Παντελεήμων Πρώιος

ice 18390023

5ο ‘Εξάμηνο

Μηχανικών Πληροφορικής και Υπολογιστών

ice18390023@uniwa.gr

Εισαγωγή στον παράλληλο υπολογισμό

Εργασία 3

ΥΠΕΥΘΗΝΟΙ ΚΑΘΗΓΗΤΕΣ

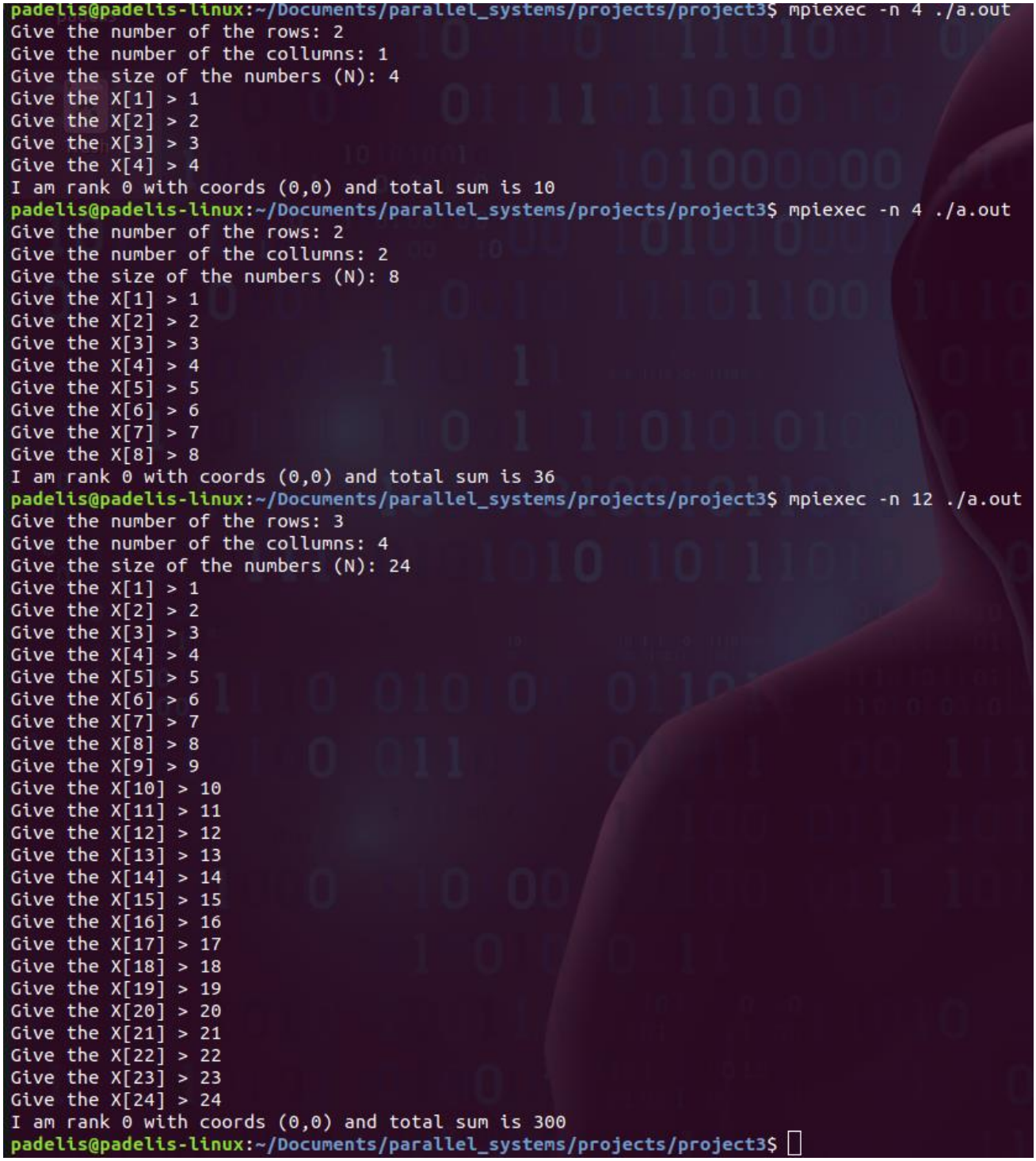
ΜΑΜΑΛΗΣ ΒΑΣΙΛΗΣ
ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

Το πρώτο τρέξιμο της εικόνας είναι για 4 επεξεργαστές, η τοπολογία είναι 2x1 για 4 στοιχεία.

