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
- Οργάνωση πρόσβασης στα δεδομένα
- Παράλληλα προγραμματιστικά μοντέλα
  - Κοινού χώρου διευθύνσεων
  - Ανταλλαγής μηνυμάτων

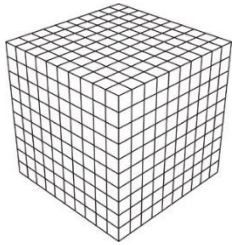
Σύνοψη παρουσίασης

- Σχεδίαση και υλοποίηση παράλληλων προγραμμάτων
  - Παραλληλισμός σε επίπεδο εργασίας (task parallelism)
  - **Παραλληλοποίηση σε επίπεδο δεδομένων (data parallelism)**
  - **Παραλληλοποίηση σε επίπεδο βρόχου (loop parallelism)**
  - **Παραλληλοποίηση σε επίπεδο συνάρτησης (function parallelism)**
- Γλώσσες και εργαλεία
  - POSIX threads, MPI, OpenMP, Cilk, Cuda, Γλώσσες PGAS
- Αλληλεπίδραση με το υλικό
  - Συστήματα κοινής μνήμης
  - Συστήματα κατανεμημένης μνήμης και υβριδικά

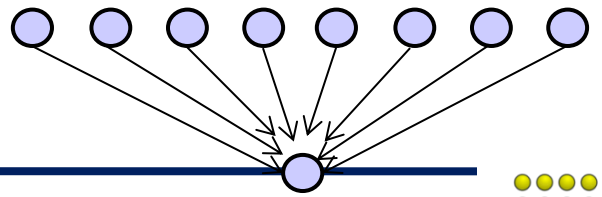
$$P_{\text{Jac}} = \begin{pmatrix} -\frac{1749}{2} & \frac{2685}{2} & 73 & \frac{3842}{5} & -\frac{40721}{4} & -\frac{1054231}{250} & \frac{730629}{100} & -\frac{128827971}{6250} \\ \frac{1490}{3} & -\frac{691}{6} & \frac{2021}{12} & -\frac{26039}{150} & \frac{196313}{30} & -\frac{2540087}{1875} & -\frac{14477359}{1500} & \frac{487255997}{31250} \\ \frac{747}{2} & -\frac{2115}{2} & -\frac{199}{2} & -291 & \frac{24885}{4} & \frac{175583}{50} & -\frac{109947}{20} & \frac{14131653}{1250} \\ \frac{1391}{3} & -\frac{3761}{6} & -\frac{8363}{12} & -\frac{9011}{30} & \frac{35339}{6} & \frac{1302962}{375} & \frac{307709}{300} & \frac{52789903}{6250} \\ \frac{620}{3} & -\frac{815}{6} & -\frac{9395}{12} & -\frac{209503}{150} & \frac{63548}{15} & \frac{2706476}{1875} & \frac{4732207}{1500} & \frac{386472719}{31250} \\ \frac{1031}{6} & -\frac{674}{3} & -\frac{5911}{12} & -\frac{13333}{30} & \frac{10199}{12} & \frac{1417027}{750} & \frac{232033}{75} & \frac{13882747}{3125} \\ \frac{343}{6} & \frac{80}{3} & -\frac{4019}{12} & -\frac{28789}{150} & \frac{19211}{60} & -\frac{2996549}{3750} & \frac{1232029}{375} & \frac{54383186}{15625} \\ \frac{200}{3} & -\frac{109}{3} & -\frac{145}{6} & -\frac{15969}{75} & \frac{3473}{15} & \frac{425546}{1875} & -\frac{56839}{750} & \frac{44325857}{15625} \end{pmatrix}$$

Data parallelism pattern

- Ο σχεδιασμός και η υλοποίηση θέτουν στο κέντρο τα δεδομένα του προβλήματος
- Αποτελεί μία προφανή προσέγγιση στα λεγόμενα **“embarrassingly parallel”** προβλήματα (π.χ. συμπίεση Ν αρχείων, επίλυση Ν ανεξάρτητων συστημάτων)
- Ταιριάζει στη λογική **map-reduce** (ανεξάρτητες εργασίες σε διαφορετικά δεδομένα και συνδυασμός στο τέλος)
- Ταιριάζει σε αλγορίθμους που χειρίζονται κανονικές δομές δεδομένων (π.χ. regular computational grids, αλγεβρικοί πίνακες)
  - Πολλαπλασιασμός πινάκων (γενικά βασικές αλγεβρικές ρουτίνες)
  - Επίλυση γραμμικών συστημάτων
  - “Stencil computations”, Game of Life



- Η διαδικασία ξεκινά από την **κατανομή** των δεδομένων σε εργάτες (δεν ασχολούμαστε με εργασίες εδώ) → **κατάλληλη προσέγγιση για προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων**
- Η υλοποίηση τυπικά βασίζεται στις προσεγγίσεις Single Program Multiple Data (**SPMD**) ή **map-reduce**



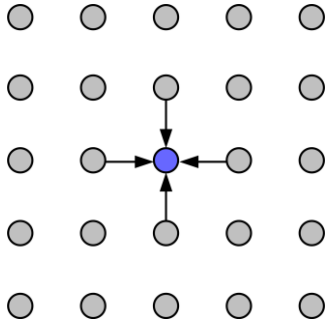
Βήματα σχεδιασμού και υλοποίησης

- **Βήμα 1:** Κατανομή δεδομένων σε εργάτες
  - Δεδομένα εισόδου, π.χ.
    - Πίνακες A_i και διανύσματα b_i για την επίλυση συστημάτων $A_i x = b_i$
    - Αρχικές συνθήκες για πειράματα monte carlo
    - Πολλαπλά αρχεία ή μέρη ενός (πολύ μεγάλου) αρχείου προς επεξεργασία (π.χ. αναζήτηση, συμπίεση, κωδικοποίηση)
  - Δεδομένα εξόδου, π.χ.
    - Γραμμές/στήλες/στοιχεία/μπλοκ
      - Του πίνακα C στον πολλαπλασιαμό πινάκων C = A x B
      - Υπολογιστικού πλέγματος στην εξίσωση θερμότητας
    - Κορυφές ενός γράφου που θέλουμε να υπολογίσουμε ελάχιστα μονοπάτια (π.χ. με τον αλγόριθμο Bellman-Ford)
  - Συνήθως ακολουθούνται κανόνες "**computer-owns**" ή "**owner computes**", δηλαδή αποφεύγονται οι πολλοί εγγραφείς σε κάποιο δεδομένο
  - Αν χρειάζεται να υπάρχουν πολλοί εγγραφείς καταφεύγουμε σε reduction ή (πιο σπάνια) σε κάποιο σχήμα κατανεμημένου κρίσιμου τμήματος
- **Βήμα 2:** Υλοποίηση (τυπικά με SPMD ή map reduce)

Παράδειγμα: Εξίσωση θερμότητας

- Αλγόριθμος του Jacobi (2-διάστατο πλέγμα Χ * Υ)

```
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])
```



