Ε-85: Ειδικά Θέματα Λογισμικού
Προγραμματισμός Συστημάτων Υψηλών Επιδόσεων

Χειμερινό Εξάμηνο 2009-10

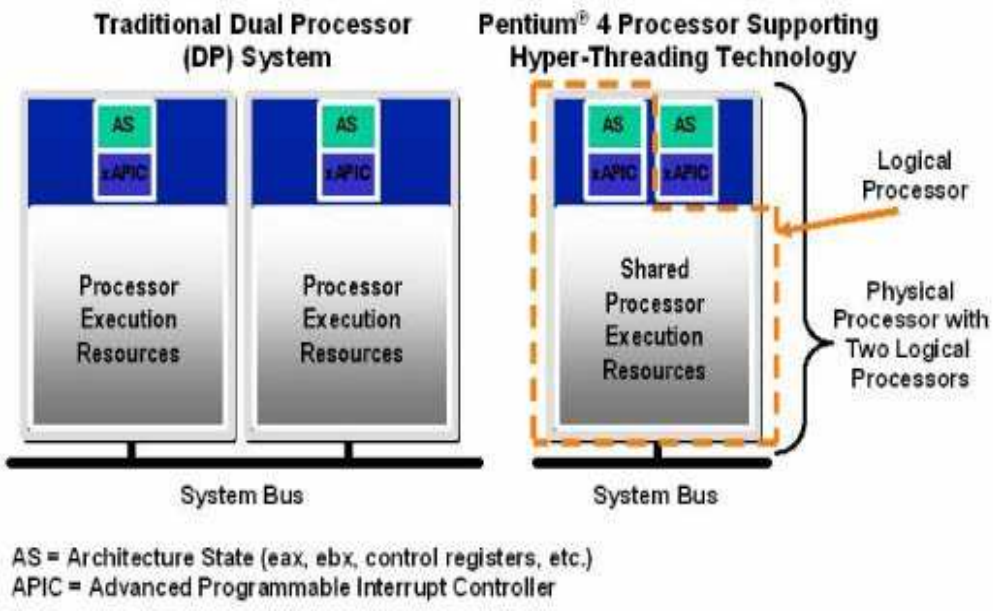
«Εισαγωγή στο OpenMP»

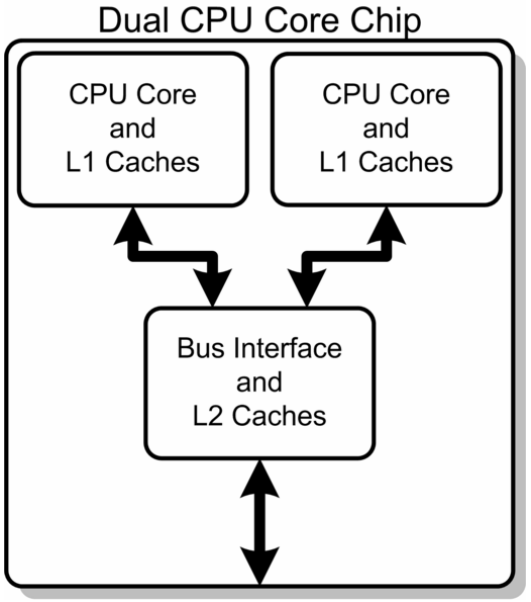
Παναγιώτης Χατζηδούκας
(Π.Δ. 407/80)

ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

Πολυεπεξεργαστές

- Επεξεργαστές με τεχνολογία Hyper-threading (SMT)
- Επεξεργαστές με τεχνολογία Multi-Core





Λογισμικό

- Οι περισσότεροι συγγραφείς εφαρμογών αγνοούν τον παραλληλισμό
  - εκτός ίσως από αδρά καταμερισμένο (βαρύ) πολυνηματισμό για GUI's και λογισμικό συστήματος
- Γιατί;
  - Οι δυσκολίες συγγραφής παράλληλου software υπερνικούν τα πλεονεκτήματα
  - Τα πλεονεκτήματα είναι ξεκάθαρα. Για να αυξηθεί ο όγκος του παράλληλου software πρέπει να ελαχιστοποιηθούν οι δυσκολίες.

