



Παράλληλη Επεξεργασία

Κεφάλαιο 6^ο

Σύγχρονος Παραλληλισμός

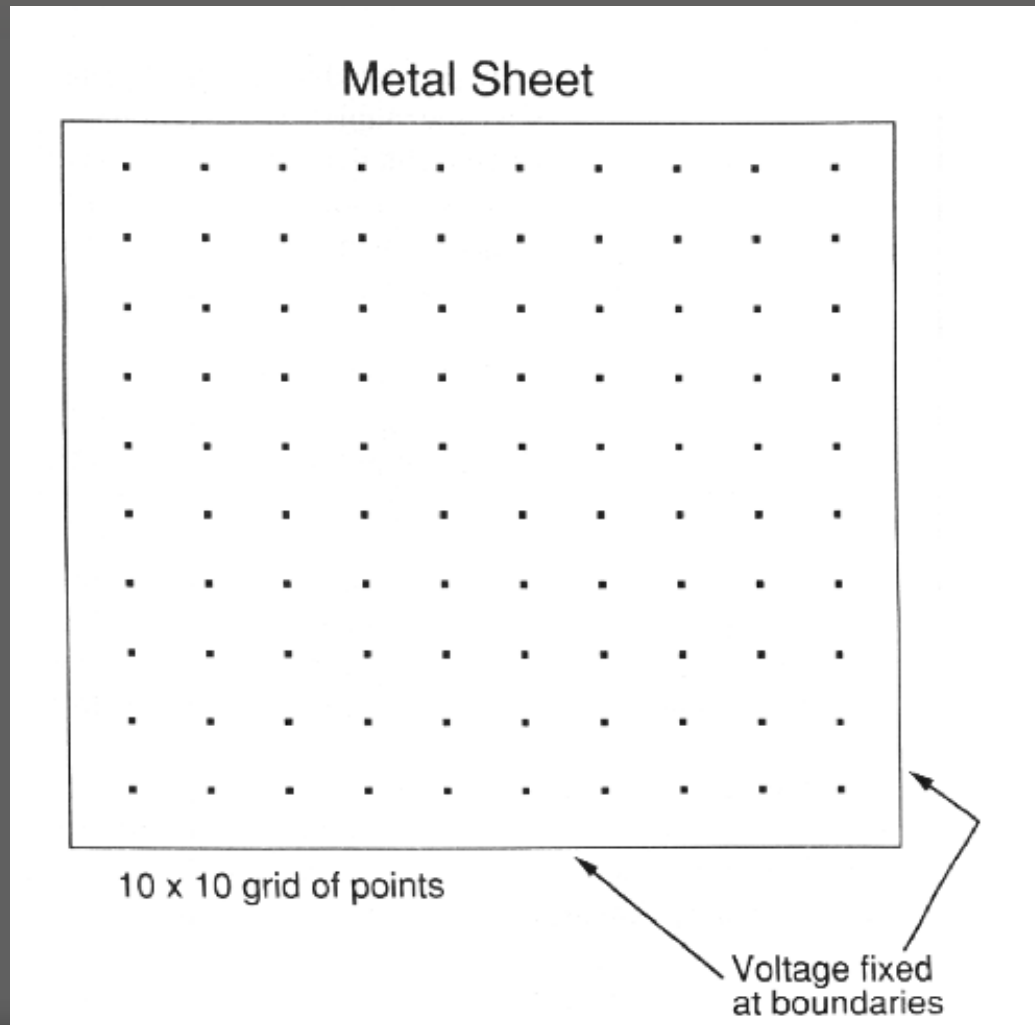
Κωνσταντίνος Μαργαρίτης
Καθηγητής
Τμήμα Εφαρμοσμένης
Πληροφορικής
Πανεπιστήμιο Μακεδονίας
kmarg@uom.gr
<http://eos.uom.gr/~kmarg>

Αρετή Καπτάν
Υποψήφια Διδάκτορας
Τμήμα Εφαρμοσμένης
Πληροφορικής
Πανεπιστήμιο Μακεδονίας
areti@uom.gr
<http://eos.uom.gr/~areti>

