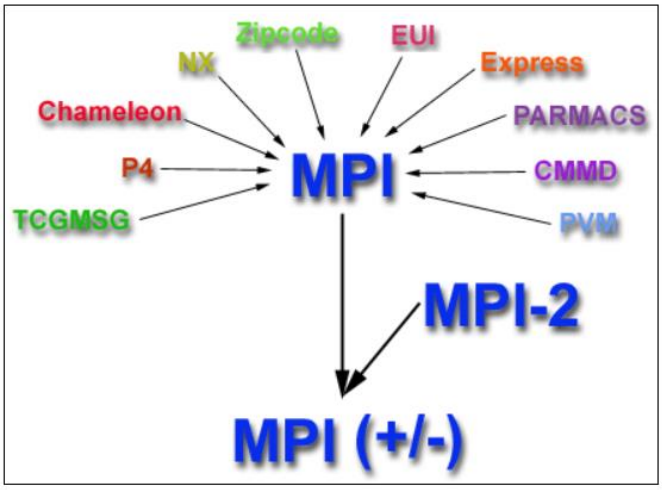
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ : ΤΜΗΜΑ
ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΑΡΑΛΛΗΛΟ ΥΠΟΛΟΓΙΣΜΟ (Ε2)

ΕΡΓΑΣΙΑ 1

(Καθηγητής κος Ιορδανάκης)

ΠΑΟΛΑ ΒΕΛΑΣΚΟ



Όνομα : Πάολα Βελάσκο
Α.Μ. : cs161020 – 9ο εξάμηνο.
E-mail: cs161020@uniwa.gr
Ημ/νια Παράδοσης: 29/11/2020

ΣΚΟΠΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Η εκπόνηση αυτής της εργαστηριακής εργασίας έχει ως σκοπό να μας εμβαθύνει τις γνώσεις μας για τον παράλληλο υπολογισμό. Συγκεκριμένα ασχολούμαστε με το MPI (Message Passing Interface) και την point-to-point επικοινωνία.

ΕΙΣΑΓΩΓΗ – ΘΕΩΡΙΑ

Το MPI είναι ένα εργαλείο/διεπαφή για ανάπτυξη παράλληλων προγραμμάτων. Ένα τέτοιο πρόγραμμα αποτελείται από πολλές διεργασία οι οποίες έχουν το δικό τους χώρο διευθύνσεων καθόλη τη διάρκεια της εκτέλεσής τους, και μπορούν να εκτελούνται παράλληλα σε ένα ή περισσότερα υπολογιστικά συστήματα/ επεξεργαστές.

Η ανταλλαγή δεδομένων μεταξύ των διεργασιών γίνεται με πέρασμα μηνυμάτων από τη μια διεργασία στην άλλη.

Συγκεκριμένα, στην υλοποίηση της εργασίας έγινε η χρήση των συναρτήσεων «αναστέλλουσας επικοινωνίας»

Έγινε η χρήση δηλαδή των:

- MPI_Send
- MPI_Recv

Το πρότυπο της συνάρτησης (MPI_Send):

```
int MPI_Send ( void * buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_COMM comm)
```

Το πρότυπο της συνάρτησης (MPI_Recv):

```
int MPI_Recv ( void *buf, int count, MPI_Datatype datatype,
int source, int tag, MPI_COMM comm)
```

Ανάπτυξη κώδικα

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

void main(int argc, char **argv)
{
    int j;
    int i;
    int size;
    int *data;
    int *data_local;
    int *numbersSend;

    int numbers;
    int number;
    int mod;
    int sum = 0;
    int flag = 0;

    int index;
    int error_index;
    int errors_index;
    int size_local;

    int my_rank, numtasks;
    int source, dest;
    int tag1 = 30; // size of local data
    int tag2 = 40; // data
    int tag3 = 50; // flag
    int tag4 = 60; // previous
    int tag5 = 70; // error

    int previous = 0;

    int rc;
    MPI_Status status;

    // start the parallelism
    rc = MPI_Init(&argc, &argv);
    if (rc != 0)
    {
        printf("MPI initialization error\n");
        MPI_Abort(MPI_COMM_WORLD, rc);
    }
    // function that returns the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    // function that returns the rank of processes
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank == 0)
    {
        printf("Give size of array: ");
        scanf("%d", &size);
```

```
        data = (int *)malloc(sizeof(int) * size);

