STUDIA UNIVERSITATIS MOLDAVIAE, 2020, nr.7(137)
Seria "Științe exacte și economice" ISSN 1857-2073 ISSN online 2345-1033 p.45-63

CZU: 519.83:004.42 **DOI: http://doi.org/10.5281/zenodo.4457508**

UN ALGORITM PARALEL DE SOLUȚIONARE A JOCURILOR BIMATRICEALE FOLOSIND SISTEMUL MATEMATICA

Boris HÂNCU, Ionel ANTOHI

Universitatea de Stat din Moldova

În articol se face o analiză a posibilităților sistemului de calcul simbolic Matematica pentru elaborarea programelor paralele pe sistemul de calcul paralel de tip DMM (clastere). Este elaborat un algoritm paralel pentru determinarea situațiilor Nash de echilibru în strategii pure pentru jocurile bimatriceale. Pentru acest algoritm sunt elaborate programe paralele utilizând sistemul Matematica și modele de programare MPI, în care se realizează diferite modalități de distribuire a calculelor pe nuclee și diferite modalități de paralelizare la nivel de date. Se realizează o analiză comparativă a timpului de calcul pentru programele elaborate.

Cuvinte-cheie: *Wolfram Matematica, algoritmi paraleli, teoria jocurilor, situații Nash de echilibru, funcții MPI, timp de calcul.*

A PARALLEL ALGORITHM FOR SOLVING BIMATRIX GAMES USING THE MATHEMATICS SYSTEM

The article makes an analysis of the possibilities of the symbolic calculation system Mathematics for the elaboration of parallel programs on the DMM type parallel system. A parallel algorithm is developed for determining Nash equilibrium profiles in pure strategies for bimatrix games. For this algorithm, parallel programs are developed using the Mathematics system and MPI programming models, in which different ways of distributing the calculations on cores and different ways of parallelization at the data level are performed. A comparative analysis of the calculation time for the developed programs is performed.

Keywords: *Wolfram Mathematic, parallel algorithm, game theory, Nash equilibrium profiles, MPI functions, time complexity.*

Introducere

La soluționarea în timp real a diferitor probleme practice cu volum foarte mare de date sunt utilizate diferite sisteme de calcul paralel. Pentru implementarea soft a algoritmilor paraleli pe sisteme de calcul cu memorie distribuită (clastere paralele) se pot utiliza diferite paradigme şi modele de programare. Din multitudinea de modele de programare paralelă în acest articol sunt utilizate modelele bazate pe funcțiile MPI şi paradigmele bazate pe utilizarea sistemului de calcul simbolic Matematica.

Pornind de la versiunea 7.0 (2008) în pachetul Matmatica, instrumentele de programare paralele au fost integrate direct în pachet, fără a necesita produse software suplimentare (Parallel Computing Toolkit). În versiunea 8.0 (2010) au fost făcute unele modificări (adăugate subnuclee), au fost implementate suporturi pentru Windows HPC Server, Microsoft Compute Cluster Server și Sun Grid. Spre deosebire de alte pachete matematice, Matematica nu se leagă de nicio bibliotecă-sistem de transfer de mesaje. Toate acțiunile sunt efectuate exclusiv cu ajutorul mediului Wolfram Matematica. La un nivel scăzut, interacțiunea este organizată folosind protocolul Wolfram Symbolic Transfer Protocol (WSTP) [1]. Deoarece toate caracteristicile paralele sunt „legate" de limbaj, este imposibil să se stabilească analogii directe cu funcțiile MPI.

Tot aici se face o analiză a posibilităților sistemului de calcul simbolic Matematica pentru elaborarea programelor paralele pe sistemul de calcul paralel de tip DMM (clustere). Problemele din teoria jocurilor bimatriceale pot fi folosite nu doar la modelarea matematică a proceselor decizionale, dar pot servi drept *benchmark* pentru testarea performanțelor diferitor sisteme de calcul. Este elaborat un algoritm paralel pentru determinarea situațiilor Nash de echilibru în strategii pure pentru jocurile bimatriceale. În acest algoritm se pune accent îndeosebi pe paralelizarea la nivel de operații (determinarea elementelor maximale și a indicilor lor) și pe paralelizarea la nivel de date (divizarea matricelor în submatrici și distribuirea lor pe nuclee (procesoare)). Pentru acest algoritm sunt elaborate programe paralele utilizând sistemul Matematica și modele de programare MPI. În programele elaborate se realizează diferite modalități de distribuire a calculelor pe nuclee și diferite modalități de paralelizare la nivel de date. Folosind jocul bimatriceal și algoritmul paralel drept benchmark test, se realizează o analiză comparativă a timpului de calcul pentru programele elaborate.

