Οδηγός OpenMp

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr

Μετάφραση του κειμένου (8/10/2014)

https://computing.llnl.gov/tutorials/parallel_comp/

Συγγραφέας: Blaise Barney, Lawrence Livermore National Laboratory

Περιεχόμενα:

Περίληψη

Το OpenMP είναι μια προγραμματιστική διεπαφή εφαρμογής , που καθορίζεται από κοινού από μια ομάδα σημαντικών προμηθευτών υλικού και λογισμικού. Το OpenMP παρέχει ένα φορητό, εξελικτικό πρότυπο για τους υπεύθυνους ανάπτυξης των παράλληλων εφαρμογών κοινόχρηστης μνήμης. Η προγραμματιστική διεπαφή εφαρμογής υποστηρίζει C/C++ και Fortran σε μια μεγάλη ποικιλία αρχιτεκτονικών. Αυτός ο οδηγός εκμάθησης καλύπτει τα περισσότερα κύρια χαρακτηριστικά του OpenMP, συμπεριλαμβανομένων των διαφόρων δομών και οδηγιών για τον καθορισμό των παράλληλων περιοχών, την κατανομή της εργασίας, τον συγχρονισμό και τα δεδομένα περιβάλλοντος. Οι βιβλιοθήκες εκτέλεσης και οι μεταβλητές περιβάλλοντος καλύπτονται επίσης. Αυτός ο οδηγός εκμάθησης περιλαμβάνει παραδείγματα κώδικα σε C και Fortran και μια εργαστηριακή άσκηση.

Επίπεδο/Προαπαιτούμενα: Ο οδηγός εκμάθησης αυτός είναι ένας από τους 8 στο εργαστήριο "Χρησιμοποιώντας τους Υπερυπολογιστές του LLNL" της 4ης μέρας. Είναι ιδανικός για αυτούς που είναι νέοι στον παράλληλο προγραμματισμό με OpenMP. Μια βασική κατανόηση του παράλληλου προγραμματισμού με γλώσσα C ή Fortran είναι απαιτούμενη. Για αυτούς που δεν είναι εξοικειωμένοι με τον παράλληλο προγραμματισμό γενικά, το υλικό που καλύπτεται στο EC3500: Introduction To Parallel Computing θα τους φανεί πολύ χρήσιμο.

Εισαγωγή

Τι είναι το OpenMP;

- **Το OpenMP είναι:**
  - Ένα Πρόγραμμα Διεπαφής Εφαρμογής(API) που μπορεί να χρησιμοποιηθεί για να κατευθύνει τον πολυνηματικό παραλληλισμό κοινόχρηστης μνήμης.
  - Αποτελείται από τρία βασικά στοιχεία:
    - Οδηγίες προς τον Μεταγλωττιστή
    - Ρουτίνες Βιβλιοθήκης Εκτέλεσης
    - Μεταβλητές Περιβάλλοντος
  - Μια συντομογραφία του:
    - Σύντομη εκδοχή:Open Multi-Processing(Ανοιχτός Πολυπρογραμματισμός)
    - Μακρά εκδοχή: Ανοιχτές προδιαγραφές για Πολυπρογραμματισμό μέσω συνεργατικής εργασίας μεταξύ ομάδων ενδιαφέροντος από την βιομηχανία Λογισμικού και Υλικού, την κυβέρνηση και την Ακαδημαϊκή κοινότητα

- **Το OpenMP δεν είναι:**
  - Προορισμένο για παράλληλα συστήματα κατανεμημένης μνήμης(από το ίδιο).
  - Απαραίτητα πανομοιότυπα υλοποιημένο για όλους τους προμηθευτές.
  - Εξασφαλισμένο ότι θα κάνει την πιο αποδοτική χρήση της κοινόχρηστης μνήμης.
  - Απαραίτητο για τον έλεγχο των εξαρτήσεων, συγκρούσεις δεδομένων, συνθήκες ανταγωνισμού ή αδιέξοδα.
  - Απαραίτητο για τον έλεγχο των ακολουθιών του κώδικα που μπορεί να κάνουν ένα πρόγραμμα να χαρακτηριστεί μη-σύμφωνο.
  - Προορισμένο να καλύψει τον αυτόματο παραλληλισμό δημιουργημένο από τον μεταγλωττιστή και τις οδηγίες προς τον μεταγλωττιστή για να βοηθήσει αυτήν την παραλληλοποίηση.
  - Σχεδιασμένο για να εγγυηθεί ότι η είσοδος ή η έξοδος στο ίδιο αρχείο είναι συγχρονισμένη όταν εκτελεσθεί παράλληλα. Ο προγραμματιστής είναι υπεύθυνος για τον συγχρονισμό της εισόδου και της εξόδου.
- **Στόχοι του OpenMP:**
  - Τυποποίηση:
    - Παρέχει ένα πρότυπο ανάμεσα σε μια ποικιλία από αρχιτεκτονικές/πλατφόρμες κοινόχρηστης μνήμης.
    - Ορίστηκε από κοινού και εγκρίθηκε από μία ομάδα σημαντικών προμηθευτών Λογισμικού και Υλικού.
  - Απλό και Αποδοτικό:

    - Καθιέρωση ενός απλού και περιορισμένου σετ οδηγιών για προγραμματισμό μηχανημάτων κοινής μνήμης.
    - Αξιοσημείωτη παραλληλοποίηση μπορεί να εφαρμοστεί χρησιμοποιώντας μόνο 3 ή 4 οδηγίες.
    - Προφανώς ο σκοπός γίνεται λιγότερο ουσιώδης με κάθε νέα έκδοση.
- Ευκολία στην χρήση:
    - Παρέχει την ικανότητα για τμηματική παραλληλοποίηση ενός σειριακού προγράμματος αντίθετα με τις βιβλιοθήκες που περνάνε μηνύματα, οι οποίες συνήθως απαιτούν μια όλα ή τίποτα αντιμετώπιση.
    - Παρέχει την δυνατότητα εφαρμόσει και "χονδροειδές" αλλά και "λεπτό" παραλληλισμό.
- Φορητότητα:
    - Η προγραμματιστική διεπαφή της εφαρμογής είναι καθορισμένη για C/C++ και Fortran.
    - Δημόσιο φόρουμ για την προγραμματιστική διεπαφή της εφαρμογής και τα μέλη.
    - Οι πιο δημοφιλείς πλατφόρμες έχουν υλοποίηση, συμπεριλαμβανομένων των πλατφορμών Unix/Linux και Windows.

➢ Ιστορία

- Στις αρχές του '90, οι προμηθευτές των μηχανημάτων κοινόχρηστης μνήμης εφοδίασαν παρόμοιες, με βάση τις οδηγίες, επεκτάσεις προγραμμάτων στην Fortran:
    - Ο χρήστης μπορούσε να βελτιώσει ένα σειριακό πρόγραμμα σε Fortran με οδηγίες, ορίζοντας ποιοι βρόγχοι θα παραλληλοποιηθούν.
    - Ο μεταγλωττιστής θα ήταν υπεύθυνος για την αυτόματη παραλληλοποίηση αυτών των βρόγχων στους επεξεργαστές συμμετρικής πολυεπεξεργασίας(SMP) .
- Οι υλοποιήσεις ήταν όλες παρόμοιες λειτουργικά, αλλά είχαν αποκλίσεις(όπως πάντα).
- Η πρώτη προσπάθεια προσδιορισμού προτύπου ήταν το σχέδιο για το ANSI X3H5 το 1994. Δεν υιοθετήθηκε ποτέ, σε μεγάλο βαθμό λόγω της μείωσης ενδιαφέροντος καθώς οι μηχανές με κατανεμημένης μνήμης έγιναν δημοφιλείς.
- Παρόλα αυτά, μετά από σύντομο διάστημα, καινούργιες αρχιτεκτονικές μηχανημάτων κοινόχρηστης μνήμης άρχισαν να γίνονται διαδεδομένες και το ενδιαφέρον συνεχίστηκε.
- Οι προδιαγραφές του προτύπου OpenMP άρχισαν την άνοιξη του 1997, συνεχίζοντας από εκεί που είχε μείνει το ANSI X3H5.
- Οδηγείται από το Συμβούλιο Αξιολόγησης Αρχιτεκτονικής του OpenMP(ARB). Τα αρχικά μέλη του ARB και οι συνεισφέροντες

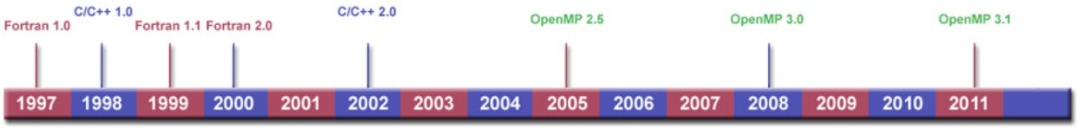
φαίνονται παρακάτω(Όλα τα ονόματα προέρχονται από το Site του OpenMP) :

Μέλη ARB	Υποστηρικτές Προγραμματιστών Εφαρμογών	Υποστηρικτές Προμηθευτών Λογισμικού
○ Compaq / Digital ○ Hewlett-Packard Company ○ Intel Corporation ○ International Business Machines (IBM) ○ Kuck & Associates, Inc. (KAI) ○ Silicon Graphics, Inc. ○ Sun Microsystems, Inc. ○ U.S. Department of Energy ASCI program	○ ADINA R&D, Inc. ○ ANSYS, Inc. ○ Dash Associates ○ Fluent, Inc. ○ ILOG CPLEX Division ○ Livermore Software Technology Corporation ○ (LSTC) ○ MECALOG SARL Oxford Molecular Group PLC ○ The Numerical Algorithms Group Ltd. (NAG)	○ Absoft Corporation ○ Edinburgh Portable Compilers ○ GENIAS Software GmBH ○ Myrias Computer Technologies, ○ Inc. ○ The Portland Group, Inc. (PGI)

- Για πιο πολλά νέα και πληροφορίες μελών για το OpenMP ARB επισκεφτείτε την διεύθυνση: openmp.org/wp/about-openmp.

➢ Ιστορία Εκδόσεων

- Το OpenMP συνεχίζει να εξελίσσεται, με νέες δομές και χαρακτηριστικά να προστίθενται με τον καιρό.
- Αρχικά οι προδιαγραφές της προγραμματιστικής διεπαφής εφαρμογής είχαν κυκλοφορήσει ξεχωριστά για C και Fortran. Από το 2005 κυκλοφορούν μαζί.
- Η χρονική γραμμή παρακάτω προσδιορίζει χρονικά τις κυκλοφορίες των εκδόσεων.



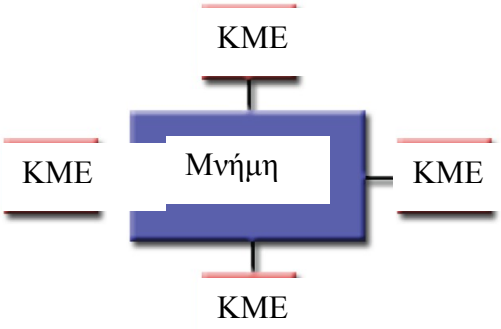
➢ Αναφορές

- Η σελίδα του OpenMP: openmp.org.
  Οι προδιαγραφές του API,συχνές ερωτήσεις, παρουσιάσεις, συζητήσεις, κυκλοφορίες media, ημερολόγιο, αίτηση μέλους και πολλά άλλα…
- Wikipedia: en.wikipedia.org/wiki/OpenMP

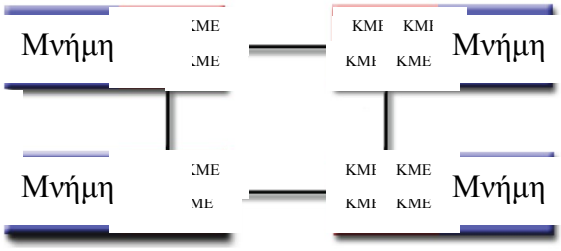
Προγραμματιστικό Μοντέλο του OpenMP

➢ Μοντέλο κοινής μνήμης

- Το OpenMP σχεδιάστηκε για πολυπύρηνα και πολυεπεξεργαστικά μηχανήματα κοινόχρηστης μνήμης. Η υποκείμενη αρχιτεκτονική μπορεί να μοιράζεται μνήμη UMA(Ομοιόμορφη Πρόσβαση στην Μνήμη) ή NUMA(Μη- Ομοιόμορφη Πρόσβαση στην Μνήμη).

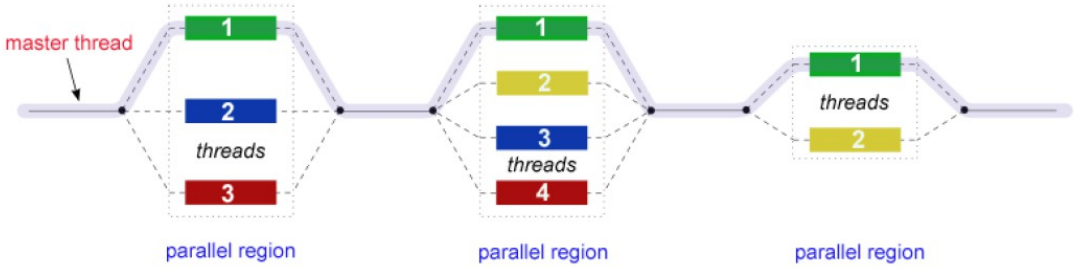


Ομοιόμορφη Πρόσβαση στη Μνήμη



Μη-Ομοιόμορφη Πρόσβαση στη Μνήμη

- **Παραλληλισμός Βασισμένος σε Νήματα**
  - Τα προγράμματα του OpenMP πετυχαίνουν την παραλληλοποίηση αποκλειστικά μέσω της χρήσης νημάτων.
  - Το νήμα της εκτέλεσης είναι η μικρότερη μονάδα της διεργασίας που μπορεί να προγραμματιστεί από ένα Λειτουργικό Σύστημα. Η ιδέα μιας υπορουτίνας που μπορεί να προγραμματιστεί να τρέχει αυτόνομα μπορεί να βοηθήσει να εξηγηθεί τι είναι ένα νήμα.
  - Τα νήματα υπάρχουν μέσα στους πόρους μιας διεργασίας. Χωρίς την διεργασία παύουν να υπάρχουν.
  - Τυπικά, ο αριθμός των νημάτων ταιριάζει με τον αριθμό των επεξεργαστών/πυρήνων του μηχανήματος. Παρόλα αυτά η πραγματική χρήση των νημάτων αφήνεται στην εφαρμογή.
- **Σαφής Παραλληλοποίηση**
  - Το OpenMP είναι ένα ρητό(μη αυτόματο) προγραμματιστικό μοντέλο, που προσφέρει πλήρη έλεγχο της παραλληλοποίησης.
  - Η παραλληλοποίηση μπορεί να είναι τόσο απλή όσο το να πάρεις ένα σειριακό πρόγραμμα και να εισάγεις οδηγίες προς τον μεταγλωττιστή.
  - Ή τόσο πολύπλοκο όσο να εισάγεις υπορουτίνες σε ένα σετ πολλαπλών βαθμών παραλληλοποίησης, κλειδωμάτων ακόμα και φωλιασμένων.
- **Το μοντέλο Fork-Join**
  - Το OpenMP χρησιμοποιεί το μοντέλο διακλάδωσης-σύνδεσης(fork-join) για παράλληλη εκτέλεση:



  - Όλα τα προγράμματα του OpenMP ξεκινούν σαν μία μόνο διεργασία: το κύριο νήμα. Το κύριο νήμα εκτελεί ακολουθίες σειριακά μέχρι την πρώτη δομή παράλληλου τμήματος που θα συναντήσει.
  - Διακλάδωση(Fork): το αρχικό νήμα τότε δημιουργεί μια ομάδα από παράλληλα *νήματα(threads)*.
  - Οι δηλώσεις στο πρόγραμμα τότε περικλείονται από την δομή του παράλληλου τμήματος και εκτελούνται παράλληλα μεταξύ των διάφορων ομάδων νημάτων.
  - Συνένωση(Join): όταν η ομάδα νημάτων τελειώσει με τις δηλώσεις στην δομή του παράλληλου τμήματος, συγχρονίζονται και τερματίζονται αφήνοντας μόνο το κύριο νήμα.
  - Ο αριθμός των παράλληλων τμημάτων και των νημάτων που περιλαμβάνουν είναι αυθαίρετος.

- **Οδηγίες βασισμένες στον Μεταγλωττιστή:**
  - Το μεγαλύτερο μέρος της παραλληλοποίησης του OpenMP γίνεται μέσω της χρήσης των οδηγιών του Μεταγλωττιστή που είναι ενταγμένες στον αρχικό κώδικα της C/C++ ή της Fortran.
- **Εμφωλευμένη Παραλληλοποίηση:**
  - Η προγραμματιστική διεπαφή εφαρμογής προβλέπει την τοποθέτηση παράλληλων τμημάτων μέσα σε άλλα παράλληλα τμήματα.
  - Οι υλοποιήσεις μπορεί να υποστηρίζουν ή όχι αυτό το χαρακτηριστικό.
- **Δυναμικά Νήματα:**
  - Η προγραμματιστική διεπαφή εφαρμογής προβλέπει το περιβάλλον εκτέλεσης ώστε να αλλάζει δυναμικά ο αριθμός των νημάτων που χρησιμοποιούνται στην εκτέλεση των παράλληλων τμημάτων. Σκοπός της είναι να προωθήσει έναν πιο αποτελεσματικό τρόπο χρήσης των πόρων, αν αυτό είναι δυνατό.
  - Οι υλοποιήσεις μπορεί να υποστηρίζουν ή όχι αυτό το χαρακτηριστικό.
- **Ε/Ε**
  - Το OpenMP δεν ορίζει τίποτα για παράλληλη Ε/Ε. Αυτό είναι ιδιαίτερα σημαντικό αν πολλαπλά νήματα προσπαθούν να γράψουν/διαβάσουν το ίδιο αρχείο.
  - Αν κάθε νήμα οδηγεί την Ε/Ε του σε διαφορετικό αρχείο, τα προβλήματα δεν είναι σημαντικά.
  - Είναι εντελώς θέμα το προγραμματιστή να εξασφαλίσει ότι η Ε/Ε διεξάγεται κανονικά μέσα στο περιεχόμενο που πολυνηματικού προγράμματος.
- **Μοντέλο μνήμης: Κάνει συχνά FLUSH;**
  - Το OpenMP παρέχει μια "χαλαρή συνοχή" και "προσωρινή" εμφάνιση της μνήμης του νήματος. Με άλλα λόγια, τα νήματα μπορούν να κάνουν "κρύψουν"(cache) τα δικά τους δεδομένα και δεν χρειάζεται να διατηρούν διαρκή συνοχή με την πραγματική μνήμη όλη την ώρα.
  - Όταν είναι κρίσιμο όλες οι μεταβλητές να βλέπουν μια κοινόχρηστη μεταβλητή ίδια, ο προγραμματιστής είναι υπεύθυνος να εξασφαλίσει ότι αυτή η μεταβλητή θα υποστεί κένωση(FLUSH) από όλα τα νήματα που χρειάζεται.
  - Περισσότερα για αυτό αργότερα…

Επισκόπηση της Διεπαφής Προγραμματισμού Εφαρμογής του OpenMP

- **Τρία στοιχεία:**

- Η προγραμματιστική διεπαφή εφαρμογής του OpenMP αποτελείται από τρία διακριτά στοιχεία. Από την έκδοση 3.1 και μετά έχουμε:
    - Οδηγίες Μεταγλωττιστή(20)
    - Ρουτίνες Βιβλιοθήκης Εκτέλεσης(32)
    - Μεταβλητές Περιβάλλοντος(9)
- Ο προγραμματιστής της εφαρμογής αποφασίζει πώς να απασχολήσει αυτά τα στοιχεία. Στην απλούστερη περίπτωση μόνο λίγα από αυτά χρειάζονται.
- Οι εφαρμογές διαφέρουν στην υποστήριξη όλων των στοιχείων της προγραμματιστικής διεπαφής εφαρμογής. Για παράδειγμα, μια εφαρμογή μπορεί να υποστηρίζει ότι υποστηρίζει εμφωλευμένη παραλληλοποίηση αλλά η προγραμματιστική διεπαφή να ξεκαθαρίζει ότι θα είναι περιορισμένη σε ένα μόνο νήμα, το κύριο νήμα. Όχι ακριβώς τι περιμένει ο προγραμματιστής;

➢ Οδηγίες Μεταγλωττιστή:

- Οι οδηγίες του μεταγλωττιστή εμφανίζονται ως σχόλια στον πηγαίο κώδικα και αγνοούνται από τους μεταγλωττιστές εκτός αν τους πούμε το αντίθετο - συνήθως ορίζοντας την κατάλληλη σημαία του μεταγλωττιστή όπως θα συζητηθεί στο κεφάλαιο "Μεταγλώττιση" αργότερα.
- Οι οδηγίες του μεταγλωττιστή του OpenMP χρησιμοποιούνται για διάφορους σκοπούς:
    - Δημιουργούν ένα παράλληλο τμήμα.
    - Χωρίζουν τα μπλοκ του κώδικα ανάμεσα στα νήματα.
    - Διανέμουν τις επαναλήψεις των βρόγχων μεταξύ των νημάτων.
    - Βάζουν σε σειρά τα τμήματα του κώδικα.
    - Συγχρονίζουν την εργασία μεταξύ των νημάτων.
- Οι οδηγίες του μεταγλωττιστή έχουν την παρακάτω σύνταξη: *sentinel directive-name [clause, ...]*

  Για παράδειγμα:

Fortran	`!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(BETA,PI)`
C/C++	`#pragma omp parallel default(shared) private(beta,pi)`

- Οι οδηγίες του μεταγλωττιστή καλύπτονται λεπτομερώς αργότερα.