        // Check if malloc was successfully completed
        if (!data)
        {
            printf("Failure");
            exit(0);
        }

        // User's input values in the array
        for (i = 0; i < size; i++)
        {
            printf("Give value for data[%d]: ", i);
            scanf("%d", (data + i));
        }

        // Create an array that keeps how many numbers should be given to
each process
        numbersSend = (int *)malloc(sizeof(int) * numtasks);
        // Check if malloc was successfully completed
        if (!numbersSend)
        {
            printf("Failure");
            exit(0);
        }

        // dividing how many numbers should be given to each process
        mod = size % numtasks;
        for (i = 1; i < numtasks; i++)
        {
            dest = i;

            if (i < mod)
            {
                *(numbersSend + i) = size / numtasks + 1;
            }
            else
            {
                *(numbersSend + i) = size / numtasks;
            }

            MPI_Send((numbersSend + i), 1, MPI_INT, dest, tag1,
MPI_COMM_WORLD);
        }

        // Αποστολή σε κάθε διεργασία διαφορετικά δεδομένα. (Διαμοίραση
δεδομένων)
        index = *(numbersSend + 1);

        for (i = 1; i < numtasks; i++)
        {
            dest = i;
            MPI_Send(&data[index - 1], 1, MPI_INT, dest, tag4,
MPI_COMM_WORLD);
            MPI_Send(&data[index], *(numbersSend + i), MPI_INT, dest, tag2,
MPI_COMM_WORLD);
```

```
            index += *(numbersSend + i);
        }

        if (0 < mod)
        {
            *numbersSend = size / numtasks + 1;
        }
        else
        {
            *numbersSend = size / numtasks;
        }

        number = *numbersSend;

        // Apothikeush sto data_local[] tou processor oi prwtoi
"arithmous_pou_tha_anatethoun_se_kathe_diergasia"
        data_local = (int *)malloc(sizeof(int) * number);
        // Check if malloc was successfully completed
        if (!data_local)
        {
            printf("Failure");
            exit(0);
        }
        for (i = 0; i < number; i++)
        {
            data_local[i] = data[i];
        }

        previous = *data_local;

        free(data);
    }
    else
    {
        MPI_Recv(&number, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);

        data_local = (int *)malloc(sizeof(int) * number);
        if (!data_local)
        {
            printf("Failure");
            exit(0);
        }
        MPI_Recv(&previous, 1, MPI_INT, 0, tag4, MPI_COMM_WORLD, &status);
        MPI_Recv(data_local, number, MPI_INT, 0, tag2, MPI_COMM_WORLD,
&status);

    }

    flag = 0;

    if (number == 1)
    {
        numbers = 2;
    }
    else
    {
```

```
        numbers = number;
    }

    for (i = 0; i < numbers - 1; i++)
    {
        if (previous <= data_local[i])
        {
            if ((sizeof(int) * number) != (sizeof(int)))
            {
                if (data_local[i] > data_local[i + 1])
                {
                    error_index = i;
                    if (my_rank != 0)
                        MPI_Send(&error_index, 1, MPI_INT, 0, tag5,
MPI_COMM_WORLD);

                    flag = 1;
                    break;
                }
            }
        }

        else
        {
            if (my_rank != 0)
            {
                error_index = -1;
                MPI_Send(&error_index, 1, MPI_INT, 0, tag5,
MPI_COMM_WORLD);
            }
            else
                error_index = i;

            flag = 1;
            break;
        }
    }

    if (my_rank == 0)
    {
        if (flag == 1)
        {
            printf("Array not sorted\n");
            printf("There's an error at data[%d]\n", error_index);
            free(data_local);
            free(numbersSend);
            MPI_Finalize();
            exit(0);
        }

        for (i = 1; i < numtasks; i++)
        {
            source = i;
            errors_index = 0;
            MPI_Recv(&flag, 1, MPI_INT, source, tag3, MPI_COMM_WORLD, &sta-
tus);
```

```
            if (flag == 1)
            {
                MPI_Recv(&error_index, 1, MPI_INT, source, tag5,
MPI_COMM_WORLD, &status);

                for (j = 0; j < source; j++)
                    errors_index = errors_index + numbersSend[i];

                errors_index = errors_index + error_index;

                printf("\nArray not sorted\n");
                printf("%d There's an error at data[%d]\n", source, er-
rors_index);
                break;
            }
        }

        if (flag == 0)
            printf("\nArray is sorted in ascending order\n");

        free(numbersSend);
        free(data_local);
    }
    else
    {
        MPI_Send(&flag, 1, MPI_INT, 0, tag3, MPI_COMM_WORLD);
        free(data_local);
    }