Data centric

- Μοιράζουμε τα δεδομένα ανά σημείο / γραμμή / στήλη / μπλοκ

Task centric

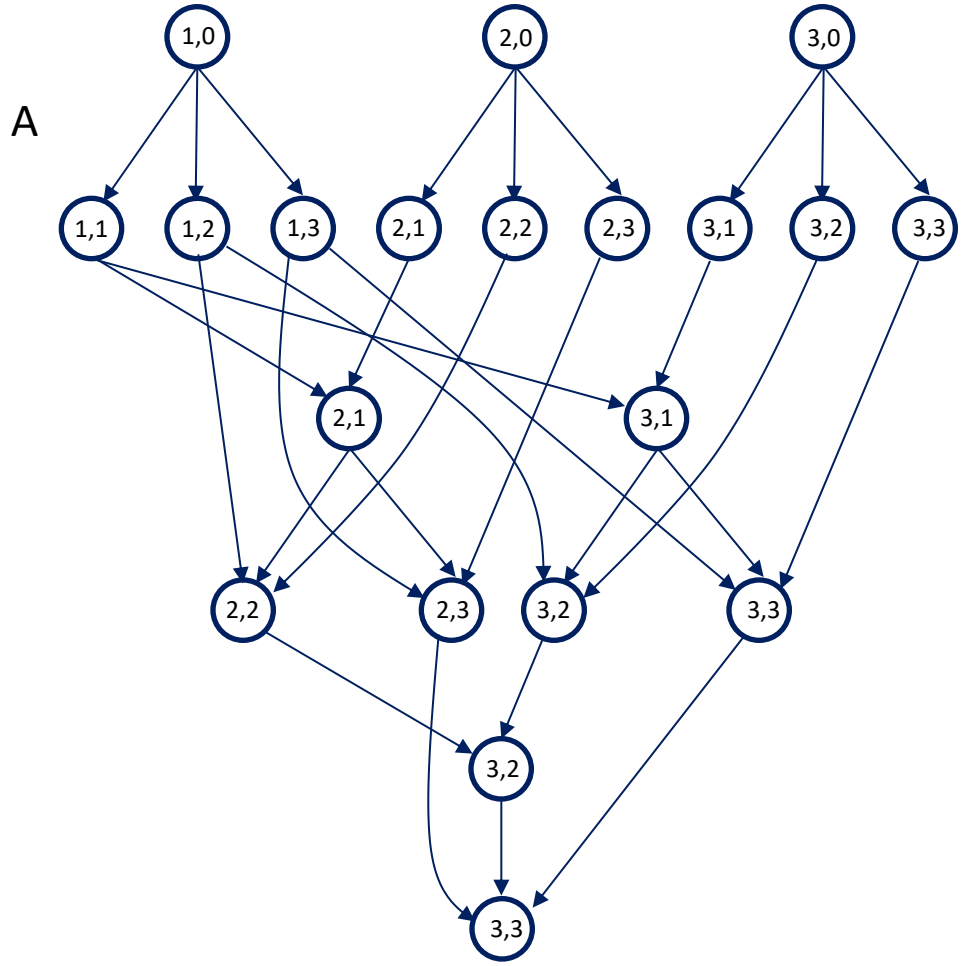
- **1 task =** υπολογισμός θερμότητας για κάθε σημείο του χωρίου
- **1 task =** υπολογισμός θερμότητας για κάθε χρονικό βήμα
- **1 task =** υπολογισμός θερμότητας για μία γραμμή / στήλη / block του δισδιάστατου πλέγματος για όλες τις χρονικές στιγμές
- **1 task =** υπολογισμός θερμότητας για μία γραμμή / στήλη / block του δισδιάστατου πλέγματος για μία χρονική στιγμή

Παράδειγμα: Task graph για LU decomposition με data parallel per row

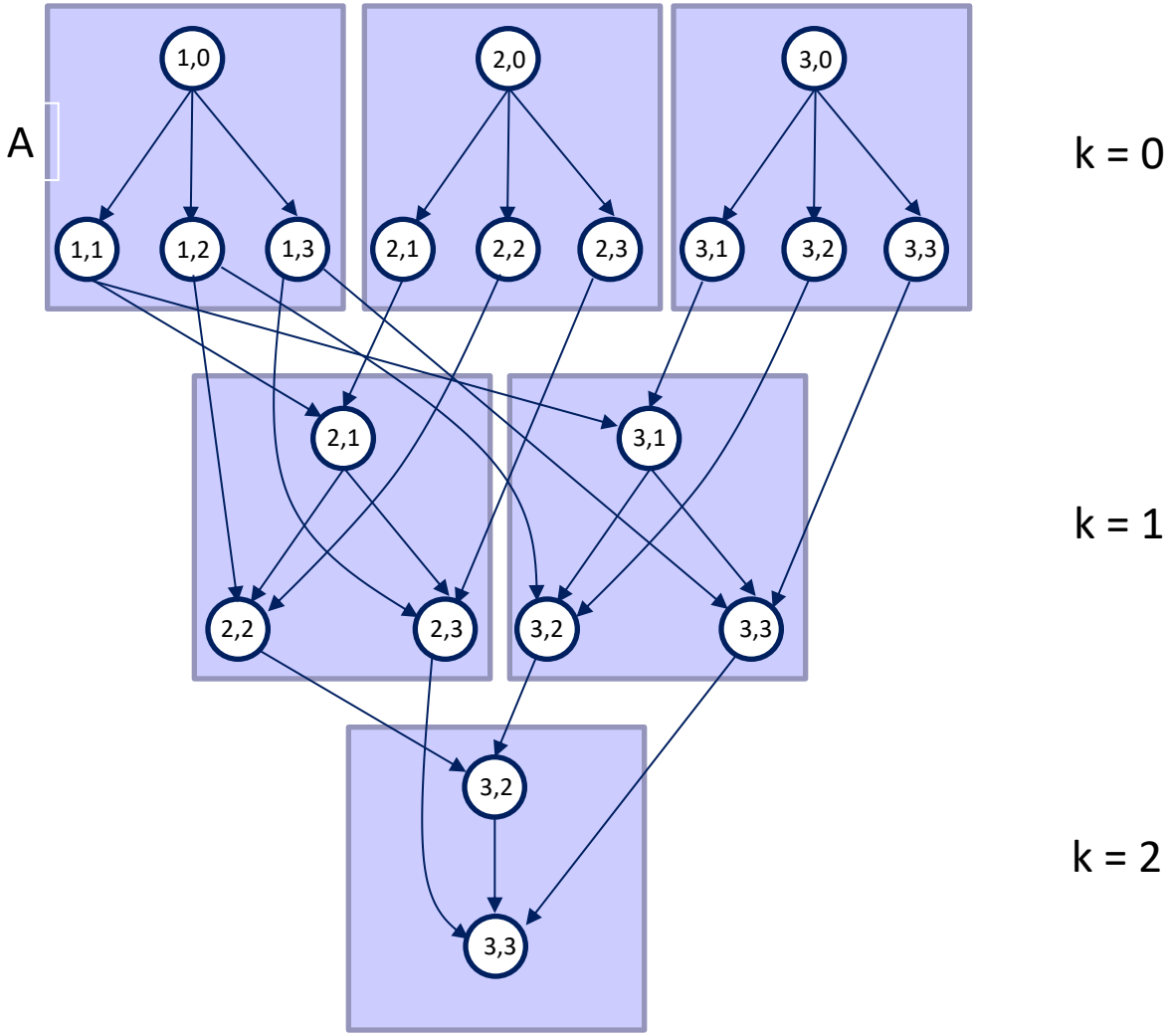
```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      A[i][k] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - A[i][k] * A[k][j];
   }
```

- **Data centric προσέγγιση:** μοιράζω **κατά γραμμές** τον πίνακα A (σημείωση: για το παράδειγμα χρησιμοποιούμε μία «in-place» έκδοση του πυρήνα LU)

