Συστήματα Παράλληλης & Κατανεμημένης Επεξεργασίας

Ενότητα: Εισαγωγή στο OpenMP

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.icte.uowm.gr/mdasyg

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός της Ενότητας

- Η παραλληλοποίηση υπολογιστών κοινής μνήμης με το OpenMP.

Περιεχόμενα

- Τι είναι το OpenMP.
- Ιστορικό.
- Στόχοι του OpenMP.
- Παραλληλοποίηση.
- Μοντέλο Μνήμης.
- Σύνταξη.
- Μοντέλο εκτέλεσης.
- Directive parallel.
- Διαμοιρασμός Εργασίας.
- Directives.
- Clauses.
- Ρουτίνες Συστήματος Εκτέλεσης.
- Μεταβλητές Περιβάλλοντος.
- OpenMP compilers.
- Βιβλιογραφία.

Τι είναι το OpenMP;

- Το OpenMP είναι μια Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface, API) το οποίο δημιουργήθηκε από κατασκευαστές υλικού και λογισμικού. Τα αρχικά αντιστοιχούν στη φράση "Open Specifications for Multi Processing".

- Το πρότυπο αυτό προσφέρει στον προγραμματιστή ένα σύνολο από οδηγίες, οι οποίες ενσωματώνονται στον κώδικα ενός προγράμματος, και έτσι μπορεί ο μεταγλωττιστής να επιτύχει παραλληλισμό στο πρόγραμμα αυτό.

OpenMP: Βασικά Συστατικά

- Το API αυτό αποτελείται κυρίως από τρία βασικά συστατικά:
  - Οδηγίες προς τον μεταγλωττιστή (compiler directives).
  - Βιβλιοθήκη με συναρτήσεις (runtime routines).
  - Μεταβλητές περιβάλλοντος (environment variables).
  - Το πρότυπο αυτό έχει ορισθεί για τις γλώσσες προγραμματισμού C/C++ και FORTRAN.
  - Ο παραλληλισμός που προσφέρει αφορά συστήματα μοιραζόμενης μνήμης, οπότε δεν μπορεί από μόνο του να εφαρμοστεί σε συστήματα κατανεμημένης μνήμης.

Ιστορικό (1/3)

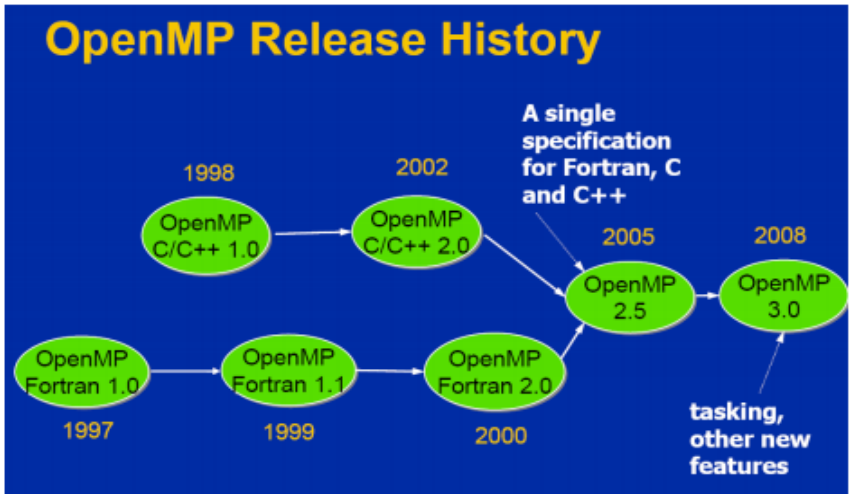
- Η ιστορία του OpenMP ξεκινάει από τις αρχές της δεκαετίας του '90. Εκείνη τη εποχή, οι εταιρίες συστημάτων μοιραζόμενης μνήμης παρείχαν παρόμοιες επεκτάσεις προγραμμάτων Fortran, βασισμένες σε οδηγίες μεταγλωττιστή.
- Ο προγραμματιστής έδινε στον μεταφραστή ένα σειριακό πρόγραμμα Fortran με οδηγίες που καθόριζαν ποιοι βρόχοι θα παραλληλοποιούνταν. Στη συνέχεια, ο μεταφραγλωτιστής ήταν υπεύθυνος για την αυτόματη παραλληλοποίηση αυτών των βρόγχων σε συμμετρικούς επεξεργαστές (SMP's).

Ιστορικό (2/3)

- Αν και οι υλοποιήσεις των διάφορων εταιριών ήταν παρόμοιες ως προς τη λειτουργικότητα, ωστόσο, διέφεραν αρκετά. Έτσι, έγινε μια προσπάθεια για να δημιουργηθεί ένα πρότυπο.
- Το πρώτο πρότυπο που δημιουργήθηκε ήταν το ANSI X3H5 το 1994, αλλά δεν υιοθετήθηκε επειδή εκείνη την εποχή το ενδιαφέρον για αρχιτεκτονικές μοιραζόμενης μνήμης έπεσε, ενώ ανέβηκε το ενδιαφέρον για συστήματα με αρχιτεκτονική κατανεμημένης μνήμης.

Ιστορικό (3/3)

- Στο OpenMP 3.0 ξεκαθαρίστηκαν κάποιες λεπτομέρειες, προστέθηκαν κάποιες νέες συναρτήσεις που μπορούν να κληθούν από τα προγράμματα (π.χ. εύρεση του nesting level).
- **Το βασικότερο όμως ήταν η προσθήκη των tasks!!!**



Στόχοι του openMP

- Οι στόχοι του OpenMP είναι να παράσχει ένα πρότυπο που θα ισχύει για αρχιτεκτονικές και πλατφόρμες μοιραζόμενης μνήμης. Ένα πρότυπο που θα είναι μικρό και εύκολα κατανοητό, δηλαδή θα παρέχει ένα απλό και περιορισμένο σύνολο από οδηγίες με το οποίο θα μπορεί να γίνεται σημαντική παραλληλοποίηση. Επίσης, θα πρέπει να είναι εύκολο στη χρήση του. Το OpenMP παρέχει:
  - Τη δυνατότητα παραλληλοποίησης ενός προγράμματος σταδιακά (εν αντιθέσει, για παράδειγμα, με το MPI όπου κάθε φορά γίνεται include όλη η βιβλιοθήκη).
  - Τη δυνατότητα τόσο αδρομερούς (coarse-grain) όσο και λεπτομερούς (fine-grain) παραλληλισμού.
  - Ο κυριότερος στόχος του OpenMP είναι η μεταφερτότητα. Υποστηρίζει Fortran και C/C++ ενώ έχει υλοποιηθεί για τις πιο πολλές από τις σημαντικές πλατφόρμες όπως Unix/Linux και Windows.

Παραλληλοποίηση (1/2)

- Συνήθως ένας αλγόριθμος παραλληλοποιείται με διάσπασή του σε πολλαπλά τμήματα τα οποία ανατίθενται σε ξεχωριστά νήματα ή διεργασίες και έτσι εκτελούνται παράλληλα σε διαφορετικές επεξεργαστικές μονάδες.
- Η παραλληλοποίηση ενός σειριακού αλγορίθμου, προκειμένου να εκμεταλλευτούμε την αύξηση των υπολογιστικών επιδόσεων που προσφέρει ο παραλληλισμός, διακρίνεται σε τέσσερα βήματα που εκτελούνται διαδοχικά:
  - Τη *διάσπαση* του ολικού υπολογισμού σε επιμέρους *εργασίες*. Εξαρτάται από το πρόβλημα και εναπόκειται στον προγραμματιστή.

Παραλληλοποίηση (2/2)

- Την *ανάθεση* εργασιών σε *οντότητες εκτέλεσης* (διακριτές διεργασίες, νήματα ή ίνες). Μπορεί να είναι είτε στατική, όπου κάθε οντότητα αναλαμβάνει ένα προκαθορισμένο σύνολο εργασιών προς εκτέλεση, ή δυναμική, όπου οι εργασίες ανατίθενται μία-μία κατά τον χρόνο εκτέλεσης· μόλις μία οντότητα ολοκληρώσει την τρέχουσα εργασία της ζητά να της δοθεί η επόμενη. Η δυναμική λύση είναι προτιμότερη σε περίπτωση που οι επεξεργαστές έχουν διαφορετικό φόρτο (π.χ. αν κάποιοι εκτελούν και άλλα προγράμματα) αλλά επισύρει κάποια χρονική επιβάρυνση.
- Την *ενορχήστρωση* των οντοτήτων. Εδώ γίνεται ο καθορισμός του τρόπου συντονισμού και επικοινωνίας μεταξύ των οντοτήτων εκτέλεσης, π.χ. φράγματα εκτέλεσης, αποστολή ή λήψη μηνυμάτων κλπ.
- Την *αντιστοίχηση* οντοτήτων σε επεξεργαστές. Μπορεί να είναι στατική, δηλαδή προκαθορισμένη από τον προγραμματιστή, ή δυναμική, δηλαδή ρυθμιζόμενη από το σύστημα κατά τον χρόνο εκτέλεσης.

