PROIECT APGA

-TRIUNGHIUL LUI PASCAL-

SERGIU JURAVLE,
GR. 3711A

1. Introducere

1.1 Tema Proiectului

Realizarea unui program in limbajul C cu scopul de a aplica elemente de programare paralelă pe o structură paralelă de calcul, folosind librăria MPI. Programul ilustrează aceste elemente prin implementarea Triunghiului lui Pascal.

1.2 Descrierea Temei

Triunghiul lui Pascal este un aranjament geometric al coeficienților binomiali, numit astfel în onoarea matematicianului francez Blaise Pascal. Înălțimea și laturile triunghiului conțin cifra 1, iar fiecare număr de pe o linie *n* reprezintă suma celor 2 numere de pe linia superioară *n-1*.

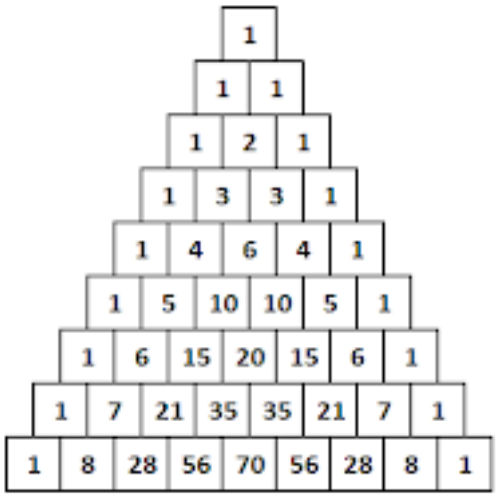

FIG.1.1 TRIUNGHIUL LUI PASCAL

Elementul de pe n rând și k coloană se notează cu (n, k). Folosind această notație putem descrie un element din triunghi astfel: (n, k) = (n – 1, k – 1) + (n – 1, k).

Putem găsi șirul lui Fibonacci dacă adunăm numerele ce formează diagonalele cu o pantă mai lină față de cele normale.

2. Librăria MPI

MPI (Message Passing Interface) este un standard pentru comunicarea prin mesaje, elaborat de MPIForum. În definirea lui au fost utilizate caracteristicile cele mai importante ale unor sisteme anterioare, bazate pe comunicația de mesaje.

2.1 Funcții MPI

Trimiterea și recepția unui mesaj sunt două concepte fundamentale în librăria MPI, acestea sunt reprezentate de următoarele funcții:

- int MPI_Send(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);

Operațiile MPI_Send()/MPI_Recv() sunt activități blocante, însemnând ca nu redau controlul apelantului până când mesajul nu a fost preluat din tamponul sursă sau încă nu fost recepționat.

Funcțiile de trimitere și recepție funcționează în felul următor, din procesul A se trimit date către procesul B, aceste date sunt împachetate într-un singur mesaj și stocate într-o zonă tampon de unde dispozitivul de comunicare este responsabil pentru a trimite mesajul către locația corectă. Locația mesajului este definită de către rangul procesului. Chiar dacă mesajul este transmis către B, procesul B trebuie să fie deschis la cereri de recepție pentru a primi datele din procesul A. Odată ce procesul B realizează acest lucru, datele au fost transmise cu succes.

În aceste funcții primul argument buf reprezintă datele din zona tampon ce vor fi trimise/recepționate. Al doilea și al treilea argument descriu numărul și tipul elementelor din zona tampon. Al patrulea și al cincilea element specifică rangul procesului și eticheta mesajului. Al șaselea argument specifică comunicantul, iar ultimul (numai la MPI_Recv) prezintă informații despre mesajul primit.

Pe lângă operațiile de trimitere/recepție mai sunt funcțiile auxiliare, care au următorul prototip:

- MPI_Init(...). Rolul acesteia este de a inițializa “mediul” în care programul va fi executat. Ea trebuie executată o singură dată, înaintea altor operații MPI.

- MPI_Initialized(...). Pentru a evita execuția sa de mai multe ori (și deci provocarea unei erori), MPI oferă posibilitatea verificării dacă MPI_Init() a fost sau nu executată. Acest lucru este făcut prin. int MPI_Initialized(int *flag);

- MPI_Finalize(...).Perechea funcției de inițializare este MPI_Finalize(), care trebuie executată de fiecare proces, pentru a "închide" mediul MPI. Forma acestei operații este următoarea. int MPI_Finalize(void);

- MPI_Comm_rank (...). Este funcția poate afla poziția unui proces în grupul asociat unui comunicator (obține rank-ul unui proces). int MPI_Comm_rank(MPI_Comm comm, int *rank);

- MPI_Comm_size (...). Este funcția ce poate furniza numărul total de procese, din grupul asociat unui comunicator (obține rank-ul unui proces). int MPI_Comm_size(MPI_Comm comm, int *count);