➢ Ρουτίνες Βιβλιοθηκών Εκτέλεσης:

- Η προγραμματιστική διεπαφή εφαρμογής του OpenMP περιλαμβάνει έναν αυξανόμενο αριθμό από ρουτίνες βιβλιοθηκών εκτέλεσης.
- Αυτές οι ρουτίνες χρησιμοποιούνται για διάφορους σκοπούς:
    - Καθορισμός και ερώτηση του αριθμού των νημάτων.

- Ερώτηση του μοναδικού αναγνωριστικού του νήματος(ID), του αναγνωριστικού του πατρικού νήματος και του μεγέθους της ομάδας των νημάτων.
- Καθορισμός και ερώτηση του χαρακτηριστικού των δυναμικών νημάτων.
- Ερώτηση αν είναι μέσα σε παράλληλο τμήμα και σε ποιο βαθμό.
- Καθορισμός και ερώτηση του εμφωλευμένου παραλληλισμού.
- Καθορισμός, αρχικοποίηση και τερματισμός των κλειδωμάτων και των εμφωλευμένων κλειδωμάτων.
- Ερώτηση του χρόνου και της επίλυσης.

- Για την C/C++ όλες οι ρουτίνες βιβλιοθηκών εκτέλεσης είναι στην πραγματικότητα υπορουτίνες. Για την Fortran κάποιες είναι πραγματικές συναρτήσεις και κάποιες υπορουτίνες. Για παράδειγμα:

Fortran	`INTEGER FUNCTION OMP_GET_NUM_THREADS()`
C/C++	`#include <omp.h>` `int omp_get_num_threads(void)`

- Σημειώστε ότι για την C/C++ συνήθως χρειάζεται να συμπεριλαμβάνετε το αρχείο κεφαλίδας `<omp.h>`.
- Οι ρουτίνες στην Fortran δεν είναι ευαίσθητες σε κεφαλαίους χαρακτήρες ενώ στην C/C++ είναι.
- Οι ρουτίνες βιβλιοθηκών εκτέλεσης αναλύονται σύντομα σαν μια επισκόπηση στο κεφάλαιο "Ρουτίνες Βιβλιοθηκών Εκτέλεσης" και πιο πολύ στο Παράρτημα Α.

➢ Μεταβλητές Περιβάλλοντος

- Το OpenMP παρέχει πολλές μεταβλητές περιβάλλοντος για τον έλεγχο της εκτέλεσης του παράλληλου κώδικα κατά την διάρκεια εκτέλεσης.
- Αυτές οι μεταβλητές περιβάλλοντος μπορούν να χρησιμοποιηθούν για να ελέγχουν πράγματα όπως:
  - Καθορισμός του αριθμού των νημάτων.
  - Καθορισμός του πως θα χωριστούν οι επαναλήψεις των βρόγχων.
  - Δέσμευση νημάτων σε επεξεργαστές.
  - Ενεργοποίηση/Απενεργοποίηση της εμφωλευμένης παραλληλοποίησης, καθορισμός του μέγιστου βαθμού εμφωλευμένης παραλληλοποίησης.
  - Ενεργοποίηση/Απενεργοποίηση των δυναμικών νημάτων.
  - Καθορισμός του μεγέθους της στοίβας του νήματος.
  - καθορισμός της πολιτικής αναμονής του νήματος.

- Ο καθορισμός των μεταβλητών περιβάλλοντος του OpenMP γίνεται όπως με τον καθορισμό οποιασδήποτε άλλης μεταβλητής περιβάλλοντος και εξαρτάται σε ποιο shell είστε. Για παράδειγμα:

csh/tcsh	`setenv OMP_NUM_THREADS 8`
sh/bash	`export OMP_NUM_THREADS=8`

- Οι μεταβλητές περιβάλλοντος του OpenMP αναλύονται στο κεφάλαιο Μεταβλητές Περιβάλλοντος παρακάτω.

➢ Παράδειγμα Δομής Κώδικα του OpenMP

Fortran - Γενική Δομή του Κώδικα

```
PROGRAM HELLO
INTEGER VAR1, VAR2, VAR3

Serial code
.
.
.
Beginning of parallel section. Fork a team of threads.
Specify variable scoping
!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
Parallel section executed by all threads
.
Other OpenMP directives
.
Run-time Library calls
.
All threads join master thread and disband
!$OMP END PARALLEL
Resume serial code
.
.
.
END
```

C / C++ - Γενική Δομή του Κώδικα

```
#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
  .
```

```
   Other OpenMP directives
   .
   Run-time Library calls
   .
   All threads join master thread and disband
}
Resume serial code
.
.
.
}
```

Μεταγλώττιση Προγραμμάτων OpenMP

- **Υλοποιήσεις OpenMP της LC:**
  - Υποστήριξή OpenMP 2.0 από όλους τους μεταγλωττιστές Linux των συστοιχιών της LC.
  - Από τον Ιούνιου του 2012 φαίνεται πως οι πιο πρόσφατες εκδόσεις μεταγλωττιστών των Intel, PGI και GNU δεν υποστηρίζουν πλήρως το OpenMP 3.0 ή 3.1. Μερικά χαρακτηριστικά φαίνεται να υποστηρίζονται παρόλα αυτά, ανάλογα τον μεταγλωττιστή και την έκδοση.
  - Χρησιμοποιείστε την εντολή `use -l compilers` για να δείτε τα πακέτα των μεταγλωττιστών κατά έκδοση.
  - Μπορείτε να δείτε και πληροφορίες για τις εκδόσεις των compilers στην σελίδα: https://computing.llnl.gov/code/compilers.html
- **Μεταγλώττιση:**
  - Όλοι οι μεταγλωττιστές της LC απαιτούν την χρήση της κατάλληλης σημαίας για να "ενεργοποιηθούν" οι μεταγλωττίσεις του OpenMP. Ο πίνακας παρακάτω δείχνει τι να χρησιμοποιήσετε για τον κάθε μεταγλωττιστή.

Μεταγλωττιστής / Πλατφόρμα	Μεταγλωττιστής	Σημαία
Intel Linux Opteron/Xeon	`icc` `icpc` `ifort`	`-openmp`
PGI Linux Opteron/Xeon	`pgcc` `pgCC` `pgf77` `pgf90`	`-mp`
GNU Linux Opteron/Xeon IBM Blue Gene	`gcc` `g++` `g77` `gfortran`	`-fopenmp`
IBM Blue Gene	`bgxlc_r, bgcc_r` `bgxlC_r, bgxlc++_r` `bgxlc89_r` `bgxlc99_r` `bgxlf_r` `bgxlf90_r` `bgxlf95_r`	`-qsmp=om`

	`bgxlf2003_r` *Σιγουρευτείτε ότι χρησιμοποιείται έναν compiler ασφαλή με τα νήματα - το όνομα του τελειώνει με _r	

- Τεκμηρίωση Μεταγλωττιστή:
  - IBM BlueGene: www-01.ibm.com/software/awdtools/fortran/ and www-01.ibm.com/software/awdtools/xlcpp
  - Intel: www.intel.com/software/products/compilers/
  - PGI: www.pgroup.com
  - GNU: gnu.org
  - Όλα: Ανατρέξτε στις σχετικές σελίδες βοήθειας και σε κάθε φάκελο που μπορεί να σχετίζεται στην διεύθυνση `/usr/local/docs`.

Οδηγίες OpenMP

Μορφή οδηγιών Fortran

- **Μορφή: (ευαίσθητη σε κεφαλαίους χαρακτήρες)**

Προστασία	Όνομα οδηγίας	Όροι
Όλες οι οδηγίες του OpenMP για την Fortran ξεκινούν με μία προστασία. Οι αποδεχόμενες προστασίες εξαρτώνται από τον τύπο της Fortran. Πιθανές προστασίες είναι: `!$OMP` `C$OMP` `*$OMP`	Μία έγκυρη οδηγία OpenMP. Πρέπει να εμφανίζεται μετά την προστασία και πριν τους όρους.	Προαιρετικοί. Οι όροι μπορεί να είναι σε οποιαδήποτε σειρά και επαναλαμβανόμενοι όσο χρειαστεί εκτός από άλλους περιορισμούς.

- **Παράδειγμα:**

```
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(BETA,PI)
```

- **Σταθερή Μορφή Κώδικα:**
  - Τα `!$OMP C$OMP *$OMP` είναι αποδεχόμενες προστασίες και πρέπει να ξεκινάν στην στήλη 1.
  - Όλοι οι κανόνες σταθερής μορφής της Fortran για το μέγεθος της γραμμής, το κενό, την συνέχεια της γραμμή και τις στήλες σχολίων ισχύουν για όλη την γραμμή της οδηγίας.
  - Οι αρχικές γραμμές οδηγίας πρέπει να έχουν ένα κενό/μηδενικό στην στήλη 6.
  - Οι γραμμές συνέχισης πρέπει να έχουν ένα μη κενό/μηδενικό στην στήλη 6.
- **Ελεύθερη Μορφή Κώδικα**
  - Το `!$OMP` είναι η μόνη αποδεκτή προστασία. Μπορεί να εμφανιστεί σε οποιαδήποτε στήλη αλλά πρέπει να προηγείται μόνο κενός χαρακτήρας.
  - Όλοι οι κανόνες ελεύθερης μορφής της Fortran για το μέγεθος της γραμμής, το κενό, την συνέχεια της γραμμή και τις στήλες σχολίων ισχύουν για όλη την γραμμή της οδηγίας.
  - Οι αρχικές γραμμές της οδηγίας πρέπει να έχουν κενό μετά από την προστασία.
  - Οι γραμμές συνέχισης πρέπει να έχουν έναν χαρακτήρα "&" σαν τον τελευταίο μη-κενό χαρακτήρα σε μια γραμμή. Η ακόλουθη γραμμή πρέπει να ξεκινάει με μια προστασία και μετά τις οδηγίες συνέχισης.
- **Γενικοί κανόνες**
  - Τα σχόλια δεν μπορούν να εμφανίζονται στην ίδια γραμμή με την οδηγία.
  - Μόνο ένα όνομα οδηγίας μπορεί να οριστεί για κάθε οδηγία.
  - Οι μεταγλωττιστές της Fortran οι οποίοι είναι συμβατοί με τις οδηγίες OpenMP γενικά περιλαμβάνουν μια επιλογή της γραμμής εντολών η οποία δίνει εντολή στον μεταγλωττιστή να ενεργοποιήσει και να μεταφράζει όλες τις οδηγίες OpenMP.
  - Αρκετές οδηγίες OpenMP για Fortran έρχονται σε ζεύγη και έχουν την μορφή που φαίνεται παρακάτω. Η οδηγία "τέλος"(end) είναι προαιρετική αλλά συνιστάται για αναγνωσιμότητα.

```
!$OMP directive
  [ structured block of code ]
!$OMP end directive
```

Μορφή οδηγιών C/C++

- **Μορφή:**

#pragma omp	Οδηγία-όνομα	Όροι	Νέα Γραμμή
Απαραίτητο για όλες τις οδηγίες C/C++ του OpenMP.	Μία έγκυρη οδηγία OpenMP.πρέπει να εμφανίζεται μετά το pragma και πριν τους όρους.	Προαιρετικοί. Οι όροι μπορεί να είναι σε οποιαδήποτε σειρά και επαναλαμβανόμενοι όσο χρειαστεί εκτός από άλλους περιορισμούς.	Απαραίτητη. Προηγείται του δομημένου μπλοκ το οποίο περικλείεται από αυτήν την οδηγία.

- **Παράδειγμα:**

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

- **Γενικοί κανόνες:**
  - Διάκριση κεφαλαίων και πεζών.
  - Οι οδηγίες ακολουθούν συμβάσεις του προτύπου C/C++ για τις οδηγίες του μεταγλωττιστή.
  - Μόνο ένα όνομα οδηγίας μπορεί να οριστεί για κάθε οδηγία.
  - Κάθε οδηγία εφαρμόζεται σε μία επιτυχημένη δήλωση το πολύ, η οποία πρέπει να είναι ένα δομημένο μπλοκ.
  - Οι γραμμές με μεγάλες οδηγίες μπορούν να συνεχιστούν στις επόμενες γραμμές παρακάμπτοντας τον χαρακτήρα νέας γραμμής με έναν χαρακτήρα "\" στο τέλος της γραμμής της οδηγίας.

Οριοθέτηση οδηγιών

Να το κάνουμε αυτό τώρα… ή να το κάνουμε αργότερα; Δεν πειράζει, ας τελειώνουμε με αυτό νωρίς.

- **Στατική (Λεξική) Έκταση**
  - Ο κώδικας λεκτικά περικλείεται μεταξύ της αρχής και του τέλους ενός δομημένου μπλοκ που ακολουθεί μια οδηγία.
  - Η στατική έκταση μιας οδηγίας δεν εκτείνει πολλαπλές ρουτίνες ή αρχεία κώδικα.
- **Ορφανή οδηγία:**
  - Μια οδηγία του OpenMP που εμφανίζεται ανεξάρτητα από μια άλλη οδηγία που περικλείει κώδικα λέγεται ορφανή οδηγία. Υπάρχει έξω από την στατική έκταση μιας άλλης οδηγίας.
  - Θα επεκτείνει ρουτίνες και πιθανώς αρχεία κώδικα.
- **Δυναμική Έκταση**
  - Η δυναμική έκταση μιας οδηγίας περιλαμβάνει και την στατική έκταση αλλά και την έκταση των ορφανών οδηγιών της.
- **Παράδειγμα:**

<table>
<tr>
<td>

```
PROGRAM TEST
...
!$OMP PARALLEL
...
!$OMP DO
DO I=...
...
CALL SUB1
...
ENDDO
...
CALL SUB2
...
!$OMP END PARALLEL
```

</td>
<td>

```
SUBROUTINE SUB1
...
!$OMP CRITICAL
...
!$OMP END CRITICAL
END
SUBROUTINE SUB2
...
!$OMP SECTIONS
...
!$OMP END SECTIONS
...
END
```

</td>
</tr>
<tr>
<td>Στατική Έκταση
Η οδηγία DO προκύπτει μέσα σε ένα περικλειόμενο τμήμα παραλληλοποίησης</td>
<td>Ορφανή οδηγία
Οι CRITICAL και SECTIONS οδηγίες προκύπτουν έξω από ένα περικλειόμενο τμήμα παραλληλοποίησης</td>
</tr>
<tr>
<td colspan="2">Δυναμική Έκταση
Οι οδηγίες CRITICAL και SECTIONS προκύπτουν μέσα στην δυναμική έκταση των DO και PARALLEL οδηγιών</td>
</tr>
</table>

- **Γιατί είναι τόσο σημαντικό;**
  - Το OpenMP ορίζει ένα νούμερο κανόνων οριοθέτησης στο πως οι οδηγίες θα συνεργαστούν(συνδεθούν) και εμφωλευτούν η μία μέσα στην άλλη.

- Κατεστραμμένα ή εσφαλμένα προγράμματα μπορεί να προκύψουν αν αγνοηθούν οι κανόνες συνεργασίας και φωλιάσματος του OpenMP.
- Ανατρέξτε το κεφάλαιο Σύνδεση Οδηγίας και Κανόνες Εμφώλευσης για συγκεκριμένες λεπτομέρειες.

Δομή παράλληλου τμήματος

➢ **Σκοπός:**

- Ένα παράλληλο τμήμα είναι ένα μπλοκ κώδικα που θα εκτελεστεί σε πολλαπλά νήματα. Αυτή είναι η θεμελιώδης δομή παραλληλοποίησης του OpenMP.

Fortran	`!$OMP PARALLEL [clause ...]` `IF (scalar_logical_expression)` `PRIVATE (list)` `SHARED (list)` `DEFAULT (PRIVATE \| FIRSTPRIVATE \| SHARED \| NONE)` `FIRSTPRIVATE (list)` `REDUCTION (operator: list)` `COPYIN (list)` `NUM_THREADS (scalar-integer-expression)` `block` `!$OMP END PARALLEL`
C/C++	`#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`

➢ **Σημειώσεις:**

- Όταν ένα νήμα φτάνει στην οδηγία PARALLEL, δημιουργεί μια ομάδα από νήματα και γίνεται ο άρχοντας(master) της ομάδας. Ο άρχοντας είναι ένας μέλος της ομάδας αυτής και έχει αριθμό νήματος 0 μέσα σε αυτήν την ομάδα.
- Αρχίζοντας από την αρχή του παράλληλου τμήματος, ο κώδικας αντιγράφεται και όλα τα νήματα θα εκτελέσουν αυτόν τον κώδικα.

- Υπάρχει ένα σιωπηλό φράγμα στο τέλος του παράλληλου τμήματος. Μόνο το κύριο νήμα θα συνεχίσει την εκτέλεση μετά από αυτό το σημείο.
- Αν κάποιο νήμα τερματιστεί μέσα σε ένα παράλληλο τμήμα, όλα τα νήματα στην ομάδα θα τερματιστούν και η εργασία που έγινε μέχρι εκείνο το σημείο θα είναι απροσδιόριστη.

Πόσα Νήματα Χρειάζονται;

- Ο αριθμός των νημάτων σε ένα παράλληλο τμήμα καθορίζεται από τους ακόλουθους παράγοντες, σε σειρά προτεραιότητας:
  1. Αξιολόγηση του όρου `IF`.
  2. Καθορισμός του όρου `NUM_THREADS`.
  3. Χρήση της συνάρτησης βιβλιοθήκης `omp_set_num_threads()`.
  4. Καθορισμός της μεταβλητής περιβάλλοντος OMP_NUM_THREADS.
  5. Προεπιλεγμένη υλοποίηση - συνήθως ο αριθμός των ΚΜΕ σε έναν κόμβο, αν και θα μπορούσε να είναι δυναμική.
- Τα νήματα αριθμούνται από το 0 (το βασικό νήμα) μέχρι το N-1.

