Συστήματα Παράλληλης & Κατανεμημένης Επεξεργασίας

Ενότητα 5: Αρχές Παράλληλου Προγραμματισμού.
Τεχνικές παραλληλοποίησης υπολογισμών.

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 κατανόηση των βασικών αρχών και τεχνικών παραλληλοποίησης.
- Η κατανόηση της ανάθεσης εργασιών σε επεξεργαστές.

Διαδικασία Παραλληλοποίησης Προγράμματος (1/3)

- Η ανάπτυξη Αλγόριθμων γίνεται συνήθως με την επεξεργασία ενός υπάρχοντος σειριακού αλγόριθμου. Τα βήματα που ακολουθούνται είναι τα εξής:
  1. **Διάσπαση** (decomposition) του αλγόριθμου σε εργασίες (tasks) οι οποίες μπορούν να εκτελεστούν σειριακά ή πιθανόν και παράλληλα.
  2. **Ανάθεση** των εργασιών σε διεργασίες (processes).
  3. **Χρονοδρομολόγηση** των διεργασιών σε επεξεργαστές:
     1. Αντιστοίχηση κάθε διεργασίας με έναν επεξεργαστή.
     2. Απόφαση για τον χρόνο που θα εκτελεστούν.

Διαδικασία Παραλληλοποίησης Προγράμματος (2/3)

Ορολογία:

- **Εργασία** (task): το μικρότερο κομμάτι κώδικα που δε διασπάται σε μικρότερα τμήματα (αυθαίρετο).
- **Χοντροκομμένη εργασία** (coarse grain task): Όταν ο κώδικας που εμπεριέχεται στην εργασία είναι σχετικά μεγάλος.
- **Ψιλοκομμένη εργασία** (fine grain task): Όταν ο κώδικας που εμπεριέχεται στην εργασία είναι σχετικά μικρός.
- **Διεργασία** (process): σύνολο από εργασίες (ομάδα εργασιών) που εκτελούνται στον ίδιο επεξεργαστή.

Διαδικασία Παραλληλοποίησης Προγράμματος (3/3)

Βήμα 1ο: Διάσπαση του αλγόριθμου σε εργασίες.

- Ο αλγόριθμος χωρίζεται σε εργασίες που θεωρούνται **μη περαιτέρω διασπάσιμες**.
- Κυριότερο Πρόβλημα: Να γίνει ανάθεση τμημάτων κώδικα σε ξεχωριστές εργασίες, ώστε να **αποκαλυφθεί όσο το δυνατόν περισσότερος παραλληλισμός** – όσο το δυνατόν πιο ανεξάρτητες μεταξύ τους.

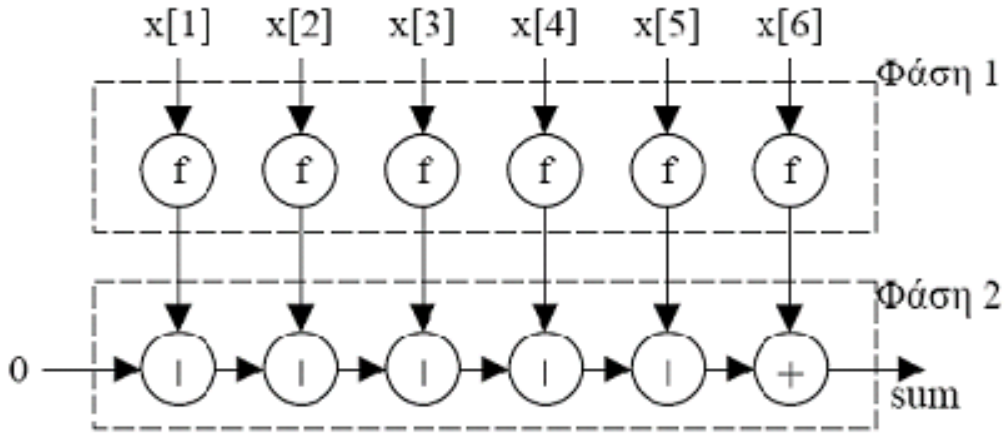
Βήμα 1ο: Διάσπαση του αλγόριθμου σε εργασίες: Παράδειγμα

- Μπορούμε να απεικονίσουμε την παρακάτω διαδικασία με έναν γράφο εξάρτησης (dependence graph), δηλαδή με ένα γράφημα που να δείχνει τις εξαρτήσεις μεταξύ των πράξεων.

```
for (i=1 to 6) {          /*Φάση 1*/
     y[i] - f(x[i]);
}
sum - 0;
for (i=1 to 6) {          /*Φάση 2*/
     sum - sum + y[i];
}
```

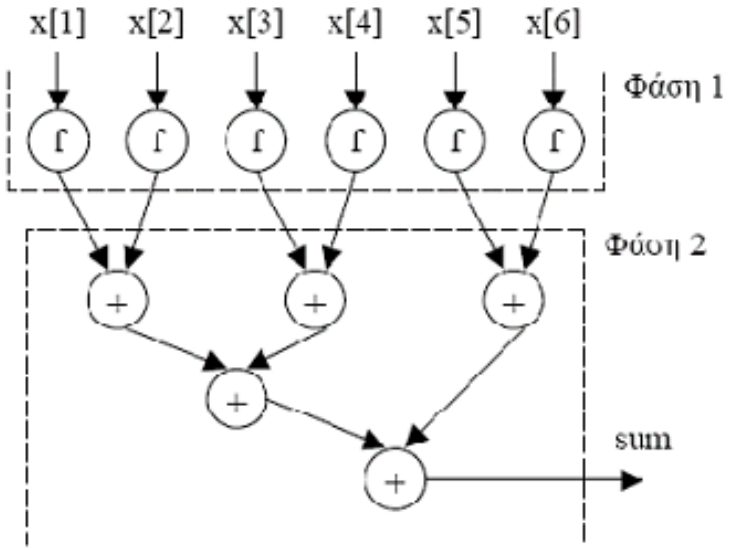
Βήμα 1ο: Διάσπαση του αλγόριθμου σε εργασίες: Γράφος Εξάρτησης (1/2)

- Θεωρούμε ότι μια εντολή είναι και μια εργασία. Η πρώτη φάση εκτελείται τελείως παράλληλα, η δεύτερη μόνο σειριακά.
- Αν υποθέσουμε ότι μια πράξη f(x) απαιτεί χρόνο Τ όσο και μια πρόσθεση τότε η σειριακή εκτέλεση απαιτεί χρόνο Τσ = 12ΤΜέγιστο μονοπάτι γράφου: Τπ=7Τ
- Ο λόγος Τσ/Τπ φανερώνει τον παραλληλισμό, εδώ: 12/7 = 1.7



Βήμα 1ο: Διάσπαση του αλγόριθμου σε εργασίες: Γράφος Εξάρτησης (2/2)

- Εδώ στη δεύτερη φάση γίνεται μια ιεραρχική άθροιση των αποτελεσμάτων της 1ης φάσης ανά 2 και αυξάνεται ο παραλληλισμός. Καλύτερη απόδοση: Τσ/Τπ = 11/4 = 2.75.



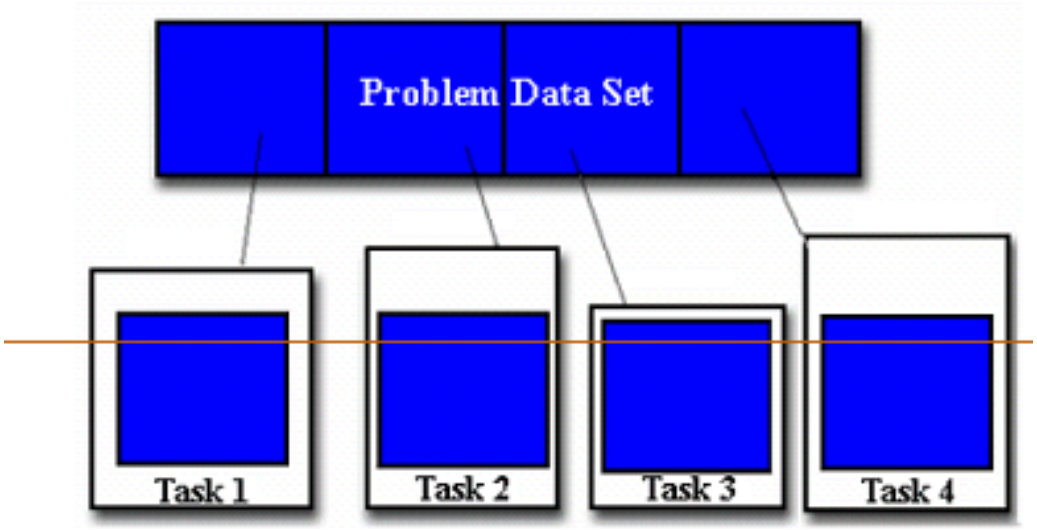
Βήμα 1ο: Διάσπαση του αλγόριθμου σε εργασίες

- Υπάρχουν 2 κύριοι τρόποι διάσπασης ενός αλγορίθμου σε εργασίες:

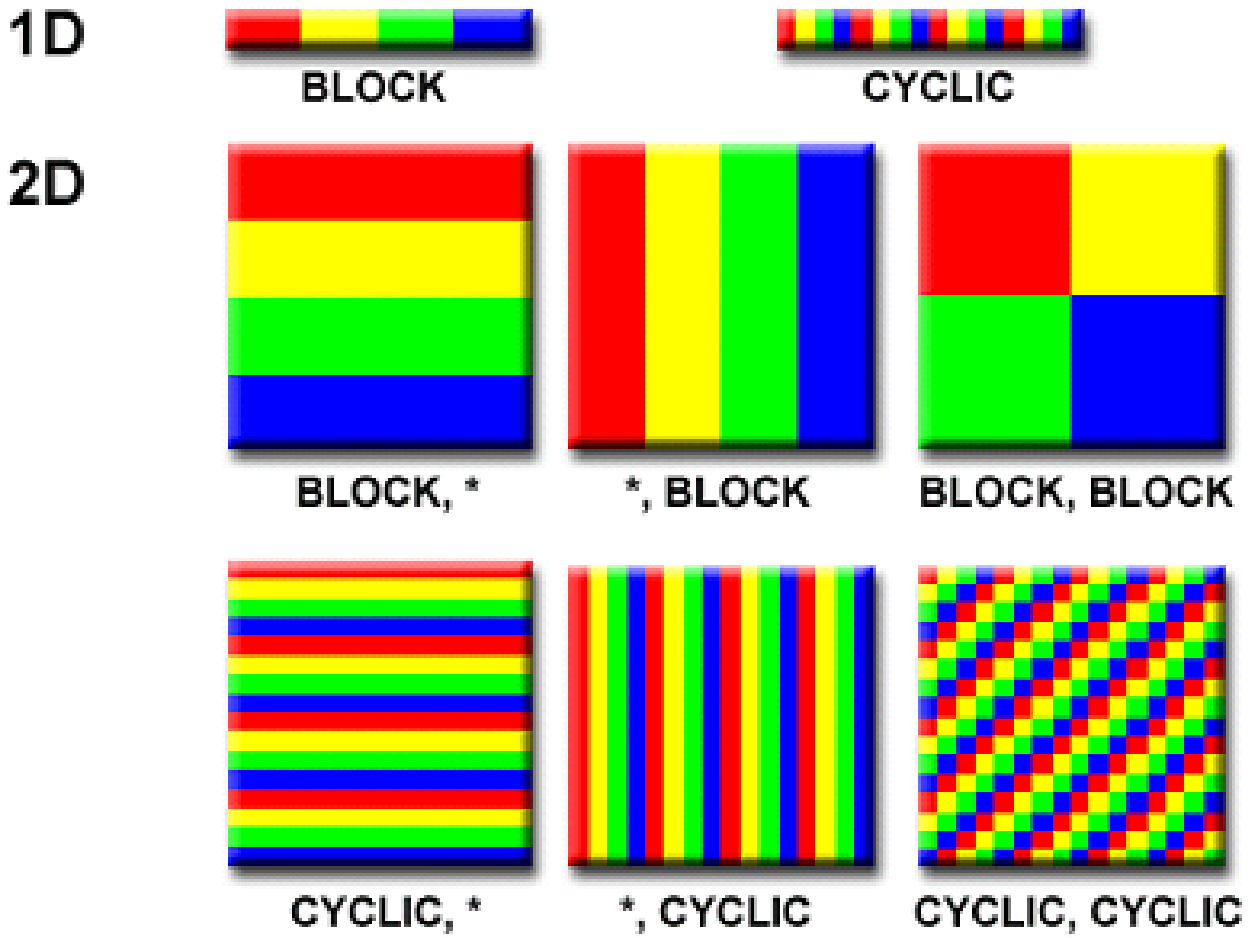
1. **Domain Decomposition** (Διάσπαση Πεδίου Ορισμού):
   - Στην διάσπαση αυτή τα δεδομένα χωρίζονται σε τμήματα και κάθε εργασία ασχολείται με ένα από αυτά τα τμήματα.
2. **Functional Decomposition** (Λειτουργική Διάσπαση):
   - Σε αυτόν τον τρόπο διάσπασης το πρόβλημα αποσυντίθεται με βάση τις εργασίες που πρέπει να γίνουν.

Διάσπαση Πεδίου Ορισμού (1/3)

- Μπορεί να εφαρμοσθεί εάν τα υποσύνολα των δεδομένων είναι πλήρως ανεξάρτητα μεταξύ τους, τότε το πρόβλημα χαρακτηρίζεται σαν “embarrassingly parallel problem”.



Πιθανές Διασπάσεις πεδίου ορισμού



Διάσπαση Πεδίου Ορισμού (2/3)

- Προσοχή:
  - Πρέπει να αποκαλύπτεται όσο το δυνατόν περισσότερος παραλληλισμός.
  - Πρέπει να ελαχιστοποιείται το κρίσιμο τμήμα.
- Παράδειγμα: Έστω θέλουμε να βρούμε το συνολικό άθροισμα των στοιχείων ενός διανύσματος.
- Ακολουθιακός κώδικας:

```
for(k=0;k<N;κ++)
{
sum=sum+a[k];
}
```

Διάσπαση Πεδίου Ορισμού (3/3)

- Μη αποδοτικός παράλληλος κώδικας για 2 νήματα με id=0 και id=1:

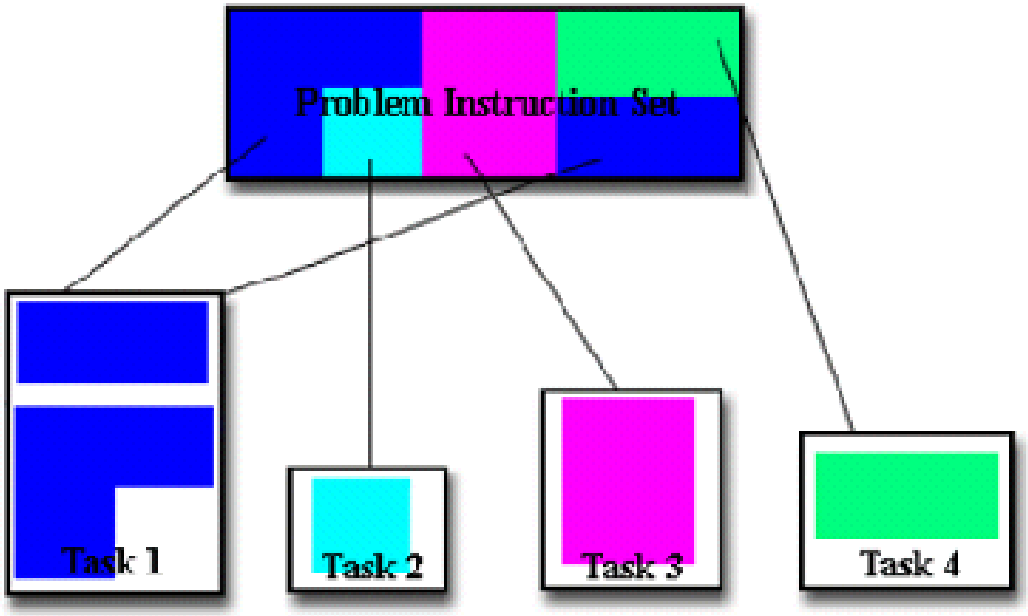
```
for(k=id*N/2 + 1;k<(id+1)*N/2;i++)
{mutex_lock(key);sum=sum+a[k];mutex_unlock(key)}
```

- Αποδοτικός παράλληλος κώδικας για 2 νήματα με id=0 και id=1:

```
for(k=id*N/2 + 1;k<(id+1)*N/2;i++)
{local_sum=local_sum+a[k];}
mutex_lock(key);
sum=sum+local_sum;
mutex_unlock(key);
```

Λειτουργική Διάσπαση

- Μπορεί να εφαρμοσθεί σε περιπτώσεις όπου όλα ή περισσότερα από τα δεδομένα πρέπει να είναι διαθέσιμα για κάθε μια από τις διεργασίες.



