UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALITATEA INFORMATICA

LUCRARE DE LABORATOR

la disciplina „Programarea paralelă și distribuită”

Lucrarea de laborator nr. 1: Un algoritm paralel pentru soluționarea jocurilor bimatriceale în strategii pure

Verificat: Hâncu Boris, dr., conf. univ.

Efectuat: Cemîrtan Cristian, studentul grupei I2101

CHIȘINĂU – 2023

Cuprins

Formularea problemei

Să se realizeze un algoritm paralel pentru determinarea situațiilor de echilibru, cu următoarele condiții:

1. Pentru orice proces t se determină submatricele $SubA^t$ și $SubB^t$;
2. Fiecare proces determină elementele maximale după linie și după coloane. Pentru aceasta se va folosi funcția MPI_Reduce și operația ALLMAXLOC creată cu funcția MPI_Op_create;
3. Procesul cu rankul 0 va determina mulțimea de situații Nash de echilibru.

Realizarea algoritmului

1. Paralelizarea la nivel de date se realizează în următoarele moduri:
    a. Procesul cu rankul 0 inițializează valorile matricelor A și B, construiește submatricele $SubA^t$, $SubB^t$ (fragment din matricea B transpusă) și le distribuie tuturor proceselor mediului de comunicare.
        - Lungimea unei submatrice este egală cu numărul de rânduri împărțit cu numărul de procese. De exemplu, avem o matrice cu 6 linii și avem 3 procese. Fiecare proces va primi câte 3 linii (submatrice 3xn) dintr-o matrice.

P0
P0
P0
P1
P1
P1

        i. Procesul cu rankul 0 inițializează valorile matricelor A și B (valorile sunt citite dintr-un fișier);
        ii. Procesul cu rankul 0 transpune matricea B;
        iii. Procesul cu rankul 0 distribuie lungimea și lățimea matricelor, la celelalte procese (MPI_Bcast);
        iv. Fiecare proces t își calculează lungimile pentru submatricele $SubA^t$ și $SubB^t$. Apoi, le alochează. Aceste lungimi apoi devin distribuite procesului cu rankul 0 (MPI_Gather);
        v. Fiecare proces t obține valorile pentru submatricele $SubA^t$ și $SubB^t$ de la procesul cu rankul 0 (MPI_Scatterv).
    b. Fiecare proces din mediul de comunicare construiește submatricele $SubA^t$, $SubB^t$ și le inițializează cu valori.
        i. Fiecare proces t își calculează lungimile pentru submatricele $SubA^t$ și $SubB^t$. Apoi, le alochează;
        ii. Fiecare proces își inițializează submatricele cu valori (valorile sunt citite dintr-un fișier). Valorile ce nu se încadrează în submatricele respective devin ignorate.
    c. Distribuirea matricelor pe procese se face astfel încât să se realizeze principiul load balancing.
        i. Distribuirea va fi realizată sub principiul algoritmului pseudo-2D ciclic (grila de 1xn procese). De exemplu, avem o matrice 4x4 cu 6 procese. Fiecare proces va primi câte un element din matrice, anexat cu poziția sa carteziană.

P0	P1	P2	P3
P4	P5	P0	P1
P2	P3	P4	P5
P0	P1	P2	P3

P0	(0,0)	(1,2)	(3,0)
P1	(0,1)	(1,3)	(3,1)
P2	(0,2)	(2,0)	(3,2)
P3	(0,3)	(2,1)	(3,3)
P4	(1,0)	(2,2)	
P5	(1,1)	(2,3)	

ii. În cazul distribuirii unui matrice de la procesul cu rankul 0, acest lucru se realizează cu funcțiile pentru comunicarea colectivă:
   - MPI_Gather: se colectează câte elemente va primi fiecare proces;
   - MPI_Scatterv: procesul cu rankul 0 expediază la fiecare proces rândul său.

2. Paralelizarea la nivel de operații se realizează:
   a. Prin utilizarea funcției MPI_Reduce și a operațiilor nou create.
      i. Se creează un nou tip de date floatIntArray, ce reprezintă un masiv din n elemente de tip MPI_FLOAT_INT, unde n este numărul de rânduri a matricei A sau a matricei B transpuse. Rezum că tipul de date reprezintă lista elementelor maxime de pe o coloană;
      ii. Se creează noua operație ALLMAXLOC:
         - Parametrii: avem tabloul de intrare (fie input) și unul de intrare/ieșire (fie output). Ambele tablouri au lungimi egale, și fiecare element din ele au tipul de date menționat în punctul precedent.
         - Pentru i = 0 până la lungimea tabloului:
            a. Se compară valoarea din input[i] cu cea din output[i]:
               i. Dacă prima valoare este mai mare decât a doua:
                  1. Se curăță tabloul al doilea, și se copie în el elementele din primul tablou;
               ii. Dacă prima valoare este egală cu a doua:
                  1. Se adaugă în tabloul al doilea elementele din primul tablou.
      iii. Temporar creăm încă un nou tip de date tempFloatIntArray, ce reprezintă un masiv din n^t elemente de tip MPI_FLOAT_INT, unde n^t este numărul de rânduri a submatricei $SubA^t$ sau $SubB^t$.
      iv. Procesele „umflă” submatricele SubAt și SubBt, unde fiecare celulă din submatrice devine transformată în tipul de date tempFloatIntArray (celula ca primul element). Efectiv transformăm matricea în 3D. După reducerea locală,