Εισαγωγή

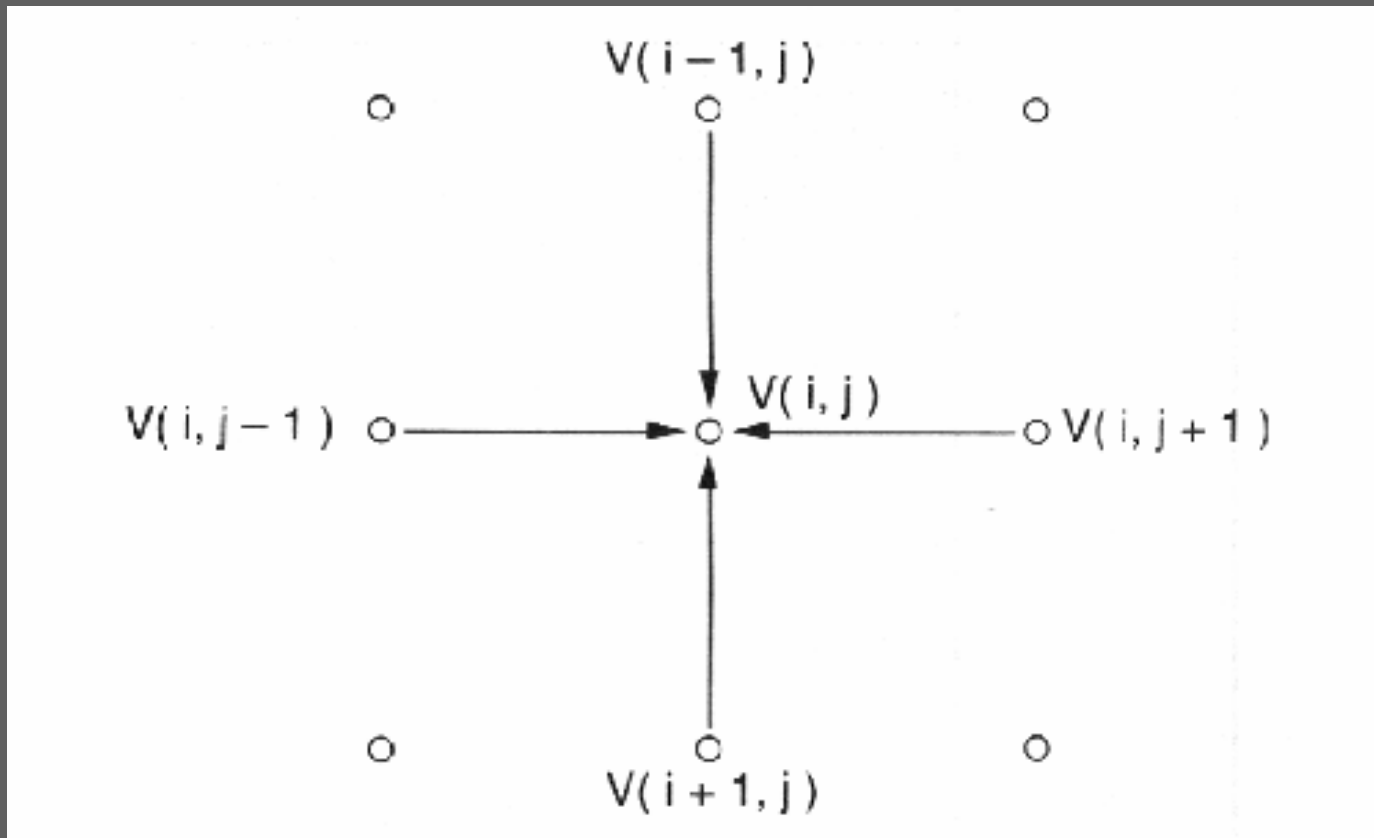
- ⇒ Επαναληπτικοί αριθμητικοί αλγόριθμοι και σύγχρονος παραλληλισμός
- ⇒ Οι διεργασίες συγχρονίζονται στο τέλος κάθε επανάληψης
- ⇒ Τεχνικές παραλληλισμού επαναληπτικών αλγορίθμων:
 - ▣ με τερματισμό διεργασιών (γραμμική καθυστέρηση)
 - ▣ με δημιουργία δυαδικού δέντρου – τεχνική Τουρνουά (λογαριθμική καθυστέρηση)
 - ▣ με τοπικό συγχρονισμό (σταθερή καθυστέρηση ανεξάρτητη από τον αριθμό των διεργασιών)
 - ▣ τεχνική συλλογής και διάδοσης **CAB** με έλεγχο σύγκλισης

Δίκτυο σημείων για την εξίσωση του Laplace



$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 0$$

Υπολογισμός της τάσης κάθε στοιχείου με την μέθοδο Jacobi Relaxation



Ακολουθιακό Πρόγραμμα αλγορίθμου Jacobi Relaxation

```
PROGRAM Jacobi;
CONST n = 32;      (*Μέγεθος του πίνακα *)
      numiter = ...;      (*Πλήθος των επαναλήψεων*)

VAR A, B: ARRAY [0..n+1,0..n+1] of Real;
      i, j, k: Integer;

BEGIN
For i:= 0 to n+1 do      (*Αρχικοποίηση των τιμών του πίνακα*)
  BEGIN
      For j:= 0 to n+1 do
        Read(A[i, j]);
        Readln;
      END;
  B:= A;
  For k:= 1 to numiter do BEGIN
      For i:= 1 to n do
        For j:= 1 to n do      (*Υπολογισμός του μέσου όρου των τεσσάρων γειτονικών σημείων*)
          B[i,j]=(A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1]) / 4;
        END;
      A:= B;
    END;
  END.
```