1. Configurarea și executarea nucleelor paralele în sistemul de calcul simbolic Matematica

Calculul paralel în mediul Mathematica [2-3] se bazează pe lansarea mai multor nuclee (procesoare, procese) și controlul acestora, oferind astfel un mediu pentru programare paralelă pe sisteme de calcul cu memorie distribuită. Oportunitățile pentru calcul paralel sunt aproape complet implementate în Mathematica și, prin urmare, sunt independente de mașină.

Bazele calculelor paralele în Mathematica sunt reprezentate de mediul MathLink. MathLink este o interfață obișnuită și flexibilă pentru interacțiunea programelor terțe părți cu pachetul Mathematica, care de asemenea este utilizat în interiorul pachetului Mathematica în sine. În MathLink, platforma și arhitectura sunt separate, ceea ce vă permite să lucrați atât local, cât și prin rețea. Această interfață poate transmite tot ceea ce Mathematica poate reprezenta și oferă, de asemenea, opțiuni pentru gestionarea pachetului. Metodele pentru conectarea nucleelor în pachetul Mathematica sunt următoarele:

- *kerneluri locale* (LocalKernels). Acestea sunt folosite pentru a organiza paralelismul pe același computer ca și procesul master al pachetului Mathematica. Acest tip de conexiune este potrivit pentru un mediu multi-core și este cel mai simplu mod de a lucra cu calcul paralel;
- *rețea ușoară* (The Lightweight Grid). Metoda este folosită pentru organizarea calculelor paralele pe diferite calculatoare, folosind procesul-master al pachetului Mathematica. Metoda folosește tehnologia Wolfram Lightweight Grid pentru a rula sistemul Mathematica pe mașini de la distanță. Această metodă este potrivită pentru rețele eterogene și atunci când nu sunt disponibile alte tehnologii de control;
- *integrare cluster* (Cluster *I*ntegration). Metoda este folosită pentru organizarea calculelor paralele pe multe computere folosind procesul-master al pachetului Mathematica, care se integrează cu un număr mare de tehnologii de cluster terțe. Cluster Integration oferă o interfață unică, perfectă pe diferite sisteme de gestionare a clusterelor (de exemplu: CCS – Microsoft Windows Compute Cluster Server 2003, HPC – Microsoft Windows High Performance Computing Server 2008, LSF — Platform Load Sharing Facility, PBS – Altair Portable Batch System Pro, SGE – Sun Grid Engine) pentru a identifica și aloca resurse și a programa calculele;
- *nuclee la distanță* (RemoteKernels). Metoda este folosită pentru organizarea calculelor paralele pe mai multe calculatoare, folosind procesul-master al pachetului Mathematica, care utilizează tehnologia de invocare a shell-urilor la distanță (de exemplu, ssh) pentru a începe calculele și este de obicei mai dificil de configurat și întreținut.

Pentru a începe calculul paralel, trebuie să configuram pachetul Mathematica prin deschiderea elementului de meniu Evaluation→Parallel Kernel Configuration, unde se specifică cum se pot executa (porni) nucleele pachetului Mathematica, se selectează acțiunile în cazul unei defecțiuni a nucleului și cum se configurează nucleele în funcție de metoda de conectare. Pentru a monitoriza starea nucleelor în sistemul Mathematica, este disponibil un instrument special – Parallel Kernel Status, care poate fi apelat în meniul Evaluation→Parallel Kernel Status. Pentru a obține informații detaliate despre configurarea sistemului pentru calculul paralel, vom utiliza funcția (comanda) SystemInformation[] cu accesarea filei Paralel.

