



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ  
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ  
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ  
www.cslab.ece.ntua.gr

## ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ 9ο εξάμηνο ΗΜΜΥ, ακαδημαϊκό έτος 2016-17

### ΑΣΚΗΣΗ 3 - Παράλληλη επίλυση εξίσωσης θερμότητας

Προθεσμίες παράδοσης:  
Επίδειξη προγραμμάτων **6 Δεκεμβρίου**  
Παράδοση Τελικής Αναφοράς **13 Δεκεμβρίου**<sup>1, 2</sup>

#### 1 Διάδοση θερμότητας σε δύο διαστάσεις

Για την επίλυση του προβλήματος της διάδοσης θερμότητας σε δύο διαστάσεις, χρησιμοποιούνται τρεις υπολογιστικοί πυρήνες, οι οποίοι αποτελούν ευρέως διαδεδομένη δομική μονάδα για την επίλυση μερικών διαφορικών εξισώσεων: η μέθοδος Jacobi, η μέθοδος Gauss-Seidel με Successive Over-Relaxation και η μέθοδος Red-Black SOR, που πραγματοποιεί Red-Black ordering στα στοιχεία του υπολογιστικού χωρίου και συνδυάζει τις δύο προηγούμενες μεθόδους.

##### 1.1 Μέθοδος Jacobi

```
for (t = 0; t < T && !converged; t++) {  
    for (i = 1; i < X - 1; i++)  
        for (j = 1; j < Y - 1; j++)  
            U[t+1][i][j] = (1/4) * (U[t][i-1][j] + U[t][i][j-1]  
                + U[t][i+1][j] + U[t][i][j+1]);  
    converged = check_convergence(U[t+1], U[t])  
}
```

##### 1.2 Μέθοδος Gauss-Seidel SOR

```
for (t = 0; t < T && !converged; t++) {  
    for (i = 1; i < X - 1; i++)  
        for (j = 1; j < Y - 1; j++)  
            U[t+1][i][j] = U[t][i][j]  
                + (omega/4) * (U[t+1][i-1][j] + U[t+1][i][j-1]
```

<sup>1</sup>filename: a3-parlabXX-final.pdf

<sup>2</sup>mail to: nikela@cslab.ece.ntua.gr, cc: goumas@cslab.ece.ntua.gr

```

        +U[t][i+1][j]+U[t][i][j+1]
        -4*U[t][i][j]);
    converged=check_convergence(U[t+1],U[t])
}

```

### 1.3 Μέθοδος Red-Black SOR

```

for (t = 0; t < T && !converged; t++) {

    //Red phase
    for (i = 1; i < X - 1; i++)
        for (j = 1; j < Y - 1; j++)
            if ((i+j)%2==0)
                U[t+1][i][j]=U[t][i][j]
                    +(omega/4)*(U[t][i-1][j]+U[t][i][j-1]
                        +U[t][i+1][j]+U[t][i][j+1]
                        -4*U[t][i][j]);

    //Black phase
    for (i = 1; i < X - 1; i++)
        for (j = 1; j < Y - 1; j++)
            if ((i+j)%1==0)
                U[t+1][i][j]=U[t][i][j]
                    +(omega/4)*(U[t+1][i-1][j]+U[t+1][i][j-1]
                        +U[t][i+1][j]+U[t][i][j+1]
                        -4*U[t][i][j]);

    converged=check_convergence(U[t+1],U[t])
}

```

## 2 Ζητούμενα

Στα αρχεία `Jacobi_serial.c`, `GaussSeidelSOR_serial.c` και `RedBlackSOR_serial.c` σας δίνονται οι σειριακές υλοποιήσεις των τριών μεθόδων. Για τις μεθόδους **Jacobi** και **Gauss-Seidel** (και προαιρετικά για τη μέθοδο **Red-Black SOR**):

1. Ανακαλύψτε τον παραλληλισμό του αλγορίθμου και σχεδιάστε την παραλληλοποίησή του σε αρχιτεκτονικές καταναμημένης μνήμης με μοντέλο ανταλλαγής μηνυμάτων.
2. Αναπτύξτε παράλληλο πρόγραμμα στο μοντέλο ανταλλαγής μηνυμάτων με τη βοήθεια της βιβλιοθήκης MPI. Στο αρχείο `mpi_skeleton.c` σας δίνεται σκελετός υλοποίησης σε MPI, στον οποίο καλείστε να συμπληρώσετε τον κώδικά σας.
3. Πραγματοποιείτε μετρήσεις επίδοσης με βάση συγκεκριμένο σενάριο που θα σας δοθεί στο εργαστήριο.
4. Συγκεντρώστε τα αποτελέσματα, τις συγκρίσεις και τα σχόλιά σας στην Τελική Αναφορά.

