Παράλληλοι Υπολογισμοί

Τμήμα Πληροφορικής και Τηλεπικοινωνιών,
ΕΚΠΑ

30 Μαρτίου 2009

Περιεχόμενα

- Βασικές έννοιες MPI
- Συναρτήσεις MPI
  - Συναρτήσεις διαχείρισης περιβάλλοντος
  - Συναρτήσεις σημείου προς σημείο επικοινωνίας
  - Συναρτήσεις συλλογικής επικοινωνίας
  - Τοπολογίες
- Παράλληλες επαναληπτικές Μέθοδοι
  - Η μέθοδος R/B SOR
  - Η μέθοδος R/B/G/O SOR
  - Αριθμητικά αποτελέσματα μεθόδου R/B SOR.
- Συναρτήσεις υλοποίησης μεθόδων με χρήση οριζόντιας διαμέρισης.
  - Συναρτήσεις διαχωρισμού σε strips και blocks
  - Συναρτήσεις επικοινωνίας
  - Συναρτήσεις υπολογισμού των κόμβων του πλέγματος
    - Η μέθοδος Jacobi
  - Συνάρτηση υπολογισμού κριτηρίου διακοπής.

Βασικές έννοιες MPI

Διεργασία (process)

Μια δραστηριότητα (activity), μια οντότητα (entity) ή ένα αφηρημένο αντικείμενο (abstract object), που καταλαμβάνει ή απασχολεί τους πόρους (resources), της μηχανής.

MPI

Αποτελεί μια βιβλιοθήκη. Μια διεργασία MPI αποτελείται από ένα πρόγραμμα σε C το οποίο επικοινωνεί με άλλες διεργασίες χρησιμοποιώντας MPI συναρτήσεις.

Χειριστής (handle)

Το MPI υποστηρίζει εσωτερικές δομές δεδομένων που έχουν σχέση με τις επικοινωνίες και καθορίζονται από τον χρήστη μέσω των χειριστών.
Οι χειριστές επιστρέφουν στον χρήστη από κάποιες MPI κλήσεις και μπορούν να χρησιμοποιηθούν από άλλες MPI κλήσεις.

Βασικές έννοιες MPI

Μεταδότης (communicator)

Ο Μεταδότης είναι ένας χειριστής (handle), ο οποίος αντιπροσωπεύει μία ομάδα (group) διεργασιών που μπορούν να επικοινωνούν μεταξύ τους και οι οποίες δεν είναι απαραίτητο να ανήκουν όλες στο ίδιο πρόγραμμα MPI. Αρκεί να ανήκουν όλες στον ίδιο Μεταδότη.

Βασικός Μεταδότης είναι ο *MPI_COMM_WORLD* . Ο μεταδότης αυτός δημιουργείται αυτόματα από το MPI και περιέχει όλες τις διεργασίες που δημιουργούνται όταν αρχίζει η εκτέλεση του προγράμματος.

Μπορούμε να ορίσουμε επιπλέον Μεταδότες, οι οποίοι θα αποτελούνται από υποσύνολα των διαθέσιμων διεργασιών, προκειμένου να μπορούμε να οργανώσουμε τις διεργασίες σε διαφορετικές τοπολογίες.

Παρατήρηση: Εκτός από τον Μεταδότη *MPI_COMM_WORLD* , μπορούμε να δημιουργήσουμε και έναν δεύτερο, ο οποίος θα περιέχει τις διεργασίες οργανωμένες σύμφωνα με τη δομή ενός πίνακα δύο διαστάσεων. Για παράδειγμα, όταν έχουμε καρτεσιανή τοπολογία.

Βασικές έννοιες MPI

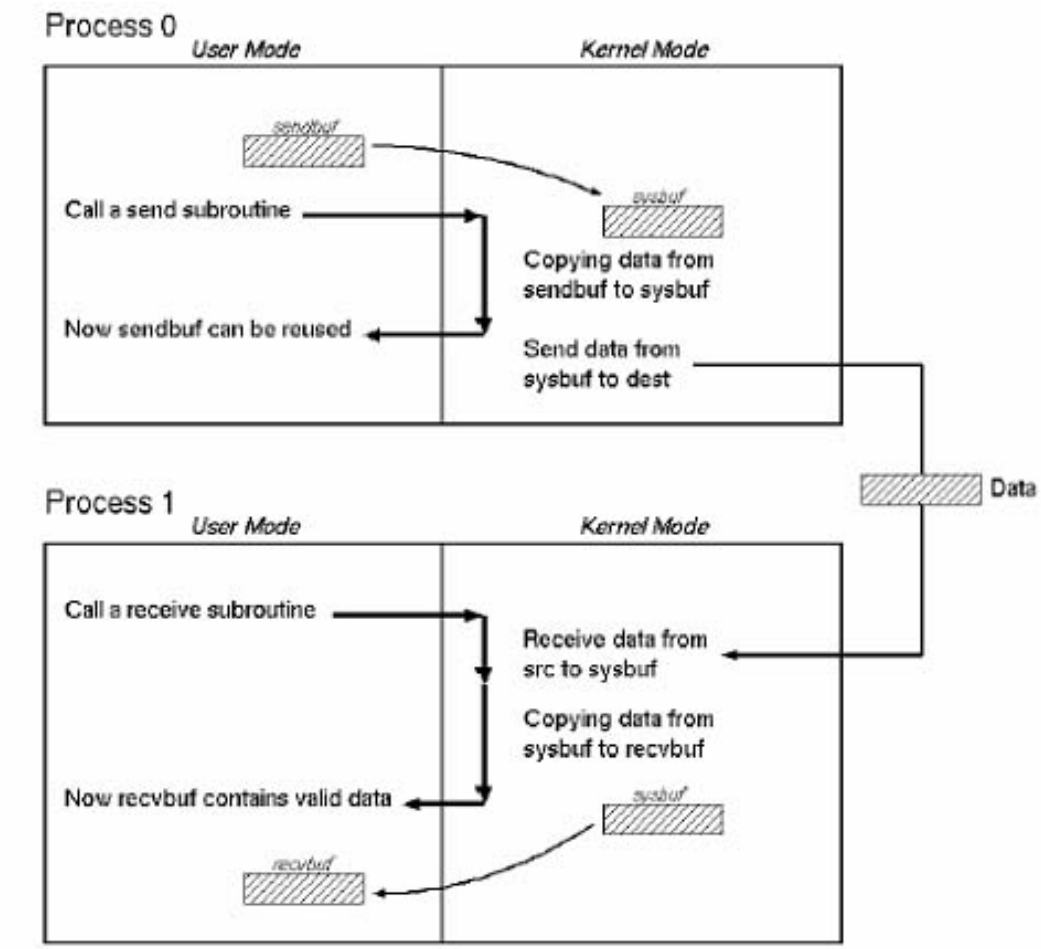
- Σημείο προς σημείο επικοινωνία (Point to Point Communication): Ένα μήνυμα αποστέλλεται από την διεργασία αποστολέα στην διεργασία παραλήπτη. Στην διεργασία αποστολέα, συμβαίνουν τα ακόλουθα γεγονότα το ένα μετά το άλλο.
  - Τα δεδομένα αντιγράφονται στον αποθηκευτικό χώρο του χρήστη (user buffer)από τον χρήστη
  - Ο χρήστης καλεί μια από τις συναρτήσεις αποστολής του MPI
  - Το σύστημα αντιγράφει τα δεδομένα από τον αποθηκευτικό χώρο του χρήστη στον αποθηκευτικό χώρο του συστήματος (system buffer)
  - Το σύστημα στέλνει τα δεδομένα από τον αποθηκευτικό χώρο του συστήματος στην διεργασία προορισμού.

... Βασικές έννοιες MPI ...

Ο όρος αποθηκευτικός χώρος του χρήστη εννοεί μεγέθη μεταβλητών ή πινάκων που χρησιμοποιούνται από το σύστημα.
Στην διεργασία παραλήπτη, συμβαίνουν τα ακόλουθα γεγονότα το ένα μετά το άλλο.

- Ο χρήστης καλεί μια από τις συναρτήσεις λήψης του MPI
- Το σύστημα λαμβάνει τα δεδομένα από την πηγαία διεργασία και τα αντιγράφει στον αποθηκευτικό χώρο του συστήματος
- Το σύστημα αντιγράφει τα δεδομένα από τον αποθηκευτικό χώρο του συστήματος στον αποθηκευτικό χώρο του χρήστη.
- Ο χρήστης χρησιμοποιεί τα δεδομένα που βρίσκονται στον αποθηκευτικό χώρο του χρήστη.

... Βασικές έννοιες MPI....



Σχήμα: Κίνηση δεδομένων σε μια σημείο προς σημείο επικοινωνία

Βασικές έννοιες MPI

Αναστέλλουσα ή εμποδιστική επικοινωνία (Blocking Communication)

Είναι οι συναρτήσεις οι οποίες αναστέλλουν την εκτέλεση της διεργασίας που τις καλεί. Αυτό σημαίνει ότι μια διεργασία που την καλεί θα περιμένει μέχρι να ληφθεί κάποιο μήνυμα.Τέτοιες συναρτήσεις είναι οι *MPI_Send*, *MPI_Recv*.

Μη Αναστέλλουσα ή μη εμποδιστική επικοινωνία (Non Blocking Communication)

Είναι οι συναρτήσεις οι οποίες επιτρέπουν στην καλούσα διεργασία να συνεχίσει την εκτέλεσή της.Τέτοιες συναρτήσεις είναι οι *MPI_ISend*, *MPI_IRecv*.

Συλλογική επικοινωνία (Collective Communication)

Η κλήση αφορά όλες τις διεργασίες μιας ομάδας.Για παράδειγμα, αποστολή του ίδιου μηνύματος σε όλες τις διεργασίες μιας ομάδας (Broadcast).

...Συναρτήσεις MPI...