Δυναμικά Νήματα:

- Χρήση της συνάρτησης βιβλιοθήκης `omp_get_dynamic()` για να καθοριστεί αν είναι ενεργοποιημένα τα δυναμικά νήματα.
- Αν υποστηρίζεται, οι δυο μέθοδοι για την ενεργοποίηση των δυναμικών νημάτων είναι:
  1. Η ρουτίνα βιβλιοθήκης `omp_set_dynamic()`.
  2. Θέτοντας την μεταβλητή περιβάλλοντος OMP_DYNAMIC σε TRUE(ΑΛΗΘΗΣ).

Εμφωλευμένα Παράλληλα Τμήματα:

- Χρήση της συνάρτησης βιβλιοθήκης `omp_get_nested()` για να καθοριστεί αν είναι ενεργοποιημένα εμφωλευμένα παράλληλα τμήματα.
- Οι δυο διαθέσιμοι μέθοδοι για ενεργοποίηση των εμφωλευμένων παράλληλων τμημάτων(αν υποστηρίζεται) είναι:
  1. Η ρουτίνα βιβλιοθήκης `omp_set_nested()`.
  2. Θέτοντας την μεταβλητή περιβάλλοντος OMP_NESTED σε TRUE(ΑΛΗΘΗΣ).
- Αν δεν υποστηρίζεται, ένα παράλληλο τμήμα φωλιασμένο μέσα σε ένα άλλο παράλληλο τμήμα έχει σαν αποτέλεσμα την δημιουργία μιας νέας ομάδας, αποτελούμενη από ένα νήμα, από προεπιλογή.

Όροι:

- Ο όρος IF: Αν είναι παρών, πρέπει να αποτιμάται ως TRUE(αληθής στην Fortran) ή μη-μηδενικό(στην C/C++) για να μπορέσει μια ομάδα από νήματα να δημιουργηθεί. Αλλιώς το τμήμα εκτελείται σειριακά από το κύριο νήμα.
- Οι υπόλοιποι όροι περιγράφονται με λεπτομέρειες παρακάτω στο κεφάλαιο Όροι Ιδιοτήτων Πεδίου Δεδομένων.

- **Περιορισμοί:**
  - Ένα παράλληλο τμήμα πρέπει να είναι ένα δομημένο μπλοκ που δεν επεκτείνει πολλαπλές ρουτίνες ή αρχεία κώδικα.
  - Είναι παράνομο να διακλαδίζεται (goto) μέσα ή έξω.
  - Μόνο ένας όρος IF επιτρέπεται.
  - Μόνο ένας όρος NUM_THREADS επιτρέπεται.

Παράδειγμα: Παράλληλο Τμήμα

- Απλό παράδειγμα προγράμματος "Hello World".
  - Κάθε νήμα εκτελεί όλο τον κώδικα που περικλείεται από το παράλληλο τμήμα.
  - Οι ρουτίνες βιβλιοθήκης του OpenMP χρησιμοποιούνται για να αποκτηθούν τα αναγνωριστικά των νημάτων και ο συνολικός αριθμός νημάτων.

Fortran - Παράδειγμα Παράλληλου Τμήματος

```
PROGRAM HELLO
INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS,
+ OMP_GET_THREAD_NUM
C Fork a team of threads with each thread having a private TID variable
!$OMP PARALLEL PRIVATE(TID)
C Obtain and print thread id
TID = OMP_GET_THREAD_NUM()
PRINT *, 'Hello World from thread = ', TID
C Only master thread does this
IF (TID .EQ. 0) THEN
NTHREADS = OMP_GET_NUM_THREADS()
PRINT *, 'Number of threads = ', NTHREADS
END IF
C All threads join master thread and disband
!$OMP END PARALLEL
END
```

C / C++ - Παράδειγμα Παράλληλου Τμήματος

```
#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 */
}
```

Άσκηση OpenMP 1

Επισκόπηση:

- Συνδεθείτε σε μία συστοιχία(cluster) της LC χρησιμοποιώντας το όνομα χρήστη του εργαστηρίου και το διακριτικό OTP σας.
- Αντιγράψτε τα αρχεία της άσκησης στον κεντρικό κατάλογο σας.
- Εξοικειωθείτε με το περιβάλλον του OpenMP.
- Γράψτε ένα απλό "Hello World" πρόγραμμα με το OpenMP.
- Μεταγλωττίστε επιτυχώς το πρόγραμμα σας.
- Τρέξτε επιτυχώς το πρόγραμμα σας.
- Τροποποιήστε τον αριθμό των νημάτων στο πρόγραμμα σας.

Πηγαίνετε στην άσκηση.

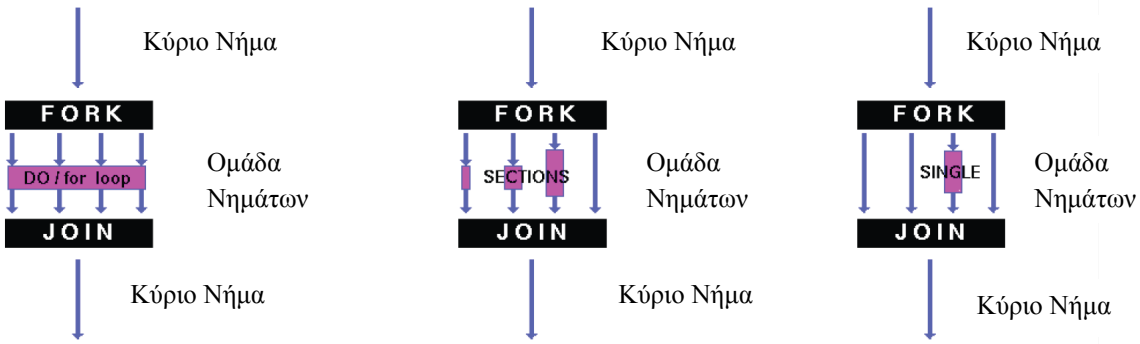
Δομές Διαμοιρασμού Εργασίας

- Μια δομή διαμοιρασμού εργασίας διαιρεί την εκτέλεση του περικλειόμενου τμήματος κώδικα ανάμεσα στα μέλη της ομάδας που το συναντά.
- Μια δομή διαμοιρασμού εργασίας δεν δημιουργεί νέα νήματα.
- Δεν υπάρχει σιωπηλό φράγμα κατά την είσοδο σε μια δομή διαμοιρασμού εργασίας, παρόλα αυτά υπάρχει ένα σιωπηλό φράγμα στο τέλος αυτής.

- **Τύποι Δομών Διαμοιρασμού Εργασίας**

  Σημείωση: Η δομή `workshare` της Fortran δεν εμφανίζεται εδώ αλλά συζητιέται παρακάτω.

  - DO/for: Μοιράζει τις επαναλήψεις ενός βρόγχου στην ομάδα. Αντιπροσωπεύει έναν τύπο "παράλληλων δεδομένων".
  - SECTIONS: Σπάει την εργασία σε ξεχωριστά, διακριτά τμήματα. Κάθε τμήμα εκτελείται από ένα νήμα. Μπορεί να χρησιμοποιηθεί για την υλοποίηση ενός τύπου "λειτουργικό παραλληλισμό".
  - SINGLE: Κάνει σειριακό ένα τμήμα του κώδικα.



- **Περιορισμοί:**
  - Μία δομή διαμοιρασμού εργασίας πρέπει να περικλείεται δυναμικά μέσα σε ένα παράλληλο τμήμα προκειμένου η οδηγία να την εκτελέσει παράλληλα.
  - Οι δομές διαμοιρασμού εργασίας πρέπει να συναντώνται από όλα τα μέλη μιας ομάδας ή από κανένα από αυτά.
  - Επιτυχημένες δομές διαμοιρασμού εργασίας πρέπει να συναντώνται από όλα τα μέλη της ομάδας με την ίδια σειρά.

Δομές Διαμοιρασμού Εργασίας

Η οδηγία DO/for

- **Σκοπός**:
  - Η οδηγία DO/for ορίζει ότι οι επαναλήψεις του βρόγχου που ακολουθεί αμέσως μετά πρέπει να εκτελεστούν παράλληλα από την ομάδα. Αυτό προϋποθέτει ότι ένα παράλληλο τμήμα έχει αρχικοποιηθεί ήδη, αλλιώς η εκτέλεση γίνεται σειριακά σε έναν και μόνο επεξεργαστή.
- **Μορφή:**

Fortran	!$OMP DO *[clause ...]* SCHEDULE *(type [,chunk])* ORDERED PRIVATE *(list)* FIRSTPRIVATE *(list)* LASTPRIVATE *(list)* SHARED *(list)* REDUCTION *(operator \| intrinsic : list)* COLLAPSE *(n)* *do_loop* !$OMP END DO [NOWAIT]
C/C++	#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*

- **Όροι:**
  - **SCHEDULE**: Περιγράφει πως οι επαναλήψεις του βρόγχου χωρίζονται μεταξύ των νημάτων στην ομάδα. Το προεπιλεγμένο χρονοδιάγραμμα εξαρτάται από την υλοποίηση.
    - STATIC: Οι επαναλήψεις του βρόγχου χωρίζονται σε κομμάτια μεγέθους *chunk* και στατικά ανατίθενται στα νήματα. Αν η μεταβλητή chunk δεν έχει οριστεί, οι επαναλήψεις χωρίζονται ομοιόμορφα(αν γίνεται), συνεχόμενα μεταξύ των νημάτων.
    - DYNAMIC: Οι επαναλήψεις του βρόγχου χωρίζονται σε κομμάτια μεγέθους *chunk* και δυναμικά προγραμματίζονται μεταξύ των νημάτων, όταν ένα νήμα τελειώνει ένα κομμάτι, του ανατίθεται δυναμικά ένα άλλο. Το προεπιλεγμένο μέγεθος του κομματιού(chunk) είναι 1.
    - GUIDED: Οι επαναλήψεις ανατίθενται δυναμικά στα νήματα σε μπλοκ, καθώς τα νήματα τα αιτούνται, μέχρι να μην υπάρχει μπλοκ για ανάθεση. Παρόμοιο με το DYNAMIC με την διαφορά ότι το μέγεθος του μπλοκ μειώνεται κάθε φορά που ένα πακέτο εργασίας ανατίθεται σε ένα νήμα. Το μέγεθος του αρχικού μπλοκ είναι ανάλογο του:
      `number_of_iterations / number_of_threads`
      Τα επόμενα μπλοκ είναι ανάλογα του:
      `number_of_iterations_remaining / number_of_threads`

Η παράμετρος chunk ορίζει το ελάχιστο μέγεθος του μπλοκ. Η προεπιλεγμένη τιμή είναι 1.
    - RUNTIME: Η απόφαση προγραμματισμού αναβάλλεται μέχρι την εκτέλεση από την μεταβλητή περιβάλλοντος OMP_SCHEDULE. Είναι παράνομο να οριστεί ένα μέγεθος chunk για αυτόν τον όρο.
    - AUTO: Η απόφαση σχεδιασμού ανατίθεται στον μεταγλωττιστή ή/και στο σύστημα εκτέλεσης.
- NO WAIT/nowait: Αν έχει οριστεί, τότε τα νήματα δεν συγχρονίζονται στο τέλος του παράλληλου βρόγχου.
- ORDERED: Ορίζει ότι οι επαναλήψεις του βρόγχου πρέπει να εκτελεστούν όπως θα εκτελούνταν σε ένα σειριακό πρόγραμμα.
- COLLAPSE: Ορίζει πόσοι βρόγχοι σε ένα έναν εμφωλευμένο βρόγχο θα πρέπει να αναδιπλωθούν σε έναν μεγάλο χώρο επανάληψης και να χωριστούν σύμφωνα με τον όρο `schedule`. Η ακολουθιακή εκτέλεση των επαναλήψεων σε όλους τους σχετιζόμενους βρόγχους καθορίζει την σειρά των επαναλήψεων στον αναδιπλωμένο χώρο επαναλήψεων.
- Άλλοι όροι περιγράφονται αργότερα στο κεφάλαιο Όροι Ιδιοτήτων Πεδίου Δεδομένων.

➢ **Περιορισμοί:**

- Ο βρόγχος DO δεν μπορεί να γίνει DO WHILE ή ένας βρόγχος χωρίς έλεγχο επαναλήψεων. Επίσης η μεταβλητή επαναλήψεων πρέπει να είναι ένας ακέραιος και οι παράμετροι του ελέγχου των επαναλήψεων πρέπει να είναι οι ίδιοι για όλα τα νήματα.
- Η ορθότητα του προγράμματος δεν πρέπει να εξαρτάται από το ποιο νήμα εκτελεί μια συγκεκριμένη επανάληψη.
- Είναι παράνομο να διακλαδίζεται (goto) έξω από έναν σχετιζόμενο βρόγχο με μια οδηγία DO/for.
- Το μέγεθος του chunk πρέπει να οριστεί σαν μια έκφραση ακεραίου, αμετάβλητη από τον βρόγχο, καθώς δεν υπάρχει συγχρονισμός κατά την διάρκεια της αξιολόγησης από διαφορετικά νήματα.
- Οι όροι ORDERED, COLLAPSE και SCHEDULE εμφανίζονται μία φορά ο καθένας.
- Κοιτάξτε το έγγραφο προδιαγραφών του OpenMP για περισσότερους περιορισμούς.

Παράδειγμα Οδηγίας DO/for

- Απλό πρόγραμμα πρόσθεσης διανυσμάτων.
    - Οι πίνακες A,B,C και η μεταβλητή N θα είναι κοινόχρηστοι από όλα τα νήματα.
    - Η μεταβλητή I θα είναι ιδιωτική στο κάθε νήμα, κάθε νήμα θα έχει το δικό του μοναδικό αντίγραφο.

- Οι επαναλήψεις του βρόγχου θα κατανεμηθούν δυναμικά σε κομμάτια μεγέθους CHUNK.
- Τα νήματα δεν θα συγχρονιστούν κατά την ολοκλήρωση του ατομικού τους έργου(NOWAIT).

Fortran - Παράδειγμα Οδηγίας DO

```
PROGRAM VEC_ADD_DO
INTEGER N, CHUNKSIZE, CHUNK, I
PARAMETER (N=1000)
PARAMETER (CHUNKSIZE=100)
REAL A(N), B(N), C(N)
! Some initializations
DO I = 1, N
A(I) = I * 1.0
B(I) = A(I)
ENDDO
CHUNK = CHUNKSIZE
!$OMP PARALLEL SHARED(A,B,C,CHUNK) PRIVATE(I)
!$OMP DO SCHEDULE(DYNAMIC,CHUNK)
DO I = 1, N
C(I) = A(I) + B(I)
ENDDO
!$OMP END DO NOWAIT
!$OMP END PARALLEL
END
```

C / C++ - Παράδειγμα Οδηγίας for

```
#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 */
}
```

Η οδηγία SECTIONS

- **Σκοπός:**
  - Η οδηγία SECTIONS είναι μία μη επαναληπτική δομή διαμοιρασμού εργασίας. Ορίζει ότι το τμήμα(τα) κώδικα που περικλείεται θα μοιραστεί μεταξύ των νημάτων της ομάδας.
  - Ανεξάρτητες οδηγίες SECTION είναι εμφωλευμένες μέσα σε μια οδηγία SECTIONS. Κάθε οδηγία SECTIONS εκτελείται μια φορά από ένα νήμα της ομάδας. Διαφορετικά τμήματα μπορεί να εκτελεστούν από διαφορετικά νήματα. Είναι πιθανό για ένα νήμα να εκτελέσει περισσότερα του ενός τμήματα αν είναι αρκετά γρήγορο και αν το επιτρέπει η υλοποίηση.

- **Μορφή:**

Fortran	`!$OMP SECTIONS [clause ...]` `PRIVATE (list)` `FIRSTPRIVATE (list)` `LASTPRIVATE (list)` `REDUCTION (operator \| intrinsic : list)` `!$OMP SECTION` `block` `!$OMP SECTION` `block` `!$OMP END SECTIONS [ NOWAIT ]`
C/C++	`#pragma omp sections [clause ...] newline` `private (list)` `firstprivate (list)` `lastprivate (list)` `reduction (operator: list)` `nowait` `{` `  #pragma omp section newline` `  structured_block` `  #pragma omp section newline` `  structured_block` `}`

- **Όροι:**
  - Υπάρχει ένα σιωπηλό φράγμα στο τέλος κάθε οδηγίας SECTIONS εκτός και αν χρησιμοποιηθεί ο όρος `NOWAIT/nowait`.
  - Οι όροι περιγράφονται αργότερα στο κεφάλαιο Όροι Ιδιοτήτων Πεδίου Δεδομένων.
- **Ερωτήσεις:**

- Τι συμβαίνει αν ο αριθμός των νημάτων και ο αριθμός των SECTION είναι διαφορετικός; Περισσότερα νήματα από τα SECTION; Λιγότερα νήματα από τα SECTION;
- Ποιο νήμα εκτελεί ποιο SECTION;

➢ **Περιορισμοί:**

- Είναι παράνομο να διακλαδίζεται (goto) μέσα και έξω από τμήματα μπλοκ.
- Οι οδηγίες SECTION πρέπει να εμφανίζονται μέσα στην λεκτική έκταση που περικλείει η οδηγία SECTIONS(όχι ορφανές SECTION).

Παράδειγμα: Η οδηγία SECTIONS

- Απλό πρόγραμμα που δείχνει ότι τα διαφορετικά μπλοκ εργασίας θα εκτελεστούν από διαφορετικά νήματα.

Fortran - Παράδειγμα Οδηγίας SECTIONS

```
PROGRAM VEC_ADD_SECTIONS
INTEGER N, I
PARAMETER (N=1000)
REAL A(N), B(N), C(N), D(N)
! Some initializations
DO I = 1, N
  A(I) = I * 1.5
  B(I) = I + 22.35
ENDDO
!$OMP PARALLEL SHARED(A,B,C,D), PRIVATE(I)
!$OMP SECTIONS
!$OMP SECTION
DO I = 1, N
  C(I) = A(I) + B(I)
ENDDO
!$OMP SECTION
DO I = 1, N
D(I) = A(I) * B(I)
ENDDO
!$OMP END SECTIONS NOWAIT
!$OMP END PARALLEL
END
```

C / C++ - Παράδειγμα Οδηγίας sections

```
#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 */
}
```

Οδηγία WORSKSHARE

➢ Σκοπός:

- Μόνο στην Fortran.
- Η οδηγία WORKSHARE διαιρεί την εκτέλεση ενός περικλειόμενου δομημένου μπλοκ σε μονάδες εργασίας, κάθε μια από τις οποίες εκτελείται μόνο μια φορά.
- Το δομημένο μπλοκ πρέπει να αποτελείται μόνο από τα ακόλουθα:
  - Αναθέσεις σε πίνακες
  - Βαθμωτές αναθέσεις
  - Δηλώσεις FORALL
  - Δομές FORALL
  - Δηλώσεις WHERE
  - Δομές WHERE
  - Δομές ατόμων
  - Κρίσιμες Δομές
  - Παράλληλες Δομές
- Ανατρέξτε στα έγγραφα σχετικά με το OpenMP API για περισσότερες πληροφορίες, ιδιαίτερα για το τι περιλαμβάνει μια "μονάδα εργασίας".

➢ Μορφή:

Fortran	`!$OMP WORKSHARE` `  structured block` `!$OMP END WORKSHARE [ NOWAIT ]`

➢ Περιορισμοί:

- Η δομή δεν πρέπει να έχει καμία συνάρτηση ορισμένη από τον χρήστη εκτός αν η συνάρτηση είναι ELEMENTAL(στοιχειακή).

Παράδειγμα: Η οδηγία WORKSHARE

- Απλός πίνακας και βαθμωτές αναθέσεις κοινόχρηστες από την ομάδα των νημάτων. Μια μονάδα εργασίας θα περιλαμβάνει:
  - Οποιαδήποτε βαθμωτή ανάθεση.
  - Για δηλώσεις ανάθεσης πίνακα, η ανάθεση κάθε στοιχείου είναι μια μονάδα εργασίας.