se re-umflă în tipul de date floatIntArray, pentru reducerea paralelă. **Aceasta strategie este necesară pentru a implementa operația ALLMAXLOC fără trișări (să nu facem operațiile suplementare după MPI_Reduce, vrem deodată să obținem rezultatul după apelarea operației de reducere).**

v. Fiecare proces își reduce local submatricele umflate, obținând un singur tablou cu elementele maxime de pe fiecare coloană;

vi. Fiecare proces își distribuie tablourile reduse între ele, și rezultatul final va avea procesul cu rankul 0 (MPI_Reduce);

- De exemplu, realizăm acele pași pentru o matrice 4x4 cu 2 procese:

P0	1	4	7
	1	4	3
P1	0	0	7
	2	4	3

P0	[(1,0), ?]	[(4,0), ?]	[(7,0), ?]
	[(1,1), ?]	[(4,1), ?]	[(3,1), ?]
P1	[(0,2), ?]	[(0,2), ?]	[(7,2), ?]
	[(2,3), ?]	[(4,3), ?]	[(3,3), ?]

P0	[(1,0), (1,1)]	[(4,0), (4,1)]	[(7,0), ?]
P1	[(2,3), ?]	[(4,3), ?]	[(7,2), ?]

P0	[(1,0), (1,1), ?, ?]	[(4,0), (4,1), ?, ?]	[(7,0), ?, ?, ?]
P1	[(2,3), ?, ?, ?]	[(4,3), ?, ?, ?]	[(7,2), ?, ?, ?]

P0	[(2,3), ?, ?, ?]	[(4,0), (4,1), (4,3), ?]	[(7,0), (7,2), ?, ?]

vii. Procesul cu rankul 0 determină situațiile Nash de echilibru prin intersectarea rândurilor reduse pentru submatricele respective.

b. Nu se utilizează funcția MPI_Reduce (master-slave).

i. Fiecare proces își determină local elementele maxime de pe coloane/linii din submatrici. Apoi, le stochează într-un tablou;

ii. Fiecare proces își distribuie tablourile procesului cu rankul 0 (MPI_Gatherv);

iii. Pentru fiecare tablou recepționat de la celelalte procese, procesul cu rankul 0 efectuează pasul întâi.

3. Să se realizeze o analiză comparativă a timpului de execuție a programelor realizate privitor la paralelizarea la nivel de date și la nivel de operații.

a. Analiza comparativă va fi efectuată pe baza matricelor din exemplul 6. 1.

$$A = \begin{pmatrix} 400 & 0 & 0 & 0 & 0 & 0 \\ 300 & 300 & 0 & 0 & 0 & 0 \\ 200 & 200 & 200 & 0 & 0 & 0 \\ 100 & 100 & 100 & 100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -100 & -100 & -100 & -100 & -100 & -100 \end{pmatrix}. \quad B = \begin{pmatrix} 0 & 200 & 100 & 0 & -100 & -200 \\ 0 & 0 & 100 & 0 & -100 & -200 \\ 0 & 0 & 0 & 0 & -100 & -200 \\ 0 & 0 & 0 & 0 & -100 & -200 \\ 0 & 0 & 0 & 0 & 0 & -200 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Paralelizarea la nivel de date/operații	Timpul de execuție la nivel de date/operații (secunde)		
	2 procese	4 procese	8 procese
a) a)	0.000672886 - 0.000771458	0.0.0008608 – 0.000187928	0.00103827 – 0.000124335
a) b)	0.000590425 – 0.00014703	0.00074361 – 0.000239393	0.00160734 – 0.00125363
b) a)	0.000703824 – 0.0001187898	0.000948384 – 0.000210447	0.00101282 – 0.000184247
b) b)	0.000663959 – 0.00016923	0.000872521 – 0.000277246	0.000982388 – 0.000338286
c) a)	0.00128344 – 7.9639e-05	0.000874284 – 0.000182795	0.000929187 – 0.000218506
c) b)	0.000381617 – 0.000114901	0.000791334 – 0.000213794	0.00106208 – 0.000284988
Rezultatul	**(2, 2), (3, 3), (4, 4)**		