Când se utilizeaza calculul paralel, nucleul principal al pachetului Mathematica lansează nucleele paralele ale pachetului Mathematica atunci când se invocă o operație paralelă. În acest caz vor fi utilizate acele nuclee care sunt specificate în Parallel Kernel Configuration. Tipul de conexiune implicit este Local Kernels. Pentru a vizualiza nucleele configurate, se poate utiliza comanda $ConfiguredKernels. Rezultatul {} indică că inițial nu avem nuclee acsesibile, în afară de nucleul master, configurate în prealabil. Dacă la executarea comenzii $ConfiguredKernels obținem rezultatul {<<4 kernels on compute-0-1>>,<<4 kernels on compute-0-2>>}, aceasta înseamnă că pentru utilizator au fost configurate *nuclee la distanță* folosind, de exemplu, elementele Evaluation→Parallel Kernel Configuration→Remote Kerneks ale meniului. Pe fiecare de pe nodurile compute-0-1 și compute-0-2 ale clasterului au fost utilizate câte 4 nuclee. În cazul în care la executarea comenzii $ConfiguredKernels obținem rezultatul

```
Out[●]={<<2 kernels on cluster hpc.local>>, LightweightGridClient
   `LightweightGrid[{Agent -> http://hpc.local:3737 /WolframLightweightGrid /Manager,
   KernelCount -> 2, LocalLinkMode Connect, Service, TimeOut -> 30}], <<2 local
   kernels>>, <<4 kernels on compute-0-1>>, <<4 kernels on compute-0-2>>}
```

Aceasta înseamnă că pentru utilizator au fost configurate: 2 nuclee *integrare cluster* (Cluster **In**tegration), 2 nuclee *rețea ușoară* (The Lightweight Grid), 2 nuclee *locale* (LocalKernels) şi 8 nuclee *la distanță* (RemoteKernels) pe nodurile compute-0-1 și compute-0-2 pe clasterul USM. În acest articol vom utiliza numai nucle la distanţă, adică RemoteKernels.

Comanda LaunchKernels[] poate fi utilizată pentru lansarea tuturor nucleelor paralele configurate în prezent. Dacă se execută comanda LaunchKernels cu indicarea numarului de nuclee, de exemplu, LaunchKernels [4], atunci se lansează numarul indicat de nuclee locale.

Vom prezenta un fragment de cod al sistemului Grid Matematica de pe clasterul USM[1] (http://hpc.usm.md/wordpress/) pentru executarea nucleelor prin metoda RemoteKernels în cazul în care aceste nuclee nu au fost configurate:

```
In[2]:=Needs["SubKernels`RemoteKernels`"]
In[3]:=LaunchKernels[RemoteMachine["compute-0-1", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect
   `4` -linkname '`2`' -subkernel -noinit", 4]]
Out[3]=KernelObject[1, compute-0-1], KernelObject[2, compute-0-1 KernelObject[3,
   compute-0-1], KernelObject[4, compute-0-1]
In[4]:=LaunchKernels[RemoteMachine["compute-0-2", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect
   `4` -linkname '`2`' -subkernel -noinit", 4]]
Out[4]=KernelObject[5, compute-0-2], KernelObject[6, compute-0-2], KernelObject[7,
   compute-0-2], KernelObject[8, compute-0-2]
```

Astfel, aceste noduri sunt disponibile pentru executarea în paralel a calculelor pe sisteme de tip DMM.

În continuare, pentru determinarea unor caracteristici ale nucleelor paralele vom utiliza funcția ParallelEvaluate. Această funcție poate fi utilizată în următoarele situaţii: ParallelEvaluate[expr] d – evaluează expresia expr pe toate nucleele paralele disponibile (configurate) și returnează lista rezultatelor obținute; ParallelEvaluate[expr, kernel] – evaluează expr pe nucleul paralel specificat prin parametru kernel; ParallelEvaluate[expr,{ker_1, ker_2, ...}] – evaluează expr pe kernelurile paralele specificate prin parametrii ker_i.