Υπολογισμός του π – Ακολουθιακή Έκδοση

```
static long num_steps;
double step;
int main ()
{      int i;
       double x, pi, sum = 0.0;
       num_steps = 100000;

       step = 1.0/(double) num_steps;

       for (i=1;i<= num_steps; i++){
              x = (i-0.5)*step;
              sum = sum + 4.0/(1.0+x*x);
       }
       pi = step * sum;

      return 0;
}
```

Υπολογισμός του π – Πολυνηματική Έκδοση

```
#include <pthread.h>
#define NUM_THREADS 2
pthread_t thread[NUM_THREADS];
pthread_mutex_t UpdateMutex;
static long num_steps;
double step;
double global_sum = 0.0;

void *Pi (void *arg)
{
  int i, start;
  double x, sum = 0.0;

  start = *(int *) arg;
  step = 1.0/(double) num_steps;

  for (i=start;i<= num_steps;
      i=i+NUM_THREADS){
       x = (i-0.5)*step;
      sum = sum + 4.0/(1.0+x*x);
  }
  pthread_mutex_lock (&UpdateMutex);
  global_sum += sum;
  pthread_mutex_unlock(&UpdateMutex);

  return 0;
}
```

```
int main ()
{
  double pi; int i;
  num_steps = 100000;

  int Arg[NUM_THREADS];

  for(i=0; i<NUM_THREADS; i++)
      threadArg[i] = i+1;

  pthread_mutex_init (&UpdateMutex, NULL);

  for (i=0; i<NUM_THREADS; i++)
      pthread_create(&thread[I], NULL, Pi,&Arg[i]);


  for (i=0; i<NUM_THREADS; i++)
      pthread_join(thread[i], NULL);

  pi = global_sum * step;

  return 0;
}
```

Υπολογισμός του π – με χρήση του OpenMP

```
#include <omp.h>
static long num_steps = 100000;
double step;
#define NUM_THREADS 2
int main ()
{      int i;
       double x, pi, sum = 0.0;

       step = 1.0/(double) num_steps;
       omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for reduction(+:sum) private(x)
       for (i=1;i<= num_steps; i++){
              x = (i-0.5)*step;
              sum = sum + 4.0/(1.0+x*x);
       }
       pi = step * sum;

      return 0;
}
```

Εισαγωγή στο OpenMP

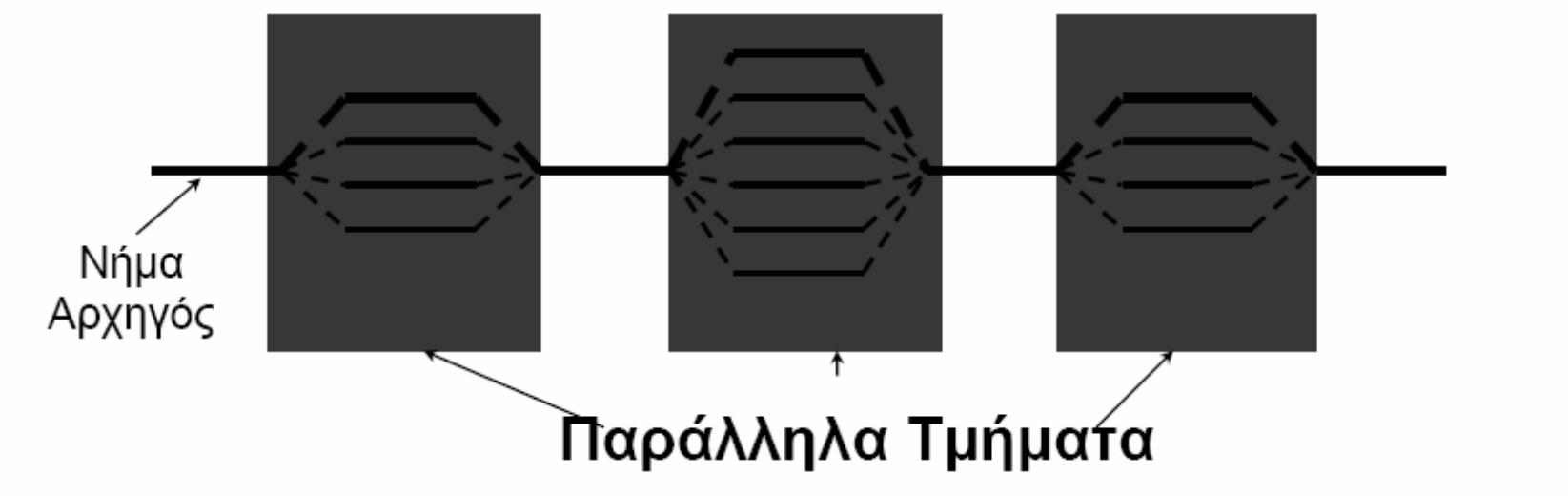
- OpenMP: API για τη συγγραφή πολυνηματικών εφαρμογών
  - Σύνολο οδηγιών προς τον μεταγλωττιστή και συναρτήσεων βιβλιοθήκης, διαθέσιμο στον προγραμματιστή παράλληλων συστημάτων
  - Διευκολύνει τη συγγραφή πολυνηματικών προγραμμάτων σε Fortran, C and C++
  - Πρότυπο που συγκεντρώνει την εμπειρία αρκετών χρόνων σε προγραμματισμό πολυεπεξεργαστικών συστημάτων