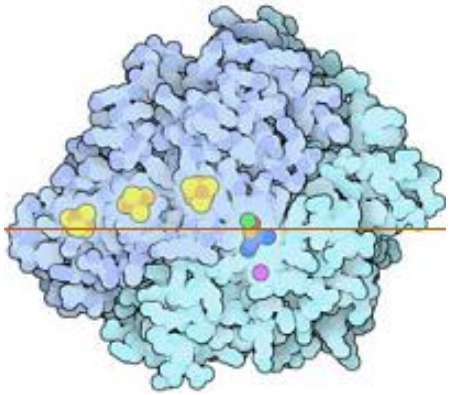
Εύρεση Δυνατότητας Παραλληλισμού (1/2)

- Πριν επιλέξουμε κάποιον από τους δύο τρόπους διάσπασης ενός αλγορίθμου, θα πρέπει να ξεκινήσουμε με το εξής ερώτημα:

  "**Είναι το προς εξέταση πρόγραμμα Παραλληλοποιήσιμο;**"

Εύρεση Δυνατότητας Παραλληλισμού Παράδειγμα 1ο

- Μόριο του μήνα (Μάρτιος 2009): Υδρογενάση, έχει την ιδιότητα να διασπά το νερό σε μοριακό Υδρογόνο.
- **Παράδειγμα 1. Υπολογίστε την δυναμική ενέργεια μερικών χιλιάδων διαμορφώσεων του πρωτεϊνικού ορίου και εντοπίστε αυτό με την χαμηλότερη ενέργεια (global minimum)**
- Είναι παραλληλοποιήσιμο γιατί κάθε διαμόρφωση του ορίου (γεωμετρία, ενέργεια) είναι ανεξάρτητη οντότητα από τις υπόλοιπες διαμορφώσεις.



Εύρεση Δυνατότητας Παραλληλισμού Παράδειγμα 2ο (1/3)

- Βρείτε τον πληθυσμό των κουνελιών που θα παραχθούν στο τέλος ενός έτους εάν κάνουμε τις παρακάτω παραδοχές:

A. Ξεκινήστε με ένα ζεύγος κουνελιών (1 αρσενικό και 1 θηλυκό) που γεννήθηκαν την 01/01.

B. Υποθέτουμε ότι όλοι οι μήνες έχουν ίδιες ημέρες.

C. Τα κουνέλια ενηλικιώνονται και αρχίζουν να ζευγαρώνουν 1 μήνα μετά την γέννησή τους και μπορούν να γεννούν κάθε 1 μήνα μετά την ενηλικίωση.

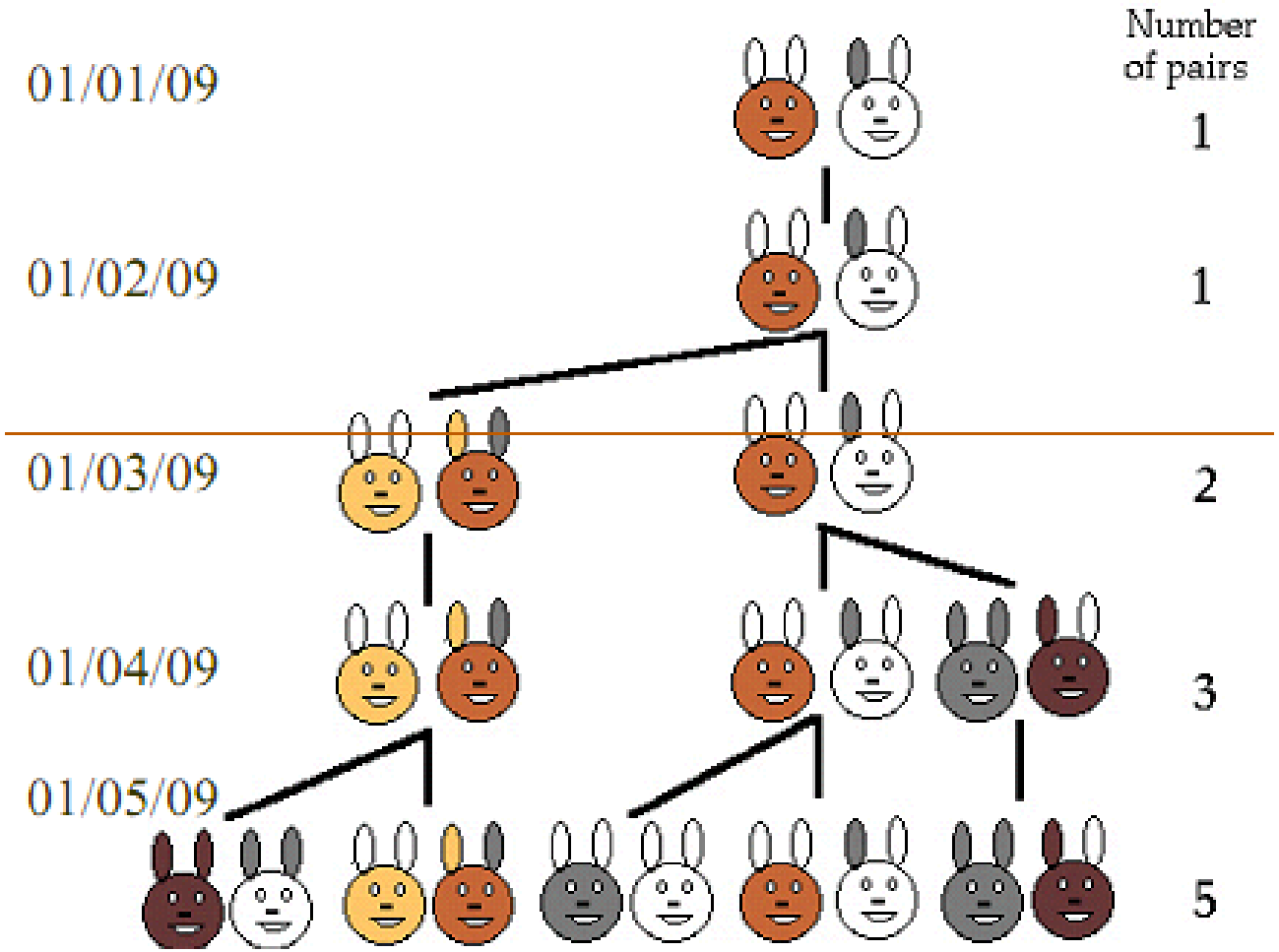
D. Κάθε φορά που έχουμε γεννητούρια παράγονται 1 αρσενικό και 1 θηλυκό.

E. Κανένα κουνέλι δεν πεθαίνει ή τρώγεται.

F. Δεν υπάρχουν ηθικοί κανόνες περί αιμομιξίας.

- **Πόσα κουνέλια θα υπάρχουν** συνολικά μέχρι το τέλος του χρόνου (31/12);

Εύρεση Δυνατότητας Παραλληλισμού Παράδειγμα 2ο (2/3)



Εύρεση Δυνατότητας Παραλληλισμού Παράδειγμα 2ο (3/3)

- Λύση στο παραπάνω πρόβλημα έδωσε ο Leonardo da Pisa "son of Bonaccio", το 1202 και είναι η γνωστή ακολουθία Fibonacci:

  F(k+2) = F(k+1) + F(k)

  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...
- Μπορεί το παραπάνω πρόβλημα να παραλληλοποιηθεί;
- Θα προτείνατε σε έναν που εκτρέφει κουνέλια να έκανε μια επένδυση σε παράλληλη μηχανή ώστε να υπολογίζει τον πληθυσμό των κουνελιών με “ακρίβεια”;
- Δεν είναι παραλληλοποιήσιμο πρόβλημα γιατί υπάρχει άμεση εξάρτηση του κάθε επόμενου όρου που πρόκειται να υπολογιστεί με τους 2 αμέσως προηγούμενους (καλύτερα ο εκτροφέας να τα επενδύσει σε περίφραξη!).

Εύρεση Δυνατότητας Παραλληλισμού (2/2)

- Αν το πρόβλημα είναι παραλληλοποιήσημο πρέπει να έχουμε σιγουρευτεί για τα παρακάτω:
  1. **Η σειριακή μορφή του προγράμματος (αν υπάρχει) παράγει αποτελέσματα.**
  2. **Ο προτεινόμενος αλγόριθμος δεν περιέχει σφάλματα (λογικά, αριθμητικά – bug free).**
- Ο νέος παράλληλος αλγόριθμος πρέπει να είναι ισοδύναμος με τον αρχικό, δηλαδή η παράλληλη μορφή του προγράμματος πρέπει εν τέλει να παράγει τα ίδια αποτελέσματα με την σειριακή του μορφή.

1ο Παράδειγμα

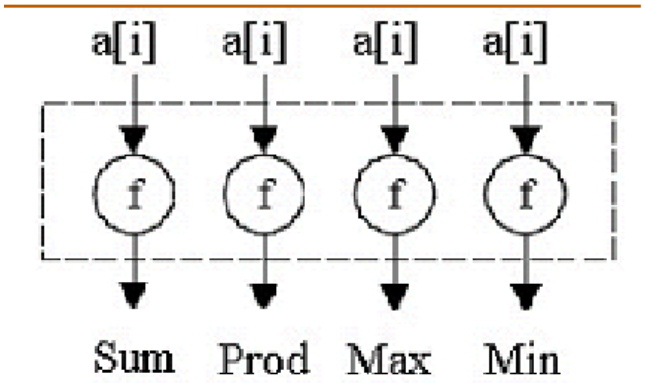
- Πρόβλημα: Να γραφεί ένα πρόγραμμα όπου από 4 στοιχεία ενός πίνακα να βρίσκει τα εξής:
  1. Άθροισμα,
  2. Γινόμενο,
  3. Μέγιστο,
  4. Ελάχιστο.

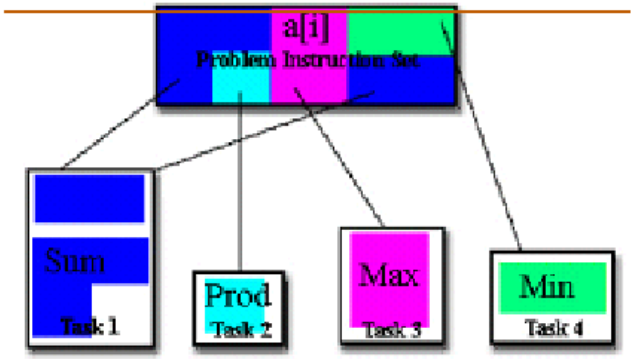
1ο Παράδειγμα Σειριακού Προγράμματος

```
#include <stdio.h>
#define SIZE 4 #define SIZE 4
int main (int argc, char* argv[])
{
int counter, sum, i;
float sum, prod, a[SIZE] =  {1.0, 2.0, 3.0, 4.0};
sum = 0.0;
prod = 1.0;
max = a[0];
min = a[0];
for (i = 0; i < SIZE; i++) {
          sum = sum + a[i];
          prod = prod * a[i];
          if (max < a[i])  max = a[i];  if (max < a[i])  max = a[i];
          if (min > a[i]) min = a[i];
          }
printf ("Sum=  %d,  Product=  %d,  Max= %f ,  Min=  %f \n", sum, prod, max, min);
return (0);
}
```

Παράδειγμα Παραλληλοποίησης Προγράμματος με την μέθοδο: Functional Decomposition (1/2)

- **Πρόβλημα:** Να γραφεί μια παράλληλη μορφή του προηγούμενου προγράμματος όπου κάθε διεργασία (εκτός της διεργασίας 0) θα αναλάβει να εκτελέσει κάθε μια από τις πράξεις 1) Άθροισμα, 2) Γινόμενο, 3)Μέγιστο, 4) Ελάχιστο.





- **Δεν υπάρχει κάποια εξάρτηση στα δεδομένα. Κάθε διεργασία μπορεί να ανατεθεί σε έναν επεξεργαστή.**

Παράδειγμα Παραλληλοποίησης Προγράμματος με την μέθοδο: Functional Decomposition (2/2)

- Το πρόγραμμα αυτό μπορεί να γίνει με OpenMPI και με 5 ranks. Το πρώτο rank στέλνει τον πίνακα στα υπόλοιπα, ενώ τα rank (1 έως 4) έχουν αναλάβει να κάνουν μια συγκεκριμένη λειτουργία από τις: άθροισμα, γινόμενο, ελάχιστο, μέγιστο.

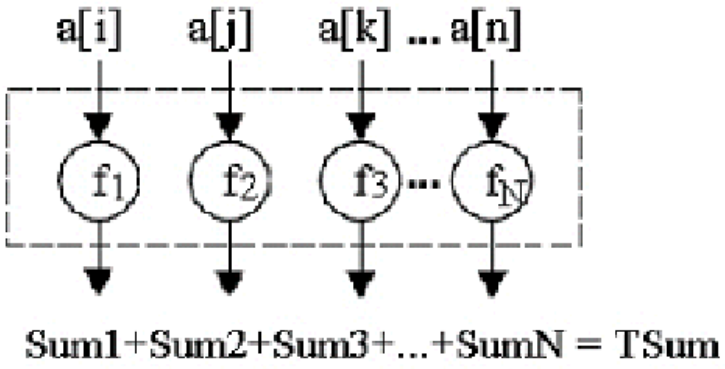
2ο Παράδειγμα Σειριακού Προγράμματος

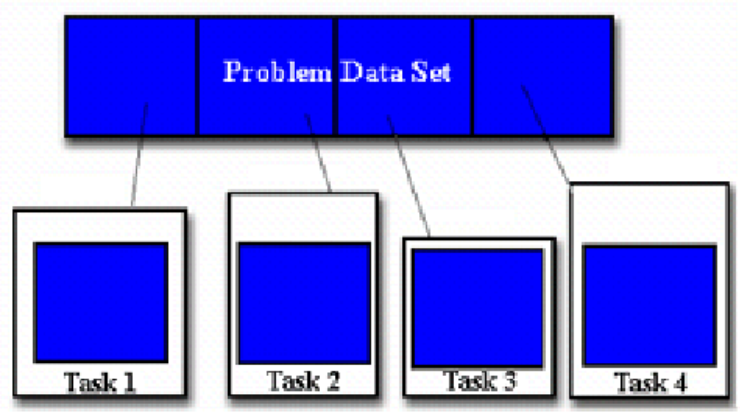
Πρόβλημα: Να γραφεί ένα πρόγραμμα που να υπολογίζει το άθροισμα των στοιχείων ενός πίνακα.

```
#include <stdio.h>
#define SIZE 50000
main (int argc, char *argv[])
int i, sum, Tsum,  data[SIZE];
for (i=0; i < SIZE; i++) {data[i] =  i+1; }
Tsum = 0;
for (i=0; i < SIZE; i++) { Tsum = Tsum +
data[i]; }
printf ("Total Sum  %d \n", Tsum);
return (0);
}
```

Παράδειγμα Παραλληλοποίησης Προγράμματος με την μέθοδο: Domain Decomposition (1/2)

- **Πρόβλημα:** Να γραφεί μια παράλληλη μορφή του προηγούμενου προγράμματος, όπου κάθε διεργασία (εκτός της διεργασίας 0) θα αναλάβει να εκτελέσει ένα τμήμα του συνολικού αθροίσματος των στοιχείων του πίνακα. Το συνολικό αποτέλεσμα να προκύπτει από το άθροισμα των επιμέρους αθροισμάτων από την διεργασία 0.