Jacobi Relaxation – Τεχνική Τερματισμού Διεργασιών

```
PROGRAM ParallelJacobi;
CONST n = 32;
      numiter = ...;

VAR A, B: ARRAY [0..n+1,0..n+1] of Real;
      i, j, k: Integer;

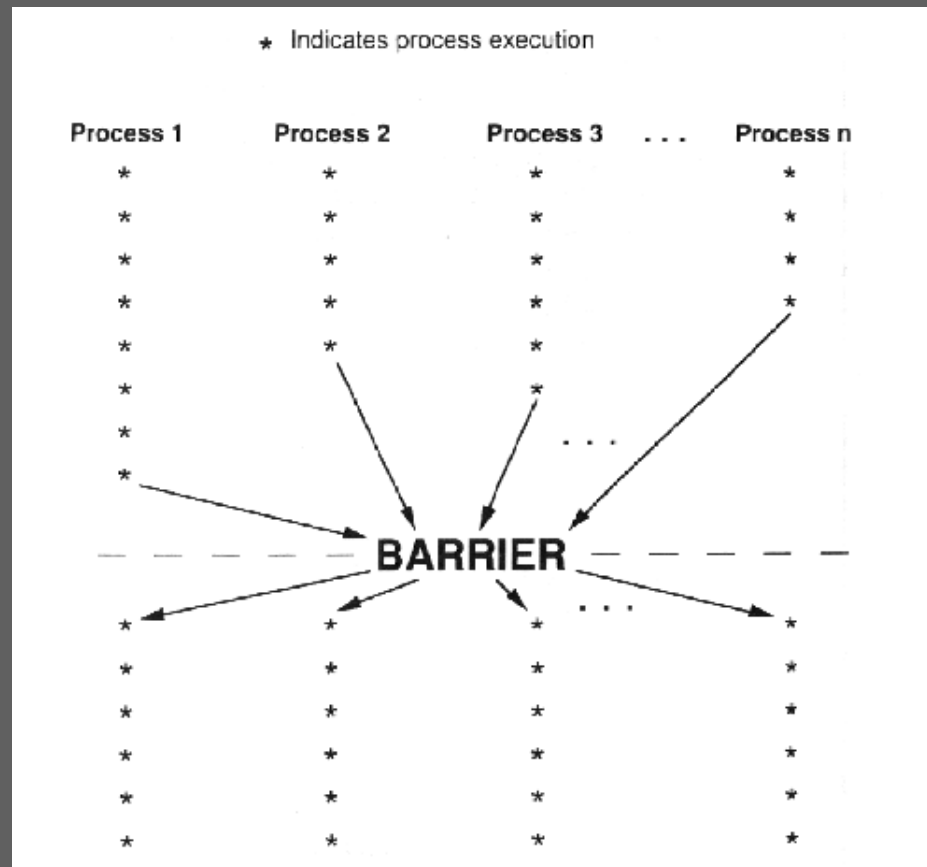
BEGIN
... (*Τοποθέτηση αρχικών τιμών στον πίνακα A*)

B:= A;
For k:= 1 to numiter do
  BEGIN
    (*Φάση I - Υπολογισμός των νέων τιμών*)
    FORALL i:=1 to n do (*Δημιουργία διεργασίας για κάθε γραμμή*)
      VAR
        j: Integer;
      BEGIN
        For j:= 1 to n do
          B[i,j]=(A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1]) / 4;
        END;
      (*Φάση II - Αντιγραφή των νέων τιμών στον A*)
      FORALL i:=1 to n do (*Αντιγραφή των νέων τιμών από τον B στον A*)
        A[i]:= B[i];
      END;
    END.
  END.
```

Εκτίμηση του αλγόριθμου με τερματισμό διεργασιών

- ⇒ 2 φάσεις ανά επανάληψη
- ⇒ Συγχρονισμός ανά φάση
- ⇒ Συγχρονισμός με την μέθοδο δημιουργία και καταστροφής διεργασιών
- ⇒ Επιπλέον κόστος εκτέλεσης:
 - Χρόνος δημιουργίας των διεργασιών
 - Χρόνος καταστροφής των διεργασιών
 - Αριθμός διεργασιών

