηση μπορεί να επιτρέπει διαφορετικό μέγεθος. Επίσης ο φλοιός του χρήστη μπορεί να επιβάλει επιπλέον περιορισμούς.
- Συνήθως το προκαθορισμένο μέγεθος είναι 4 με 8 MB και εύκολα εξαντλείται (αντιστοιχεί περίπου με πίνακα 700X700 αριθμών double).
- Νήματα που ξεπερνούν το διαθέσιμο μέγεθος μπορεί να προκαλέσουν segmentation fault ή να συνεχίσουν καταστρέφοντας δεδομένα στη μνήμη.
- Παραδείγματα ελέγχου και τροποποίησης του μεExamples for increasing the thread stack size to 12 MB at LC:

Ενέργεια	Εντολές Bash
Έλεγχος	<code>ulimit -s</code>
Τροποποίηση	<code>ulimit -s 12288</code> <code>export KMP_STACKSIZE=12000000</code>

► Μεταγλώττιση:

- Στη μεταγλώττιση από τη γραμμή εντολών πρέπει να εισαθεί το αντίστοιχο flag:

Μεταγλωττιστής	Flag
Intel	<code>-openmp</code>
GNU	<code>-fopenmp</code>

► Τεκμηρίωση:

- Intel: www.intel.com/software/products/compilers/
- GNU: gnu.org

Τέλος.

- Το αρχικό κείμενο είναι του [Blaise Barney](#), lawrence Livermore Laboratory, USA. Η μετάφραση, καθώς και ορισμένες προσαρμογές και τροποποιήσεις οφείλονται στο [Κώστα Μαργαρίτη](#), Πανεπιστήμιο Μακεδονίας.
- Στο επίσημο site του OpenMP web site, μπορείτε να βρείτε τις τελευταίες εκδόσεις και πλήρεις αναφορές στα C/C++ και Fortran API's. www.openmp.org
- Γενικότερο υλικό μπορείτε να βρείτε στο pdplab.it.uom.gr