

ΜSc-CNT: Τεχνολογίες Υπολογισμού & Δικτύων ΠΑΡΑΛΛΗΛΑ & ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Άσκηση 2η - Β' Εξάμηνο 2016-17

Αθανάσιος Ροκόπουλος (16009) – Μιχαήλ Γαλλιάκης (16003)
7/6/2017

Εκφώνηση :

A.

Θεωρώντας ένα παράλληλο περιβάλλον '**p**' επεξεργαστών, σας ζητείται να γράψετε **MPI** πρόγραμμα σε **C** το οποίο να υλοποιεί τον πολλαπλασιασμό $C=A \times B$ δύο πινάκων **A** και **B** διάστασης $N \times N$, χρησιμοποιώντας τον αλγόριθμο κατανομής κατά γραμμές (**row-based**). Αρχικά τα στοιχεία των δύο πινάκων θεωρείστε ότι τα διαβάζει (είτε από την οθόνη είτε από αρχείο) ο επεξεργαστής '0'. Στο τέλος ο επεξεργαστής '0' θα πρέπει επίσης να συγκεντρώνει (μαζεύει) τον πίνακα αποτέλεσμα (**C**) και να τον εμφανίζει στην οθόνη. Για τη λεπτομερή περιγραφή του row-based αλγόριθμου μπορείτε να συμβουλευτείτε τις διαφάνειες #9-#13 του αρχείου **MSc-CNT-Lecture#8.ppt** (Μάθημα #8) το οποίο έχει αναρτηθεί στο Eclass. Αναπτύξτε τον κώδικά σας παραμετρικά έτσι ώστε να συμπεριφέρεται σωστά για οποιονδήποτε αριθμό πολλαπλών επεξεργαστών '**p**' – θεωρώντας ότι το '**N**' είναι ακέραιο πολλαπλάσιο του '**p**'.

B.

Θεωρώντας ένα παράλληλο περιβάλλον '**p**' επεξεργαστών, σας ζητείται να γράψετε **MPI** πρόγραμμα σε **C** το οποίο να υλοποιεί τη μέθοδο **Jacobi**, για την επίλυση ενός γραμμικού συστήματος $Ax=b$ (δοθέντος ενός πίνακα συντελεστών A $n \times n$, και ενός διανύσματος σταθερών b $n \times 1$). Για τη λεπτομερή περιγραφή της μεθόδου μπορείτε να συμβουλευτείτε τις διαφάνειες #14-#19 του αρχείου **MSc-CNT-Lecture#9.ppt** (Μάθημα #9) το οποίο έχει αναρτηθεί στο Eclass. Ο χρήστης θα δίνει σαν είσοδο τον πίνακα **A**, το διάνυσμα **b** καθώς και το διάνυσμα αρχικών τιμών $x(0)$. Επίσης θα παρέχει και τα όρια ξ (για τον έλεγχο σύγκλισης) και λ (για το μέγιστο αριθμό επαναλήψεων). Το πρόγραμμα θα επιστρέφει το διάνυσμα x (λύση του συστήματος) ή θα ανακοινώνει στο χρήστη ότι η μέθοδος δεν συγκλίνει. Αναπτύξτε τον κώδικά σας παραμετρικά έτσι ώστε να συμπεριφέρεται σωστά για οποιονδήποτε αριθμό πολλαπλών επεξεργαστών '**p**' – θεωρώντας ότι το '**n**' είναι ακέραιο πολλαπλάσιο του '**p**'.

**Παραδοτέα: κώδικας, σχολιασμός/τεκμηρίωση, ενδεικτικά
τρεξίματα/αποτελέσματα**

Μέρος Α:

Μέσα στο φάκελο erot1, που βρίσκεται μέσα στο zip αρχείο μαζί με αυτό το pdf, θα βρείτε τα ακόλουθα αρχεία, που αφορούν το πρώτο ερώτημα [Τα ίδια αρχεία υπάρχουν και στο rc9...]:

- erot1.0.c (Πρώτη προσέγγιση με στατικούς 2D πίνακες [για $N=p$])
- erot1.1.c (Δεύτερη προσέγγιση με δείκτες [για $N=p$])
- erot1.2.c (Ολοκληρωμένη λύση της εκφώνησης (μαζί με σχόλια) για N πολλαπλάσιο του p)

Input matrix A:

```
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
```

Input matrix B:

```
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
```

Matrix product A*B

```
14 20 26 32
20 30 40 50
26 40 54 68
32 50 68 86
```

Screenshots με αποτελέσματα:

$N=4$ & $p=4$

```
[msc16003@mpi9 erot1]$ mpiexec -n 4 erot1.2
[0] [1] [2] [3]
[1] [2] [3] [4]
[2] [3] [4] [5]
[3] [4] [5] [6]
Result:
[14] [20] [26] [32]
[20] [30] [40] [50]
[26] [40] [54] [68]
[32] [50] [68] [86]
[msc16003@mpi9 erot1]$ _
```

Input matrix A:

```
0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11 12
6 7 8 9 10 11 12 13
7 8 9 10 11 12 13 14
```

Input matrix B:

```
0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11 12
6 7 8 9 10 11 12 13
7 8 9 10 11 12 13 14
```

Matrix product A*B

```
140 168 196 224 252 280 308 336
168 204 240 276 312 348 384 420
196 240 284 328 372 416 460 504
224 276 328 380 432 484 536 588
252 312 372 432 492 552 612 672
280 348 416 484 552 620 688 756
308 384 460 536 612 688 764 840
336 420 504 588 672 756 840 924
```

$N=8$ & $p=4$

```
[msc16003@mpi9 erot1]$ mpiexec -n 4 erot1.2
[0] [1] [2] [3] [4] [5] [6] [7]
[1] [2] [3] [4] [5] [6] [7] [8]
[2] [3] [4] [5] [6] [7] [8] [9]
[3] [4] [5] [6] [7] [8] [9] [10]
[4] [5] [6] [7] [8] [9] [10] [11]
[5] [6] [7] [8] [9] [10] [11] [12]
[6] [7] [8] [9] [10] [11] [12] [13]
[7] [8] [9] [10] [11] [12] [13] [14]
Result:
[140] [168] [196] [224] [252] [280] [308] [336]
[168] [204] [240] [276] [312] [348] [384] [420]
[196] [240] [284] [328] [372] [416] [460] [504]
[224] [276] [328] [380] [432] [484] [536] [588]
[252] [312] [372] [432] [492] [552] [612] [672]
[280] [348] [416] [484] [552] [620] [688] [756]
[308] [384] [460] [536] [612] [688] [764] [840]
[336] [420] [504] [588] [672] [756] [840] [924]
[msc16003@mpi9 erot1]$ _
```