Τρόπος Επικοινωνίας (1/2)

- Τα βήματα αυτά είναι πανομοιότυπα για προγράμματα γραμμένα τόσο στο μοντέλο κοινού χώρου διευθύνσεων όσο και στο μοντέλο μεταβίβασης μηνυμάτων· αυτό που αλλάζει στις δύο περιπτώσεις είναι ο τρόπος που γίνεται η επικοινωνία μεταξύ των οντοτήτων εκτέλεσης ώστε να διαμοιραστεί ο υπολογιστικός φόρτος, να συντονιστούν οι υπολογισμοί και να συλλεχθούν στο τέλος τα επιμέρους αποτελέσματα. Ο όρος «εργασία» αναφέρεται σε ένα τμήμα υπολογισμού που μπορεί να εκτελεστεί παράλληλα και ανεξάρτητα με τα υπόλοιπα.
- Κάθε οντότητα εκτέλεσης μπορεί να γίνει αντιληπτή ως ένα σύνολο εργασιών που εκτελούνται στον ίδιο επεξεργαστή σειριακά, ενώ συνήθως ο προγραμματιστής κατασκευάζει τόσες οντότητες όσοι είναι οι διαθέσιμοι επεξεργαστές.

Τρόπος Επικοινωνίας (2/2)

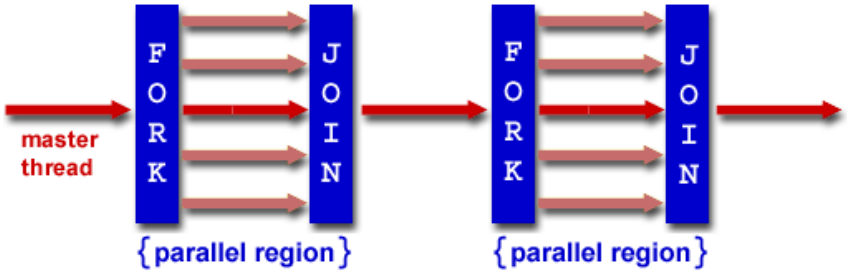
- Η ανάθεση εργασιών σε οντότητες εκτέλεσης είναι πολύ σημαντική γιατί πρέπει να ισοκατανέμεται ο φόρτος μεταξύ των επεξεργαστών, δηλαδή κάθε οντότητα να εκτελεί περίπου ίδιο όγκο υπολογισμών, και σχετιζόμενες εργασίες να ανατίθενται στην ίδια οντότητα ώστε να ελαχιστοποιείται η ανάγκη επικοινωνίας μεταξύ διαφορετικών επεξεργαστών (πολυπολογιστές) ή επεξεργαστών και μνήμης (πολυεπεξεργαστές).
- Η διάσπαση σε εργασίες είναι επίσης σημαντική και μπορεί να γίνει με πολλούς τρόπους, ανάλογα με το εκάστοτε πρόβλημα. Συνήθως υπάρχει μία κύρια οντότητα εκτέλεσης η οποία δημιουργεί τις υπόλοιπες όταν χρειάζεται και στο τέλος συλλέγει τα αποτελέσματα των υπολογισμών τους.

Παραλληλοποίηση και νήματα (1/2)

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

Παραλληλοποίηση και νήματα (2/2)

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



Μοντέλο Μνήμης

- Το μοντέλο μνήμης του OpenMP είναι ένα χαλαρό μοντέλο μοιραζόμενης μνήμης. Όλα τα νήματα έχουν πρόσβαση στη κύρια μνήμη. Κάθε νήμα μπορεί να έχει μια προσωρινή άποψη (temporary view) της μνήμης. Με αυτόν τον τρόπο αποφεύγεται η συνεχής προσπέλαση στη μνήμη. Επίσης κάθε νήμα έχει και την ιδιωτική του μνήμη (thread-private memory), όπου τα υπόλοιπα νήματα δεν έχουν πρόσβαση. Επιγραμματικά στο OpenMP ισχύουν τα εξής:
  - Σε μια οδηγία παραλληλισμού μπορεί ορίζονται μοιραζόμενες και ιδιωτικές μεταβλητές. Κάθε αναφορά σε μοιραζόμενη μεταβλητή είναι μία αναφορά στην ίδια την μεταβλητή, ενώ μια αναφορά σε ιδιωτική μεταβλητή είναι μια αναφορά σε ένα τοπικό αντίγραφο στην ιδιωτική μνήμη του νήματος.
  - Για την προσπέλαση των μοιραζόμενων μεταβλητών από διαφορετικά νήματα απαιτείται συγχρονισμός.
  - Σε περίπτωση εμφωλευμένων οδηγιών παραλληλισμού μία μεταβλητή που είναι ιδιωτική στην εξωτερική παράλληλη περιοχή, μπορεί να είναι μοιραζόμενη στην εσωτερική παράλληλη περιοχή, εκτός αν έχει οριστεί στην εσωτερική οδηγία ως ιδιωτική.

Σύνταξη

Οι περισσότερες οδηγίες στο OpenMP είναι οδηγίες προς τον compiler και η σύνταξη τους έχει την ακόλουθη μορφή:

– C/C++

#pragma omp directive-name [clauses ...]

– Fortran

C$OMP directive-name [clauses …]

!$OMP directive-name [clauses…]

*$OMP directive-name [clauses…]

Επομένως είναι δυνατή η αγνόηση τους από τον compiler.

Μοντέλο εκτέλεσης

```
#include <omp.h>
main () {
int var1, var2, var3;
Serial code
...
Beginning of parallel section. Fork a team of threads.
Specify variable scoping.
#pragma omp parallel private(var1, var2) shared(var3)
{
Parallel section executed by all threads
...
All threads join master thread and disband
}
Resume serial code
...
}
```

Directive parallel (1/2)

```
#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
```

Directive parallel (2/2)

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

“Hello world” program (1/3)

Σειριακό πρόγραμμα σε C

```
#include <stdio.h>
int main() {
printf(“Hello world!!”);
}
```

- Το μεταγλωτίζουμε και δημιουργούμε το εκτελέσιμο.
- Το εκτελούμε και παρατηρούμε ότι το μήνυμα εμφανίζεται μία φορά στην οθόνη.

“Hello world” program (2/3)

Παραλληλοποίηση του “hello world”