- **Δεν υπάρχει κάποια εξάρτηση στα δεδομένα. Κάθε διεργασία μπορεί να ανατεθεί σε έναν επεξεργαστή. Υπάρχει εξάρτηση στο συνολικό άθροισμα.**

Παράδειγμα Παραλληλοποίησης Προγράμματος με την μέθοδο: Domain Decomposition (2/2)

- Το πρόγραμμα αυτό μπορεί να γίνει με OpenMPI στο οποίο υπολογίζονται ποια δεδομένα θα σταλούν σε κάθε rank (μπορεί να χρησιμοποιηθεί η συνάρτηση scatter).
- Το κάθε rank χρησιμοποιεί τα δεδομένα που του έχουν σταλεί και κάνει τον υπολογισμό. Στο τέλος στέλνει στο αρχικό rank το άθροισμα.
- Το αρχικό rank προσθέτει τα μερικά αθροίσματα και υπολογίζει το συνολικό.

Συνοψίζοντας

- Υπάρχουν 2 κύριοι τρόποι διάσπασης ενός αλγορίθμου σε εργασίες:
  - Domain Decomposition (Διάσπαση Πεδίου Ορισμού).
  - Functional Decomposition (Λειτουργική Διάσπαση).
- Μελετάμε το πρόβλημα αν όντως είναι παραλληλοποιήσημο:
  - Ο γράφος εξάρτησης αποκαλύπτει σχέσεις εξάρτησης μεταξύ των εργασιών.
  - Η ύπαρξη της σειριακής μορφής ενός προγράμματος βοηθά την ανάλυση για την κατασκευή της παράλληλης μορφής, καθώς επίσης και για επαλήθευση των αποτελεσμάτων.

Βήμα 2ο: Ανάθεση εργασιών σε Διεργασίες

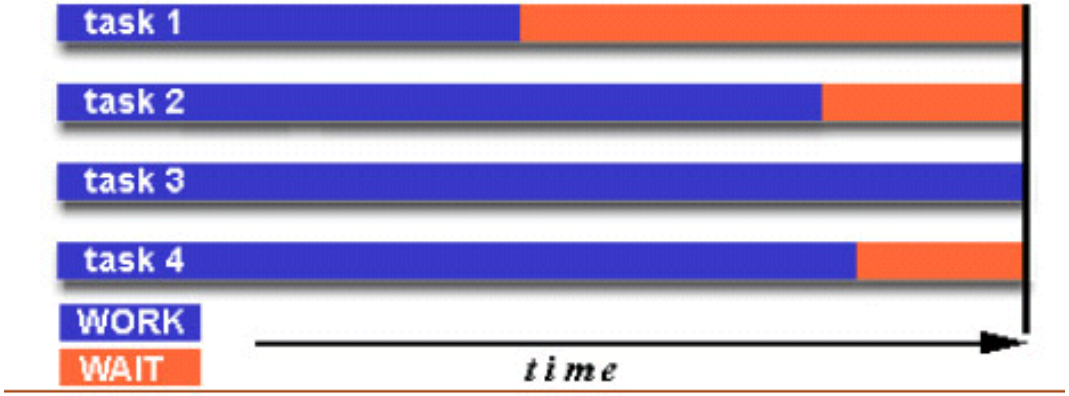
- Στο βήμα αυτό αναζητείται η βέλτιστη ομαδοποίηση των εργασιών, ώστε κάθε ομάδα που καλείται διεργασία να εκτελείται σε έναν και μόνο επεξεργαστή.
- Οι στόχοι της ανάθεσης είναι τρεις:
  1. Η εξισορρόπηση του φόρτου εργασίας.
  2. Η ελαχιστοποίηση της επικοινωνίας μεταξύ των διεργασιών.
  3. Η μείωση του χρόνου που απαιτείται για τη διαχείριση της ανάθεσης.

Η εξισορρόπηση του φόρτου εργασίας (load balancing) (1/2)

- Η εξισορρόπηση αυτή σχετίζεται:
  A. Με το ποσοστό του κώδικα που εκτελεί κάθε διεργασία.
  B. Με τη χρήση της εισόδου και της εξόδου.
  C. Με την χρήση της μνήμης.
  D. Με την επικοινωνία μεταξύ των διεργασιών.
- Επιδιώκεται να ισοκατανέμεται ο φόρτος υπολογισμού, ώστε οι διεργασίες να έχουν όσο το δυνατόν μικρότερη διαφορά στους χρόνους περάτωσης. Η πιο αργή διεργασία καθορίζει και τον μέγιστο χρόνο εκτέλεσης του προγράμματος.

Η εξισορρόπηση του φόρτου εργασίας (load balancing) (2/2)

- Οι 4 εργασίες έχουν να εκτελέσουν κάποιο κομμάτι κώδικα και έχουν μεταξύ τους διαφορετικό χρόνο περάτωσης της δουλειάς που τους έχει ανατεθεί.
- Αποτέλεσμα: Η εργασία 3 είναι η πιο χρονοβόρα και καθυστερεί την ολοκλήρωση ολόκληρου του υπολογισμού. Επιπλέον: στην εργασία 1 έχει ανατεθεί συγκριτικά αρκετά μικρότερος όγκος υπολογισμού.

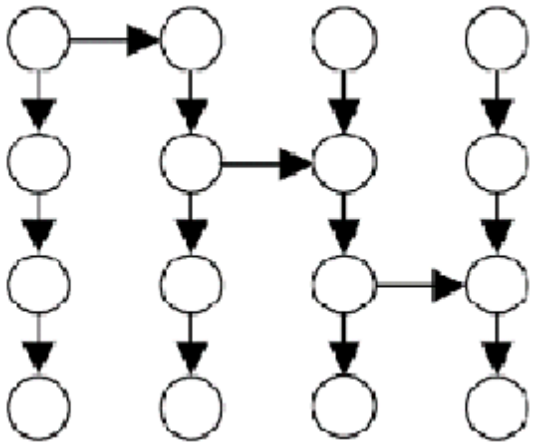


Η ελαχιστοποίηση της επικοινωνίας μεταξύ των διεργασιών (1/3)

- Η επικοινωνία μεταξύ διεργασιών συνεπάγεται τελικά επικοινωνία μεταξύ επεξεργαστών. Κοστίζει σε: Α) Χρήμα Β) Χρόνο.
- **Χρήμα**: Διότι ο σχεδιασμός συνεπάγεται και ένα κόστος σε εξοπλισμό. Μη συνετός σχεδιασμός==> Υψηλό κόστος χωρίς βέλτιστη απόδοση.
- **Χρόνο**: Λόγο της τοπολογίας του Δικτύου διασύνδεσης πολλές φορές η επικοινωνία απαιτεί πολλά βήματα. Επιπλέον η συχνή ανταλλαγή μηνυμάτων προκαλεί κυκλοφοριακή συμφόρηση (φαινόμενο bottleneck) και μειώνεται η συνολική απόδοση.
- Επομένως, η ανταλλαγή μηνυμάτων πρέπει να περιορίζεται σε όσο το δυνατόν χαμηλότερα επίπεδα.

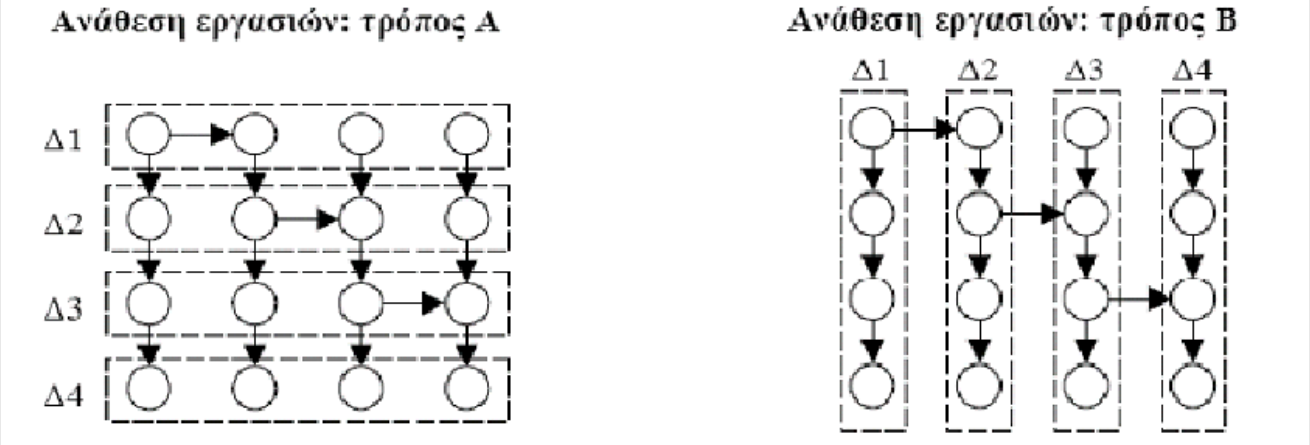
Η ελαχιστοποίηση της επικοινωνίας μεταξύ των διεργασιών (2/3)

- Οι εργασίες που τροφοδοτούν η μια την άλλη με σειριακό τρόπο (δημιουργώντας μια αλυσίδα στον γράφο εξάρτησης) είναι προτιμότερο να ανήκουν στην ίδια διεργασία. Γιατί;
- Οι εργασίες εκτελούνται όλες στον ίδιο επεξεργαστή ==> Με αυτόν τον τρόπο **η επικοινωνία είναι πιο γρήγορη** και επιπλέον δεν φορτώνεται το δίκτυο.
- Παράδειγμα: 12 εργασίες που επικοινωνούν σε διάφορα σημεία, με τον παρακάτω γράφο εξάρτησης:



Η ελαχιστοποίηση της επικοινωνίας μεταξύ των διεργασιών (3/3)

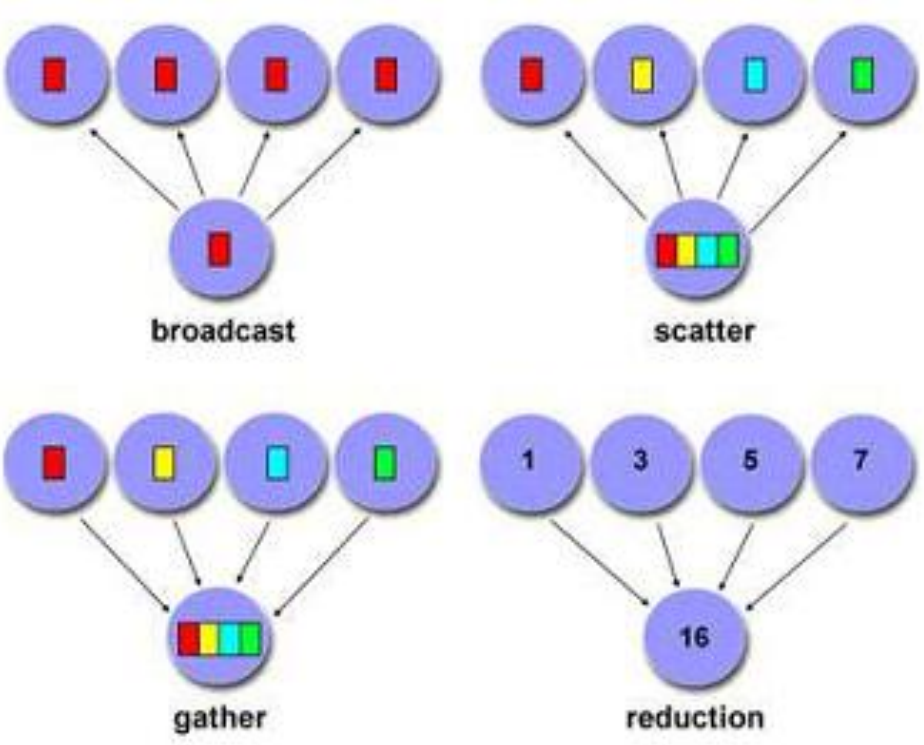
- Δύο πιθανές ομαδοποιήσεις των διεργασιών σε ομάδες των 4 εργασιών:



- Κέρδος: Μόνο 3 γραμμές στον Β τρόπο επικοινωνίας, έναντι 12 στον Α.
- Γενικά επιδιώκεται να γίνεται επικοινωνία των εργασιών μέσα στην ίδια διεργασία (εσωτερική επικοινωνία).

Είδη Επικοινωνίας

- Από σημείο-προς-σημείο.
- Συλλογικές (broadcast, scatter, gather, reduction).



Η μείωση του χρόνου που απαιτείται για τη διαχείριση της ανάθεσης

- Η απόφαση της ανάθεσης εργασιών σε ομάδες - διεργασίες μπορεί να γίνει με 3 τρόπους:
  - **Α. Στατικός Τρόπος**: Όταν ο γράφος εξάρτησης είναι σχετικά απλός γιατί υπάρχουν ανεξάρτητες σειριακές αλυσίδες εργασιών και έτσι η ομαδοποίηση τους είναι εύκολη ==>Ταχεία διαδικασία ανάθεσης.
  - **Β. Ημι-στατικός τρόπος**: Οι εργασίες ομαδοποιούνται με κάποιον ευρηστικό αλγόριθμο, οι ομαδοποιήσεις δεν είναι πάντα οι βέλτιστες.

Συνήθως οι ομαδοποιήσεις αναθεωρούνται ώστε να βρεθούν βέλτιστες.

  - **C. Δυναμικός τρόπος ανάθεσης**: Χρησιμοποιείται όταν η εξέλιξη των εργασιών είναι απρόβλεπτη (π.χ. κάποια μνήμη κάνει πιο συχνά swap στον δίσκο).

Χρονοδρομολόγηση των διεργασιών σε επεξεργαστές (1/4)

- Στο βήμα αυτό το σύνολο των διεργασιών που έχουν παραχθεί στα βήματα 1 και 2 πρέπει να ανατεθούν προς εκτέλεση σε κάποιους επεξεργαστές. Επομένως εδώ πρέπει να δοθεί μια απάντηση στα εξής ερωτήματα:
  A. Πού θα τρέξω την εφαρμογή; (Σε ποιους επεξεργαστές).
  B. Πότε θα τρέξω την εφαρμογή; (Σε ποια χρονική στιγμή).
- Η απάντηση στις 2 παραπάνω ερωτήσεις δίνεται από αλγόριθμους που αναλαμβάνουν την κατάρτιση χρονοδιαγράμματος εκτέλεσης των εργασιών που λέγεται χρονοδρομολόγηση (scheduling).

Χρονοδρομολόγηση των διεργασιών σε επεξεργαστές (2/4)

- **Σημαντικό:** Να ανατεθούν οι διεργασίες στους κατάλληλους επεξεργαστές ώστε να ελαχιστοποιηθεί ο συνολικός χρόνος εκτέλεσης της εφαρμογής.
- Οι μέθοδοι χρονοδρομολόγησης διακρίνονται σε 2 κατηγορίες:
  A. **Ντετερμινιστικές μέθοδοι** χρονοδρομολόγησης (σε γενικές γραμμές ο χρόνος εκτέλεσης είναι σταθερός).
  B. **Μη ντετερμινιστικές μέθοδοι** χρονοδρομολόγησης (ο χρόνος εκτέλεσης δεν είναι σταθερός).

