Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Παράλληλος προγραμματισμός:
Σχεδίαση και υλοποίηση παράλληλων προγραμμάτων

Συστήματα Παράλληλης Επεξεργασίας

9ο Εξάμηνο

National Technical University of Athens
CSLab

Σύνοψη παρουσίασης

- Παράλληλες υπολογιστικές πλατφόρμες
  - PRAM: Η ιδανική παράλληλη πλατφόρμα
  - Η ταξινόμηση του Flynn
  - Συστήματα κοινής μνήμης
  - Συστήματα κατανεμημένης μνήμης
- Ανάλυση παράλληλων προγραμμάτων
  - Μετρικές αξιολόγησης επίδοσης
  - Ο νόμος του Amdahl
  - Μοντελοποίηση παράλληλων προγραμμάτων
- Σχεδίαση παράλληλων προγραμμάτων
  - Κατανομή υπολογισμών σε υπολογιστικές εργασίες (tasks)
  - Ορισμός ορθής σειράς εκτέλεσης (χρονοδρομολόγηση)
  - Οργάνωση πρόσβασης στα δεδομένα (συγχρονισμός / επικοινωνία)
  - Ανάθεση εργασιών (απεικόνιση) σε οντότητες εκτέλεσης (processes, threads)

Σύνοψη παρουσίασης

- Παράλληλα προγραμματιστικά μοντέλα
  - Κοινού χώρου διευθύνσεων
  - Ανταλλαγής μηνυμάτων
- Παράλληλες προγραμματιστικές δομές
  - SPMD
  - fork / join
  - task graphs
  - parallel for
- Γλώσσες και εργαλεία
  - POSIX threads, MPI, OpenMP, Cilk, Cuda, Γλώσσες PGAS
- Αλληλεπίδραση με το υλικό
  - Συστήματα κοινής μνήμης
  - Συστήματα κατανεμημένης μνήμης και υβριδικά

«Μιλώντας» παράλληλα

- **Πρώτο βήμα:** σχεδιασμός παράλληλων προγραμμάτων **= σκέψη**
- **Δεύτερο βήμα:** υλοποίηση παράλληλων προγραμμάτων **= ομιλία**
- Όπως και στις φυσικές γλώσσες, η σκέψη και η ομιλία είναι δύο στενά συνδεδεμένες λειτουργίες
- **Ερώτημα 1:** Τι προγραμματιστικές δομές χρειάζομαι για να μιλήσω παράλληλα; Πώς μπορώ **να περιγράψω το γράφο εξαρτήσεων** που δημιούργησα στο βήμα του σχεδιασμού; Πώς **δημιουργώ** και **τερματίζω** εργασίες (**tasks**) και οντότητες εκτέλεσης (**processes, threads**); Τι προγραμματιστικές δομές υπάρχουν για το χαρακτηρισμό των δεδομένων, τον έλεγχο πρόσβασης, το **συγχρονισμό** και την **επικοινωνία** ανάμεσα στις εργασίες/οντότητες εκτέλεσης;
- **Ερώτημα 2**: Τι υποστήριξη χρειάζεται (από την αρχιτεκτονική, το λειτουργικό, τη γλώσσα προγραμματισμού, τη βιβλιοθήκη χρόνου εκτέλεσης της γλώσσας) για να υλοποιηθούν οι παραπάνω δομές με **αποδοτικό** τρόπο;
- 2 κρίσιμα ζητήματα **απόδοσης**
  - Υψηλή επίδοση του παραγόμενου κώδικα **(performance)**
  - Γρήγορη και εύκολη υλοποίηση **(productivity)**

Προγραμματιστικά μοντέλα

- Κοινού χώρου διευθύνσεων
  - Υποστηρίζει κοινά δεδομένα ανάμεσα στα νήματα
  - Επιταχύνει τον προγραμματισμό
  - Μπορεί να οδηγήσει σε δύσκολα ανιχνεύσιμα race conditions
  - Δεν μπορεί να υλοποιηθεί αποδοτικά σε πλατφόρμες που δεν παρέχουν πρόσβαση σε κοινή μνήμη στο υλικό
    - Κατάλληλο για συστήματα κοινής μνήμης
    - Περιορισμένος συνολικός αριθμός νημάτων
- Ανταλλαγής μηνυμάτων
  - Κάθε διεργασία (νήμα) βλέπει μόνο τα δικά της δεδομένα
  - Τα δεδομένα μπορεί να είναι μόνο distributed ή replicated
  - Οδηγεί σε χρονοβόρο προγραμματισμό ακόμα και για απλά προγράμματα (fragmented κώδικας)
  - Μπορεί να υποστηριχθεί από πολύ μεγάλης κλίμακας συστήματα
- Υβριδικό: συνδυασμός των παραπάνω 2

Αρχιτεκτονικές και Προγραμματιστικά Μοντέλα από την οπτική του προγραμματιστή

		Αρχιτεκτονική	
		Κοινής μνήμης (shared memory)	Κατανεμημένης μνήμης (distributed memory)
Προγραμματιστικό μοντέλο	Κοινός χώρος διευθύνσεων (shared address space)	+ Ευκολία υλοποίησης + Προγραμματιστική ευκολία + Υψηλή επίδοση	+ Προγραμματιστική ευκολία - Δυσκολία υλοποίησης - Χαμηλή επίδοση
	Ανταλλαγή μηνυμάτων (message-passing)	+ Ευκολία υλοποίησης + Υψηλή επίδοση - Προγραμματιστική δυσκολία	+ Ευκολία υλοποίησης + Υψηλή επίδοση - Προγραμματιστική δυσκολία

Παράλληλες προγραμματιστικές δομές

- Με αύξουσα ανάγκη σε υποστήριξη από το run-time σύστημα της γλώσσας
  - SPMD
  - parallel for
  - fork / join
  - task graphs

SPMD

- SPMD = Single program multiple data
- Όλες οι διεργασίες εκτελούν το ίδιο τμήμα κώδικα
- Κάθε διεργασία έχει το δικό της σύνολο δεδομένων
- Το αναγνωριστικό της διεργασίας χρησιμοποιείται για να διαφοροποιήσει την εκτέλεσή της
- Αποτελεί ευρύτατα διαδεδομένη προγραμματιστική τεχνική για παράλληλα προγράμματα
- Έχει υιοθετηθεί από το MPI

SPMD

- Ταιριάζει σε εφαρμογές όπου εκτελούνται όμοιες λειτουργίες σε διαφορετικά δεδομένα (data parallelism)
- Απαιτεί σχήματα συγχρονισμού και ανταλλαγής δεδομένων μεταξύ των διεργασιών
  - Σε πρωτόγονο επίπεδο τα παραπάνω παρέχονται από το λογισμικό συστήματος
  - Το μοντέλο SPMD μπορεί να υλοποιηθεί χωρίς επιπρόσθετη υποστήριξη
    - Παρόλα αυτά το MPI διευκολύνει τον προγραμματισμό παρέχοντας μη πρωτόγονες ρουτίνες και ανεξαρτησία από την πλατφόρμα εκτέλεσης
- Θεωρείται κοπιαστική προσέγγιση (non-productive) καθώς τα αφήνει (σχεδόν) όλα στον προγραμματιστή
- Μπορεί να οδηγήσει σε υψηλή επίδοση καθώς ο προγραμματιστής έχει τον (σχεδόν) πλήρη έλεγχο της υλοποίησης