N=12 & p=4

```
0 1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11 12
2 3 4 5 6 7 8 9 10 11 12 13
3 4 5 6 7 8 9 10 11 12 13 14
4 5 6 7 8 9 10 11 12 13 14 15
5 6 7 8 9 10 11 12 13 14 15 16
6 7 8 9 10 11 12 13 14 15 16 17
7 8 9 10 11 12 13 14 15 16 17 18
8 9 10 11 12 13 14 15 16 17 18 19
9 10 11 12 13 14 15 16 17 18 19 20
10 11 12 13 14 15 16 17 18 19 20 21
11 12 13 14 15 16 17 18 19 20 21 22
```

```
[msc16003@mpi9 erot1]$ mpiexec -n 4 erot1.2
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
[2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]
[3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14]
[4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]
[5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16]
[6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17]
[7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18]
[8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19]
[9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20]
[10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21]
[11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22]
Result:
[506] [572] [638] [704] [770] [836] [902] [968] [1034] [1100] [1166] [1232]
[572] [650] [728] [806] [884] [962] [1040] [1118] [1196] [1274] [1352] [1430]
[638] [728] [818] [908] [998] [1088] [1178] [1268] [1358] [1448] [1538] [1628]
[704] [806] [908] [1010] [1112] [1214] [1316] [1418] [1520] [1622] [1724] [1826]
[770] [884] [998] [1112] [1226] [1340] [1454] [1568] [1682] [1796] [1910] [2024]
[836] [962] [1088] [1214] [1340] [1466] [1592] [1718] [1844] [1970] [2096] [2222]
[902] [1040] [1178] [1316] [1454] [1592] [1730] [1868] [2006] [2144] [2282] [2420]
[968] [1118] [1268] [1418] [1568] [1718] [1868] [2018] [2168] [2318] [2468] [2618]
[1034] [1196] [1358] [1520] [1682] [1844] [2006] [2168] [2330] [2492] [2654] [2816]
[1100] [1274] [1448] [1622] [1796] [1970] [2144] [2318] [2492] [2666] [2840] [3014]
[1166] [1352] [1538] [1724] [1910] [2096] [2282] [2468] [2654] [2840] [3026] [3212]
[1232] [1430] [1628] [1826] [2024] [2222] [2420] [2618] [2816] [3014] [3212] [3410]
[msc16003@mpi9 erot1]$
```

Input matrix B:

```
0 1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11 12
2 3 4 5 6 7 8 9 10 11 12 13
3 4 5 6 7 8 9 10 11 12 13 14
4 5 6 7 8 9 10 11 12 13 14 15
5 6 7 8 9 10 11 12 13 14 15 16
6 7 8 9 10 11 12 13 14 15 16 17
7 8 9 10 11 12 13 14 15 16 17 18
8 9 10 11 12 13 14 15 16 17 18 19
9 10 11 12 13 14 15 16 17 18 19 20
10 11 12 13 14 15 16 17 18 19 20 21
11 12 13 14 15 16 17 18 19 20 21 22
```

Matrix product A*B

```
506 572 638 704 770 836 902 968 1034 1100 1166 1232
572 650 728 806 884 962 1040 1118 1196 1274 1352 1430
638 728 818 908 998 1088 1178 1268 1358 1448 1538 1628
704 806 908 1010 1112 1214 1316 1418 1520 1622 1724 1826
770 884 998 1112 1226 1340 1454 1568 1682 1796 1910 2024
836 962 1088 1214 1340 1466 1592 1718 1844 1970 2096 2222
902 1040 1178 1316 1454 1592 1730 1868 2006 2144 2282 2420
968 1118 1268 1418 1568 1718 1868 2018 2168 2318 2468 2618
1034 1196 1358 1520 1682 1844 2006 2168 2330 2492 2654 2816
1100 1274 1448 1622 1796 1970 2144 2318 2492 2666 2840 3014
1166 1352 1538 1724 1910 2096 2282 2468 2654 2840 3026 3212
1232 1430 1628 1826 2024 2222 2420 2618 2816 3014 3212 3410
```

N=12 & p=2

```
[msc16003@mpi9 erot1]$ mpiexec -n 2 erot1.2
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
[2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]
[3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14]
[4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]
[5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16]
[6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17]
[7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18]
[8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19]
[9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20]
[10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21]
[11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22]
Result:
[506] [572] [638] [704] [770] [836] [902] [968] [1034] [1100] [1166] [1232]
[572] [650] [728] [806] [884] [962] [1040] [1118] [1196] [1274] [1352] [1430]
[638] [728] [818] [908] [998] [1088] [1178] [1268] [1358] [1448] [1538] [1628]
[704] [806] [908] [1010] [1112] [1214] [1316] [1418] [1520] [1622] [1724] [1826]
[770] [884] [998] [1112] [1226] [1340] [1454] [1568] [1682] [1796] [1910] [2024]
[836] [962] [1088] [1214] [1340] [1466] [1592] [1718] [1844] [1970] [2096] [2222]
[902] [1040] [1178] [1316] [1454] [1592] [1730] [1868] [2006] [2144] [2282] [2420]
[968] [1118] [1268] [1418] [1568] [1718] [1868] [2018] [2168] [2318] [2468] [2618]
[1034] [1196] [1358] [1520] [1682] [1844] [2006] [2168] [2330] [2492] [2654] [2816]
[1100] [1274] [1448] [1622] [1796] [1970] [2144] [2318] [2492] [2666] [2840] [3014]
[1166] [1352] [1538] [1724] [1910] [2096] [2282] [2468] [2654] [2840] [3026] [3212]
[1232] [1430] [1628] [1826] [2024] [2222] [2420] [2618] [2816] [3014] [3212] [3410]
[msc16003@mpi9 erot1]$
```

Παρατηρήσεις:

- Το output δείχνει στην αρχή τον πίνακα A (ο οποίος είναι ίδιος με τον πίνακα B) και σαν αποτέλεσμα, εμφανίζει τον πίνακα C.
- Στα δύο τελευταία screenshots φαίνεται η περίπτωση που λύνουν το “πρόβλημα” 4 και 2 υπολογιστές (Όπου N(12) είναι ακέραιο πολλαπλάσιο του p (2 & 4 αντίστοιχα)...