Χρονοδρομολόγηση των διεργασιών σε επεξεργαστές (3/4)

- Οι μέθοδοι χρονοδρομολόγησης κατηγοριοποιούνται επίσης και με την δυναμική ή όχι ανάθεση των διεργασιών:

Α. Στατικές μέθοδοι χρονοδρομολόγησης

– Αποφασίζεται εξαρχής σε ποιόν επεξεργαστή θα εκτελεστεί η κάθε διεργασία. Εφόσον γίνει αυτή η ανάθεση, η διεργασία εκτελείται στον ίδιο επεξεργαστή κάθε φορά που ζητείται η εκτέλεσή της ανεξάρτητα από τον φόρτο εργασίας.

Β. Δυναμικές μέθοδοι χρονοδρομολόγησης

– Η ανάθεση σε κάποιον επεξεργαστή αποφασίζεται κατά την ώρα εκτέλεσης του προγράμματος λαμβάνοντας υπόψιν τον φόρτο εργασίας κάθε επεξεργαστή.

Χρονοδρομολόγηση των διεργασιών σε επεξεργαστές (4/4)

- **Πλεονέκτημα** των στατικών μεθόδων δρομολόγησης: Η ταχύτητα απόφασης είναι μεγάλη αφού η ανάθεση έχει προαποφασιστεί και δεν αλλάζει.
- **Πλεονέκτημα** των δυναμικών μεθόδων: Κάνουν καλύτερη διαχείριση του συστήματος αφού η ευελιξία τους στην ανάθεση διεργασιών εξισορροπεί το φόρτο λειτουργίας των επεξεργαστών και διευκολύνει την καλύτερη απόδοση του συστήματος.
- **Μειονέκτημα** των στατικών μεθόδων είναι η μη εξισορρόπηση του φόρτου λειτουργίας των επεξεργαστών.
- **Μειονέκτημα** των δυναμικών μεθόδων: σπαταλούν χρόνο εκτέλεσης για να αποφασίσουν εκείνη τη στιγμή τις αναθέσεις.

Ντετερμινιστικό Μοντέλο με μηδενικό χρόνο επικοινωνίας

Ντετερμινιστικές Μέθοδοι Χρονοδρομολόγησης

- Στις ντετερμινιστικές μεθόδους χρονοδρομολόγησης θεωρείται ότι οι σχέσεις μεταξύ των εργασιών και ο χρόνος εκτέλεσης είναι σταθερός και γνωστός εκ των προτέρων.
- Στην πράξη δεν ισχύει πάντα, π.χ. περιπτώσεις όπου υπάρχουν άλματα στον αλγόριθμο υπό συνθήκη ή καθυστερήσεις λόγο εμφάνισης swaps στον δίσκο.

Ντετερμινιστικό Μοντέλο με μηδενικό χρόνο επικοινωνίας (1/5)

Κάνουμε τις εξής δύο προσεγγίσεις:

A. Οι χρόνοι εκτέλεσης των εργασιών είναι εκ των προτέρων γνωστοί και δεν εξαρτώνται από αστάθμητους παράγοντες, όπως αποτυχίες της cache, disk swaps, κλπ. (ντετερμινιστικό μοντέλο - δηλαδή δεν υπεισέρχεται ο παράγων τύχη).

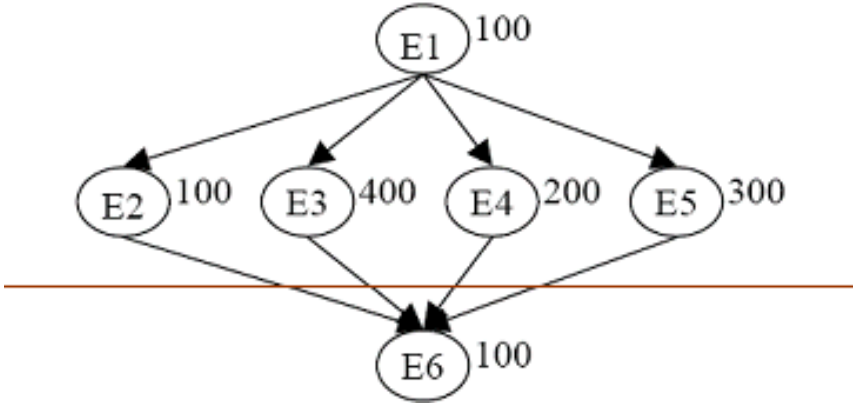
B. Ο χρόνος επικοινωνίας μεταξύ των επεξεργαστών είναι μηδενικός.

- Για να καταλάβουμε τη σημασία του συνδυασμού ανάθεσης και χρονοδρομολόγησης ακόμη και στην απλή αυτή περίπτωση ας θεωρήσουμε τον αλγόριθμο που περιγράφεται από τον παρακάτω ντετερμινιστικό Γράφο Εξάρτησης.

Ντετερμινιστικό Μοντέλο με μηδενικό χρόνο επικοινωνίας (2/5)

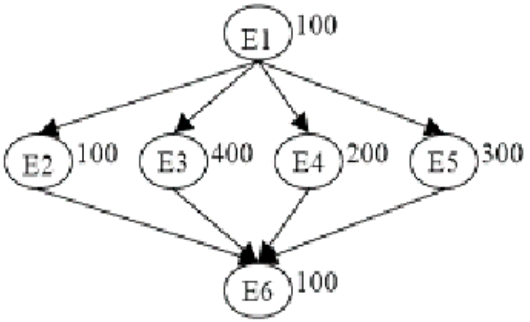
- Κάθε κόμβος του γράφου είναι και μια διαφορετική εργασία (task) η οποία μπορεί να εξαρτάται από τα αποτελέσματα κάποιων άλλων εργασιών ενώ ταυτόχρονα μπορεί να τροφοδοτεί άλλες εργασίες με τα δικά της αποτελέσματα.
- Δίπλα σε κάθε εργασία αναγράφεται και ο χρόνος εκτέλεσής της σε υποθετικές μονάδες χρόνου (π.χ. nsec, ή κύκλοι επεξεργαστή).

Παράδειγμα αλγόριθμου με ντετερμινιστικό Γράφο Εξάρτησης:

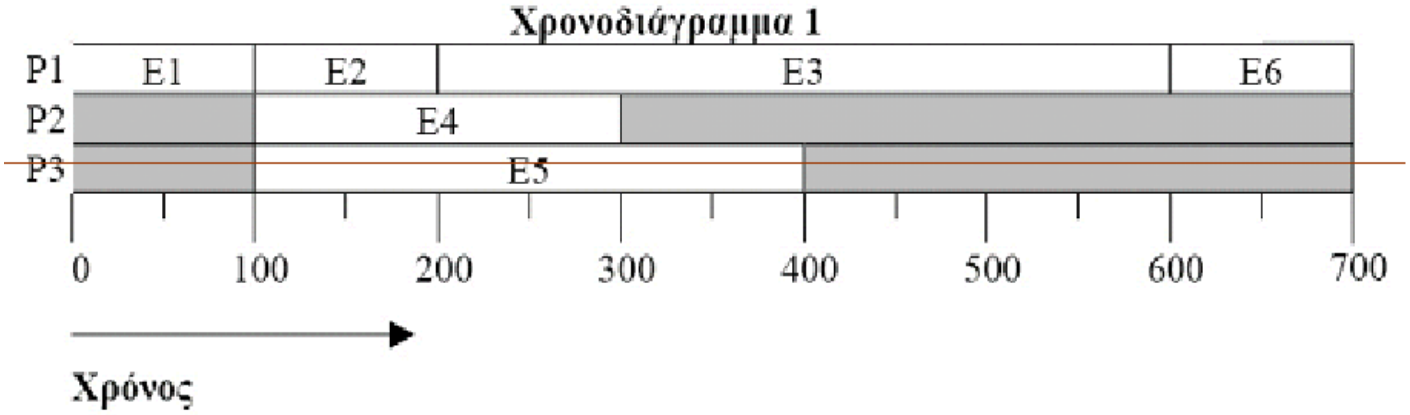


Ντετερμινιστικό Μοντέλο με μηδενικό χρόνο επικοινωνίας (3/5)

- Δίνονται δύο διαφορετικές αναθέσεις και χρονοδιαγράμματα εκτέλεσης σε ένα σύστημα με τρεις επεξεργαστές:

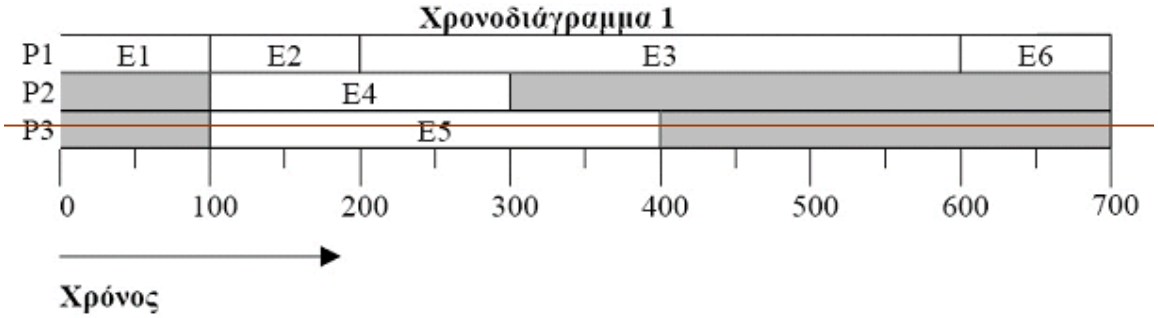


- P1, P2, P3, με βάση τις παραπάνω παραδοχές. Ανάθεση 1:



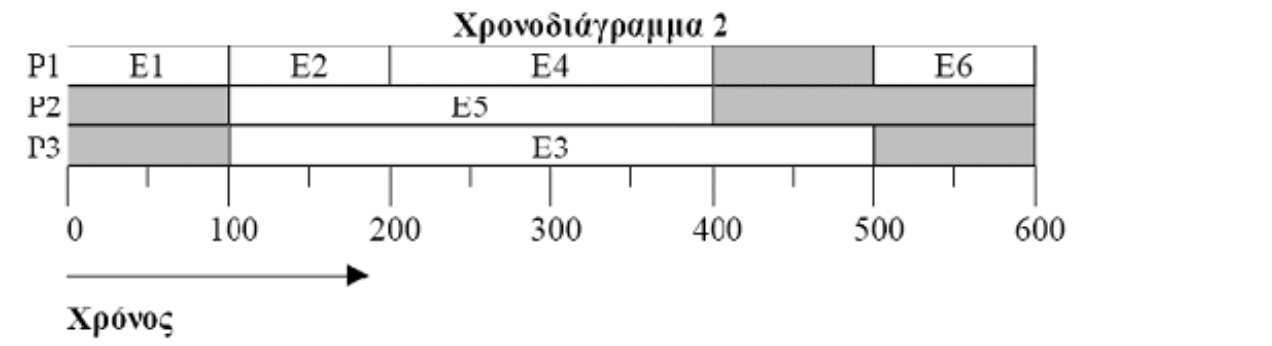
Ντετερμινιστικό Μοντέλο με μηδενικό χρόνο επικοινωνίας (4/5)

- Χαρακτηριστικά 1ης ανάθεσης:
  - Οι εργασίες Ε1, Ε2, Ε3, Ε6 αποτελούν την πρώτη διεργασία, η οποία εκτελείται στον επεξεργαστή Ρ1.
  - Η εργασία Ε4 αποτελεί τη δεύτερη διεργασία που εκτελείται στον Ρ2.
  - Η εργασία Ε5 αποτελεί την τρίτη διεργασία που εκτελείται στον Ρ3.



Ντετερμινιστικό Μοντέλο με μηδενικό χρόνο επικοινωνίας (5/5)

- 2η ανάθεση:



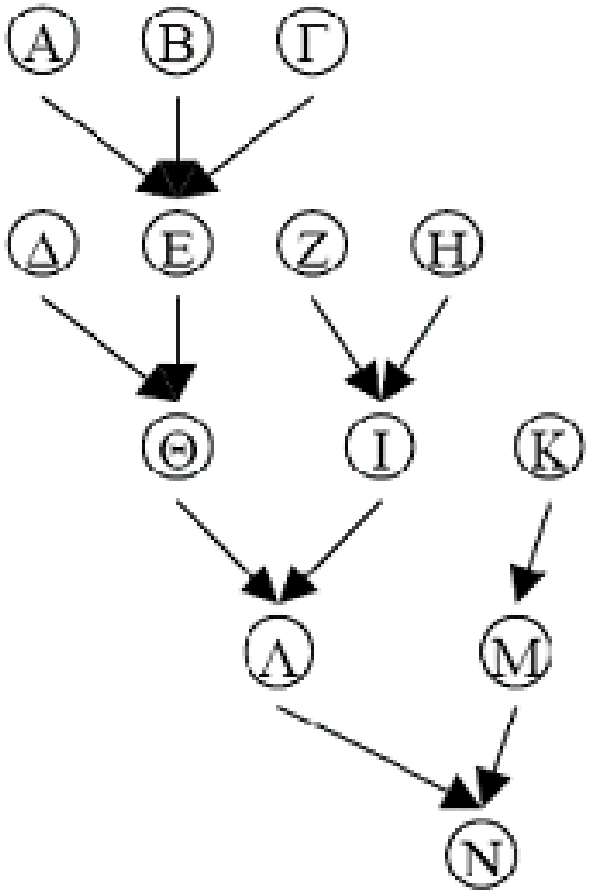
- Οι εργασίες E1, E2, E4, E6 αποτελούν την πρώτη διεργασία η οποία εκτελείται στον επεξεργαστή P1.
- Η εργασία E5 αποτελεί τη δεύτερη διεργασία που εκτελείται στον P2.
- Η εργασία E3 αποτελεί την τρίτη διεργασία που εκτελείται στον P3.

Συμπεράσματα αναθέσεων

- Από τα παραπάνω φαίνεται ότι οι διαφορετικές αναθέσεις σε συνδυασμό με τα διαφορετικά χρονοδιαγράμματα δίνουν διαφορετικά αποτελέσματα ως προς το συνολικό χρόνο εκτέλεσης Τ του αλγορίθμου.
- Ο πρώτος τρόπος χρονοδρομολόγησης δίνει **Τ=700**, ενώ ο δεύτερος δίνει **Τ=600**.
- Επομένως, έχοντας σαν δεδομένα:
  A. το γράφο εξάρτησης των εργασιών.
  B. τους χρόνους εκτέλεσης της κάθε εργασίας.
  C. το πλήθος των διαθέσιμων επεξεργαστών.
- **Πώς μπορούμε να βρούμε την καλύτερη ανάθεση** και το καλύτερο χρονοδιάγραμμα; Απάντηση: **Δεν υπάρχουν βέλτιστες λύσεις** γενικής μορφής, αλλά κατά περίπτωση.

Περίπτωση όπου ο γράφος εξάρτησης των εργασιών είναι δάσος εισόδου (1/5)

- Ορισμός: Ένας γράφος εξάρτησης (ΓΕ) καλείται δάσος εισόδου όταν κάθε κόμβος έχει μόνο μια ακμή εξόδου. Αντίστοιχα καλείται δάσος εξόδου αν κάθε κόμβος έχει μόνο μια ακμή εισόδου.
- Παράδειγμα: Ο παρακάτω ΓΕ είναι ένα δάσος εισόδου (διότι κάθε κόμβος έχει μια μόνο έξοδο ενώ οι είσοδοι κάθε κόμβου αποτελούν ένα "δάσος").