Συγχρονισμός Φράγματος Διεργασιών



Jacobi Relaxation – Συγχρονισμός με Καθολικό Φράγμα

```
PROGRAM JacobiBarrier;
CONST n = 32;
      numiter = ...;

VAR a, b : Array [0..n+1,0..n+1] of Real;
      i, j : Integer;

BEGIN
...
B := A;
FORALL i := 1 to n do (*Δημιουργία μιας διεργασίας για κάθε γραμμή*)
  VAR j, k : Integer;
  BEGIN
    For k := 1 to numiter do
      BEGIN
        For j := 1 to n do (*Υπολογισμός του μέσου όρου για τις
                           τέσσερις γειτονικές διεργασίες *)
          B[i,j] := (A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1]) / 4;
          Barrier;
          A[i] := B[i];
          Barrier;
        END;
      END;
    END;
  END;
END.
```

Χαρακτηριστικά Φράγματος

- ⇒ Ορισμός φράγματος
- ⇒ Οι διεργασίες συγχρονίζονται 2 φορές σε κάθε επανάληψη
- ⇒ 2 φάσεις:
 - ▣ Υπολογισμός νέων τιμών
 - ▣ Αντικατάσταση παλαιών τιμών
- ⇒ Οι διεργασίες δημιουργούνται μία φορά
- ⇒ Ελαχιστοποίηση χρόνου για δημιουργία διεργασιών
- ⇒ Επιπλέον κόστος λόγω φράγματος
- ⇒ Υλοποίηση φράγματος με:
 - ▣ Κλείδωμα
 - ▣ Κανάλια
 - ▣ Σηματοφορείς

Γραμμικό Φράγμα