OpenMP

- Μοντέλο προγραμματισμού για παράλληλα συστήματα
- Οδηγίες προς τον μεταγλωττιστή ώστε να παράγει κώδικα με νήματα
- Όλες οι μεγάλες εταιρίες το υποστηρίζουν ή/και αποφασίζουν για την εξέλιξή του
  - Intel, SUN, IBM, HP, SGI, …
  - Microsoft (Visual Studio 2005)
  - GNU GCC 4.2
- Ερευνητικοί compilers
  - Omni (Ιαπωνία), NANOS (Ισπανία)
  - OpenUH (ΗΠΑ), OMPi (Ελλάδα - UoI)

Προγραμματιστικό Μοντέλο

- Παραλληλισμός τύπου Fork-Join:
  - Το νήμα αρχηγός δημιουργεί ομάδα νημάτων σύμφωνα με τις ανάγκες
  - Ο παραλληλισμός προστίθεται βαθμιαία
    - το ακολουθιακό πρόγραμμα εξελίσσεται σε παράλληλο πρόγραμμα



Τυπική Χρήση

- Το OpenMP συνήθως χρησιμοποιείται για την παραλληλοποίηση loops:
  - Βρες τα πιο χρονοβόρα loops.
  - Μοίρασε τις επαναλήψεις μεταξύ νημάτων.

Διαίρεσε αυτό το loop μεταξύ πολλαπλών νημάτων

```
void main()
{
  double Res[1000];

  for (int i=0;i<1000;i++) {
      do_huge_comp(Res[i]);
  }
}
```

Ακολουθιακό πρόγραμμα

```
void main()
{
   double Res[1000];
  #pragma omp parallel for
  for (int i=0;i<1000;i++) {
      do_huge_comp(Res[i]);
  }
}
```

Παράλληλο πρόγραμμα

Αλληλεπίδραση Νημάτων

- Πλεονέκτημα: Απόκρυψη «λεπτομερειών» από τον προγραμματιστή
- Μειονέκτημα: Απόκρυψη πολλών «λεπτομερειών» από τον προγραμματιστή μπορεί να βλάψει την επίδοση
- Το OpenMP είναι μοντέλο κοινής μνήμης
  - Τα νήματα επικοινωνούν μέσω διαμοιραζόμενων μεταβλητών
- Ακούσια διαμοίραση δεδομένων μπορεί να προκαλέσει races:
  - race : Το αποτέλεσμα του προγράμματος αλλάζει με τυχαίο τρόπο αν τα νήματα εκτελεστούν με διαφορετική σειρά
- Εξάλειψη races :
  - Χρήση συγχρονισμού στα σημεία που χρειάζεται
- Ο συγχρονισμός είναι «ακριβός», άρα:
  - Αλλαγή της δομής του προγράμματος και της οργάνωσης των δεδομένων ώστε να μειωθούν οι απαιτήσεις συγχρονισμού