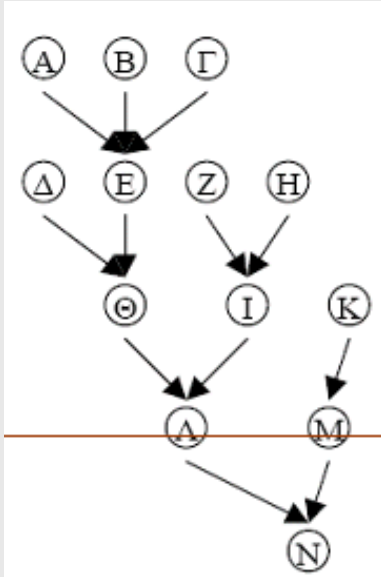


Περίπτωση όπου ο γράφος εξάρτησης των εργασιών είναι δάσος εισόδου (2/5)

- Στην συγκεκριμένη περίπτωση χρησιμοποιώντας την έννοια της προτεραιότητας μπορούμε να κάνουμε βέλτιστη ανάθεση και βέλτιστη χρονοδρομολόγηση.
- Διαδικασία:
  - Δίνουμε μεγαλύτερη προτεραιότητα στους κόμβους ανάλογα με την απόστασή τους από τον τελευταίο κόμβο του γράφου.
  - Ο κόμβος Ν είναι τελευταίος=> παίρνει προτεραιότητα 1.
  - Οι κόμβοι Λ, Μ, του αμέσως προηγούμενου επιπέδου παίρνουν προτεραιότητα 2.
  - Οι κόμβοι Θ, Ι, Κ, του πιο πάνω επιπέδου παίρνουν προτεραιότητα 3. κ.ο.κ.
  - Με τον τρόπο αυτό προκύπτει ο παρακάτω πίνακας όλων των προτεραιοτήτων.

Περίπτωση όπου ο γράφος εξάρτησης των εργασιών είναι δάσος εισόδου (3/5)

- Πίνακας όλων των προτεραιοτήτων που προκύπτουν από τον ΓΕ.

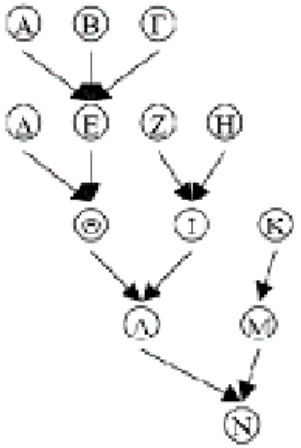


Κόμβος	Προτεραιότητα
Α	5
Β	5
Γ	5
Δ	4
Ε	4
Ζ	4
Η	4
Θ	3
Ι	3
Κ	3
Λ	2
Μ	2
Ν	1

- Στη συνέχεια αναθέτουμε την εκτέλεση των εργασιών με σειρά προτεραιότητας στους διαθέσιμους επεξεργαστές λαμβάνοντας υπόψιν τις ακμές εξάρτησης. Αν για παράδειγμα έχουμε 3 επεξεργαστές P1, P2, P3, τότε ο πίνακας χρονισμού έχει ως εξής:

Περίπτωση όπου ο γράφος εξάρτησης των εργασιών είναι δάσος εισόδου (4/5)

	t=1	t=2	t=3	t=4	t=5	t=6
P1	A^5	Δ^4	H^4	I^3	Λ^2	N^1
P2	B^5	E^4	Θ^3	M^2		
P3	Γ^5	Z^4	K^3			



- Ο εκθέτης που αναγράφεται σε κάθε εργασία αντιστοιχεί στην προτεραιότητά της. Τα γκρίζα βελάκια αντιστοιχούν στις εξαρτήσεις μεταξύ των εργασιών (σύμφωνα με το γράφο εξάρτησης).
- Η χρονοδρομολόγηση γίνεται, αναθέτοντας πρώτα τις εργασίες με τη μεγαλύτερη προτεραιότητα στους επεξεργαστές που είναι διαθέσιμοι και κατόπιν τις εργασίες με τη μικρότερη προτεραιότητα.

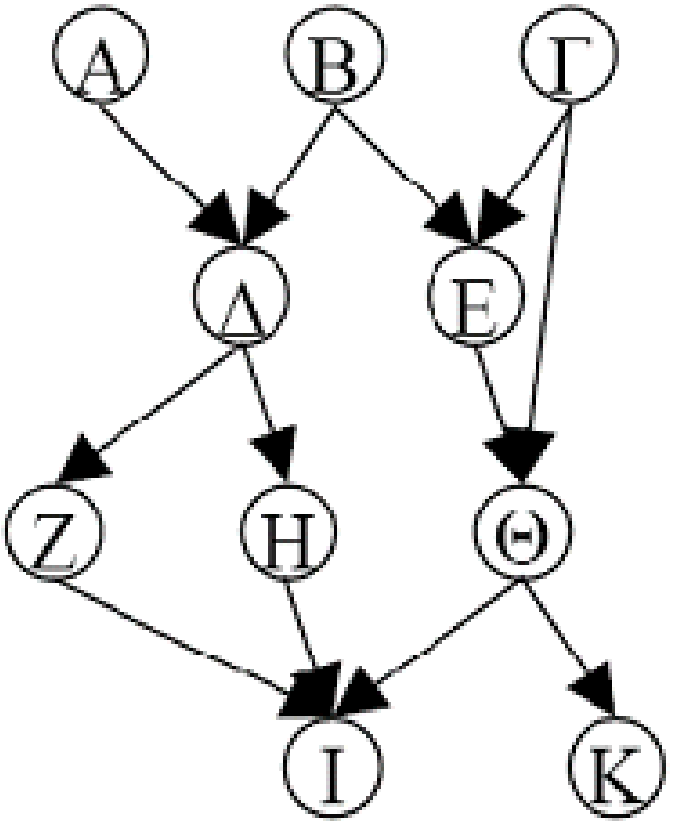
Περίπτωση όπου ο γράφος εξάρτησης των εργασιών είναι δάσος εισόδου (5/5)

- Αναθέτουμε πρώτα τις τρεις εργασίες Α, Β, Γ με τη μεγαλύτερη προτεραιότητα στους τρεις επεξεργαστές για κύκλο t=1.
- Έπειτα αναθέτουμε τις εργασίες Δ, Ε, Ζ, κατά τον κύκλο t=2.
- Στον κύκλο t=3 αναθέτουμε μόνο τις εργασίες Η, Θ, Κ, και όχι την εργασία Ι διότι πρέπει να προηγείται η Η από τη Ι.
- Στον κύκλο 4 δε μπορούμε να εκτελέσουμε την Λ διότι εξαρτάται από την Ι.
- Στον κύκλο 5 δεν μπορούμε να εκτελέσουμε τη Ν διότι εξαρτάται από τη Λ. Έτσι ο πίνακας χρονισμού που προκύπτει είναι ο καλύτερος δυνατός όσον αφορά το συνολικό χρόνο εκτέλεσης του αλγορίθμου.

Ειδική περίπτωση: Εάν υπάρχουν μόνο 2 επεξεργαστές (1/6)

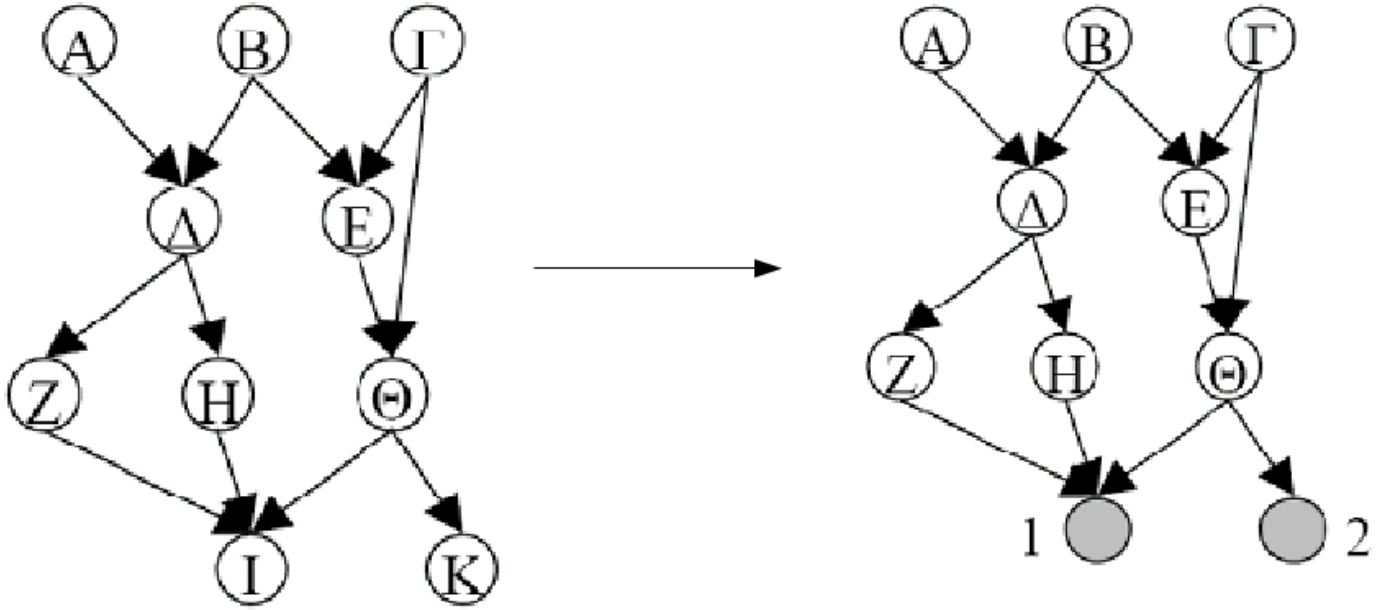
- Ποιος είναι ο αλγόριθμος ανάθεσης και χρονοδρομολόγησης;
- Όποια και εδώ η βασική ιδέα είναι η ανάθεση προτεραιότητας στους κόμβους.
- Παράδειγμα:

Έστω ο γράφος εξάρτησης:



Ειδική περίπτωση: Εάν υπάρχουν μόνο 2 επεξεργαστές (2/6)

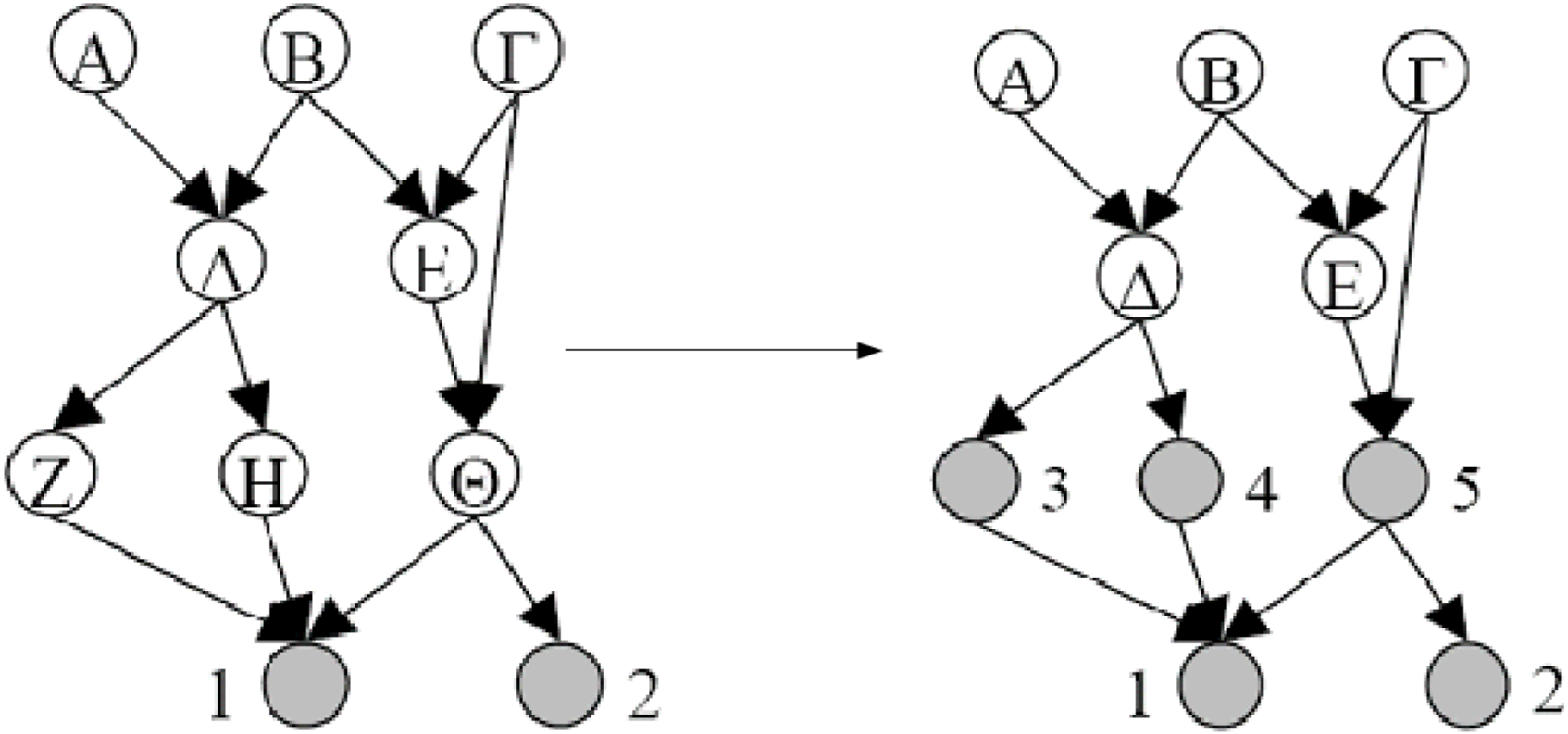
- Βήμα 1. Δίνουμε αυθαίρετα προτεραιότητες 1, 2, ..., Μ στα φύλλα του γράφου, δηλαδή στους κόμβους που δεν έχουν παιδιά. Στη συγκεκριμένη περίπτωση οι κόμβοι αυτοί είναι δύο: Ι, Κ. Χρωματίζουμε τους κόμβους αυτούς γκρι.