Βήματα υλοποίησης στο σχήμα SPMD

- Αρχικοποίηση
- Λήψη αναγνωριστικού
- [Κατανομή δεδομένων – σε προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων]
- Εκτέλεση του ίδιου κώδικα σε όλους τους κόμβους και διαφοροποίηση ανάλογα με το αναγνωριστικό
  - Εναλλακτικές ροές ελέγχου
  - Ανάληψη διαφορετικών επαναλήψεων σε βρόχο
- Τερματισμός

Παράδειγμα: εξίσωση θερμότητας σε κοινό χώρο διευθύνσεων

```
for (steps=0; steps < T; steps++)
  for (i=1; i<X-1; i++)
    for (j=1; j<Y-1, j++)
       A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] +
                              A[step][i+1][j] + A[step][i][j-1] +
                              A[step][i][j+1])
```

```
myid = get_my_id_from_system();
chunk = X / NUM_PRCS; // υποθέτει X % NUM_PRCS == 0
mystart = myid*chunk + 1;
myend = myend = mystart + chunk;
for (steps=0; steps < T; steps++){
  for (i=mystart; i<myend; i++)
    for (j=1; j<Y-1, j++)
       A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] +
                              A[step][i+1][j] + A[step][i][j-1] +
                              A[step][i][j+1])

  synchronize;
}
```

Homework: εξίσωση θερμότητας με ανταλλαγή μηνυμάτων

```
for (steps=0; steps < T; steps++)
  for (i=1; i<X-1; i++)
    for (j=1; j<Y-1, j++)
       A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] +
                              A[step][i+1][j] + A[step][i][j-1] +
                              A[step][i][j+1])
```

parallel for

- Η παραλληλοποίηση των for-loops αποτελεί σημαντικότατη προσέγγιση στο σχεδιασμό και την υλοποίηση ενός παράλληλου προγράμματος
- Όπως είδαμε, μπορεί να επιτευχθεί στο μοντέλο SPMD αλλά για προγραμματιστική ευκολία έχει ενσωματωθεί σε γλώσσες και εργαλεία:
  - OpenMP
  - Cilk
  - TBBs
  - PGAS
- Χρειάζεται υποστήριξη από το σύστημα
  - Αυτόματη μετάφραση του parallel for σε κώδικα
  - Διαχείριση των δεδομένων
  - Δρομολόγηση των νημάτων/εργασιών
- Είναι ευθύνη του προγραμματιστή να αποφασίσει αν ένα loop είναι παράλληλο

automatic parallelization

- Η εύρεση των παράλληλων for αποτελεί τον κυριότερο στόχο της αυτόματης παραλληλοποίησης
- Ειδικά περάσματα optimizing compilers αναζητούν:
  - Αν ένα loop είναι παράλληλο
  - Αν αξίζει να παραλληλοποιηθεί
- Η “απόδειξη” της παραλληλίας ενός loop είναι δύσκολη και βασίζεται στα λεγόμενα dependence tests

dependence analysis

- Πότε ένα loop είναι παράλληλο;
  - Όταν δεν υπάρχουν εξαρτήσεις ανάμεσα στις επαναλήψεις του
- Τέλεια φωλιασμένοι βρόχοι:

```
for i1 = 1 to U1
    for i2 = 1 to U2
        …
        for in = 1 to Un
            …
```

- Διανύσματα εξαρτήσεων **d** εκφράζουν τις εξαρτήσεις σε κάθε επίπεδο του φωλιάσματος
- Πίνακας εξαρτήσεων D, περιέχει κατά στήλες τα διανύσματα εξάρτησης
- Διανύσματα απόστασης (distance vectors)
  - Στοιχεία του πίνακα D d_{ij} είναι σταθεροί ακέραιοι αριθμοί
- Διανύσματα κατεύθυνσης (direction vectors):
  - > (υπάρχει εξάρτηση από προσβάσεις στη μνήμη προηγούμενων επαναλήψεων)
  - < (υπάρχει εξάρτηση από προσβάσεις στη μνήμη επόμενων επαναλήψεων)
  - * (υπάρχει εξάρτηση από προσβάσεις στη μνήμη προηγούμενων ή επόμενων επαναλήψεων)

Παράδειγμα: εξίσωση θερμότητας

```
for (steps=0; steps < T; steps++)
  for (i=1; i<X-1; i++)
    for (j=1; j<Y-1, j++)
       A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] +
                             A[step][i+1][j] + A[step][i][j-1] +
                             A[step][i][j+1])
```

$$D = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Παράδειγμα: Floyd-Warshall

```
for (k=0; k<N; k++)
  for (i=0; i<N; i++)
    for (j=0; j<N, j++)
       A[(k+1)%2][i][j] = min(A[k%2][i][j], A[k%2][i][k] + A[k%2][k][j])
```

$$D = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & * \\ 0 & * & 0 \end{bmatrix}$$

Κανόνες παραλληλίας loop

- Ένας βρόχος στο επίπεδο *i* ενός φωλιασμένου βρόχου είναι παράλληλος αρκεί να ισχύει οτιδήποτε από τα παρακάτω:
  1. το *i*-οστό στοιχείο **ΟΛΩΝ** των διανυσμάτων εξάρτησης είναι **0**
  2. **ΟΛΑ** τα υποδιανύσματα από 0 εως *i-1* να είναι **λεξικογραφικά ΘΕΤΙΚΑ** (= το πρώτο μη μηδενικό στοιχείο είναι θετικό)
- **Επισήμανση**: Για τον εξωτερικότερο βρόχο (*i=0*) μπορεί να εφαρμοστεί μόνο ο πρώτος κανόνας (δεν υπάρχουν υποδιανύσματα από 0 εως -1)
- **Άσκηση:** Ποια loops παραλληλοποιούνται στην εξίσωση θερμότητας και στον αλγόριθμο FW?

Παράδειγμα: LU decomposition

```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      A[i][k] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - A[i][k] * A[k][j];
   }
```

Παράδειγμα: LU decomposition

```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - A[i][k] / A[k][k] * A[k][j];
   }
```

