

ΚΑΤΑΝΕΜΗΜΕΝΑ ΚΑΙ ΔΙΑΔΙΚΤΥΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΑΝΑΦΟΡΑ 1^{ης} ΕΡΓΑΣΙΑΣ

Ονοματεπώνυμο	ΑΕΜ
Κωνσταντίνος Τσιγκίνης	2452
Νικόλαος Καράμπινας	2355
Αριστείδης Νούλης	2390

Διεύθυνση αποθετηρίου git:

<https://github.com/anoulis/KDSproject01>

● Αρχεία κώδικα και Επεξηγήσεις

Αρχείο generator.c

Σε αυτό το κομμάτι κώδικα γίνεται ουσιαστικά η αρχή της υλοποίησης του προγράμματος. Συγκεκριμένα, γίνεται η εγγραφή των τυχαίων συντεταγμένων που δημιουργήσαμε, σε ένα ξεχωριστό αρχείο.

Αρχείο examine.c

Με αυτό το κομμάτι γίνεται κατ' επέκταση η συνέχεια του προγράμματος, αφού σ' αυτό το σημείο ο κώδικάς μας διαβάζει τα δεδομένα που δημιουργήσαμε πιο πριν. Γίνεται η χρήση 2 συναρτήσεων, όπου η πρώτη κάνει τις συγκρίσεις μεταξύ των συντεταγμένων και η δεύτερη μετράει το χρόνο εκτέλεσης. Τέλος, και οι δυο τυπώνουν τα αποτελέσματά τους

Αρχείο `examineOPENMP.c*`

Σε αυτό το μέρος γίνεται η προσέγγιση του openMP στο πρόγραμμα. Παραλληλοποιείται το κομμάτι κώδικα που είναι υπεύθυνο για το διάβασμα και τη σύγκριση των συντεταγμένων έτσι ώστε να συμπεράνουμε το ποσοστό που είναι εντός των ορίων.

```
#pragma omp nowait parallel for firstprivate(size) reduction(+:count1) reduction(+:count2)
for(i=0;i<size;i++)
{ if(buffer[i]!=' ' && buffer[i]!='\n')
{
    c[k]=buffer[i];
    k++;
}
else
{
    x=atof(c);
    if (x>=limit1 && x<=limit2)
        count2++;
    count1++;
    k=0;
}
}
```

Αρχείο `examineOPENMPI.c*`

Σε αυτό το κομμάτι γίνεται η πρόσθεση του openMPI στο πρόγραμμα. Μετά την αρχικοποίηση μέσω της συνάρτησης `MPI_Init()`, δημιουργούμε το `bufsize` όπου χωρίζουμε το διάβασμα του αρχείου ανά σειρές ανάλογα του αριθμού των διεργασιών. Στη συνέχεια, με τις συναρτήσεις `MPI_Reduce()` αθροίζουμε συνολικά το πλήθος των συντεταγμένων και των αριθμών που βρίσκονται μέσα στα όρια. Τέλος, τερματίζουμε την MPI λειτουργία με τη συνάρτηση `MPI_Finalize()`.

* Τα αρχεία βρίσκονται στο git και αφορούν μόνο τις υλοποιήσεις openMP και openMPI.

Bufsize και `MPI_Reduce()`:

```
MPI_Reduce(&count1, &count1_gloabl, 1, MPI_INT, MPI_SUM, 0,MPI_COMM_WORLD);
MPI_Reduce(&count2, &count2_gloabl, 1, MPI_INT, MPI_SUM, 0,MPI_COMM_WORLD);

MPI_Finalize();
```

```

if(number%size==0)
{ bufsize=(number/size)*30;
  if(rank==size-1)
  {
    start2 = rank * bufsize;
    end2   = start2 + bufsize;
  }
  else
  {
    start2 = rank * bufsize;
    end2   = start2 + bufsize - 1;
  }
}
else
{
  if(rank==size-1)
  {
    bufsize=((number-(number%size))/size)*30+(number%size)*30;
    start2 = rank * ((number-(number%size))/size)*30;
    end2   = start2 + bufsize ;
  }
  else
  {
    bufsize=((number-(number%size))/size)*30;
    start2 = rank * bufsize;
    end2   = start2 + bufsize - 1;
  }
}
}

```

Αρχείο examine_parallel.c

Σε αυτό το σημείο γίνεται ο συνδυασμός του openMP μαζί με το openMPI στον κώδικα. Επίσης γίνεται η τοποθέτηση πέντε νέων συναρτήσεων έτσι ώστε να γίνεται ο έλεγχος των ορισμάτων. Η πρώτη λέγεται conflictstest() και φορά τον έλεγχο του πρώτου ορίσματος δηλαδή τον αριθμό συντεταγμένων. Η δεύτερη λέγεται timetest() και αφορά τον έλεγχο του δεύτερου ορίσματος δηλαδή τον χρόνο εκτέλεσης. Η τρίτη ονομάζεται filetest() και αφορά τον έλεγχο του τρίτου ορίσματος δηλαδή την ύπαρξη του αρχείου. Η τέταρτη λέγεται threadstest() και αφορά την εκτέλεση του προγράμματος και τον αριθμό των threads. Τέλος, η τελευταία συνάρτηση είναι η procstest() και αφορά των αριθμό των processes που θα δημιουργηθούν στην εκτέλεση του προγράμματος.

Η συνάρτηση threadstest():

```

int threadstest(int threads)
{
  if(threads==-1)
    return omp_get_max_threads();
  else if(threads==0 || threads<0)
  {printf("The program will exit \nbecause the number of threads is not right.\n");
    exit(0);
  }
  else
  { if(threads<=omp_get_max_threads())
    {omp_set_dynamic(0);
     omp_set_num_threads(threads);
     return threads;
    }
    else
      return omp_get_max_threads();
  }
}

```

● Γραφήματα