Σύνταξη Οδηγιών

- Οι περισσότερες «εντολές» OpenMP είναι directives προς τον compiler ή pragmas.
- Για την C και C++, τα pragmas έχουν τη μορφή:
  - #pragma omp construct [clause [clause]…]
- Για τη Fortran, τα directives έχουν μία από τις ακόλουθες μορφές:
  - C$OMP construct [clause [clause]…]
  - !$OMP construct [clause [clause]…]
  - *$OMP construct [clause [clause]…]
- Αφού οι «εντολές» είναι directives και pragmas
  - ένα πρόγραμμα OpenMP μπορεί να μεταγλωττιστεί από compilers που δεν υποστηρίζουν OpenMP
  - οι τελευταίοι απλά αγνοούν τα directives / pragmas

Δομημένα Τμήματα (blocks)

- Οι περισσότερες «εντολές» OpenMP εφαρμόζονται σε δομημένα τμήματα (blocks) κώδικα.
  - Δομημένο τμήμα: ένα τμήμα κώδικα με ένα σημείο εισόδου στην κορυφή και ένα σημείο εξόδου στο τέλος. Οι μόνες επιτρεπτές διακλαδώσεις είναι εντολές STOP της Fortran και exit() της C/C++.

```
C$OMP PARALLEL
10        wrk(id) = garbage(id)
          res(id) = wrk(id)**2
          if(conv(res(id)) goto 10
C$OMP END PARALLEL
          print *,id
```

Δομημένο τμήμα

```
C$OMP PARALLEL
10        wrk(id) = garbage(id)
30        res(id)=wrk(id)**2
          if(conv(res(id))goto 20
          go to 10
C$OMP END PARALLEL
          if(not_DONE) goto 30
20        print *, id
```

Μη δομημένο τμήμα

Παράλληλα Τμήματα

- Νήματα δημιουργούνται στο OpenMP (στη C/C++) με το pragma “omp parallel”
- Για παράδειγμα, για να δημιουργηθεί ένα παράλληλο τμήμα με 4 νήματα:

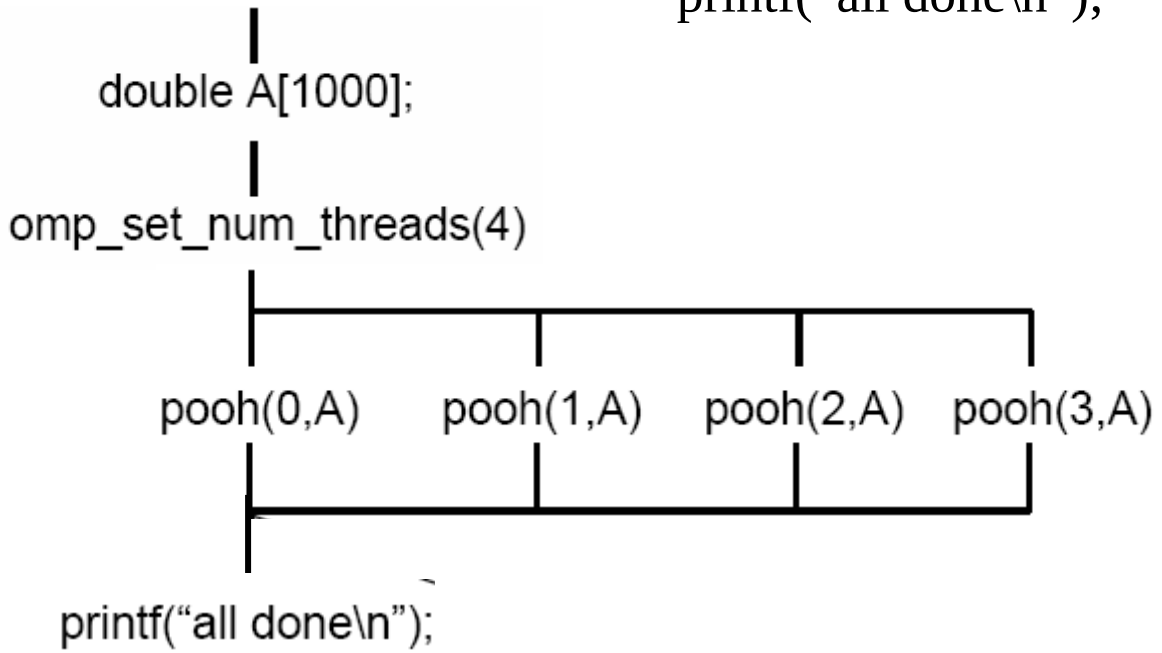
```
double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
   int ID = omp_thread_num();
   pooh(ID,A);
}
```