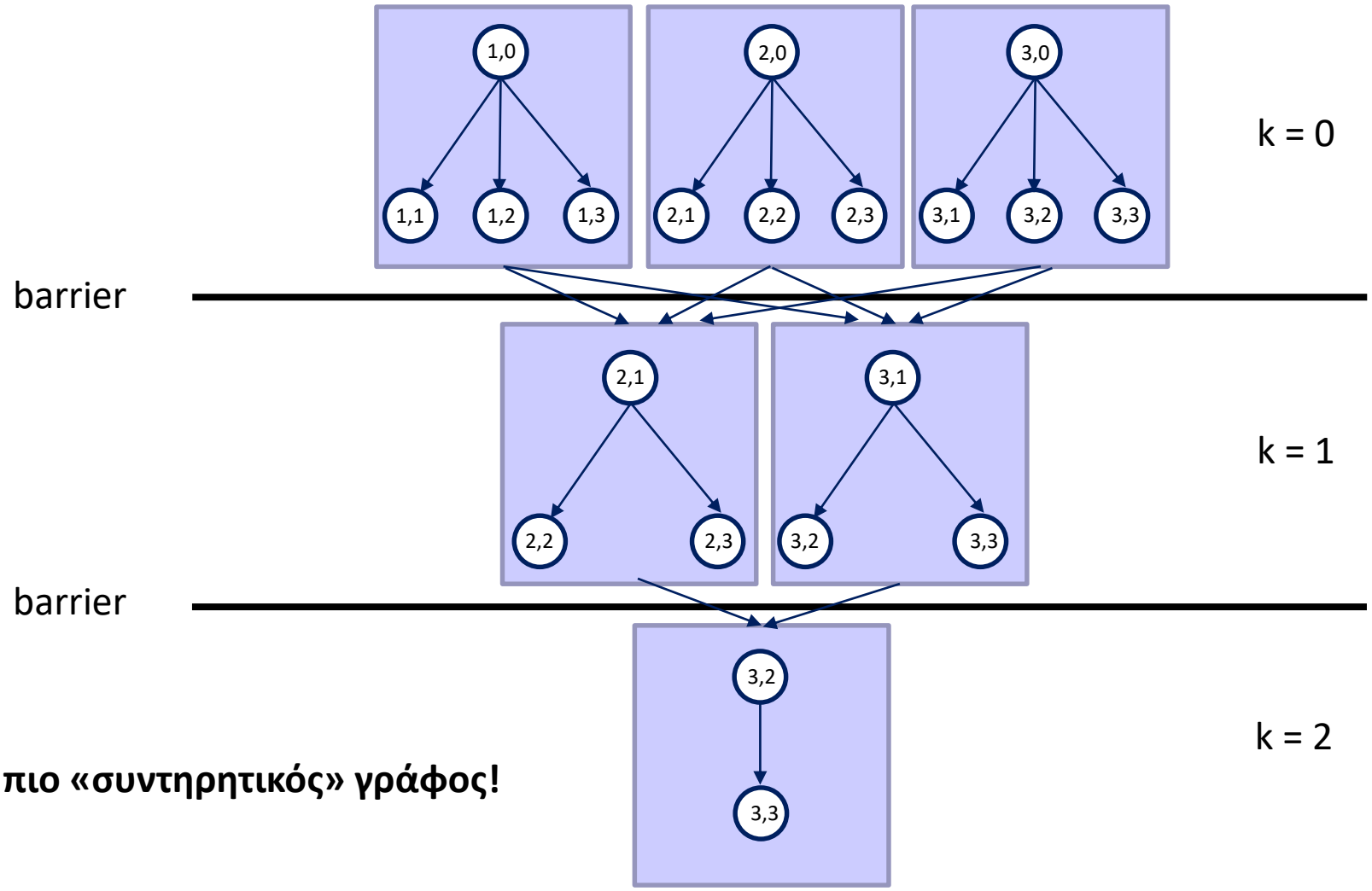
$$D = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & < & < \\ 0 & < & < & 0 \end{bmatrix}$$

Παράδειγμα: LU decomposition

```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   parallel for(i = k+1; i < N; i++){
      A[i][k] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - A[i][k] * A[k][j];
   }
```

Τυπικά το parallel for επιβάλει καθολικό συγχρονισμό στο τέλος του (barrier)

Παράδειγμα: Παραλληλοποίηση LU με parallel for

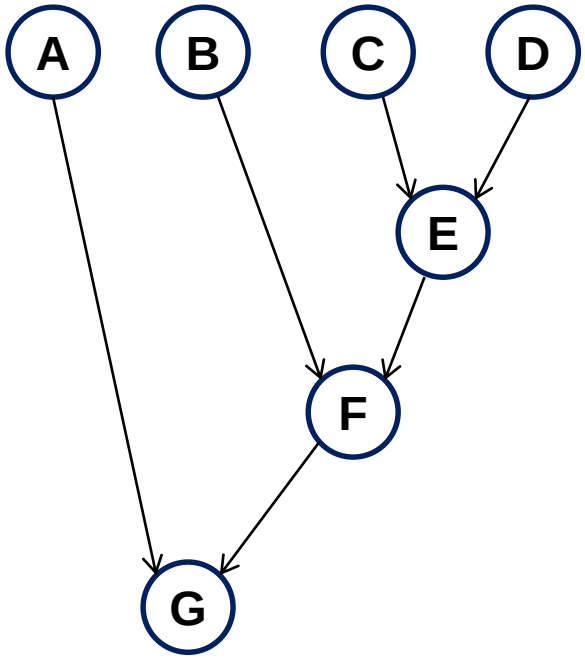


πιο «συντηρητικός» γράφος!

fork / join

- Αφορά εφαρμογές με δυναμική ανάγκη για δημιουργία / τερματισμό tasks
- Τα tasks δημιουργούνται (fork) και τερματίζονται (join) δυναμικά
- Π.χ. αλγόριθμος σχεδιασμένος με την Divide and Conquer στρατηγική
- OpenMP tasks
  - `#pragma omp task`
  - `#pragma omp taskwait`
  - `#pragma omp taskgroup`
- Cilk
  - `spawn`
  - `sync`

Παράδειγμα fork - join




```
#pragma omp parallel
{
   #pragma omp single
   {
      #pragma omp task A();
      #pragma omp task if (0)
      {
            #pragma omp task B();
            #pragma omp task if (0)
            {
               #pragma omp task C();
               D();
               #pragma omp taskwait
               E();
            }
            #pragma omp taskwait
            F();
      }
      #pragma omp taskwait
      G();
    }
}
```

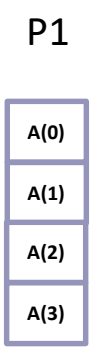
Χρονοδρομολόγηση εργασιών

- Αριθμός εργασιών *T*
- Αριθμός διεργασιών / νημάτων *P*
- Κάθε εργασία μπορεί:
  - Να παράξει άλλες εργασίες
  - Να περιμένει τα παιδιά της
- Στόχοι χρονοδρομολογητή:
  - Ισοκατανομή φορτίου
  - Αποδοτική χρήση των πόρων
  - Μικρή επιβάρυνση

Χρονοδρομολόγηση εργασιών

- Δύο βασικές στρατηγικές:
  - **Work sharing:** Όταν δημιουργούνται νέες εργασίες ο ΧΔ προσπαθεί να τις "στείλει" σε ανενεργούς επεξεργαστές.
  - **Work stealing:** Οι ανενεργοί επεξεργαστές προσπαθούν να "κλέψουν" εργασίες.
- Γενικά προτιμάται η τακτική work stealing
  - καλύτερο locality
  - μικρότερη επιβάρυνση συγχρονισμού
  - βέλτιστη θεωρητικά όρια ως προς χρόνο, χώρο [Blumofe and Leiserson '99]