Folosind această funcție și funcţiile $KernelID – atribuie fiecărui nucleu care rulează un număr de identificare unic, $ProcessID – atribuie, de către sistemul de operare sub care este rulat, procesului de pe nucleul dat un identificator unic, $MachineName – atribuie numele calculatorului unde se execută funcția, se poate obține o informaţie suplimentară despe nucleele accesibile. Această informaţie este foarte utilă cand se realizează distribuirea calculelor pe nuclee concrete. În acest sens, vom prezenta, drept continuare a fragmentelor anterioare, următorul fragment de program în care se determină pentru nucleele disponibile identificatoarele proceselor, numele nodurilor și identificatoarele nucleelor:

```
In[9]:=ParallelEvaluate[{$ProcessID,$MachineName,$KernelID}]
Out[9]={{3046, compute-0-1, 1}, {3156, compute-0-1, 2}, {3397, compute-0-1, 3},
      {3731, compute-0-1, 4}, {9257, compute-0-2, 5}, {9367, compute-0-2, 6}, {9604,
      compute-0-2, 7}, {9938, compute-0-2, 8}}
```

Comanda Kernels[] oferă lista de nuclee (identificatoarele și numele nodulrilor) disponibile pentru calcul paralel. Comanda $KernelCount oferă numărul de nuclee (subnuclee) disponibile pentru calcule paralele. Comanda Kernels[][[i]] returnează identificatorul nuceului i. Dacă dorim să returnăm identificatorul procesului, numele nodului și identificatorul nucleului i, utilizăm comanda ParallelEvaluate [{$ProcessID,$MachineName, $KernelID},Kernels[][[i]]].

Utilizand funcția Drop[list,n] – excluderea primelor n elemente din lista list, folosind comanda ParallelEvaluate [{$ProcessID,$MachineName,$KernelID},Drop[Kernels[],2]] vom returna identificatoarele proceselor, numele nodurilor și identificatoarelor nuclelor de la i la $KernelCount, adică a ultimelor i-$KernelCount nuclee active. Această modalitate este utilizată în cazul în care dorim să returnăm informaţia despre identificatoarele numai ale unei clase (grupe) de nuclee active.

[1] Toate programele din acest articol au fost elaborate şi testate pe sistemul Matematica 11.1 cu următoarea licență L3886-4248, numărul de licențe 2, numărul de nuclee 16.

2. Modalități de distribuire a calculelor paralele pe sisteme de calcul paralel cu memorie distribiută folosind sistemul Matematica

2.1. Modalități de distribuire a calculelor pe nuclee

Folosind funcția Kernels[][[i]] putem realiza unele aspecte de *distribuire a calculelor pe nucleele active*. Spre exemplu, dacă dorim ca exp să fie evaluate de nucleul cu identificatorul 36 de pe nodul compute-0-2, atunci putem folosi comanda ParallelEvaluate[exp, Kernels[][[8]]] consultând în prealabil rezultatul comenzii Kernels[]. Menționăm următoarele: executarea paralelă a oricăror operații va începe numai după ce nucleele configurate vor fi **lansate, activate**. Acest lucru este necesar numai dacă ați utilizat în program una dintre funcțiile: CloseKernels[] – încheie (termină) toate nucleele paralele din lista Kernels[], CloseKernels[k] – încheie (termină) nucleul k; CloseKernels[{k_1, k_2, ...}]– încheie nucleele k_1, k_2, ... și doriți să continuați calculele în același program. Lansarea, activarea nucleelor se face utilizând una dintre următoarele variante ale comenzii LaunchKernes: LaunchKernels[] – lansează toate subnucleele paralele configurate în prezent; LaunchKernels[n] – lansează n subnuclee locale pe computerul curent; LaunchKernels[ker] – lansează nucleul specificat de parametrul ker; LaunchKernels[{ker1, ker_2, ...}] – lansează kernelele ker_i.

Vom prezenta prin fragmente de cod ale sistemului de calcul simbolic Matematica diverse modalități de distribuire a calculelor pe nucleele paralele de pe nodurile clasterului. Nuclee disponibile vor fi următoarele:

```
In[]:= $ConfiguredKernels
Out[]={<<2 local kernels>>, <<4 kernels on compute-0-1>>,<<4 kernels on compute-0-2
        >>}
```

Pentru ca codul de program să fie mai scurt, în cazul indicării identificatorului procesului sau a identificatorului nucleului vom folosi o listă, de exemplu kernel, volosind comanda kernel= ParallelEvaluate[$KernelID]. Astfel, dacă dorim să evaluam (executăm) exp pe nucleele cu identificatoarele kernel[[3]] și kernel[[9]], vom utiliza comanda ParallelEvaluate[exp, {kernel[[3]], kernel[[9]]}].