Το δεύτερο τρέξιμο της εικόνας είναι για 4 επεξεργαστές, η τοπολογία είναι 2x2 για 8 στοιχεία.

Το τρίτο τρέξιμο της εικόνας είναι για 12 επεξεργαστές, η τοπολογία είναι 3x4 για 24 στοιχεία.



```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#define ROOT 0

int read_size(int );
int *read_sequence(int );
int *create_malloc(int );
void readMN(int , int *);
int summary( int *, int);


int main(int argc, char** argv){

    int rank;
    int menu_response;
    int N, p;
    int *X;
    int tmp;
    int i;
    int MN[2];
    int ndims = 2;
    int reorder = 1;
    int coords[2];
    int root_cart_rank;
    int loc_sizeX;
    int *loc_X;
    int direction;
    int displ;
    int source, dest;
    int loc_sum, total_sum;
    int recieved, send;

    int root_cords[2];
    root_cords[0] = root_cords[1] = 0;

    int periods[2];
    periods[0] = periods[1] = 0;

    // ========================= MPI Initalized ========================

    // Η MPI_Init, επιστρέφει MPI_SUCCESS, στην επιτυχεία.
    // Στην μεταβλητή tmp, εκχωρείται η επιστρεφόμενη τιμή της MPI_Init και
    // η συνθήκη, ελέγχει αν η tmp δεν είναι MPI_SUCCESS, έτσι ώστε να πράξει
    // καταλλήλος, αλλιώς να συνεχίση κανονικά.

    tmp = MPI_Init( NULL, NULL);

    if (tmp != MPI_SUCCESS){
        perror("MPI initalization");
        // Η MPI_Abort, τερματίζει το περιβάλλων MPI του communicator
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    MPI_Status status;
    MPI_Comm cart_comm;

    // κάθε processor μαθαίνει τον αριθμό του, για τον Communicator
MPI_COMM_WORLD
    // και εκχωρείται στην rank.
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Πόσοι υπάρχουν, υπο τον MPI_COMM_WORLD
    // θα είναι ο ίδιος αριθμός για όλους του MPI_COMM_WORLD
    MPI_Comm_size(MPI_COMM_WORLD, &p);



    // Η συνθήκη θα είναι αληθής μόνο για τον processor που έχει
    // rank == ROOT == 0.
    if ( rank == ROOT){

        // Διαβάζει το M, που είναι οι σειρές και το N, που είναι οι στήλες
        // και τα επιστρέφει στον πίνακα MN, με πρώτο στοιχείο τις σειρές
        // και δεύτερο τις στήλες.
        readMN(p, MN);

        // debug
        //printf("M = %d, N = %d\n", MN[0], MN[1]);

        // Καλεί την συνάρτηση με παράμετρο τον πολλαπλασιασμό των καρτεσιανών
        // συντεταγμένων, όπου προδίδουν τους επεξεργαστές που θα συμμετάσχουν
        // και επιστρέφει τον αριθμό των ακεραίων που θα διαβάσει.
        N = read_size( MN[0] * MN[1] );

        // Επιστρέφει την διεύθηνση των ακεραίων που διαβάστηκαν.
        X = read_sequence(N);

        // debug
        //for( i = 0; i < N; printf("X[%d] = > %d\n", ++i, X[i]));

        // Το τοπικό μέγεθος, είναι το μέγεθος των αριθμών πρός τον
        // πολλαπλασιασμό των συντεταγμένων.
        loc_sizeX = N / ( MN[0] * MN[1] );

    }

    // Γίνονται broadcast οι τιμές των στηλών και σειρών
    MPI_Bcast( MN, 2, MPI_INT, ROOT, MPI_COMM_WORLD);

    // Δημιουργείται μια εικονική καρτεσιανή τοπολογία
    tmp = MPI_Cart_create( MPI_COMM_WORLD, ndims, MN, periods, reorder,
&cart_comm);


    // Ελέγχεται η επιστρεφόμενη τιμή της MPI_Cart_create.
    if (tmp != MPI_SUCCESS){