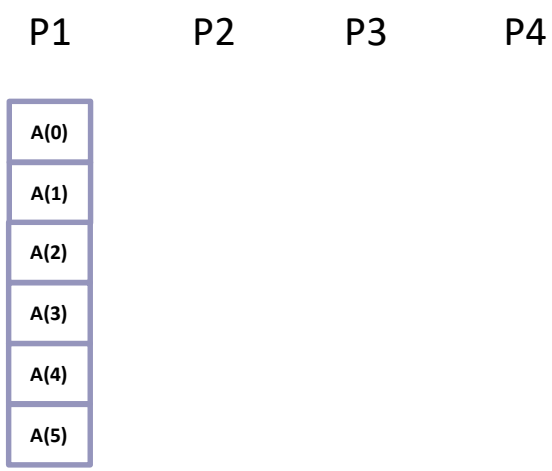
work stealing




```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

task creation

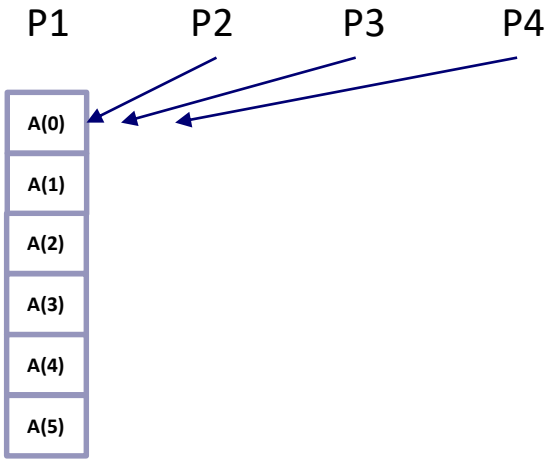
work stealing



task creation

```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

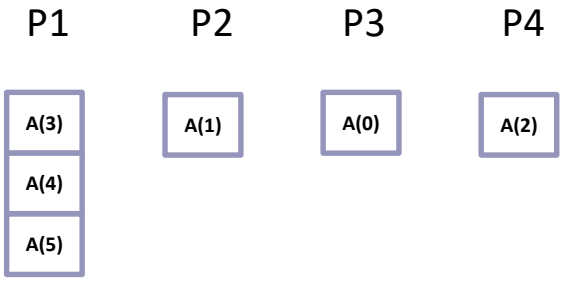
work stealing




```
#pragma omp parallel
{
    #pragma omp single
    {
        for (i = 0; i < n; i++)
        #pragma omp task A(i);
        #pragma omp taskwait
    }
}
```

work stealing: τυχαία επιλογή μη κενής ουράς (εδώ υπάρχει μόνο μία)
ανάγκη συγχρονισμού για ορθή αφαίρεση
αφαίρεση από την κορυφή (παλαιότερες εργασίες –σεβασμός τοπικότητας της διεργασίας)

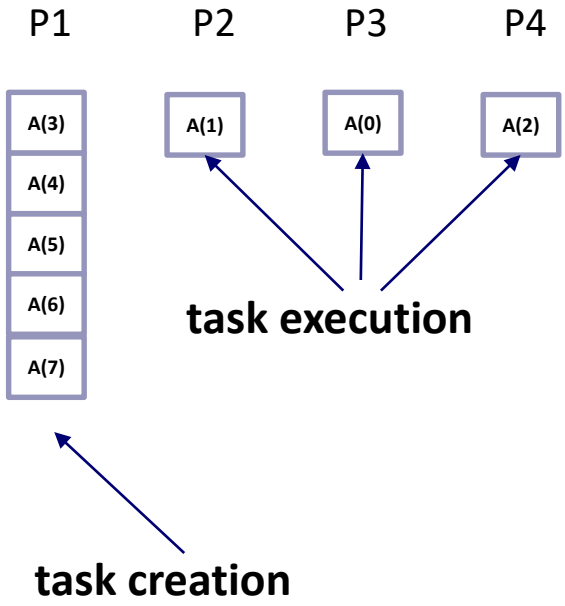
work stealing




```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

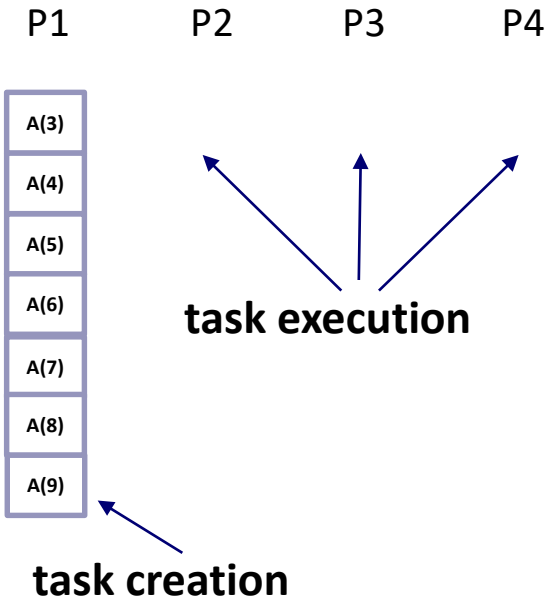
work stealing: τυχαία επιλογή μη κενής ουράς (εδώ υπάρχει μόνο μία)
ανάγκη συγχρονισμού για ορθή αφαίρεση
αφαίρεση από την κορυφή (παλαιότερες εργασίες –σεβασμός τοπικότητας της διεργασίας)

work stealing



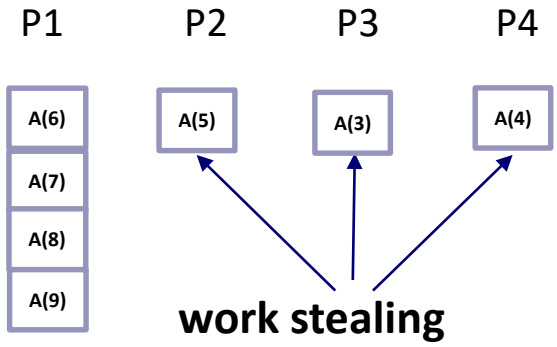

```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

work stealing



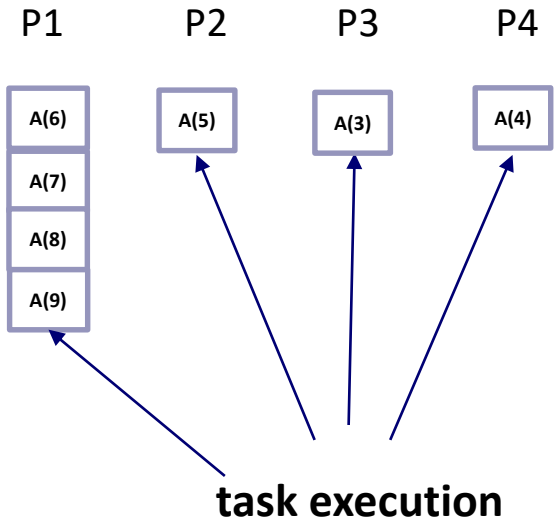

```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

work stealing



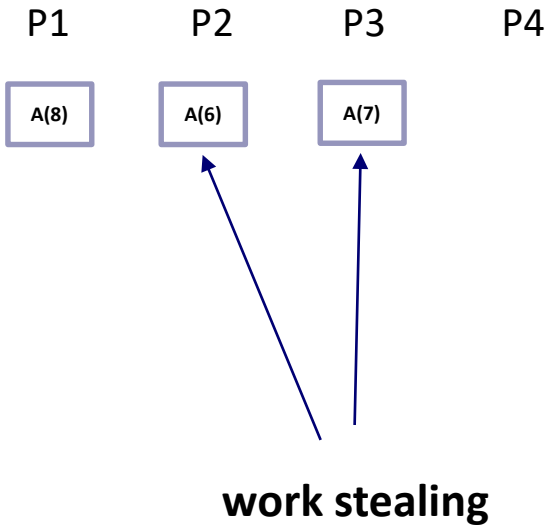