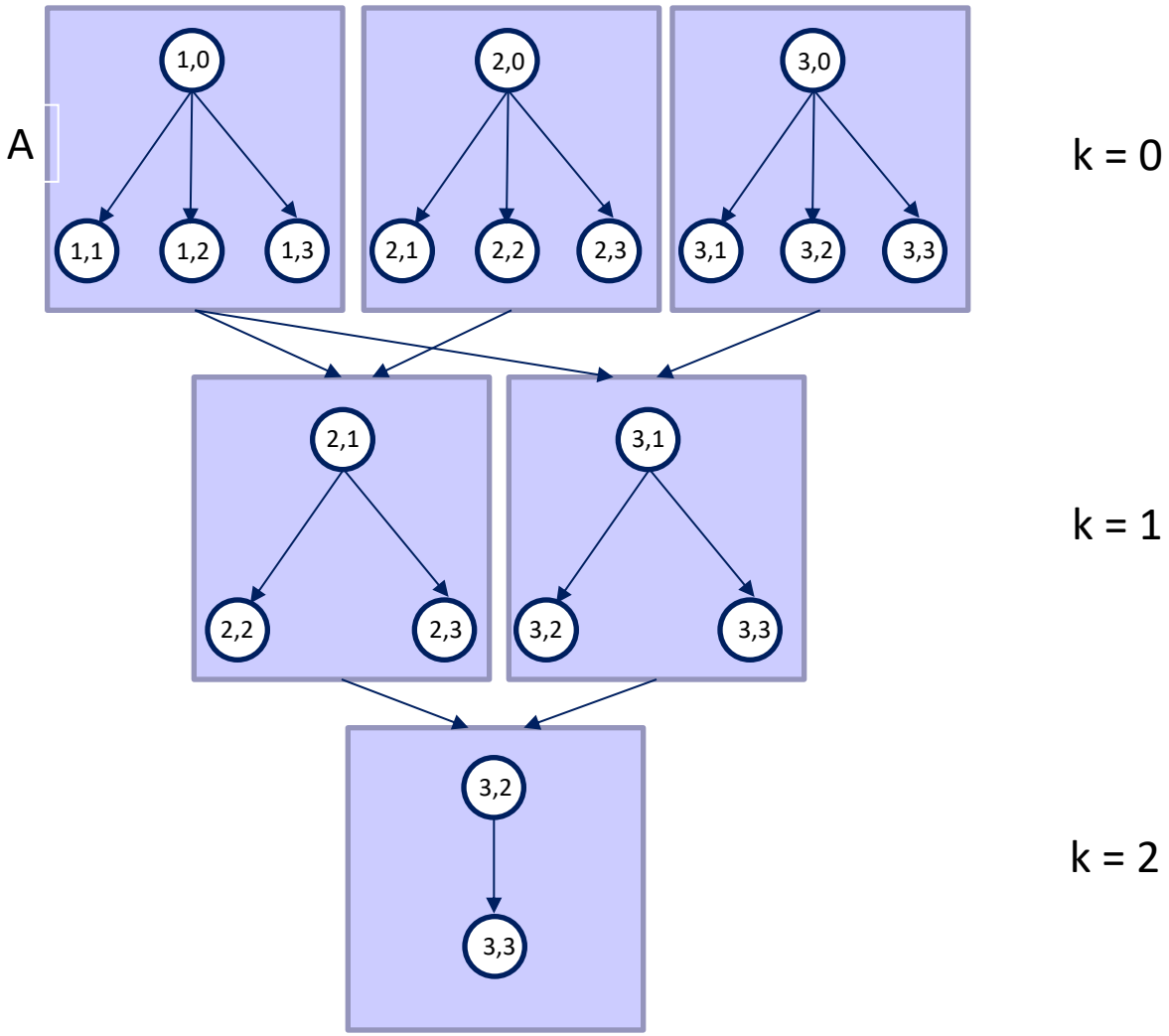
Παράδειγμα: Task graph για LU decomposition με data parallel per row



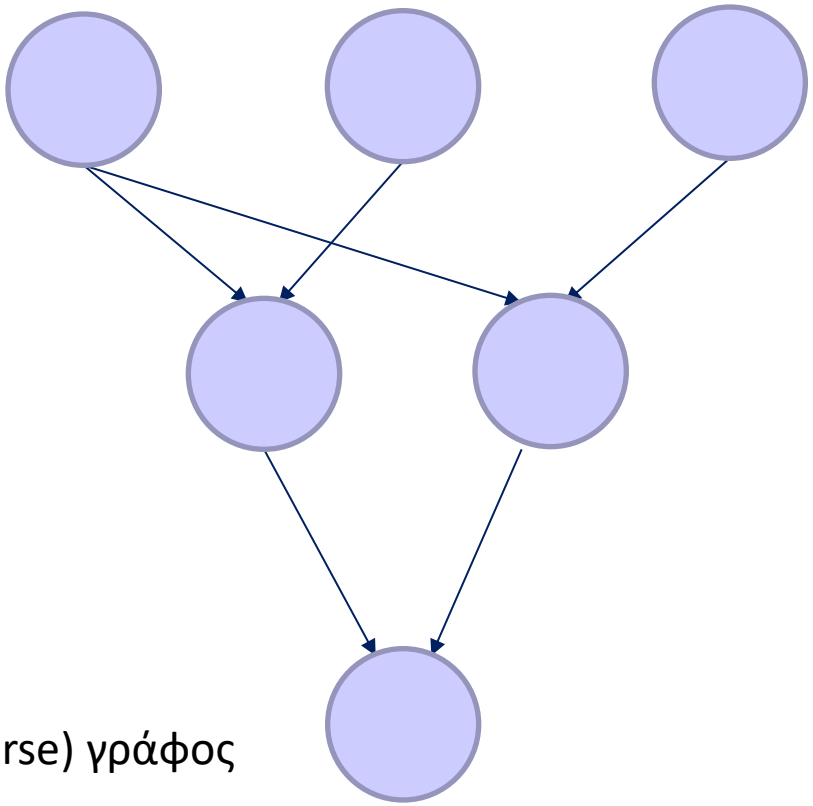
Παράδειγμα: Task graph για LU decomposition με data parallel per row