Fortran - Παράδειγμα Οδηγίας WORKSHARE

```
PROGRAM WORKSHARE
INTEGER N, I, J
PARAMETER (N=100)
REAL AA(N,N), BB(N,N), CC(N,N), DD(N,N), FIRST, LAST
! Some initializations
DO I = 1, N
DO J = 1, N
AA(J,I) = I * 1.0
BB(J,I) = J + 1.0
ENDDO
ENDDO
!$OMP PARALLEL SHARED(AA,BB,CC,DD,FIRST,LAST)
!$OMP WORKSHARE
CC = AA * BB
DD = AA + BB
FIRST = CC(1,1) + DD(1,1)
LAST = CC(N,N) + DD(N,N)
!$OMP END WORKSHARE NOWAIT
!$OMP END PARALLEL
```

Οδηγία SINGLE

➢ Σκοπός:

- Η οδηγία SINGLE ορίζει ότι ο κλεισμένος κώδικας θα εκτελεστεί μόνο από ένα νήμα σε κάθε ομάδα.
- Μπορεί να είναι χρήσιμο κατά την ενασχόληση με τμήματα κώδικα που δεν είναι ασφαλή με νήματα(όπως η E/E).

➢ Μορφή:

Fortran	`!$OMP SINGLE [clause ...]` `PRIVATE (list)` `FIRSTPRIVATE (list)` `block` `!$OMP END SINGLE [ NOWAIT ]`
C/C++	`#pragma omp single [clause ...] newline` `private (list)` `firstprivate (list)` `nowait`

	structured_block

➢ **Όροι:**

- Τα νήματα της ομάδας που δεν εκτελούν την οδηγία SINGLE, περιμένουν στο τέλος του κλειστού μπλοκ κώδικα, εκτός αν έχει οριστεί ένας όρος `NOWAIT/nowait`.
- Οι όροι περιγράφονται με λεπτομέρειες αργότερα στο κεφάλαιο Όροι Ιδιοτήτων Πεδίου Δεδομένων.

➢ **Περιορισμοί:**

- Απαγορεύεται να διακλαδίζονται προς τα μέσα ή έξω σε ένα μπλοκ οδηγίας SINGLE.

Συνδυασμένη Δομή Παράλληλου Διαμοιρασμού Εργασίας

- Το OpenMP παρέχει 3 οδηγίες που είναι ιδανικές:
  - PARALLEL DO / parallel for
  - PARALLEL SECTIONS
  - PARALLEL WORKSHARE (μόνο για την Fortran)
- Ως επί το πλείστον, αυτές οι οδηγίες συμπεριφέρονται πανομοιότυπα με μια ατομική οδηγία PARALLEL, ακολουθούμενη από μια ξεχωριστή οδηγία διαμοιρασμού εργασίας.
- Οι περισσότεροι από τους κανόνες, όρους και περιορισμούς που εφαρμόζονται και στις δυο οδηγίες είναι σε ισχύ. Ανατρέξτε στην προγραμματιστική διεπαφή εφαρμογής του OpenMP για λεπτομέρειες.
- Ένα παράδειγμα της χρήσης της συνδυασμένης οδηγίας PARALLEL DO/parallel for φαίνεται παρακάτω.

Fortran - PARALLEL DO Directive Example

```
PROGRAM VECTOR_ADD
INTEGER N, I, CHUNKSIZE, CHUNK
PARAMETER (N=1000)
PARAMETER (CHUNKSIZE=100)
REAL A(N), B(N), C(N)
! Some initializations
DO I = 1, N
A(I) = I * 1.0
B(I) = A(I)
ENDDO
CHUNK = CHUNKSIZE
!$OMP PARALLEL DO
!$OMP& SHARED(A,B,C,CHUNK) PRIVATE(I)
!$OMP& SCHEDULE(STATIC,CHUNK)
DO I = 1, N
```

```
C(I) = A(I) + B(I)
ENDDO
!$OMP END PARALLEL DO
END
```

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];
}
```

Δομή TASK

- **Σκοπός:**
  - Νέα δομή με το OpenMP 3.0
  - Η δομή TASK ορίζει μία σαφή εργασία, η οποία μπορεί να εκτελεστεί από το νήμα που θα την συναντήσει ή να αναβληθεί για εκτέλεση από οποιοδήποτε άλλο νήμα στην ομάδα.
  - Τα δεδομένα περιβάλλοντος της εργασίας προσδιορίζονται από τους χαρακτηριστικούς όρους των κοινόχρηστων δεδομένων.
  - Η εκτέλεση της εργασίας υπόκειται σε προγραμματισμό εργασιών - ανατρέξτε στο έγγραφο προδιαγραφών του OpenMP 3.0 για περισσότερες λεπτομέρειες.

- **Μορφή:**

```
!$OMP TASK [clause ...]
```

Fortran	IF *(scalar logical expression)* FINAL *(scalar logical expression)* UNTIED DEFAULT (PRIVATE | FIRSTPRIVATE | SHARED | NONE) MERGEABLE PRIVATE *(list)* FIRSTPRIVATE *(list)* SHARED *(list)* *block* !$OMP END TASK
C/C++	#pragma omp task *[clause ...] newline* if *(scalar expression)* final *(scalar expression)* untied default (shared | none) mergeable private *(list)* firstprivate *(list)* shared *(list)* *structured_block*

- **Όροι και περιορισμοί:**
  - Κοιτάξτε το έγγραφο προδιαγραφών του OpenMP 3.0 για περισσότερους περιορισμούς.

Άσκηση OpenMP 2

Επισκόπηση:

- Συνδεθείτε σε μία συστοιχία(cluster) της LC, αν δεν έχετε συνδεθεί ήδη.
- Παραδείγματα δομής διαμοιρασμού εργασίας Do/for: επανεξετάστε, μεταγλωττίστε και εκτελέστε τα παραδείγματα.
- Παραδείγματα δομής διαμοιρασμού εργασίας SECTIONS: επανεξετάστε, μεταγλωττίστε και εκτελέστε τα παραδείγματα.

Πηγαίνετε στην άσκηση.

Δομές Συγχρονισμού

- Σκεφτείτε ένα απλό παράδειγμα όπου δυο νήματα σε δυο διαφορετικές διεργασίες προσπαθούν να αυξήσουν μια μεταβλητή x την ίδια στιγμή(ας υποθέσουμε ότι αρχικά η τιμή 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)`

- Μία πιθανή ακολουθία εκτέλεσης είναι η εξής:
  1. Το νήμα 1 φορτώνει την τιμή του x στον καταχωρητή A.
  2. Το νήμα 2 φορτώνει την τιμή του x στον καταχωρητή A.
  3. Το νήμα 1 προσθέτει 1 στον καταχωρητή A.
  4. Το νήμα 2 προσθέτει 1 στον καταχωρητή A.
  5. Το νήμα 1 αποθηκεύει τον καταχωρητή A στην διεύθυνση της x.
  6. Το νήμα 2 αποθηκεύει τον καταχωρητή A στην διεύθυνση της x.

  Η προκύπτουσα τιμή θα είναι 1 και όχι 2 όπως θα έπρεπε να είναι.

- Για να αποφύγουμε καταστάσεις σαν και αυτή, η αύξηση της x πρέπει να γίνεται συγχρονισμένα μεταξύ των δύο νημάτων για να εξασφαλίσουμε ότι θα παραχθεί το σωστό αποτέλεσμα.
- Το OpenMP παρέχει ποικιλία δομών συγχρονισμού που μπορούν να ελέγξουν πως προχωρά η εκτέλεση του κάθε νήματος , σε σχέση με τα άλλα νήματα.

Η οδηγία MASTER

- **Σκοπός:**
  - Η οδηγία MASTER ορίζει ένα τμήμα που θα εκτελεστεί μόνο από το κύριο νήμα. Όλα τα άλλα νήματα της ομάδας θα προσπεράσουν αυτό το κομμάτι κώδικα.

- Δεν υπάρχει σιωπηλό φράγμα που να συνδέεται με αυτήν την οδηγία.

➢ **Μορφή:**

Fortran	`!$OMP MASTER` *block* `!$OMP END MASTER`
C/C++	`#pragma omp master` *newline* *structured_block*

➢ **Περιορισμοί:**

- Είναι παράνομο να διακλαδίζεται μέσα ή έξω από το μπλοκ της οδηγίας MASTER.

Η οδηγία CRITICAL

➢ **Σκοπός:**

- Η οδηγία CRITICAL ορίζει ένα τμήμα κώδικα που πρέπει να εκτελείται μόνο από ένα νήμα κάθε φορά.

➢ **Μορφή:**

Fortran	`!$OMP CRITICAL` *[name]* *block* `!$OMP END CRITICAL` *[name]*
C/C++	`#pragma omp critical` *[name] newline* *structured_block*

➢ **Σημειώσεις:**

- Αν ένα νήμα εκτελεί μέσα σε ένα τμήμα CRITICAL και άλλο ένα νήμα φτάνει σε αυτό το τμήμα και προσπαθήσει να το εκτελέσει, θα μπλοκαριστεί μέχρι το πρώτο νήμα να βγει από το τμήμα αυτό.
- Το προαιρετικό όνομα επιτρέπει σε διαφορετικά τμήματα CRITICAL να υπάρξουν:
  - Τα ονόματα δρουν σαν καθολικά αναγνωριστικά. Διαφορετικά τμήματα CRITICAL με το ίδιο όνομα αντιμετωπίζονται σαν το ίδιο τμήμα.
  - Όλα τα τμήματα CRITICAL που δεν έχουν όνομα, αντιμετωπίζονται σαν το ίδιο τμήμα.

➢ **Περιορισμοί:**

- Είναι παράνομο να διακλαδίζεται μέσα ή έξω από το μπλοκ της οδηγίας CRITICAL.
- Μόνο για την Fortran: Τα ονόματα της critical δομής είναι καθολικές οντότητες του προγράμματος. Αν ένα όνομα συγκρούεται με μία άλλη οντότητα, η συμπεριφορά του προγράμματος είναι απροσδιόριστη.

Παράδειγμα: Δομή CRITICAL

- Όλα τα νήματα της ομάδας θα προσπαθήσουν να εκτελεστούν παράλληλα, παρόλα αυτά, εξαιτίας της δομής CRITICAL που περικλείει την αύξηση του x, μόνο ένα νήμα θα είναι ικανό να διαβάσει/αυξήσει/γράψει το x κάθε στιγμή.

Fortran-Παράδειγμα οδηγίας CRITICAL

```
PROGRAM CRITICAL
INTEGER X
X = 0
!$OMP PARALLEL SHARED(X)
!$OMP CRITICAL
X = X + 1
!$OMP END CRITICAL
!$OMP END PARALLEL
END
```

C/C++ Παράδειγμα οδηγίας CRITICAL

```
#include <omp.h>
main()
{
int x;
x = 0;
#pragma omp parallel shared(x)
{
#pragma omp critical
x = x + 1;
} /* end of parallel section */

}
```

Οδηγία BARRIER

- **Σκοπός:**
  - Η οδηγία BARRIER συγχρονίζει όλα τα νήματα της ομάδας.
  - Όταν βρεθεί μια οδηγία BARRIER, το νήμα θα περιμένει σε εκείνο το σημείο μέχρι τα άλλα νήματα να φτάσουν σε εκείνο το φράγμα. Όλα τα νήματα τότε συνεχίζουν την παράλληλη εκτέλεση του κώδικα που ακολουθεί το φράγμα.

- **Μορφή:**

Fortran	`!$OMP BARRIER`
C/C++	`#pragma omp barrier` *`newline`*

- **Περιορισμοί:**
  - Όλα τα νήματα της ομάδας(ή κανένα) πρέπει να εκτελέσουν το τμήμα BARRIER.
  - Η ακολουθία των τμημάτων διαμερισμού εργασίας και των τμημάτων φράγματος πρέπει να είναι ίδια για κάθε νήμα της ομάδας.

Οδηγία TASKWAIT

- **Σκοπός:**
  - Νέο με OpenMP 3.0
  - Η δομή TASKWAIT ορίζει μία αναμονή στη ολοκλήρωση των θυγατρικών εργασιών που δημιουργήθηκαν από την αρχή της τωρινής εργασίας.

- **Μορφή:**

Fortran	`!$OMP TASKWAIT`
C/C++	`#pragma omp taskwait newline`

- **Περιορισμοί:**

- Επειδή η δομή taskwait δεν έχει κάποιο όρισμα στη γλώσσα C ως μέρος του συντακτικού της, υπάρχουν κάποιοι περιορισμοί στη θέση της μέσα στο πρόγραμμα. Η οδηγία taskwait μπορεί να τοποθετηθεί μόνο στο σημείο όπου μια δήλωση της βασικής γλώσσας επιτρέπεται. Η οδηγία taskwait μπορεί να μη μπορεί να χρησιμοποιηθεί στη θέση του ορισμού ακολουθούμενη από μια if,do switch, ή label.Ανατρέξτε στις διευκρινίσεις του κειμένου OpenMP 3.0 για λεπτομέρειες.

Οδηγία ATOMIC

- **Σκοπός:**
  - Η οδηγία ATOMIC ορίζει ότι μια συγκεκριμένη θέση μνήμης πρέπει να ενημερώνεται αυτόματα, αντί να επιτρέπεται σε πολλαπλά νήματα να προσπαθούν να γράψουν σε αυτήν. Στην ουσία, αυτή η οδηγία παρέχει ένα μικρό CRITICAL τμήμα.

- **Μορφή:**

Fortran	`!$OMP ATOMIC` `  statement_expression`
C/C++	`#pragma omp atomic newline` `  statement_expression`

- **Περιορισμοί:**
  - Η οδηγία έχει εφαρμογή μόνο σε μία μονή δήλωση, η οποία ακολουθεί αμέσως μετά την οδηγία.
  - Μία ατομική δήλωση πρέπει να ακολουθεί συγκεκριμένη σύνταξη. Ανατρέξτε στα πιο πρόσφατα χαρακτηριστικά του OpenMP για λεπτομέρειες.

Οδηγία FLUSH

- **Σκοπός:**
  - Η οδηγία FLUSH αναγνωρίζει ένα σημείο συγχρονισμού στο οποίο η υλοποίηση πρέπει να παρέχει μια συνεπή εικόνα της μνήμης. Οι μεταβλητές που είναι ορατές από τα νήματα γράφονται πίσω στην μνήμη σε αυτό το σημείο.
  - Υπάρχει ένα σημαντικό ποσοστό συζήτησης για αυτήν την οδηγία στους κύκλους του OpenMP το οποίο μπορείτε να συμβουλευτείτε για περισσότερες πληροφορίες. Μερικά από αυτά είναι δύσκολα να τα καταλάβετε; Για το API:

- Εάν η διασταύρωση των σετ των μηδενισμών(flush) από 2 μηδενισμούς που πραγματοποιείται από δυο διαφορετικά νήματα δεν είναι άδεια , τότε οι δυο μηδενισμοί πρέπει να ολοκληρώσουν όσο είναι μέσα σε κάποια ακολουθιακή εντολή, που είναι ορατή από όλα τα νήματα.

Τι λέτε;

- Παράθεση από τις συχνές ερωτήσεις και απαντήσεις του Openmp.org .

Ερώτηση 17: Είναι η οδηγία !$omp flush απαραίτητη σε ένα σύστημα συνεκτικής κρυφής μνήμης;

Απάντηση 17: Ναι η οδηγία flush είναι απαραίτητη. Κοιτάξτε στις προδιαγραφές του OpenMP για παραδείγματα της χρήσης της. Η οδηγία είναι απαραίτητη για να αναθέσει στον μεταγλωττιστή την εντολή ότι αυτή η μεταβλητή πρέπει να γράφεται/διαβάζεται από την μνήμη του συστήματος πχ ότι η μεταβλητή δεν μπορεί να κρατηθεί σε έναν τοπικό καταχωρητή ΚΜΕ μετά την "δήλωση" της flush στον κώδικα σας.

Η συνεκτικότητα της κρυφής μνήμης κάνει σίγουρο ότι αν μία ΚΜΕ εκτελεί μια οδηγία ανάγνωσης ή εγγραφής από/προς την μνήμη και όλες οι άλλες ΚΜΕ στο σύστημα θα πάρουν την ίδια τιμή από αυτήν την διεύθυνση μνήμης όταν αποκτήσουν πρόσβαση σε αυτήν. Όλες οι κρυφές μνήμες θα δείχνουν μια συνεκτική τιμή. Παρόλα αυτά, στο πρότυπο του OpenMP πρέπει να υπάρχει ένας τρόπος να ανατεθεί στον μεταγλωττιστή η εντολή να εισάγει αυτές τις οδηγίες μηχανής για την ανάγνωση/εγγραφή και να μην τις αναβάλει. Η διατήρηση μιας μεταβλητής σε έναν καταχωρητή σε έναν βρόγχο είναι πολύ κοινή όταν παράγεται αποδοτικός κώδικας γλώσσας μηχανής για έναν βρόγχο.

- Επίσης δείτε τα πιο πρόσφατα κεφάλαια OpenMP για λεπτομέρειες.

➢ Μορφή:

Fortran	`!$OMP FLUSH (list)`
C/C++	`#pragma omp flush (list) newline`

➢ Σημειώσεις:

- Η προαιρετική λίστα περιέχει μια λίστα με ονοματισμένες μεταβλητές οι οποίες θα μηδενιστούν ώστε να αποφευχθεί ο μηδενισμός όλων των μεταβλητών. Για τους δείκτες της λίστας, σημειώστε ότι ο δείκτης του μηδενίζεται όχι το αντικείμενο που δείχνει.
- Οι υλοποιήσεις πρέπει να εξασφαλίζουν ότι οποιεσδήποτε προηγούμενες τροποποιήσεις σε μεταβλητές που είναι ορατές από τα νήματα, είναι ορατές σε όλα τα νήματα μετά από αυτό το σημείο, πχ. οι μεταγλωττιστές πρέπει να επαναφέρουν τις τιμές από τους

καταχωρητές στην μνήμη, το υλικό μπορεί να πρέπει να μηδενίσει γραμμένες προσωρινές μνήμες κτλ.

- Η οδηγία FLUSH υπονοείται για τις οδηγίες που φαίνονται στον πίνακα παρακάτω. Η οδηγία δεν υπονοείται εάν η οδηγία NOWAIT είναι παρούσα.

Fortran	C/C++
BARRIER END PARALLEL CRITICAL and END CRITICAL END DO END SECTIONS END SINGLE ORDERED και END ORDERED	`barrier` `parallel` - κατά την είσοδο και την έξοδο `critical` - κατά την είσοδο και την έξοδο `ordered` - κατά την είσοδο και την έξοδο `for` - κατά την έξοδο `sections` - κατά την έξοδο `single` - κατά την έξοδο

Οδηγία ORDERED

- **Σκοπός:**
  - Η οδηγία ORDERED ορίζει ότι οι επαναλήψεις του περικλειόμενου βρόγχου θα εκτελεστούν με την ίδια σειρά σαν να εκτελούνταν σε έναν σειριακό επεξεργαστή.
  - Τα νήματα θα πρέπει να περιμένουν πριν εκτελέσουν το κομμάτι των επαναλήψεων τους εάν οι προηγούμενες επαναλήψεις δεν έχουν ολοκληρωθεί ακόμα.
  - Χρησιμοποιείται μέσα ένα DO/for βρόγχο με έναν όρο ORDERED.
  - Η οδηγία DIRECTIVE παρέχει ένα τρόπο τελειοποίησης για το που πρόκειται να εφαρμοστεί η διάταξη σε έναν βρόγχο. Αλλιώς δεν χρειάζεται.

- **Μορφή:**

Fortran	`!$OMP DO ORDERED` *`[clauses...]`* *`(loop region)`* `!$OMP ORDERED` *`(block)`* `!$OMP END ORDERED` *`(end of loop region)`* `!$OMP END DO`
C/C++	`#pragma omp for ordered` *`[clauses...]`* *`(loop region)`* `#pragma omp ordered` *`newline`* *`structured_block`* *`(end of loop region)`*

- **Περιορισμοί:**
  - Μια οδηγία ORDERED μπορεί μόνο να εμφανιστεί στη δυναμική έκταση των παρακάτω οδηγιών:
    - DO ή PARALLEL DO (Fortran)
    - `for` ή `parallel for` (C/C++)
  - Μόνο ένα νήμα επιτρέπεται σε ένα συγκεκριμένο τμήμα κάθε στιγμή.
  - Είναι παράνομο να διακλαδιστεί μέσα ή έξω από το μπλοκ της εντολής ORDERED .
  - Μια επανάληψη του βρόγχου δεν πρέπει να εκτελέσει την ίδια οδηγία ORDERED πάνω από μια φορά και δεν πρέπει να εκτελέσει πάνω από μια οδηγία ORDERED.
  - Ένας βρόγχος ο οποίος περιέχει μια οδηγία ORDERED, πρέπει να είναι ένας βρόγχος με έναν όρο ORDERED.

Οδηγία THREADPRIVATE

- **Σκοπός:**
  - Η οδηγία THREADPRIVATE χρησιμοποιείται για να δημιουργήσει μεταβλητές καθολικής εμβέλειας αρχείων (C/C++) ή κοινά μπλοκ (Fortran) τοπικά και διατηρούμενα σε ένα νήμα μέσα από την εκτέλεση των πολλαπλών παράλληλων τμημάτων.
- **Μορφή:**

Fortran	`!$OMP THREADPRIVATE (/cb/, ...)` *`cb is the name of a common block`*
C/C++	`#pragma omp threadprivate` *`(list)`*

- **Σημειώσεις:**
  - Η οδηγία πρέπει να εμφανίζεται μετά από την δήλωση της λίστας μεταβλητών/κοινών μπλοκ. Κάθε νήμα τότε παίρνει το δικό του αντίγραφο των μεταβλητών/κοινών μπλοκ, ώστε τα εγγεγραμμένα δεδομένα από ένα νήμα δεν είναι ορατό στα άλλα νήματα. Για παράδειγμα:

Fortran - Παράδειγμα Οδηγίας THREADPRIVATE