```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

work stealing



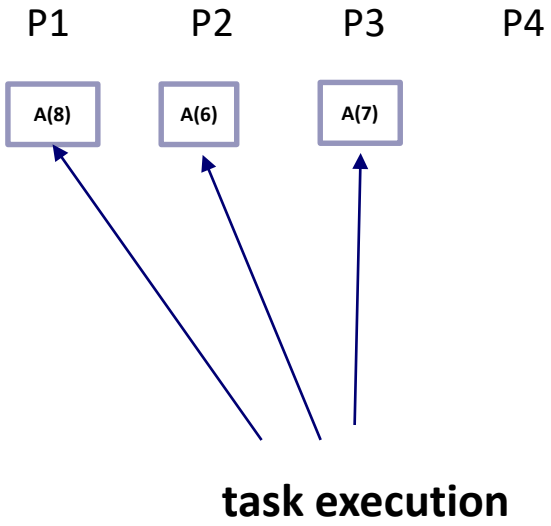

```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

work stealing



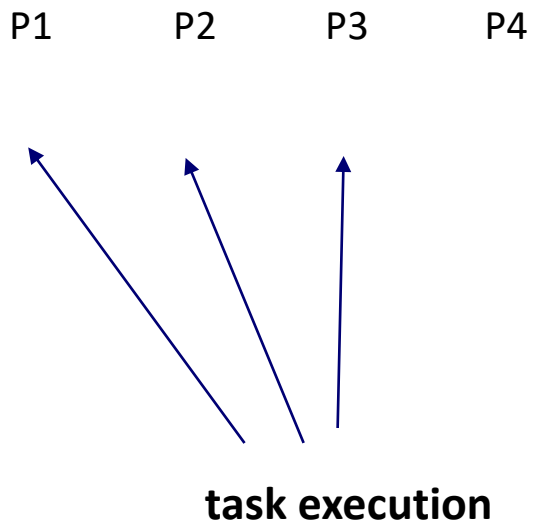

```
#pragma omp parallel
{
    #pragma omp single
    {
        for (i = 0; i < n; i++)
        #pragma omp task A(i);
        #pragma omp taskwait
    }
}
```

work stealing




```
#pragma omp parallel
{
   #pragma omp single
   {
      for (i = 0; i < n; i++)
      #pragma omp task A(i);
      #pragma omp taskwait
   }
}
```

work stealing




```
#pragma omp parallel
{
    #pragma omp single
    {
        for (i = 0; i < n; i++)
        #pragma omp task A(i);
        #pragma omp taskwait
    }
}
```

work stealing

Παράλληλη δημιουργία tasks

```
#pragma omp parallel
{
    #pragma omp for
    for (i = 0; i < n; i++)
        #pragma omp task A(i);
    #pragma omp taskwait
}
```

work stealing

- Για να υποστηρίζεται η κλοπή εργασίων πρέπει η κάθε εργασία να μπορεί να μεταφερθεί σε διαφορετικό επεξεργαστή
- Τι χρειάζεται η κάθε εργασία για να εκτελεστεί;
  - Κώδικα (πχ δείκτη σε συνάρτηση)
  - Θέση στον κώδικα
  - Δεδομένα:
    - Στοίβα
    - Σωρός

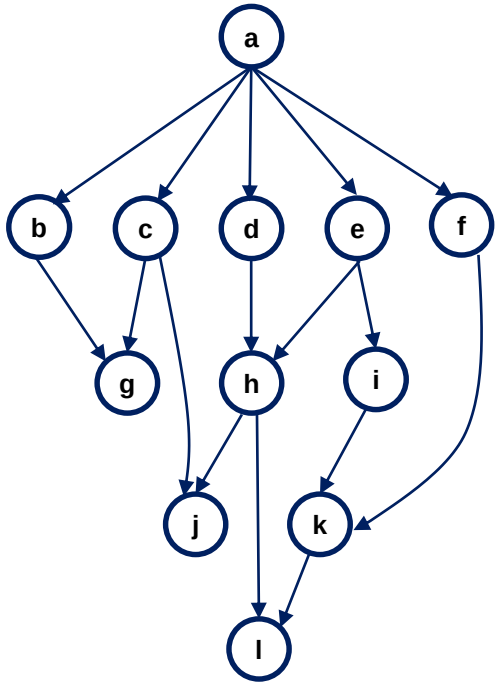
Παράδειγμα: LU decomposition με tasks

```
//LU decomposition kernel
for (k = 0; k < N-1; k++) {
    for(i = k+1; i < N; i++)
      task {
         A[i][k] = A[i][k] / A[k][k];
         for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - A[i][k] * A[k][j];
      }
    taskwait
}
```

Ποιος γράφος εκτελείται σε αυτή την περίπτωση;

Task graphs

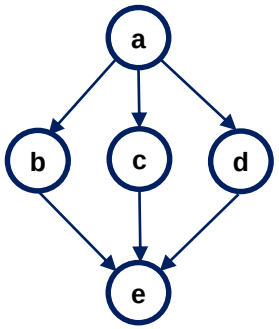
- Η fork-join δομή δεν μπορεί να περιγράψει όλα τα task graphs. Π.χ. η υλοποίηση του παρακάτω γράφου με fork-join οδηγεί σε πιο περιοριστική εκτέλεση.



- Όταν ένα task graph είναι γνωστό στατικά, τότε μπορούμε να το περιγράψουμε ρητά

Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

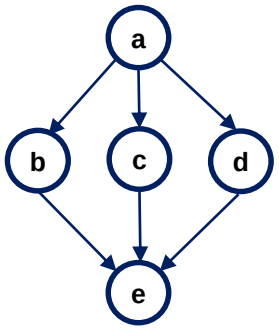


Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

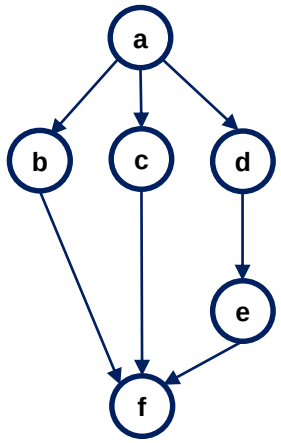
```
a();
task b();
task c();
task d();
taskwait();
task d();
```

OK



Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

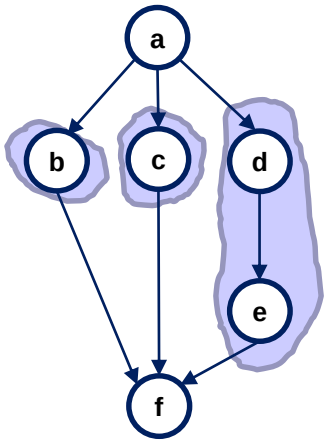


Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμού για κάθε task (ή ομάδα tasks)

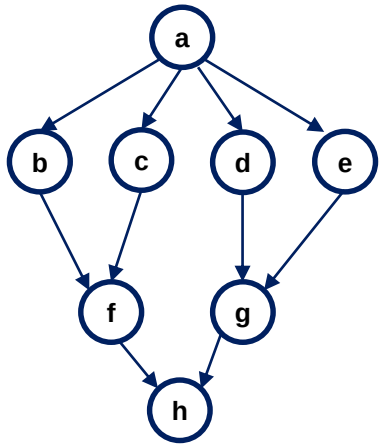
```
a();
task b();
task c();
task {d(); e()}
taskwait();
task f();
```

OK



Περιορισμοί του μοντέλου fork-join

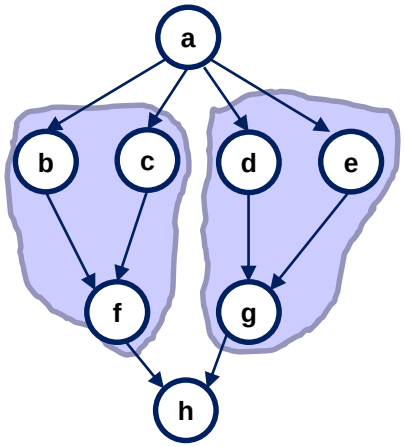
- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)



Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