- ⇒ Χρήση μεταβλητής αθροιστή
- ⇒ Χρήση μεταβλητών κλειδώματος Spinlock
- ⇒ 2 φάσεις συγχρονισμού:
 - ▣ Άφιξη
 - ▣ Αναχώρηση
- ⇒ Κόστος ανάλογο του πλήθους των παράλληλων διεργασιών

Υλοποίηση Γραμμικού Φράγματος με Κλειδώμα

```
PROGRAM JacobiBarrier;
CONST n = 32;

VAR .....
    count : Integer;
    Arrival, Departure : Spinlock;

PROCEDURE Barrier;
BEGIN

    (*Φάση Άφιξης - Άθροιση των διεργασιών καθώς εισέρχονται*)
    Lock(Arrival);
    count := count + 1;
    If count < n
        then Unlock(Arrival)      (*Συνέχιση της Φάσης Άφιξης*)
        else Unlock(Departure);  (*Τέλος της Φάσης Άφιξης*)

    (*Φάση Αναχώρησης - Άθροιση των διεργασιών καθώς εξέρχονται*)
    Lock(Departure);
    count := count - 1;
    If count > 0
        then Unlock(Departure)   (*Συνέχιση της Φάσης Αναχώρησης*)
        else Unlock(Arrival);    (*Τέλος της Φάσης Αναχώρησης*)
    END;

BEGIN (*Κυρίως Πρόγραμμα*)

    count := 0; (*Αρχικοποίηση της μεταβλητής count και των κλειδωμάτων*)
    Unlock(Arrival);
    Unlock(Departure);

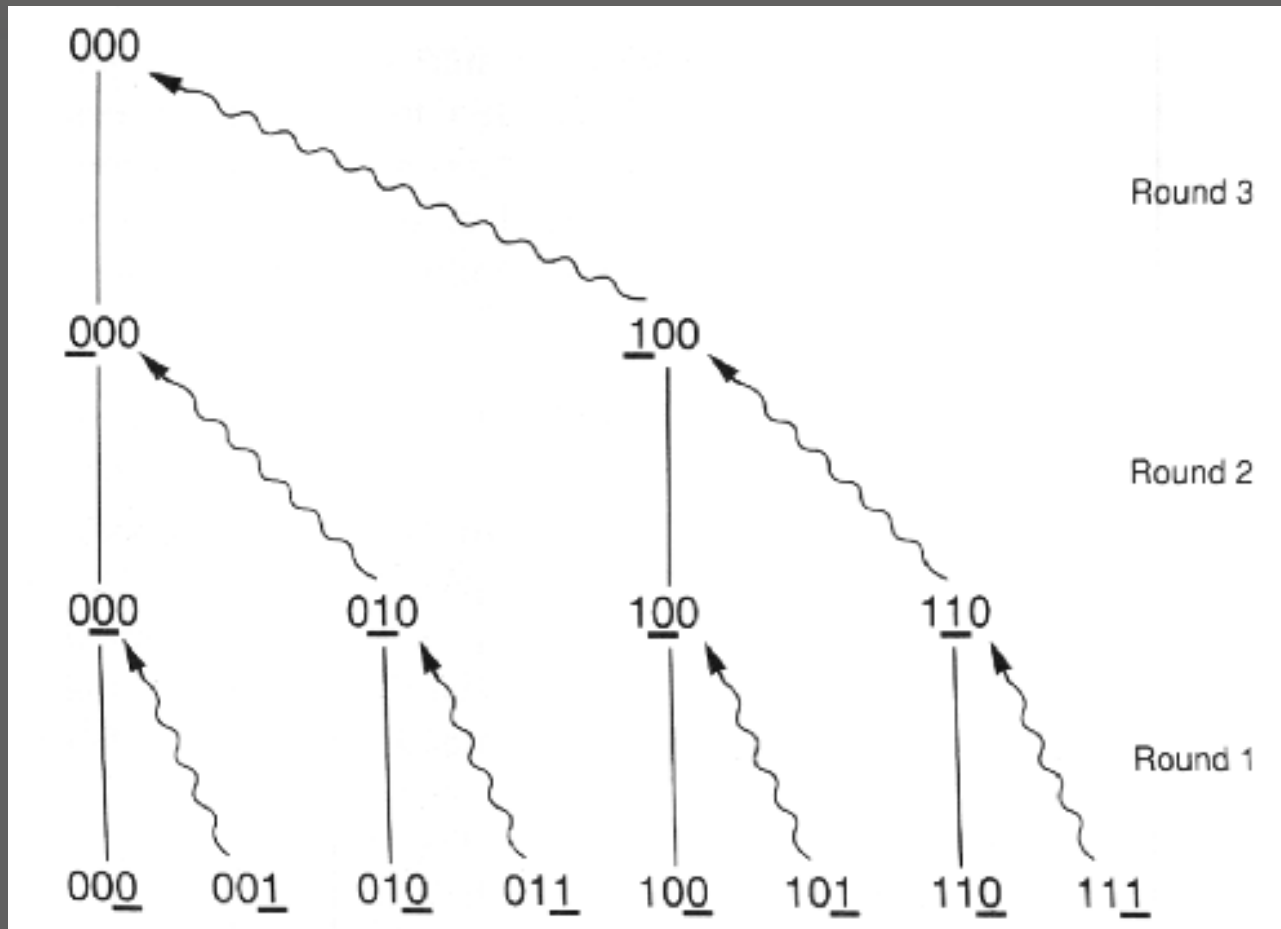
    .....

END.
```