- Συναρτήσεις Διαχείρησης περιβάλλοντος :
  - *MPI_Init* και *MPI_Finalize*. Η σύνταξη τους είναι η εξής:

$$int\ MPI\_Init(int\ ^{*}argc,\ char\ ^{**}argv);$$

Για την αρχικοποίηση του *MPI*

argc: Ο αριθμός των παραμέτρων που δίνονται στη γραμμή εντολών όταν εκτελείται το πρόγραμμα.
argv: Πίνακας χαρακτήρων ο οποίος περιέχει αυτές τις παραμέτρους.

$$int\ MPI\_Finalize(void);$$

Για τον τερματισμό του *MPI*.

...Συναρτήσεις MPI...

MPI_Comm_size.

Προσδιορισμός πλήθους διεργασιών που υπάρχουν σε ένα μεταδότη.

$$int\ MPI\_Comm\_size\ (int\ MPI\_Comm\ comm,\ int\ ^{*}size);$$

size: Αριθμός διεργασιών που περιέχονται στον Μεταδότη
$comm = MPI\_COMM\_WORLD$.

...Συναρτήσεις MPI...

MPI_Comm_rank.

Προσδιορισμός της τάξης μιας διεργασίας σε ένα Μεταδότη.

$$int\ MPI\_Comm\_rank\ (MPI\_Comm\ comm,\ int\ ^{*}rank);$$

$$comm = MPI\_COMM\_WORLD$$

Ο Μεταδότης στον οποίο ανήκουν όλες οι διεργασίες.
rank: Στην παράμετρο αυτή επιστρέφεται η τάξη της διεργασίας η οποία κάνει την κλήση, δηλ., δηλώνει τον αριθμό που αντιστοιχεί σε κάθε διεργασία.

...Συναρτήσεις MPI....

MPI_Get_processor_name

Δήλωση ονόματος επεξεργαστή στον οποίο εκτελείτε μια διεργασία.

$$int\ MPI\_Get\_processor\_name\ (char\ processor\_name,\ int\ ^{*}namelen);$$

processor_name : Η ενδιάμεση μνήμη στην οποία επιστρέφεται ένα string με το όνομα του επεξεργαστή. Πρέπει να έχει μέγεθος τουλάχιστον, όσο είναι η τιμή της σταθεράς *MPI_MAX_PROCESSOR_NAME*.
namelen : Το μήκος του string σε χαρακτήρες.

...Συναρτήσεις MPI....

MPI_Comm_free

Απελευθερώνει τους πόρους του συστήματος που σχετίζονται με τον μεταδότη comm .

$$int\ MPI\_Comm\_free\ (MPI\_Comm\ ^{*}comm);$$

MPI_Wtime().

Αποτελεί κλήση συστήματος για τον υπολογισμό της ώρας σε κλάσματα δευτερολέπτου.
Η σύνταξη της είναι η εξής:

$$double\ MPI\_Wtime\ (void).$$

...Συναρτήσεις MPI....

Παράδειγμα

/*Δήλωση μεταβλητών*/
/*Συνάρτηση αρχικοποίησης MPI*/

$$MPI\_Init(\&argc, \&argv);$$

/*Εύρεση τάξης διεργασίας σε ένα μεταδότη*/

$$MPI\_Comm\_rank(MPI\_COMM\_WORLD, \&my\_rank)$$

/*Εύρεση αριθμού διεργασιών που υπάρχουν σε ένα μεταδότη*/

$$MPI\_Comm\_size(MPI\_COMM\_WORLD, \&size)$$

/*Εύρεση ονόματος επεξεργαστή στον οποίο εκτελείτε μια διεργασία*/

$$MPI\_Get\_processor\_name(proc\_name, \&namelen)$$

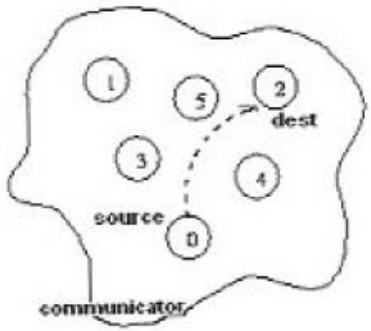
/*Εντολές*/
/*Συνάρτηση τερματισμού */

$$MPI\_Finalize();$$

... Συναρτήσεις MPI....

Συναρτήσεις σημείου προς σημείο επικοινωνίας

Στην σημείο προς σημείο επικοινωνία πάντα εμπλέκονται ακριβώς δύο διεργασίες. Η μια διεργασία στέλνει ένα μήνυμα στην άλλη. Αυτό τις διακρίνει σε σχέση με άλλες συναρτήσεις επικοινωνίας του MPI, όπως οι συλλογικές επικοινωνίες.



Σχήμα: Στην σημείο προς σημείο επικοινωνία μία διεργασία στέλνει μήνυμα σε μια άλλη συγκεκριμένη διεργασία.

... Συναρτήσεις MPI....

MPI_Send

Η συνάρτηση αυτή στέλνει ένα μήνυμα αναστέλλοντας την εκτέλεση της διεργασίας αποστολέα.

$$int\ MPI\_Send(void\ ^{*}buf,\ int\ count,\ MPI\_Datatype\ datatype,\ int\ dest,\ int\ tag,\ MPI\_Comm\ comm)$$

όπου
buf : Η διεύθυνση αρχής της ενδιάμεσης μνήμης όπου βρίσκεται το προς αποστολή μήνυμα.
count : Το πλήθος των προς αποστολή στοιχείων (μέγεθος της ενδιάμεσης μνήμης).
datatype : Τύπος δεδομένων που αποστέλλονται
(*MPI_INT* για ακέραιο, *MPI_CHAR* για χαρακτήρα, *MPI_DOUBLE* για αριθμό διπλής ακρίβειας, κ.λ.π.).
dest : Είναι η διεργασία προορισμού για το μήνυμα. Καθορίζεται από την τάξη της διεργασίας προορισμού μέσα στην ομάδα που σχετίζεται με τον Μεταδότη *comm*
tag : Είναι μια ετικέτα που χρησιμοποιείται από τον αποστολέα για να διακρίνει διαφορετικούς τύπους μηνυμάτων. Η ετικέτα αυτή χρησιμοποιείται από τον προγραμματιστή για τον απόλυτο καθορισμό των μηνυμάτων. Μπορεί να επιλεγεί οποιοσδήποτε ακέραιος αριθμός μεταξύ 0 και $2^{32} - 1$.
comm : Είναι ο Μεταδότης στον οποίο βρίσκονται οι διεργασίες αποστολής και λήψης. Μόνο οι διεργασίες που βρίσκονται στον ίδιο μεταδότη μπορούν να επικοινωνήσουν.

... Συναρτήσεις MPI....

Σχήμα: *MPI_Send*.Η διεργασία αποστολέας ελπίζει ότι η άλλη διεργασία θα παραλάβει το μήνυμα.

... Συναρτήσεις MPI....

MPI_Recv

Η συνάρτηση αυτή λαμβάνει ένα μήνυμα αναστέλλοντας την εκτέλεση της διεργασίας-παραλήπτη.

$$int\ MPI\_Recv(void\ ^{*}buf,\ int\ count,\ MPI\_Datatype\ datatype,\ int\ source,\ int\ tag,\ MPI\_Comm\ comm,\ MPI\_Status\ status)$$

όπου

buf	: Η διεύθυνση αρχής της ενδιάμεσης μνήμης στην οποίο θα αποθηκευθεί το μήνυμα.
count	: Το μέγιστο πλήθος στοιχείων που θα ληφθούν.
datatype	: Ο τύπος των δεδομένων που θα ληφθούν σε κάθε στοιχείο.
source	: Η τάξη της διεργασίας-αποστολέα.
tag	: Ο τύπος του μηνύματος.
comm	: Ο μεταδότης στον οποίο ανήκει η διεργασία αποστολέας.
status	: Επιστρέφει πληροφορίες σχετικά με το αποτέλεσμα του *receive*.

... Συναρτήσεις MPI....

MPI_Status status.

Η συνάρτηση *MPI_Status* χρησιμοποιείται για να δηλώσει την μεταβλητή status η οποία χρησιμοποιείται για να λάβει πληροφορίες σχετικές με το μέγεθος, το tag και την πηγαία διεργασία του μηνύματος που λαμβάνεται.

... Συναρτήσεις MPI....

Παράδειγμα...

Πρόγραμμα κατά το οποίο όλοι οι επεξεργαστές εκτός του 0 τυπώνουν το μήνυμα hello ενώ ο επεξεργαστής 0 λαμβάνει το μήνυμα.

```
#include < stdio.h >
#include "mpi.h"
int main(argc, argv)
int argc;
char *argv[];
{
int myrank;  / * Rank of process * /
int numprocs;  / * Number of processes * /
int source;  / * Rank of sender * /
int dest;  / * Rank of receiver * /
char message[100];  / * Storage for the message * /
MPI_Status status;  / * Return status for receive * /
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

... Συναρτήσεις MPI....

...Παράδειγμα