## 3 Διευκρινίσεις

- Τα βοηθητικά αρχεία για την άσκηση βρίσκονται στον `scirouter`, στο φάκελο:  
`/home/parallel/pps/2016-2017/a3`
- Για οδηγίες σύνδεσης, μεταγλώττισης, εκτέλεσης κ.λ.π. των προγραμμάτων σας συμβουλευτείτε τις "ΟΔΗΓΙΕΣ ΕΡΓΑΣΤΗΡΙΟΥ" που σας έχουν δοθεί. Το αρχείο με τις οδηγίες είναι διαθέσιμο στο:  
<http://www.cslab.ece.ntua.gr/courses/pps/files/fall2016/pps-lab-guide.pdf>.

- Σε όλες τις εκδόσεις του πυρήνα, χρησιμοποιούνται πραγματικοί αριθμοί διπλής ακρίβειας.
- Η μνήμη που θα χρησιμοποιήσετε θα δεσμεύεται δυναμικά (π.χ. με malloc).
- Το πρόγραμμά σας πρέπει να είναι παραμετρικό.
- Στο παράλληλο πρόγραμμα στο μοντέλο της ανταλλαγής μηνυμάτων, αρχικά μία διεργασία έχει όλο τον πίνακα A. Στη διεργασία αυτή επιστρέφονται τα αποτελέσματα της παράλληλης εκτέλεσης.
- Για τη μέτρηση των χρόνων εκτέλεσης χρησιμοποιείται η συνάρτηση βιβλιοθήκης `gettimeofday` του `sys/time.h`. Παρατηρείστε ότι κατά την μέτρηση χρόνων ενδιαφέρει **μόνο** το υπολογιστικό κομμάτι του αλγορίθμου, και όχι η φάση αρχικοποίησης ή π.χ. εκτύπωσης των αποτελεσμάτων. Για το λόγο αυτό πραγματοποιείται κατάλληλος συγχρονισμός των διεργασιών ή νημάτων πριν τις μετρήσεις χρόνου. Στον κώδικα που σας δίνεται, έχουν ήδη οριστεί οι μετρητές για το συνολικό χρόνο εκτέλεσης του υπολογιστικού πυρήνα. Αντίστοιχα, θα μετρήσετε το χρόνο που καταναλώνεται σε υπολογισμούς και επικοινωνία.

## 4 Χρήσιμες συναρτήσεις του MPI

### 4.1 Point-to-point communication

- `int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
- `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
- `int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)`
- `int MPI_Isend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)`
- `int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)`
- `int MPI_Wait(MPI_Request *request, MPI_Status *status)`
- `int MPI_Waitall(int count, MPI_Request array_of_requests[], MPI_Status *array_of_statuses)`
- `int MPI_Waitsome(int incount, MPI_Request array_of_requests[], int *outcount, int array_of_indices[], MPI_Status array_of_statuses[])`
- `int MPI_Waitany(int count, MPI_Request array_of_requests[], int *index, MPI_Status *status)`

### 4.2 Collective Communication

- `int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
- `int MPI_Scatterv(const void *sendbuf, const int sendcounts[], const int displs[], MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
- `int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
- `int MPI_Gatherv(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype, int root, MPI_Comm comm)`

- int MPI\_Bcast(void \*buffer, int count, MPI\_Datatype datatype, int root, MPI\_Comm comm)
- int MPI\_Reduce(const void \*sendbuf, void \*recvbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm)
- int MPI\_Allreduce(const void \*sendbuf, void \*recvbuf, int count, MPI\_Datatype datatype, MPI\_Op op, MPI\_Comm comm)

### 4.3 Cartesian Communicators

- int MPI\_Cart\_create(MPI\_Comm comm\_old, int ndims, const int dims[], const int periods[], int reorder, MPI\_Comm \*comm\_cart)
- int MPI\_Cart\_coords(MPI\_Comm comm, int rank, int maxdims, int coords[])
- int MPI\_Cart\_shift(MPI\_Comm comm, int direction, int disp, int \*rank\_source, int \*rank\_dest)

### 4.4 Datatypes

- int MPI\_Type\_vector(int count, int blocklength, int stride, MPI\_Datatype oldtype, MPI\_Datatype \*newtype)
- int MPI\_Type\_contiguous(int count, MPI\_Datatype oldtype, MPI\_Datatype \*newtype)
- int MPI\_Type\_create\_resized(MPI\_Datatype oldtype, MPI\_Aint lb, MPI\_Aint extent, MPI\_Datatype \*newtype)
- int MPI\_Type\_commit(MPI\_Datatype \*datatype)