Παράδειγμα: Task graph για LU decomposition με data parallel per row



Παράδειγμα: Task graph για LU decomposition με data parallel per row



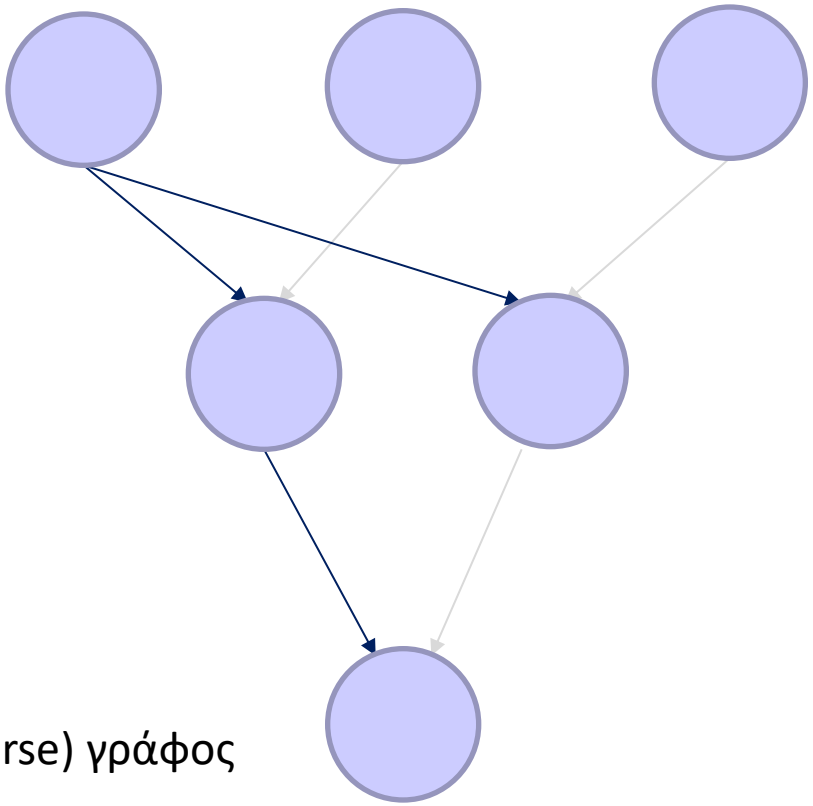
k = 0

k = 1

k = 2

Εκτελείται πιο αδρός (coarse) γράφος

Παράδειγμα: Task graph για LU decomposition με data parallel per row



k = 0

k = 1

k = 2

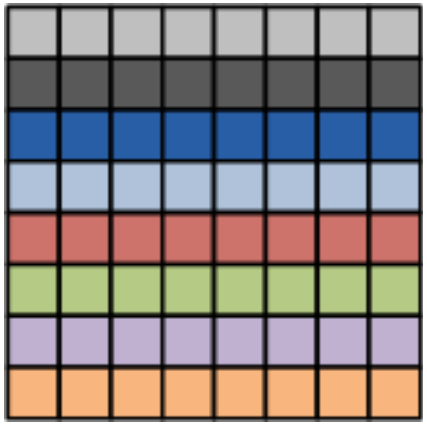
Εκτελείται πιο αδρός (coarse) γράφος

Δεν απαιτείται ανταλλαγή δεδομένων

Διαμοιρασμός κανονικών δομών

1-dimensional

(ανά γραμμή/στήλη)



+ Απλή στην υλοποίηση
- Περιοριστική
(το task που προκύπτει εξαρτάται από το N του πίνακα)

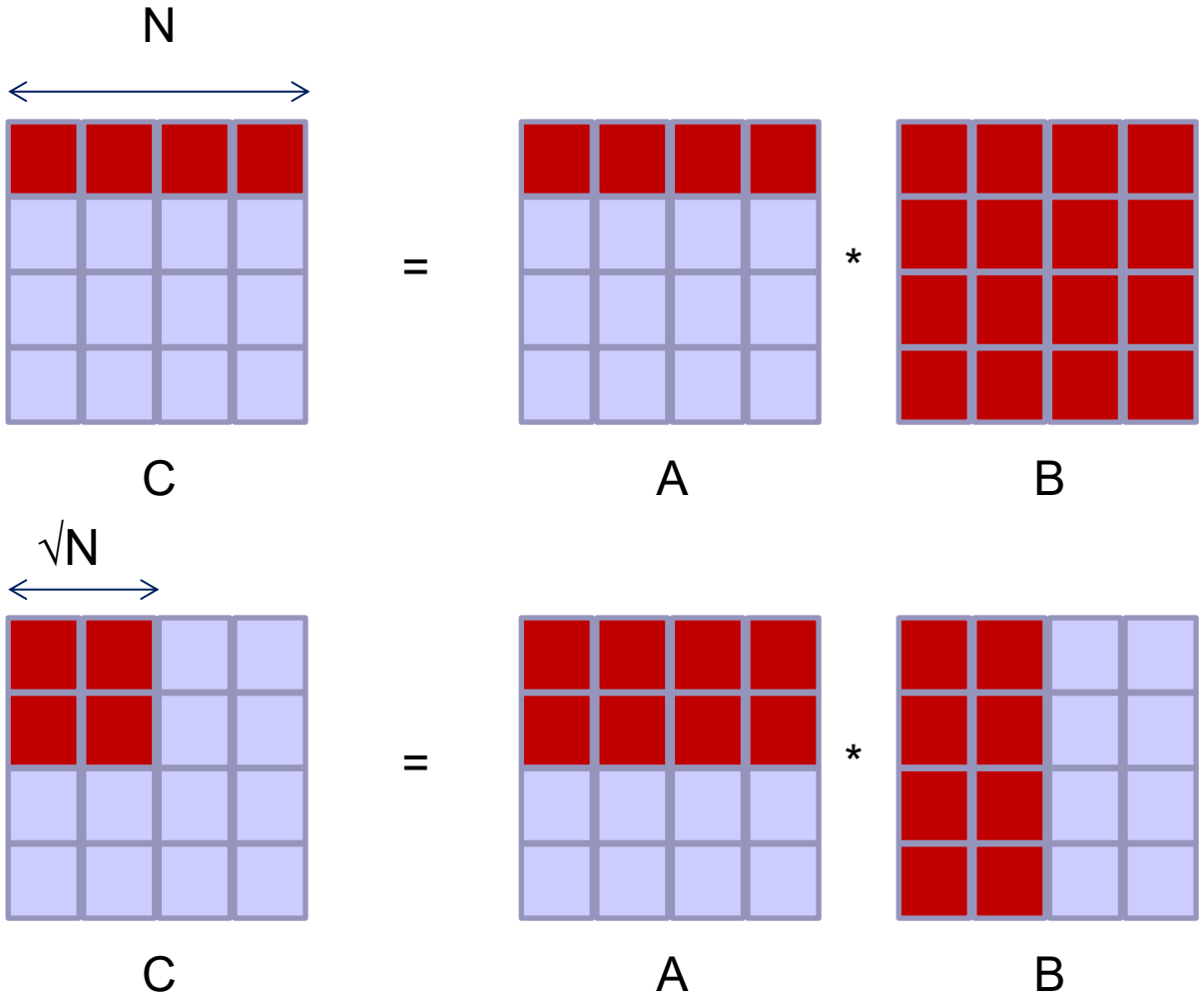
2-dimensional

(ανά μπλοκ)

- Δυσκολότερη στην υλοποίηση
+ Πιο ευέλικτη
(ο προγραμματιστής ελέγχει το μέγεθος του task)

Μπορεί να οδηγούν σε διαφορετικές ανάγκες για επικοινωνία (παράδειγμα stencil vs matrix multiplication)