        perror("MPI cart create");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }


    // Όποι επεξεργαστές δεν συμμετάσχουν, τους έχει επιστραφεί στον communicator
    // κατά την δημιουργεία της καρτεσιανής τοπολογίας, MPI_COMM_NULL.
    if ( cart_comm != MPI_COMM_NULL ){

        // Οι επεξεργαστές ξέρουν τις συντεταγμένες τους
        MPI_Cart_coords ( cart_comm, rank, ndims, coords);

        // Γίνεται broadcast του μέγεθους της τοπικής τιμής
        MPI_Bcast( &loc_sizeX, 1, MPI_INT, ROOT, cart_comm);
```

```
        // Δεσμεύεται μνήμη
        loc_X = create_malloc(loc_sizeX);

        // Παραδίδονται οι τοπικές τιμές
        MPI_Scatter( X, loc_sizeX, MPI_INT, loc_X, loc_sizeX, MPI_INT, ROOT,
cart_comm);

        // Υπολογίζεται το τοπικό άθροισμα
        loc_sum = summary(loc_X, loc_sizeX);

        direction = 0;

        displ = -1;

        // Κάθε επεξεργαστής μαθαίνει τους γύρο του, για την ακρίβεια τον κάτω
και
        // πάνω, έτσι ώστε να επικονονήσει μαζί του.
        MPI_Cart_shift(cart_comm, direction, displ, &source, &dest);



        recieved = 0;


        // Αν ο αριθμός είναι αρνιτικός, τότε δεν έχει γείτονα προς αυτήν την
        // κατεύθηνση.
        if( !(source < 0) ){
            // Παραλαμβάνει το αποτέλεσμα απο τον κάτω επεξεργαστή.
            MPI_Recv( &recieved, 1, MPI_INT, source, 0, cart_comm, &status);
        }

        // Συναθρίζεται το τοπικό άθροισμα και το ληφθείσαν.
        send = loc_sum + recieved;

        // Αν ο αριθμός είναι αρνιτικός, τότε δεν έχει γείτονα προς αυτήν την
        // κατεύθηνση.
        if( !(dest < 0) ){
            // Στέλνει το αποτέλεσμα στον επάνω επεξεργαστή.
            MPI_Send( &send, 1, MPI_INT, dest, 0, cart_comm);
        }

        // Οτι δεν στάλθηκε σε απο πάνω επεξεργαστή επειδή τα coords είναι (0,j)
        // υπάρχει στην send.


        direction = 1;

        displ = -1;

        // Κάθε επεξεργαστής μαθαίνει τους γύρο του, για την ακρίβεια τον δεξί
και
        // αριστερό, έτσι ώστε να επικονονήσει μαζί του.
        MPI_Cart_shift(cart_comm, direction, displ, &source, &dest);

        // Αν η συντεταγμένη της σειράς είναι 0, τότε ανήκει στην πρώτη σειρά.
        if(coords[0] == 0){

            recieved = 0;

            // Αν ο αριθμός είναι αρνιτικός, τότε δεν έχει γείτονα προς αυτήν την
```

```
            // κατεύθηνση.
            if( !(source < 0) ){
                // Παραλαμβάνει το αποτέλεσμα απο τον κάτω επεξεργαστή.
                MPI_Recv( &recieved, 1, MPI_INT, source, 0, cart_comm, &status);
            }

            send += recieved;

            // Αν ο αριθμός είναι αρνιτικός, τότε δεν έχει γείτονα προς αυτήν την
            // κατεύθηνση.
            if( !(dest < 0) ){
                // Στέλνει το αποτέλεσμα στον επάνω επεξεργαστή.
                MPI_Send( &send, 1, MPI_INT, dest, 0, cart_comm);
            }

        }


        // Όλοι ξέρουν το rank που έχει της συντεταγμένες του (0,0) και
        // συλέχθηκαν εκεί όλες οι πληροφορίες.
        MPI_Cart_rank( cart_comm, root_cords, &root_cart_rank);

        // Ο rank με συντεταγμένες (0,0) εμφανίζει κατάλληλο μήνυμα στην οθόνη.