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char *argv[]) {
#pragma omp parallel
{
int tid=omp_get_thread_num();
printf("Hello world from thread %d\n", tid);
}
return 0;
}
```

- Το κάνουμε εκτελέσιμο με την εντολή

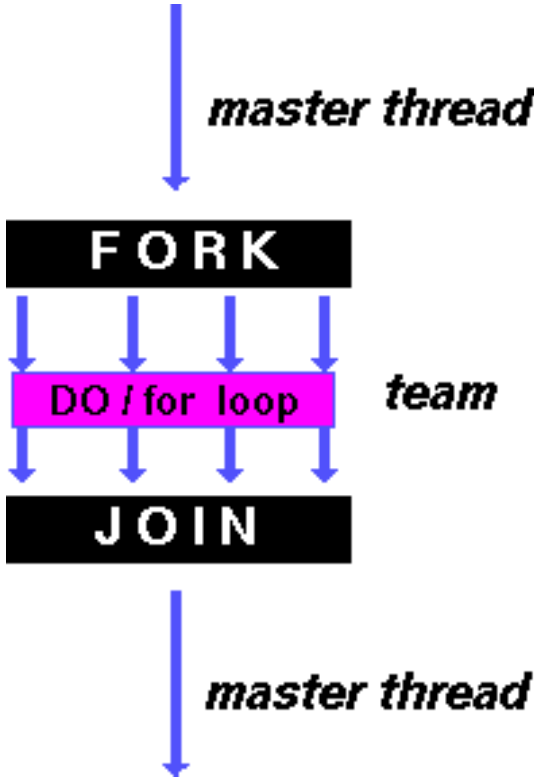
gcc -fopenmp hello.c -o hello.out

“Hello world” program (3/3)

- Εκτελούμε το παράλληλο πρόγραμμα με ./hello.out και βλέπουμε ότι το μήνυμα εμφανίζεται δύο φορές ενώ υπάρχει μία φορά στο κώδικα μας και δεν υπάρχει βρόγχος επανάληψης.
- Φυσικά αν έχουμε μόνο ένα πυρήνα δεν θα δούμε καμία διαφορά.
- Η εντολή #pragma omp parallel αντιλαμβάνεται από το μεταγλωτιστή gcc σαν σχόλιο αν δεν υπάρχει η εντολή –fopenmp κατά τη μεταγλώττιση, πράγμα που επιτρέπει ένα πρόγραμμα που γράφουμε να είναι portable.

Διαμοιρασμός Εργασίας (1/2)

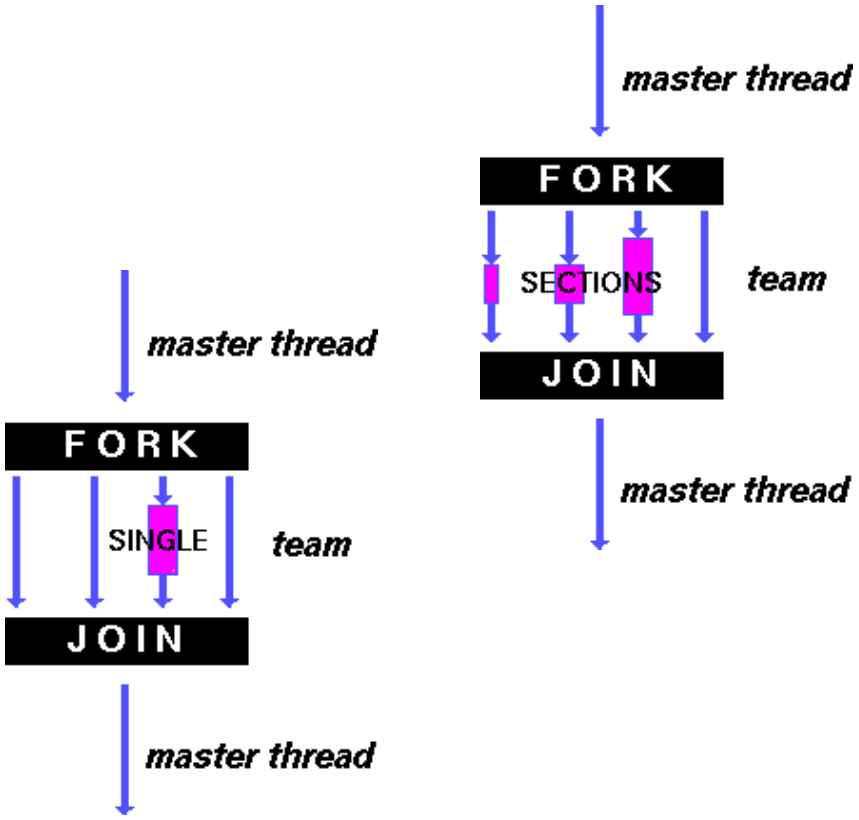
- Μια οδηγία διαμοιρασμού εργασίας κατανέμει την εκτέλεση του κώδικα που περικλείεται στην παράλληλη περιοχή μεταξύ των νημάτων της περιοχής. Οι οδηγίες διαμοιρασμού εργασίας δεν δημιουργούν καινούρια νήματα και δεν υπάρχει κάποιος συγχρονισμός κατά την είσοδο σε μια τέτοια οδηγία, υπάρχει όμως συγχρονισμός κατά την έξοδο. Υπάρχουν τρεις τύποι οδηγιών διαμοιρασμού εργασίας:
  - **FOR:** Διαμοιράζει τις επαναλήψεις ενός βρόχου for στα νήματα της τρέχουσας παράλληλης περιοχής. Αντιστοιχεί στο μοντέλο Παραλληλισμού Δεδομένων (Data Parallelism).



Διαμοιρασμός Εργασίας (2/2)

- **SECTIONS:** Διαμοιράζει την εργασία σε διακριτά δομικά blocks. Κάθε block εκτελείται από ένα νήμα. Αντιστοιχεί στο μοντέλο Συναρτησιακού Παραλληλισμού (Functional Parallelism).

- **SINGLE:** Σειριοποιεί ένα τμήμα του κώδικα ώστε να εκτελεστεί υποχρεωτικά από ένα νήμα.



Directive for

```
#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
```

- Όλα τα νήματα θα εκτελέσουν το ίδιο τμήμα κώδικα, αλλά σε διαφορετικά δεδομένα.

Directive sections

```
#pragma omp sections [clause ...]
 newline private (list)
firstprivate (list)
lastprivate (list)
 reduction (operator: list)
 nowait
```

- Καθορίζει ότι τα εσωκλειώμενα τμήματα κώδικα θα διαμοιραστούν μεταξύ των νημάτων της ομάδας.
- Ακολουθεί πρόγραμμα που δείχνει διαφορετικά blocks να εκτελούνται από διαφορετικά νήματα. Ένα νήμα υλοποιεί πρόσθεση και το άλλο πολλαπλασιασμό.

Παράδειγμα sections

```
void QuickSort (int numList[], int nLower, int nUpper) {
if (nLower < nUpper)
{
// create partitions
int nSplit = Partition (numList,nLower,nUpper);
#pragma omp parallel sections {
#pragma omp section QuickSort(numList,nLower, nSplit - 1);
#pragma omp section QuickSort(numList, nSplit + 1,nUpper);
      }
    }
}
```

Directive Single

```
#pragma omp single [clause ...]
newline private (list)
firstprivate (list)
nowait
structured_block
```

- Η οδηγία single καθορίζει ότι ο κώδικας που εσωκλείεται από αυτή, θα εκτελεστεί μόνο από ένα νήμα. Το νήμα που θα φτάσει πρώτο στην single οδηγία, αυτό θα εκτελέσει τον κώδικα.

Directive Parallel For

#pragma omp parallel for [clause[[,] clause] ...]new-line

for-loop

- Η οδηγία omp parallel for συνδυάζει τις οδηγίες omp parallel και omp for.

Υπολογισμός π (σειριακά)

Σειριακό πρόγραμμα

```
#define N 1000000
double pi = 0.0, W = 1.0/N;
main() {
     int i;
     for (i = 0; i < N; i++)
          pi += 4*W / (1 + (i+0.5)*(i+0.5)*W*W);
          printf("pi = %.10lf\n", pi);
     }
```

Υπολογισμός π (παράλληλα)

Παράλληλο πρόγραμμα

```
#include <omp.h>
#define N 1000000
double pi = 0.0, W = 1.0/N;
main () {
int i;
#pragma omp parallel for reduction(+:pi)
for (i=0; i < N; i++)
{ pi += 4*W / (1 + (i+0.5)*(i+0.5)*W*W); }
//master thread only continues
printf("pi = %.10lf\n", pi);
}
```

Directive Parallel Sections

```
#pragma omp parallel sections
[clause[ [, ]clause] ...]new-line
{
[#pragma omp section new-line]
structured-block
[#pragma omp section new-line
structured-block ] ...
}
```

Directive Task (1/2)

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

Directive Task (2/2)