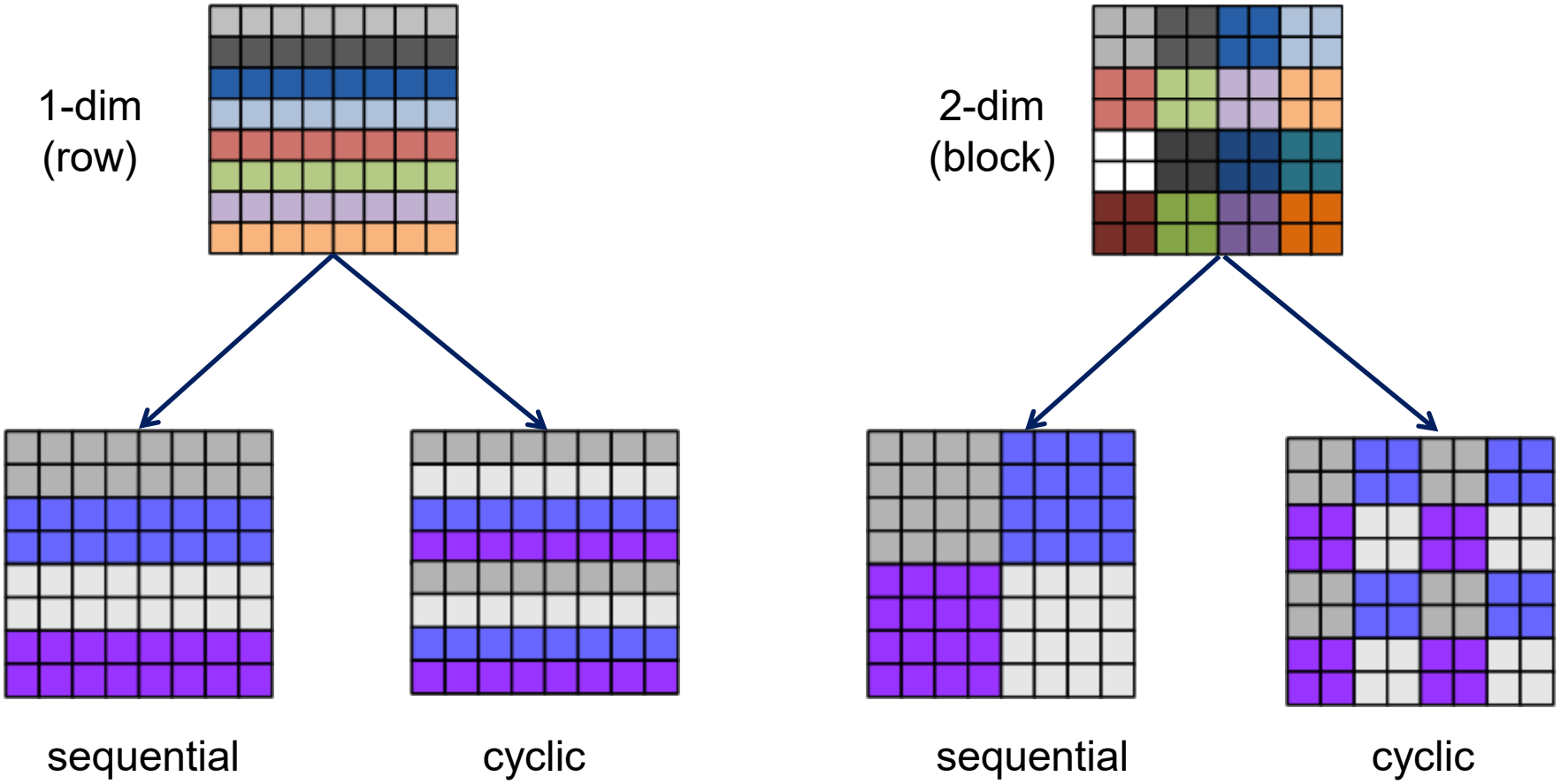
Κατανομή κατά γραμμή/στήλη ή κατά μπλοκ?



$N + N^2$

$2\sqrt{N} * N$

Διαμοιρασμός κανονικών δομών



Παράδειγμα: Επιλογή απεικόνισης για LU και stencil

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

```
//stencil
for (t = 1; t < T; t++){
   for (i = 1; i < X; i++)
      for (j = 1; j < Y; j++)
         A[t][i][j] = 0.2*(A[t-1][i][j] + A[t-1][i-1][j] +
                           A[t-1][i+1][j] + A[t-1][i][j-1] +
                           A[t-1][i][j]);
}
```

Υλοποίηση με 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
- Είναι γενική τακτική, μπορεί να υλοποιηθεί και σε προγραμματιστικά μοντέλα κοινού χώρου διευθύνσεων και σε ανταλλαγή μηνυμάτων
- Ταιριάζει ιδιαίτερα σε data parallel υλοποίηση για προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων
- Έχει υιοθετηθεί από το MPI

Υλοποίηση με SPMD

- 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
  - Ο προγραμματιστής αναλαμβάνει να «οδηγήσει» σωστά κάθε διεργασία στα σωστά δεδομένα και στο σωστό τμήμα κώδικα (διαδικασία που μπορεί να οδηγήσει σε σφάλματα)
- Μπορεί να οδηγήσει σε υψηλή επίδοση καθώς ο προγραμματιστής έχει τον (σχεδόν) πλήρη έλεγχο της υλοποίησης

Βήματα υλοποίησης στο σχήμα SPMD

- Αρχικοποίηση - Δημιουργία οντοτήτων εκτέλεσης
- 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 (π.χ. client – server, producer – consumer)
  - Ανάληψη διαφορετικών επαναλήψεων σε βρόχο
- Τερματισμός

Παράδειγμα: εξίσωση θερμότητας με SPMD 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])
```

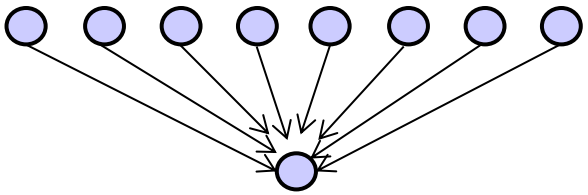
```
myid = get_my_id_from_system();
chunk = X / NUM_PRCS; // υποθέτει X % NUM_PRCS == 0
mystart = myid*chunk + 1;
myend = myend = mystart + chunk;
for (steps=0; steps < T; steps++){
  for (i=mystart; i<myend; i++)
    for (j=1; j<Y-1, j++)
       A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j]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

  synchronize;
}
```

Homework: εξίσωση θερμότητας με SPMD σε 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])
```

map-reduce

- Τακτική (και πλατφόρμες) υλοποίησης που ταιριάζει στο pattern του data parallelism
- Αφορά προβλήματα που έχουν παραλληλισμό στα δεδομένα εισόδου
- Αναπτύχθηκε από την ανάγκη επεξεργασίας τεράστιου όγκου δεδομένων (High Performance Data Analytics)
- Η επεξεργασία υποσυνόλου των δεδομένων ανατίθεται σε mappers και ο συνδυασμός των αποτελεσμάτων σε reducers
- Γνωστά frameworks: Hadoop (https://hadoop.apache.org/) , Spark (https://spark.apache.org/), κλπ



Loop parallelism

- Η παραλληλοποίηση των for-loops αποτελεί σημαντικότατη προσέγγιση στο σχεδιασμό και την υλοποίηση ενός παράλληλου προγράμματος
- «Ξέχνα το σχεδιασμό – εστίασε στα for loops»
- 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 χρόνου εκτέλεσης
  - 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

Loop parallelism – Ζητήματα υλοποίησης

- Υλοποιείται κυρίως στο προγραμματιστικό μοντέλο κοινού χώρου διευθύνσεων
- Ο προγραμματιστής τυπικά επισημαίνει:
  - Τα loop που είναι παράλληλα (δηλαδή τα loop των οποίων οι επαναλήψεις είναι ανεξάρτητες και μπορούν να ανατεθούν σε διαφορετικά νήματα)
  - Τον τρόπο πρόσβασης στα δεδομένα (στο OpenMP: shared, private = copied, reduction)
  - Το σχήμα δρομολόγησης των επαναλήψεων (στατικά ή δυναμικά)