        if (rank == root_cart_rank){
            total_sum = send;
            printf("I am rank %d with coords (%d,%d) and total sum is %d\n",
                rank, coords[0], coords[1], total_sum);
        }

    } /* if (cart_comm == MPI_COMM_NULL) */



    // Ο ROOT αποδευσμεύει την μνήμη που έχει δεσμεύσει, το οποίο μπορεί να γίνει
    // μετά την MPI_Scatter που διαμοιράζει τα δεδομένα, για να μην γίνεται
    // κατάχρηση πόρων.
    if (rank == ROOT){
        free(X);
    }

    // Τέλος της παραλληλίας και της χρήσης των συναρτήσεων MPI.
    MPI_Finalize();

    return 0;
}


// Η συνάρτηση read_size, έχει ως παράμετρο το μέγεθος των processor.
// Εκτυπώνει στο stdout κατάλληλο μήνυμα και περιμένει για έναν ακέραιο αριθμό.
// Ο αριθμός πρέπει να είναι μεγαλύτερος ή ίσος με τους επεξεργαστές,
// αλλιώς θα εκτυπωθεί κατάλληλο μήνυμα και θα ξανά ζητηθεί αριθμός, μέχρις
// ότου να είναι μεγαλύτερος ή ισος απο τον αριθμό των επεξεργαστών και η
// διαίρεση N πρός p να μην έχει υπόλοιπο. Τέλος, θα επιστραφεί αυτός ο αριθμός.
int read_size(int p){
    int N;

    do{
        printf("Give the size of the numbers (N): ");
        scanf("%d", &N);
```

```
        if ( (N < p) || (N%p != 0) ){
            printf("N >= processors, N mod processors == 0\n");
        }
    }while( (N < p) || (N%p != 0) );

    return N;
}


// Η συνάρτηση read_sequence, δέχεται ως παράμετρο, το μέγεθος των στοιχείον
// προς ανάγνωση από το stdin. Επιστρέφει την διεύθηνση.
int *read_sequence(int N){

    int i, tmp;

    // Εκχωρεί στην ptr την επιστρεφόμενη διεύθηνση απο την create_malloc
    int *ptr = create_malloc(N);

    // Κάνει N επαναλήψεις για την ανάγνωση όλων των δεδομένων απο το stdin.
    for ( i = 0; i < N; i++){
        printf("Give the X[%d] > ", i+1);
        scanf("%d", ptr+i);
    }
    return ptr;
}


// Η συνάρτηση create_malloc, δέχεται ως παράμετρο το πλήθος των ακεραίων
// που θα γίνουν allocate, κάνει έλεγχο και επιστρέφει την διεύθυνση.
int *create_malloc(int len){

    int *addr = NULL;

    addr = (int *) malloc( len * sizeof(int) );

    if (addr == NULL){
        perror("Malloc error");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    return addr;

}


// Η συνάρτηση readMN, δέχεται ως πρώτη παράμετρο, το μέγεθος των επεξεργαστών
// και ως δεύτερη έναν πίνακα 2 στοιχείων. Στέλνει κατάλληλα μηνύματα στο stdout
// και διαβάζει 2 ακέραιους αριθμούς οι οποίοι πρέπει να είναι μεγαλύτερη ή ίση
// με το 1 και ο πολλαπλασιασμός αυτόν των 2 να είνα μικρότερος ή ίσος με το p.
void readMN(int p, int *arr){

    int flag;

    do{

        flag = 0;

        printf("Give the number of the rows: ");
        scanf("%d", arr);

        printf("Give the number of the collumns: ");
```

```
        scanf("%d", arr+1);

        if (( arr[0] == 0 ) || ( arr[1] == 0 ) || arr[0] * arr[1] > p ){
            printf("row >= 1, collumn >= 1, row * collumn <= process\n");
            flag = 1;
        }

    }while( flag );

}



// Επιστρέφει το άθροισμα του πίνακα arr, μεγέθους size.
int summary( int *arr, int size){

    int sum = 0;
    int i = 0;

    for (; i < size; sum += arr[i++] );

    return sum;
}
```