Ειδική περίπτωση: Εάν υπάρχουν μόνο 2 επεξεργαστές (3/6)

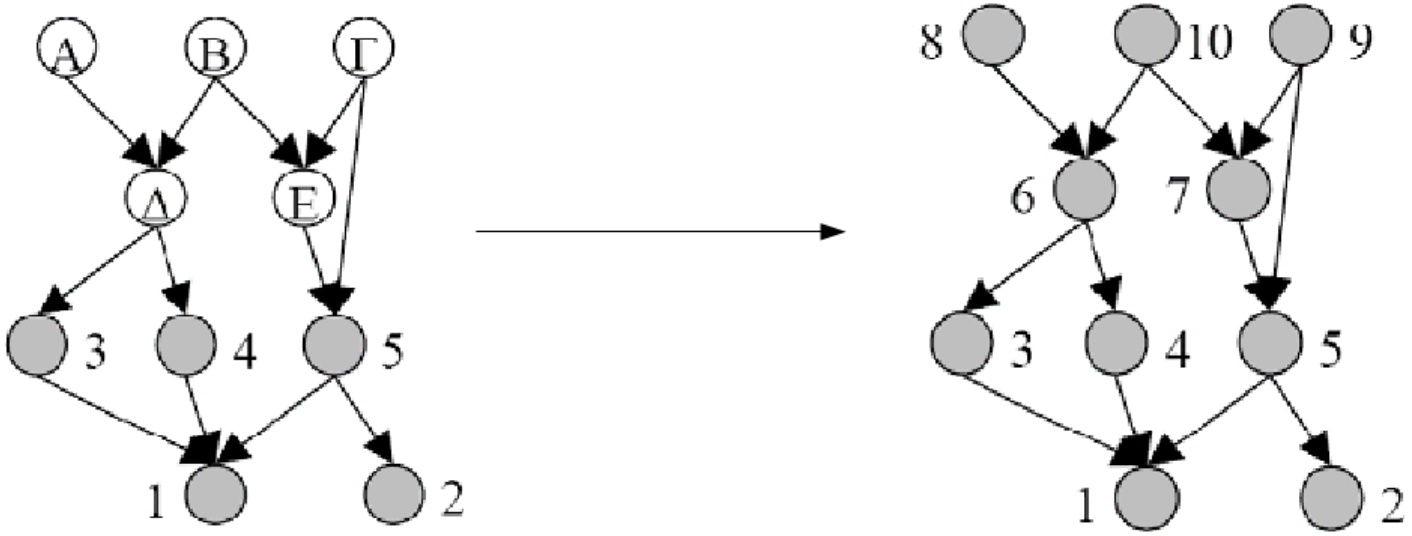
- Βήμα 2. Βρίσκουμε όλους τους κόμβους που έχουν όλα τους τα παιδιά γκρι.
- Δηλαδή οι κόμβοι: Ζ, Η, Θ.
- Τους κόμβους αυτούς τους κατατάσσουμε σε σειρά **ανάλογα με τη μεγαλύτερη προτεραιότητα** των παιδιών τους=> Ο Θ έχει μεγαλύτερη προτεραιότητα από τους Ζ, Η, διότι το παιδί του Θ, το Κ, έχει προτεραιότητα 2 ενώ τα παιδιά των Ζ, Η, έχουν προτεραιότητα 1. Δίνουμε στους κόμβους αυτούς προτεραιότητες Μ+1, ..., Μ', δηλαδή, Ζ=3, Η=4, Θ=5
- Χρωματίζουμε τους κόμβους Ζ, Η, Θ, γκρι.

Ειδική περίπτωση: Εάν υπάρχουν μόνο 2 επεξεργαστές (4/6)



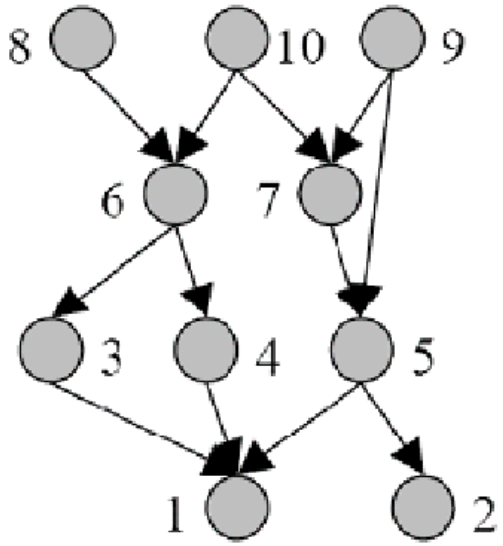
Ειδική περίπτωση: Εάν υπάρχουν μόνο 2 επεξεργαστές (5/6)

- Βήμα 3. Επαναλαμβάνουμε το Βήμα 2 μέχρι όλοι οι κόμβοι να χρωματιστούν γκρι οπότε έχουν όλοι και μια διαφορετική προτεραιότητα.
- Έτσι τελικά παίρνουμε τις παρακάτω προτεραιότητες:



Ειδική περίπτωση: Εάν υπάρχουν μόνο 2 επεξεργαστές (6/6)

- Βήμα 4. Αναθέτουμε τις εργασίες με σειρά προτεραιότητας (μεγαλύτερη προτεραιότητα πρώτα) στον πρώτο επεξεργαστή που είναι ελεύθερος. Έτσι παίρνουμε το χρονοδιάγραμμα του παρακάτω πίνακα:



	t=1	t=2	t=3	t=4	t=5
P1	B^{10}	A^{8}	Δ^{6}	H^{4}	K^{2}
P2	Γ^{9}	E^{7}	Θ^{5}	Z^{3}	I^{1}

Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας

Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (1/7)

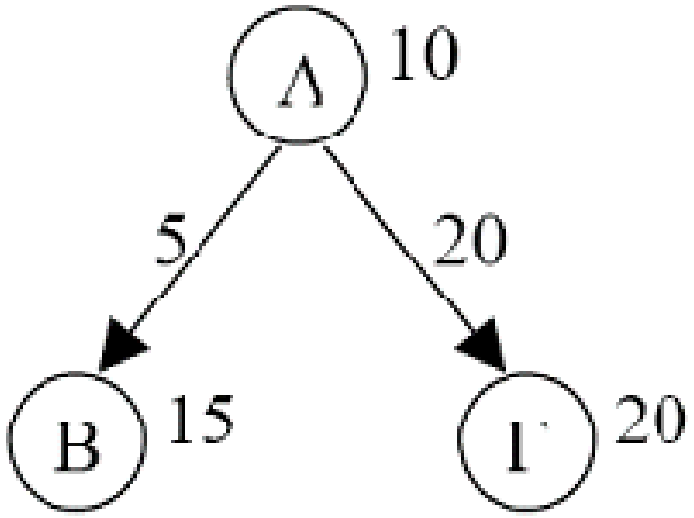
- Στην περίπτωση αυτή υποθέτουμε ότι ο χρόνος εκτέλεσης κάθε εργασίας είναι πάλι γνωστός και δεν εξαρτάται από τυχαίες παραμέτρους, **αλλά το κόστος** (δηλ. ο χρόνος) επικοινωνίας μεταξύ των διεργασιών **δεν είναι αμελητέο.**
- Ο χρόνος επικοινωνίας μεταξύ διεργασιών είναι ουσιαστικά ο χρόνος που απαιτείται για την ανταλλαγή δεδομένων μεταξύ των επεξεργαστών που απαιτείται για την ανταλλαγή δεδομένων μεταξύ των επεξεργαστών που εκτελούν αυτές τις εργασίες.
- Αν οι εργασίες Α, Β δεν ανταλλάσσουν δεδομένα ή αν εκτελούνται στον ίδιο επεξεργαστή τότε θεωρούμε ότι ο χρόνος επικοινωνίας Α`Β είναι μηδενικός.
- Αν υπάρχει ακμή που ενώνει τον κόμβο Α με τον κόμβο Β του ΓΕ τότε οι εργασίες Α, Β ανταλλάσσουν δεδομένα μεταξύ τους. Στην περίπτωση αυτή υπάρχει μη μηδενικό κόστος επικοινωνίας Α`Β αν και μόνο αν οι εργασίες Α, Β εκτελούνται σε διαφορετικούς επεξεργαστές.

Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (2/7)

- Όταν λοιπόν κάνουμε ανάθεση και δρομολόγηση εργασιών προσπαθούμε να βάλουμε τις χρονοβόρες ακμές μέσα στην ίδια διεργασία έτσι ώστε να εκτελεστούν από τον ίδιο επεξεργαστή και να μηδενιστεί το κόστος επικοινωνίας.
- Η χρονοδρομολόγηση γίνεται τώρα πιο πολύπλοκη απ' ότι στην περίπτωση όπου δε λαμβάνουμε υπ' όψη ας το κόστος επικοινωνίας (μοντέλο Α) Παρ' όπου δε λαμβάνουμε υπ' όψη ας το κόστος επικοινωνίας (μοντέλο Α) Παρ' όλ' αυτά το μοντέλο αυτό είναι πιο ρεαλιστικό από το μοντέλο Α.
- Η εύρεση της βέλτιστης ανάθεσης και της βέλτιστης χρονοδρομολόγησης είναι δύσκολη. Στην περίπτωση αυτή χρησιμοποιούνται ευρηστικές μέθοδοι (heuristics) οι οποίες προσπαθούν να καλύψουν το κόστος επικοινωνίας και να αναθέσουν τις εργασίες στους διάφορους επεξεργαστές.

Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (3/7)

- Έστω ότι έχουμε τον παρακάτω γράφο εξάρτησης όπου δίπλα σε κάθε κόμβο αναγράφεται ο χρόνος εκτέλεσης κάθε εργασίας και δίπλα σε κάθε ακμή αναγράφεται ο χρόνος επικοινωνίας αν η ακμή αυτή μεταφέρει δεδομένα ανάμεσα σε δύο διαφορετικούς επεξεργαστές.

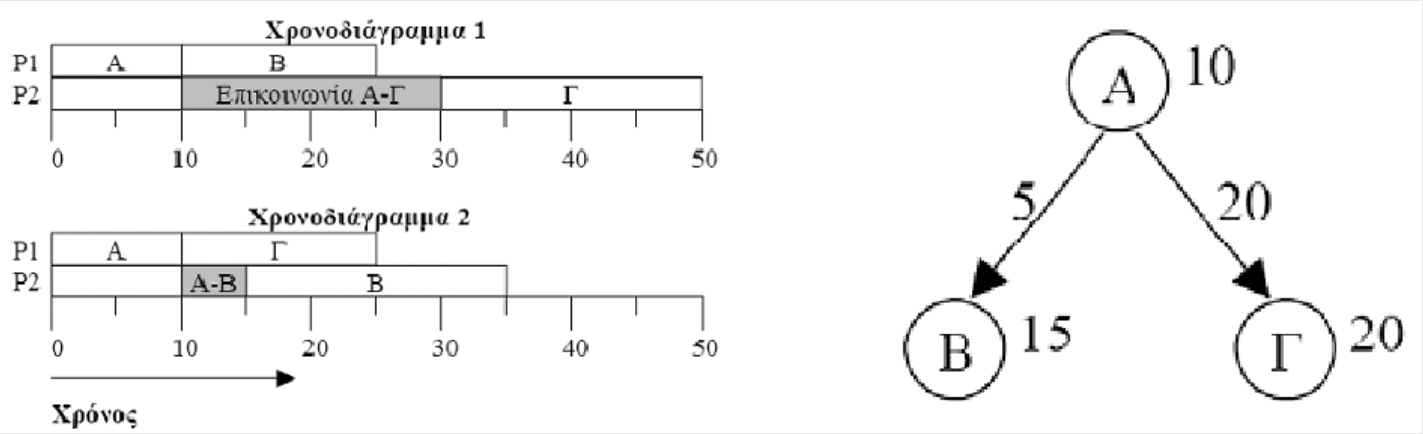


Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (4/7)

- Γενικά, είναι προτιμότερο να αναθέτουμε τους κόμβους που **συνδέουν ακμές με μεγάλο κόστος επικοινωνίας**, όπως η ακμή Α`Γ, στην ίδια διεργασία έτσι ώστε οι κόμβοι Α και Γ να εκτελεστούν τελικά στον ίδιο επεξεργαστή και να μην πληρώσουμε το κόστος επικοινωνίας Α`Γ.
- Υποθέτουμε ότι ο χρόνος που χρειάζεται ένας επεξεργαστής για να επικοινωνήσει με τον εαυτό του είναι μηδενικός.

Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (5/7)

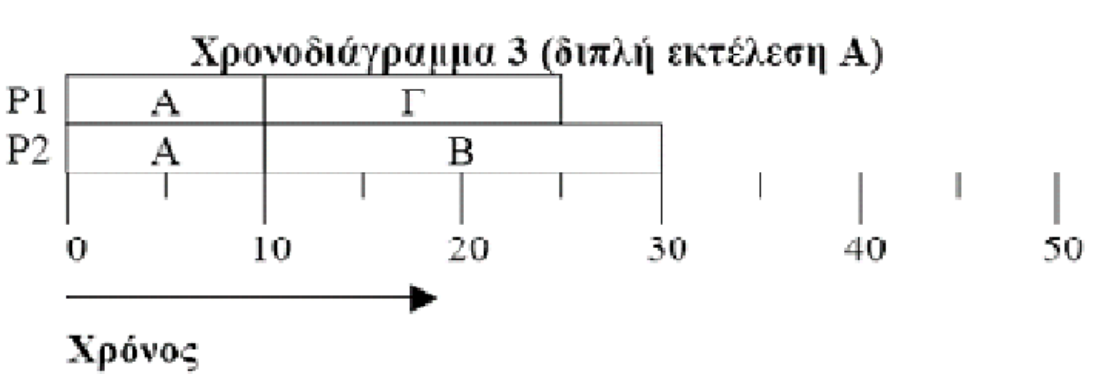
- Έστω για παράδειγμα ότι έχουμε δύο επεξεργαστές P1, P2. Δύο πιθανές αναθέσεις με τα αντίστοιχα χρονοδιαγράμματα έχουν ως εξής:

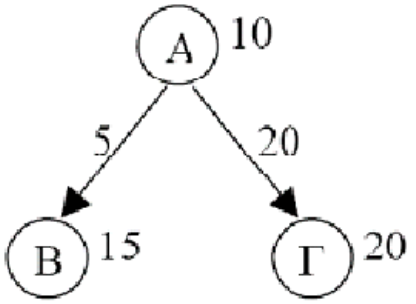


- Προφανώς το χρονοδιάγραμμα 2 είναι καλύτερο διότι αναθέτουμε τις εργασίες Α, Γ οι οποίες έχουν πολύ μεγάλο κόστος επικοινωνίας στον ίδιο επεξεργαστή και έτσι το κόστος επικοινωνίας μεταξύ τους μηδενίζεται.

Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (6/7)

- Μια άλλη μέθοδος για την μείωση του χρόνου επικοινωνίας είναι **η διπλή εκτέλεση** μιας διεργασίας σε δύο διαφορετικούς επεξεργαστές. Η εφαρμογή της μεθόδου αυτής γίνεται φανερή με το παρακάτω χρονοδιάγραμμα που αναφέρεται στον ίδιο γράφο εξάρτησης.





Ντετερμινιστικό Μοντέλο με μη μηδενικό χρόνο επικοινωνίας (7/7)

- Το χρονοδιάγραμμα αυτό έχει συνολικό χρόνο εκτέλεσης Τ=30 σε αντίθεση με το χρον/μα 1 που έχει Τ=50, και το χρον/μα 2 που έχει Τ=35.
- Αυτό επετεύχθη εκτελώντας την εργασία Α δύο φορές, μια στον επεξεργαστή Ρ1, και μια στον Ρ2.
- Έτσι αποφεύγουμε τελείως και το κόστος επικοινωνίας Α-Γ, και το κόστος επικοινωνίας Α-Β, αφού οι Α, Γ εκτελούνται στον ίδιο επεξεργαστή Ρ1 ενώ οι Α, Β, εκτελούνται στον ίδιο επεξεργαστή Ρ2.

Ευρηστικοί Αλγόριθμοι σε μη μηδενικό χρόνο επικοινωνίας (1/3)

Α. Δρομολόγηση με λίστα προτεραιότητας (list scheduling).

- Κάθε κόμβος του γράφου παίρνει μια τιμή προτεραιότητας. Δημιουργείται μια ουρά με εργασίες έτοιμες προς εκτέλεση οι οποίες κατατάσσονται με σειρά προτεραιότητας. Μια εργασία λέγεται έτοιμη προς εκτέλεση αν έχουν εκτελεστεί όλες οι εργασίες από τις οποίες λαμβάνει είσοδο.

Ευρηστικοί Αλγόριθμοι σε μη μηδενικό χρόνο επικοινωνίας (2/3)

- Για όσο η ουρά δεν είναι άδεια κάνε τα εξής (βήματα):
  1. Πάρε την εργασία που βρίσκεται στην κορυφή της ουράς.
  2. Επέλεξε ένα επεξεργαστή που τώρα δεν εργάζεται και ανάθεσε την εργασία σ’ αυτόν.
  3. Πρόσθεσε μια εργασία στην ουρά όταν είναι έτοιμη, δηλαδή όταν έχουν εκτελεστεί όλες οι εργασίες που την τροφοδοτούν με αποτελέσματα.