automatic parallelization

- Η εύρεση των παράλληλων for αποτελεί τον κυριότερο στόχο της αυτόματης παραλληλοποίησης
- Σε 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 (δεν παράγεται κάτι σε μία επανάληψη που καταναλώνεται σε μία άλλη)
- 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 (καθώς 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];
   }
```

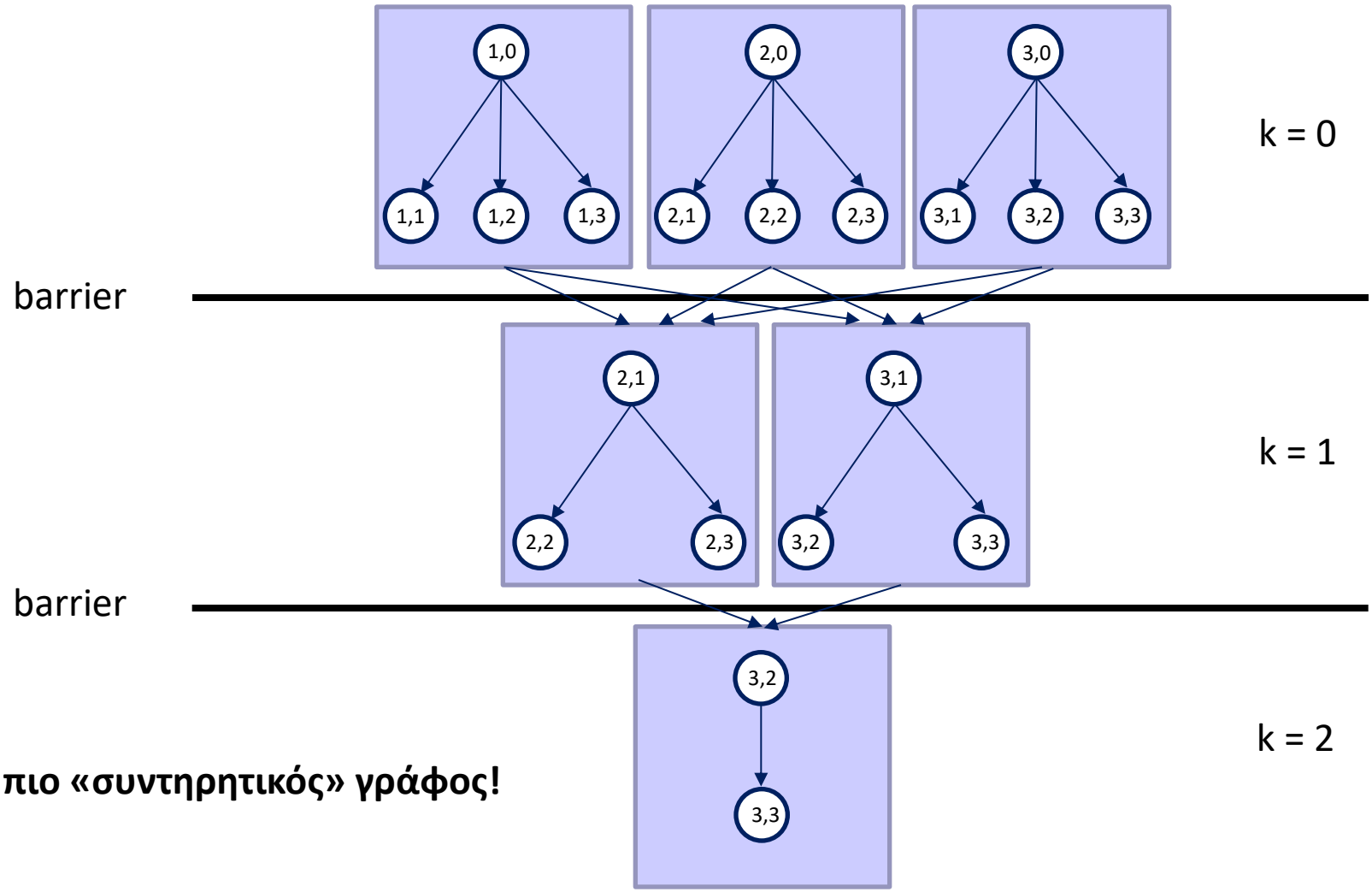
$$D = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & < & < \\ 0 & < & < & 0 \end{bmatrix}$$

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];
   }
```

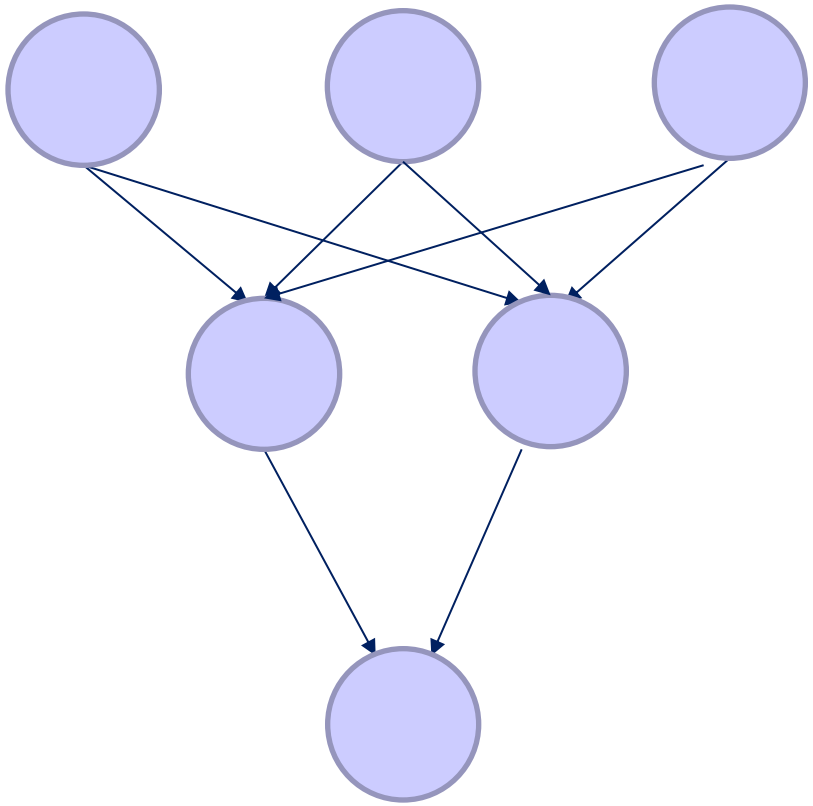
Τυπικά το parallel for επιβάλει καθολικό συγχρονισμό στο τέλος του (barrier)

Παράδειγμα: Παραλληλοποίηση LU με parallel for



πιο «συντηρητικός» γράφος!

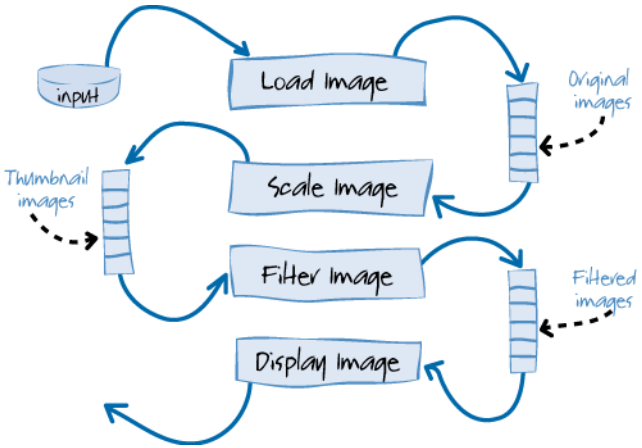
Παράδειγμα: Παραλληλοποίηση LU με parallel for



πιο «συντηρητικός» γράφος!