```
PROGRAM THREADPRIV
INTEGER A, B, I, TID, OMP_GET_THREAD_NUM
REAL*4 X
COMMON /C1/ A
!$OMP THREADPRIVATE(/C1/, X)
C Explicitly turn off dynamic threads
CALL OMP_SET_DYNAMIC(.FALSE.)
PRINT *, '1st Parallel Region:'
!$OMP PARALLEL PRIVATE(B, TID)
```

```
TID = OMP_GET_THREAD_NUM()
A = TID
B = TID
X = 1.1 * TID + 1.0
PRINT *, 'Thread',TID,': A,B,X=',A,B,X
!$OMP END PARALLEL
PRINT *, '***********************************'
PRINT *, 'Master thread doing serial work here'
PRINT *, '***********************************'
PRINT *, '2nd Parallel Region: '
!$OMP PARALLEL PRIVATE(TID)
TID = OMP_GET_THREAD_NUM()
PRINT *, 'Thread',TID,': A,B,X=',A,B,X
!$OMP END PARALLEL
END

-----------------------------------------------

Έξοδος:
1st Parallel Region:
Thread 0 : A,B,X= 0 0 1.000000000
Thread 1 : A,B,X= 1 1 2.099999905
Thread 3 : A,B,X= 3 3 4.300000191
Thread 2 : A,B,X= 2 2 3.200000048
***********************************
Master thread doing serial work here
***********************************
2nd Parallel Region:
Thread 0 : A,B,X= 0 0 1.000000000
Thread 2 : A,B,X= 2 0 3.200000048
Thread 3 : A,B,X= 3 0 4.300000191
Thread 1 : A,B,X= 1 0 2.099999905
```

C/C++ - Παράδειγμα Οδηγίας threadprivate

```
#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 */
}

-----------------------------------------------

Έξοδος:
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
```

- Στην πρώτη είσοδο του παράλληλου τμήματος, τα δεδομένα των μεταβλητών THREADPRIVATE και κοινών μπλοκ πρέπει να

χαρακτηριστούν ως απροσδιόριστα εκτός αν έχει οριστεί ένας όρος COPYIN στην παράλληλη οδηγία.

- Οι μεταβλητές THREADPRIVATE διαφέρουν από τις μεταβλητές PRIVATE(θα συζητηθούν αργότερα) επειδή αυτές είναι ικανές να διατηρηθούν ανάμεσα σε διαφορετικά παράλληλα τμήματα κώδικα.

➢ **Περιορισμοί:**

- Τα δεδομένα στα αντικείμενα των THREADPRIVATE είναι εγγυημένο ότι θα διατηρηθούν μόνο αν ο μηχανισμός των δυναμικών νημάτων έχει απενεργοποιηθεί και ο αριθμός των νημάτων σε διαφορετικές παράλληλες περιοχές παραμένει συνεχής. Ο καθορισμός της επιλογής των δυναμικών νημάτων είναι απροσδιόριστος.
- Η οδηγία THREADPRIVATE πρέπει να εμφανιστεί πριν από κάθε δήλωση μιας ιδιωτικής μεταβλητής/κοινού μπλοκ ενός νήματος.
- Fortran: Μόνο κοινά μπλοκ που έχουν ονομαστεί μπορούν να γίνουν THREADPRIVATE.

Όροι Ιδιοτήτων Πεδίου Δεδομένων

- Επίσης ονομάζονται και Όροι Ιδιοτήτων Διαμοιρασμού Δεδομένων.
- Μια σημαντική σκέψη για τον προγραμματισμό με OpenMP είναι η κατανόηση και χρήση της οριοθέτησης των δεδομένων.
- Επειδή το OpenMP είναι βασισμένο στο μοντέλο προγραμματισμού κοινόχρηστης μνήμης, οι περισσότερες μεταβλητές είναι κοινόχρηστες από προεπιλογή.
- Οι καθολικές μεταβλητές περιέχουν:
  - Fortran: μπλοκ COMMON , μεταβλητές SAVE, μεταβλητές MODULE.
  - C: Μεταβλητές οριοθέτησης αρχείου, στατικά.

- Οι ιδιωτικές μεταβλητές περιλαμβάνουν:
  - Μεταβλητές του δείκτη του βρόγχου.
  - Μεταβλητές στοίβας σε υπορουτίνες που έχουν κληθεί από παράλληλα τμήματα.
  - Fortran: Αυτόματες μεταβλητές μέσα μια δήλωση μπλοκ .

- Οι όροι ιδιοτήτων πεδίων δεδομένων του OpenMP χρησιμοποιούνται ρητά για να ορίσουν το πεδίο εφαρμογής των μεταβλητών. Αυτοί περιλαμβάνουν:
    - PRIVATE
    - FIRSTPRIVATE
    - LASTPRIVATE
    - SHARED
    - DEFAULT
    - REDUCTION
    - COPYIN
- Οι όροι ιδιοτήτων πεδίων δεδομένων χρησιμοποιούνται σε συνδυασμό με αρκετές οδηγίες(PARALLEL,DO/for και SECTIONS) για να ελέγχουν την οριοθέτηση των περικλειόμενων μεταβλητών.
- Αυτές οι δομές παρέχουν την ικανότητα να ελέγχουν το περιβάλλον δεδομένων κατά τη διάρκεια εκτέλεσης των παράλληλων δομών.
    - Αυτές καθορίζουν πως και ποιες μεταβλητές δεδομένων στο σειριακό τμήμα του προγράμματος μεταφέρονται στα παράλληλα τμήματα των προγραμμάτων του προγράμματος(και πίσω).
    - Αυτές καθορίζουν ποιες μεταβλητές θα είναι ορατές σε όλα τα νήματα στα παράλληλα τμήματα και ποιες μεταβλητές θα κατανεμηθούν ιδιωτικά σε όλα τα νήματα.
- Οι όροι ιδιοτήτων πεδίων δεδομένων είναι αποτελεσματικά μόνο μέσα στις λεξιλογικές/στατικές τους επεκτάσεις.
- **Σημαντικό:** Παρακαλώ συμβουλευτείτε το πιο πρόσφατο χαρακτηριστικά του OpenMP για σημαντικές λεπτομέρειες και συζήτηση πάνω στο θέμα.
- Παρέχεται ένας Συνοπτικός Πίνακας Οδηγιών/Όρων για ευκολία

Όρος PRIVATE

- **Σκοπός:**
    - Ο όρος PRIVATE ορίζει ότι οι μεταβλητές στην λίστα του είναι κρυφές από το κάθε νήμα.
- **Μορφή:**

Fortran	`PRIVATE (list)`
C/C++	`private (list)`

- **Σημειώσεις:**
    - Οι μεταβλητές PRIVATE συμπεριφέρονται όπως ακολουθεί:
        - Ένα νέο αντικείμενο του ίδιου τύπου ορίζεται για κάθε νήμα της ομάδας.

- Όλες οι αναφορές στο αρχικό αντικείμενο αντικαθιστώνται από αναφορές στο καινούργιο αντικείμενο.
- Μεταβλητές που ορίζονται PRIVATE πρέπει να θεωρηθεί ότι είναι αρχικοποιημένες για κάθε νήμα.

- Σύγκριση μεταξύ PRIVATE και THREADPRIVATE:

	PRIVATE	THREADPRIVATE
Αντικείμενο Δεδομένων	C/C++: μεταβλητή Fortran: μεταβλητή ή κοινό μπλοκ	C/C++: μεταβλητή Fortran: κοινό μπλοκ
Που ορίζεται	Στην αρχή του τμήματος ή της ομάδας διαμοιρασμού εργασίας	Στις δηλώσεις κάθε ρουτίνας που χρησιμοποιεί οριοθέτηση μπλοκ ή καθολικού αρχείου
Διατηρείται;	Όχι	Ναι
Έκταση	Λεκτική μόνο - εκτός αν περαστεί σαν όρισμα σε υπορουτίνα	Δυναμική
Αρχικοποίηση	Χρήση FIRSTPRIVATE	Χρήση COPYIN

Όρος SHARED

- **Σκοπός:**
  - Ο όρος SHARED ορίζει ότι οι μεταβλητές στην λίστα του θα είναι κοινόχρηστες μεταξύ όλων των νημάτων της ομάδας.
- **Μορφή:**

Fortran	`SHARED (list)`
C/C++	`shared (list)`

- **Σημειώσεις:**
  - Μια κοινόχρηστη μεταβλητή υπάρχει μόνο σε μια τοποθεσία μνήμης και όλα τα νήματα μπορούν να διαβάσουν ή να γράψουν σε αυτήν την διεύθυνση.
  - Είναι ευθύνη του προγραμματιστή να διασφαλίσει ότι πολλαπλά νήματα έχουν σωστή πρόσβαση στις μεταβλητές SHARED (όπως στα τμήματα CRITICAL).

Όρος DEFAULT

- **Σκοπός:**

- Ο όρος DEFAULT επιτρέπει στον χρήστη ορίσει μία προεπιλεγμένη οριοθέτηση για όλες τις μεταβλητές στην λεξική έκταση κάθε παράλληλου τμήματος.

➢ **Μορφή:**

Fortran	`DEFAULT (PRIVATE | FIRSTPRIVATE | SHARED | NONE)`
C/C++	`default (shared | none)`

➢ **Σημειώσεις:**

- Συγκεκριμένες μεταβλητές μπορεί να εξαιρούνται από τις προεπιλογές χρησιμοποιώντας τους όρους PRIVATE, SHARED, FIRSTPRIVATE, LASTPRIVATE, και REDUCTION.
- Οι προδιαγραφές του OpenMP για την C/C++ δεν περιλαμβάνουν τους όρους private ή firstprivate σαν μια πιθανή προεπιλογή. Παρόλα αυτά οι πραγματικές υλοποιήσεις μπορεί να παρέχουν αυτήν την επιλογή.
- Χρησιμοποιώντας NONE σαν προεπιλογή απαιτεί ότι ο προγραμματιστής ρητά οριοθετεί όλες τις μεταβλητές .

➢ **Περιορισμοί**:

- Μόνο ένας όρος DEFAULT μπορεί να καθοριστεί σε μία οδηγία PARALLEL.

Όρος FIRSTPRIVATE

➢ **Σκοπός:**

- Ο όρος FIRSTPRIVATE συνδυάζει την συμπεριφορά του όρου PRIVATE με αυτόματη αρχικοποίηση των μεταβλητών στην λίστα.

➢ **Μορφή:**

Fortran	`FIRSTPRIVATE` *`(list)`*
C/C++	`firstprivate` *`(list)`*

➢ **Σημειώσεις:**

- Οι μεταβλητές στην λίστα αρχικοποιούνται σύμφωνα με την τιμή των αρχικών αντικειμένων τους πριν από την έναρξη της παράλληλης δομής ή της δομής διαμοιρασμού εργασίας.

Όρος LASTPRIVATE

- **Σκοπός:**
  - Ο όρος LASTTPRIVATE συνδυάζει την συμπεριφορά του όρου PRIVATE με ένα αντίγραφο από την τελευταία επανάληψη του βρόγχου ή του τμήματος του αρχικού αντικειμένου της μεταβλητής.
- **Μορφή:**

Fortran	`LASTPRIVATE (list)`
C/C++	`lastprivate (list)`

- **Σημειώσεις:**
  - Η μεταβλητή που αντιγράφεται πίσω στο αρχικό αντικείμενο της μεταβλητής αποκτάται από την τελευταία (σειριακή) επανάληψη ή το τμήμα από την περικλειόμενη την δομή.

    Για παράδειγμα το μέλος της ομάδας το οποίο εκτελεί την τελευταία επανάληψη του for ενός τμήματος DO, ή το μέλος το οποίο έκανε το τελευταίο SECTION από ένα πλαίσιο SECTIONS κάνει την αντιγραφή με τις δικές του τιμές.

Όρος COPYIN

- **Σκοπός:**
  - Ο όρος COPYIN παρέχει έναν τρόπο για ανάθεση της ίδιας τιμής στις TRHEADPRIVATE μεταβλητές για όλα τα νήματα στην ομάδα.
- **Μορφή:**

Fortran	`COPYIN (list)`
C/C++	`copyin (list)`

- **Σημειώσεις:**
  - Η λίστα περιέχει τα ονόματα των μεταβλητών προς αντιγραφή. Στην Fortran η λίστα μπορεί να περιέχει και ονόματα από κοινά μπλοκ και από ονόματα μεταβλητών.

- Η μεταβλητή του κύριου νήματος χρησιμοποιείται σαν την πηγή της αντιγραφής. Τα νήματα της ομάδας αρχικοποιούνται με την τιμή του κατά την είσοδο στην παράλληλη δομή.

Όρος COPYPRIVATE

- **Σκοπός:**
  - Ο όρος COPYPRIVATE μπορεί να χρησιμοποιηθεί για να αναμεταδώσει τιμές αποκτημένες από ένα μόνο νήμα απευθείας σε όλες τις οντότητες των ιδιωτικών μεταβλητών στα άλλα νήματα.
  - Συσχετίζεται με την οδηγία SINGLE.
  - Κοιτάξτε το νεότερο έγγραφο προδιαγραφών για περισσότερες πληροφορίες και παραδείγματα.
- **Μορφή:**

Fortran	`COPYPRIVATE (list)`
C/C++	`copyprivate (list)`

Όρος REDUCTION

- **Σκοπός:**
  - Ο όρος REDUCTION πραγματοποιεί μια μείωση στις μεταβλητές που εμφανίζονται στην λίστα.
  - Ένα ιδιωτικό αντίγραφο για κάθε μεταβλητή της λίστας δημιουργείται για κάθε νήμα. Στο τέλος της μείωσης, η μεταβλητή μείωσης εφαρμόζεται σε όλα τα ιδιωτικά αντίγραφα της κοινόχρηστης μεταβλητής και τα τελικά αποτελέσματα γράφονται στην καθολική κοινόχρηστη μεταβλητή.
- **Μορφή:**

Fortran	`REDUCTION (operator|intrinsic: list)`
C/C++	`REDUCTION (operator|intrinsic: list)`

- **Παράδειγμα: REDUCTION - Vector Dot Product:**

- Οι επαναλήψεις του παράλληλου βρόγχου θα κατανεμηθούν σε ίσου μεγέθους μπλοκ το κάθε ένα σε ένα νήμα της ομάδας(SCHEDULE STATIC).
- Στο τέλος της δομής του παράλληλου βρόγχου, όλα τα νήματα θα προσθέσουν τις δικές τους τιμές του "αποτελέσματος" για να ενημερωθεί το καθολικό αντίγραφο του κύριου νήματος.

Fortran - Παράδειγμα Όρου REDUCTION

```
PROGRAM DOT_PRODUCT
INTEGER N, CHUNKSIZE, CHUNK, I
PARAMETER (N=100)
PARAMETER (CHUNKSIZE=10)
REAL A(N), B(N), RESULT
! Some initializations
DO I = 1, N
A(I) = I * 1.0
B(I) = I * 2.0
ENDDO
RESULT= 0.0
CHUNK = CHUNKSIZE
!$OMP PARALLEL DO
!$OMP& DEFAULT(SHARED) PRIVATE(I)
!$OMP& SCHEDULE(STATIC,CHUNK)
!$OMP& REDUCTION(+:RESULT)
DO I = 1, N
RESULT = RESULT + (A(I) * B(I))
ENDDO
!$OMP END PARALLEL DO
PRINT *, 'Final Result= ', RESULT
END
```

C / C++ - Παράδειγμα Όρου reduction

```
#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 μεταβλητές στην λίστα πρέπει να είναι ονοματισμένες βαθμωτές μεταβλητές. Δεν μπορούν να είναι μεταβλητές τύπου πίνακα ή δομές. Πρέπει να είναι επίσης δηλωμένες SHARED στο περικλειόμενο πλαίσιο.
  - Δραστηριότητες μείωσης μπορεί να μην συσχετίζονται με πραγματικούς αριθμούς.
  - Ο όρος REDUCTION προορίζεται να χρησιμοποιηθεί σε μια περιοχή ή μια δομή διαμοιρασμού εργασίας στην οποία η μεταβλητή μείωσης θα χρησιμοποιηθεί μόνο σε μια από τις παρακάτω μορφές:

Fortran	C/C++
x = x operator expr *x = expr operator x (except subtraction)* *x = intrinsic(x, expr)* *x = intrinsic(expr, x)*	*x = x op expr* *x = expr op x (except subtraction)* *x binop = expr* *x++* *++x* *x--* *--x*
○ ***x*** είναι μια βαθμωτή μεταβλητή στην λίστα ○ ***expr*** είναι μια βαθμωτή έκφραση που δεν αναφέρεται στην *x* ○ ***intrinsic*** είναι ένα από τα MAX, MIN, IAND, IOR, IEOR ○ ***operator*** είναι ένα από τα +, *, -, .AND., .OR., .EQV., .NEQV.	○ ***x*** είναι μια βαθμωτή μεταβλητή στην λίστα ○ ***expr*** είναι μια βαθμωτή έκφραση που δεν αναφέρεται στην *x* ○ ***op*** δεν είναι υπερφορτωμένος και είναι ένας από τους +, *, -, /, &, ^, |, &&, || ○ ***binop*** δεν είναι υπερφορτωμένος και είναι ένας από τους +, *, -, /, &, ^, |

Σύνοψη Όρων/Οδηγιών

- Ο πίνακας παρακάτω συνοψίζει ποιο όρος δέχεται ποια οδηγία του OpenMP.