- Κάθε νήμα εκτελεί για λογαριασμό του τον κώδικα μέσα στο δομημένο block του παράλληλου τμήματος
- Κάθε νήμα καλεί την pooh(ID) για ID = 0 έως 3

Παράλληλα Τμήματα

- Κάθε νήμα εκτελεί για λογαριασμό του τον ίδιο κώδικα
- Ένα μοναδικό αντίγραφο του A μοιράζεται μεταξύ των νημάτων
- Η εκτέλεση προχωρά μόνο όταν έχουν τελειώσει όλα τα νήματα (barrier)

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



Πολυνηματικό Πρόγραμμα «Hello world»

- Γράψτε ένα πολυνηματικό πρόγραμμα στο οποίο κάθε νήμα τυπώνει ένα απλό νήματα (π.χ. “hello world”).
- Χρησιμοποιήστε 2 ξεχωριστές εντολές printf και το thread ID:

```
int ID = omp_get_thread_num();
printf(“ hello(%d) ”, ID);
printf(“ world(%d) ”, ID);
```

- Τι γίνεται όταν εκτελείται ταυτόχρονα I/O από πολλαπλά νήματα;

Μερικές Πρώτες Λεπτομέρειες

- Dynamic mode (default):
  - Ο αριθμός των νημάτων που χρησιμοποιούνται για την εκτέλεση παράλληλων τμημάτων μπορεί να διαφέρει μεταξύ διαφορετικών τμημάτων
  - Ο ορισμός του αριθμού των νημάτων αφορά τον μέγιστο αριθμό νημάτων και ενδεχομένως η εκτέλεση να γίνει με λιγότερα νήματα
- Static mode:
  - Ο αριθμός των νημάτων είναι σταθερός και μάλιστα ακριβώς αυτός που καθορίζεται από τον προγραμματιστή
- Το OpenMP υποστηρίζει εμφωλευμένα παράλληλα τμήματα, όμως…
  - Ένας compiler ενδεχομένως να επιλέξει να εκτελέσει σειριακά όλα τα επίπεδα μετά το 1ο

Δομές Διαμοίρασης Έργου

- Το omp “for” κατανέμει τις επαναλήψεις ενός loop μεταξύ των νημάτων μιας ομάδας

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

- Εξ’ ορισμού υπονοείται barrier στο τέλος του omp for
- Για να αφαιρεθεί το barrier χρησιμοποιούμε την παράμετρο “nowait”

Χρησιμότητα Δομών Διαμοίρασης

- Ακολουθιακός κώδικας

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

- Παράλληλο τμήμα OpenMP

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

- Παράλληλο τμήμα OpenMP με omp for για διαμοίραση έργου

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

Δομές Διαμοίρασης Έργου

- Η δομή διαμοίρασης έργου sections αναθέτει ένα διαφορετικό δομημένο block σε κάθε νήμα

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

- Εξ' ορισμού υπονοείται barrier στο τέλος του omp sections
- Για να αφαιρεθεί το barrier χρησιμοποιούμε την παράμετρο "nowait"