```
#pragma omp task [clause[ [, ]clause] ...]
new-line structured-block
if(scalar-expression)
untied
default(shared | none)
private(list)
firstprivate(list)
shared(list)
```

- Το thread που συναντά ένα #pragma omp task directive δημιουργεί ένα task με τον κώδικα που περιέχει το structured-block και το βάζει σε ένα task pool.

Task

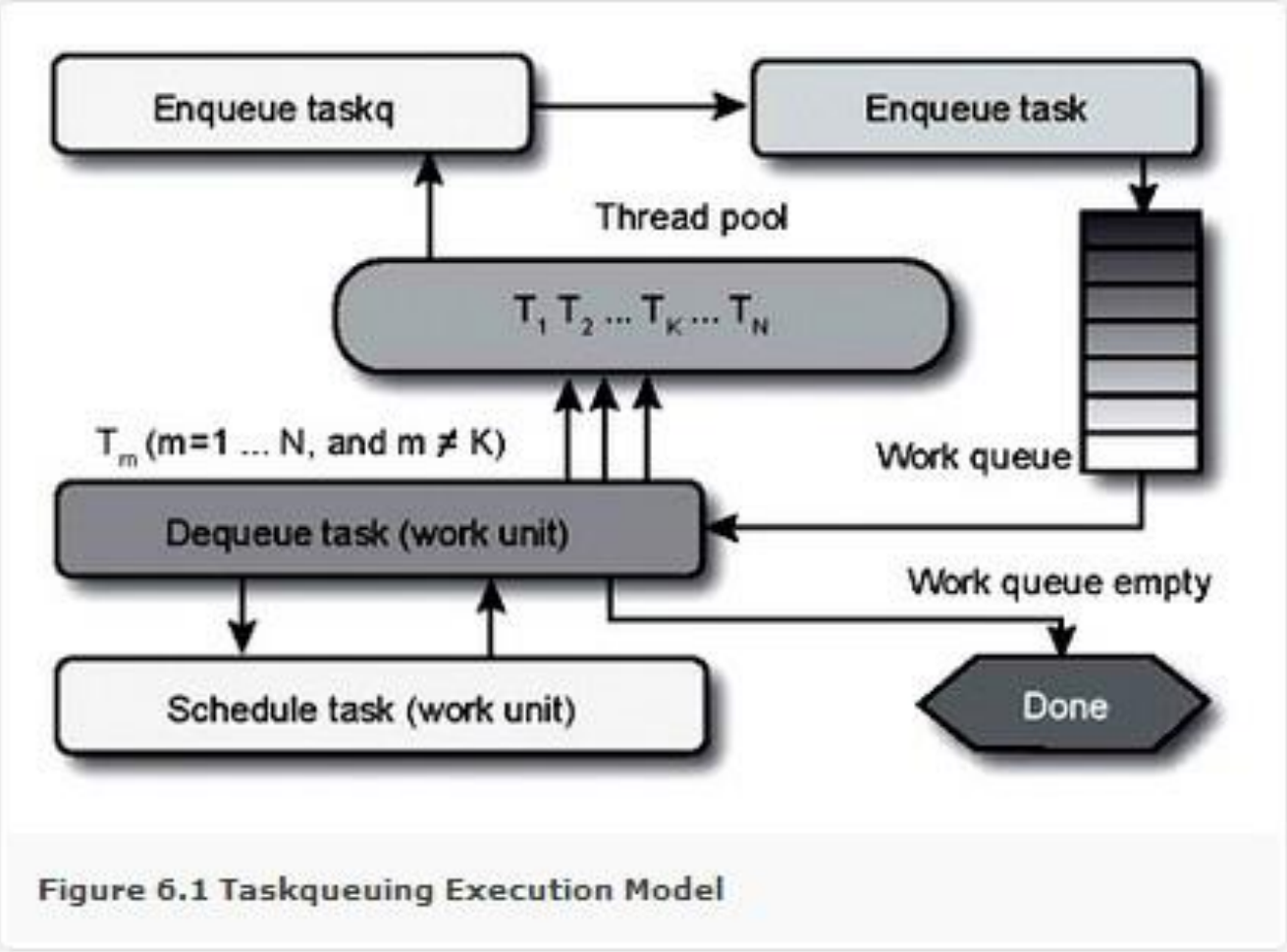


Figure 6.1 Taskqueuing Execution Model

Directives barrier/flush

- Οδηγία BARRIER:

Η οδηγία barrier συγχρονίζει όλα τα νήματα της ομάδας απαιτώντας από κάθε νήμα να σταματήσει, προσωρινά την εκτέλεσή του, στο σημείο όπου υπάρχει η οδηγία barrier οδηγία, μέχρις ότου όλα τα νήματα φτάσουν σε αυτό το σημείο.

#pragma omp barrier *newline*

- Οδηγία FLUSH:

Η οδηγία flush καθορίζει ένα σημείο συγχρονισμού στο οποίο η υλοποίηση του κώδικα πρέπει να παρέχει ένα συνεπές στιγμιότυπο της μνήμης.

#pragma omp flush *(var1, var2, ...) newline*

Directives critical/master

- Οδηγία CRITICAL:

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

#pragma omp critical ***[name] newline***

- Οδηγία MASTER:

Η οδηγία αυτή ορίζει ένα τμήμα κώδικα το οποίο θα εκτελεστεί από το master thread μόνο.

#pragma omp master ***newline***

Παράδειγμα barrier/master

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

Directives atomic/ordered

- Οδηγία ATOMIC:

Η οδηγία atomic καθορίζει ότι η συγκεκριμένη θέση μνήμης πρέπει να ενημερώνεται ατομικά (atomic action), μην επιτρέποντας πολλά νήματα να ενημερώσουν ταυτόχρονα τη συγκεκριμένη θέση μνήμης

#pragma omp atomic *newline*

- Οδηγία ORDERED:

Η οδηγία ordered καθορίζει ότι οι επαναλήψεις του εσωκλειώμενου βρόγχου θα εκτελεστούν με τη σειρά όπως θα εκτελούνταν σε ένα σειριακό υπολογιστή.

#pragma omp for ordered *[clauses...]*

#pragma omp ordered *newline*

Παράδειγμα directives

```
void ordered_example(int
lb, int ub, int stride)
{
int i;
#pragma omp parallel for
ordered schedule(dynamic)
for (i=lb; i<ub; i+=stride)
work(i);
}
```

```
void work(int k)
{
#pragma omp ordered
printf(" %d\n", k);
}
```

```
int main()
{
ordered_example(0,
100, 5);
return 0;
}
```

Directives threadprivate/taskwait

- Οδηγία THREADPRIVATE:

Καθορίζει ότι καθολικά αντικείμενα (ή μεταβλητές) μπορούν να γίνουν ιδιωτικά για κάποιο νήμα αλλά καθολικό μέσα στο νήμα. Με αυτό τον τρόπο, μπορούμε να ορίσουμε καθολικά αντικείμενα, αλλά να μετατρέψουμε την εμβέλειά τους και να τα κάνουμε τοπικά για κάποιο νήμα. Οι μεταβλητές για τις οποίες ισχύει η οδηγία threadprivate συνεχίζουν να είναι ιδιωτικές, για κάθε νήμα, ακόμα και σε διαφορετικές παράλληλες περιοχές

#pragma omp threadprivate

- Οδηγία TASKWAIT:

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

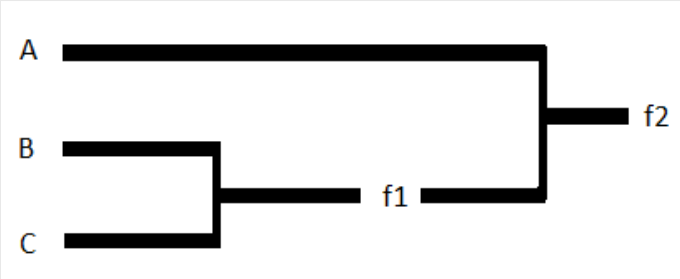
#pragma omp taskwait

Tasks

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

Παράδειγμα task

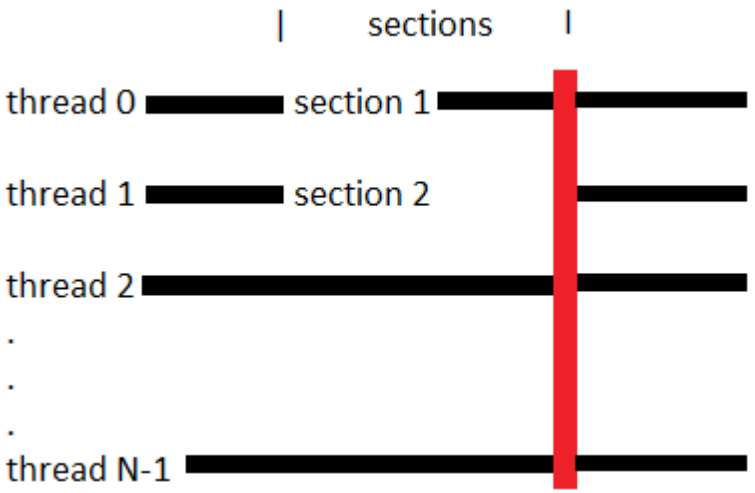
```
void foo () {
        int a, b, c, x, y;
        #pragma omp parallel {
                #pragma omp single //serial creation of tasks {
                        #pragma omp task shared (a)
                                a = A();
                        #pragma omp task shared (b, c, x) {
       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



Sections VS Tasks (1/3)

- Η εντολή sections περιγράφει ένα σύνολο από ενότητες (sections) οι οποίες θα διαμοιραστούν ανάμεσα στα νήματα.
- Κάθε ενότητα εκτελείται μόνο μια φορά από ένα νήμα.
- Έστω ότι έχουμε 1 νήμα. Στο παράδειγμα που βλέπουμε, το πρώτο θα πάρει το πρώτο section και όταν τελειώσει θα πάρει το δεύτερο.
- Αν είχαμε περισσότερα από 2 νήματα τότε τα δυο πρώτα θα έπαιρναν από ένα section και τα υπόλοιπα απλά θα περίμεναν.