```
a();
task {
  task b();
  task c();
  taskwait;
  f();
}
task {
  task d();
  task e();
  taskwait;
  g();
}
taskwait;
h();
```

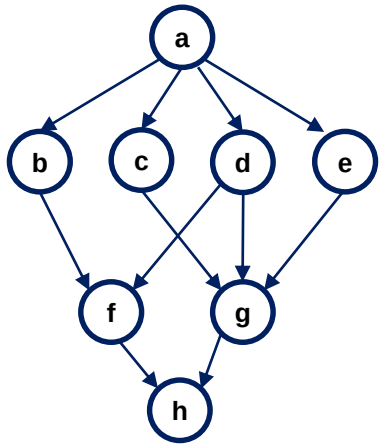


OK

Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

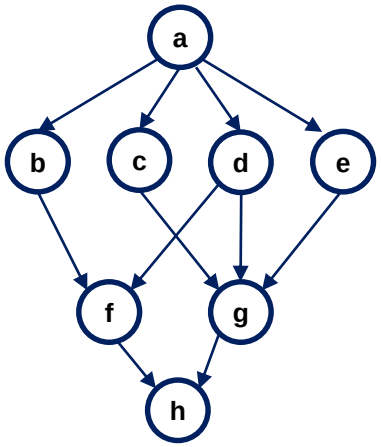
?



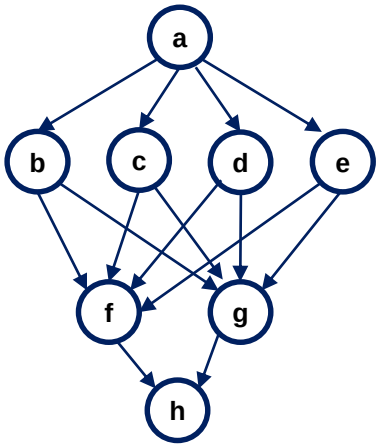
Περιορισμοί του μοντέλου fork-join

- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

```
a();
task b();
task c();
task d();
task e();
taskwait;
task f();
task g();
taskwait;
h();
```



Αρχικός γράφος



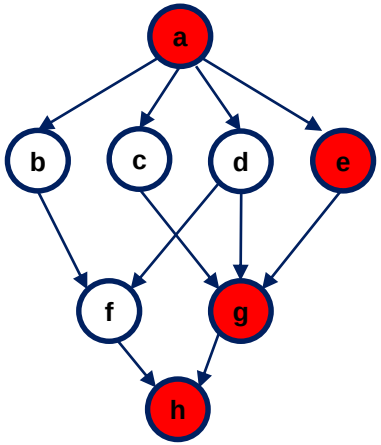
Γράφος fork/join

Περιορισμοί του μοντέλου fork-join

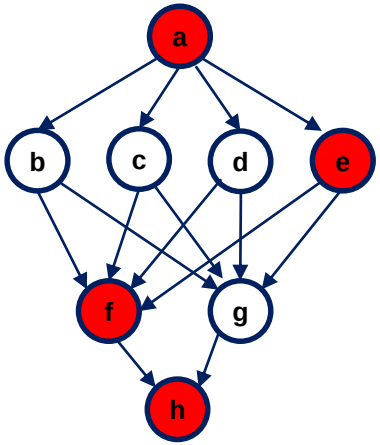
- Η fork-join δομή δεν είναι κατάλληλη για την περιγραφή οποιουδήποτε task graph
- Επιβάλλει να υπάρχει ένα σημείο συγχρονισμό για κάθε task (ή ομάδα tasks)