Συνδυασμοί Εντολών

- Συνδυασμός της Εντολής parallel με δομές διαμοίρασης έργου

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

- Υπάρχει και “parallel sections”

Υπολογισμός του π

- Πρόγραμμα που χρησιμοποιεί αριθμητική ολοκλήρωση για τον υπολογισμό προσέγγισης του π
- Παραλληλοποίηση τους προγράμματος χρησιμοποιώντας OpenMP. Υπάρχουν εναλλακτικές επιλογές:
  - Σαν πρόγραμμα SPMD με χρήση παράλληλου τμήματος μόνο
  - Με χρήση δομής διαμοίρασης έργου
- Διασφάλιση ότι το ένα νήμα δεν θα γράφει πάνω στις μεταβλητές του άλλου

Πρόγραμμα π – Ακολουθιακή Έκδοση

```
static long num_steps = 100000;
double step;
void main ()
{       int i; double x, pi, sum = 0.0;
        step = 1.0/(double) num_steps;
        for (i=1;i<= num_steps; i++){
                x = (i-0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
        }
        pi = step * sum;
}
```

Έκδοση με Παράλληλο Τμήμα (SPMD)

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2

void main ()
{
     int i; double x, pi, sum[NUM_THREADS];
     step = 1.0/(double) num_steps;
     omp_set_num_threads(NUM_THREADS)
#pragma omp parallel
{    int i; double x; int id;
     id = omp_get_thread_num();
     for (i=id, sum[id]=0.0;i< num_steps; i=i+NUM_THREADS) {
          x = (i+0.5)*step;
          sum[id] += 4.0/(1.0+x*x);
     }
}
     for(i=0, pi=0.0;i<NUM_THREADS;i++) pi += sum[i] * step;
}
```

SPMD: Κάθε νήμα εκτελεί τον ίδιο κώδικα. Το ID του νήματος χρησιμοποιείται για τη διαφοροποίηση της συμπεριφοράς

Έκδοση με Δομή Διαμοίρασης Έργου

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{
     int i; double x, pi, sum[NUM_THREADS];
     step = 1.0/(double) num_steps;
     omp_set_num_threads(NUM_THREADS)
#pragma omp parallel
{    int i; double x; int id;
     id = omp_get_thread_num();          sum[id] = 0.0;
#pragma omp for
     for (i=id, i< num_steps; i++){
               x = (i+0.5)*step;
               sum[id] += 4.0/(1.0+x*x);
     }
}
     for(i=0, pi=0.0;i<NUM_THREADS;i++) pi += sum[i] * step;
}
```

Εμβέλεια των Εντολών OpenMP

- Εντολές OpenMP μπορεί να επεκτείνονται σε πολλά αρχεία.
- foo.f: Static ή lexical extent του παράλληλου τμήματος
- bar.f: Dynamic extent του παράλληλου τμήματος
  - Περιλαμβάνει το static extent
- Orphan directives: εμφανίζονται εκτός παράλληλου τμήματος

foo.f

```
C$OMP PARALLEL
      call whoami
C$OMP END PARALLEL
```

bar.f

```
      subroutine whoami
      external omp_get_thread_num
      integer iam, omp_get_thread_num
      iam = omp_get_thread_num()
C$OMP CRITICAL
      print*,'Hello from ', iam
C$OMP END CRITICAL
      return
      end
```

Χαρακτηριστικά Αποθήκευσης

- Προγραμματιστικό μοντέλο κοινής μνήμης:
  - Οι περισσότερες μεταβλητές είναι by default κοινές
- Οι global μεταβλητές είναι κοινές μεταξύ των νημάτων
  - Fortran: COMMON blocks, SAVE variables, MODULE variables
  - C: File scope variables, static
- Όμως δεν είναι τα πάντα κοινά...
  - Οι stack variables σε υπο-προγράμματα που καλούνται από παράλληλα τμήματα είναι ιδιωτικές
  - Οι automatic variables μέσα σε ένα δομημένο block εντολών είναι ιδιωτικές.