Ευρηστικοί Αλγόριθμοι σε μη μηδενικό χρόνο επικοινωνίας (3/3)

Β. Δρομολόγηση με ομαδοποίηση (clustering):

- Βήμα 1: Ανάθεση. Ομαλοποίησε τις εργασίες με κάποιο κριτήριο ελαχιστοποίησης των ακμών επικοινωνίας μεταξύ των ομάδων. Οι ακμές αυτές αντιστοιχούν σε επικοινωνία μεταξύ επεξεργαστών η οποία κοστίζει πολύ σε χρόνο.
- Βήμα 2: Απεικόνιση των ομάδων σε Ν επεξεργαστές. Αν υπάρχουν περισσότερες από Ν ομάδες (clusters) ένωσε κάποιες ομάδες μεταξύ τους ώστε να φτιαχτούν Ν ομάδες. Αντιστοίχησε κάθε ομάδα σε ένα διαφορετικό επεξεργαστή.
- Βήμα 3: Χρονοδρομολόγηση. Για κάθε επεξεργαστή όρισε μια σειρά εκτέλεσης των εργασιών έτσι ώστε να γίνονται σεβαστές οι ακμές εξάρτησης.

Μη Ντετερμινιστικές Μέθοδοι Χρονοδρομολόγησης (1/2)

- Εφαρμόζονται όταν ο χρόνος εκτέλεσης δεν είναι σταθερός και γνωστός εκ των προτέρων γιατί κάποιες πληροφορίες μπορεί να μην είναι γνωστές πριν την έναρξη της εκτέλεσης ενός προγράμματος.
- Π.χ. σε περιπτώσεις βρόγχων και διακλαδώσεις επιλογής (loop, if).
- Μπορεί δηλαδή ολόκληρα υποπρογράμματα να μην εκτελούνται. Όλα αυτά αυξάνουν την πολυπλοκότητα στην διαδικασία χρονοδρομολόγησης.
- Η διαδικασία χρονοδρομολόγησης γίνεται δυναμικά (on the fly).

Μη Ντετερινιστικές Μέθοδοι Χρονοδρομολόγησης (2/2)

- Μειονεκτήματα: μεγάλοι χρόνοι επικοινωνίας, μεγάλοι χρόνοι χρονοδρομολόγησης, ίσως συνεχής μετακίνηση εργασιών μεταξύ επεξεργαστών (task thrashing).
- Τελικά όμως, ο χρόνος συνολικής εκτέλεσης να είναι ικανοποιητικός, παρά τις καθυστερήσεις (όχι πάντα).

Βασικές Τεχνικές Σχεδιασμού Παράλληλων Αλγορίθμων

- Ισοζυγισμένα Δένδρα.
- Διπλασιασμός Δεικτών.
- Διαδρομή Euler.
- Διαμέριση.

Ισοζυγισμένα Δένδρα (1/2)

- Η δενδρική δομή είναι βασική δομή παράλληλου υπολογισμού.
- Δεν είναι μόνο δομή δεδομένων, αλλά δομή υπολογισμού.
- Δημιουργούμε ένα ισοζυγισμένο δένδρο.
- Υπολογίζονται οι εσωτερικοί κόμβοι από τα δεδομένα εισόδου.
- Χρησιμοποιούνται οι εσωτερικοί κόμβοι για τον υπολογισμό των επόμενων εσωτερικών κόμβων (που θα είναι λιγότεροι σε αριθμό) κ.ο.κ.
- Παράδειγμα: άθροιση στοιχείων ενός πίνακα.

Ισοζυγισμένα Δένδρα (2/2)

- π.χ. α1,α2,α3,α4,α5,α6,α7,α8.
- Υπολογίζουμε ενδιάμεσους κόμβους (α1,α2) (α3,α4) (α5,α6),(α7,α8).
- Από αυτούς τους ενδιάμεσους κόμβους υπολογίζουμε τους επόμενους ενδιάμεσους κόμβους β1,β2.
- Από αυτούς τους κόμβους υπολογίζουμε το τελικό αποτέλεσμα.
- Υπολογίζεται λοιπόν αντί για χρόνο **T(n)=O(n)** σε χρόνο **T(n)=O(logn).**

Η τεχνική του διπλασιασμού των δεικτών (1/8)

- Χρήσιμη τεχνική σχεδιασμού.
- Χρησιμοποιείται σε συνδεσμικές λίστες ή σε δενδρικές δομές.
- Έστω μια λίστα n κόμβων και ζητείται η ετικέτα του τελευταίου κόμβου στη λίστα. Σε κάθε βήμα του υπολογισμού, κάθε κόμβος αντικαθιστά το δείκτη του με τον δείκτη του επόμενου κόμβου στη λίστα (ή του πατέρα του).
- Μετά από logn βήματα κάθε κόμβος της λίστας ή του δένδρου “δείχνει” στον ίδιο κόμβο, δηλαδή στον τελευταίο κόμβο (ή στη ρίζα).

Η τεχνική του διπλασιασμού των δεικτών (2/8)

- Παράδειγμα: Έστω το πρόβλημα του υπολογισμού της τάξης (rank) κάθε κόμβου k μιας συνδεσμικής λίστας L (list ranking).
- Έστω L η λίστα. Για κάθε κόμβο i, 1<=i<=n το στοιχείο NEXT[i] περιέχει ένα δείκτη στον κόμβο, ο οποίος ακολουθεί τον κόμβο i.
- Ένας ακολουθιακός αλγόριθμος θα απαιτούσε γραμμικό χρόνο (T(n)=O(n)).
- Ο χρόνος που απαιτεί είναι T(n)=O(logn).

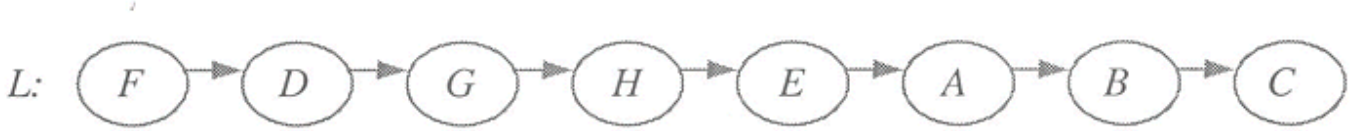
Η τεχνική του διπλασιασμού των δεικτών (3/8)

```
begin
    1. for 1≤i≤n pardo
           M[i] := NEXT[i]
           if ( M[i] = 0 ) then DIST[i] := 0
                           else DIST[i] := 1
       2. for 1≤h≤log n do
              for 1≤i≤n pardo
                  if ( M[i] != 0 ) then
                      begin
                          DIST[i] := DIST[i] + DIST[M[i]]
                          M[i] := M[M[i]]
                      end
end
```

Η τεχνική του διπλασιασμού των δεικτών (4/8)

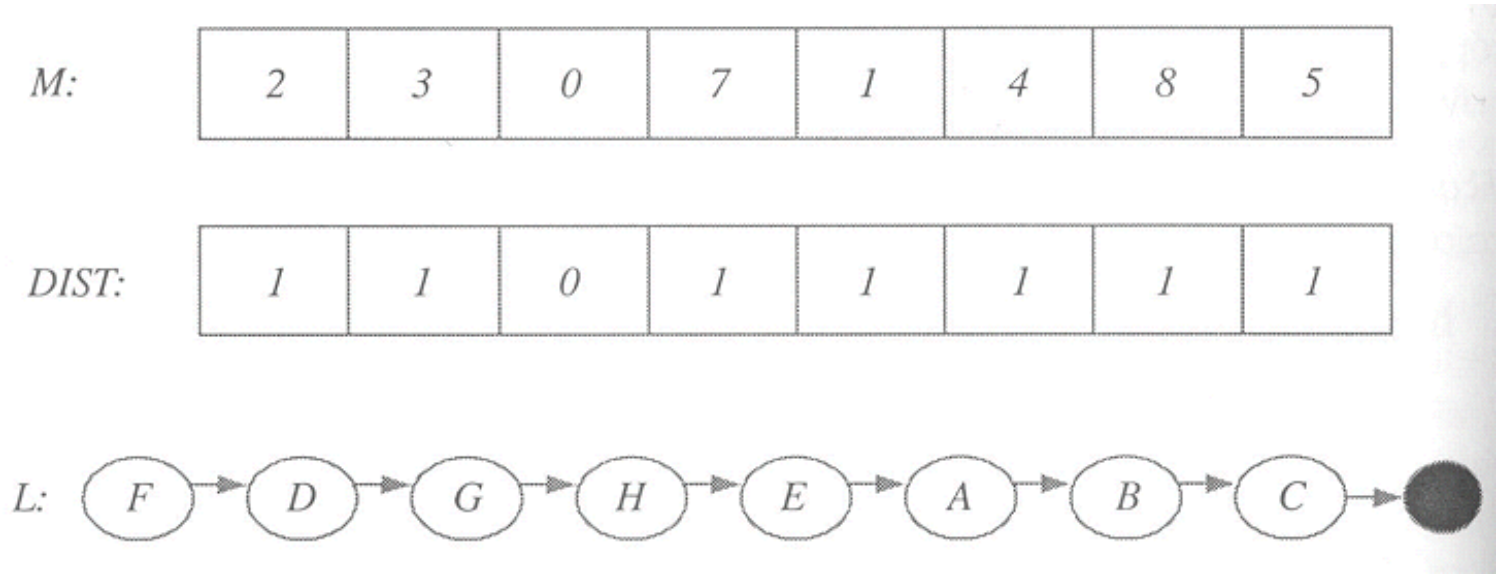
- Έστω ότι έχουμε τη λίστα L.
- Δηλαδή, το A έχει επόμενο αυτό που είναι στη 2 θέση, δηλαδή το B, κ.ο.κ.

	A	B	C	D	E	F	G	H
NEXT:	2	3	0	7	1	4	8	5



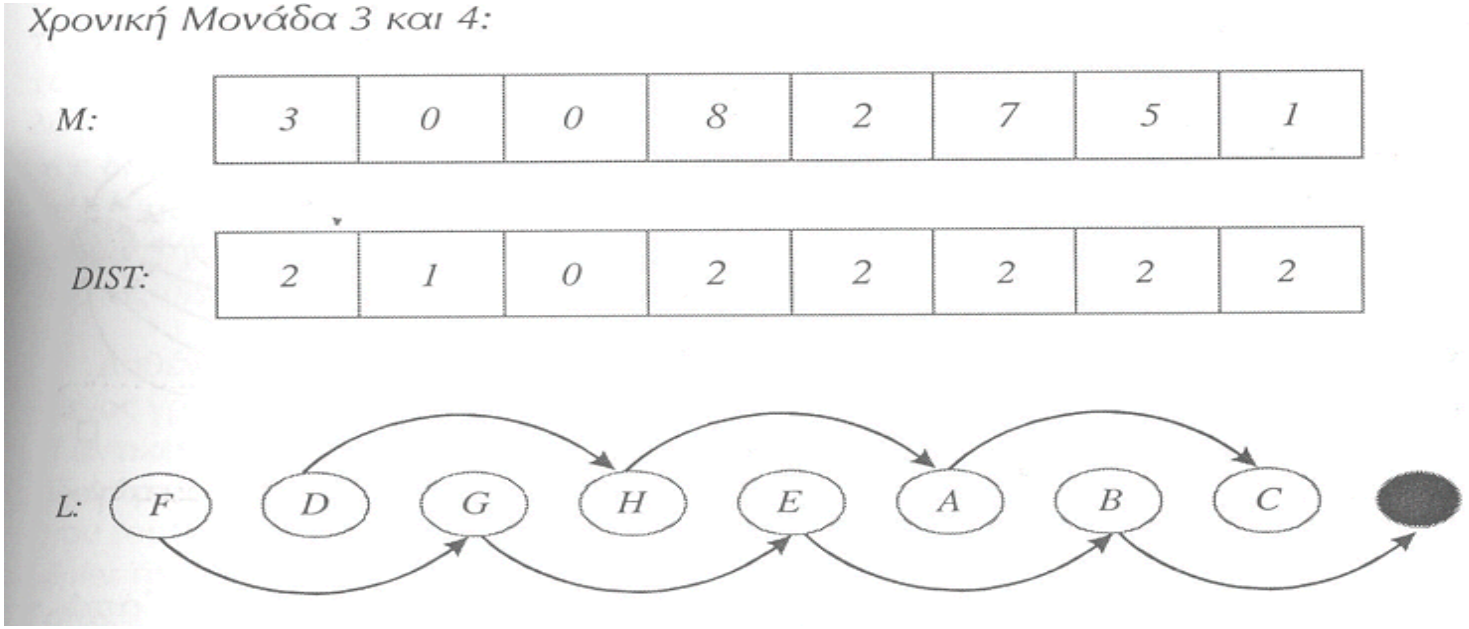
Η τεχνική του διπλασιασμού των δεικτών (5/8)

- Κατά το βήμα 1 ενημερώνονται τα διανύσματα M και DIST.
- Μετά το τέλος λοιπόν του βήματος 1 έχουμε τα παρακάτω:



Η τεχνική του διπλασιασμού των δεικτών (6/8)

- Στο βήμα 2 ενημερώνονται τα M και DIST κατάλληλα.
- Μετά το τέλος της χρονικής μονάδας 8 κάθε κόμβος γνωρίζει την απόσταση DIST.