Pentru indicarea identificatorului unui nucleu sau a identificatorului procesului pentru acest nucleu vom folosi lista kernel**.** Dacă dorim ca o expresie matematică să fie evaluată de un nucleu, identificatorul căruia se determină în mod aleatoriu, atunci vom indica identificatorul nucleului astfel: Kernels[][[RandomInteger [{1,$KernelCount}]]]] sau kernel[[RandomInteger[{1, $KernelCount}]]], cum se vede din următorul exemplu: x=10;y=20; ParallelEvaluate[{Print["Nucleul ", $KernelID," calculează x+y=",x+y]}, kernel[[RandomInteger[{ 1, $KernelCount}]]]].

Vom analiza cazul când calculele pot fi distribuite pe acele nuclee, identificatoarele cărora verifică careva condiții. Vom utiliza, de exemplu, funcția Select care este utilizată pentru a selecta elementele din listă care îndeplinesc criteriile specificate: Select[list, crit] – selectează toate elementele e_i ale listei list pentru care funcția criteriu crit [e_i] este setată pe True. De exemplu, dacă dorim ca operațiile să fie executate de nuclee cu identificatoarele impare, atunci vom utiliza următorul fragment de cod: kernel=ParallelEvaluate[$KernelID]; ParallelEvaluate[{ x=45,y=50, Print["Valoarea x=", x," y=" ,y," și x+y=", x+y, " sunt evaluate de nucleele ", $KernelID]} ,Select[kernel,OddQ]].

2.2. Modalități de paralelizare la nivel de date folosind sistemul Matematica

Paralelizarea la nivel de date este un proces de calcul paralel care conține următoarele etape [4]:

- divizarea (împărțirea) datelor intr-un număr finit de părți de volume aproximativ egale;
- distribuirea datelor pe o mulțime de procese care formează un mediu de comunicare;
- culegerea (adunarea) datelor de la procesele unui mediu de comunicare.

Vom analiza câteva aspecte ale paralelizarii la nivel de date în cazul executării paralele în sistemul de calcul simbolic Wolfram Matematica.

Analizam următorul fragment de cod:

```
In[74]:=ParallelEvaluate[z=3,kernel[[3]]]
Out[74]=3
In[75]:=ParallelEvaluate[y=5,kernel[[5]]]
Out[75]=5
```

```
In[76]:=z+y
Out[76]=y+z
In[77]:=ParallelEvaluate[y+z]
Out[77]={y+z,y+z,3+y,y+z,5+z,y+z,y+z,y+z,y+z,y+z}
```

Observăm că numai nucleul 3 cunoaște valoarea variabilei z și nucleul 5 a variabilei y, pe când în următorul fragment toate nucleele cunosc valorile varibilelor z și y:

```
In[89]:=z=5;y=3;
In[90]:=ParallelEvaluate[y+z]
Out[90]={8,8,8,8,8,8,8,8,8,8}
In[91]:=ParallelEvaluate[{z=5,y=3,z+y}]
Out[91]={{5,3,8},{5,3,8},{5,3,8},{5,3,8},{5,3,8},{5,3,8},
        {5,3,8},{5,3,8},{5,3,8},{5,3,8}}
      (* Nucleu Master nu cunoaste valorile  z și y*)
In[92]:=z+y
Out[92]=z+y
```