Όροι	Οδηγίες					
	PARALLEL	**DO/for**	**SECTIONS**	**SINGLE**	**PARALLEL DO/for**	**PARALLEL SECTIONS**
IF	□				□	□
PRIVATE	□	□	□	□	□	□
SHARED	□	□			□	□
DEFAULT	□				□	□
FIRSTPRIVATE	□	□	□	□	□	□
LASTPRIVATE		□	□		□	□
REDUCTION	□	□	□		□	□
COPYIN	□				□	□
COPYPRIVATE				□		
SCHEDULE		□			□	
ORDERED		□			□	
NOWAIT		□	□	□		

- Οι ακόλουθες οδηγίες του OpenMP δεν δέχονται όρους:
  - MASTER
  - CRITICAL
  - BARRIER
  - ATOMIC
  - FLUSH
  - ORDERED
  - THREADPRIVATE
- Οι υλοποιήσεις μπορεί (και το κάνουν) να διαφέρουν από το πρότυπο στο οποίο οι όροι υποστηρίζονται από κάθε οδηγία.

Κανόνες Οδηγιών Σύνδεσης και Φωλιάσματος

- Αυτό το τμήμα παρέχεται κυρίως σαν μια γρήγορη αναφορά στους κανόνες που διέπουν τις οδηγίες και την σύνδεση του OpenMP. Οι χρήστες πρέπει να συμβουλευτούν την τεκμηρίωση της υλοποίησης τους και το πρότυπο του OpenMP για άλλους κανόνες και περιορισμούς.
- Εκτός αν υποδεικνύεται διαφορετικά, οι κανόνες εφαρμόζονται τόσο στην υλοποίηση της Fortran όσο και στης C/C++.
- Σημείωση: Η προγραμματιστική διεπαφή εφαρμογής της Fortran ορίζει επίσης έναν αριθμό από κανόνες Μεταβλητών Περιβάλλοντος. Οι κανόνες αυτοί δεν έχουν αναπαραχθεί εδώ.

➢ Οδηγία Συνδέσμου:

- Οι οδηγίες DO/for,SECTIONS , MASTER και BARRIER συνδέονται στο δυναμικά περικλειόμενο PARALLEL, εάν υπάρχει. Αν δεν υπάρχει κανένα παράλληλο τμήμα που να έχει εκτελεστεί, οι οδηγίες δεν έχουν κανένα αποτέλεσμα.
- Η οδηγία ORDERED συνδέεται στις δυναμικά περικλειόμενες DO/for.
- Η οδηγία ATOMIC επιβάλλει αποκλειστική πρόσβαση σε σχέση με τις οδηγίες ATOMIC σε όλα τα νήματα, όχι μόνο στην τρέχουσα ομάδα.
- Η οδηγία CRITICAL επιβάλλει αποκλειστική πρόσβαση σε σχέση με τις οδηγίες CRITICAL σε όλα τα νήματα, όχι μόνο στην τρέχουσα ομάδα.
- Μια οδηγία δε μπορεί ποτέ να συνδεθεί σε καμία οδηγία εκτός της πλησιέστερης περικλειόμενης PARALLEL.

➢ Οδηγία Εμφώλευσης:

- Μία περιοχή διαμοιρασμού εργασίας μπορεί να μην είναι στενά εμφωλευμένη μέσα σε μια περιοχή διαμοιρασμού εργασίας, ρητής εργασίας, κρίσιμη, διατεταγμένη, ατομική ή κύρια.
- Μία περιοχή φράγματος μπορεί να μην είναι στενά εμφωλευμένη μέσα σε μια περιοχή διαμοιρασμού εργασίας, ρητής εργασίας, κρίσιμη, διατεταγμένη, ατομική ή κύρια.
- Μία κύρια περιοχή μπορεί να μην είναι στενά εμφωλευμένη μέσα μια περιοχή διαμοιρασμού εργασίας, ατομική ή ρητής εργασίας.
- Μία διατεταγμένη περιοχή μπορεί να μην είναι στενά εμφωλευμένη μέσα μια κρίσιμη περιοχή, ατομική ή ρητής εργασίας.
- Μια διατεταγμένη περιοχή πρέπει να είναι στενά εμφωλευμένη σε μια περιοχή βρόγχου(ή σε μια περιοχή παράλληλου βρόγχου) με έναν διατεταγμένο όρο.
- Μια κρίσιμη περιοχή μπορεί να μην είναι εμφωλευμένη (στενά ή όχι) μέσα σε μια κρίσιμη περιοχή με το ίδιο όνομα. Σημειώστε ότι αυτός ο περιορισμός δεν είναι αρκετός για να εμποδίσει το αδιέξοδο.
- Παράλληλες, flush,κρίσιμες, ατομικές, απόδοσης εργασίας και ρητής εργασίας περιοχές μπορεί να μην είναι στενά εμφωλευμένες σε μια ατομική περιοχή.

Βιβλιοθήκες ρουτινών εκτέλεσης

➢ Επισκόπηση

- Η προγραμματιστική διεπαφή εφαρμογής του OpenMP περιλαμβάνει έναν αυξανόμενος αριθμό ρουτινών βιβλιοθηκών εκτέλεσης.
- Αυτές οι ρουτίνες χρησιμοποιούνται για διάφορους σκοπούς όπως φαίνεται στον πίνακα παρακάτω:

Ρουτίνα	Σκοπός
OMP_SET_NUM_THREADS	Καθορίζει τον αριθμό των νημάτων που θα χρησιμοποιηθούν στο επόμενο παράλληλο τμήμα.
OMP_GET_NUM_THREADS	Επιστρέφει τον αριθμό των νημάτων που βρίσκονται αυτή τη στιγμή στην ομάδα που εκτελεί το παράλληλο τμήμα από το οποίο καλέστηκε.
OMP_GET_MAX_THREADS	Επιστρέφει τη μέγιστη τιμή η οποία μπορεί να επιστραφεί από την κλήση της συνάρτησης OMP_GET_NUM_THREADS.
OMP_GET_THREAD_NUM	Επιστρέφει τον αριθμό του νήματος από το νήμα, μέσα από την ομάδα, που έκανε αυτήν την κλήση.
OMP_GET_THREAD_LIMIT	Επιστρέφει το μέγιστο αριθμό των νημάτων του OpenMP που είναι διαθέσιμα σε ένα πρόγραμμα.
OMP_GET_NUM_PROCS	Επιστέφει τον μέγιστο αριθμό των επεξεργαστών που είναι διαθέσιμοι σε ένα πρόγραμμα .
OMP_IN_PARALLEL	Χρησιμοποιείται για να αποφασίσει εάν το τμήμα του κώδικα που εκτελείται είναι παράλληλο ή όχι.
OMP_SET_DYNAMIC	Ενεργοποιεί ή απενεργοποιεί την δυναμική προσαρμογή(από το σύστημα εκτέλεσης) του αριθμού των νημάτων που χρειάζονται για την εκτέλεση του παράλληλου τμήματος.
OMP_GET_DYNAMIC	Χρησιμοποιείται για να καθορίσει αν η δυναμική προσαρμογή νημάτων είναι ενεργοποιημένη ή όχι.
OMP_SET_NESTED	Χρησιμοποιείται για να ενεργοποιήσει ή να απενεργοποιήσει την εμφωλευμένη παραλληλοποίηση.
OMP_GET_NESTED	Χρησιμοποιείται για να καθορίσει αν είναι ενεργοποιημένη ή όχι η εμφωλευμένη παραλληλοποίηση
OMP_SET_SCHEDULE	Θέτει την πολιτική προγραμματισμού του βρόγχου

	όταν χρησιμοποιείται η "runtime" σαν είδος χρονοδιαγράμματος στην οδηγία OpenMP.
OMP_GET_SCHEDULE	Επιστρέφει την πολιτική προγραμματισμού του βρόγχου όταν χρησιμοποιείται η "runtime" σαν είδος χρονοδιαγράμματος στην οδηγία OpenMP.
OMP_SET_MAX_ACTIVE_LEVELS	Θέτει τον μέγιστο αριθμό εμφωλευμένων παράλληλων τμημάτων.
OMP_GET_MAX_ACTIVE_LEVELS	Επιστρέφει τον μέγιστο αριθμό εμφωλευμένων παράλληλων τμημάτων.
OMP_GET_LEVEL	Επιστρέφει τον τρέχοντα βαθμό από εμφωλευμένα παράλληλα τμήματα.
OMP_GET_ANCESTOR_THREAD_NUM	Επιστρέφει, για έναν δοσμένο βαθμό φωλιάσματος το νούμερο του πατρικού νήματος.
OMP_GET_TEAM_SIZE	Επιστρέφει, για έναν δοσμένο βαθμό φωλιάσματος το μέγεθος της ομάδας νημάτων.
OMP_GET_ACTIVE_LEVEL	Επιστρέφει το μέγεθος των εμφωλευμένων, ενεργών παράλληλων τμημάτων που περικλείουν την εργασία που περιέχει την κλήση.
OMP_IN_FINAL	Επιστρέφει αληθές αν η ρουτίνα της εκτέλεσης είναι στην τελική περιοχή της εργασίας αλλιώς επιστρέφει ψευδές.
OMP_INIT_LOCK	Αρχικοποιεί μια κλειδαριά που συνδέεται με την μεταβλητή lock.
OMP_DESTROY_LOCK	Αποσυνδέει την μεταβλητή lock από οποιαδήποτε κλειδαριά.
OMP_SET_LOCK	Αποκτά την κυριότητα μιας κλειδαριάς.
OMP_UNSET_LOCK	Απελευθερώνει μια κλειδαριά.
OMP_TEST_LOCK	Δοκιμάζει να ορίσει μια κλειδαριά αλλά δεν μπλοκάρει αν η κλειδαριά δεν είναι διαθέσιμη.
OMP_INIT_NEST_LOCK	Αρχικοποιεί μία εμφωλευμένη κλειδαριά συνδεδεμένη με την μεταβλητή lock.
OMP_DESTROY_NEST_LOCK	Αποσυνδέει την εμφωλευμένη μεταβλητή κλειδαριάς από οποιαδήποτε κλειδαριά.
OMP_SET_NEST_LOCK	Αποκτά κυριότητα μιας εμφωλευμένης κλειδαριάς.
OMP_UNSET_NEST_LOCK	Απελευθερώνει μια εμφωλευμένη κλειδαριά.
OMP_TEST_NEST_LOCK	Δοκιμάζει να ορίσει μια εμφωλευμένη κλειδαριά αλλά δεν μπλοκάρει αν η κλειδαριά δεν είναι διαθέσιμη.
OMP_GET_WTIME	Παρέχει μια φορητή ρουτίνα ώρας.
OMP_GET_WTICK	Επιστρέφει μία τιμή διπλής ακρίβειας ίση με τον αριθμό των δευτερολέπτων μεταξύ δύο επιτυχόντων χτύπων του ρολογιού.

- Για την C/C++ όλες οι βιβλιοθήκες ρουτινών εκτέλεσης είναι στην πραγματικότητα υπορουτίνες. Για την Fortran μερικές είναι στην πραγματικότητα συναρτήσεις και μερικές υπορουτίνες. Για παράδειγμα.

Fortran	`INTEGER FUNCTION OMP_GET_NUM_THREADS()`
C/C++	`#include <omp.h>` `int omp_get_num_threads(void)`

- Σημειώστε ότι για την C/C++ συνήθως χρειάζεται να συμπεριλάβετε το αρχείο κεφαλίδων `<omp.h>`.
- Οι ρουτίνες της Fortran δεν είναι ευαίσθητες σε κεφαλαίους χαρακτήρες ενώ οι ρουτίνες της C/C++ είναι.
- Για τις ρουτίνες/συναρτήσεις Κλειδώματος:
  - Η μεταβλητή lock πρέπει να είναι προσβάσιμη μόνο μέσω των ρουτινών κλειδώματος.
  - Για την Fortran, η μεταβλητή κλειδώματος πρέπει να είναι τύπου ακεραίου και αρκετά μεγάλη για να κρατήσει μια διεύθυνση.
  - Για την C/C++, η μεταβλητή κλειδώματος πρέπει να έχει τύπο `omp_lock_t` ή τύπο `omp_nest_lock_t`, ανάλογα με την συνάρτηση που θα χρησιμοποιηθεί.
- Σημειώσεις υλοποίησης:
  - Οι υλοποιήσεις μπορεί να υποστηρίζουν ή όχι όλα τα χαρακτηριστικά της προγραμματιστικής διεπαφής εφαρμογής του OpenMP. Για παράδειγμα, αν υποστηρίζεται η εμφωλευμένη παραλληλοποίηση, μπορεί να είναι μόνο ονομαστική και σε ένα εμφωλευμένο τμήμα κώδικα να έχει μόνο ένα νήμα.
  - Συμβουλευτείτε την τεκμηρίωση της υλοποίησης σας ή πειραματιστείτε και βρείτε το μόνοι σας αν δεν μπορείτε να το βρείτε στην τεκμηρίωση.
- Οι βιβλιοθήκες ρουτινών εκτέλεσης συζητούνται με περισσότερες λεπτομέρειες στο Appendix A.

Μεταβλητές Περιβάλλοντος

- Το OpenMP παρέχει τις ακόλουθες μεταβλητές περιβάλλοντος για τον έλεγχο της εκτέλεσης του παράλληλου κώδικα.
- Όλα τα ονόματα των μεταβλητών περιβάλλοντος είναι με κεφαλαία. Οι τιμές που ανατίθενται σε αυτές δεν είναι ευαίσθητες σε κεφαλαίους χαρακτήρες.

- OMP_SCHEDULE: Ισχύει μόνο για τις DO,PARALLEL DO(Fortran) και for,parallel for(C/C++) οδηγίες οι οποίες έχουν τους όρους προγραμματισμού τους ορισμένους σε RUNTIME. Η τιμή αυτής της μεταβλητής καθορίζει πως οι επαναλήψεις του βρόγχου προγραμματίζονται στον επεξεργαστή. Για παράδειγμα:
  ```
  setenv OMP_SCHEDULE "guided, 4"
  setenv OMP_SCHEDULE "dynamic"
  ```
- OMP_NUM_THREADS: Θέτει τον μέγιστο αριθμό από νήματα που θα χρησιμοποιηθεί κατά την διάρκεια της εκτέλεσης. Για παράδειγμα:
  ```
  setenv OMP_NUM_THREADS 8
  ```
- OMP_DYNAMIC: Ενεργοποιεί ή απενεργοποιεί την δυναμική ρύθμιση του αριθμού των νημάτων που είναι διαθέσιμα για εκτέλεση σε παράλληλα τμήματα. Οι έγκυρες τιμές είναι TRUE ή FALSE. Για παράδειγμα:
  ```
  setenv OMP_DYNAMIC TRUE
  ```
  Σημειώσεις υλοποίησης:
  - Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.
- OMP_PROC_BIND: Νέο χαρακτηριστικό με το OpenMP 3.0. Ενεργοποιεί ή απενεργοποιεί την δέσμευση νημάτων στον επεξεργαστή. Οι έγκυρες τιμές είναι TRUE ή FALSE. Για παράδειγμα:
  ```
  setenv OMP_PROC_BIND TRUE
  ```
  Σημειώσεις υλοποίησης:
  - Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.
- OMP_NESTED: Ενεργοποιεί ή απενεργοποιεί την εμφωλευμένη παραλληλοποίηση. Οι έγκυρες τιμές είναι TRUE ή FALSE. Για παράδειγμα:
  ```
  setenv OMP_NESTED TRUE
  ```
  Σημειώσεις υλοποίησης:
  - Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.
- OMP_STACKSIZE: Νέο χαρακτηριστικό με το OpenMP 3.0. Ελέγχει το μέγεθος της στοίβας για τα νήματα που έχουν δημιουργηθεί(όχι το κύριο). 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
```

Σημειώσεις υλοποίησης:

- Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.

o OMP_WAIT_POLICY: Νέο χαρακτηριστικό με το OpenMP 3.0. Παρέχει μια υπόδειξη στην υλοποίηση του OpenMP σχετικά με την επιθυμητή συμπεριφορά των νημάτων που περιμένουν. Μία υλοποίηση συμβατή με το OpenMP μπορεί να συμμορφώνεται ή όχι με τον καθορισμό των μεταβλητών περιβάλλοντος. Οι έγκυρες τιμές είναι ACTIVE και PASSIVE. Η ACTIVE ορίζει ότι τα νήματα που περιμένουν πρέπει να είναι κυρίως ενεργά, πχ να καταναλώνουν κύκλους του επεξεργαστή όσο θα περιμένουν. Η PASSIVE ορίζει ότι τα νήματα που περιμένουν πρέπει να είναι κυρίως παθητικά, πχ να μην καταναλώνουν κύκλους του επεξεργαστή όσο περιμένουν. Οι λεπτομέρειες για την συμπεριφορά των ACTIVE και PASSIVE ορίζονται από την υλοποίηση. Παραδείγματα:

```
setenv OMP_WAIT_POLICY ACTIVE
setenv OMP_WAIT_POLICY active
setenv OMP_WAIT_POLICY PASSIV
setenv OMP_WAIT_POLICY passive
```

Σημειώσεις υλοποίησης:

- Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.

o OMP_MAX_ACTIVE_LEVELS: Νέο χαρακτηριστικό με το OpenMP 3.0. Ελέγχει τον μέγιστο αριθμό των εμφωλευμένων ενεργών παράλληλων τμημάτων. Η τιμή αυτής της μεταβλητής περιβάλλοντος πρέπει να είναι θετικός ακέραιος. Η συμπεριφορά του προγράμματος ορίζεται από την υλοποίηση, αν η τιμή που ζητήθηκε από την OMP_MAX_ACTIVE_LEVELS είναι μεγαλύτερη από μέγιστο αριθμό των εμφωλευμένων ενεργών παράλληλων τμημάτων που μπορεί να υποστηρίξει η υλοποίηση ή αν η τιμή είναι ένας μη μηδενικός ακέραιος. Παράδειγμα:

```
setenv OMP_MAX_ACTIVE_LEVELS 2
```

Σημειώσεις υλοποίησης:

- Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.

o OMP_THREAD_LIMIT: Νέο χαρακτηριστικό με το OpenMP 3.0. Ορίζει τον αριθμό των νημάτων του OpenMP που θα χρησιμοποιηθούν από ολόκληρο το πρόγραμμα. Η τιμή αυτής της μεταβλητής περιβάλλοντος πρέπει να είναι ένας θετικός ακέραιος. Η συμπεριφορά του προγράμματος ορίζεται από την υλοποίηση, αν τιμή που ζητήθηκε από την OMP_THREAD_LIMIT είναι μεγαλύτερη από μέγιστο αριθμό των νημάτων που μπορεί να

υποστηρίξει η υλοποίηση ή αν η τιμή είναι ένας αρνητικός ακέραιος. Παράδειγμα:

```
setenv OMP_THREAD_LIMIT 8
```

Σημειώσεις υλοποίησης:

- Η υλοποίηση σας μπορεί να υποστηρίζει αυτό το χαρακτηριστικό ή όχι.

Μέγεθος στοίβας Νήματος

- Το πρότυπο OpenMP δεν προσδιορίζει πόσο χώρο στην στοίβα μπορεί να έχε ένα νήμα. Κατά συνέπεια, οι υλοποιήσεις διαφέρουν στο προεπιλεγμένο μέγεθος στοίβας νήματος.
- Το προεπιλεγμένο μέγεθος στοίβας νήματος μπορεί να εξαντληθεί εύκολα. Μπορεί επίσης να μην είναι φορητό μεταξύ των μεταγλωττιστών. Για παράδειγμα, ο παρακάτω πίνακας δείχνει μερικά όρια μεγέθους στοίβας νημάτων κατά προσέγγιση που χρησιμοποιούνται στους προεπιλεγμένους μεταγλωττιστές της LC.(Αύγουστος 2011).

Μεταγλωττιστής	Όριο Στοίβας κατά Προσέγγιση	Μέγεθος Πίνακα κατά Προσέγγιση(διπλός)
Linux icc, ifort	4MB	700*700
Linux pgcc, pgf90	8MB	1000*1000
Linux gcc, gfortran	2MB	500*500

- Τα νήματα που ξεπερνάνε την κατανομή της μνήμης τους μπορεί να εμφανίσουν σφάλμα κατάτμησης ή όχι. Μια εφαρμογή μπορεί να συνεχίσει να τρέχει ενώ τα δεδομένα καταστρέφονται.
- Στατικά συνδεδεμένοι κώδικες μπορεί να υπόκεινται σε περεταίρω περιορισμούς στοίβας.
- Το κέλυφος εισόδου ενός χρήστη μπορεί επίσης να περιορίσει το μέγεθος της στοίβας.
- Αν ο μεταγλωττιστής σας υποστηρίζει OpenMP 3.0 την μεταβλητή περιβάλλοντος `OMP_STACKSIZE`(καλύπτεται σε προηγούμενη ενότητα) μπορείτε να την χρησιμοποιήσετε για να θέσετε το μέγεθος της στοίβας των νημάτων πριν από την εκτέλεση του προγράμματος. Για παράδειγμα:

```
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
```

- Αλλιώς, στην LC, μπορείτε να χρησιμοποιήσετε την παρακάτω μέθοδο για συστοιχίες με Linux. Το παράδειγμα δείχνει πώς να οριστεί το μέγεθος της στοίβας του νήματος στα 12MB και σαν μια προφύλαξη θέτει το μέγεθος της στοίβας του κελύφους σε απεριόριστο.

csh/tcsh	`setenv KMP_STACKSIZE 12000000` `limit stacksize unlimited`
ksh/sh/bash	`export KMP_STACKSIZE=12000000` `ulimit -s unlimited`

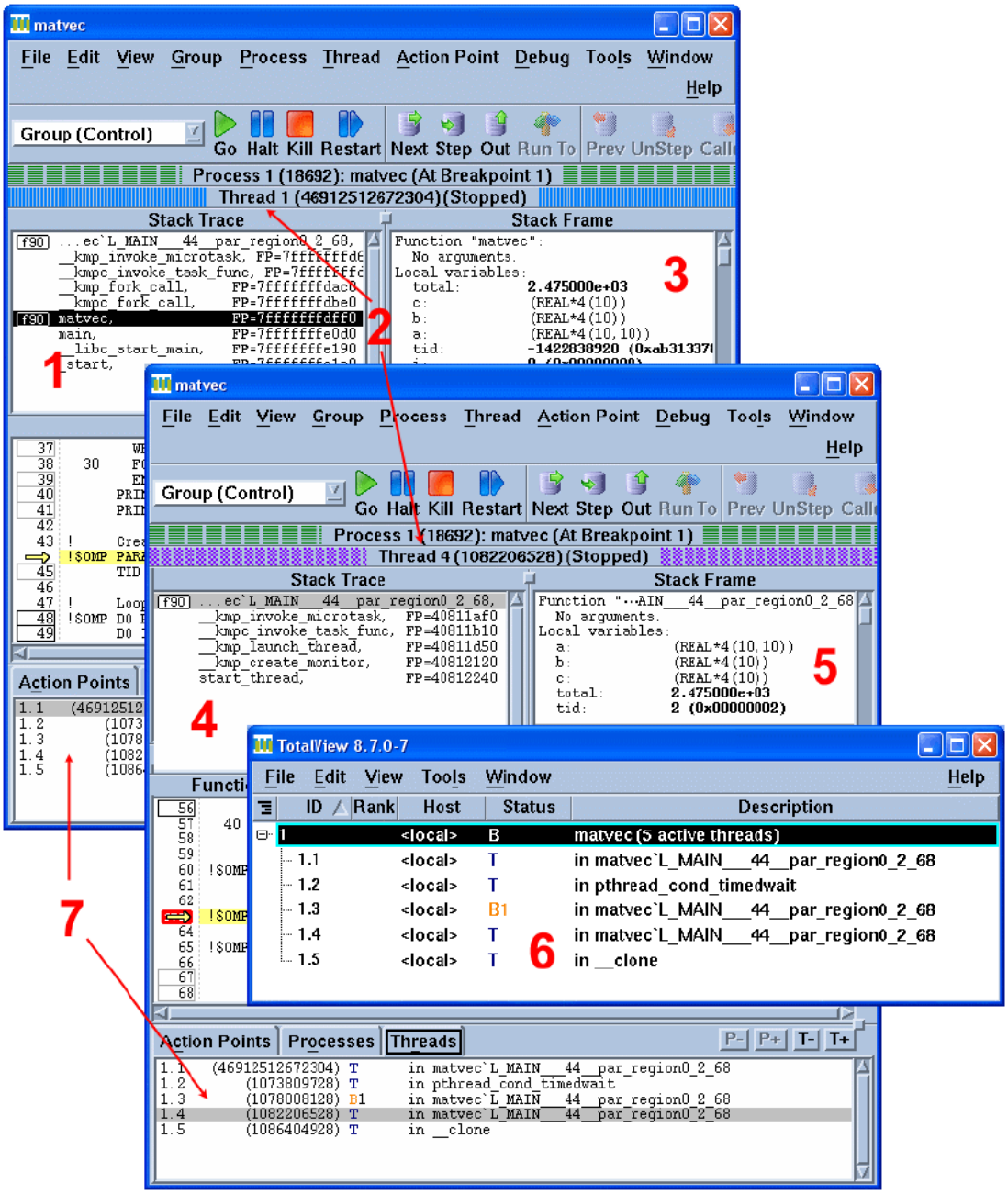
➢ Σύνδεση Νημάτων

- Σε μερικές περιπτώσεις το πρόγραμμα θα εκτελεστεί καλύτερα αν τα νήματα είναι δεσμευμένα με τον επεξεργαστή/πυρήνα.
- "Συνδέοντας" ένα νήμα σε έναν επεξεργαστή σημαίνει ότι το νήμα θα προγραμματιστεί από το λειτουργικό σύστημα για να τρέχει πάντα στον ίδιο επεξεργαστή. Αλλιώς τα νήματα μπορεί να προγραμματιστούν για εκτέλεση σε οποιονδήποτε επεξεργαστή και να πηγαίνουν και να έρχονται μεταξύ των επεξεργαστών σπαταλώντας χρόνο.
- Ονομάζεται επίσης "συνάφεια νήματος" ή "συνάφεια επεξεργαστή".
- Η σύνδεση νημάτων με επεξεργαστές μπορεί να οδηγήσει σε καλύτερη χρήση της κρυφής μνήμης, με αυτόν τον τρόπο μειώνοντας τις δαπανηρές προσβάσεις στην μνήμη. Αυτό είναι το πρωταρχικό κίνητρο για σύνδεση νημάτων στους επεξεργαστές.
- Ανάλογα με την πλατφόρμα σας, το λειτουργικό σύστημα, τον μεταγλωττιστή και την υλοποίηση του OpenMP, η σύνδεση νημάτων με τον επεξεργαστή μπορεί να γίνει με πολλούς διαφορετικούς τρόπους
  Η έκδοση 3.1 της προγραμματιστικής διεπαφής εφαρμογής του OpenMP προσφέρει μια μεταβλητή περιβάλλοντος που μπορεί να ενεργοποιεί ή να απενεργοποιεί την σύνδεση του επεξεργαστή. Για παράδειγμα:
  `setenv OMP_PROC_BIND TRUE`
  `setenv OMP_PROC_BIND FALSE`
- Σε μεγαλύτερο επίπεδο, διεργασίες μπορούν επίσης να συνδεθούν με επεξεργαστές.
- Αναλυτικές πληροφορίες για συνδέσεις διεργασιών και νημάτων σε επεξεργαστές στις συστοιχίες Linux της LC μπορείτε να βρείτε ΕΔΏ.

Παρακολούθηση, Αποσφαλμάτωση και Εργαλεία Ανάλυσης Απόδοσης για το OpenMP

- **Παρακολούθηση και Αποσφαλμάτωση Νημάτων:**

  - Οι αποσφαλματωτές διαφέρουν στην ικανότητα να χειρίζονται τα νήματα. Ο αποσφαλματωτής TotalView είναι ο αποσφαλματωτής που συνιστάται από την LC για τα παράλληλα προγράμματα. Είναι κατάλληλος για παρακολούθηση αλλά και για αποσφαλμάτωση προγραμμάτων με νήματα.
  - Ένα στιγμιότυπο παραδείγματος από την χρήση του TotalView με ένα κώδικα σε OpenMP φαίνεται παρακάτω:
    1. Παράθυρο Ίχνους Στοίβας Κυρίως Νήματος: δείχνει την αρχική ρουτίνα
    2. Μπάρα κατάστασης Διεργασίας/νήματος διαφοροποιώντας τα νήματα
    3. Παράθυρο Frame Στοίβας Κυρίως Νήματος: δείχνει τις κοινές μεταβλητές
    4. Παράθυρο Ίχνους Στοίβας Εργάτη Νήματος: δείχνει την σκιαγραφημένη ρουτίνα
    5. Παράθυρο Frame Στοίβας Εργάτη Νήματος
    6. Αρχικό Παράθυρο showing all threads
    7. Παράθυρο Νημάτων δείχνοντας όλα τα νήματα και τα επιλεγμένα



- Ανατρέξτε στον Οδηγό Εκμάθησης Αποσφαλματωτή TotalView για περισσότερες λεπτομέρειες.
- Η εντολή των Linux ps προσφέρει αρκετές σημαίες για την εμφάνιση πληροφοριών νημάτων. Μερικά παραδείγματα φαίνονται παρακάτω. Κοιτάξτε την σελίδα βοήθειας για περισσότερες λεπτομέρειες.

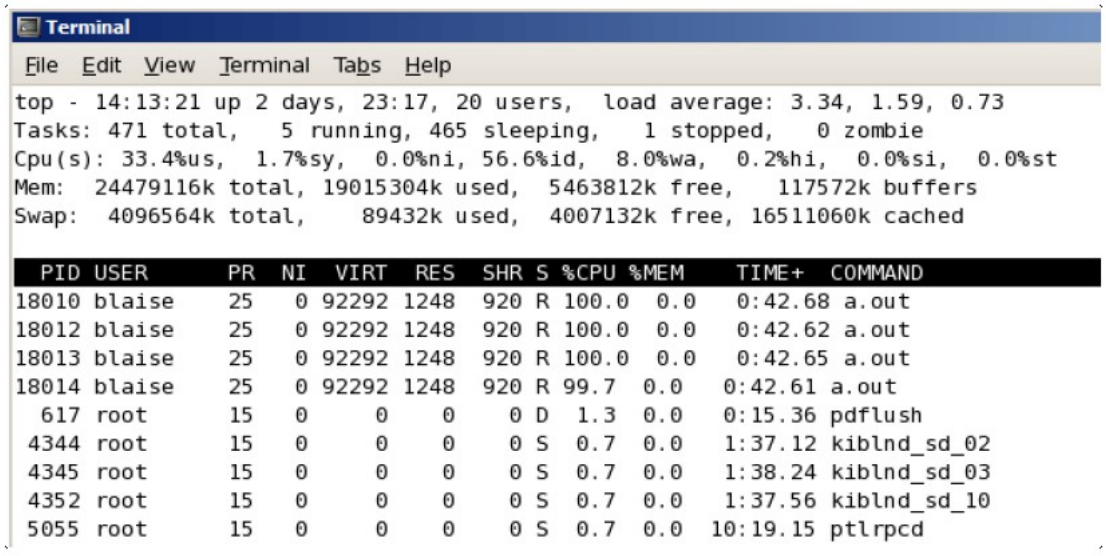
```
% ps -Lf
UID        PID  PPID   LWP  C  NLWP STIME TTY      TIME     CMD
```

```
blaise 22529 28240 22529  0    5  11:31 pts/53 00:00:00 a.out
blaise 22529 28240 22530 99    5  11:31 pts/53 00:01:24 a.out
blaise 22529 28240 22531 99    5  11:31 pts/53 00:01:24 a.out
blaise 22529 28240 22532 99    5  11:31 pts/53 00:01:24 a.out
blaise 22529 28240 22533 99    5  11:31 pts/53 00:01:24 a.out

% ps -T
PID    SPID  TTY     TIME    CMD
22529 22529 pts/53 00:00:00 a.out
22529 22530 pts/53 00:01:49 a.out
22529 22531 pts/53 00:01:49 a.out
22529 22532 pts/53 00:01:49 a.out
22529 22533 pts/53 00:01:49 a.out

% ps -Lm
PID       LWP  TTY      TIME     CMD
22529         - pts/53 00:18:56 a.out
       - 22529 -       00:00:00 -
       - 22530 -       00:04:44 -
       - 22531 -       00:04:44 -
       - 22532 -       00:04:44 -
       - 22533 -       00:04:44 -
```

- Οι συστοιχίες Linux της LC προσφέρουν την εντολή `top` για να παρακολουθούνται οι διεργασίες σε έναν κόμβο. Αν χρησιμοποιηθεί με την σημαία `-H` τα νήματα που περιέχονται σε μία διεργασία θα είναι ορατά. Ένα παράδειγμα της χρήσης της εντολής `top -H` φαίνεται παρακάτω. Η πατρική διεργασία έχει PID 18010 η οποία δημιούργησε τρία νήματα, τα οποία φαίνονται με τα PID 18012, 18013, 18014.



- **Εργαλεία Ανάλυσης Απόδοσης:**
  - Υπάρχει ποικιλία εργαλείων ανάλυσης απόδοσης που μπορούν να χρησιμοποιηθούν με τα προγράμματα του OpenMP. Μια αναζήτηση στον ιστό θα παρέχει μια πληθώρα πληροφοριών.
  - Στην LC , η λίστα με τα υποστηριζόμενα υπολογιστικά εργαλεία μπορεί να βρεθεί στην διεύθυνση: computing.llnl.gov/code/content/software_tools.php.
  - Αυτά τα εργαλεία διαφέρουν σημαντικά στην πολυπλοκότητα, την λειτουργικότητα και την εκμάθηση. Η κάλυψη αυτών με λεπτομέρειες είναι πέρα από το πεδίο αυτού του οδηγού εκμάθησης.
  - Μερικά εργαλεία που πρέπει να κοιτάξετε, ειδικά για κώδικες με OpenMP, είναι:
    - Open|SpeedShop
    - TAU
    - PAPI
    - Intel VTune Amplifier
    - ThreadSpotter

Άσκηση OpenMP 3

Ποικιλία

Επισκόπηση:

- Συνδεθείτε σε μία συστοιχία(cluster) της LC, αν δεν έχετε συνδεθεί ήδη.
- Παραδείγματα ορφανής οδηγίας: επανεξετάστε, μεταγλωττίστε και εκτελέστε τα παραδείγματα.
- Αποκτήστε πληροφορίες για την υλοποίηση περιβάλλοντος του OpenMP.
- Κοιτάξτε τα προγράμματα με σφάλματα.

Πηγαίνετε στην άσκηση.

Αυτό ολοκληρώνει αυτό τον οδηγό εκμάθησης.

Παρακαλώ συμπληρώστε την online αξιολόγηση - εκτός αν κάνετε την άσκηση, όπου παρακαλούμε συμπληρώστε την στο τέλος της άσκησης.

Που θέλετε να πάτε τώρα

- Άσκηση
- Ατζέντα

Αναφορές και περισσότερες πληροφορίες

- Συγγραφέας: Blaise Barney, Livermore Computing.
- Η ιστοσελίδα του OpenMP, η οποία περιέχει τα έγγραφα για της Προγραμματιστικής Διεπαφής Εφαρμογής για C/C++ και Fortran http://openmp.org/wp/

Παράρτημα Α: Ρουτίνες Βιβλιοθήκης Εκτέλεσης

OMP_SET_NUM_THREADS

- **Σκοπός:**
  - Καθορίζει τον αριθμό των νημάτων που θα χρησιμοποιηθούν στο επόμενο παράλληλο τμήμα.
- **Μορφή:**

Fortran	`SUBROUTINE OMP_SET_NUM_THREADS(scalar_integer_expression)`
C/C++	`#include <omp.h>`

	`void omp_set_num_threads(int num_threads)`

- **Σημειώσεις και Περιορισμοί:**
  - Ο μηχανισμός δυναμικών νημάτων τροποποιεί την δράση αυτής της ρουτίνας
    - Ενεργοποιημένος: Ορίζει τον μέγιστο αριθμό νημάτων που μπορούν να χρησιμοποιηθούν για οποιοδήποτε παράλληλο τμήμα από τον μηχανισμό δυναμικών νημάτων.
    - Απενεργοποιημένος: Ορίζει τον ακριβή αριθμό νημάτων που θα χρησιμοποιηθεί μέχρι την επόμενη κλήση αυτής της ρουτίνας.
  - Αυτή η ρουτίνα μπορεί να καλεστεί μόνο από σειριακά τμήματα του κώδικα
  - Αυτή η κλήση έχει προτεραιότητα έναντι της μεταβλητής περιβάλλοντος OMP_NUM_THREADS.

OMP_GET_NUM_THREADS

- **Σκοπός:**
  - Επιστρέφει τον αριθμό των νημάτων που βρίσκονται αυτή τη στιγμή στην ομάδα που εκτελεί το παράλληλο τμήμα από το οποίο καλέστηκε.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_NUM_THREADS()`
C/C++	`#include <omp.h>` `int omp_get_num_threads(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Αν αυτή η κλήση γίνει από ένα σειριακό τμήμα του προγράμματος ή ένα εμφωλευμένο παράλληλο τμήμα που σειριοποιημένο, θα επιστρέψει 1
  - Ο προεπιλεγμένος αριθμός νημάτων εξαρτάται από τη υλοποίηση.

OMP_GET_MAX_THREADS

- **Σκοπός:**
  - Επιστρέφει τη μέγιστη τιμή η οποία μπορεί να επιστραφεί από την κλήση της συνάρτησης OMP_GET_NUM_THREADS.
- **Μορφή:**

Fortra	`INTEGER FUNCTION OMP_GET_MAX_THREADS()`

n	
C/C++	`#include <omp.h>` `int omp_get_max_threads(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Γενικά αντανακλά τον αριθμό των νημάτων όπως έχει τεθεί από την μεταβλητή περιβάλλοντος OMP_NUM_THREADS ή από την ρουτίνα βιβλιοθήκης OMP_SET_NUM_THREADS().
  - Μπορεί να καλεστεί και από το σειριακό αλλά και από το παράλληλο τμήμα κώδικα.

OMP_GET_THREAD_NUM

- **Σκοπός:**
  - Επιστρέφει τον αριθμό του νήματος από το νήμα, μέσα από την ομάδα, που έκανε αυτήν την κλήση.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_THREAD_NUM()`
C/C++	`#include <omp.h>` `int omp_get_thread_num(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Αν καλείται από ένα εμφωλευμένο παράλληλο τμήμα ή ένα σειριακό τμήμα, η συνάρτηση επιστρέφει 0.
- **Παραδείγματα:**
  - Το παράδειγμα 1 είναι ο σωστός τρόπος να καθοριστεί ο αριθμός των νημάτων σε ένα παράλληλο τμήμα.
  - Το παράδειγμα 2 είναι λάθος - η μεταβλητή TID πρέπει να είναι PRIVATE.
  - Το παράδειγμα 3 είναι λάθος - η κλήση της OMP_GET_THREAD_NUM είναι έξω από το παράλληλο τμήμα.

Fortran - προσδιορισμός του αριθμού των νημάτων σε ένα παράλληλο τμήμα

Παράδειγμα 1: Σωστό