```
if (myrank! = 0) / * My rank is not 0, so I must send greeting * /
{sprintf(message, "Greetings from process %d!", myrank);
dest = 0;
/ * Use strlen(message) + 1 to include 0 * /
MPI_Send(message, strlen(message) + 1, MPI_CHAR, dest, 15, MPI_COMM_WORLD);
}
else
{
/ * My rank is 0, so I must receive the greetings * /
for(source = 1; source < numprocs; source + +)
{MPI_Recv(message, 100, MPI_CHAR, source, 15, MPI_COMM_WORLD, &status);
printf( "%s", message);
}
}
MPI_Finalize();
}
```

... Συναρτήσεις MPI....

Συναρτήσεις συλλογικής επικοινωνίας.

MPI_Bcast

Αποστέλλει το ίδιο μήνυμα από τη διεργασία-ρίζα σε όλες τις υπόλοιπες διεργασίες.

$$int\ MPI\_Bcast(void\ ^*buffer,\ int\ count,\ MPI\_Datatype\ datatype,\ int\ root,\ MPI\_Comm\ comm)$$

όπου

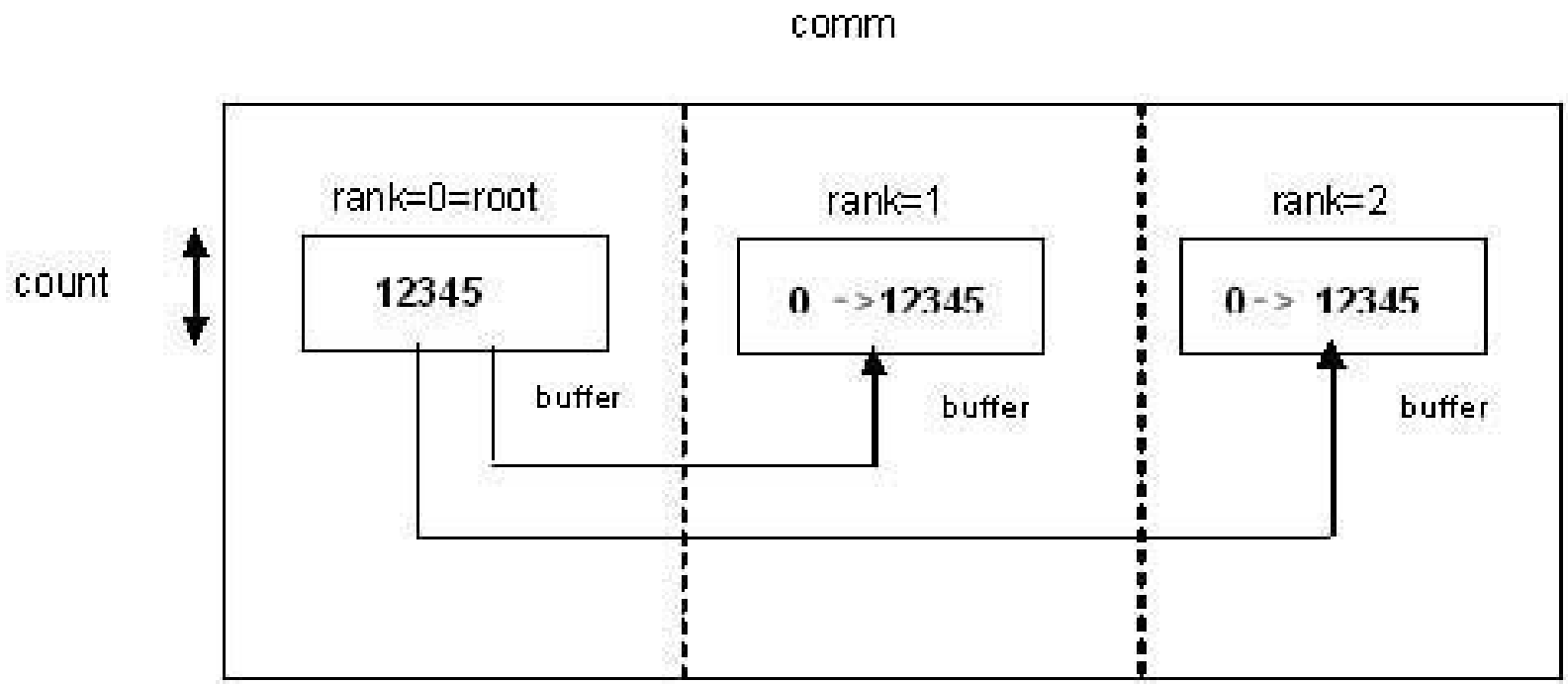
buffer : Η διεύθυνση της ενδιάμεσης μνήμης που περιέχει το προς αποστολή μήνυμα.

count : Ο αριθμός των στοιχείων που υπάρχουν στην ενδιάμεση μνήμη.

datatype : Ο τύπος δεδομένων της ενδιάμεσης μνήμης.

root : Η τάξη της διεργασίας-ρίζας.

comm : Ο Μεταδότης της συλλογικής επικοινωνίας.
($comm = MPI\_COMM\_WORLD$)



Σχήμα: *MPI_Bcast*. Η διεργασία 0 στέλνει τον αριθμό 12345 στις διεργασίες 1 και 2.

... Συναρτήσεις MPI....

MPI_Barrier.

Χρησιμοποιείται για να συγχρονίσει όλες τις διεργασίες οι οποίες συμμετέχουν σε μια συλλογική επικοινωνία.

$$int\ MPI\_Barrier\ (MPI\_Comm\ comm),$$

όπου $comm(= MPI\_COMM\_WORLD)$ είναι ο Μεταδότης της συλλογικής επικοινωνίας.

... Συναρτήσεις MPI....

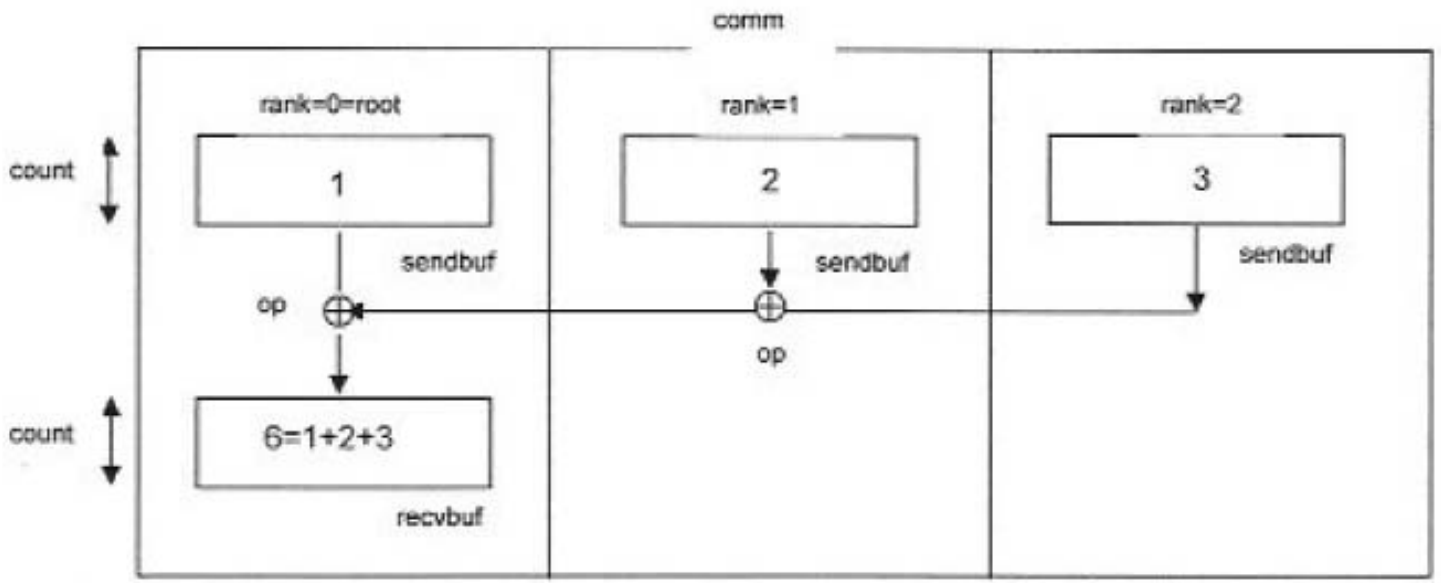
MPI_Reduce

Η συνάρτηση *MPI_Reduce* επιτρέπει μια ή περισσότερες λειτουργίες ρύθμισης στις τιμές που υποβλήθηκαν από όλες τις διεργασίες στον Μεταδότη.

$$int\ MPI\_Reduce(void\ ^*sendbuf,\ void\ ^*recvbuf,\ int\ count,$$
$$MPI\_Datatype\ datatype, MPI\_Op\ op,\ int\ root,\ MPI\_Comm\ comm)$$

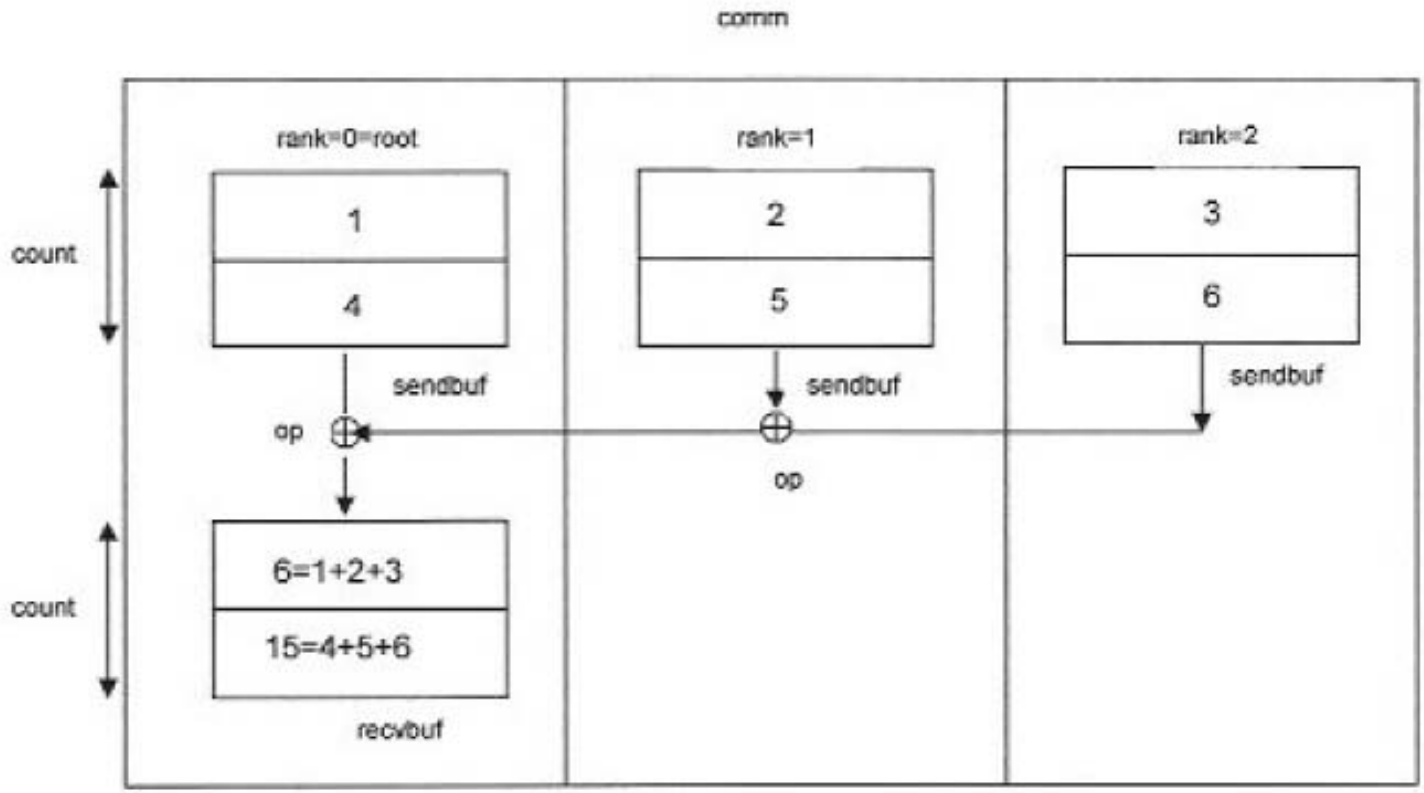
όπου,
sendbuf : Η διεύθυνση αρχής της ενδιάμεσης μνήμης αποστολής
recvbuf : Η διεύθυνση αρχής της ενδιάμεσης μνήμης λήψης
count : Το πλήθος των στοιχείων που υπάρχουν στην ενδιάμεση μνήμη αποστολής.
datatype : Ο τύπος δεδομένων της ενδιάμεσης μνήμης.
op : Ο τελεστής ρύθμισης
root : Η τάξη της διεργασίας-ρίζας.
comm : Ο Μεταδότης στον οποίο ανήκουν οι διεργασίες.
Παρατήρηση: Η συνάρτηση *MPI_Reduce* καλείται από όλες τις διεργασίες χρησιμοποιώντας τα ίδια ορίσματα για τις παραμέτρους count, datatype, op και root . Αν μια ακολουθία στοιχείων εκχωρήθηκε σε μια διεργασία, τότε η λειτουργία ρύθμισης πρόκειται να εκτελεστεί στοιχείο προς στοιχείο, σε κάθε είσοδο της ακολουθίας.

... Συναρτήσεις MPI....



Σχήμα: *MPI_Reduce* για απλές μεταβλητές.

... Συναρτήσεις MPI....



Σχήμα: *MPI_Reduce* για πίνακες.

...Συναρτήσεις MPI....

Προκαθορισμένοι τελεστές

MPI_MAX	Υπολογισμός του μέγιστου
MPI_MIN	Υπολογισμός του ελάχιστου
MPI_SUM	Υπολογισμός του αθροίσματος
MPI_PROD	Υπολογισμός του γινομένου
MPI_LAND	Υπολογισμός του λογικού *AND*
MPI_BAND	Υπολογισμός του *bitwise AND*
MPI_LOR	Υπολογισμός του λογικού *OR*
MPI_BOR	Υπολογισμός του *bitwise OR*
MPI_LXOR	Υπολογισμός του λογικού αποκλειστικού *OR*
MPI_BXOR	Υπολογισμός του *bitwise* αποκλειστικού *OR*

... Συναρτήσεις MPI....

Παράδειγμα...

Πρόγραμμα που θα υπολογίζει το άθροισμα των τετραγώνων $1^2 + 2^2 + ... + n^2$ με χρήση των συναρτήσεων *MPI_Bcast()* και *MPI_Reduce()*.