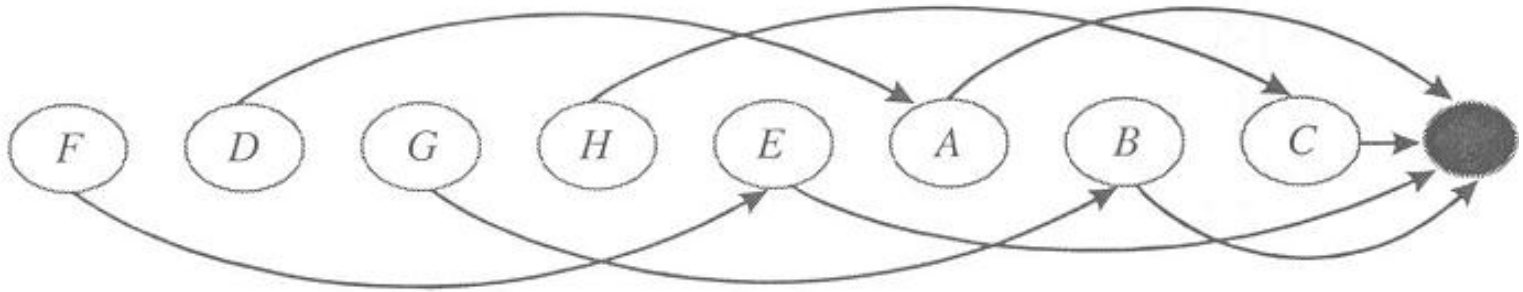


Η τεχνική του διπλασιασμού των δεικτών (7/8)

Χρονική Μονάδα 5 και 6:

M:	0	0	0	1	0	5	2	3

DIST:	2	1	0	4	3	4	4	4

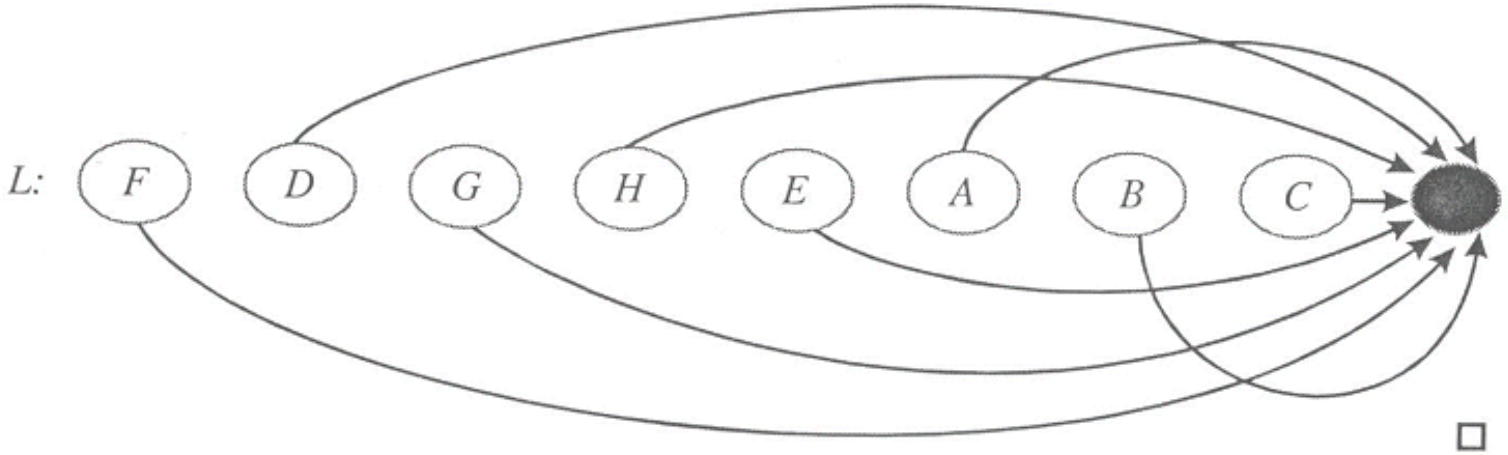


Η τεχνική του διπλασιασμού των δεικτών (8/8)

Χρονική Μονάδα 7 και 8:

M:	0	0	0	0	0	0	0	0

DIST:	2	1	0	6	3	7	5	4



□

Η τεχνική της διαδρομής του Euler (1/8)

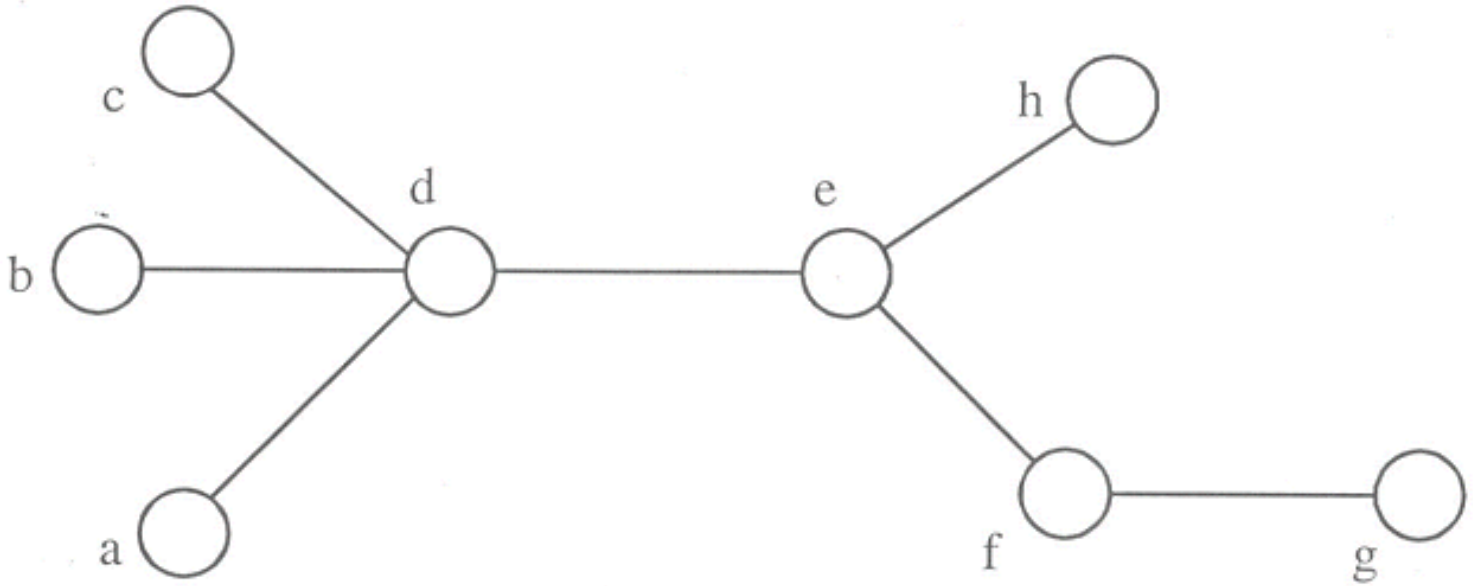
- Ένα πρόβλημα είναι η κατασκευή ενός εντεινόμενου δένδρου (spanning tree) και ο υπολογισμός απλών συναρτήσεων σε αυτό το δένδρο, όπως της "προδιαταγμένης και μεταδιαταγμένης αρίθμησης (preorder/postorder numbering)".
- Δοθέντος ενός δένδρου χωρίς ρίζα T=(V,E) ο κατευθυνόμενος γράφος T'=(V,E'), που προκύπτει αν κάθε ακμή {u,v} αντικατασταθεί από δύο κατευθυνόμενες ακμές (u,v) και (v,u) με την ιδιότητα: ο εσωτερικός βαθμός (indegree) κάθε κόμβου ισούται με τον έξω βαθμό (outdegree).

Η τεχνική της διαδρομής του Euler (2/8)

- Η προηγούμενη ιδιότητα συνεπάγεται ότι ο γράφος είναι Euler δηλαδή, υπάρχει κάποιο κατευθυνόμενο κύκλωμα (directed circuit) που:
  - Περνάει από όλους τους κόμβους Τ‘.
  - Περνάει από κάθε ακμή ακριβώς μια φορά.
  - Καταλήγει στον κόμβο από τον οποίο ξεκίνησε.

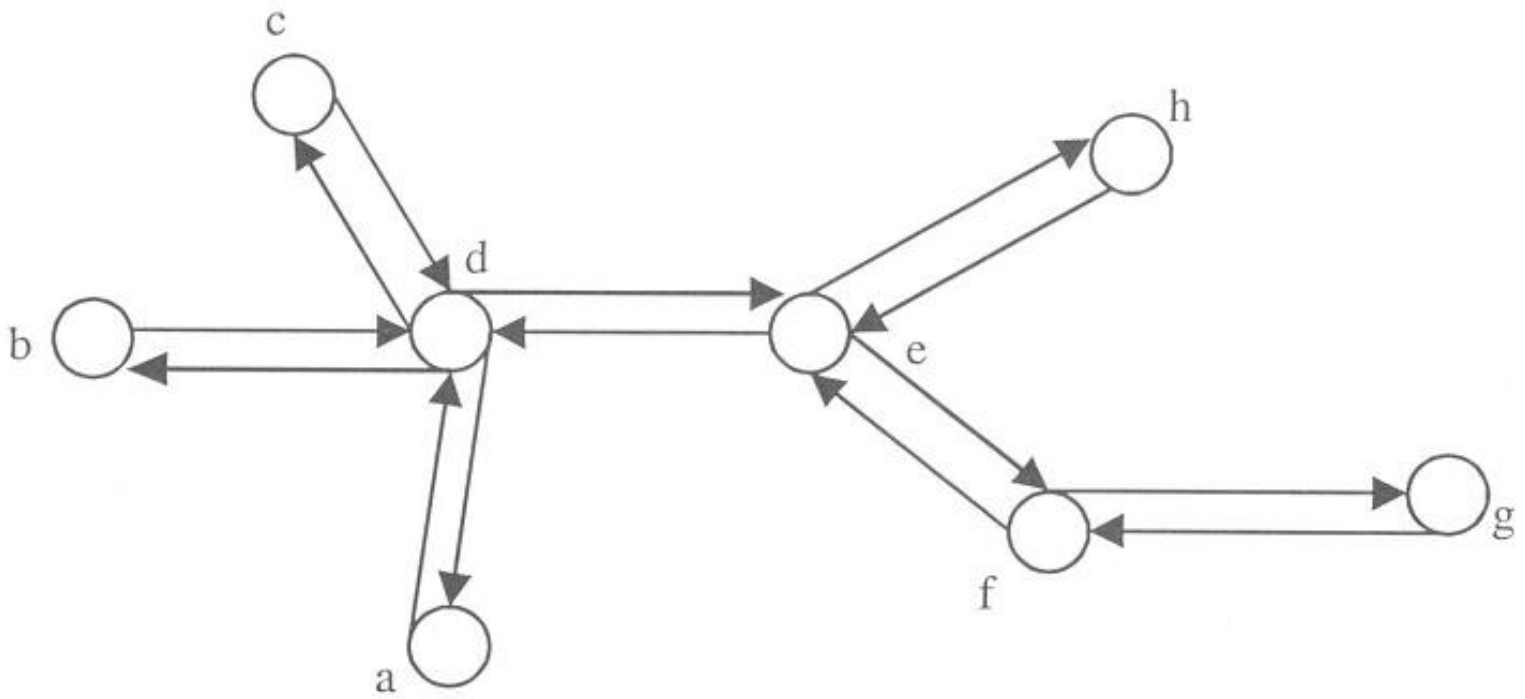
Η τεχνική της διαδρομής του Euler (3/8)

- Έστω το δένδρο T=(V,E)



Η τεχνική της διαδρομής του Euler (4/8)

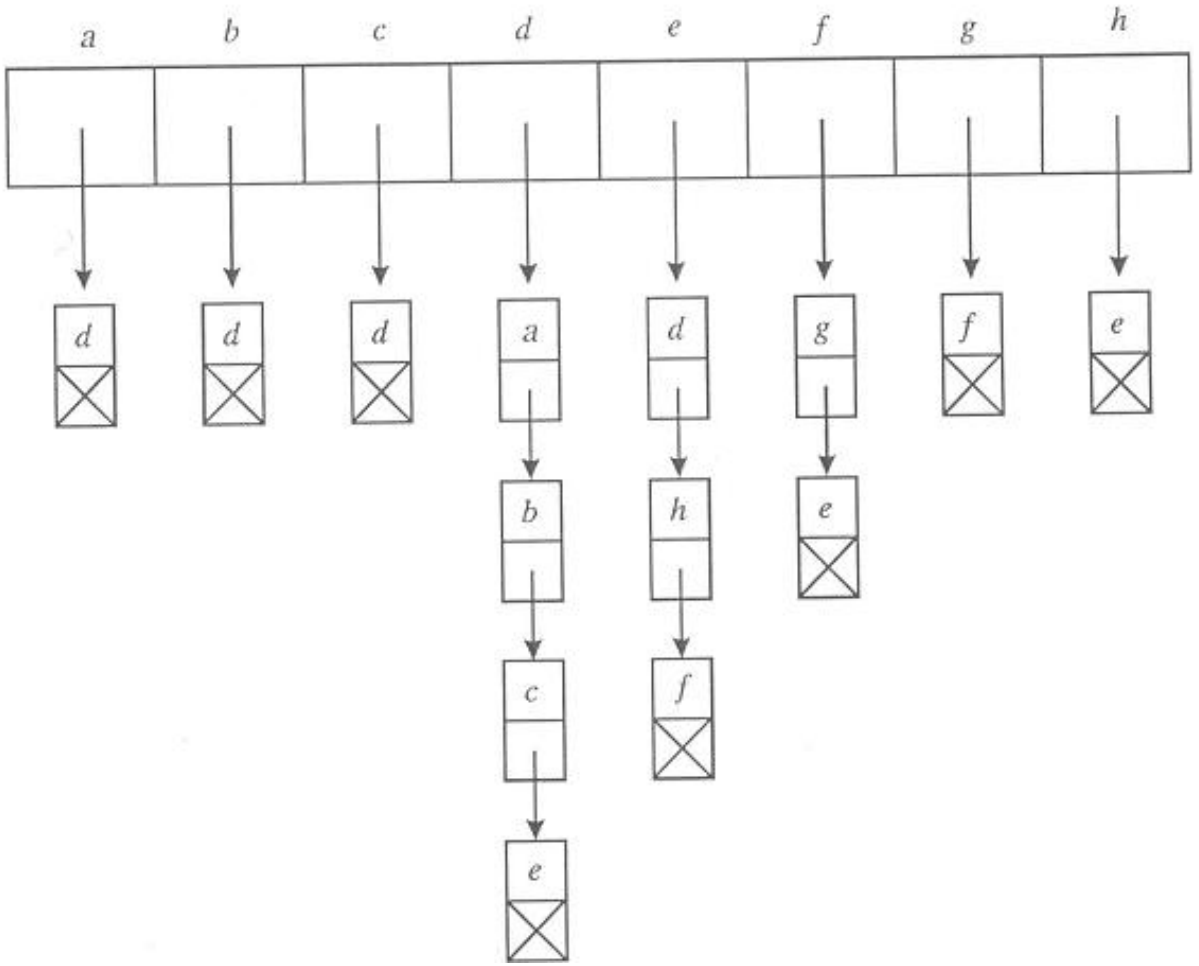
- Αντικαθιστούμε κάθε ακμή με δύο κατευθυνόμενες ακμές.



Η τεχνική της διαδρομής του Euler (5/8)

- Προσδιορίζουμε την επόμενη ή διάδοχο:
  - Για κάθε κόμβο v καθορίζουμε μια συγκεκριμένη διάταξη που πρόσκεινται στο v.
  - Για να καθοριστεί η διάταξη, χρησιμοποιούμε τις λίστες γειτνίασης των κόμβων..
  - Αν d είναι ο βαθμός ενός κόμβου u και v0,v1,...είναι οι γείτονες με τη σειρά που εμφανίζονται στη λίστα γειτνασης u τότε η επόμενη ακμή στο κύκλωμα του Euler είναι η ακμή (u,v(i+1)modd) , i=0,1,...,d-1.

Η τεχνική της διαδρομής του Euler (6/8)



Η τεχνική της διαδρομής του Euler (7/8)

Παράδειγμα 4.3 Έστω ότι, δίνεται το δένδρο $T=(V,E)$ του Σχήματος 4.2. Ο κατευθυνόμενος γράφος $T'=(V,E')$, ο οποίος προκύπτει από το δένδρο T, φαίνεται στο ίδιο σχήμα, ενώ η λίστα γειτνίασης του T (η οποία είναι ίδια με αυτήν του T') δίνεται στο Σχήμα 4.3. Θα υπολογίσουμε τις επόμενες ακμές όλων των ακμών, που πρόσκεινται στον κόμβο e. Από τη λίστα γειτνίασης προκύπτει ότι, οι γείτονες κόμβοι του e είναι οι κόμβοι d, h και f με αυτήν τη συγκεκριμένη σειρά. Επομένως, έχουμε $u=e$, $v_0=d$, $v_1=h$ και $v_2=f$. Η επόμενη ακμή της (d,e) είναι η (e,h), και η επόμενη ακμή της (h,e) είναι η (e,f). Συνεχίζοντας κατά τον ίδιο τρόπο και για τους άλλους κόμβους, βρίσκουμε ότι, οι επόμενες ακμές των ακμών του T' είναι:

Η τεχνική της διαδρομής του Euler (8/8)

Ακμή	Επόμενη Ακμή
(a,d)	(d,b)
(b,d)	(d,c)
(c,d)	(d,e)
(d,a)	(a,d)
(d,b)	(b,d)
(d,c)	(c,d)
(d,e)	(e,h)
(e,f)	(f,g)
(e,d)	(d,a)
(e,h)	(h,e)
(f,g)	(g,f)
(f,e)	(e,d)
(g,f)	(f,e)
(h,e)	(e,f)

Τέλος Ενότητας