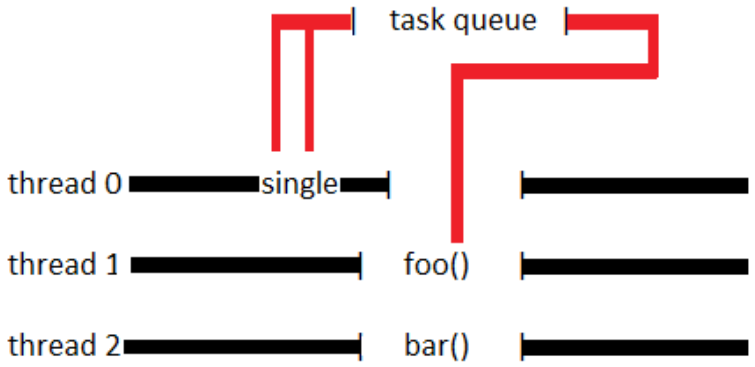
```
#pragma omp sections
{
#pragma omp section
foo();
#pragma omp section
bar();
}
```



Sections VS Tasks (2/3)

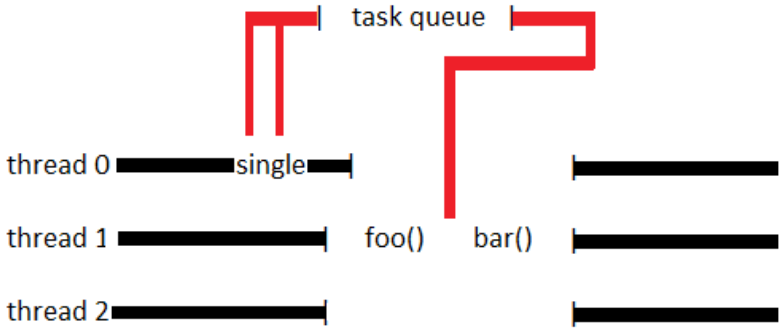
- Οι εργασίες είναι στην ουρά και εκτελούνται όποτε είναι δυνατό.
- Κατά τη διάρκεια της εκτέλεσης επιτρέπεται τα task να αλλάζουν threads, ακόμα και στο μέσο χρόνο της ζωής τους.
- Ένα task μπορεί να αρχίσει να εκτελείται σε ένα νήμα, και σε κάποια στιγμή (ακόμα και στη μέση της ζωής του) μπορεί να αρχίσει να εκτελείται σε άλλο νήμα.
- Το taskwait λειτουργεί σαν το barrier αλλά για τις διεργασίες.
- Διασφαλίζει ότι η τρέχουσα ροή εκτέλεσης θα πρέπει να διακοπεί έως ότου όλα τα task έχουν εκτελεστεί.

```
#pragma omp single nowait
{
#pragma omp task
foo();
#pragma omp task
bar();
}
#pragma omp taskwait
```



Sections VS Tasks (3/3)

- Το thread 1 και 2 αναστέλλει ότι λειτουργία εκτελεί μέχρι εκείνη τη στιγμή και ξεκινά να εκτελεί τα tasks από την ουρά.
- Μετά την ολοκλήρωση τα thread συνεχίζουν τη λειτουργία που εκτελούσαν πριν.
- Το thread 0 μπορεί να τελειώσει μετά από την ολοκλήρωση του taskwait.
- Επίσης το thread 1 μπορεί να έχει τελειώσει την εργασία που εκτελεί (foo()) και να ζητήσει και άλλη εργασία (bar()), ακόμα και όταν τα άλλα treads δεν είναι σε θέση για να ζητήσουν μια εργασία.



Clauses (1/3)

- **private(*list);***

Λίστα ιδιωτικών μεταβλητών.

- **shared(*list);***

Λίστα κοινόχρηστων μεταβλητών.

- **default(shared|none);**

Ορίζει ένα προκαθορισμένο τύπο πρόσβασης για όλες τις μεταβλητές σε μια παράλληλη περιοχή περιοχές του κώδικα.

- **firstprivate *(list)***

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

- **reduction(*operator:list);***

Ένας τρόπος για να ενώσουμε τα επιμέρους αποτελέσματα που έχουν υπολογίσει τα threads π.χ. reduction(+:sum).

Reduction

Οι REDUCTION μεταβλητές πρέπει να είναι δηλωμένες ως SHARED, πρέπει να είναι βαθμωτές και δεν μπορεί να είναι πίνακες ή δομές.

- Οι πράξεις αναγωγής μπορεί να μη λειτουργούν παρόμοια σε πραγματικούς αριθμούς.
- Οι φράσεις REDUCTION μπορεί να έχουν μία απο τις ακόλουθες μορφές:

x = x op expr

x = expr op x (εκτός από αφαίρεση)

x binop = expr

x++

++x

x--

--x

- **X:** είναι βαθμωτή μεταβλητή στη λίστα.
- **expr:** είναι βαθμωτή έκφραση που δεν αναφέρεται στο x.
- **op:** δεν είναι υπερφορτωμένος (overloaded), και είναι ένας από τους +, *, -, /, &, ^, |, &&, ||
- **binop:** δεν είναι υπερφορτωμένος (overloaded), και είναι ένας από τους +, *, -, /, &, ^, |

Παράδειγμα (reduction)

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

```
#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);
}
```

Clauses (2/3)

- **copyin *(list)***

Δίνει αρχική τιμή σε threadprivate μεταβλητές.

- **num_threads *(integer-expression)***

Ο αριθμός threads που θα χρησιμοποιηθούν.

- **structured_block**

Ορισμός του block που θα γίνει η παραλληλοποίηση.

- **untied**

Κανονικά, ένα task είναι «δεμένο» (“tied”) με το νήμα που θα ξεκινήσει να το εκτελεί – δηλαδή θα εκτελεστεί από το νήμα αυτό μέχρι τέλους. Σε ένα ελεύθερο task (“untied”), η εκτέλεσή του μπορεί να διακόπτεται και να συνεχίζει την εκτέλεσή του άλλο νήμα, κλπ.

- **lastprivate**

Επιπλέον ανάθεση της τελικής τιμής όπως αυτή προβλέπεται από την ακολουθιακή εκτέλεση.

Clauses (3/3)

schedule ***(type [,chunk])***

Πώς θα γίνει ο καταμερισμός των επαναλήψεων.

- ordered

Καθορίζει πως οι επαναλήψεις του loop πρέπει να εκτελεστούν με την σειρά που επιβάλλει ο σειριακός κώδικας.

- **collapse** ***(n)***

Καθορίζει πόσα επίπεδα από loops θα συσχετιστούν με την δομή διαμοίρασης εργασίας σε loop.

- **nowait**

Αφαιρεί το barrier. Εξ' ορισμού υπονοείται barrier στο τέλος του omp for.

- **copyprivate**

Χρησιμοποιείται για την διανομή τιμών μεταβλητής από ένα νήμα σε όλα τα αντίγραφα της μεταβλητής στα υπόλοιπα νήματα.

firstprivate VS lastprivate

- Η φράση **firstprivate** χρησιμοποιείται με τον ίδιο τρόπο όπως και η private.
- Η μεταβλητή που έχει δηλωθεί ως firstprivate, αρχικοποιείται στην τιμή που είχε πριν από το τμήμα κώδικα, αμέσως μετά τη νέα δήλωσή της. Έτσι η μεταβλητή φέρει την ίδια τιμή, αλλά με το τέλος του μπλοκ κώδικα οποιαδήποτε αλλαγή σε αυτή την τιμή χάνεται, γιατί ο κώδικας φεύγει από την ορατότητά της.
- Το firstprivate χρησιμοποιείται στις οδηγίες for, parallel, sections, single.
- Η φράση **lastprivate** συναντάται μόνο σε οδηγίες for και sections και είναι αντίστοιχη με τη firstprivate.
- Η διαφορά είναι ότι η μεταβλητή δηλώνεται ξανά μέσα στο μπλοκ κώδικα, και όταν αυτό τελειώσει, ενημερώνει την τιμή της μέχρι τότε σκιασμένης μεταβλητής.
- Το lastprivate χρησιμοποιείται στις οδηγίες for, sections.

Παράδειγμα (1/2)

```
int A, B, C;
A = B = C = 1;
#pragma omp parallel shared(A) private(B) firstprivate(C)
{
#pragma omp single
A++;
B++;
C++;
printf("%d, %d, %d", A,B,C);
}
printf("%d, %d, %d", A,B,C);
```

- Μέσα στο παράλληλο τμήμα θα τυπωθεί :
  - 2 (ή 1), <τυχαία τιμή>, 2
- Μετά το παράλληλο τμήμα θα τυπωθεί:
  - 2, 1, 1

copyin VS copyprivate

- Η φράση **copyin** ακολουθούμενη από μια λίστα μεταβλητών παρέχει ένα μηχανισμό για την αντιγραφή τιμών από μια μεταβλητή threadprivate του νήματος-αρχηγού σε καθεμία threadprivate μεταβλητή των άλλων μελών της ομάδας νημάτων που εκτελούν μια παράλληλη περιοχή.
- Το copyin χρησιμοποιείται στις οδηγίες parallel, for, sections
- Η φράση **copyprivate** ακολουθούμενη από μια λίστα μεταβλητών παρέχει ένα μηχανισμό για τη χρήση μιας ιδιωτικής μεταβλητής για μεταφορά τιμής από ένα νήμα στα άλλα νήματα της ίδιας ομάδας.
- Το copyprivate χρησιμοποιείται στην οδηγία single.