```
#include < stdio.h >
#include "mpi.h"
main(int argc, char** argv)
{
intmy_rank, p, k, root, a1local, a2_local, local_num, endnum, local_res, final_res, namelen;
charproc_name[MPI_MAX_PROCESSOR_NAME];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Get_processor_name(proc_name, &namelen);
```

... Συναρτήσεις MPI....

...Παράδειγμα

```
if(my_rank == 0)
{
printf("Dose plithos arithmwn :");
scanf("%d", &endnum);
}
root = 0;
MPI_Bcast(&endnum, 1, MPI_INT, root, MPI_COMM_WORLD);
local_res = 0;
local_num = endnum/p;
a1_local = (my_rank * local_num) + 1;
a2_local = a1_local + local_numffl1;
for(k = a1_local; k <= a2_local; k++)
local_res = local_res + (k * k);
printf("Process %d on %s : local result = %d", my_rank, proc_name, local_res);
root = 0;
MPI_Reduce(&local_res, &final_res, 1, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD);
if(my_rank == 0)
{
printf("Total result for N = %d is equal to : %d", endnum, final_res);
}
MPI_Finalize();
```

... Συναρτήσεις MPI....

MPI_Allreduce

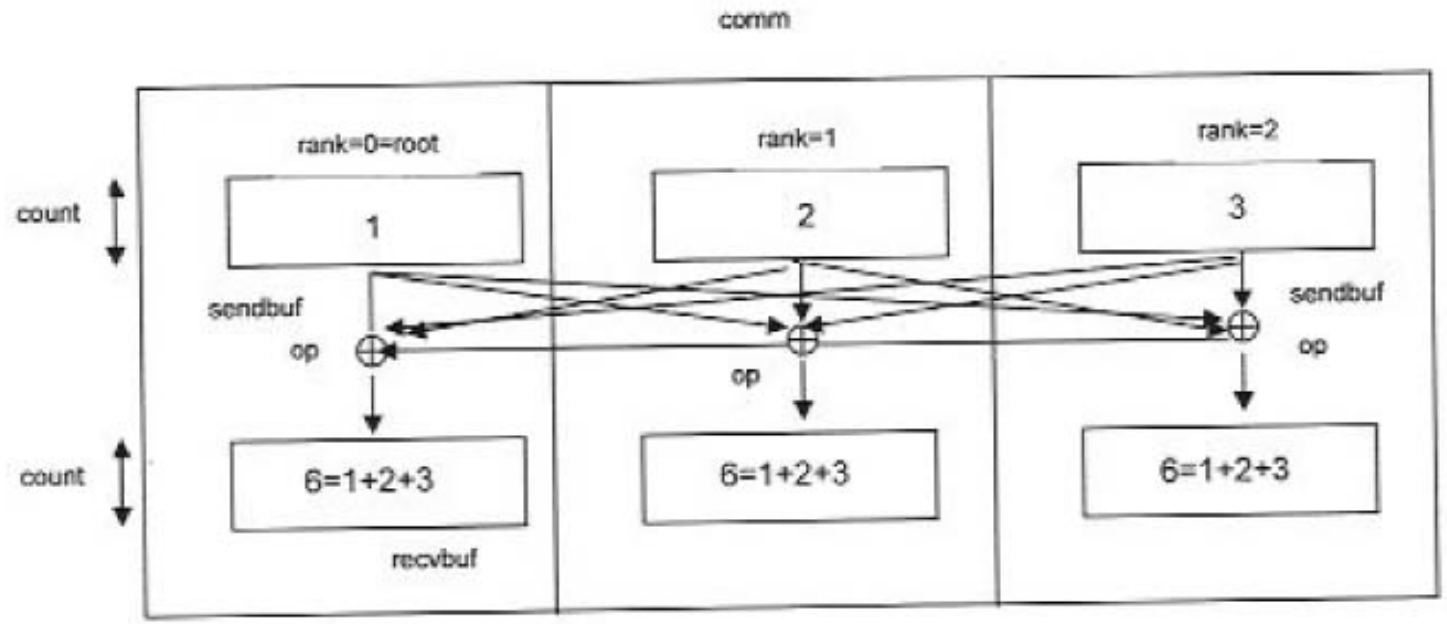
Η συνάρτηση είναι παρόμοια με την *MPI_Reduce* με τη διαφορά ότι συνδυάζει τιμές από όλες τις διεργασίες και στέλνει το αποτέλεσμα πίσω σε όλες τις διεργασίες (συμπεριλαμβανομένου του εαυτού της).

$$int\ MPI\_Allreduce(void\ ^*sendbuf,\ void\ ^*recvbuf,\ int\ count,$$
$$MPI\_Datatype\ datatype, MPI\_Op\ op, MPI\_Comm\ comm)$$

όπου

sendbuf : Η διεύθυνση αρχής της ενδιάμεσης μνήμης αποστολής.
recvbuf : Η διεύθυνση αρχής της ενδιάμεσης μνήμης λήψης.
count : Το πλήθος των στοιχείων που υπάρχουν στην ενδιάμεση μνήμη αποστολής.
datatype : Ο τύπος δεδομένων της ενδιάμεσης μνήμης.
op : Ο τελεστής ρύθμισης
comm : Ο Μεταδότης στον οποίο ανήκουν οι διεργασίες.

... Συναρτήσεις MPI....



Σχήμα: *MPI_Allreduce*.

Τοπολογίες

Τοπολογία του υπολογιστή ή τοπολογία του δικτύου διασύνδεσης :

Η περιγραφή του τρόπου με την οποία συνδέονται οι διεργασίες σε έναν παράλληλο υπολογιστή.

Εικονική τοπολογία (virtual topology)

Στα περισσότερα παράλληλα προγράμματα, κάθε διεργασία επικοινωνεί άμεσα μόνο με κάποιες άλλες διεργασίες. Ο τρόπος αυτής της επικοινωνίας μεταξύ των διεργασιών καλείται εικονική τοπολογία.
Ο κύριος σκοπός της δημιουργίας εικονικών τοπολογιών είναι η απλούστευση του κώδικα.

... Τοπολογίες ...

Είδη εικονικών τοπολογιών

Καρτεσιανές εικονικές τοπολογίες

Κάθε διεργασία είναι ¨συνδεδεμένη¨ με τους γείτονές της σε μία διάταξη εικονικού πλέγματος (με ή χωρίς εξωτερικές συνδέσεις). Η τάξη κάθε διεργασίας εξαρτάται από τη θέση της στο πλέγμα και υπάρχει η έννοια των συντεταγμένων μιας διεργασίας.

Εικονικές τοπολογίες γραφημάτων

Κάθε διεργασία μπορεί να είναι ¨συνδεδεμένη¨ με οποιεσδήποτε άλλες διεργασίες. Η αρίθμηση των διεργασιών είναι αυθαίρετη, και φυσικά, δεν υπάρχει η έννοια των συντεταγμένων μιας διεργασίας.

... Τοπολογίες ...

Συναρτήσεις καρτεσιανής τοπολογίας

MPI_Cart_create

Η συνάρτηση αυτή δημιουργεί μια καρτεσιανή διάσπαση των διεργασιών επιστρέφοντας ένα δείκτη σε ένα νέο μεταδότη που περιέχει πληροφορίες για την καρτεσιανή τοπολογία των διεργασιών.

$$\textit{int MPI\_Cart\_create}\,(\textit{MPI\_Comm old\_commn},\ \textit{int ndims},\ \textit{int }^{*}\textit{dims},\ \textit{int }^{*}\textit{periods},\ \textit{int reorder},\ \textit{MPI\_Comm new\_comm})$$

όπου

old_comm : Υπάρχων μεταδότης. Εξ' ορισμού είναι ο *MPI_COMM_WORLD*. Αποτελεί μεταβλητή εισόδου.

ndims : Περιέχει το πλήθος των διαστάσεων του πλέγματος για την εικονική τοπολογία.

dims : Πίνακας μεγέθους ndims ο οποίος δείχνει το πλήθος των διεργασιών σε κάθε διάσταση, δηλ., το μήκος κάθε διάστασης. Π.χ. η είσοδος j δηλώνει τον αριθμό των διεργασιών στην διάσταση j .

periods : Πίνακας μεγέθους ndims ο οποίος προσδιορίζει την κατάσταση περιοδικότητας σε κάθε διάσταση και περιέχει ένα στοιχείο για κάθε διάσταση της τοπολογίας.

reorder : Επιτρέπει την αναδιάταξη των διεργασιών.

new_comm : Νέος μεταδότης για την καρτεσιανή τοπολογία. Αποτελεί μεταβλητή εξόδου.

... Τοπολογίες ...

MPI_Cart_shift

Κάθε διεργασία, χρειάζεται να στείλει και να λάβει δεδομένα από τους γείτονες της. Σε μια μονοδιάστατη διάσπαση, υπάρχουν γείτονες πάνω και κάτω. Επομένως αυτό που χρειάζεται είναι η εύρεση των γειτόνων.

$$int\ MPI\_Cart\_shift(MPI\_Comm\ comm, int\ direction, int\ disp, int\ *rank\_source, int\ *rank\_dest)$$

όπου
comm : Ο μεταδότης της Καρτεσιανής τοπολογίας
direction : Δηλώνει την κατεύθυνση μετακίνησης. Παίρνει τιμές μεταξύ 0 και $ndims - 1$.
disp : Είναι ο αριθμός των συντεταγμένων διεργασίας στην κατεύθυνση στην οποία έχουμε μετακίνηση. Ο αριθμός αυτός μπορεί να είναι θετικός ή αρνητικός. Αν είναι θετικός έχουμε κατακόρυφη μετακίνηση ενώ αν είναι αρνητικός έχουμε οριζόντια μετακίνηση.
rank_source : Δηλώνει την τάξη της διεργασίας αποστολέα (πηγής) μετά την μετακίνηση
rank_dest : Δηλώνει την τάξη της διεργασίας παραλήπτη (προορισμού) μετά την μετακίνηση.

Παράδειγμα