```
a();
task b();
task c();
task d();
task e();
taskwait;
task f();
task g();
taskwait;
h();
```

Αν a = b = c = d = g = 1,
e = f = 100



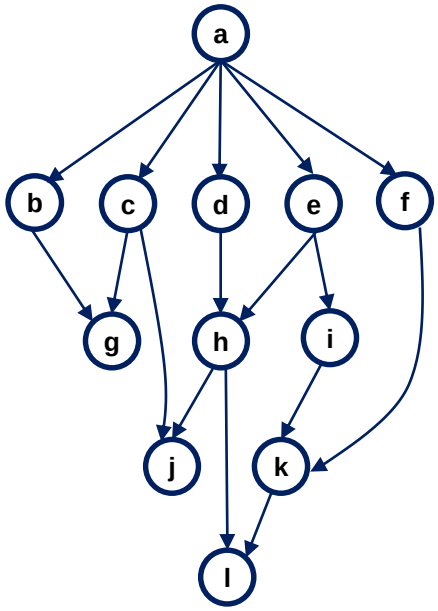
Αρχικός γράφος
Κρίσιμο μονοπάτι: **103**



Γράφος fork/join
Κρίσιμο μονοπάτι: **202**

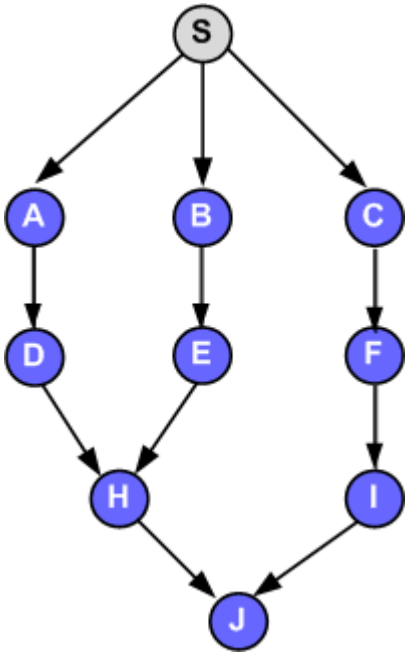
Περιορισμοί του μοντέλου fork-join

- Υπάρχουν πολλές περιπτώσεις γράφων με σύνθετες εξαρτήσεις, όπου το fork-join μοντέλο οδηγεί σε περιοριστική εκτέλεση
- Όταν ένα task graph είναι γνωστό στατικά, τότε μπορούμε να το περιγράψουμε άμεσα
- Υπάρχουν εργαλεία παράλληλου προγραμματισμού που επιτρέπουν την έκφραση task graphs με δήλωση του γράφου
  - Τα tasks δηλώνονται ως κορυφές του γράφου
  - Οι εξαρτήσεις μεταξύ των tasks δηλώνονται ως ακμές του γράφου
- Η βιβλιοθήκη των Threading Building Blocks προσφέρει τη διεπαφή *flow graph* για την παραλληλοποίηση οποιουδήποτε task graph



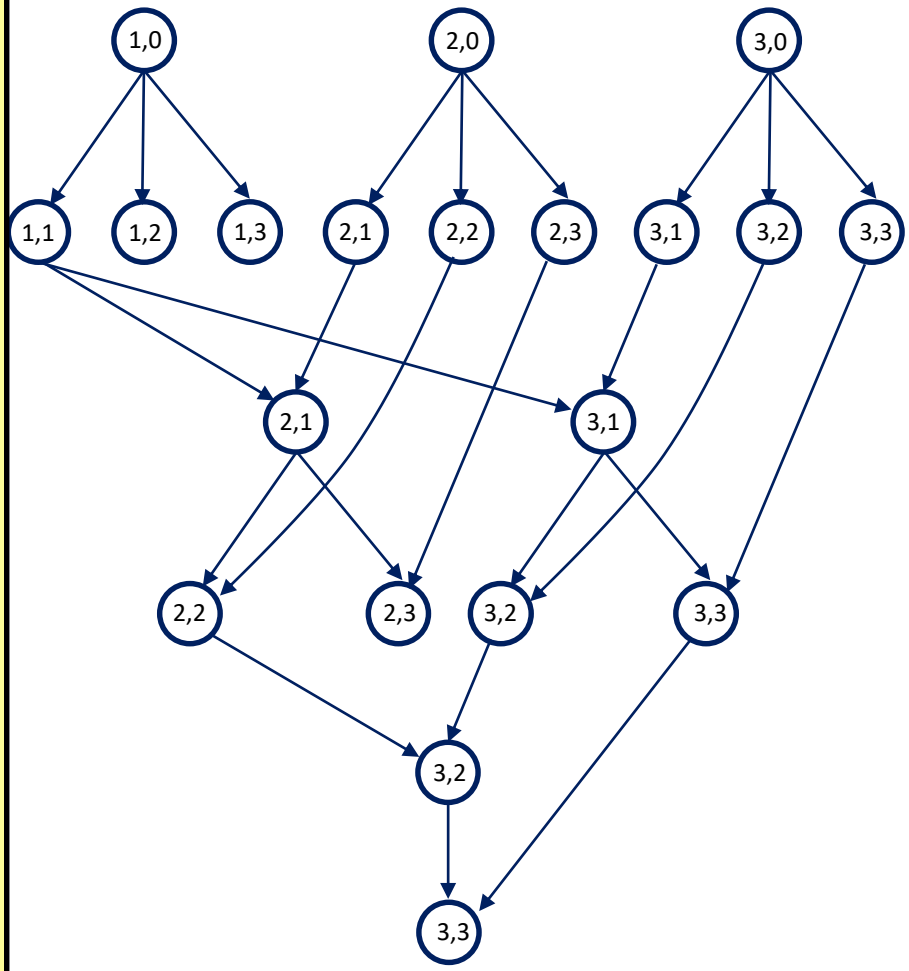
```
graph g;
source_node s;  //source node

//each node executes a different function body
node a( g, body_A() );
node b( g, body_B() );
node c( g, body_C() );
node d( g, body_D() );
node e( g, body_E() );
node f( g, body_F() );
node h( g, body_H() );
node i( g, body_I() );
node j( g, body_J() );

//create edges
make_edge( s, a );
make_edge( s, b );
make_edge( s, c );
make_edge( a, d );
make_edge( b, e );
make_edge( c, f );
make_edge( d, h );
make_edge( e, h );
make_edge( f, i );
make_edge( h, j );
make_edge( i, j );

s.start(); // start parallel execution of task graph
g.wait_for_all();  //wait for all to complete
```

Παράδειγμα task graph



Παράδειγμα: Task graph για LU decomposition με Threading Building Blocks (ψευδοκώδικας)

```
namespace tbb::flow
graph g;
broadcast_node s;  //source node
                   //to start execution
for (k = 0 ; k < N - 1 ; k++)
  for (i = k+1 ; i < N ; i++) {
      node(k,i,j) = new continue_node {g,
          A[i][k] = A[i][k]/A[k][k];
      }
      if (k == 0)
         make_edge(s, node(k,i,j));
      else {
         make_edge(node(k-1,i,k), node(k,i,k));
         make_edge(node(k-1,k,k), node(k,i,k));
      }
      for(j = k+1 ; j < N ; j++) {
         node(k,i,j) = new continue_node {g,
            A[i][j] -= A[i][k]*A[k][j];
         }
         make_edge(node(k,i,k), node(k,i,j)
         if (k > 0)
            make_edge(node(k-1,i,j), node(k,i,j));
      }
    }
 s.try_put();       //fire!
 g.wait_for_all();  //and wait for everything
```



POSIX threads

- Αποτελούν πρωτόγονο τρόπο πολυνηματικού προγραμματισμού για systems programming
- Συνδυαζόμενα με TCP/IP sockets αποτελούν την πιο βασική πλατφόρμα εργαλείων για παράλληλο προγραμματισμό
- SPMD ή Master / Slave
- + : ο χρήστης έχει απόλυτο έλεγχο της εκτέλεσης
- - : ιδιαίτερα κοπιαστικός και επιρρεπής σε σφάλματα τρόπος προγραμματισμού

MPI

- Μοντέλο ανταλλαγής μηνυμάτων για συστήματα κατανεμημένης μνήμης
- Αποτελεί τον de facto τρόπο προγραμματισμού σε υπερυπολογιστικά συστήματα
- SPMD (και Master / Slave)
- Απλοποιεί την υλοποίηση της επικοινωνίας σε σχέση με τη βιβλιοθήκη των sockets καθώς υποστηρίζει μεγάλο αριθμό από χρήσιμες ρουτίνες
- Βελτιστοποιεί την επικοινωνία (κυρίως σε collective επικοινωνία)
- Ο χρήστης αναλαμβάνει το διαμοιρασμό των δεδομένων και την επικοινωνία μέσω μηνυμάτων
  - “fragmented” τρόπος προγραμματισμού

OpenMP

- Προγραμματιστικό εργαλείο που βασίζεται σε οδηγίες (directives) προς το μεταγλωττιστή
- Αφορά κατά κύριο λόγο αρχιτεκτονικές κοινής μνήμης
- Μπορεί να αξιοποιηθεί για την εύκολη παραλληλοποίηση ήδη υπάρχοντος σειριακού κώδικα
- Παρέχει ευελιξία στο προγραμματιστικό στυλ:
  - SPMD
  - Master / Workers
  - parallel for
  - Fork / Join (ενσωμάτωση των tasks, 2008)

Cilk

- Πολυνηματικός προγραμματισμός για αρχιτεκτονικές κοινής μνήμης
- Επεκτείνει τη C με λίγες επιπλέον λέξεις κλειδιά
- Κάθε πρόγραμμα γραμμένο σε Cilk έχει ορθή σειριακή σημασιολογία (μπορεί να εκτελεστεί σωστά σειριακά)
- Σχεδιασμός προγραμμάτων με χρήση αναδρομής
- Σχήμα Fork / Join
- Υποστήριξη parallel for
- Βασικές λέξεις κλειδιά: cilk, spawn, sync

C

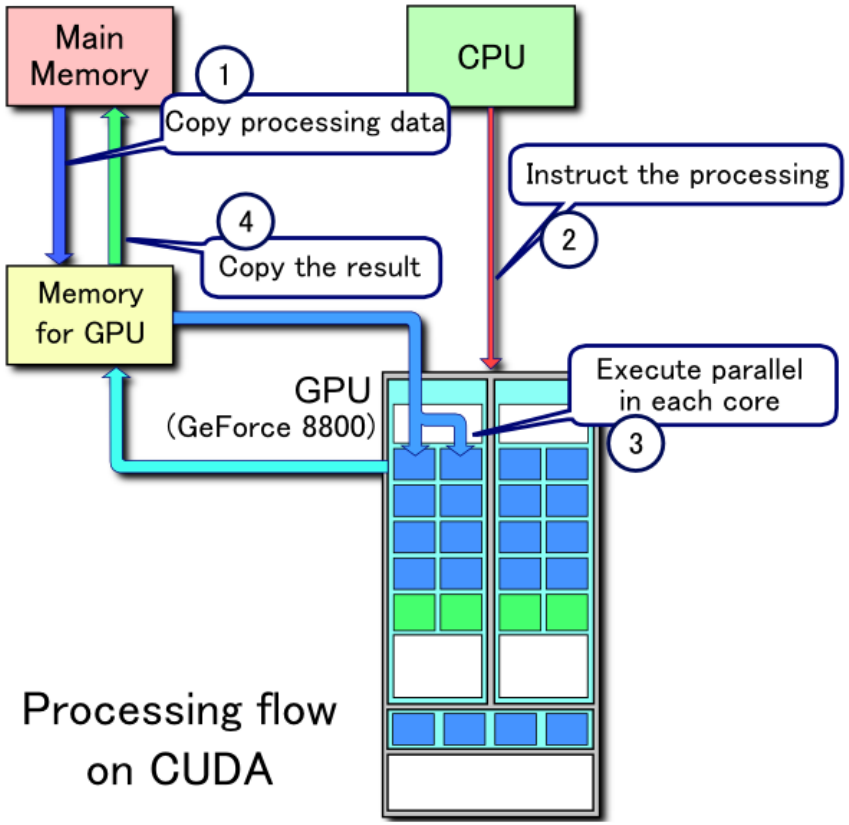
```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