Αλλαγή Χαρακτηριστικών Αποθήκευσης

- Ο προγραμματιστής μπορεί να αλλάξει τα χαρακτηριστικά αποθήκευσης των μεταβλητών χρησιμοποιώντας ένα από τα ακόλουθα
  - SHARED
  - PRIVATE
  - FIRSTPRIVATE
  - THREADPRIVATE
- Η τιμή μιας ιδιωτικής μεταβλητής εντός ενός παράλληλου loop μπορεί να «μεταδοθεί» σαν καθολική τιμή εκτός του loop με τη:
  - LASTPRIVATE
- Η default συμπεριφορά μπορεί να μεταβληθεί με τη:
  - DEFAULT (PRIVATE | SHARED | NONE)
- Όλες οι εντολές δεδομένων εφαρμόζονται σε παράλληλα τμήματα και δομές διαμοίρασης έργου εκτός της “shared” η οποία εφαρμόζεται μόνο σε παράλληλα τμήματα.
- Όλες οι εντολές αυτής της διαφάνειας έχουν ισχύ στο lexical extent της εντολής OpenMP.

Εντολή private

- Η private(var) δημιουργεί ιδιωτικό αντίγραφο της var για κάθε νήμα
  - Η τιμή δεν είναι αρχικοποιημένη
  - Το ιδιωτικό αντίγραφο δεν συσχετίζεται (όσον αφορά τον χώρο αποθήκευσης) με το αυθεντικό

```
                program wrong
                IS = 0
C$OMP PARALLEL DO PRIVATE(IS)
                DO J=1,1000
                        IS = IS + J
1000            CONTINUE
                print *, IS
```

- Το IS δεν έχει αρχικοποιηθεί μέσα στο loop
- Παρά την αρχικοποίηση, το IS έχει ακαθόριστη τιμή μετά το τέλος του loop

Εντολή firstprivate

- firstprivate: ειδική περίπτωση του private.
  - Το ιδιωτικό αντίγραφο κάθε νήματος αρχικοποιείται με την αντίστοιχη τιμή του νήματος αρχηγού.

```
      program almost_right
      IS = 0
C$OMP PARALLEL DO FIRSTPRIVATE(IS)
      DO J=1,1000
            IS = IS + J
1000  CONTINUE
      print *, IS
```

- Κάθε νήμα έχει ιδιωτικό αντίγραφο του IS με αρχική τιμή 0
- Παρά την αρχικοποίηση, το IS έχει ακαθόριστη τιμή σε αυτό το σημείο

Εντολή lastprivate

- Περνά την τιμή της ιδιωτικής μεταβλητής από την τελευταία επανάληψη που εκτελέστηκε στην καθολική μεταβλητή

```
            program closer
            IS = 0
C$OMP PARALLEL DO FIRSTPRIVATE(IS)
C$OMP+ LASTPRIVATE(IS)
            DO J=1,1000
                  IS = IS + J
1000        CONTINUE
            print *, IS
```

- Κάθε νήμα έχει ιδιωτικό αντίγραφο του IS με αρχική τιμή 0
- Το IS έχει την τιμή που είχε στην τελευταία επανάληψη (δηλ. για J=1000)

Περιβάλλον Δεδομένων

- Παράδειγμα με χρήση των PRIVATE και FIRSTPRIVATE

```
int A, B, C;
A = B = C = 1;
#pragma omp parallel private(B) firstprivate(C)
{
        …
}
```

- Μέσα στο παράλληλο τμήμα :
  - Το “A” είναι κοινό μεταξύ των νημάτων και ίσο με 1
  - Τα “B” και “C” είναι ιδιωτικά σε κάθε νήμα.
    - Το B έχει ακαθόριστη αρχική τιμή
    - Το C έχει αρχική τιμή 1
- Μετά το παράλληλο τμήμα :
  - Οι τιμές των B και C είναι ακαθόριστες