```
/* Initialization */
/* Get a new communicator for a decomposition of the domain */
ndims = 1;        /* number of dimentions */
dims[0] = size;       /* number of processes */
periods[0] = 0;       /* indicate whether the processes at the ''ends'' are connected */
reorder = 1;       /* we allow to MPI to find a good way to assign
the process to the elements of the decomposition */
/* MPI_Cart_create, creates a new communicator in the sixth argument
from the communicator in the first argument */
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder, &new_comm);
/* Get my position in this communicator */
MPI_Comm_rank(new_comm, &rank);
/* Get my neighbors in this communicator */
MPI_Cart_shift(new_comm, 0, 1, &nbrbottom, &nbrtop);
/* Compute the actual decomposition */
MPE_DECOMP1D(nx, size, rank, &s, &e);
/* printf(''I'm process %d with neighbors %d and %d and s = %d, e =
%d'', rank, nbrbottom, nbrtop, s, e); */

MPI_Barrier(MPI_COMM_WORLD);
```

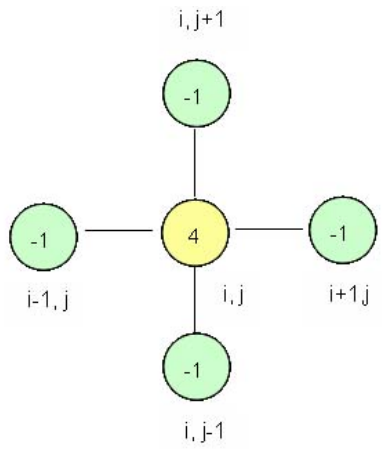
Το πρόβλημα μοντέλο

Εξίσωση Poisson $->$ Ελλειπτική Μερική Διαφορική Εξίσωση (ΜΔΕ)

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(x,y) = 0 \tag{1}$$

$$h = \frac{1}{n+1},\ x_i = ih,\ y_j = jh,\ u_{i,j} -> u \ \text{στο} \ (x_i, y_j)$$

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} + h^2 f_{i,j} = 0 \tag{2}$$



Σχήμα: Μόριο 5 σημείων.

... Το πρόβλημα μοντέλο ...

SOR

$$u_{i,j}^{(n+1)} = (1-\omega)u_{i,j}^{(n)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(n+1)} + u_{i,j-1}^{(n+1)} + u_{i,j+1}^{(n)} + u_{i+1,j}^{(n)}),$$

για $i, j = 1(1)n$

SSOR

$$u_{i,j}^{(n+1/2)} = (1-\omega)u_{i,j}^{(n)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(n+1/2)} + u_{i,j-1}^{(n+1/2)} + u_{i,j+1}^{(n)} + u_{i+1,j}^{(n)}),$$

για $i, j = 1(1)n$

$$u_{i,j}^{(n+1)} = (1-\omega)u_{i,j}^{(n+1/2)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(n+1/2)} + u_{i,j-1}^{(n+1/2)} + u_{i,j+1}^{(n+1)} + u_{i+1,j}^{(n+1)}),$$

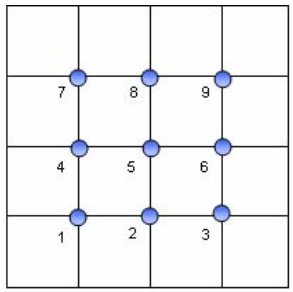
για $i, j = n(-1)1$

... Το πρόβλημα μοντέλο ...

PSD

$$u_{i,j}^{(n+1/2)} = (1-\tau)u_{i,j}^{(n)} + \frac{\omega}{4}(u_{i-1,j}^{(n+1/2)} + u_{i,j-1}^{(n+1/2)}) + \frac{\tau-\omega}{4}(u_{i-1,j}^{(n)} + u_{i,j-1}^{(n)}) +$$
$$+\frac{\tau}{4}(u_{i+1,j}^{(n)} + u_{i,j+1}^{(n)} + \frac{\tau}{4}h^2 f_{i,j}, \qquad \text{για } i,j = 1(1)n$$

$$u_{i,j}^{(n+1)} = u_{i,j}^{(n+1/2)} + \frac{\omega}{4}(u_{i+1,j}^{(n+1)} + u_{i,j+1}^{(n+1)}) - \frac{\omega}{4}(u_{i+1,j}^{(n)} + u_{i,j+1}^{(n)}),$$
$$\text{για } i,j = n(-1)1$$



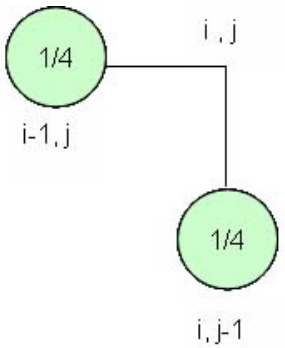
Σχήμα: Φυσική διάταξη.

Το πρόβλημα μοντέλο -Μέθοδος SOR

SOR: Μέθοδος απόλυτα ακολουθιακή

$$u_{i,j}^{(n+1)} = (1-\omega)u_{i,j}^{(n)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(n+1)} + u_{i,j-1}^{(n+1)} + u_{i,j+1}^{(n)} + u_{i+1,j}^{(n)}),$$

$$\text{για } i,j = 1(1)n$$



Σχήμα: $L = D^{-1}C_L$

Το πρόβλημα μοντέλο -Μέθοδος R/B SOR

R/B SOR: Μέθοδος απόλυτα παραλληλοποιήσιμη

Red points

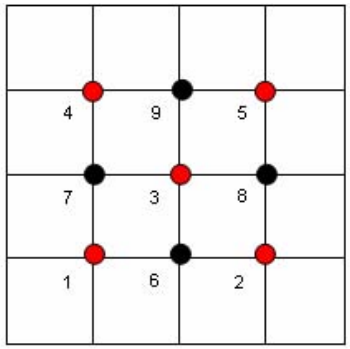
$$u_{i,j}^{(n+1)} = (1-\omega)u_{i,j}^{(n)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} + u_{i+1,j}^{(n)}),$$

για $i,j = 1(1)n$

Black points

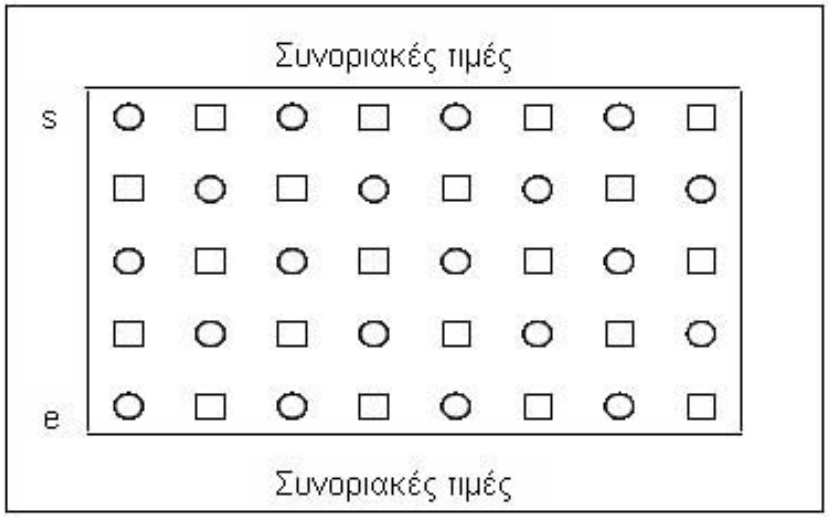
$$u_{i,j}^{(n+1)} = (1-\omega)u_{i,j}^{(n)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(n+1)} + u_{i,j-1}^{(n+1)} + u_{i,j+1}^{(n+1)} + u_{i+1,j}^{(n+1)}),$$

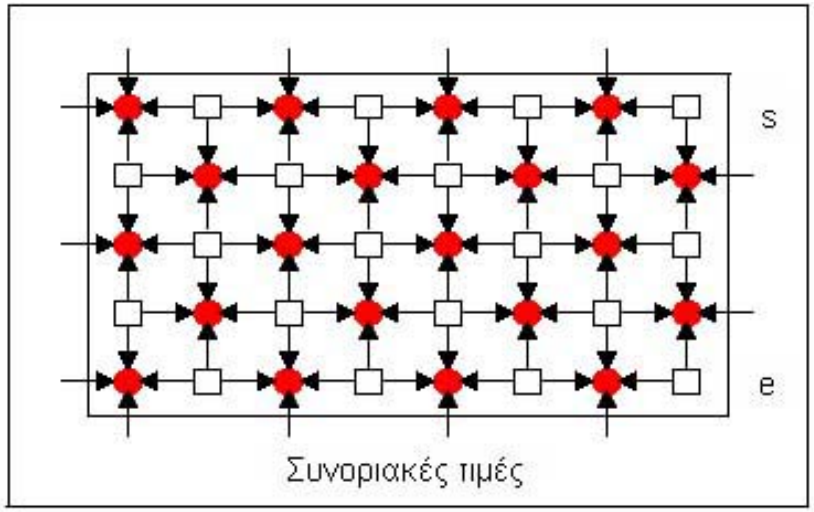
για $i,j = 1(1)n$

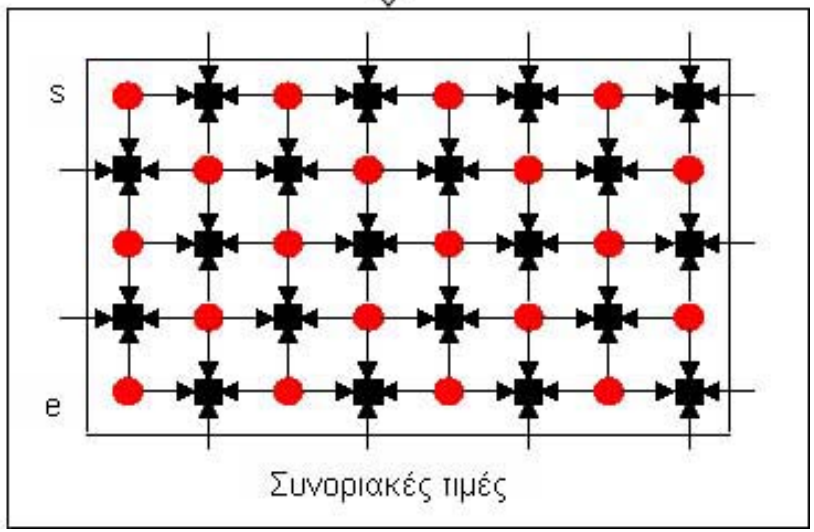


Σχήμα: Red-Black διάταξη

Επικοινωνία R/B SOR







R/B SOR μέθοδος.

Συμβολίζουμε με κύκλο τα κόκκινα σημεία και με τετράγωνα τα μαύρα σημεία. Όταν αυτά δεν είναι χρωματισμένα, δεν έχει πραγματοποιηθεί ακόμα ο υπολογισμός τους. Ορίζουμε s =$1^{η}$ γραμμή, e=$ν^{η}$ γραμμή σε κάθε ζώνη.