Cilk

```
cilk void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    spawn vadd (A, B, n/2;
    spawn vadd (A+n/2, B+n/2, n-n/2;
  } sync;
}
```

CUDA

- **C**ompute **U**nified **D**evice **A**rchitecture
  - Παράλληλη αρχιτεκτονική που προτάθηκε από την NVIDIA
  - Βασίζεται σε GPUs (GP-GPUs = General Purpose-Graphical processing Units)
- Προγραμματισμός: C extensions για αρχιτεκτονικές CUDA
- Ανάθεση υπολογιστικά απαιτητικών τμημάτων του κώδικα στην κάρτα γραφικών
- GPU = accelerator
- massively data parallel
- Manycore system



Προγραμματισμός σε περιβάλλον CUDA

- SIMT = **S**ingle **I**nstruction **M**ultiple Threads
- Τα threads δρομολογούνται σε ομάδες που λέγονται warps
- Το κόστος δρομολόγησης των threads είναι μηδενικό
- Σε κάθε κύκλο όλα τα threads εκτελούν την ίδια εντολή
  - Εκτελούνται όλα τα control paths
  - Τα μη έγκυρα control paths ματαιώνονται (aborted) στο τέλος
- Οι αλγόριθμοι πρέπει να σχεδιάζονται με data parallel λογική
  - Τα branches και ο συγχρονισμός κοστίζουν
- Η χωρική τοπικότητα αξιοποιείται κατά μήκος των threads και όχι εντός μιας CPU όπως γίνεται στους συμβατικούς επεξεργαστές
- Αρκετά «σκιώδη» ζητήματα επηρεάζουν την επίδοση και χρήζουν ιδιαίτερης προσοχής
  - Παραλληλισμός
  - Πρόσβαση στα δεδομένα
  - Συγχρονισμός

Γλώσσες PGAS

- **P**artitioned **G**lobal **A**ddress **S**pace
- Προσπαθεί να έχει τα θετικά και από τους δύο κόσμους:
  - Προγραμματιστική ευκολία όπως το μοντέλο του κοινού χώρου διευθύνσεων
  - Επίδοση και κλιμακωσιμότητα όπως το μοντέλο της ανταλλαγής μηνυμάτων
- Υποθέτει καθολικό χώρο διευθύνσεων (global address space) που κατανέμεται (partitioned) στις τοπικές μνήμες των επεξεργαστών ενός συστήματος κατανεμημένης μνήμης
- Απαιτεί πολύ ισχυρό run-time σύστημα
- Για προσβάσεις σε δεδομένα που βρίσκονται σε απομακρυσμένη μνήμη χρησιμοποιεί 1-sided communication
- Υλοποιήσεις:
  - UPC
  - Fortress
  - Co-array Fortrean
  - X10
  - Chapel

Languages and tools

	SPMD	Task graph	parallel for	fork / join
MPI	√			
OpenMP	√		√	√
Cilk			√ (Cilk++)	√
TBB	√	√	√	√
CUDA	√			

Βελτιστοποίηση: μερικές παρατηρήσεις

- Σχεδιασμός:
  - Η πολυπλοκότητα (συνολικός αριθμός πράξεων - work) παίζει πρωταρχικό ρόλο (μην ξεχνάμε τα βασικά!)
  - Οι εξαρτήσεις ανάμεσα στα tasks (task graph) φανερώνει τις προοπτικές του παραλληλισμού
  - Η πρόσβαση στη μνήμη και η επικοινωνία κοστίζουν (πολύ!). Προσοχή στο operational intensity και τα data movements
- Υλοποίηση και εκτέλεση:
  - Είναι σημαντικό κατά την υλοποίηση να μη χάνουμε τον παραλληλισμό που αναδείξαμε κατά το σχεδιασμό
  - Προσοχή στον τρόπο πρόσβασης στα δεδομένα! (locality awareness)
    - Μείωση του reuse distance ανάμεσα σε δύο προσβάσεις σε μία θέση μνήμης
    - πρόσδεση (pinning) threads/processes σε πυρήνες
    - tradeoff ανάμεσα σε στατικά (λιγότερος παραλληλισμός – καλύτερο locality) και δυναμικά (περισσότερος παραλληλισμός – χειρότερο locality) allocations
  - Γιατί το πρόγραμμά μου δεν κλιμακώνει; Θυμηθείτε το Νόμο του Amdahl!

Ερωτήσεις;