- MPI_Get_count(…), întoarce lungimea mesajului recepționat: int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *cnt);

3. Programare

Programul începe prin preluarea argumentelor din linia de comandă și inițializează mediul MPI cu aceste argumente, apoi utilizatorul trebuie să introducă numărul de rânduri dorit pentru a calcula coeficienții binomiali (FIG 3.1). Rank-ul procesului va fi preluat folosind funcția MPI_Comm_rank, iar numărul de procese va fi preluat cu MPI_Comm_size.

```
13
14 void main(int argc, char** argv)
15 {
16     int rank;
17     int nrNoduri;
18     int nrRanduri;
19
20     MPI_Init(&argc, &argv);
21     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
22
23     if(rank == 0)
24     {
25         MPI_Comm_size(MPI_COMM_WORLD, &nrNoduri);
26         printf("\nNumarul de noduri este %d.\n\n", nrNoduri);
27
28         printf("Introdu numarul de randuri: ");
29         scanf("%d", &nrRanduri);
30         printf("\n");
31
32         startMaster(nrRanduri, nrNoduri);
33     }
34     else
35     {
36         startSlave(rank);
37     }
38
39     MPI_Finalize();
40 }
41
```

FIG.3.1 FUNCȚIA MAIN

Funcția startMaster(int n, int m) este apelată pentru procesul master, cel cu rank-ul 0. Aceasta are rolul de a calcula numărul de rânduri pe nod (numărul de noduri trebuie sa fie divizibil cu numărul de rânduri). După aflarea numărului de rânduri pe nod, fiecărui proces rămas îi va fi transmis un mesaj conținând un șir de numere reprezentând numărul de rânduri urmat rândurile respective. Rândurile au fost distribuite într-un mod cât de cât egal, folosind formula (poziția rândului din șir * numărul de procese + poziția procesului) (FIG 3.2 FUNCȚIA MASTER (SEND)).

```
42  void startMaster(int n, int m)
43  {
44      MPI_Status status;
45      int nrRanduriPerNod = n / (m - 1);
46
47      for(int i = 1; i < m; i++)
48      {
49          int randuriVec[nrRanduriPerNod + 1];
50
51          for(int j = 0; j < nrRanduriPerNod; j++)
52          {
53              randuriVec[j + 1] = j * (m - 1) + (i - 1);
54          }
55
56          randuriVec[0] = nrRanduriPerNod;
57          char* mesaj = transVecInSir(randuriVec, nrRanduriPerNod + 1);
58
59          MPI_Send(randuriVec, 20, MPI_INT, i, 99, MPI_COMM_WORLD);
60          printf("[MASTER] Am trimis randurile: '%s' catre SLAVE[%d].\n", mesaj, i);
61          free(mesaj);
62      }
63
64      printf("\n");
```

FIG.3.2 FUNCȚIA MASTER (SEND)

Funcția startSlave(int m) este apelată pentru procesele cu rank mai mare ca 0. Aceasta are rolul de a calcula elementele de pe fiecare rând primit de la master, apoi trimite la master un mesaj conținând un șir cu numărul de elemente și elementele respective. Elementele din rânduri sunt calculate cu formula următoare: factorial(numărul rândului) / (factorial(poziția elementului în rând) / factorial(rândul respectiv – poziția elementului în rând)).
(FIG 3.3 FUNCȚIA SLAVE)

```
106 void startSlave(int m)
107 {
108     MPI_Status status;
109     int randuriVec[20];
110     MPI_Recv(randuriVec, 20, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
111
112     int n = randuriVec[0];
113     char* mesaj = transVecInSir(randuriVec, n + 1);
114     printf("[SLAVE-%d] Am primit randurile: '%s'.\n", m, mesaj);
115
116     int rezVec[100];
117
118     int lungCurenta = 1;
119
120     for(int i = 1; i < n + 1; i++) {
121
122         for(int j = 0; j <= randuriVec[i]; j++)
123         {
124             rezVec[lungCurenta++] = factorial(randuriVec[i]) / (factorial(j) * factorial(randuriVec[i] - j));
125         }
126     }
127
128     rezVec[0] = lungCurenta - 1;
129
130     MPI_Send(rezVec, 100, MPI_INT, 0, 99, MPI_COMM_WORLD);
131     mesaj = transVecInSir(rezVec, lungCurenta);
132     printf("[SLAVE-%d] Am trimis urmatoarele valori: '%s'\n\n", m, mesaj);
133     free(mesaj);
134 }
```