```
PROGRAM HELLO
INTEGER TID, OMP_GET_THREAD_NUM
!$OMP PARALLEL PRIVATE(TID)
TID = OMP_GET_THREAD_NUM()
PRINT *, 'Hello World from thread = ', TID
...
```

```
!$OMP END PARALLEL
END
```

Παράδειγμα 2: Λάθος

```
PROGRAM HELLO
INTEGER TID, OMP_GET_THREAD_NUM
!$OMP PARALLEL
TID = OMP_GET_THREAD_NUM()
PRINT *, 'Hello World from thread = ', TID
...
!$OMP END PARALLEL
END
```

Παράδειγμα 3: Λάθος

```
PROGRAM HELLO
INTEGER TID, OMP_GET_THREAD_NUM
TID = OMP_GET_THREAD_NUM()
PRINT *, 'Hello World from thread = ', TID
!$OMP PARALLEL
...
!$OMP END PARALLEL
END
```

OMP_GET_THREAD_LIMIT

- **Σκοπός:**
  - Επιστρέφει το μέγιστο αριθμό των νημάτων του OpenMP που είναι διαθέσιμα σε ένα πρόγραμμα.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_THREAD_LIMIT`
C/C++	`#include <omp.h>` `int omp_get_thread_limit (void)`

- **Σημειώσεις:**
  - Κοιτάξτε επίσης την μεταβλητή περιβάλλοντος `OMP_THREAD_LIMIT`.

OMP_GET_NUM_PROCS

- **Σκοπός:**
  - Επιστέφει τον μέγιστο αριθμό των επεξεργαστών που είναι διαθέσιμοι σε ένα πρόγραμμα.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_NUM_PROCS()`
C/C++	`#include <omp.h>` `int omp_get_num_procs(void)`

OMP_IN_PARALLEL

- **Σκοπός:**
  - Χρησιμοποιείται για να αποφασίσει εάν το τμήμα του κώδικα που εκτελείται είναι παράλληλο ή όχι.
- **Μορφή:**

Fortran	`LOGICAL FUNCTION OMP_IN_PARALLEL()`
C/C++	`#include <omp.h>` `int omp_in_parallel(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Στην Fortran, αυτή η συνάρτηση επιστρέφει .TRUE. αν έχει καλεστεί μέσα από την δυναμική έκταση μιας περιοχής που εκτελείται παράλληλα διαφορετικά .FALSE. Στην C/C++ θα επιστρέψει μη-μηδενικό ακέραιο σε παράλληλα τμήματα διαφορετικά μηδέν.

OMP_SET_DYNAMIC

- **Σκοπός:**
  - Ενεργοποιεί ή απενεργοποιεί την δυναμική προσαρμογή(από το σύστημα εκτέλεσης) του αριθμού των νημάτων που χρειάζονται για την εκτέλεση του παράλληλου τμήματος.
- **Μορφή:**

Fortran	`SUBROUTINE OMP_SET_DYNAMIC(scalar_logical_expression)`
C/C++	`#include <omp.h>` `void omp_set_dynamic(int dynamic_threads)`

- **Σημειώσεις και Περιορισμοί:**
  - Στην Fortran, αν καλεστεί με το όρισμα .TRUE. τότε ο αριθμός των νημάτων που είναι διαθέσιμα για τα ακόλουθα παράλληλα τμήματα μπορεί να τροποποιηθεί αυτόματα από το περιβάλλον εκτέλεσης. Αν καλεστεί με .FALSE. τότε η δυναμική τροποποίηση απενεργοποιείται.
  - Στην C/C++, αν το όρισμα dynamic_threads αποτιμάται σε μη-μηδενικό, τότε ο μηχανισμός ενεργοποιείται, διαφορετικά απενεργοποιείται.
  - Η υπορουτίνα OMP_SET_DYNAMIC έχει προτεραιότητα έναντι της μεταβλητής περιβάλλοντος OMP_DYNAMIC.
  - Η προεπιλεγμένη ρύθμιση εξαρτάται από την υλοποίηση.
  - Πρέπει να καλείται από ένα σειριακό τμήμα κώδικα.

OMP_GET_DYNAMIC

- **Σκοπός:**
  - Χρησιμοποιείται για να καθορίσει αν η δυναμική προσαρμογή νημάτων είναι ενεργοποιημένη ή όχι.
- **Μορφή:**

Fortran	`LOGICAL FUNCTION OMP_GET_DYNAMIC()`
C/C++	`#include <omp.h>` `int omp_get_dynamic(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Στην Fortran, αυτή η συνάρτηση επιστρέφει .TRUE. αν η δυναμική τροποποίηση νημάτων είναι ενεργοποιημένη, διαφορετικά επιστρέφει .FALSE.
  - Στην C/C++, ένας μη-μηδενικός αριθμός θα επιστραφεί αν η δυναμική τροποποίηση νημάτων είναι ενεργοποιημένη, διαφορετικά θα επιστραφεί μηδέν.

OMP_SET_SCHEDULE

- **Σκοπός:**
  - Νέο χαρακτηριστικό με το OpenMP 3.0.
  - Αυτή η ρουτίνα θέτει τον τύπο προγραμματισμού που εφαρμόζεται όταν η οδηγία του βρόγχου ορίζει ένα πρόγραμμα εκτέλεσης.

- **Μορφή:**

Fortran	`SUBROUTINE OMP_SET_SCHEDULE(KIND, MODIFIER)` `INTEGER (KIND=OMP_SCHED_KIND) KIND` `INTEGER MODIFIER`
C/C++	`#include <omp.h>` `void omp_set_schedule(omp_sched_t kind, int modifier)`

OMP_GET_ SCHEDULE

- **Σκοπός:**
  - Νέο χαρακτηριστικό με το OpenMP 3.0.
  - Αυτή η ρουτίνα επιστρέφει τον τύπο προγραμματισμού που εφαρμόζεται όταν η οδηγία του βρόγχου ορίζει ένα πρόγραμμα εκτέλεσης.
- **Μορφή:**

Fortran	`SUBROUTINE OMP_GET_SCHEDULE(KIND, MODIFIER)` `INTEGER (KIND=OMP_SCHED_KIND) KIND` `INTEGER MODIFIER`
C/C++	`#include <omp.h>` `void omp_get_schedule(omp_sched_t * kind, int * modifier )`

- **Σημειώσεις και Περιορισμοί:**
  - Στην Fortran, αυτή η συνάρτηση επιστρέφει .TRUE. αν η εμφωλευμένη παραλληλοποίηση είναι ενεργοποιημένη, διαφορετικά επιστρέφει .FALSE.
  - Στην C/C++, ένας μη-μηδενικός αριθμός θα επιστραφεί αν η εμφωλευμένη παραλληλοποίηση είναι ενεργοποιημένη, διαφορετικά θα επιστραφεί μηδέν.

OMP_SET_MAX_ACTIVE_LEVELS

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Αυτή η ρουτίνα περιορίζει τον αριθμό των ενεργών εμφωλευμένων παράλληλων τμημάτων.
- **Μορφή:**

Fortran	`SUBROUTINE OMP_SET_MAX_ACTIVE_LEVELS (MAX_LEVELS)` `INTEGER MAX_LEVELS`
C/C++	`#include <omp.h>` `void omp_set_max_active_levels (int max_levels)`

- **Σημειώσεις και Περιορισμοί:**

- Αν ο αριθμός των βαθμών παραλληλοποίησης ξεπεράσει τον αριθμό των βαθμών παραλληλοποίησης που υποστηρίζει η υλοποίηση, η τιμή θα τεθεί στον αριθμό των βαθμών παραλληλοποίησης που υποστηρίζει η υλοποίηση.
- Η ρουτίνα έχει την περιγραφόμενη δράση μόνο όταν καλείται από το ακολουθιακό κομμάτι του κώδικα. Όταν καλείται μέσα από ένα ρητά παράλληλο τμήμα, η δράση της ορίζεται από την υλοποίηση.

OMP_GET_ MAX_ACTIVE_LEVELS

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Αυτή η ρουτίνα επιστρέφει τον τύπο προγραμματισμού που εφαρμόζεται όταν η οδηγία του βρόγχου ορίζει ένα πρόγραμμα εκτέλεσης.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_MAX_ACTIVE_LEVELS()`
C/C++	`#include <omp.h>` `int omp_get_max_active_levels(void)`

OMP_GET_ LEVEL

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Αυτή η ρουτίνα επιστρέφει το νούμερο των εμφωλευμένων παράλληλων τμημάτων που περικλείουν την εργασία που περιέχει την κλήση.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_LEVEL()`
C/C++	`#include <omp.h>` `int omp_get_level(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Η ρουτίνα omp_get_level επιστρέφει το νούμερο των εμφωλευμένων παράλληλων τμημάτων (ενεργά ή ανενεργά) που περικλείουν την εργασία που περιέχει την κλήση χωρίς να υπολογίζει το υπονοούμενο παράλληλο τμήμα. Αυτή η ρουτίνα πάντα επιστρέφει έναν μη αρνητικό ακέραιο και επιστρέφει 0 αν καλείται από το ακολουθιακό μέρος του προγράμματος.

OMP_GET_ANCESTOR_THREAD_NUM

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Αυτή η ρουτίνα επιστρέφει για έναν δοσμένο αριθμό εμφωλευμένων παράλληλων τμημάτων του τρέχοντος νήματος, τον αριθμό νήματος του προγόνου του τρέχοντος νήματος.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_ANCESTOR_THREAD_NUM(LEVEL)` `INTEGER LEVEL`
C/C++	`#include <omp.h>` `int omp_get_ancestor_thread_num(int level)`

- **Σημειώσεις και Περιορισμοί:**
  - Αν ο ζητούμενος βαθμός φωλιάσματος είναι εκτός του εύρους του 0 και του βαθμού φωλιάσματος του τρέχοντος νήματος, θα επιστραφεί -1, όπως γίνεται και με την ρουτίνα omp_get_level.

OMP_GET_TEAM_SIZE

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Αυτή η ρουτίνα επιστρέφει για έναν δοσμένο αριθμό εμφωλευμένων παράλληλων τμημάτων του τρέχοντος νήματος, το μέγεθος της ομάδας νημάτων στην οποία ανήκει το τρέχον νήμα ή ο πρόγονος του.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_TEAM_SIZE(LEVEL)` `INTEGER LEVEL`
C/C++	`#include <omp.h>` `int omp_get_team_size(int level);`

- **Σημειώσεις και Περιορισμοί:**
  - Αν ο ζητούμενος βαθμός φωλιάσματος είναι εκτός του εύρους του 0 και του βαθμού φωλιάσματος του τρέχοντος νήματος, θα επιστραφεί -1, όπως γίνεται και με την ρουτίνα omp_get_level. Ανενεργά παράλληλα τμήματα θεωρούνται σαν ενεργά παράλληλα τμήματα που εκτελούνται με ένα νήμα.

OMP_GET_ACTIVE_LEVEL

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Η ρουτίνα omp_get_active_level επιστρέφει τον αριθμό των εμφωλευμένων, παράλληλων τμημάτων που περικλείουν την εργασία που περιέχει την κλήση.
- **Μορφή:**

Fortran	`INTEGER FUNCTION OMP_GET_ACTIVE_LEVEL()`
C/C++	`#include <omp.h>` `int omp_get_active_level(void);`

- **Σημειώσεις και Περιορισμοί:**
  - Η ρουτίνα πάντα επιστρέφει έναν μη αρνητικό ακέραιο και επιστρέφει 0 αν καλεστεί από ακολουθιακό τμήμα του προγράμματος.

OMP_IN_FINAL

- **Σκοπός:**
  - Αυτή η ρουτίνα είναι καινούργια στην έκδοση OpenMP 3.0.
  - Η ρουτίνα επιστρέφει αληθές(true) αν εκτελεστεί στην τελική περιοχή της εργασίας αλλιώς επιστρέφει ψευδές(false).
- **Μορφή:**

Fortran	`LOGICAL FUNCTION OMP_IN_FINAL()`
C/C++	`#include <omp.h>` `int omp_in_final(void)`

- **Σημειώσεις και Περιορισμοί:**
  - Η ρουτίνα πάντα επιστρέφει έναν μη αρνητικό ακέραιο και επιστρέφει 0 αν καλεστεί από ακολουθιακό τμήμα του προγράμματος.

OMP_INIT_LOCK
OMP_INIT_NEST_LOCK

- **Σκοπός:**
  - Αυτή η υπορουτίνα αρχικοποιεί μια κλειδαριά που σχετίζεται με μια μεταβλητή lock.

- Η ρουτίνα φωλιάσματος είναι καινούργια στην έκδοση OpenMP 3.0.

➢ Μορφή:

Fortran	`SUBROUTINE OMP_INIT_LOCK(var)` `SUBROUTINE OMP_INIT_NEST_LOCK(var)`
C/C++	`#include <omp.h>` `void omp_init_lock(omp_lock_t *lock)` `void omp_init_nest_lock(omp_nest_lock_t *lock)`

➢ Σημειώσεις και Περιορισμοί:

- Η αρχική κατάσταση είναι ξεκλείδωτη.
- Για την Fortran, η μεταβλητή *var* πρέπει να είναι ένας ακέραιος αρκετά μεγάλος για να μπορεί να κρατήσει μια διεύθυνση, όπως ένας INTEGER** στα συστήματα 64 bit.

OMP_DESTROY_LOCK
OMP_DESTROY_NEST_LOCK

➢ Σκοπός:

- Αυτή η υπορουτίνα αποσυνδέει την δοσμένη μεταβλητή lock από οποιαδήποτε κλειδαριά.
- Η ρουτίνα φωλιάσματος είναι καινούργια στην έκδοση OpenMP 3.0.

➢ Μορφή:

Fortran	`SUBROUTINE OMP_DESTROY_LOCK(var)` `SUBROUTINE OMP_DESTROY_NEST_LOCK(var)`
C/C++	`#include <omp.h>` `void omp_destroy_lock(omp_lock_t *lock)` `void omp_destroy_nest_lock(omp_nest_lock_t *lock)`

➢ Σημειώσεις και Περιορισμοί:

- Είναι παράνομο να καλεστεί αυτή η ρουτίνα με μια μεταβλητή lock που δεν έχει αρχικοποιηθεί.
- Για την Fortran, η μεταβλητή *var* πρέπει να είναι ένας ακέραιος αρκετά μεγάλος για να μπορεί να κρατήσει μια διεύθυνση, όπως ένας INTEGER** στα συστήματα 64 bit.

OMP_SET_LOCK
OMP_SET_NEST_LOCK

➢ Σκοπός:

- Αυτή η υπορουτίνα αναγκάζει το νήμα εκτέλεσης να περιμένει μέχρι η ορισμένη κλειδαριά να είναι διαθέσιμη. Το νήμα αποκτά την κυριότητα της κλειδαριάς όταν η κλειδαριά γίνεται διαθέσιμη.
- Η ρουτίνα φωλιάσματος είναι καινούργια στην έκδοση OpenMP 3.0.

➢ Μορφή:

Fortran	`SUBROUTINE OMP_SET_LOCK(var)` `SUBROUTINE OMP_SET_NEST_LOCK(var)`
C/C++	`#include <omp.h>` `void omp_set_lock(omp_lock_t *lock)` `void omp_set_nest__lock(omp_nest_lock_t *lock)`

➢ Σημειώσεις και Περιορισμοί:

- Είναι παράνομο να καλεστεί αυτή η ρουτίνα με μια μεταβλητή lock που δεν έχει αρχικοποιηθεί.
- Για την Fortran, η μεταβλητή *var* πρέπει να είναι ένας ακέραιος αρκετά μεγάλος για να μπορεί να κρατήσει μια διεύθυνση, όπως ένας INTEGER** στα συστήματα 64 bit.

OMP_UNSET_LOCK
OMP_UNSET_NEST_LOCK

➢ Σκοπός:

- Αυτή η υπορουτίνα απελευθερώνει την κλειδαριά από την ρουτίνα εκτέλεσης.
- Η ρουτίνα φωλιάσματος είναι καινούργια στην έκδοση OpenMP 3.0.

➢ Μορφή:

Fortran	`SUBROUTINE OMP_UNSET_LOCK(var)` `SUBROUTINE OMP_UNSET_NEST_LOCK(var)`
C/C++	`#include <omp.h>` `void omp_unset_lock(omp_lock_t *lock)` `void omp_unset_nest__lock(omp_nest_lock_t *lock)`

➢ Σημειώσεις και Περιορισμοί:

- Είναι παράνομο να καλεστεί αυτή η ρουτίνα με μια μεταβλητή lock που δεν έχει αρχικοποιηθεί.
- Για την Fortran, η μεταβλητή *var* πρέπει να είναι ένας ακέραιος αρκετά μεγάλος για να μπορεί να κρατήσει μια διεύθυνση, όπως ένας INTEGER** στα συστήματα 64 bit.

OMP_TEST_LOCK
OMP_TEST_NEST_LOCK

- **Σκοπός:**
  - Αυτή η υπορουτίνα επιχειρεί να ορίσει ένα κλείδωμα, αλλά δεν μπλοκάρει αν η κλειδαριά δεν είναι διαθέσιμη.
  - Η ρουτίνα φωλιάσματος είναι καινούργια στην έκδοση OpenMP 3.0.
- **Μορφή:**

Fortran	`SUBROUTINE OMP_TEST_LOCK(var)` `SUBROUTINE OMP_TEST_NEST_LOCK(var)`
C/C++	`#include <omp.h>` `int omp_test_lock(omp_lock_t *lock)` `int omp_test_nest__lock(omp_nest_lock_t *lock)`

- **Σημειώσεις και Περιορισμοί:**
  - Για την Fortran, επιστρέφεται .TRUE. αν το κλείδωμα ήταν επιτυχές, διαφορετικά επιστρέφεται .FALSE.
  - Για την Fortran, η μεταβλητή *var* πρέπει να είναι ένας ακέραιος αρκετά μεγάλος για να μπορεί να κρατήσει μια διεύθυνση, όπως ένας INTEGER** στα συστήματα 64 bit.
  - Για την C/C++, επιστρέφεται ένας μη μηδενικός αριθμός αν το κλείδωμα ήταν επιτυχές, διαφορετικά επιστρέφεται μηδέν.
  - Είναι παράνομο να καλεστεί αυτή η ρουτίνα με μια μεταβλητή lock που δεν έχει αρχικοποιηθεί.

OMP_GET_WTIME

- **Σκοπός:**
  - Παρέχει μια φορητή ρουτίνα ρολογιού.
  - Επιστρέφει μια τιμή δεκαδικού διπλής ακρίβειας ίση με τον αριθμό των δευτερολέπτων από κάποια στιγμή στο παρελθόν. Συνήθως χρησιμοποιείται σε "ζευγάρι", με την τιμή από την πρώτη κλήση να αφαιρείται από την τιμή της δεύτερης κλήσης για να αποκτηθεί ο χρόνος που έχει παρέλθει για ένα μπλοκ κώδικα.
  - Σχεδιασμένη για να χρησιμοποιείται κατά νήμα, και για αυτό μπορεί να μην είναι καθολικά συνεπής μεταξύ όλων των νημάτων σε μια ομάδα - εξαρτάται από το τι κάνει το νήμα σε σύγκριση με τα άλλα νήματα.
- **Μορφή:**

Fortran	`DOUBLE PRECISION FUNCTION OMP_GET_WTIME()`
C/C++	`#include <omp.h>` `double omp_get_wtime(void)`

OMP_GET_WTICK

- **Σκοπός:**
  - Παρέχει μια φορητή ρουτίνα ρολογιού.
  - Επιστρέφει έναν δεκαδικό αριθμό διπλής ακρίβειας ίσο με τον αριθμό των δευτερολέπτων μεταξύ δύο χτύπων του ρολογιού.
- **Μορφή:**

Fortran	`DOUBLE PRECISION FUNCTION OMP_GET_WTICK()`
C/C++	`#include <omp.h>` `double omp_get_wtick(void)`