Ακολουθούν 2 screenshots από την ενδεικτική λύση (erot1.2.c):

```
erot1.2.c  ✕
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#define N 12 //Σταθερά N για τις διαστάσεις των πινάκων.(Το N πρέπει να είναι πολλαπλάσιο του p)
#define ROOT 0 //Σταθερή τιμή για το rank του 0 ώστε να αναφερόμαστε σε αυτόν με το όνομα ROOT
#define TAG1 100 //Σταθερή τιμή για το TAG που χρειάζεται για τα send-receive...
int main(int argc, char** argv)
{
    //Δηλώνουμε τους 3 πίνακες:
    int *matrixA ;
    int *matrixB ;
    int *matrixC ;

    int rank,size,root;
    //Αρχικοποιείται το MPI
    MPI_Init(&argc, &argv);
    //Παίρνει ο κάθε επεξεργαστής-υπολογιστής το rank του
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //Παίρνει ο κάθε επεξεργαστής-υπολογιστής το πλήθος των επεξεργαστών.
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    //Δηλώνουμε πιο rank έχει ο τελευταίος επεξεργαστής στην σειρά.
    const int LAST = size-1 ;
    MPI_Status status ;
    int k,j,i ;
    int rowIndex ;

    if (rank == ROOT) //0 "0" μόνο
    {
        //Δημιουργεί 3 πίνακες N x N
        matrixA = malloc(N * N * sizeof(int)) ;
        matrixB = malloc(N * N * sizeof(int)) ;
        matrixC = malloc(N * N * sizeof(int)) ;
        //Και στην συνέχεια τους καταχωρεί τιμές, της μορφής: //0,1,2,3
        for (k=0; k<N; k++) //1,2,3,4
        { //2,3,4,5
            //3,4,5,6
            rowIndex = k*N ;
            for (j=0; j<N; j++) {
                matrixA[rowIndex+j] = k+j;
                matrixB[rowIndex+j] = k+j;
                //matrixC[rowIndex+j] = 0 ;
                printf("[%d] ", matrixA[rowIndex+j]);
                //printf("mB[%d][%d]=%d | ", k,j, matrixB[rowIndex+j]);
            }
            printf("\n");
        }
    }

    int quota = N/size ; //Υπολογίζεται το μερίδιο που θα έχει ο κάθε επεξεργαστής
    int quotaCrowd = quota*N ; //Υπολογίζεται το πλήθος αριθμών που θα έχει ο κάθε επεξεργαστής
    int q,w ;

    //Δημιουργεί κάθε επεξεργαστής 3 πίνακες μεγέθους quota X N
    int *matrixA_quotaRows = malloc(quotaCrowd*sizeof(int)) ;
    int *matrixB_quotaRows = malloc(quotaCrowd*sizeof(int)) ;
    int *matrixC_quotaRows = malloc(quotaCrowd*sizeof(int)) ;

    //0 "0" μοιράζει τα περιεχόμενα του matrixA και matrixB στους τοπικούς πίνακες του κάθε επεξεργαστή...
    MPI_Scatter(matrixA, quotaCrowd, MPI_INT, matrixA_quotaRows, quotaCrowd, MPI_INT, ROOT, MPI_COMM_WORLD);
    MPI_Scatter(matrixB, quotaCrowd, MPI_INT, matrixB_quotaRows, quotaCrowd, MPI_INT, ROOT, MPI_COMM_WORLD);
    int rowIndexAandC,rowIndexB ;
```

Συνέχεια:

```
//δημιουργεί κάθε επεξεργαστής 3 πίνακες μεγέθους quota * N
int *matrixA_quotaRows = malloc(quotaCrowd*sizeof(int)) ;
int *matrixB_quotaRows = malloc(quotaCrowd*sizeof(int)) ;
int *matrixC_quotaRows = malloc(quotaCrowd*sizeof(int)) ;

//Ο "0" μοιράζει τα περιεχόμενα του matrixA και matrixB στους τοπικούς πίνακες του κάθε επεξεργαστή...
MPI_Scatter(matrixA, quotaCrowd, MPI_INT, matrixA_quotaRows, quotaCrowd, MPI_INT, ROOT, MPI_COMM_WORLD);
MPI_Scatter(matrixB, quotaCrowd, MPI_INT, matrixB_quotaRows, quotaCrowd, MPI_INT, ROOT, MPI_COMM_WORLD);
int rowIndexAandC, rowIndexB ;

//Με βάση τον αλγόριθμο, ο κάθε επεξεργαστής υπολογίζει τις τιμές του δικού του τοπικού (υπο) πίνακα C[matrixC_quotaRows]
for(k=0;k<quota;k++){
    rowIndexB = k*N ;
    for(j=0;j<quota;j++){
        rowIndexAandC = j*N ;
        for(i=0;i<N;i++){
            matrixC_quotaRows[rowIndexAandC+i] += matrixA_quotaRows[rowIndexAandC+k+(rank*quota)] * matrixB_quotaRows[rowIndexB+i] ;
            //printf("%d[%d] ",rank,matrixC_quotaRows[rowIndexAandC+i]);
        }
        //printf("\n") ;
    }
}

for(w=1;w<size;w++) //Για όσο είναι το πλήθος των επεξεργαστών -1 (Γιατί έγινε στο προηγούμενο βήμα...)
{
    //Στέλνει ο κάθε επεξεργαστής στον προηγούμενο του γείτονα και παραλαμβάνει από τον επόμενο του γείτονα
    //(Τοπολογία δακτυλίου), το περιεχόμενο των γραμμών που αναλογούν σε κάθε επεξεργαστή από τον πίνακα B (με βάση τον row-based).
    MPI_Send(matrixB_quotaRows,quotaCrowd,MPI_INT, (rank>0)?(rank-1):LAST,TAG1, MPI_COMM_WORLD);
    MPI_Recv(matrixB_quotaRows,quotaCrowd,MPI_INT, (rank+1)%size,TAG1,MPI_COMM_WORLD,&status);
    //Με βάση τον αλγόριθμο, ο κάθε επεξεργαστής υπολογίζει τις τιμές του δικού του τοπικού (υπο) πίνακα C[matrixC_quotaRows]
    //με βάση τις νέες τιμές που έχει μόλις πάρει από τον γείτονα του όπου αφορούν τον πίνακα B.
    for(k=0;k<quota;k++){
        rowIndexB = k*N ;
        for(j=0;j<quota;j++){
            rowIndexAandC = j*N ;
            for(i=0;i<N;i++){
                matrixC_quotaRows[rowIndexAandC+i] += matrixA_quotaRows[rowIndexAandC+k+((rank*quota)+(w*quota))%N] * matrixB_quotaRows[rowIndexB+i] ;
                //printf("%d[%d] ",rank,k+((rank*quota)+(w*quota))%N);
            }
            //printf("\n") ;
        }
    }
}

//Εφόσον έχουν τελειώσει όλα τα προηγούμενα βήματα, ο "0" μαζεύει τα περιεχόμενα όλων των (υπο) πινάκων C[matrixC_quotaRows] του κάθε
//επεξεργαστή στον δικό του πίνακα C.
MPI_Gather(matrixC_quotaRows, quotaCrowd, MPI_INT,matrixC, quotaCrowd, MPI_INT, ROOT, MPI_COMM_WORLD);

if (rank == ROOT) ////Ο "0" μόνο
{
    printf("Result:\n"); //Εμφανίζει τα αποτελέσματα.
    for (k=0; k<N; k++) { //Δηλαδή τα περιεχόμενα του πίνακα C, ο οποίος έχει τις υπολογισμένες
        rowIndex = k*N ; // (παράλληλα) τιμές από τον πολλαπλασιασμό του πίνακα A με τον πίνακα B.
        for (j=0; j<N; j++)
            printf("[%d] ",matrixC[rowIndex+j]);
        printf("\n") ;
    }
}

MPI_Finalize(); //Τερματίζει το MPI
return 0 ;
}
```

- Θα μπορούσαμε να διαβάζουμε από αρχείο τους όποιους πίνακες, αλλά λόγω του ότι έχουμε και άλλες εργασίες, μας απασχόλησε κυρίως το καθαρά κομμάτι του αλγορίθμου (row-based)...
- Επίσης, η υλοποίηση είναι ενδεικτική και έγινε με βάση τα βήματα του αλγορίθμου όπως αναφέρονται στις διαφάνειες. Που σημαίνει, ότι ενώ λειτουργεί σωστά, θα μπορούσε πιθανόν να γραφτεί καλύτερα ο κώδικας (Ισχύει και εδώ το γεγονός ότι “τρέχουν” και άλλες εργασίες)...

Μέρος Β:

Μέσα στο φάκελο erot2, που βρίσκεται μέσα στο zip αρχείο μαζί με αυτό το pdf, θα βρείτε τα ακόλουθα αρχεία, που αφορούν το πρώτο ερώτημα [Τα ίδια αρχεία υπάρχουν και στο rc9...]:

- erot2.c (Ολοκληρωμένη λύση της εκφώνησης (μαζί με σχόλια) για N πολλαπλάσιο του p)
- jacobiInput.txt (Input παράδειγμα που συγκλίνει με την μέθοδο Jacobi)
- jacobiInput2.txt (Input παράδειγμα που συγκλίνει με την μέθοδο Jacobi)
- jacobiInput3.txt (Input παράδειγμα που δεν συγκλίνει με την μέθοδο Jacobi)

Screenshots με output του προγράμματος:

Input το jacobiInput.txt

```
[mscl6003@mpi9 erot2]$ cat jacobiInput.txt
4
20
0.000001
10 -1 2 0
-1 11 -1 3
2 -1 10 -1
0 3 -1 8
6 25 -11 15
0 0 0 0
```

Another example [\[edit \]](#)

Suppose we are given the following linear system:

$$\begin{aligned}10x_1 - x_2 + 2x_3 &= 6, \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25, \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11, \\ 3x_2 - x_3 + 8x_4 &= 15.\end{aligned}$$

If we choose (0, 0, 0, 0) as the initial approximation, the

$$\begin{aligned}x_1 &= (6 + 0 - 0)/10 = 0.6, \\ x_2 &= (25 - 0 - 0)/11 = 25/11 = 2.2727, \\ x_3 &= (-11 - 0 - 0)/10 = -1.1, \\ x_4 &= (15 - 0 - 0)/8 = 1.875.\end{aligned}$$

Using the approximations obtained, the iterative process

x_1	x_2	x_3	x_4
0.6	2.2727	-1.1	1.875
1.04727	1.7159	-0.80522	0.88522
0.93263	2.05330	-1.0493	1.13088
1.01519	1.95369	-0.9681	0.97384
0.98899	2.0114	-1.0102	1.02135

The exact solution of the system is (1, 2, -1, 1).

```
[mscl6003@mpi9 erot2]$ mpiexec -n 4 erot2.2
Βρέθηκε το αρχείο και ξεκινάμε να το διαβάζουμε!
n=4, l=20, ex=0.00000100
* * * Matrix A * * *
[10.000000] [-1.000000] [2.000000] [0.000000]
[-1.000000] [11.000000] [-1.000000] [3.000000]
[2.000000] [-1.000000] [10.000000] [-1.000000]
[0.000000] [3.000000] [-1.000000] [8.000000]
* * * Vector b * * *
[6.000000] [25.000000] [-11.000000] [15.000000]
* * * Vector X * * *
[0.000000] [0.000000] [0.000000] [0.000000]
Τέλος αρχικοποίησης
k=0, Rank=0, Norma_loc=0.36000001, Norma_all=3.20170474, Ex=0.00000100
k=0, Rank=0 [0.600000] [2.272727] [-1.100000] [1.875000]
k=1, Rank=0, Norma_loc=0.20005283, Norma_all=1.25564337, Ex=0.00000100
k=1, Rank=0 [1.047273] [1.715909] [-0.805227] [0.885227]
k=2, Rank=0, Norma_loc=0.01314148, Norma_all=0.49690536, Ex=0.00000100
k=2, Rank=0 [0.932636] [2.053306] [-1.049341] [1.130881]
k=3, Rank=0, Norma_loc=0.00681654, Norma_all=0.21908499, Ex=0.00000100
k=3, Rank=0 [1.015199] [1.953696] [-0.968109] [0.973843]
k=4, Rank=0, Norma_loc=0.00068682, Norma_all=0.08974510, Ex=0.00000100
k=4, Rank=0 [0.988991] [2.011415] [-1.010286] [1.021351]
k=5, Rank=0, Norma_loc=0.00020185, Norma_all=0.03927436, Ex=0.00000100
k=5, Rank=0 [1.003199] [1.992241] [-0.994522] [0.994434]
k=6, Rank=0, Norma_loc=0.00002571, Norma_all=0.01632333, Ex=0.00000100
k=6, Rank=0 [0.998129] [2.002307] [-1.001972] [1.003594]
k=7, Rank=0, Norma_loc=0.00000623, Norma_all=0.00708720, Ex=0.00000100
k=7, Rank=0 [1.000625] [1.998670] [-0.999036] [0.998888]
k=8, Rank=0, Norma_loc=0.00000090, Norma_all=0.00297279, Ex=0.00000100
k=8, Rank=0 [0.999674] [2.000448] [-1.000369] [1.000619]
k=9, Rank=0, Norma_loc=0.00000020, Norma_all=0.00128318, Ex=0.00000100
k=9, Rank=0 [1.000119] [1.999768] [-0.999828] [0.999786]
k=10, Rank=0, Norma_loc=0.00000003, Norma_all=0.00054142, Ex=0.00000100
k=10, Rank=0 [0.999942] [2.000085] [-1.000068] [1.000108]
k=11, Rank=0, Norma_loc=0.00000001, Norma_all=0.00023273, Ex=0.00000100
k=11, Rank=0 [1.000022] [1.999959] [-0.999969] [0.999960]
k=12, Rank=0, Norma_loc=0.00000000, Norma_all=0.00009858, Ex=0.00000100
k=12, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=13, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=13, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=14, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=14, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=15, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=15, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=16, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=16, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=17, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=17, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=18, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=18, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
Υπάρχει σύγκλιση! και η λύση του συστήματος είναι:
[1.000000] [2.000000] [-1.000000] [1.000000]
```

```
k=10, Rank=0, Norma_loc=0.00000003, Norma_all=0.00054142, Ex=0.00000100
k=10, Rank=0 [0.999942] [2.000085] [-1.000068] [1.000108]
k=11, Rank=0, Norma_loc=0.00000001, Norma_all=0.00023273, Ex=0.00000100
k=11, Rank=0 [1.000022] [1.999959] [-0.999969] [0.999960]
k=12, Rank=0, Norma_loc=0.00000000, Norma_all=0.00009858, Ex=0.00000100
k=12, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=13, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=13, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=14, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=14, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=15, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=15, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=16, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=16, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=17, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=17, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
k=18, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000000, Ex=0.00000100
k=18, Rank=0 [1.000000] [2.000000] [-1.000000] [1.000000]
Υπάρχει σύγκλιση! και η λύση του συστήματος είναι:
[1.000000] [2.000000] [-1.000000] [1.000000]
```

Input to jacobiInput2.txt

```
[msc16003@mpi9 erot2]$ cat jacobiInput2.txt
3
20
0.000005
8 1 1
1 8 1
1 1 8
10 10 10
0 0 0
```

Διακρίνεται (με πράσινο) ότι πρέπει το n να είναι ακέραιο πολλαπλάσιο του p όπως ορίζει η εκφώνηση!

Input to jacobiInput3.txt (Δεν συγκλίνει)

```
[msc16003@mpi9 erot2]$ cat jacobiInput3.txt
3
30
0.000001
8 2 5
2 2 3
1 4 5
2 6 3
0 0 0
```

```
[msc16003@mpi9 erot2]$ mpiexec -n 3 erot2.2
Βρέθηκε το αρχείο και ξεκινάμε να το διαβάσουμε!
n=3, l=20, ex=0.00000500
* * * Matrix A * * *
[8.000000][1.000000][1.000000]
[1.000000][8.000000][1.000000]
[1.000000][1.000000][8.000000]
* * * Vector b * * *
[10.000000][10.000000][10.000000]
* * * Vector X * * *
[0.000000][0.000000][0.000000]
```

```
Τέλος αρχικοποίησης
k=0, Rank=0, Norma_loc=1.56250000, Norma_all=2.16506362, Ex=0.00000500
k=0, Rank=0[1.250000] [1.250000] [1.250000]
k=1, Rank=0, Norma_loc=0.09765625, Norma_all=0.54126590, Ex=0.00000500
k=1, Rank=0[0.937500] [0.937500] [0.937500]
k=2, Rank=0, Norma_loc=0.00610352, Norma_all=0.13531648, Ex=0.00000500
k=2, Rank=0[1.015625] [1.015625] [1.015625]
k=3, Rank=0, Norma_loc=0.00038147, Norma_all=0.03382912, Ex=0.00000500
k=3, Rank=0[0.996094] [0.996094] [0.996094]
k=4, Rank=0, Norma_loc=0.00002384, Norma_all=0.00845728, Ex=0.00000500
k=4, Rank=0[1.000977] [1.000977] [1.000977]
k=5, Rank=0, Norma_loc=0.00000149, Norma_all=0.00211432, Ex=0.00000500
k=5, Rank=0[0.999756] [0.999756] [0.999756]
k=6, Rank=0, Norma_loc=0.00000009, Norma_all=0.00052858, Ex=0.00000500
k=6, Rank=0[1.000061] [1.000061] [1.000061]
k=7, Rank=0, Norma_loc=0.00000001, Norma_all=0.00013214, Ex=0.00000500
k=7, Rank=0[0.999985] [0.999985] [0.999985]
k=8, Rank=0, Norma_loc=0.00000000, Norma_all=0.00003304, Ex=0.00000500
k=8, Rank=0[1.000004] [1.000004] [1.000004]
k=9, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000826, Ex=0.00000500
k=9, Rank=0[0.999999] [0.999999] [0.999999]
k=10, Rank=0, Norma_loc=0.00000000, Norma_all=0.00000206, Ex=0.00000500
k=10, Rank=0[1.000000] [1.000000] [1.000000]
Υπάρχει σύγκλιση! και η λύση του συστήματος είναι:
[1.000000] [1.000000] [1.000000]
[msc16003@mpi9 erot2]$
```

συνέχεια από το δίπλα:

```
Βρέθηκε το αρχείο και ξεκινάμε να το διαβάσουμε!
n=3, l=30, ex=0.00000100
* * * Matrix A * * *
[8.000000][2.000000][5.000000]
[2.000000][2.000000][3.000000]
[1.000000][4.000000][5.000000]
* * * Vector b * * *
[2.000000][6.000000][3.000000]
* * * Vector X * * *
[0.000000][0.000000][0.000000]
Τέλος αρχικοποίησης
k=0, Rank=0, Norma_loc=0.06250000, Norma_all=3.06960893, Ex=0.00000100
k=0, Rank=0[0.250000] [3.000000] [0.600000]
k=1, Rank=0, Norma_loc=1.26562500, Norma_all=2.93097687, Ex=0.00000100
k=1, Rank=0[-0.875000] [1.850000] [-1.850000]
k=2, Rank=0, Norma_loc=3.30785155, Norma_all=5.25917101, Ex=0.00000100
k=2, Rank=0[0.943750] [6.650000] [-0.705000]
k=3, Rank=0, Norma_loc=3.66961956, Norma_all=5.81774855, Ex=0.00000100
k=3, Rank=0[-0.971875] [3.113750] [-4.908750]
k=4, Rank=0, Norma_loc=12.32997513, Norma_all=9.49929810, Ex=0.00000100
k=4, Rank=0[2.539531] [11.335000] [-1.696625]
k=5, Rank=0, Norma_loc=16.50707626, Norma_all=11.78461361, Ex=0.00000100
k=5, Rank=0[-1.523359] [3.005407] [-8.975906]
k=6, Rank=0, Norma_loc=43.98274231, Norma_all=18.00921440, Ex=0.00000100
k=6, Rank=0[5.108590] [17.987219] [-1.499654]
k=7, Rank=0, Norma_loc=70.86457062, Norma_all=23.80254173, Ex=0.00000100
k=7, Rank=0[-3.309521] [0.140893] [-14.811493]
k=8, Rank=0, Norma_loc=163.36625671, Norma_all=34.98381424, Ex=0.00000100
k=8, Rank=0[9.471960] [28.526760] [1.149191]
k=9, Rank=0, Norma_loc=291.44961548, Norma_all=47.73166656, Ex=0.00000100
k=9, Rank=0[-7.599935] [-8.195747] [-24.115801]
k=10, Rank=0, Norma_loc=623.56304932, Norma_all=68.70616150, Ex=0.00000100
k=10, Rank=0[17.371311] [46.773636] [8.676585]
k=11, Rank=0, Norma_loc=1172.21240234, Norma_all=95.23617554, Ex=0.00000100
```

```
k=12, Rank=0[32.279778] [80.306030] [25.882206]
k=13, Rank=0, Norma_loc=4662.52197266, Norma_all=189.47431946, Ex=0.00000100
k=13, Rank=0[-36.002884] [-68.103088] [-70.100784]
k=14, Rank=0, Norma_loc=9426.78710938, Norma_all=268.33813477, Ex=0.00000100
k=14, Rank=0[61.088760] [144.154053] [62.283043]
k=15, Rank=0, Norma_loc=18442.77148438, Norma_all=376.38769531, Ex=0.00000100
k=15, Rank=0[-74.715416] [-151.513321] [-126.940994]
k=16, Rank=0, Norma_loc=36933.86328125, Norma_all=531.57006836, Ex=0.00000100
k=16, Rank=0[117.466446] [268.126923] [136.753738]
k=17, Rank=0, Norma_loc=72748.46875000, Norma_all=747.09777832, Ex=0.00000100
k=17, Rank=0[-152.252808] [-319.597046] [-237.394821]
k=18, Rank=0, Norma_loc=144988.71875000, Norma_all=1053.64062500, Ex=0.00000100
k=18, Rank=0[228.521027] [511.345032] [286.728210]
k=19, Rank=0, Norma_loc=286559.34375000, Norma_all=1482.32812500, Ex=0.00000100
k=19, Rank=0[-306.791382] [-655.613342] [-454.180237]
k=20, Rank=0, Norma_loc=569734.18750000, Norma_all=2089.06250000, Ex=0.00000100
k=20, Rank=0[448.015991] [991.061768] [586.448914]
k=21, Rank=0, Norma_loc=1127975.75000000, Norma_all=2940.50805664, Ex=0.00000100
k=21, Rank=0[-614.046021] [-1324.689331] [-881.852661]
k=22, Rank=0, Norma_loc=2239890.00000000, Norma_all=4142.61083984, Ex=0.00000100
k=22, Rank=0[882.580200] [1939.825073] [1183.160645]
k=23, Rank=0, Norma_loc=4438445.00000000, Norma_all=5832.51074219, Ex=0.00000100
k=23, Rank=0[-1224.181641] [-2654.321289] [-1727.776123]
k=24, Rank=0, Norma_loc=8808265.00000000, Norma_all=8215.40332031, Ex=0.00000100
k=24, Rank=0[1743.690430] [3818.845703] [2368.893311]
k=25, Rank=0, Norma_loc=17461616.00000000, Norma_all=11568.20507812, Ex=0.00000100
k=25, Rank=0[-2435.019775] [-5294.030273] [-3403.214844]
k=26, Rank=0, Norma_loc=34642484.00000000, Norma_all=16292.95312500, Ex=0.00000100
k=26, Rank=0[3450.766846] [7542.841797] [4722.828125]
k=27, Rank=0, Norma_loc=68690864.00000000, Norma_all=22943.77929688, Ex=0.00000100
k=27, Rank=0[-4837.228027] [-10532.008789] [-6723.826660]
k=28, Rank=0, Norma_loc=136255920.00000000, Norma_all=32313.11914062, Ex=0.00000100
k=28, Rank=0[6835.643555] [14925.968750] [9393.653320]
k=29, Rank=0, Norma_loc=270205152.00000000, Norma_all=45504.90234375, Ex=0.00000100
k=29, Rank=0[-9602.275391] [-20923.125000] [-13307.302734]
Δεν υπάρχει σύγκλιση!
[msc16003@mpi9 erot2]$
```

Ακολουθούν 5 screenshots (3 της main + 2 μιας συνάρτησης) από την ενδεικτική λύση (erot2.c):

```
/*
 * Athanasios Rokopoulos (cnt16009)
 * Michael Galliakis (cnt16003)
 * MPI - Erotima B
 * Methodos Jacobi gia epilysi
 * grammikon systimaton
 * Date: 7/06/2017
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mpi.h"
#define ROOT 0 //Σταθερή τιμή για το rank του 0 ώστε να αναφερόμαστε σε αυτόν με το όνομα ROOT

//Συνάρτηση για να γεμίσουμε με αρχικές τιμές τα l,n,ex,A,b,x, είτε από αρχείο είτε με default.
void fillInitialValues(const char[],int*, int*, float*,float *[],float *[], float *[]) ;
//Συναρτήσεις για διάφορα print:
void printFinalVector(float[], int) ;
void printVector(float[], int, float) ;
void printVectorDebug(int, int, float[], int);
void printNorma(int, int, float, float, float);

int main(int argc, char** argv)
{
    int rank,size;
    //Αρχικοποιείται το MPI
    MPI_Init(&argc, &argv);
    //Παίρνει ο κάθε επεξεργαστής-υπολογιστής το rank του
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //Παίρνει ο κάθε επεξεργαστής-υπολογιστής το πλήθος των επεξεργαστών.
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //Δηλώνουμε τους 6 πίνακες που χρειαζόμαστε:
    float *matrix_A ;
    float *matrix_A_RowsLoc ;
    float *vector_b ;
    float *vector_b_elements ;
    float *vector_x ;
    float *vector_x_new ;
    int k,i,j,q ; //Διάφοροι μετρητές που χρειάζονται
    int l,n ; //Το λ και n του Jacobi αλγορίθμου
    float ex ; //Το "ξ" του Jacobi αλγορίθμου

    if (rank == ROOT) //Ο "0" μόνο
    {
        //Δηλώνουμε όνομα αρχείου για να διαβάσουμε από εκεί τις αρχικοποιήσεις
        const char path[] = "jacobiInput.txt" ;
        //Αρχικοποιεί τα n,l,ex,matrix_A,vector_b,vector_x
        fillInitialValues(path,&n, &l, &ex, &matrix_A, &vector_b, &vector_x) ;
    }

    //Ο "0" στέλνει σε όλους τις τιμές του l,ex,n
    MPI_Bcast(&l, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
    MPI_Bcast(&ex, 1, MPI_REAL, ROOT, MPI_COMM_WORLD);
    MPI_Bcast(&n, 1, MPI_INT, ROOT, MPI_COMM_WORLD);

    if(rank!=ROOT) //Κάθε επεξεργαστής, εκτός του "0"
        vector_x = malloc(n *sizeof(float)) ; //Δεσμεύει μνήμη για το διάνυσμα vector_x
```


Συνέχεια της main:

```
MPI_Bcast(&l, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
MPI_Bcast(&ex, 1, MPI_REAL, ROOT, MPI_COMM_WORLD);
MPI_Bcast(&n, 1, MPI_INT, ROOT, MPI_COMM_WORLD);

if(rank!=ROOT) //Κάθε επεξεργαστής, εκτός του "0"
    vector_x = malloc(n * sizeof(float)) ; //Δεσμεύει μνήμη για το διάνυσμα vector_x

//0 "0" στέλνει σε όλους τις τιμές του πίνακα vector_x
MPI_Bcast(vector_x, n, MPI_REAL, ROOT, MPI_COMM_WORLD);

int quota = n/size ; //Υπολογίζεται το μερίδιο που θα έχει ο κάθε επεξεργαστής
int quotaCrowd = quota*n ; //Υπολογίζεται το πλήθος αριθμών που θα έχει ο κάθε επεξεργαστής

//Δεσμεύετε μνήμη για το τοπικό διάνυσμα vector_x_new για κάθε επεξεργαστή
vector_x_new = malloc(quota * sizeof(float)) ;

//Δεσμεύετε μνήμη για τον πίνακα matrix_A_RowsLoc (που είναι τοπικός [υπο]πίνακας του matrix_A για κάθε επεξεργαστή...)
matrix_A_RowsLoc = malloc(quotaCrowd * sizeof(float)) ;
//0 "0" μοιράζει τα περιεχόμενα του matrix_A στους τοπικούς [υπο]πίνακες του κάθε επεξεργαστή...
MPI_Scatter(matrix_A, quotaCrowd, MPI_REAL, matrix_A_RowsLoc, quotaCrowd, MPI_REAL, ROOT, MPI_COMM_WORLD);

//Δεσμεύετε μνήμη για το διάνυσμα vector_b_elements (που είναι τοπικό [υπο]διάνυσμα του vector_b για κάθε επεξεργαστή...)
vector_b_elements = malloc(quota * sizeof(float)) ;
//0 "0" μοιράζει τα περιεχόμενα του vector_b στα τοπικά [υπο]διανύσματα του κάθε επεξεργαστή...
MPI_Scatter(vector_b, quota, MPI_REAL, vector_b_elements, quota, MPI_REAL, ROOT, MPI_COMM_WORLD);

float sum ;
int rowIndex ;
float norma_loc, norma_all ;
for (k=0;k<l;k++)
{
    norma_loc = 0 ; //Μηδενίζει στην αρχή της κάθε επανάληψης η τοπική νόρμα
    for (q=0;q<quota;q++) // Για όσο είναι το "μερίδιο" του κάθε επεξεργαστή
    {
        rowIndex = q*n ; //Υπολογίζετε ο δείκτης της κάθε γραμμής για τον πίνακα matrix_A_RowsLoc

        //Στην συνέχεια γίνονται οι απαραίτητοι υπολογισμοί, με βάση τον αλγόριθμο του Jacobi
        //ώστε να βρει ο κάθε επεξεργαστής τοπικά τις νέες δικές του τιμές για τον πίνακα x.
        sum = -matrix_A_RowsLoc[rowIndex+(q+(rank*quota))] * vector_x[q+(rank*quota)] ;
        for(j=0;j<n;j++)
            sum += matrix_A_RowsLoc[rowIndex+j] * vector_x[j];
        vector_x_new[q] = (vector_b_elements[q]-sum)/matrix_A_RowsLoc[rowIndex+q+(rank*quota)] ;

        //Σύμφωνα με τον αλγόριθμο, υπολογίζει κάθε επεξεργαστής τοπικά την νόρμα του.
        norma_loc += pow((vector_x_new[q]-vector_x[q+(rank*quota)]),2) ; //power 2
    }

    //Με την Allgather, γίνεται gather και broadcast, ώστε κάθε επεξεργαστής να έχει
    //το νέο πίνακα x, με τις νέες υπολογισμένες τιμές που έχει βρει ο κάθε επεξεργαστής,
    //ώστε αν έχουμε βρει την λύση, να εμφανιστεί το διάνυσμα στον χρήστη και αν όχι
    //ακόμη, να χρησιμοποιηθεί αυτό το νέο διάνυσμα x στην επόμενη επανάληψη του βρόχου.
    MPI_Allgather(vector_x_new, quota, MPI_REAL, vector_x, quota, MPI_REAL, MPI_COMM_WORLD);

    //Με την Allreduce, γίνεται reduce και broadcast, ώστε κάθε επεξεργαστής να έχει
    //την συνολική νόρμα (χωρίς να έχει υπολογιστεί ακόμη η ρίζα...)
    MPI_Allreduce(&norma_loc, &norma_all, 1, MPI_REAL, MPI_SUM, MPI_COMM_WORLD);

    //Με την Allgather, γίνεται gather και broadcast, ώστε κάθε επεξεργαστής να έχει
    //το νέο πίνακα x, με τις νέες υπολογισμένες τιμές που έχει βρει ο κάθε επεξεργαστής,
    //ώστε αν έχουμε βρει την λύση, να εμφανιστεί το διάνυσμα στον χρήστη και αν όχι
    //ακόμη, να χρησιμοποιηθεί αυτό το νέο διάνυσμα x στην επόμενη επανάληψη του βρόχου.
    MPI_Allgather(vector_x_new, quota, MPI_REAL, vector_x, quota, MPI_REAL, MPI_COMM_WORLD);

    //Με την Allreduce, γίνεται reduce και broadcast, ώστε κάθε επεξεργαστής να έχει
    //την συνολική νόρμα (χωρίς να έχει υπολογιστεί ακόμη η ρίζα...)
    MPI_Allreduce(&norma_loc, &norma_all, 1, MPI_REAL, MPI_SUM, MPI_COMM_WORLD);

    //Κάθε επεξεργαστής βρίσκει την ρίζα της συνολικής νόρμας
    norma_all = sqrt(norma_all) ;

    if(rank==ROOT){//Για λόγους επαλήθευσης, ο "0" μόνο, εμφανίζει κάποια μηνύματα
        //printVector(vector_x,n,norma_all) ;
        printNorma(k,rank,norma_loc,norma_all,ex) ;
        printVectorDebug(k,rank,vector_x,n) ;
    }

    //Εάν η νόρμα είναι μικρότερη από το "ε" που έχει δοθεί, σημαίνει ότι υπάρχει σύγκλιση
    if (norma_all<=ex)
        break ;//Εάν υπάρχει σύγκλιση, τότε κάνουν όλοι οι επεξεργαστές break για να βγουν από το loop
}

if (rank==ROOT) //ο "0" μόνο
{
    //Εμφανίζει αν υπάρχει ή δεν υπάρχει σύγκλιση και εφόσον υπάρχει
    //εμφανίζει και τον διάνυσμα x με τη λύση του γραμμικού συστήματος
    if(k==l) //Αν k=l, σημαίνει ότι δεν έγινε κάποιο break μέσα στο loop...
        printf("Δεν υπάρχει σύγκλιση!\n") ;
    else{
        printf("Υπάρχει σύγκλιση! και η λύση του συστήματος είναι:\n") ;
        printFinalVector(vector_x,n) ;
    }
}

MPI_Finalize(); //Τερματίζει το MPI
return 0 ;
}
```

Συνάρτηση για να φορτώσουμε αρχικές τιμές από αρχείο...

```
void fillInitialValues(const char path[],int *n, int *l, float *ex, float *matrix_A[],float *vector_b[], float *vector_x[])
{
    FILE *f;
    int i,j,rowIndex ;

    f = fopen(path,"r");
    if (f){
        printf("Βρέθηκε το αρχείο και ξεκινάμε να το διαβάσουμε!\n") ;
        fscanf(f, "%d %d %f", n,l,ex);
        printf("n=%d, l=%d, ex=%.*f\n",*n,*l,8,*ex) ;

        *matrix_A = (float*)malloc( *n * *n * sizeof(float)) ;
        printf("** * Matrix A * * *\n") ;
        for(i=0;i<*n;i++){
            rowIndex = i* *n ;
            for(j=0;j<*n;j++){
                fscanf(f, "%f",&(*matrix_A)[rowIndex+j]);
                printf("[%f]",(*matrix_A)[rowIndex+j]) ;
            }
            printf("\n") ;
        }
        *vector_b = (float*)malloc(*n *sizeof(float)) ;
        *vector_x = (float*)malloc(*n *sizeof(float)) ;
        printf("** * Vector b * * *\n") ;
        for(j=0;j<*n;j++){
            fscanf(f, "%f",&(*vector_b)[j]);
            printf("[%f]",(*vector_b)[j]) ;
        }
        printf("\n* * Vector X * * *\n") ;
        for(j=0;j<*n;j++){
            fscanf(f, "%f",&(*vector_x)[j]);
            printf("[%f]",(*vector_x)[j]) ;
        }
        printf("\nΤέλος αρχικοποίησης\n") ;
        fclose(f);
    }
    else
    {
        printf("Δεν βρέθηκε το αρχείο και γεμίζουμε με default τιμές!\n") ;
        *n = 4;
        *l = 20;
        *ex = 0.000001 ;
        printf("n=%d, l=%d, ex=%.*f\n",*n,*l,8,*ex) ;

        *matrix_A = (float*) malloc(*n * *n *sizeof(float)) ;
        (*matrix_A)[0] = 10; (*matrix_A)[1] = -1 ;(*matrix_A)[2] = 2 ;(*matrix_A)[3] = 0 ;
        (*matrix_A)[4] = -1 ; (*matrix_A)[4+1] = 11 ;(*matrix_A)[4+2] = -1 ;(*matrix_A)[4+3] = 3 ;
        (*matrix_A)[8] = 2 ; (*matrix_A)[8+1] = -1 ;(*matrix_A)[8+2] = 10 ;(*matrix_A)[8+3] = -1 ;
        (*matrix_A)[12] = 0 ; (*matrix_A)[12+1] = 3 ;(*matrix_A)[12+2] = -1 ;(*matrix_A)[12+3] = 8 ;
    }
}
```

```
        printf("\nΤέλος αρχικοποίησης\n") ;
        fclose(f);
    }
    else
    {
        printf("Δεν βρέθηκε το αρχείο και γεμίζουμε με default τιμές!\n") ;
        *n = 4;
        *l = 20;
        *ex = 0.000001 ;
        printf("n=%d, l=%d, ex=%.*f\n",*n,*l,8,*ex) ;

        *matrix_A = (float*) malloc(*n * *n *sizeof(float)) ;
        (*matrix_A)[0] = 10; (*matrix_A)[1] = -1 ;(*matrix_A)[2] = 2 ;(*matrix_A)[3] = 0 ;
        (*matrix_A)[4] = -1 ; (*matrix_A)[4+1] = 11 ;(*matrix_A)[4+2] = -1 ;(*matrix_A)[4+3] = 3 ;
        (*matrix_A)[8] = 2 ; (*matrix_A)[8+1] = -1 ;(*matrix_A)[8+2] = 10 ;(*matrix_A)[8+3] = -1 ;
        (*matrix_A)[12] = 0 ; (*matrix_A)[12+1] = 3 ;(*matrix_A)[12+2] = -1 ;(*matrix_A)[12+3] = 8 ;

        printf("** * Matrix A * * *\n") ;
        for(i=0;i<*n;i++){
            rowIndex = i* *n ;
            for(j=0;j<*n;j++){
                printf("[%f]",(*matrix_A)[rowIndex+j]) ;
            }
            printf("\n") ;
        }

        *vector_b = (float*) malloc(*n *sizeof(float)) ;
        *vector_x = (float*) malloc(*n *sizeof(float)) ;

        (*vector_b)[0] = 6 ;
        (*vector_b)[1] = 25 ;
        (*vector_b)[2] = -11 ;
        (*vector_b)[3] = 15 ;

        printf("** * Vector b * * *\n") ;
        for(j=0;j<*n;j++){
            printf("[%f]",(*vector_b)[j]) ;
        }

        (*vector_x)[0] = 0 ;
        (*vector_x)[1] = 0 ;
        (*vector_x)[2] = 0 ;
        (*vector_x)[3] = 0 ;

        printf("\n* * Vector X * * *\n") ;
        for(j=0;j<*n;j++){
            printf("[%f]",(*vector_x)[j]) ;
        }

        printf("\nΤέλος αρχικοποίησης\n") ;
    }
}
```

Σημείωση:

Για να γίνει compile, χρειάζεται και η παράμετρος “-lm”

```
msc16003@mpi9 erot2]$ mpicc -o erot2.2 erot2.2.c -lm  
msc16003@mpi9 erot2]$ mpiexec -n 4 erot2.2
```

Επειδή υπάρχει χώρος στην τελευταία σελίδα, ακολουθεί & μια χαμογελαστή φατσούλα



Τέλος