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 2021-22

ΑΣΚΗΣΗ 3 - Παράλληλη επίλυση εξίσωσης θερμότητας

Προθεσμία παράδοσης: **12 Ιανουαρίου** [1, 2]
Προθεσμία παράδοσης ελέγχου προόδου: **23 Δεκεμβρίου**

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]
                                   +U[t][i+1][j]+U[t][i][j+1]
```

[1] filename: a3-parlabXX-final.pdf
[2] Υποβολή στο site του μαθήματος

```
                                        -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)%2==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+1][i+1][j]+U[t+1][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` σάς 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 κατανεμημένης 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. Παραδώστε τον παράλληλο κώδικά σας για τη μέθοδο Jacobi, και αναφέρετε το χρόνο εκτέλεσης για χωρίο 2048 x 2048, σε 32 διεργασίες, χωρίς έλεγχο σύγκλισης, για τον έλεγχο προόδου.
4. Πραγματοποιείστε μετρήσεις επίδοσης βάσει του σεναρίου μετρήσεων που σας έχει δοθεί μαζί με την εκφώνηση της άσκησης, στη σελίδα του μαθήματος.
5. 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/2021-2022/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 στην ιστοσελίδα του μαθήματος.

- 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)`