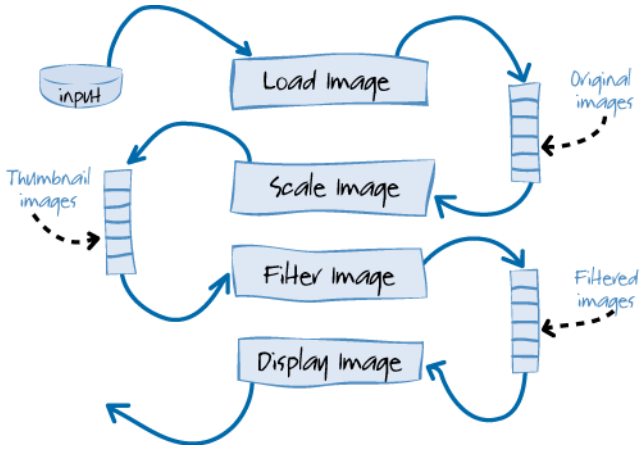
Function parallelism

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση (specialized accelerators)
  - Η παραλληλοποίηση σε κάποιες φάσεις με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης ή λόγω μεγάλου σειριακού μέρους – βλ. νόμο Amdahl)

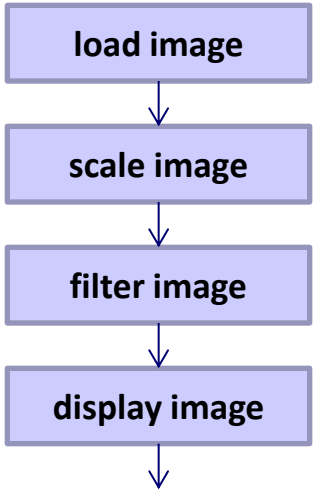


Function parallelism

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση (specialized accelerators)
  - Η παραλληλοποίηση σε κάποιες φάσεις με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης ή λόγω μεγάλου σειριακού μέρους – βλ. νόμο Amdahl)

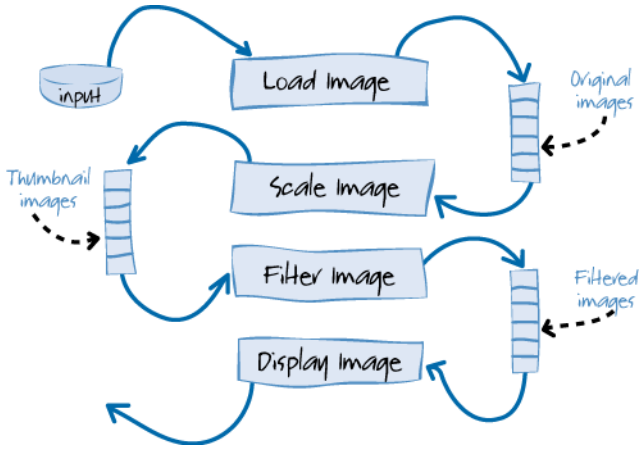


data centric:
π.χ. 32 threads ανά φάση

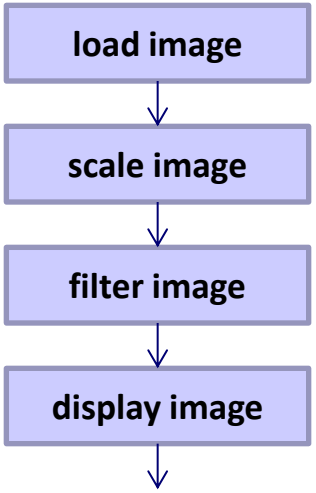


Rule of thumb

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση (specialized accelerators)
  - Η παραλληλοποίηση σε κάποιες φάσεις με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης ή λόγω μεγάλου σειριακού μέρους – βλ. νόμο Amdahl)



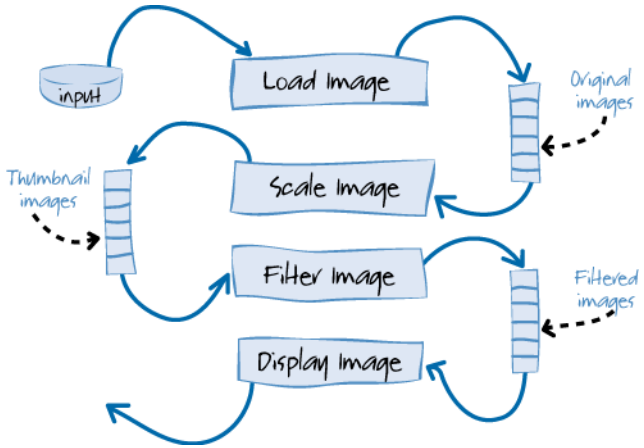
data centric:
π.χ. 32 threads ανά φάση



Αν κάθε φάση κλιμακώνει μέχρι τα 8 threads, έχουμε προφανή σπατάλη πόρων

Rule of thumb

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση (specialized accelerators)
  - Η παραλληλοποίηση σε κάποιες φάσεις με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης ή λόγω μεγάλου σειριακού μέρους – βλ. νόμο Amdahl)



load image (#4)	scale image (#3)	filter image (#2)	display image (#1)
8 threads	8 threads	8 threads	8 threads

load image (#5)	scale image (#4)	filter image (#3)	display image (#2)
8 threads	8 threads	8 threads	8 threads

Σύνοψη

- Παραλληλισμός σε επίπεδο εργασίας (task parallelism)
- Παραλληλοποίηση σε επίπεδο δεδομένων (data parallelism)
- Παραλληλοποίηση σε επίπεδο βρόχου (loop parallelism)
- Παραλληλοποίηση σε επίπεδο συνάρτησης (function parallelism)

- Μερικές επισημάνσεις
  - Ο παραλληλισμός σε επίπεδο εργασίας είναι ένα πολύτιμο μοντέλο για τη σχεδίαση των προγραμμάτων, δεν δίνει όμως εύκολα αποδοτικές υλοποιήσεις
    - όχι καλή υποστήριξη σε προγραμματιστικά μοντέλα ανταλλαγής μηνυμάτων
    - από πλευράς επίδοσης υστερεί σε σχέση με data/loop parallelism λόγω απώλειας της τοπικότητας των δεδομένων
    - είναι όμως μια εξαιρετική λύση για υλοποίηση σε προγραμματιστικό μοντέλο κοινού χώρου διευθύνσεων όταν δεν υπάρχουν for-loops (αναδρομή, while loops, ακανόνιστες/απρόβλεπτες/ανισοκατανεμημένες εφαρμογές)
  - Για μεγάλης κλίμακας προγράμματα τυπική προσέγγιση είναι ο παραλληλισμός σε επίπεδο δεδομένων με υλοποίηση SPMD

Σύνοψη

- Μερικές επισημάνσεις – cnt.
  - Ο παραλληλισμός σε επίπεδο βρόχου σε συνδυασμό με την πολύ καλή υποστήριξη για δυναμική απεικόνιση των επαναλήψεων αποτελεί μια πολύ καλή επιλογή υλοποίησης για προγ. μοντέλα κοινού χώρου διευθύνσεων, όμως:
    - μπορεί να επιβάλει περισσότερο συγχρονισμό από αυτόν που απαιτείται από το πρόβλημα (πιο συντηρητικός γράφος εξαρτήσεων)
    - δεν έχει καλή υποστήριξη σε προγ. μοντέλα ανταλλαγής μηνυμάτων
  - Ο παραλληλισμός σε επίπεδο συνάρτησης είναι κατάλληλος για προβλήματα με ροή δεδομένων και διαφορετικές φάσεις (συναρτήσεις) που επενεργούν στα δεδομένα. Χρειάζεται μελέτη για το κατά πόσον υπερτερεί έναντι πιο απλών τρόπων παραλληλοποίησης (π.χ. loop ή data parallelism ανά συνάρτηση)