FIG.3.3 FUNCȚIA SLAVE

Revenind în funcția master, rezultatele calculate de procesele slave sunt recepționate și sunt salvate într-un singur șir, fiecare element pe poziția lui în triunghi. Șirul rezultant este apoi afișat într-o formă potrivită noțiunii temei.

```
66      int totalVec[500];
67      int totalLung = 0;
68
69      for(int i = 1; i < m; i++)
70      {
71          int rezVec[100];
72          MPI_Recv(rezVec, 100, MPI_INT, i, 99, MPI_COMM_WORLD, &status);
73
74          int lung = rezVec[0];
75          totalLung += lung;
76
77          char* mesaj = transVecInSir(rezVec, lung + 1);
78          printf("[MASTER] Am primit valorile '%s' de la SLAVE[%d].\n", mesaj, i);
79          free(mesaj);
80
81          int nrElem = 0;
82
83          for(int rand = 0; rand < nrRanduriPerNod; rand++) {
84              int r = rand * (m - 1) + (i - 1);
85              int startPos = 0;
86
87              if (r > 0) {
88                  startPos = r * (r + 1) / 2;
89              }
90
91              for (int j = startPos; j <= startPos + r; j++)
92              {
93                  totalVec[j] = rezVec[nrElem + j - startPos + 1];
94              }
95
96              nrElem += r + 1;
97          }
98      }
99
100     printf("\n");
101     char* mesaj = transVecInSir(totalVec, totalLung);
102     printf("%s\n\n", mesaj);
103     afiseazaRezultat(totalVec, n);
104 }
105
```

FIG.3.4 FUNCȚIA MASTER (RECEIVE)

4. Manual de utilizare

4.1 Compilare

Pentru a compila programul rulați următoarea instrucțiune:

```
student_09@master:~/cloud/Juravle_Sergiu/Proiect$ mpicc TriunghiulLuiPascal.c -o TriunghiulLuiPas
cal.out
```

4.2 Execuție

Pentru a executa programul rulați următoarea instrucțiune:

```
student_09@master:~/cloud/Juravle_Sergiu/Proiect$ mpirun -np 5 ./TriunghiulLuiPascal.out
```

Programul executat cu 8 rânduri și 4 noduri slave. (FIG.4.1 PROGRAM RUN)

```
student_09@master:~/cloud/Juravle_Sergiu/Proiect$ mpirun -np 5 ./TriunghiulLuiPascal.out

Numarul de noduri este 5.

Introdu numarul de randuri: 8

[MASTER] Am trimis randurile: '[2,0,4]' catre SLAVE[1].
[MASTER] Am trimis randurile: '[2,1,5]' catre SLAVE[2].
[MASTER] Am trimis randurile: '[2,2,6]' catre SLAVE[3].
[MASTER] Am trimis randurile: '[2,3,7]' catre SLAVE[4].

[SLAVE-2] Am primit randurile: '[2,1,5]'.
[SLAVE-2] Am trimis urmatoarele valori: '[8,1,1,1,5,10,10,5,1]'

[SLAVE-3] Am primit randurile: '[2,2,6]'.
[SLAVE-3] Am trimis urmatoarele valori: '[10,1,2,1,1,6,15,20,15,6,1]'

[SLAVE-1] Am primit randurile: '[2,0,4]'.
[SLAVE-1] Am trimis urmatoarele valori: '[6,1,1,4,6,4,1]'

[SLAVE-4] Am primit randurile: '[2,3,7]'.
[SLAVE-4] Am trimis urmatoarele valori: '[12,1,3,3,1,1,7,21,35,35,21,7,1]'

[MASTER] Am primit valorile '[6,1,1,4,6,4,1]' de la SLAVE[1].
[MASTER] Am primit valorile '[8,1,1,1,5,10,10,5,1]' de la SLAVE[2].
[MASTER] Am primit valorile '[10,1,2,1,1,6,15,20,15,6,1]' de la SLAVE[3].
[MASTER] Am primit valorile '[12,1,3,3,1,1,7,21,35,35,21,7,1]' de la SLAVE[4].

[1,1,1,1,2,1,1,3,3,1,1,4,6,4,1,1,5,10,10,5,1,1,6,15,20,15,6,1,1,7,21,35,35,21,7,1]

       1
      1 1
     1 2 1
    1 3 3 1
   1 4 6 4 1
  1 5 10 10 5 1
 1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
student_09@master:~/cloud/Juravle_Sergiu/Proiect$
```

FIG.4.1 PROGRAM RUN

5. Rezultate

	Master	Slave 1	Slave 2	Slave 3	Slave 4
4 randuri	0.0286335945	0.0000009537	0.0000011921	0.0000009537	0.0000016689
8 randuri	0.0224308968	0.0000009537	0.0000009537	0.0000014305	0.0000019073

6. Bibliografie

1. https://en.wikipedia.org/wiki/Pascal%27s_triangle
2. mpitutorial.com/tutorials/
3. http://www.cplusplus.com/doc/tutorial/