În cazul în care o careva variabilă este inițializată de nucleul 0, Master, atunci această valoare „este vizibilă" pentru toate nucleele. Acest lucru se aseamană foarte mult cu variabile de tipul LastPrivate din modelul de programare paralelă OpenMP. Acest lucru se confirmă de următorul fragment:

```
In[1]:= kernel=ParallelEvaluate[$KernelID]
Launching kernels...
Out[1]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
In[2]:= A=56
Out[2]= 56
In[3]:=ParallelEvaluate[{x=45,y=50,Print["Val. x=", x,"
      y=" ,y," și x+y+A=", x+y+A, " sunt evaluate de
      nucl. ",$KernelID]}, Select[kernel,OddQ]]
From KernelObject[1, local]:
Val. x=45 y=50 și x+y+A=151 sunt evaluate de nucl. 1
From KernelObject[3, compute-0-1]:
Val. x=45 y=50 și x+y+A=151 sunt evaluate de nucl. 3
From KernelObject[5, compute-0-1]:
Val. x=45 y=50 și x+y+A=151 sunt evaluate de nucl. 5
From KernelObject[7, compute-0-2]:
Val. x=45 y=50 și x+y+A=151 sunt evaluate de nucl. 7
From KernelObject[9, compute-0-2]:
Val. x=45 y=50 și x+y+A=151 sunt evaluate de nucl. 9
Out[3]= {{45, 50, Null}, {45, 50, Null}, {45, 50, Null},
      {45, 50, Null},{45, 50, Null}}
In[4]:= x+y+A
Out[4]= 56 + x + y
```

Memoria virtuală partajată este un model de programare care permite procesoarelor pe o mașină cu memorie distribuită pentru a fi programată ca și cum ar fi memorie partajată. Un software specializat are grijă de cele necesare pentru ca comunicarea să se realizeze într-un mod transparent. Mathematica oferă funcții care implementează memoria virtuală partajată pentru Remote Kernels. Dezavantajul unei variabile partajate este că fiecare acces pentru citire sau scriere necesită **comunicare prin rețea,** deci este mai lentă decât accesul la o variabilă locală nepartajată.

Astfel, pentru ca unele date să fie accesibile (vizibile) de toate nucleele, avem doua opțiuni: a) nucleul Master, adică nucleul cu identificatorul 0, atribuie valorile necesare; b) folosim funcțiile SetSharedVariable[$s_1,s_2,...$] – declară simbolurile și ca variabile partajate ale căror valori sunt sincronizate între toate nucleele paralele și $SharedVariables – returnează lista de variabile care sunt în prezent distribuite între nucleele paralele. În fragmentul de mai jos deja toate procesele (nucleele) cunosc valorile variabilelor pe care le inițializează numai două nuclee:

```
In[19]:=ParallelEvaluate[$SharedVariables]
Out[19]={{},{},{},{},{},{},{},{},{},{}}
In[20]:=SetSharedVariable[z,y]
In[21]:=$SharedVariables
Out[21]={Hold[y],Hold[z]}
In[22]:=ParallelEvaluate[$SharedVariables]
Out[22]={{Hold[y],Hold[z]},{Hold[y],Hold[z]},{Hold[y],
      Hold[z]}, {Hold[y],Hold[z]},{Hold[y],Hold[z]}, {Hold[y],
      Hold[z]},{Hold[y],Hold[z]},{Hold[y], Hold[z]},
      {Hold[y],Hold[z]},{Hold[y],Hold[z]}}
In[24]:=kernel=ParallelEvaluate[$KernelID]
Out[24]={1,2,3,4,5,6,7,8,9,10}
In[25]:=ParallelEvaluate[y=5,kernel[[4]]]
Out[25]=5
In[26]:=ParallelEvaluate[z=3,kernel[[6]]]
Out[26]=3
In[27]:=z+y
Out[27]=8
In[28]:=ParallelEvaluate[z+y,kernel[[8]]]
Out[28]=8
```

Astfel, vom face următoarea remarcă.

Remarcă. *Penrtu sisteme de calcul de tipul DMM, comanda SetSharedVariable[$s_1,s_2,...$] este similară comenzii MPI_Bcast, și anume: nucleul (procesul) Master transmite variabilele $s_1,s_2,...$ proceselor (nucleeler) de tipul RemoteKernels.*