Σύγκριση χρόνων εκτέλεσης

```
for (i = 0; i < N; i++)
{
for (j = 0; j < N; j++)
for (k = sum = 0; k < N;
k++)
sum += A[i][k]*B[k][j];
C[i][j] = sum;
}
```
Χρόνος: 130msec

```
#pragma omp parallel for
private(j,k,sum)
for (i = 0; i < N; i++)
{
for (j = 0; j < N; j++)
for (k = sum = 0; k < N; k++)
sum += A[i][k]*B[k][j];
C[i][j] = sum;
}
```
Χρόνος: 40msec

```
#pragma omp parallel for
for (i = 0; i < N; i++)
{
for (j = 0; j < N; j++)
for (k = sum = 0; k < N;
k++)
sum += A[i][k]*B[k][j];
C[i][j] = sum;
}
```
Χρόνος: 700msec

Τι είναι Ρουτίνες Συστήματος Εκτέλεσης

- Το πρότυπο OpenMP ορίζει ένα API με κλήσεις σε ρουτίνες του συστήματος εκτέλεσης για διάφορες λειτουργίες:
  - Εύρεση/Καθορισμός διαθέσιμων νημάτων/ επεξεργαστών.
  - Κλειδώματα (semaphores).
  - Μέτρηση χρόνου εκτέλεσης.
  - Δυναμική προσαρμογή αριθμού νημάτων κλπ.

Ρουτίνες Συστήματος Εκτέλεσης (1/5)

- OMP_GET_NUM_THREADS:

Η συνάρτηση αυτή επιστρέφει το πλήθος των νημάτων που εκτελούνται εκείνη τη στιγμή.

- OMP_SET_NUM_THREADS:

Ο αριθμός των νημάτων που δημιουργούνται κατά την είσοδο σε μια παράλληλη περιοχή.

- OMP_GET_MAX_THREADS:

Η συνάρτηση αυτή επιστρέφει την μέγιστη τιμή που η συνάρτηση OMP_GET_NUM_THREADS μπορεί να επιστρέψει.

- OMP_GET_THREAD_NUM:

Η συνάρτηση αυτή επιστρέφει τον αριθμό του νήματος που την καλεί. Το master thread έχει την τιμή 0.

“Hello world” program 2 (1/2)

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

“Hello world” program 2 (2/2)

```
#include <omp.h>
main () {
int nthreads, tid;
/* Ορίζει μια ομάδα από threads όπου κάθε ένα έχει μια ιδιωτική μεταβλητή tid*/
#pragma omp parallel private(tid)
{
/* Βρίσκει και τυπώνει το thread id*/
tid = omp_get_thread_num();
printf("Hello World from thread = %d\n", tid);
/*Μόνο το master thread εκτελεί αυτό το κομμάτι */
if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n", nthreads);
}
}
}
```

Ρουτίνες Συστήματος Εκτέλεσης (2/5)

- OMP_GET_NUM_PROCS:

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

- OMP_IN_PARALLEL:

Η συνάρτηση αυτή ορίζει αν τη στιγμή της κλήσης της βρισκόμαστε σε παράλληλη ή σειριακή περιοχή.

- OMP_GET_DYNAMIC:

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

- OMP_SET_DYNAMIC:

Ορίζει αν θα ενεργοποιηθεί ή όχι η δυναμική προσαρμογή του πλήθους των νημάτων.

Παράδειγμα (ρουτίνες συστήματος)

Για να ελεγχθεί ο αριθμός των νημάτων που εκτελούν ένα πρόγραμμα αρχικά απενεργοποιείται το dynamic mode και κατόπιν καθορίζεται ο αριθμός των νημάτων.

```
#include <omp.h>
void main()
{
omp_set_dynamic(0);
omp_set_num_threads(4);
#pragma omp parallel
{
int id=omp_get_thread_num();
do_lots_of_stuff(id);
}
}
```

Ρουτίνες Συστήματος Εκτέλεσης (3/5)

- OMP_SET_NESTED:

Με τη συνάρτηση αυτή μπορούμε να ενεργοποιήσουμε και να απενεργοποιήσουμε τη δυνατότητα για ένθετο παραλληλισμό.

- OMP_GET_NESTED:

Εξετάζει αν επιτρέπεται ένθετος παραλληλισμός ή όχι.

- OMP_SET_SCHEDULE:

Θέτει τον τρόπο διαμοίρασης εργασίας σε νήματα.

- OMP_GET_SCHEDULE:

Επιστρέφει τον τρόπο διαμοίρασης εργασίας σε νήματα και το μέγεθος του chunk.

Ρουτίνες Συστήματος Εκτέλεσης (4/5)

- OMP_GET_THREAD_LIMIT:

Επιστρέφει τον μέγιστο αριθμό των νημάτων που είναι διαθέσιμα στο πρόγραμμα.

- OMP_SET_MAX_ACTIVE_LEVELS:

Θέτει το μέγιστο βάθος που επιτρέπεται για δημιουργία εμφωλιασμένο παραλληλισμό.

- OMP_GET_MAX_ACTIVE_LEVELS:

Επιστρέφει το μέγιστο βάθος που επιτρέπεται για δημιουργία εμφωλιασμένο παραλληλισμό.

- INT OMP_GET_LEVEL:

Επιστρέφει το πλήθος των εμφωλιασμένων δομών “parallel” για το έργο που καλεί την συνάρτηση.

Ρουτίνες Συστήματος Εκτέλεσης (5/5)

- OMP_GET_THREAD_LIMIT:

Επιστρέφει το μέγιστο πλήθος νημάτων που μπορεί να χρησιμοποιήσει το πρόγραμμα.

- INT OMP_GET_ANCESTOR_THREAD_NUM:

Επιστρέφει για το τρέχον νήμα τον αριθμό του νήματος-προγόνου στο επίπεδο εμφωλιασμού.

- OMP_GET_TEAM_SIZE:

Επιστρέφει το πλήθος των νημάτων που αποτελούν την ομάδα στο επίπεδο εμφωλιασμού.

- OMP_GET_ACTIVE_LEVEL:

Επιστρέφει το πλήθος των εμφωλιασμένων δομών “parallel” για το έργο που καλεί την συνάρτηση.

Ρουτίνες Συστήματος Εκτέλεσης Lock (1/3)

- OMP_INIT_LOCK:

Αρχικοποιεί μια κλειδαριά στην οποία αναφέρεται ο δείκτης *lock.*

- OMP_INIT_NEST_LOCK:

Η αρχική κατάσταση της κλειδαριάς είναι "*unlock*".

- OMP_DESTROY_LOCK:

Καταστρέφει την κλειδαριά στην οποία αναφέρεται ο δείκτης *lock*.

- OMP_DESTROY_NEST_LOCK:

Απαγορεύεται να κληθεί αυτή η συνάρτηση με μεταβλητή *lock* η οποία δεν έχει αρχικοποιηθεί.

Ρουτίνες Συστήματος Εκτέλεσης Lock (2/3)

- OMP_SET_LOCK:

Αναγκάζει το νήμα που εκτελείται να περιμένει μέχρις ότου η κλειδαριά στην οποία δείχνει το *lock,* να γίνει διαθέσιμη.

- OMP_SET_NEST_LOCK:

Απαγορεύεται να κληθεί αυτή η συνάρτηση με μεταβλητή *lock* η οποία δεν έχει αρχικοποιηθεί.

- OMP_UNSET_LOCK:

Αποδεσμεύει τη κλειδαριά από το νήμα που την χρησιμοποιούσε.

- OMP_UNSET_NEST_LOCK:

Απαγορεύεται να κληθεί αυτή η συνάρτηση με μεταβλητή *lock* η οποία δεν έχει αρχικοποιηθεί.

Ρουτίνες Συστήματος Εκτέλεσης Lock (3/3)

- OMP_TEST_LOCK:

Προσπαθεί να αρχικοποιήσει μια κλειδαριά, αλλά δεν κάνει block το νήμα που εκτελεί αυτή τη συνάρτηση, ακόμα κι αν η κλειδαριά δεν είναι διαθέσιμη. Επιστρέφει 1 αν η κλειδαριά έχει αρχικοποιηθεί επιτυχώς και 0 σε αντίθετη περίπτωση.

- OMP_TEST_NEST_LOCK:

Απαγορεύεται να κληθεί αυτή η συνάρτηση με μεταβλητή *lock* η οποία δεν έχει αρχικοποιηθεί.

Ρουτίνες Συστήματος Εκτέλεσης Time

- OMP_GET_WTIME:

Ρουτίνα για τη μέτρηση χρόνου εκτέλεσης (wall clock όχι CPU).

- OMP_GET_WTICK:

Η τιμή που επιστρέφει ισούται με τον αριθμό των δευτερολέπτων μεταξύ δύο διαδοχικών παλμών ρολογιού, του χρονομέτρου που χρησιμοποιεί η *omp_get_time*.