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

OpenMP

- Προγραμματιστικό μοντέλο: **κοινός χώρος διευθύνσεων**
- Τακτική υλοποίησης: **parallel for, fork-join, SPMD**
- 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)

C++ TBB

- Προγραμματιστικό μοντέλο: **κοινός χώρος διευθύνσεων**
- Τακτική υλοποίησης: **parallel for, fork-join, task graph, SPMD**
- Ολοκληρωμένη λύση από την STL της C++ με ισχυρό ΣΧΕ
- Επιπλέον: παράλληλοι αλγόριθμοι και ταυτόχρονες δομές δεδομένων

Cilk

- Προγραμματιστικό μοντέλο: **κοινός χώρος διευθύνσεων**
- Τακτική υλοποίησης: **parallel for, fork-join, SPMD**
- 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
- Βασικές λέξεις κλειδιά: cilk, spawn, sync

C

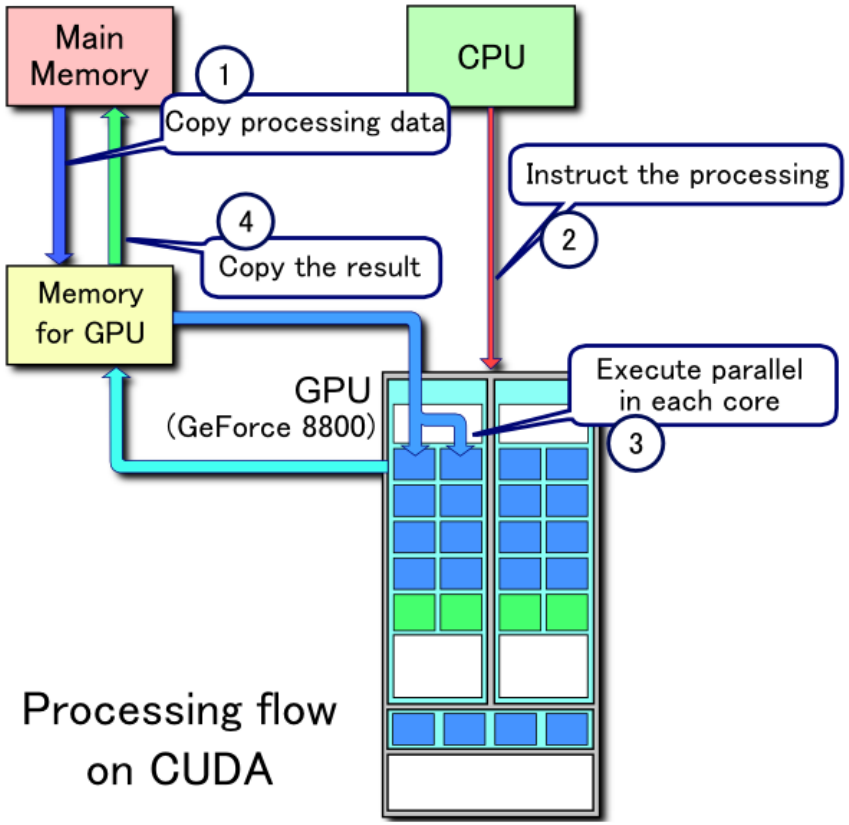
```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

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

- Προγραμματιστικό μοντέλο: **κοινός χώρος διευθύνσεων**
- Τακτική υλοποίησης: **SPMD (**SIMT = **S**ingle **I**nstruction **M**ultiple **T**hreads)
- 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

MPI

- Προγραμματιστικό μοντέλο: **ανταλλαγή μηνυμάτων**
- Τακτική υλοποίησης: **SPMD**
- Αποτελεί τον de facto τρόπο προγραμματισμού σε υπερυπολογιστικά συστήματα
- 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

Γλώσσες PGAS (Partitioned Global Address Space)

- Προγραμματιστικό μοντέλο: **κοινός χώρος διευθύνσεων**
- Τακτική υλοποίησης: **SPMD, parallel for**
- Υποθέτει καθολικό χώρο διευθύνσεων (global address space) που κατανέμεται (partitioned) στις τοπικές μνήμες των επεξεργαστών ενός συστήματος κατανεμημένης μνήμης
- 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
- 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

Charm++

- Προγραμματιστικό μοντέλο: **ανταλλαγή μηνυμάτων**
- Τακτική υλοποίησης: **task graph**
- Το πιο εξελιγμένο framework για task-based παραλληλισμό σε προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων
- Ο προγραμματιστής περιγράφει το task graph, το χαρακτηρισμό των δεδομένων (π.χ. read only δεδομένα) και τη ροή των δεδομένων ανάμεσα στα tasks.
- Βασίζεται στη φιλοσοφία του parallel slack (δημιουργώ πολύ περισσότερα tasks από τους διαθέσιμους επεξεργαστές)
- Το ΣΧΕ αναλαμβάνει την εκτέλεση των tasks σε επεξεργαστές, την υλοποίηση της ροή των δεδομένων, την ισοκατανομή φορτίου (υποστηρίζεται task migration) και την ανοχή σε σφάλματα (fault tolerance)

Languages and tools

		SPMD	parallel for	fork / join	task graph
Shared address space	*Pthreads*	√			
	OpenMP	√	√	√	
	Cilk		√	√	
	TBBs	√	√	√	√
	CUDA	√			
	UPC	√	√		
Message passing	*MPI*	√			
	Charm++				√

Βελτιστοποίηση: μερικές παρατηρήσεις

- Σχεδιασμός:
  - Η πολυπλοκότητα (συνολικός αριθμός πράξεων - work) παίζει πρωταρχικό ρόλο (μην ξεχνάμε τα βασικά!)
  - Οι εξαρτήσεις ανάμεσα στα tasks (task graph) φανερώνουν τις προοπτικές του παραλληλισμού και την ανάγκη για επικοινωνία
  - Η πρόσβαση στη μνήμη, ο συγχρονισμός και η επικοινωνία κοστίζουν (πολύ!). Προσοχή στο operational intensity, τις μεταφορές δεδομένων και το συγχρονισμό.
- Υλοποίηση και εκτέλεση:
  - Είναι σημαντικό κατά την υλοποίηση να μη χάνουμε τον παραλληλισμό που αναδείξαμε κατά το σχεδιασμό (ή αν το κάνουμε να είναι αποτέλεσμα κάποιου μελετημένου tradeoff επίδοσης – ευκολίας στον προγραμματισμό)
  - 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!

Σχετικά ζητήματα απεικόνισης (δρομολόγησης)

- Δρομολόγηση εργασιών σε ένα υπερυπολογιστικό σύστημα
  - *N* κόμβοι, c πυρήνες ανά κόμβο
  - *K* jobs, κάθε job *i* ζητάει N_i κόμβους και c_i πύρήνες
  - Στατικές προσεγγίσεις
- Δρομολόγηση παράλληλων εφαρμογών σε επίπεδο λειτουργικού
  - Πολυπύρηνο σύστημα
  - *K* εφαρμογές με διαφορετικός αριθμό από threads η κάθε μία
  - Οι τρέχουσες προσεγγίσεις δεν είναι ικανοποιητικές
- Δρομολόγηση των tasks μίας παράλληλης εφαρμογής σε ένα πολυπύρηνο σύστημα
  - Επαναλήψεις ενός παράλληλου loop
  - Tasks μίας παράλληλης εφαρμογής

Ερωτήσεις;