Αλγόριθμος R/B SOR

Για $n = 1, 2, \cdots, p$

B1: Υπολογισμός του $u_{i,j}^{(k+1)}$ στο $\Omega_{h,n}$ χρησιμοποιώντας την

$$u_{i,j}^{(k+1)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)} + u_{i+1,j}^{(k)}),$$

B2: Στείλε το $u_{i,j}^{(k+1)}$ του $\Omega_{h,n}^1$ από τον επεξεργαστή n στον n-1 για $n = 2, 3, \cdots, p$.

B3: Στείλε το $u_{i,j}^{(k+1)}$ του $\Omega_{h,n}^n$ από τον επεξεργαστή n στον n+1 για $n = 1, 2, 3, \cdots, p-1$.

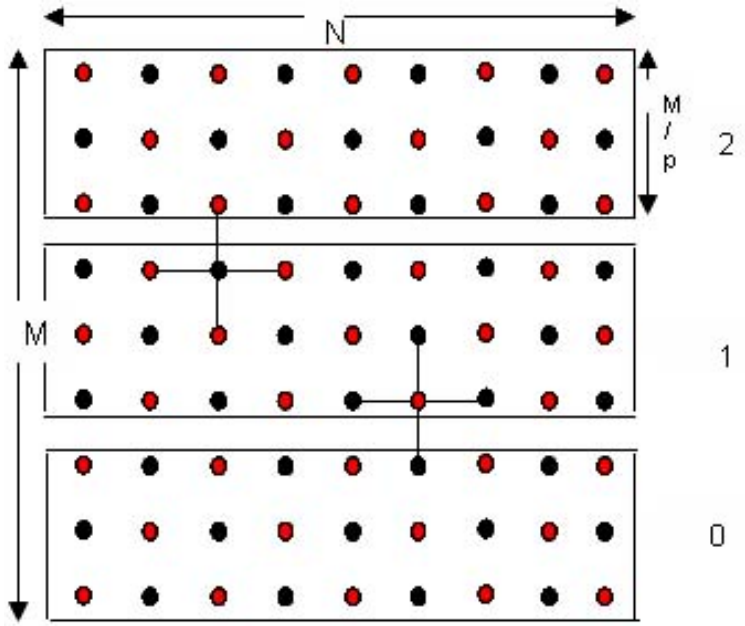
B4: Υπολογισμός του $u_{i,j}^{(k+1)}$ στο $\Omega_{h,n}$ χρησιμοποιώντας την

$$u_{i,j}^{(k+1)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{4}(h^2 f_{i,j} + u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k+1)} + u_{i+1,j}^{(k+1)}),$$

B5: Στείλε το $u_{i,j}^{(k+1)}$ του $\Omega_{h,n}^1$ από τον επεξεργαστή n στον n-1 για $n = 2, 3, \cdots, p$.

B6: Στείλε το $u_{i,j}^{(k+1)}$ του $\Omega_{h,n}^n$ από τον επεξεργαστή n στον n+1 για $n = 1, 2, 3, \cdots, p-1$.

... Αλγόριθμος R/B SOR



Σχήμα: Οριζόντια διαμέριση

Συμπέρασμα

Στην R/B SOR η σειρά των υπολογισμών τροποποιείται σε σχέση με την κανονική SOR μέθοδο, το οποίο σημαίνει ότι μπορεί το αποτέλεσμα να διαφοροποιείται σε ακρίβεια.

... Θεωρητικά αποτελέσματα της R/B SOR

Βασικές έννοιες

$t_s \longrightarrow$ Χρόνος αρχικοποίησης της επικοινωνίας
$t_c \longrightarrow$ Χρόνος υπολογισμού μιας πράξης
$t_w \longrightarrow$ Χρόνος υπολογισμού μιας λέξης

Επιτάχυνση-Απόδοση

$$T_p = N\frac{M}{p}t_c + 4(t_s + Nt_w)$$

$$S_p = \frac{T_1}{T_p} = \frac{NMt_c}{N\frac{M}{p}t_c + 4(t_s + Nt_w)}$$

$$E_p = \frac{S_p}{p} = \frac{NMt_c}{NMt_c + 4p(t_s + Nt_w)}$$

Συμπέρασμα

Επειδή είναι $S_p \geq 1$, και επειδή ένα βήμα ενός παράλληλου αλγορίθμου χρησιμοποιώντας p επεξεργαστές χρειάζεται p βήματα το πολύ όταν υλοποιηθεί σειριακά, θα ισχύει ότι $T_1 \leq pT_p$. Συνεπώς θα ισχύει ότι: $1 \leq S_p \leq p$ και $0 \leq E_p \leq 1$.

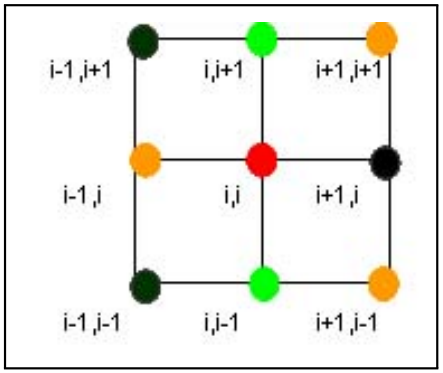
Θεωρητικά αποτελέσματα της R/B SOR

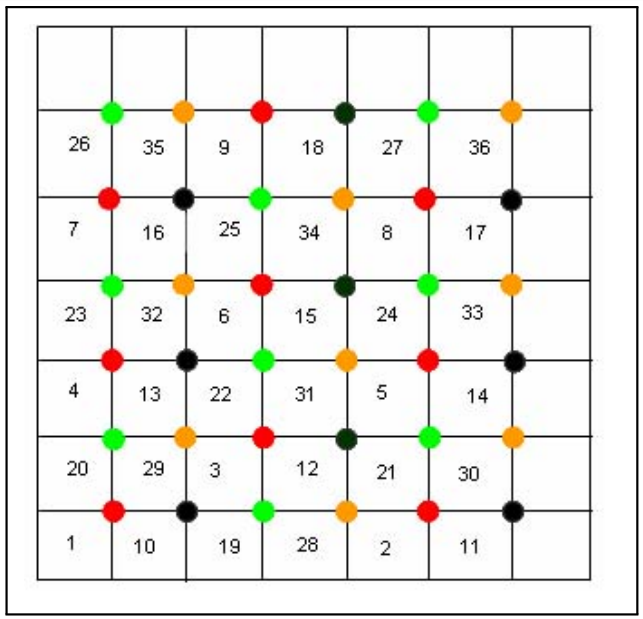
p	T_p	S_p	E_p
1	205.905645	1.000000	1.000000
2	102.953520	1.999986	0.999993
3	68.636145	2.999959	0.999986
4	51.477457	3.999919	0.999980
8	25.739426	7.999621	0.999953
12	20.591820	9.999390	0.999939
16	17.160082	11.999106	0.999926
20	12.870410	15.998375	0.999898
40	10.296607	19.997427	0.999871
60	5.149001	39.989436	0.999736

Μέθοδος R/B/G/O SOR

Διακριτοποίηση της εξίσωσης Poisson με βάση τον τύπο των 9 σημείων

$$20u_{i,j} - 4u_{i+1,j} - 4u_{i-1,j} - 4u_{i,j+1} - 4u_{i,j-1} - u_{i+1,j-1} - u_{i-1,j-1} -$$
$$-u_{i-1,j+1} - u_{i+1,j+1} = h^2 f_{i,j}, \qquad \text{στο } \Omega_h$$





Μόριο 9 σημείων - Τεσσάρων χρωμάτων διάταξη για το μόριο των 9 σημείων

Μέθοδος R/B/G/O SOR

Red points

$$u_{i,j}^{(k+1)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{5}(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)} + u_{i+1,j}^{(k)}) + \\ + \frac{\omega}{20}(h^2 f_{i,j} + u_{i-1,j+1}^{(k)} + u_{i-1,j-1}^{(k)} + u_{i+1,j+1}^{(k)} + u_{i+1,j-1}^{(k)})$$

Black points

$$u_{i,j}^{(k+2)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{5}(u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)} + u_{i+1,j}^{(k)}) + \\ + \frac{\omega}{20}(h^2 f_{i,j} + u_{i-1,j+1}^{(k)} + u_{i-1,j-1}^{(k)} + u_{i+1,j+1}^{(k+1)} + u_{i+1,j-1}^{(k+1)})$$

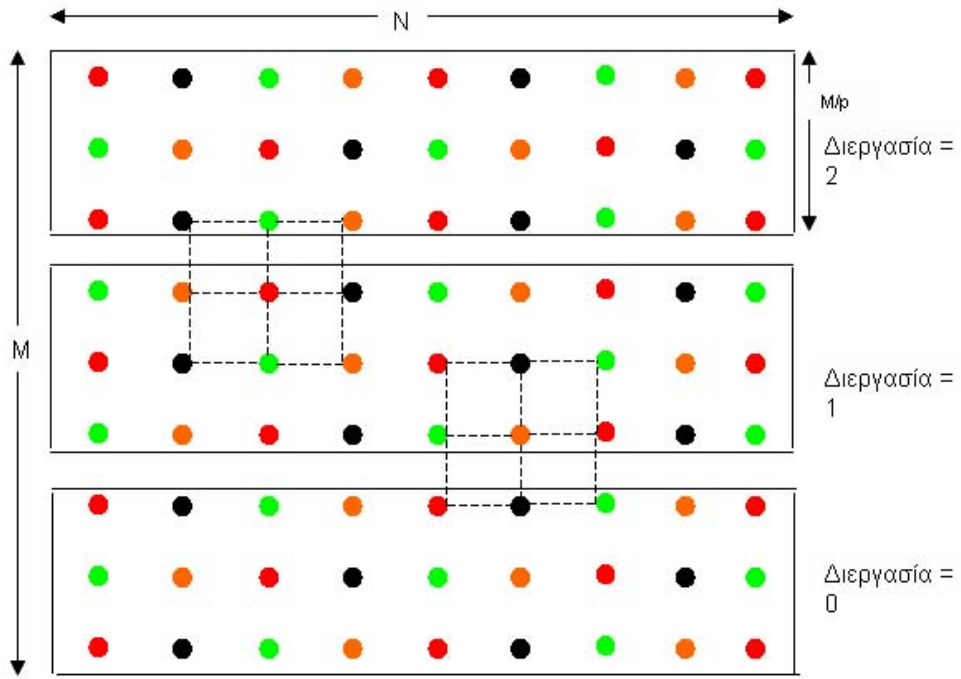
Green points

$$u_{i,j}^{(k+3)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{5}(u_{i-1,j}^{(k+2)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k+1)} + u_{i+1,j}^{(k)}) + \\ + \frac{\omega}{20}(h^2 f_{i,j} + u_{i-1,j+1}^{(k)} + u_{i-1,j-1}^{(k)} + u_{i+1,j+1}^{(k+2)} + u_{i+1,j-1}^{(k+2)})$$

Orange points

$$u_{i,j}^{(k+4)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{5}(u_{i-1,j}^{(k+3)} + u_{i,j-1}^{(k+2)} + u_{i,j+1}^{(k+2)} + u_{i+1,j}^{(k+1)}) + \\ + \frac{\omega}{20}(h^2 f_{i,j} + u_{i-1,j+1}^{(k+1)} + u_{i-1,j-1}^{(k+1)} + u_{i+1,j+1}^{(k+3)} + u_{i+1,j-1}^{(k+3)})$$

Θεωρητική μελέτη R/B/G/O SOR



Σχήμα: Οριζόντια διαμέριση πεδίου

Παράλληλοι χρόνοι

$$T_{comp} = N\frac{M}{p}t_c$$

$$T_{comm} = 8(t_s + Nt_w)$$

Σύγκριση R/B SOR - R/B/G/O SOR

Σύγκριση χρόνων επικοινωνίας

$$T_{comm}^{R/B\ SOR} = 4(t_s + Nt_w)$$

$$T_{comm}^{R/B/G/O\ SOR} = 8(t_s + Nt_w)$$

Ερώτημα

Υπάρχει μέθοδος με την ίδια ταχύτητα σύγκλισης αλλά με μικρότερη επικοινωνία;

Αριθμητικά αποτελέσματα

Αρχικές τιμές: $u^{(0)} = 1$
Συνοριακές συνθήκες: $u_x = 0,\ at\ x = 0,\ x = 1$
Ακριβής λύση: $u = 0$
$h = 1/(n+1)$
$F(x, y) = 0$
$\omega_{opt} = 2/(1 + \sin \pi h)$
Υπολογισμός σχετικού σφάλματος με βάση τον τύπο:

$$||u^{(k)} - u||_2 / ||u^{(0)} - u||_2 = \sqrt{\frac{h}{1-h}} ||u^{(k)}||_2$$

όπου $||\ .\ ||_2$ είναι η Ευκλείδεια νόρμα.

... Αριθμητικά αποτελέσματα ...

Η επιτάχυνση δεν αυξάνει γραμμικά με την αύξηση των επεξεργαστών, αλλά σταδιακά τείνει να γίνει γραμμική εξομαλύνοντας την οποιαδήποτε κλήση $->$ Νόμος του Amdahl: από ένα πλήθος επεξεργαστών και μετά η επιτάχυνση παραμένει η ίδια.

Επειδή η επιτάχυνση δεν μας δίνει κάποια πληροφορία για το πόσο χρήσιμοι είναι οι επεξεργαστές στην βελτίωση αυτή, θα πρέπει να συσχετίσουμε το όφελος στην ταχύτητα με το πόσοι επεξεργαστές χρειάζονται για να το επιτύχουμε. Αυτή η συσχέτιση γίνεται με τον υπολογισμό της αποδοτικότητας.

... Αριθμητικά αποτελέσματα ...

Συμπέρασμα

- Όσο μεγαλώνει το μέγεθος του προβλήματος παράγεται μεγαλύτερη επιτάχυνση και αποδοτικότητα για τον ίδιο αριθμό επεξεργαστών, παρόλο που και η επιτάχυνση και η αποδοτικότητα συνεχίζονται να μειώνονται καθώς αυξάνει ο αριθμός των επεξεργαστών.

Χρόνοι Εκτέλεσης της R/B SOR

p	T_p	T_{comp}	T_{comm}	T_{other}
1	159.252968	159.252968	0.000000	0.000000
2	82.350070	81.461649	0.833322	0.055099
3	57.043024	55.923836	1.073877	0.045310
8	44.639640	43.084344	1.514022	0.041274
10	29.927297	28.102512	1.787568	0.037217
12	20.010802	18.111324	1.865079	0.034399
16	17.703155	15.899697	1.769481	0.033977
20	15.664148	13.485503	2.143862	0.034783
40	11.595191	9.262831	2.301834	0.030526
60	7.049587	5.146275	1.873685	0.029628

Επαναλήψεις

Επεξεργαστές	**Επαναλήψεις R/B SOR**		
p	$N = 500$	$N = 1000$	$N = 1500$
1	1019	2000	2978
2	1019	2000	2978
3	1019	2000	2978
4	1019	2000	2978
8	1019	2000	2978
10	1019	2000	2978
12	1019	2000	2978
16	1019	2000	2978
20	1019	2000	2978
40	1019	2000	2978
60	1019	2000	2978

Διαχωρισμός σε ζώνες

Συνάρτηση διαχωρισμού σε ζώνες και καθορισμός ορίων της κάθε διεργασίας

void MPE-DECOMP1D(int n, int size, int rank,int *s, int *e)
{ *intnlocal*, *deficit*;
/*υπολογισμός του πηλίκου*/
$nlocal = n/size$;
/*υπολογισμός του αριθμού της πρώτης γραμμής σε κάθε ζώνη*/
$*s = 1 + rank * nlocal$; /*υπολογισμός του υπόλοιπου*/
$deficit = n\%size$;
$*s =* s + min(rank, deficit)$;
/*Αν $rank >= deficit$ τότε η τιμή της τελευταίας γραμμής του πλέγματος θα είναι εκτός συνόρων */
$if(rank < deficit)$
$nlocal = nlocal + 1$;
/*υπολογισμός του αριθμού της τελευταίας γραμμής σε κάθε ζώνη*/
$*e =* s + nlocal - 1$; /*αν ο αριθμός που προκύπτει για την τελευταία γραμμή του πλέγματος είναι μεγαλύτερος από το μέγεθος του πλέγματος, τότε η τιμή του γίνεται ίση με το μέγεθος του πλέγματος, με αποτέλεσμα στην τελευταία ζώνη να περιέχεται το υπόλοιπο των γραμμών του πλέγματος*/
$if(*e > n||rank == size - 1)$
$*e = n$;
return; }

Διαχωρισμός σε blocks

Συνάρτηση επιστροφής της ρίζας του αριθμού των επεξεργαστών που έχουμε προκειμένου να ορίσουμε το μέγεθος του block σε κάθε επεξεργαστή

```
int pow_10(int number)
 {
                    int i, value;
                  value = -1;
for(i = 1; i < 10; ++ i)
      {
    if(i * i == number)
                    {
                    value = i;
                       i = 10;
                    }
      }
                  return value;
}
```

Διαχωρισμός σε blocks

Ορισμός δυναμικού διδιάστατου πίνακα mxn με τέτοιο τρόπο ώστε να καταχωρείται σε συνεχόμενο block μνήμης, το οποίο απαιτείται από τον *MPI_Type_vector*. Δηλαδή, η συνάρτηση αυτή δημιουργεί τα blocks.

```
double **matrix_2D(int m, int n)
{
  int i;
  double **a;
  a = new double * [n];
  a[0] = new double[m * n];
  for(i = 1; i < n; i++)
      {
      a[i] = a[0] + i * n;
      }
  return a;
}
```

Διαχωρισμός σε blocks

Καταχωρεί τις συνοριακές τιμές κάθε block σε κάθε cpu...

```
void bc(int m, int n, double **u, int k, int p)
{ int i, val;
val = pow_10(p);
init_array(m, n, u);
if (p > 1) / * An exoume pano apo mia cpu * /
{
/ * An einai i process = 0 tote : i sinoriaki stili aristera einai miden i sinoriaki grammi kato einai miden * /
if (k == 0){
for (i = 0; i < n; i + +){
u[i][0] = 0.0;
}
for(i = 0; i < m; i + +){
u[0][i] = 0.0;
}
}
/ * An i process einai megaliteri tou 0 kai mikroteri tis rizas tou epeksergasti, tote einai stin proti grammi, ara :
i sinoriaki grammi kato einai miden * /
if (k > 0 && k < val){
for (i = 0; i < m; i + +){
u[0][i] = 0.0;
}
}
```

Διαχωρισμός σε blocks

... Καταχωρεί τις συνοριακές τιμές κάθε block σε κάθε cpu

```
/* An i process +
1 einai isi me tin riza tou arithmou ton cpu, tote einai stin proti grammi kai teleutaio block deksia, ara :
i sinoriaki stili deksia einai miden i sinoriaki grammi kato einai miden alla exei kataxorithei idi. */
if (k + 1 == val){
for (i = 0; i < m; i++){
u[i][m − 1] = 0.0;
}
}
/* An i process+1 einai pollaplasio tis rizas tou arithmou ton cpu, tote einai sto teleutaio block deksia, ara :
i sinoriaki stili deksia einai miden */ if ((k + 1)%val == 0) {
for (i = 0; i < m; i++){
u[i][m − 1] = 0.0;
}
} /* An i process einai pollaplasio tis rizas tou arithmou ton cpu, tote einai sto proto aristero block, ara :
i sinoriaki stili aristera einai miden */
if (k%val == 0) {
for (i = 0; i < n; i++){
u[i][0] = 0.0;
}
}
```

Διαχωρισμός σε blocks

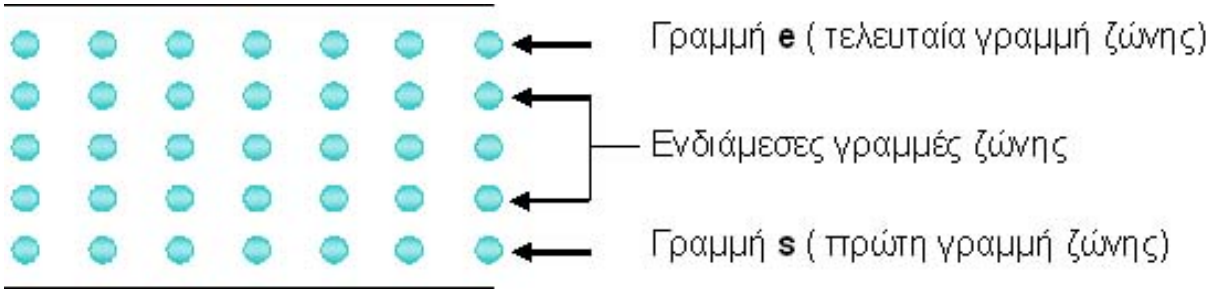
... Καταχωρεί τις συνοριακές τιμές κάθε block σε κάθε cpu

```
/ * An i process einai megaliteri i isi me val * (val − 1), tote einai stin teleutaia grammi, ara :
i sinoriaki grammi pano einai miden * /
if (k >= val * (val − 1)) {
for (i = 0; i < m; i + +){
u[n − 1][i] = 1.0;
}}}
/ *
An exoume mia cpu, tote thetoume sinoriakes stiles miden, tin kato sinoriaki grammi miden kai tin pano isi me ena.
/
else if (p == 1) {
for (i = 0; i < n; i + +){
u[i][0] = 0.0;
u[i][n − 1] = 0.0;
}
```

Κλήση τους στη main

```
val = pow_10(p); //briskoume tin riza tou arithmou tonepeksergaston
sub = (m − 2)/val + 2; //briskoume poses grammes kai stiles exei kathe block
m = stiles pinaka me tis sinoriakes
v = matrix_2D(sub, sub); / * dimiourgia pinaka sub x sub * /
```

Συνάρτηση για επικοινωνία έχοντας strips



Σχήμα: Ονομασία γραμμών σε κάθε ζώνη του πλέγματος

Αποστολή και λήψη μιας γραμμής με χρήση φανταστικών γραμμών

```
void exchng1(double **U0, int nx, int s, int e, MPI_Comm new_comm, int nbrbottom, int nbrtop, int rank)
{
MPI_Status status;
MPI_Send(&U0[s][1], nx, MPI_DOUBLE, nbrbottom, 1, new_comm);
MPI_Recv(&U0[e + 1][1], nx, MPI_DOUBLE, nbrtop, 1, new_comm, &status);
return;
}
```

Συνάρτηση υπολογισμού της μεθόδου jacobi

Μόριο 5 σημείων

```
void sweep1d(double **U0, double **f, int nx, int s, int e, double **U1, double h)
{
register int i, j;
for (i = 1; i <= nx; i++)/ * den ypologizoume synoriakes times, diladi gia i = 0, i = nx + 1 * /
U1[s][i] = (h * h * f[s][i] + U0[s][i − 1] + U0[s − 1][i] + U0[s][i + 1] + U0[s + 1][i]);
return;
}
```

Υπολογισμός Κριτηρίου διακοπής

Κριτήριο διακοπής

Σύγκλιση επιτυγχάνεται όταν όταν η διαφορά μεταξύ δύο διαδοχικών προσεγγιστικών λύσεων είναι μικρότερη από το 10^{-5}, δηλ. όταν πληρείται το ακόλουθο κριτήριο:

$$||u^{(n+1)} - u^{(n)}||_2 = \sqrt{\Sigma_{i=1}^{N} \left(u_i^{(n+1)} - u_i^{(n)}\right)^2} < \epsilon,\ \epsilon = 10^{-5}$$

Ο έλεγχος του κριτηρίου διακοπής πραγματοποιείται λοιπόν για $\epsilon = 10^{-5}$ και περιλαμβάνει καθολική επικοινωνία, δηλ., επικοινωνία με όλους τους επεξεργαστές. Ο τρόπος υλοποίησης για τον έλεγχο του κριτηρίου διακοπής γίνεται με την κλήση της συνάρτησης *MPI_Allreduce* η οποία συνδυάζει τιμές από όλες τις διεργασίες και στέλνει το αποτέλεσμα πίσω σε όλες τις διεργασίες (συμπεριλαμβανομένου του εαυτού της. Κάθε επεξεργαστής P_i υπολογίζει το και το στέλνει στον P_0, ο οποίος αφού λάβει όλα τα επιμέρους αποτελέσματα, με την κλήση της *MPI_Allreduce*, υπολογίζει το τελικό αποτέλεσμα συνδυάζοντας αυτά τα οποία έλαβε, και το στέλνει σε όλους τους επεξεργαστές.

Υπολογισμός Κριτηρίου διακοπής

Συνάρτηση υπολογισμού Κριτηρίου διακοπής

```
double diff(double **U0, double **U1, int nx int s, int e, double h)
{                                   double sum; register int i, j;
sum = 0.0;
for (j = s; j <= e; j++)                          /* j-> grammi kathe strip */
{for (i = 1; i <= nx; i++)                          /* i-> stili kathe strip */
{/* ipologismos tou athroismatos gia to kritirio diakopis */
sum = sum + (U0[j][i] - U1[j][i]) * (U0[j][i] - U1[j][i]);
U0[j][i] = U1[j][i];
}
}
return sum;
}
```