Omp_get_time

- Σύνταξη:

```
t_start = omp_get_wtime();
#pragma omp parallel
    { .
      .
      .
    }
t_end = omp_get_wtime() - t_start
```

Παράδειγμα (omp_get_wtime) (1/2)

```
#include <stdio.h>
#include <omp.h>
long STEPS = 1000000;
#define NUM_THREADS 5
int main (int argc, char *argv[])
{
int i;
double step, x, sum, pi = 0;
double start, total;
start = omp_get_wtime();
step = 1.0 /(double) STEPS;
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for reduction (+:sum) private(x)
for (i=0;i<STEPS; i++){
x = (i+0.5)*step;
sum = sum + 4.0/(1.0+x*x);
}
pi = step * sum;
total = omp_get_wtime() - start;
printf(" pi: %lf\n steps = %ld\n num_threads = %d\n",pi, STEPS, NUM_THREADS);
printf("Total time = %lf \n" , total);
return 0;
}
```

Παράδειγμα (omp_get_wtime) (2/2)

- Ο παραπάνω κώδικας εμφανίζει τα εξής αποτελέσματα:

```
openuser@snf-17468:~/openmp$ gcc -fopenmp c2.c -o c2
openuser@snf-17468:~/openmp$ ./c2
pi: 3.141593
steps = 1000000
num_threads = 5
Total time = 0.037573
openuser@snf-17468:~/openmp$
```

Παράδειγμα (lock)

```
#include <omp.h>
omp_lock_t *new_locks()
{
int i;
omp_lock_t *lock = new omp_lock_t[1000];
#pragma omp parallel for private(i)
for (i=0; i<1000; i++)
omp_init_lock(&lock[i]);
return lock;
}
```

Τι Είναι Μεταβλητές Περιβάλλοντος

- Το OpenMP παρέχει μια σειρά μεταβλητών περιβάλλοντος που βοηθούν στην εκτέλεση του παράλληλου κώδικα.
- Τα ονόματα των μεταβλητών γράφονται πάντα σε κεφαλαία.
- Επηρεάζουν τις τιμές των εσωτερικών μεταβλητών.
- Αλλαγές στις μεταβλητές περιβάλλοντος μετά την έναρξη εκτέλεσης ενός προγράμματος αγνοούνται.
- Ακόμα και αν γίνουν από το ίδιο το πρόγραμμα.

Μεταβλητές Περιβάλλοντος schedule (1/3)

- OMP_SCHEDULE:

Η τιμή αυτής της μεταβλητής καθορίζει το τρόπο εκτέλεσης των επαναλήψεων.

- Η τιμή της μεταβλητής έχει την μορφή:

  type[,chunk]

- Το “type” μπορεί να είναι: **static, dynamic, guided, auto, runtime.**
- Το “chunk” είναι προαιρετικό. Καθορίζει το μέγεθος κάθε chunk:

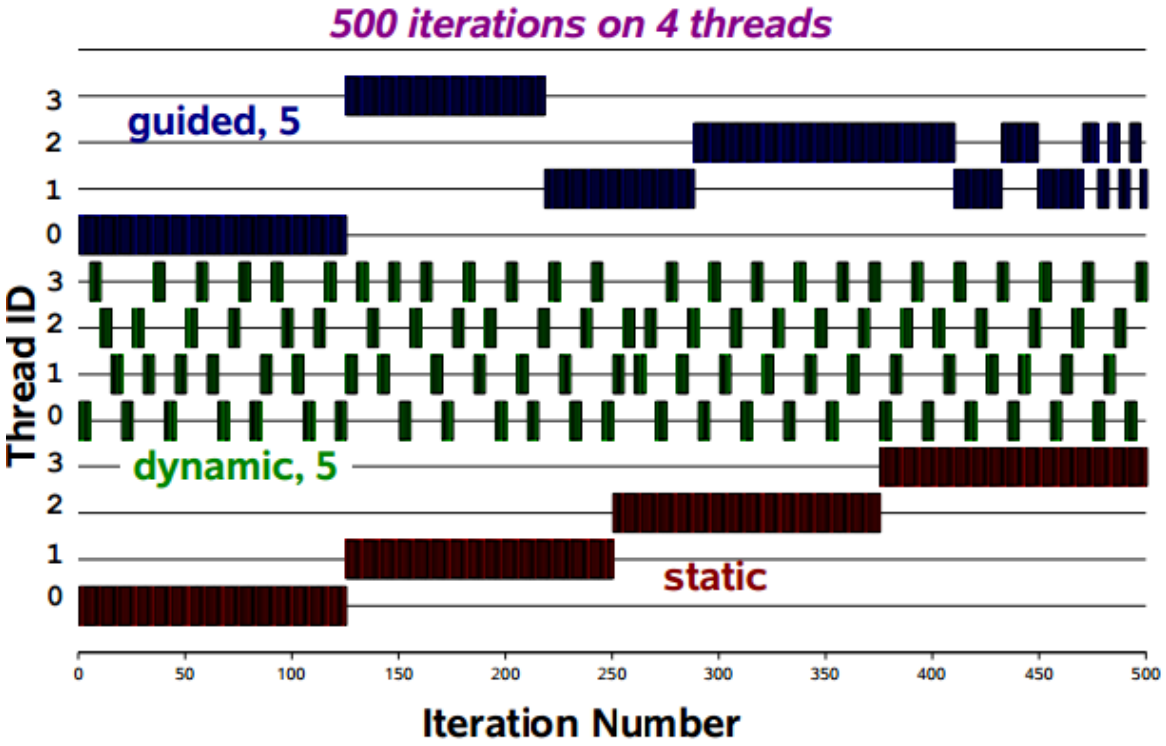
  export OMP_SCHEDULE=“guided,4”

Μεταβλητές Περιβάλλοντος schedule (2/3)

- OMP_SCHEDULE:
  - **static:** round-robin στατική κατανομή.
  - **dynamic:** δυναμική κατανομή σε ανενεργά νήματα.
  - **guided:** δυναμική κατανομή με εκθετική μείωση.
  - **auto:** ο προγραμματιστής δίνει στην εφαρμογή την «ελευθερία» να επιλέξει οποιαδήποτε πιθανή χαρτογράφηση.
  - **runtime:** η απόφαση σχεδιασμού αναβάλλεται μέχρι να αρχίσει να εκτελείται το πρόγραμμα.

Μεταβλητές Περιβάλλοντος schedule (3/3)

- Το παρακάτω σχήμα δείχνει πως θα γίνει η κατανομή των επαναλήψεων σε κάθε νήμα στην περίπτωση που έχουμε 500 επαναλήψεις και 4 threads.



Παράδειγμα (1/3)

```
int main( ) {
int nStatic1[NUM_LOOPS], nStaticN[NUM_LOOPS];
int nDynamic1[NUM_LOOPS], nDynamicN[NUM_LOOPS];
int nGuided[NUM_LOOPS];
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel {
        #pragma omp for schedule(static, 1)
                for (int i = 0 ; i < NUM_LOOPS ; ++i) {
                        if ((i % SLEEP_EVERY_N) == SLEEP_EVERY_N)
                                Sleep(0);
                        nStatic1[i] = omp_get_thread_num( ); }
        #pragma omp for schedule(static, STATIC_CHUNK)
                for (int i = 0 ; i < NUM_LOOPS ; ++i) {
                        if ((i % SLEEP_EVERY_N) == SLEEP_EVERY_N)
                                Sleep(0);
                        nStaticN[i] = omp_get_thread_num( ); }
        #pragma omp for schedule(dynamic, 1)
                for (int i = 0 ; i < NUM_LOOPS ; ++i) { I
                        if ((i % SLEEP_EVERY_N) == SLEEP_EVERY_N)
                                Sleep(0);
                        nDynamic1[i] = omp_get_thread_num( ); }
```

Παράδειγμα (2/3)

```
#pragma omp for schedule(dynamic, DYNAMIC_CHUNK)
        for (int i = 0 ; i < NUM_LOOPS ; ++i) {
                if ((i % SLEEP_EVERY_N) == SLEEP_EVERY_N)
                        Sleep(0);
                nDynamicN[i] = omp_get_thread_num( ); }
#pragma omp for schedule(guided)
        for (int i = 0 ; i < NUM_LOOPS ; ++i) {
                if ((i % SLEEP_EVERY_N) == SLEEP_EVERY_N)
                        Sleep(0);
                nGuided[i] = omp_get_thread_num( ); }
        }
printf_s("-----------------------------------------\n");
printf_s("| static | static | dynamic | dynamic | guided |\n");
printf_s("| 1 | %d | 1 | %d | |\n", STATIC_CHUNK, DYNAMIC_CHUNK);
printf_s("-----------------------------------------\n");  for (int i=0; i<NUM_LOOPS; ++i) {
printf_s("| %d | %d | %d | %d |" " %d |\n", nStatic1[i], nStaticN[i], nDynamic1[i], nDynamicN[i],
nGuided[i]);
}
printf_s("-----------------------------------------\n");
}
```

Παράδειγμα (3/3)

static 1	static 5	dynamic 1	dynamic 5	guided
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
0	0	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
0	1	0	0	0
1	1	0	0	0
2	2	0	0	0
3	2	0	0	0
0	2	0	0	0
1	2	0	0	0
2	2	0	0	0
3	3	0	0	0
0	3	0	0	0
1	3	0	0	0
2	3	0	0	0
3	3	0	0	0

- Η έξοδος του παραπάνω παραδείγματος φαίνεται δίπλα.