    MPI_Finalize();
}
```

Υλοποίηση κώδικα

(Δεν υλοποιήθηκε η επέκταση προγράμματος με την επιλογή μενού)

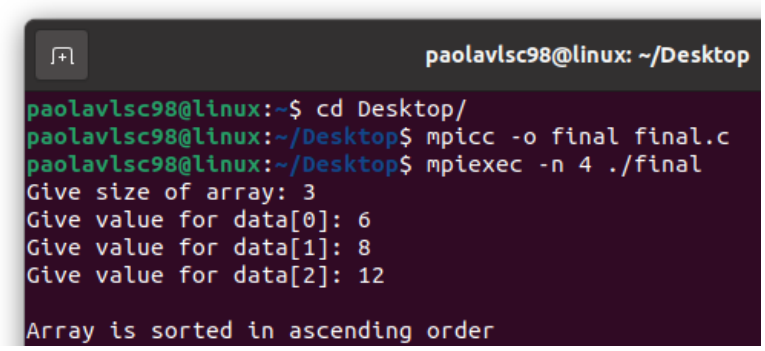
Η διεργασία 0 είναι υπεύθυνος για την εισαγωγή από το χρήστη του μήκους της ακολουθίας και η εισαγωγή των αριθμών σε πίνακα. Χρησιμοποιήθηκαν δυναμικοί πίνακας για την αποθήκευση των αριθμών αυτών. Κάθε διεργασία κάνει τον ίσο αριθμό συγκρίσεων όταν ο αριθμός των επεξεργαστών είναι ίσος με το μήκος, ενώ αν είναι μεγαλύτερος και μη πολλαπλάσιο χωρίζονται οι επιπλέον συγκρίσεις με το mod.

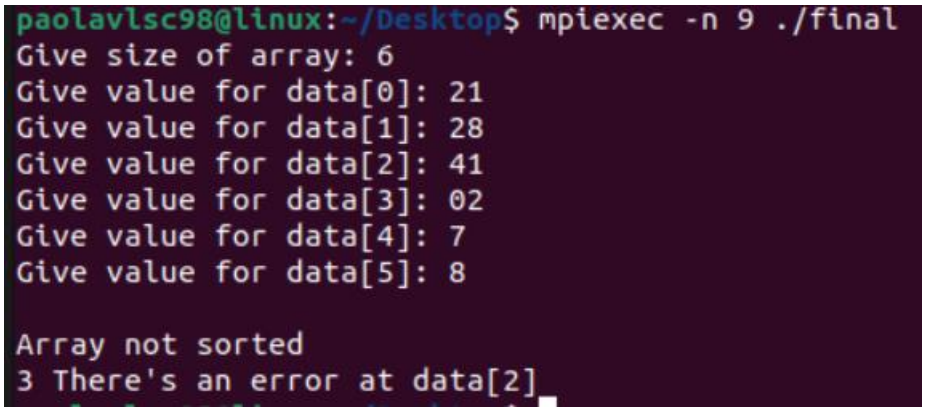
ΕΚΤΕΛΕΣΗ ΕΡΓΑΣΙΑΣ

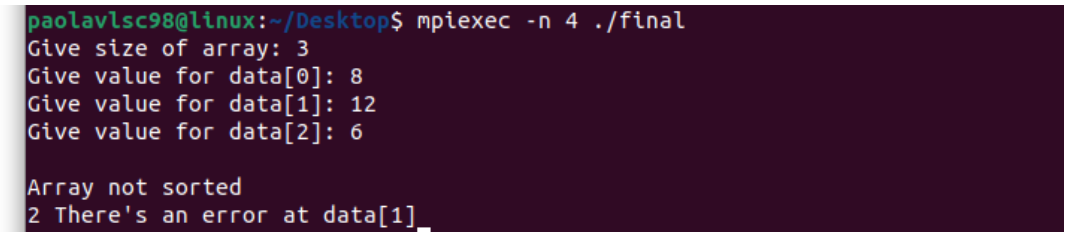
Ενδεικτικά τρεξίματα:

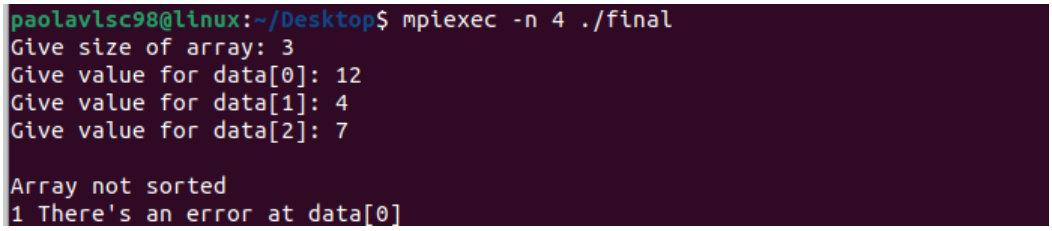
n : μήκος ακολουθίας
p : αριθμός επεξεργαστών/ διεργασιών

1. Για n < p









2. Για n > p

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 4 ./final
Give size of array: 15
Give value for data[0]: 1
Give value for data[1]: 2
Give value for data[2]: 5
Give value for data[3]: 8
Give value for data[4]: 3
Give value for data[5]: 0
Give value for data[6]: 41
Give value for data[7]: 5
Give value for data[8]: 7
Give value for data[9]:
-8
Give value for data[10]: 5
Give value for data[11]: 7
Give value for data[12]: 5
Give value for data[13]: 2
Give value for data[14]: 0

Array not sorted
1 There's an error at data[3]
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 3 ./final
Give size of array: 10
Give value for data[0]: 1
Give value for data[1]: 2
Give value for data[2]: 3
Give value for data[3]: 4
Give value for data[4]: 5
Give value for data[5]: 9
Give value for data[6]: 5
Give value for data[7]: 7
Give value for data[8]: 0
Give value for data[9]: 1

Array not sorted
2 There's an error at data[5]
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 3 ./final
Give size of array: 8
Give value for data[0]: 1
Give value for data[1]: 2
Give value for data[2]: 3
Give value for data[3]: 4
Give value for data[4]: 5
Give value for data[5]: 9
Give value for data[6]:
8
Give value for data[7]: 7

Array not sorted
2 There's an error at data[3]
```