3. Implementarea soft a algoritmului paralel pentru determinarea situațiilor de echilibru în strategii pure

3.1. Utilizarea sistemului de calcul simbolic Matematica pentru soluționarea în paralel a jocurilor bimatriceale

Fie dat un joc bimatriceal $\Gamma = \langle I, J, A, B \rangle$ unde I – mulțimea de indici ai liniilor matricelor, J – mulțimea de indici ai coloanelor matricelor, iar $A = \|a_{ij}\|_{\substack{i\in I \\ j\in J}}$, $B = \|b_{ij}\|_{\substack{i\in I \\ j\in J}}$ reprezintă matricele de câștig ale

jucătorilor. Situaţia de echilibru [5] este perechea de indici (i^*, j^*), pentru care se verifică sistemul de inegalităţi: $(i^*, j^*) \Leftrightarrow \begin{cases} a_{i^* j^*} \geq a_{ij^*} \ \forall i \in I, \\ b_{i^* j^*} \geq b_{i^* j} \ \forall j \in J. \end{cases}$

Utilizând această definiție în mod firesc poate fi elaborat ***Algoritmul de determinare a situaţiei de echilibru în strategii pure pentru jocurile bimatriceale.***

Algoritmul 3.1

a) Pentru orice coloană fixată în matricea A notăm (evidenţiem) toate elementele maximale după linie. Cu alte cuvinte, se determină valorile aplicației multivoce de tipul „best responce" $i^*(j) = \arg\max_{i \in I} a_{ij}$ pentru orice jϵJ.

b) Pentru orice linie fixată în matricea B notăm toate elementele maximale de pe coloane. Cu alte cuvinte, se determină valorile aplicației multivoce de tipul „best responce" $j^*(i) = \arg\max_{j \in J'} b_{ij}$ pentru orice iϵI.

c) Selectăm acele perechi de indici care concomitent sunt selectate atât în matricea A, cât și în matricea B. Pentru aceasta se poate proceda astfel. Se construiește graficul aplicației i^*, adică $gri^* = \{(i,j): i = i^*(j), \forall j \in J\}$ și corespunzător graficul aplicației j^*, adică $grj^* = \{(i,j): j = j^*(i), \forall i \in I\}$. Situaţii de echilibru sunt toate situaţiile cate aparţin intersecției acestor două grafice, adică $NE = gri^* \cap grj^*$.

Vom implementa algoritmul descris mai sus pe sisteme DMM folosind sistemul de calcul simbolic Matematica. La început vom identifica acele comenzi ale sistemului Matematica care vor fi ulterior utilizate pentru implementarea soft a algoritmului descris mai sus.

Pentru determinarea indicelui tuturor liniilor și a indicelui tuturor coloanelor care conțin elementul maximal au fost construite următoarele funcții:

GraphMaxIndRow[list_]:=Join @@ MapIndexed[Thread[{First /@ Position[#, Max@#], First@#2}]&, Transpose[list]],

GraphMaxIndCol[list_]:=Join @@ MapIndexed[Thread[{ First@#2, First /@ Position[#, Max@#]}]&, list].

La elaborarea lor au fost utilizate următoarele comenzi: Join[list1,list2…] – concatenează listele sau alte expresii; f @@ expr sau Apply[f, expr] – se aplică expr în f, de exemplu, f @@ {a,b,c,d} va returna f[a,b,c,d], sau Plus @@ {a,b,c,d} va returna a+b+c+d ; MapIndexed[f,expr] – se aplică f elementelor expr, oferind specificația părții fiecărui element ca un al doilea argument al lui f. De exemplu, MapIndexed[f, {a, b, c, d}] va returna {f[a,{1}],f[b,{2}],f[c,{3}],f[d,{4}]}; Thread[{a, b, c} == {x, y, z}] va returna {a==x,b==y,c==z}; Thread[f[{a, b, c}]] va returna {f[a],f[b],f[c]}; Thread[f[{a, b, c}, x]] va returna {f[a,x],f[b,x],f[c,x]}; Thread[f[{a, b, c}, {x, y, z}]] va returna {f[a,x],f[b,y],f[c,z]}; First[{{a, b}, {c, d}}] va returna {a,b}.