Pe o matrice 500x500 cu 55 situații de echilibru:

Paralelizarea la nivel de date/operații	Timpul de execuție la nivel de date/operații (secunde)		
	16 procese	32 procese	48 procese
a) a)	0.2111171 - 0.052345	0.243148 – 0.0535603	0.253092 – 0.0418813
a) b)	0.19457 – 0.107431	0.315245 – 0.172174	0.297962 – 0.118244
b) a)	0.173171 – 0.00938866	0.3017 – 0.0118887	0.273364 – 0.0130051
b) b)	0.188924 – 0.0713894	0.327413 – 0.131977	0.300369 – 0.0327625
c) a)	0.227386 – 2.69776	0.2821 – 4.34713	0.295263 – 4.60589
c) b)	0.225276 – 0.0148161	0.323232 – 0.0149809	0.313213 – 0.0149187
Rezultatul	**(35, 489), (37, 132), (40, 229), (61, 284), (62, 41), (67, 424), (69, 321), (77, 243), (97, 42), (106, 313), (107, 44), (116, 425), (120, 429), (127, 297), (128, 388), (133, 362), (147, 217), (153, 381), (155, 166), (156, 361), (161, 169), (163, 64), (165, 110), (200, 318), (225, 119), (232, 322), (246, 160), (247, 97), (261, 206), (263, 326), (266, 297), (276, 364), (293, 365), (301, 245), (317, 383), (318, 185), (321, 307), (340, 81), (352, 141), (354, 407), (358, 210), (362, 10), (362, 103), (397, 154), (407, 390), (416, 27), (418, 319), (423, 95), (428, 487), (434, 378), (440, 190), (444, 31), (457, 260), (493, 211), (498, 28)**		

Fișierele .cpp

- allmaxloc.h:
    - FloatIntArrayAllMaxLoc – funcția utilizată pentru crearea operației MPI_ALLMAXLOC;
    - ProcessMatrix – paralelizarea la nivel de operații prevăzută în punctul a).
- independent.h
    - CalculatePartData – calcularea dimensiunii submatricei;
    - ReadMatrix – paralelizarea la nivel de date prevăzută în punctul b);
    - ProcessMatrixSerial – paralelizarea la nivel de operații prevăzută în punctul c).
- loadbalancing.h
    - ReadMatrixLB – paralelizarea la nivel de date prevăzută în punctul b) împreună cu punctul c);
    - DistributeMatrixLB – paralelizarea la nivel de date prevăzută în punctul a) împreună cu punctul c);
- masterslave.h
    - DistributeMatrix – paralelizarea la nivel de date prevăzută în punctul a);
    - PrepareMatrices – funcția ce selectează paralelizarea la nivel de date, conform cerinței utilizatorului.
- header.h
    - Am importat bibliotecile, am creat macro-definiții și variabile globale.
- lab1.cpp
    - Programul întreg ce implementează cerințele sarcinii.

Rezultatele rulării programului

```
/home/I01/CemirtanCristian/6$ cat mtx2.txt
3 3
2 0 1
1 2 0
0 1 2

1 0 2
2 1 0
0 2 1
/home/I01/CemirtanCristian/6$ mpiexec -host compute-1-0,compute-1-1 a.out mtx2.txt stat
Situatiile Nash de echilibru:
Nu exista nici una.
/home/I01/CemirtanCristian/6$ cat mtx3.txt
4 4
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
/home/I01/CemirtanCristian/6$ mpiexec -host compute-1-0,compute-1-1 a.out mtx3.txt
serial stat
Situatiile Nash de echilibru:
(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2),
(2, 3), (3, 0), (3, 1), (3, 2), (3, 3),
Total: 16
/home/I01/CemirtanCristian/6$ cat mtx4.txt
4 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
/home/I01/CemirtanCristian/6$ mpiexec -host compute-1-0,compute-1-1 a.out mtx4.txt
independent stat
Situatiile Nash de echilibru:
(0, 0), (1, 1), (2, 2), (3, 3),
Total: 4
/home/I01/CemirtanCristian/6$ cat mtx6.txt
2 10
1 2 3 4 5 6 7 8 9 10
9 8 7 6 5 4 3 2 1 0

10 9 8 7 99 5 4 3 2 1
0 1 2 3 99 5 6 7 8 9
/home/I01/CemirtanCristian/6$ mpiexec -host compute-1-0,compute-1-1 a.out mtx6.txt lb
stat
Situatiile Nash de echilibru:
(0, 4), (1, 4),
Total: 2
```