3. Για n > k*p , όπου k ακέραιος αριθμός

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 4 ./final
Give size of array: 8
Give value for data[0]: 1
Give value for data[1]: 2
Give value for data[2]: 3
Give value for data[3]: 4
Give value for data[4]: 8
Give value for data[5]: 9
Give value for data[6]: 12
Give value for data[7]: 17

Array is sorted in ascending order
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 4 ./final
Give size of array: 8
Give value for data[0]: 6
Give value for data[1]: 4
Give value for data[2]: 2
Give value for data[3]: 7
Give value for data[4]: 8
Give value for data[5]: 2
Give value for data[6]:
5
Give value for data[7]: 7
Array not sorted
There's an error at data[0]
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 3 ./final
Give size of array: 6
Give value for data[0]: 4
Give value for data[1]: 1
Give value for data[2]: 8
Give value for data[3]: 7
Give value for data[4]: 4
Give value for data[5]: 2
Array not sorted
There's an error at data[0]
```

4. Για n = p

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 4 ./final
Give size of array: 4
Give value for data[0]: 1
Give value for data[1]: 2
Give value for data[2]: 3
Give value for data[3]: 4

Array is sorted in ascending order
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 4 ./final
Give size of array: 4
Give value for data[0]: 8
Give value for data[1]: 5
Give value for data[2]: 4
Give value for data[3]: 6

Array not sorted
1 There's an error at data[0]
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 4 ./final
Give size of array: 4
Give value for data[0]: -8
Give value for data[1]: 9
Give value for data[2]: 2
Give value for data[3]: 1

Array not sorted
2 There's an error at data[1]
```

```
paolavlsc98@linux:~/Desktop$ mpiexec -n 9 ./final
Give size of array: 9
Give value for data[0]: 7
Give value for data[1]:
2
Give value for data[2]: 2
Give value for data[3]: 6
Give value for data[4]: 7
Give value for data[5]: 4
Give value for data[6]: 01
Give value for data[7]: 5
Give value for data[8]: 2

Array not sorted
1 There's an error at data[0]
```