Εντολή default

- Η εξ' ορισμού τιμή είναι DEFAULT(SHARED) (οπότε αν αυτή είναι η επιθυμητή συμπεριφορά δεν χρειάζεται να κάνουμε τίποτε)
- Για να αλλάξουμε την εξ' ορισμού συμπεριφορά:
  - DEFAULT(PRIVATE)
    - κάθε μεταβλητή στο static extent του παράλληλου τμήματος γίνεται ιδιωτική (σαν να είχε καθοριστεί αυτό ρητά με εντολή private)
  - DEFAULT(NONE)
    - καμία εξ' ορισμού συμπεριφορά για τις μεταβλητές στο static extent. Πρέπει να καθοριστεί ρητά η συμπεριφορά για όλες τις μεταβλητές
- Μόνο το Fortran API υποστηρίζει default(private)
- Η C/C++ υποστηρίζει μόνο default(shared) ή default(none)

Εντολή threadprivate

- Κάνει global δεδομένα ιδιωτικά σε κάθε νήμα
  - Fortran: COMMON blocks
  - C: File scope και static variables
- Διαφορετική συμπεριφορά από το PRIVATE
  - Με το PRIVATE οι global μεταβλητές αποκρύπτονται.
  - Το THREADPRIVATE διατηρεί το global scope σε κάθε νήμα
- Οι μεταβλητές threadprivate μπορούν να αρχικοποιηθούν χρησιμοποιώντας εντολές COPYIN ή DATA

Παράδειγμα threadprivate

- Θεωρήστε 2 διαφορετικές συναρτήσεις που καλούνται εντός του ίδιου παράλληλου τμήματος.
- Εξαιτίας της εντολής threadprivate, κάθε νήμα που εκτελεί αυτές τις συναρτήσεις έχει δικό του αντίγραφο του common block /buf/.

```
      subroutine poo
      parameter (N=1000)
      common/buf/A(N),B(N)
C$OMP THREADPRIVATE(/buf/)
      do i=1, N
            B(i)= const* A(i)
      end do
      return
      end
```

```
      subroutine bar
      parameter (N=1000)
      common/buf/A(N),B(N)
C$OMP THREADPRIVATE(/buf/)
      do i=1, N
            A(i) = sqrt(B(i))
      end do
      return
      end
```

Εντολή reduction

- Επηρεάζει στην ουσία τον τρόπο «διαμοίρασης» των μεταβλητών:
  - reduction (op : list)
- Οι μεταβλητές στο “list” πρέπει να είναι shared στο παράλληλο τμήμα στου οποίου το scope βρισκόμαστε.
- Εντός μια δομής parallel ή διαμοίρασης εργασίας:
  - Δημιουργείται τοπικό αντίγραφο κάθε μεταβλητής της λίστας και αρχικοποιείται (ανάλογα με την πράξη “op” π.χ. 0 για “+”)
  - Τα τοπικά αντίγραφα συνδυάζονται ώστε να εκφυλιστούν σε ένα μοναδικό global αντίγραφο στο τέλος της δομής

Παράδειγμα reduction

```
#include <omp.h>
#define NUM_THREADS 2

double func(int i);

void main ()
{
    int i;
    double ZZ, res=0.0;
    omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for reduction(+:res) private(ZZ)
    for (i=0; i< 1000; i++){
        ZZ = func(i);
        res = res + ZZ;
    }
}
```

Παράδειγμα reduction

- Επιστρέψτε στο πρόγραμμα υπολογισμού προσέγγισης του π και αυτή τη φορά, χρησιμοποιήστε private, reduction και μια δομή διαμοίρασης έργου για την παραλληλοποίησή του.
- Πόσο μοιάζει στο αρχικό, ακολουθιακό πρόγραμμα;

Παράδειγμα reduction

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{
    int i; double x, pi, sum = 0.0;
    step = 1.0/(double) num_steps;
    omp_set_num_threads(NUM_THREADS)
#pragma omp parallel for reduction(+:sum) private(x)
    for (i=1;i<= num_steps; i++){
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }
    pi = step * sum;
}
```

- *Το OpenMP προσθέτει 2 - 4 γραμμές κώδικα*