Φράγμα Διαδικού Δέντρου

- ⇒ Αποκέντρωση της τεχνικής συγχρονισμού
- ⇒ Χαρακτηριστικό διαδικού δέντρου:
 - ▣ Συγκεντρωτικό στη ρίζα
 - ▣ Αποκεντρωτικό στα φύλλα
- ⇒ Τεχνική Τουρνουά
- ⇒ Καθυστέρηση ανάλογη του λογαρίθμου του πλήθους των διεργασιών

Συγχρονισμός Διαδικασίας Δέντρου



Αλγόριθμος Δημιουργίας Δέντρου Συγκέντρωση και καθυστέρηση των διεργασιών

```
position := 1;
while (Bit(mynumber, position) = 0) AND (position < n) Do
  BEGIN
    Receive message from losing partner ;
    position := position * 2;
  END;
If mynumber <> 0 then
  BEGIN
    winningPartner := ClearBit(mynumber, position);
    Send message for winningPartner;
    wait for reply message;
  END;
```

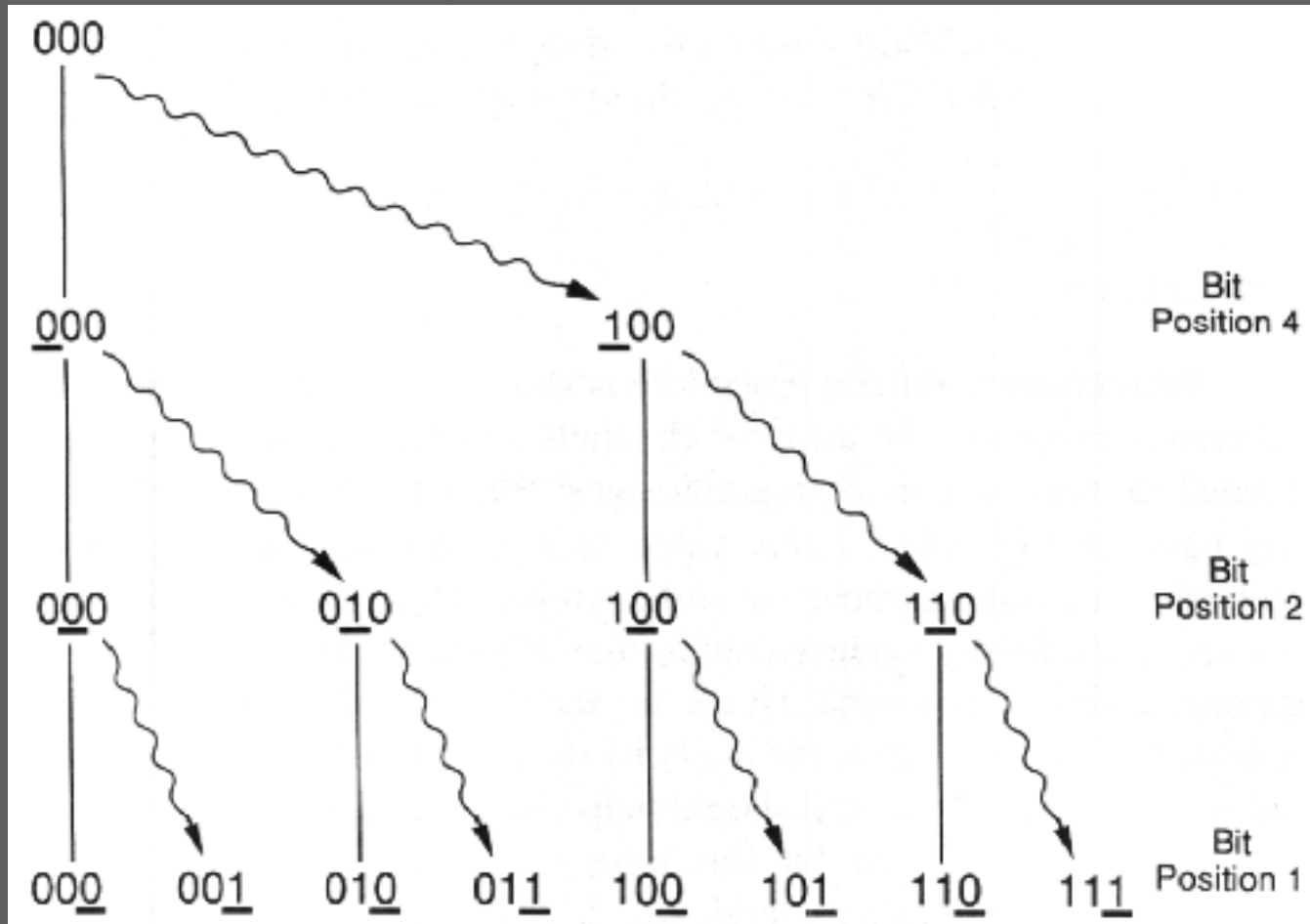
Συναρτήσεις για τη δημιουργία του δυναδικού δέντρου

⇒ `Bit(i,p)`: $i \text{ DIV } p \text{ MOD } 2$

⇒ `ClearBit(i,p)`: $i - p$

⇒ `SetBit(i,p)`: $i + p$

Απελευθέρωση διεργασιών σε αναμονή από το φράγμα



Αλγόριθμος Δημιουργίας Δέντρου Απελευθέρωση εγκλωβισμένων διεργασιών

```
(*Η μεταβλητή position λαμβάνει μια μέγιστη τιμή από τον αλγόριθμο
   που τη δημιουργεί*)
while position > 1 do
  BEGIN
    position := position DIV 2;
    LosingPartner := SetBit(mynumber, position);
    Send message to LosingPartner;
  END;
```

```

PROGRAM JacobiBarrier;
CONST n = 32;

VAR
  ....
  synchan: Array [0..n-1] of Channel of Integer;

Procedure Barrier(me: Integer);
VAR position, dummy: Integer;

BEGIN
  position := 1;
  while (me DIV position MOD 2 = 0) AND (position < n) Do
    BEGIN
      dummy:=synchan[me]; (*Λήψη μηνύματος από τον αντίπαλο που έχει
                           χάσει*)
      position := position * 2;
    END;
  If me <> 0 then
    BEGIN
      synchan[me-position] := 1;      (*Διάδοση μηνύματος στη διεργασία
                                       νικητή*)
      dummy := synchan[me];          (*Αναμονή για απάντηση*)
    END;
  while position > 1 do
    BEGIN
      position := position DIV 2;
      synchan[me+position] := 1;      (*Αποστολή μηνύματος στη διεργασία
                                       που
                                       είχε χάσει προηγουμένως*)
    END;
  END;

BEGIN (*κυρίως Πρόγραμμα*)
  ....

  FORALL I := 1 to n do      (*Δημιουργία μιας διεργασίας για κάθε
                             σειρά του πίνακα*)
    ....
    Barrier(i-1);
    ....
    Barrier(i-1);
    ....
  END.