Μεταβλητές Περιβάλλοντος num_threads-dynamic-nested

- OMP_NUM_THREADS:

Ελέγχει το πλήθος των νημάτων που δημιουργούνται σε μια δομή “parallel”. Η τιμή της μεταβλητής πρέπει να είναι ένας θετικός ακέραιος

export OMP_NUM_THREADS=16

- OMP_DYNAMIC:

Ελέγχει την δυνατότητα δυναμικής προσαρμογής του πλήθους των νημάτων που χρησιμοποιούνται σε μια δομή “parallel”

export OMP_DYNAMIC=true

- OMP_NESTED:

Επιτρέπει ή απαγορεύει ένθετο παραλληλισμό, αν το επιτρέπει η υλοποίηση.

export OMP_NESTED=true

Μεταβλητές Περιβάλλοντος stacksize-wait_policy

- OMP_STACKSIZE:

Ελέγχει το μέγεθος στοίβας των (non-master) νημάτων.

export OMP_STACKSIZE=2000500B

export OMP_STACKSIZE=3000k

- OMP_WAIT_POLICY:

Ορίζει αν η αναμονή των νημάτων στην οδηγία wait θα είναι ενεργός (busy waiting) ή όχι.

export OMP_WAIT_POLICY=ACTIVE

export OMP_WAIT_POLICY=PASSIVE

Μεταβλητές Περιβάλλοντος thread_limit-max_active_levels

- OMP_THREAD_LIMIT:

Ορίζει το μέγιστο αριθμό νημάτων που μπορεί να χρησιμοποιήσει ένα πρόγραμμα OpenMP.

export OMP_THREAD_LIMIT=200

- OMP_MAX_ACTIVE_LEVELS:

Ελέγχει το μέγιστο επιτρεπτό βαθμό εμφωλιασμένου παραλληλισμού.

export OMP_MAX_ACTIVE_LEVEL=4

Παράδειγμα (parallel firstprivate)

```
#include <assert.h>
int A[2][2] = {1, 2, 3, 4};
void f(int n, int B[n][n], int C[]) {
        int D[2][2] = {1, 2, 3, 4};
        int E[n][n];
        assert(n >= 2);
        E[1][1] = 4;
        #pragma omp parallel firstprivate(B, C, D, E) {
                assert(sizeof(B) == sizeof(int (*)[n]));
                assert(sizeof(C) == sizeof(int*));
                assert(sizeof(D) == 4 * sizeof(int));
                assert(sizeof(E) == n * n * sizeof(int));
                /* Private B and C have values of original B and C. */
                assert(&B[1][1] == &A[1][1]);
                assert(&C[3] == &A[1][1]);
                assert(D[1][1] == 4);
                assert(E[1][1] == 4);
                }
                }
int main() {
f(2, A, A[0]);
return 0;
}
```

Παράδειγμα (1/3)

```
#include <stdio.h>
#include <omp.h>
int main() {
omp_set_dynamic(1);
omp_set_num_threads(10);
#pragma omp parallel // parallel region 1
{
#pragma omp single
printf("Num threads in dynamic region is = %d\n", omp_get_num_threads());
}
printf("\n");
omp_set_dynamic(0);
omp_set_num_threads(10);
#pragma omp parallel // parallel region 2
{
#pragma omp single printf("Num threads in non-dynamic region is = %d\n", omp_get_num_threads());
}
printf("\n");
```

Παράδειγμα (2/3)

```
omp_set_dynamic(1);
omp_set_num_threads(10);
#pragma omp parallel // parallel region 3
{
        #pragma omp parallel
        {
        #pragma omp single
        printf("Num threads in nesting disabled region is = %d\n", omp_get_num_threads());
        }
}
printf("\n");
omp_set_nested(1);
#pragma omp parallel // parallel region 4
{
        #pragma omp parallel
{
                #pragma omp single
        printf("Num threads in nested region is = %d\n", omp_get_num_threads());
        }
} }
```

Παράδειγμα (3/3)

Ο παρακάτω κώδικας εμφανίζει:

- Num threads in dynamic region is = 2
- Num threads in non-dynamic region is = 10
- Num threads in nesting disabled region is = 1
- Num threads in nesting disabled region is = 1
- Num threads in nested region is = 2
- Num threads in nested region is = 2

Ολοκλήρωση Τραπεζίου (1/2)

Το παρακάτω πρόγραμμα υπολογίζει το ολοκλήρωμα από 1 έως 4 της συνάρτησης f (x)=∫(3x)^2 + 1.

```
int main(int argc, char**
argv)
{
int n,thread_count;
float a=1, b=4,
global_result=0.0;
printf("Dwse ton aritho twn
trapeziwn : \n");
scanf("%d",&n);
# pragma omp parallel
num_threads(thread_count)
trap(a,b,n,&global_result);
printf("Oloklhrwma :
%f\n",global_result);
return 0;
}
```

```
float f(float x)
{
return ((3*x*x)+1);
}
```

Ολοκλήρωση Τραπεζίου (2/2)

```
void trap(float a2,float b2, int n2,float* result )
{
float interg,x,h,n,a,b;
int i, nthreads, thread_count;
nthreads = omp_get_thread_num();
thread_count = omp_get_num_threads();
h=(b2-a2)/n2 ;
n=n2/thread_count;
a=a2+nthreads*n*h;
b=a+n*h;
interg=(f(a)+f(b))/2.0;
for(i=1;i<=n-1;i++)
{
x=a+i*h;
interg=interg+f(x);
}
interg= interg*h;
#pragma omp critical
*result+=interg;
}
```

OpenMP compilers

- GCC:
  - Υποστήριξη tasks από την έκδοση 4.3.x (όμως άσχημη υλοποίηση).
  - Καλή υλοποίηση από έκδοση 4.4.x.
- ICC:
  - Intel compiler & tools.
  - Εξαιρετικά εργαλεία, ταχύτατα σειριακά προγράμματα για x86.
  - Έχει εξαγοράσει την Cilk Arts.
- SUNCC:
  - SUN C compiler.
  - Γενικά καλές και σταθερές επιδόσεις, όχι όμως ο καλύτερες.
- OMPi :
  - Ανοιχτού κώδικα, πολύ καλές επιδόσεις, επεκτάσεις κ.α.
- Άλλοι ερευνητικοί, ελεύθερου κώδικα, με όχι πλήρη υποστήριξη OpenMP 3:
  - OpenUH.
  - Mercirium (Nanaos).
  - OdinMP.

Βιβλιογραφία (1/2)

- http://el.wikipedia.org/wiki/%CE%A0%CE%B1%CF%81%CE%AC%CE%BB%CE%BB%CE%B7%CE%BB%CE%BF%CF%82_%CF%80%CF%81%CE%BF%CE%B3%CF%81%CE%B1%CE%BC%CE%BC%CE%B1%CF%84%CE%B9%CF%83%CE%BC%CF%8C%CF%82
- http://www.tem.uoc.gr/~vagelis/Courses/EM361/Ch5_Software_Basic-Applications_I.pdf
- http://el.scribd.com/doc/78091055/Introduction-to-Openmp
- http://www.cslab.ntua.gr/courses/pps/files/fall2012/OpenMPpresentation-Fall2012.pdf
- http://www.it.uom.gr
- http://www.cs.uoi.gr/~dimako/Courses/Fall12/openmp.pdf
- http://www.cs.uoi.gr/~phadjido/courses/E-85/lectures/E-85-Lec06.pdf
- http://www.cslab.ece.ntua.gr/courses/pps/files/fall2004/OpenMPpresentation2004-2005.pdf
- http://www.openmp.org/mp-documents/spec30.pdf

Βιβλιογραφία (2/2)

- http://www.openmp.org/mp-documents/OpenMP3.1.pdf
- http://www.openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf
- http://openmp.org/mp-documents/omp-hands-on-SC08.pdf
- http://sc.tamu.edu/shortcourses/SC-openmp/OpenMPSlides_tamu_sc.pdf
- http://msdn.microsoft.com/en-us/magazine/cc163717.aspx

Τέλος Ενότητας

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης