Επιστημονικοί Υπολογισμοί - Μέρος ΙΙΙ: Παράλληλοι Υπολογισμοί

Χαρμανδάρης Βαγγέλης, Τμήμα Εφαρμοσμένων Μαθηματικών

Πανεπιστήμιο Κρήτης, Εαρινό Εξάμηνο 2013/14

Κεφάλαιο 6: Εφαρμογές ΙΙ

- **Παράλληλοι Υπολογισμοί Πινάκων.**
- **Επίλυση Συστήματος Γραμμικών Εξισώσεων.**
- **Μερικές Διαφορικές Εξισώσεις.**

Παράλληλοι Υπολογισμοί Πινάκων

- Due to their regular structure, parallel computations involving matrices and vectors readily lend themselves to data-decomposition.

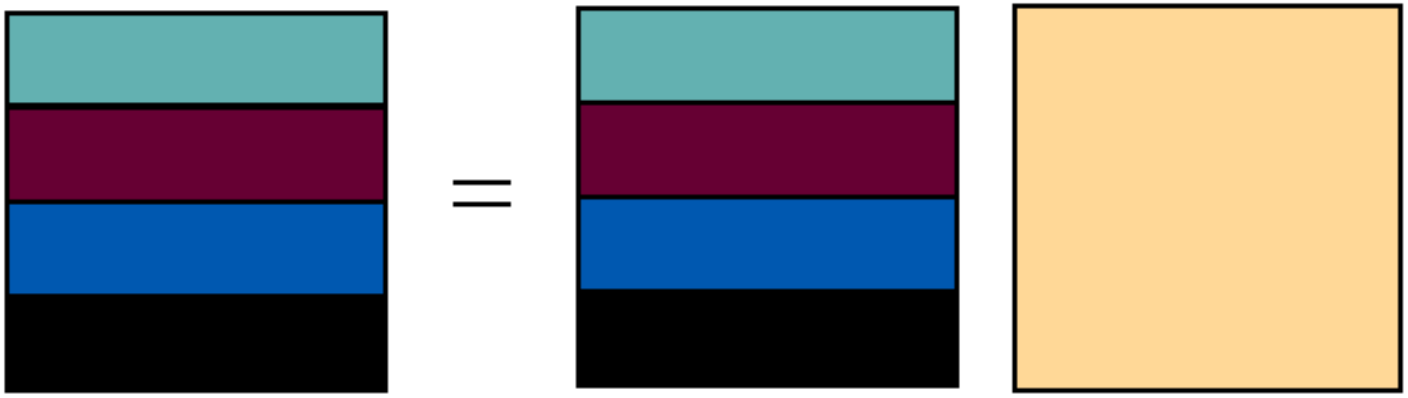
- Typical algorithms rely on input, output, or intermediate data decomposition.

- Most algorithms use one- and two-dimensional block, cyclic, and block-cyclic partitionings.

Παράλληλοι Υπολογισμοί Πινάκων

Decomposition for matrix-matrix multiplication in C.

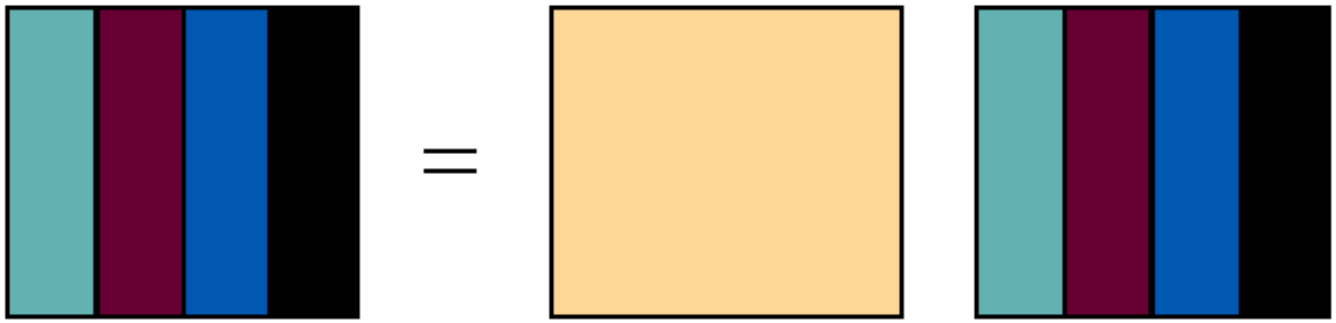
- **A** = **B***C product.
- Matrices **A** and **B** are shown with multicolored rows. Each color denotes a different process. Matrix **C** is broadcast to all processes.



Παράλληλοι Υπολογισμοί Πινάκων

Decomposition for matrix-matrix multiplication in Fortran.

- **A** = **B***C product.
- Matrices **A** and **C** are shown with multicolored columns. Each color denotes a different process. Matrix **B** is broadcast to all processes.



Παράλληλοι Υπολογισμοί Πινάκων

The basic steps of the algorithm (in C language) are:

1. Distribute the columns of **C** among the processors using a scatter operation.

2. Broadcast the matrix **B** to every processor.

3. Form the product of **B** with the columns of **C** on each processor. These are the corresponding columns of **A**.

4. Bring the columns of **A** back to one processor using a gather operation.

Παράλληλοι Υπολογισμοί Πινάκων

- Assume **A**(n,n), **B**(n,n), **C**(n,n) matrices and p processes.

/* Data distribution */

```
if( myrank == 0 ) {
    for( i=1; i<p; i++ ) {
      MPI_Send( &a[from[i]], n*n/p, MPI_INT, i, tag,
          MPI_COMM_WORLD );
      MPI_Send( &b, n*n, MPI_INT, i, tag, MPI_COMM_WORLD );
    }

} else {
    MPI_Recv( &a[from[myrank]], n*n/p, MPI_INT, 0, tag,
                  MPI_COMM_WORLD, &status );
    MPI_Recv( &b, n*n, MPI_INT, 0, tag, MPI_COMM_WORLD,
        &status );

}
```

Παράλληλοι Υπολογισμοί Πινάκων

```
 /* Computation */

 for ( i=from[myrank]; i<to[myrank]; i++) {
   for (j=0; j<n; j++) {
     C[i][j]=0;
     for (k=0; k<n; k++)
       C[i][j] += A[i][k]*B[k][j]; }
   }


/* Result gathering */
if (myrank!=0) {
  for( i=1; i<p; i++ )
  MPI_Recv( &c[from[i]], n*n/p, MPI_INT, i, tag, MPI_COMM_WORLD, &status);
else
  MPI_Send( &c[from[myrank]], n*n/p, MPI_INT, 0, tag, MPI_COMM_WORLD);
}
```

Παράλληλοι Υπολογισμοί Πινάκων

• Use of collective communication routines

```
main(int argc, char *argv[])

{
    ….
    from = (myrank * n)/p;
    to = ((myrank+1) * n)/p;

    MPI_Scatter (a, n*n/p, MPI_INT, a, n*n/p, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast (b,n*n, MPI_INT, 0, MPI_COMM_WORLD);
    ...
    /* Computation */

    MPI_Gather (C[from], n*n/p, MPI_INT, c[from], n*n/p,  MPI_INT, 0,
                    MPI_COMM_WORLD);

}
```

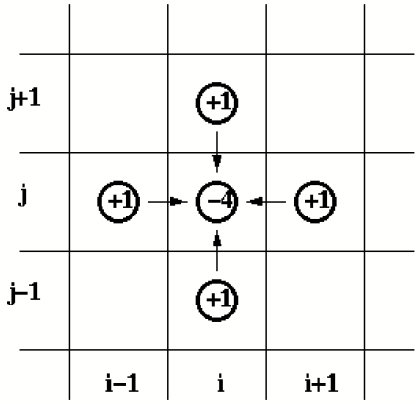
Partial Differential Equations

Πρόβλημα – Μοντέλο: Laplace συνοριακών τιμών:

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = f(x,y)$$

Διαμέριση: $h = \frac{1}{N+1}, \qquad x_i = i*h, \;\; y_i = j*h, \qquad i,j = 0,1,\ldots,N+1$

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



Partial Differential Equations

Boundary Conditions (Συνοριακές συνθήκες):

Consider simple 1D bar discretised into 8 intervals

- Interior values U_2, U_3 ... U_9 determined by PDE
- U_1 and U_{10} determined by boundary conditions

$$U_{i-1} - 2U_i + U_{i+1} = 0$$

U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_{10}

Classification of BCs:

- Dirichlet: specify value, e.g. $U_1 = c_0$
- Neumann: specify derivative , e.g. $\left.\frac{\partial U}{\partial x}\right|_{x=0} = c_0$
- Discrete form , e.g. $U_1 - U_2 = c_0$

Partial Differential Equations

Η διαμέριση του χώρου ολοκλήρωσης οδηγεί σε σύστημα γραμμικών εξισώσεων το οποίο λύνεται επαναληπτικά. Διαφορετικοί μέθοδοι επίλυσης.

Έστω η *i* εξίσωση: $\sum_{j=1}^{M} a_{i,j} U_j = b_i$

Jacobi:

➢Η λύση (στην *k*+1 επανάληψη) είναι:

$$U_i^{(k+1)} = \frac{b_i - \sum_{j \neq i}^{M} a_{i,j} U_j^{(k)}}{a_{i,j}}$$

Πλεονεκτήματα:

- ➢Εύκολη στη χρήση.
- ➢Ανεξάρτητοι υπολογισμοί (εύκολος παραλληλισμός του σειριακού κώδικα).

Μειονεκτήματα:

- ➢Πολύ αργός ρυθμός σύγκλισης.
- ➢Συνήθως χρησιμοποιείται μόνο για εκπαιδευτικούς σκοπούς.

Partial Differential Equations

Gauss – Seidel:

Χρησιμοποιεί τιμές για την U μόλις αυτές υπολογισθούν.

➢Η λύση (στην k+1 επανάληψη) είναι:

$$U_i^{(k+1)} = \frac{b_i - \sum_{j<i}^{M} a_{i,j} U_j^{(k+1)} - \sum_{j>i}^{M} a_{i,j} U_j^{(k)}}{a_{i,j}}$$

Πλεονεκτήματα:

- ➢Γρηγορότερος ρυθμός σύγκλισης.
- ➢Σχετικά απλή η ανάπτυξη του σειριακού κώδικα.
- ➢Ανεξάρτητοι υπολογισμοί (εύκολος παραλληλισμός του σειριακού κώδικα.

Μειονεκτήματα:

- ➢Οι πράξεις είναι σειριακές (ακολουθιακές): ο παραλληλισμός απαιτεί τροποποίηση της μεθόδου.

Partial Differential Equations

Successive Over-Relaxation, SOR:

Επέκταση της Gauss-Seidel: Λαμβάνει υπόψη την προηγούμενη επανάληψη μέσω ενός βάρους w.

➢Η λύση (στην k+1 επανάληψη) είναι:

$$U_i^{(k+1)} = w\left(\frac{b_i - \sum_{j<i}^{M} a_{i,j}U_j^{(k+1)} - \sum_{j>i}^{M} a_{i,j}U_j^{(k)}}{a_{i,j}}\right) + (1-w)U_i^{(k)} = w\bar{U}_i^{(k+1)} + (1-w)U_i^{(k)}$$

➢Για την σταθερότητα της μεθόδου απαιτείται: $0 < w < 2$

➢$w = 1$: Gauss-Seidel, $w < 1$: Under-relaxation, $w > 1$: Over-relaxation.

Partial Differential Equations

Red/Black SOR: Κριτήριο Σύγκλισης

- Σχεδόν πάντα απαιτείται-εξετάζεται η ακρίβεια στη λύση. Συνεπώς ο αλγόριθμος χρειάζεται να τρέχει έως ότου συγκλίνει και όχι για συγκεκριμένο αριθμό επαναλήψεων.

Πιθανά Κριτήρια Σύγκλισης:

A) Η διαφορά μεταξύ διαδοχικών επαναλήψεων για όλα τα σημεία είναι μικρότερη ενός μικρού αριθμού $\max\left|U_{i,j}^{(k+1)} - U_{i,j}^{(k)}\right| \le \varepsilon$

B) Η Ευκλείδια νόρμα μεταξύ διαδοχικών επαναλήψεων είναι μικρότερη ενός μικρού αριθμού $\sqrt{\sum_i \sum_j \left(U_{i,j}^{(k+1)} - U_{i,j}^{(k)}\right)^2} \le \varepsilon$

Partial Differential Equations

Model Problem: Laplace (*f*=0).

$$U_{i,j} = \frac{1}{4}\left(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1}\right)$$

Jacobi:

$$U_{i,j}^{(k+1)} = \frac{1}{4}\left(U_{i+1,j}^{(k)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k)}\right)$$

Gauss-Seidel:

$$U_{i,j}^{(k+1)} = \frac{1}{4}\left(U_{i+1,j}^{(k)} + U_{i-1,j}^{(k+1)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k+1)}\right)$$

SOR:

$$U_{i,j}^{(k+1)} = \frac{w}{4}\left(U_{i+1,j}^{(k)} + U_{i-1,j}^{(k+1)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k+1)}\right) + (1-w)U_{i,j}^{(k)}$$

PDEs – Model Problem (Laplace)

SOR: Serial Implementation

Ο σειριακός αλγόριθμος περιλαμβάνει τα παρακάτω βήματα:

A) Αρχικοποίηση του grid: δίνεται μια αρχική τιμή στο πίνακα $U(i,j)$ (== $Uold(i,j)$) για όλα τα i, j.

B) Ξεκίνημα της επαναληπτικής διαδικασίας. Υπολογισμός της καινούριας εκτίμησης του $U(i,j)$.

C) Έλεγχος της σύγκλισης: εάν η Ευκλείδια νόρμα είναι μικρότερη μιας προκαθορισμένης μικρής τιμής.

B) Εάν η σύγκλιση έχει επιτευχθεί ή ο αριθμός των επαναλήψεων έχει ξεπεράσει μια μέγιστη τιμή τερματισμός του προγράμματος. Αλλιώς επιστροφή στο βήμα (B).

PDEs – Model Problem (Laplace)

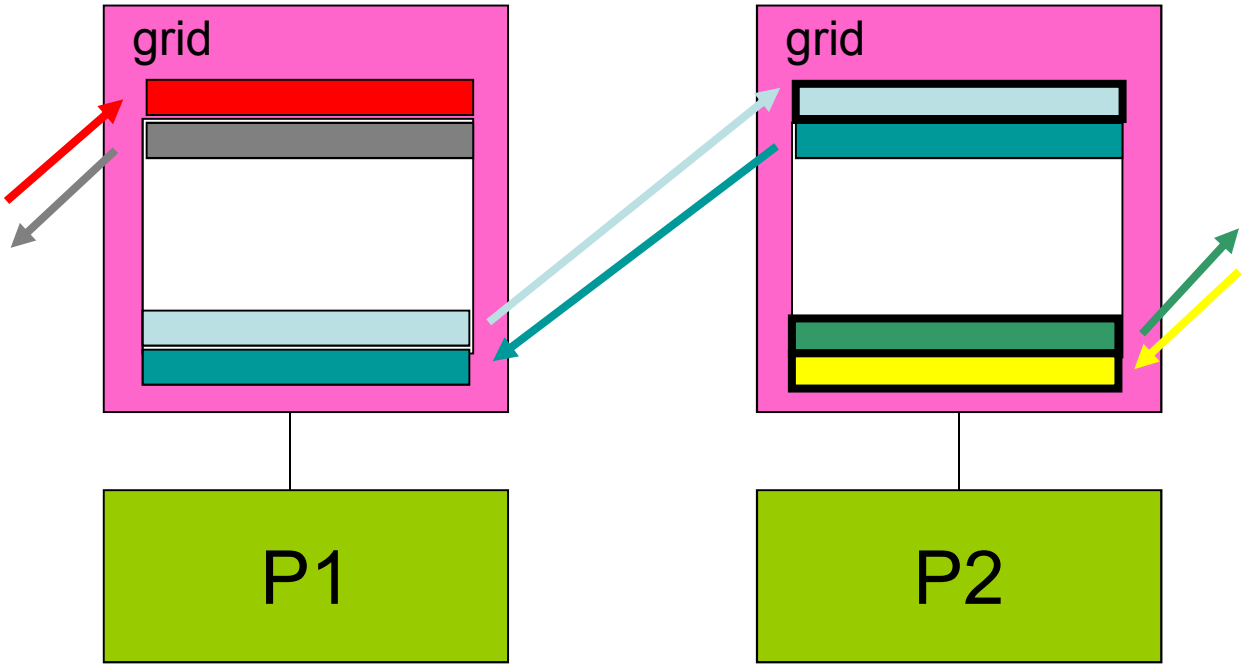
SOR: Serial Implementation

```
main(int argc, char *argv[])
{
      /* Initialize grid*/
      ….
     for (k==1; k<maxiter; k++) {
     ….
     /* Computation */
      for( i==1; i<N; i++ ) {
        for( j==1; j<N; j++ ) {
         U(i,j) = (1-w)*Uold(i,j) + w*(U(i-1,j) + Uold(i+1,j) + U(i,j-1) + Uold(i,j+1));
        }
      }
     …
     /* Check convergence */
      sum = 0
      for( i==1; i<N; i++ ) {
        for( j==1; j<N; j++ ) { sum = sum + (U(i,j)-Uold(i,j))**2; }
      }
       if (sqrt(sum) < TOL) exit(1);
       Uold  = U;
     }
}
```

PDEs – Parallel Implementation

Διαχωρισμός του grid:

➢**Επικοινωνία δεδομένων:** κάθε επεξεργαστής στέλνει/λαμβάνει δεδομένα από τον προηγούμενο και τον επόμενο.



PDEs – Parallel Implementation

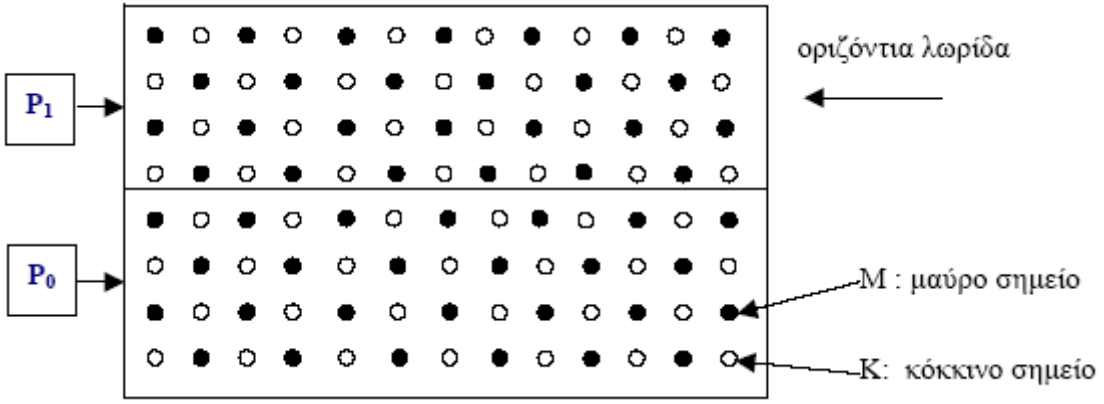
➢**Επικοινωνία Δεδομένων:** κάθε επεξεργαστής στέλνει/λαμβάνει δεδομένα από τον προηγούμενο και τον επόμενο.

```
if (myrank != 0) {
    MPI_Send (grid[from], n, MPI_DOUBLE, myrank-1, tag, MPI_COMM_WORLD);
    MPI_Recv (grid[from-1], n, MPI_DOUBLE,myrank-1, tag, MPI_COMM_WORLD, &status);
}

if (myrank != p-1) {
    MPI_Send (grid[to-1],   n, MPI_DOUBLE, myrank+1, tag,  MPI_COMM_WORLD);
    MPI_Recv (grid[to], n, MPI_DOUBLE,  myrank+1, tag, MPI_COMM_WORLD, &status);
}
```

Παράλληλη Εφαρμογή της SOR: Κόκκινο/Μαύρο (Red/Black) SOR

➢Το πλέγμα διαχωρίζεται επιπλέον σε κόκκινα και μαύρα σημεία, π.χ. μονά σημεία: «κόκκινα», ζυγά σημεία: «μαύρα».



➢**Κόκκινα σημεία:**

$$U_{i,j}^{(k+1)} = \frac{w}{4}\left(U_{i+1,j}^{(k)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k)}\right) + (1-w)U_{i,j}^{(k)}$$

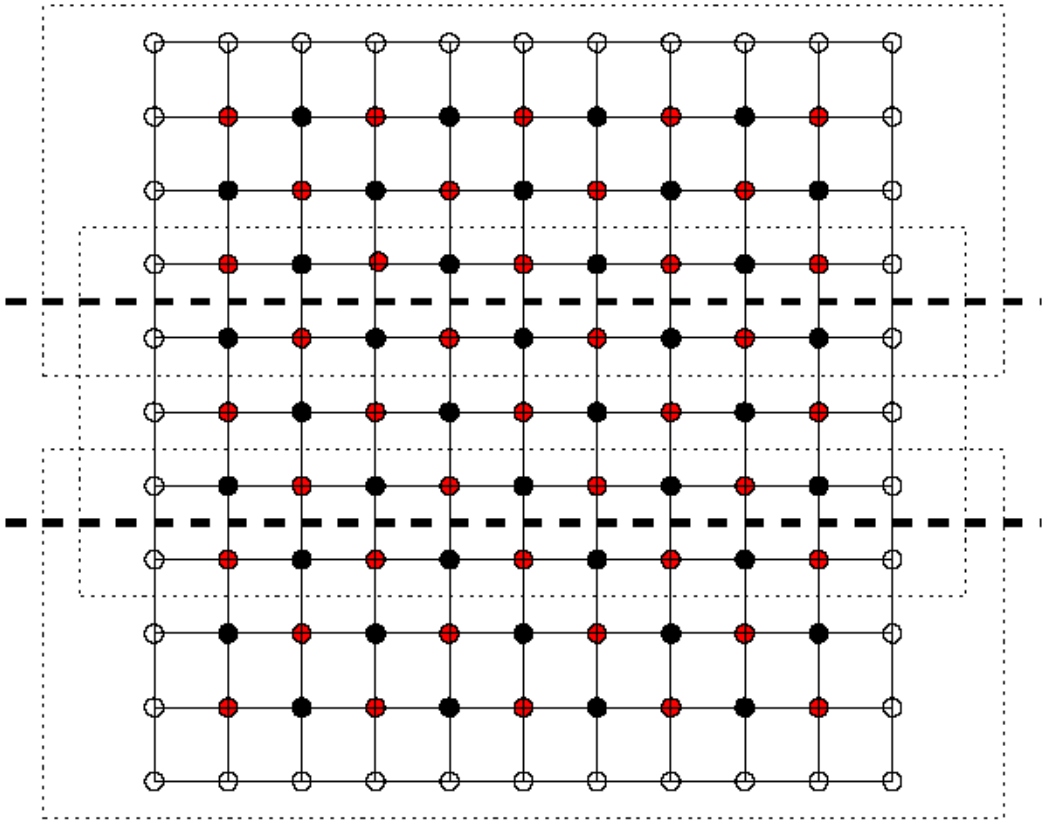
➢**Μαύρα σημεία:**

$$U_{i,j}^{(k+1)} = \frac{w}{4}\left(U_{i+1,j}^{(k+1)} + U_{i-1,j}^{(k+1)} + U_{i,j+1}^{(k+1)} + U_{i,j-1}^{(k+1)}\right) + (1-w)U_{i,j}^{(k)}$$

PDEs – Parallel Implementation

Red/Black SOR: Παραλληλίσιμος Αλγόριθμος

Έστω N εξισώσεις σε m=1,2,…P επεξεργαστές. Στον 1-D διαχωρισμό του grid έχουμε N/P γραμμές σε κάθε επεξεργαστή:



PDEs – Parallel Implementation

Red/Black SOR: Παραλληλίσιμος Αλγόριθμος

Ο R/B SOR αλγόριθμος περιλαμβάνει τα παρακάτω βήματα:

A) Αρχικοποίηση - Διαχωρισμός του grid: Δίνεται μια αρχική τιμή του U στις N/P γραμμές κάθε επεξεργαστή.

B) Ξεκίνημα της επαναληπτικής διαδικασίας. Παράλληλος υπολογισμός της καινούριας εκτίμησης του $U(i,j)$.

C) Έλεγχος της σύγκλισης: κάθε επεξεργαστής υπολογίζει την Ευκλείδια νόρμα των στοιχείων του. Τα αθροίσματα μεταφέρονται στον κεντρικό (master) επεξεργαστή όπου υπολογίζεται το συνολικό άθροισμα και γίνεται ο έλεγχος της σύγκλισης.

D) Εάν η σύγκλιση έχει επιτευχθεί ή ο αριθμός των επαναλήψεων έχει ξεπεράσει μια μέγιστη τιμή τερματισμός του προγράμματος. Αλλιώς επιστροφή στο βήμα (B).

PDEs – Parallel Implementation

Red/Black SOR: Παραλληλίσιμος αλγόριθμος

Βήμα (Β) – Παράλληλος Υπολογισμός της Καινούριας Εκτίμησης του *U*:

- Υπολογίζουμε όλα τα κόκκινα *U* σημεία της *k*+1 επανάληψης: $U_{i,j}^{(k+1)}$

- Στέλνουμε τα κόκκινα σημεία της γραμμής 1 στον X-1 επεξεργαστή για *m*=2,3,…*P*.
- Στέλνουμε τα κόκκινα σημεία της γραμμής *N*/*P* στον X+1 επεξεργαστή για *m*=1,2,…*P*-1.

- Υπολογίζουμε όλα τα **μαύρα *U* σημεία της *k*+1 επανάληψης**: $U_{i,j}^{(k+1)}$

- Στέλνουμε τα **μαύρα** σημεία της γραμμής 1 στον X-1 επεξεργαστή για *m*=2,3,…*P*.
- Στέλνουμε τα **μαύρα** σημεία της γραμμής *N*/*P* στον X+1 επεξεργαστή για *m*=1,2,…*P*-1.

PDEs – Parallel Implementation

Red/Black SOR: Κριτήριο Σύγκλισης σε Παράλληλο Αλγόριθμο

- Χρήση πράξεων περιστολής, π.χ. MPI_Reduce.

```
…

/* Check convergence */
my_sum = 0
for( i==1; i<N/P; i++ ) {
  for( j==1; j<N; j++ ) { my_sum = my_sum + (U(i,j)-Uold(i,j))**2; }
}
MPI_Reduce(&my_sum, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD); if

if (rank==0) {
  if (sqrt(sum) < TOL) exit(1); }

...
}
```

Βιβλιογραφία

- *Parallel Programming,* B. Wilkinson, M. Allen, Prentice Hall, 2nd Ed. 2005.
- *Introduction to Parallel Computing*, A. Grama, A. Gupta, G. Karypis, V. Kumar, Addison-Wesley, 2003.
- *Parallel Computing: Theory and Practice*, M. J. Quinn, McGraw-Hill, 1994.
- *Designing and Building Parallel Programs*, Ian Foster, Addison-Wesley 1994.