```

Υλοποίηση φράγματος με δυαδικό δέντρο

Αλγόριθμος με Τοπικό Συγχρονισμό

- ⇒ Εξαρτάται από το μέγεθος της τοπικής γειτονιάς
- ⇒ Ανάθεση ξεχωριστού καναλιού για την επικοινωνία με κάθε διεργασία-γείτονα
- ⇒ Αρχικά αποστολή μηνυμάτων στους γείτονες κι εν συνεχεία λήψη μηνυμάτων, διαφορετικά αδιέξοδο
- ⇒ Jacobi Relaxation:
 - Συνολικά καθολική ροή δεδομένων
 - Σε κάθε επανάληψη τοπική ροή δεδομένων

```
PROGRAM JacobiBarrier;
```

```
CONST n = 32;
```

```
    numiter = ...;
```

```
VAR a, b : Array [0..n+1,0..n+1] of Real;
```

```
    i, j : Integer;
```

```
    higher, lower : Array [1..n] of Channel of Integer;
```

```
Procedure LocalBarrier (I: Integer);
```

```
VAR dummy : Integer;
```

```
BEGIN
```

```
If i > 1 then higher[i-1] := 1; (*Αποστολή στη διεργασία I-1*)
```

```
If i < n then BEGIN
```

```
    lower[i+1] := 1; (*Αποστολή στη διεργασία I+1*)
```

```
    dummy := higher[i]; (*Λήψη από τη διεργασία I-1*)
```

```
END;
```

```
If i > 1 then dummy := lower[i]; (*Λήψη από τη διεργασία I-1*)
```

```
END;
```

```
BEGIN (*Κυρίως πρόγραμμα*)
```

```
...
```

```
B := A;
```

```
FORALL i := 1 to n do (*Δημιουργία μιας διεργασία για κάθε γραμμή*)
```

```
    VAR j, k : Integer;
```

```
    BEGIN
```

```
        For k := 1 to numiter do BEGIN
```

```
            For j := 1 to n do (*Υπολογισμός του μέσου όρου για τους τέσσερις γείτονες*)
```

```
                B[i,j] := (A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1])/4;
```

```
                LocalBarrier(i);
```

```
                A[i] := B[i];
```

```
                LocalBarrier(i);
```

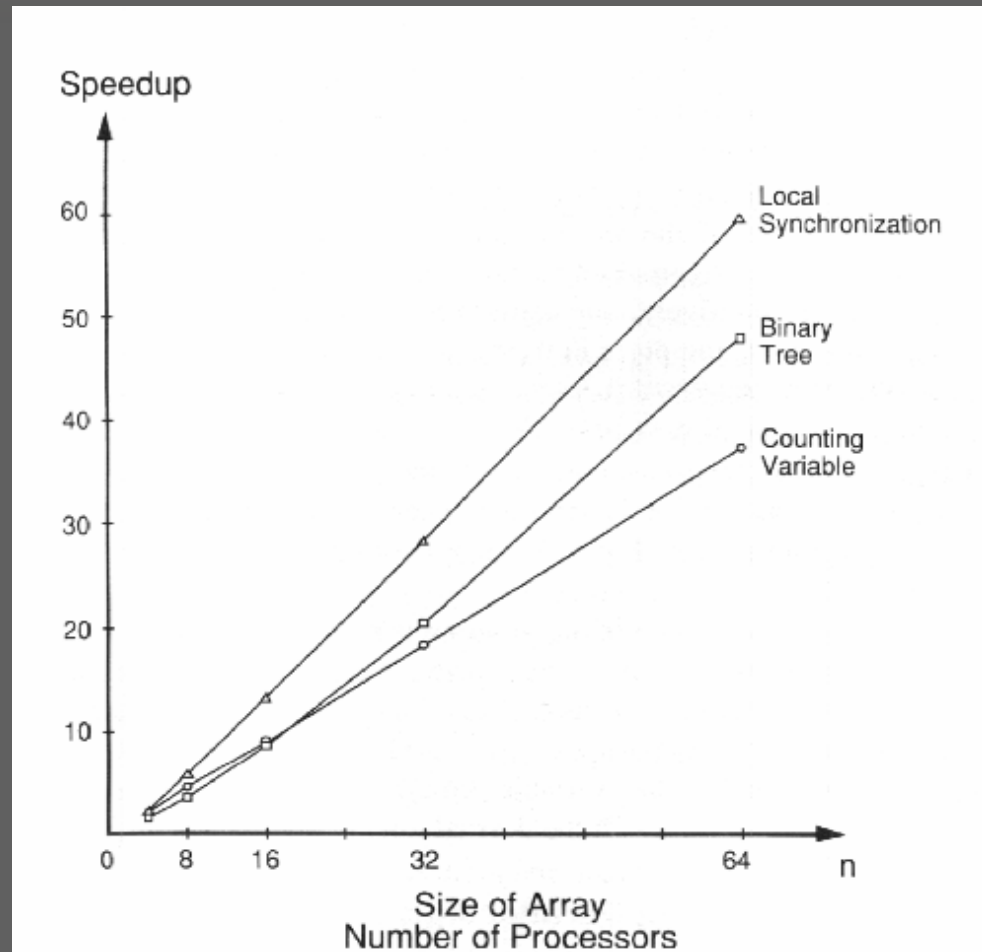
```
            END;
```

```
        END;
```

```
END.
```

Αλγόριθμος Jacobi με Τοπικό Συγχρονισμό

Σχετική Απόδοση των Εφαρμογών Φράγματος



```

PROGRAM Jacobi;
CONST n = 32;
      tolerance = .01;

VAR A, B : Array [0..n+1,0..n+1] of Real;
      i,j : Integer;
      change, maxchange : Real;

BEGIN

.... (*Ανάγνωση αρχικών τιμών για τον πίνακα A*)

B := A;
REPEAT (*Υπολογισμός των νέων τιμών μέχρι να προσεγγιστεί η
επιθυμητή ανοχή*)
  maxchange := 0;
  For i := 1 to n do
    For j := 1 to n do Begin (*Υπολογισμός των νέων τιμών και της αλλαγής τους
από τις παλιές*)
      B[i,j] := (A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1]) / 4;
      change := ABS(B[i,j] - A[i,j]);
      If change > maxchange then
        maxchange := change;
    END;
  A := B;
Until maxchange < tolerance;
END.

```

Ακολουθιακός αλγόριθμος Jacobi με Έλεγχο Σύγκλισης

```
PROGRAM JacobiConv;
```

```
VAR ...
```

```
count : Integer;  
Arrival, Departure : Spinlock;  
globaldone : Boolean;
```

```
Function Aggregate (mydone : Boolean) : Boolean;
```

```
BEGIN
```

```
(*Φάση Άφιξης - Άθροιση των διεργασιών καθώς εισέρχονται στο φράγμα*)  
Lock(Arrival);
```

```
count := count+1;  
globaldone := globaldone AND mydone; (*Βήμα σύγκλισης*)
```

```
If count < n
```

```
then Unlock(Arrival) (*Συνέχιση της φάσης Άφιξης*)  
else Unlock(Departure); (*Τερματισμός της φάσης Άφιξης*)
```

```
(*Φάση Αναχώρησης - Άθροιση των διεργασιών καθώς εξέρχονται  
από το φράγμα*)
```

```
Lock(Departure);
```

```
count := count - 1;
```

```
Aggregate:=globaldone; (*Επιστροφή της σημαίας flag στη διεργασία*)
```

```
If count > 0 then Unlock(Departure) (*Συνέχιση της φάσης Αναχώρησης*)
```

```
else BEGIN
```

```
globaldone := TRUE; (*Αλλαγή της τιμής της μεταβλητής για την επόμενη επανάληψη*)  
Unlock(Arrival); (*Τερματισμός της φάσης Αναχώρησης*)
```

```
END;
```

```
END;
```

```
BEGIN (*Κυρίως Πρόγραμμα*)
```

```
count := 0; (*Αρχικοποίηση της μεταβλητής count και των κλειδωμάτων*)
```

```
Unlock(Arrival);
```

```
Lock(Departure);
```

```
globaldone := TRUE; (*Αρχικοποίηση της τοπικής σημαίας*)
```

```
...
```

Φράγμα Συλλογής και Διάδοσης με Κλείδωμα


```

PROGRAM JacobiConv;
CONST tolernce = .01;
      n = 32;
VAR  A, B : Array {0..n+1,0..n+1} of Real;
      i,j : integer;

Procedure Barrier (me : Integer);
... (*Ιδια με πριν*)

Function Aggregate (mydone : Boolean) : Boolean;
... (*Δες το σχήμα 6.14*)

BEGIN (*Κυρίως πρόγραμμα*)
...

B := A;
FORALL i := 1 to n do (*Δημιουργία των διεργασιών*)
VAR j: Integer;
      change, maxchange : Real;
      done : Boolean;
BEGIN
  Repeat
    maxchange := 0;
    For j := 1 to n do BEGIN (*Υπολογισμός νέων τιμών για κάθε στοιχείο της γραμμής*)
      B[i,j] := (A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1])/4;
      change := ABS(B[i,j]-A[i,j]);
      If change > maxchange then maxchange := change;
    END;
    Barrier;
    A[i] := B[i];
    done := Aggregate(maxchange<tolerance);
  Until done; (*Επανάληψη μέχρι τον καθορισμό του τοπικού τερματισμού*)
END;
END.

```

Παράλληλο Πρόγραμμα Jacobi με Έλεγχο Σύγκλισης