Vom descrie matematic algoritmul paralel pentru determinarea situaţiilor Nash de echilibru în strategii pure pentru jocul bimatriceal $G = \langle I, j, A, B \rangle$, unde $A = \|a_{ij}\|_{i\in I}^{j\in J}$, $B = \|b_{ij}\|_{i\in I}^{j\in J}$. Vom presupune că matricea A este divizată în submatrici de tip coloane și matricea B este divizată în submatrici de tip linii. Adică, vom obține un șir de submatrice $SubA^k = \|a_{ij}\|_{i\in I}^{j\in J_k}$ și $SubB^k = \|b_{ij}\|_{i\in I_k}^{j\in J}$, unde $I_k = \{i_k, i_{k+1}, \dots, i_{k+p}\}$ și $J_k = \{j_k, j_{k+1}, \dots, j_{k+p}\}$. $SubA^k$ este o submatrice care constă din p coloane ale matricei A și $SubB^k$ este o submatrice care constă din p linii ale matricei B. Folosind algoritmul descris mai sus, nucleul cu numarul t va determina pentru orice $j_k \in J_k$ graficul aplicației mutivoce $i^*(j_k) = Argmax_{i\in I} a_{ij_k}$, adica $gr_k i^*$. Similar, nucleul cu numarul t va determina pentru orice $i_k \in I_k$ graficul aplicației mutivoce $j^*(i_k) = Argmax_{j\in J} b_{i_k j}$, adica $gr_k j^*$. În final, același nucleu t va determina $LineGr^t = \cup_k gr_k i^*$ și $ColGr^t = \cup_k gr_k j^*$. Pentru determinarea mulțimii $LineGr^t$, cum s-a menționat mai sus, vom folosi următoarea funcție elaborată în codul Wolfram Matematica:**GraphMaxIndRow[list_]:=Join @@ MapIndexed[Thread[{First /@ Position[#, Max@#], First@#2}]&, Transpose[list]].** Pentru determinarea $ColGr^t$ vom folosi următoarea funcție elaborată în codul Wolfram Matematica:

- cazul când indicele liniei este urmat de indicele coloanei *(i,j)*

GraphMaxIndCol[list_]:=Join @@ MapIndexed[Thread[{ First@#2, First /@ Position[#, Max@#]}]&, list]

- cazul cand indicele coloanei este urmat de indicele liniei *(j,i)*

GraphMaxIndCol[list_]:=Join @@ MapIndexed[Thread[{First /@ Position[#, Max@#], First@#2}]&, list]

Implementăm aceste funcții pentru algoritmul paralel descris mai sus. Una dintre problemele noi care sunt rezultatul paralelizării la nivel de date, este determinarea valorilor indicilor globali construiți în baza indicilor locali. Fie dat următorul șir de indici: BrSubA2={{1, 1}, {2, 1}, {3, 1}, {4, 1}, {1, 2}, {2, 2}, {3, 2}, {4, 2}}, care indică indicii elementelor de pe coloanele 1 și 2 ale unei matrice cu 4 linii. Folosind instrucțiunea Join[Map[ReplacePart[#1, 2->3]&, Drop[BrSubA2,4]], Map[ReplacePart[#1,2->4]&, Drop[BrSubA2,-4]]] vom înlocui coloana **1** prin colana **3** și coloana **2** prin coloana **4**. De fapt, avem următoarea problemă: fie dată o listă de perechi de indici $\{\{j,i\}\}$, să se creeze o altă listă în care dacă i=1, atunci să se inlocuiască cu 3, și dacă i=2, atunci să se înlocuiască cu 4. De exemplu, dacă L={{1, 1}, {1, 2}, {2, 1}, {2, 2}, {3, 1}, {3, 2}, {4, 1}, {4, 2}}, trebuie să se creeze lista L={{1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 3}, {3, 4}, {4, 3}, {4, 4}}. Aceasta poate fi realizată folosind instrucțiunea For[i=1,i<First@Dimensions[L]+1,i++, If[L[[i,2]]==1, L[[i,2]]=3,L[[i,2]]=4]]. Astfel, avem toate funcțiile și comenzile necesare care ne permit să elaboram programe paralele în sistemul de calcul simbolic Mathematica pentru determinarea situaţiilor Nash de echilibru în strategii pure în jocurile bimatriceale de dimensiuni foarte mari.

Distribuirea pe nuclee a coloanelor matricei A se vace cu ajutorul vectorilor ColStart și ColEnd, pe când distribuirea liniilor matricei B pe nuclee se face prin intermediul vectorilor LineStart și LineEnd, unde indicele i indică identificatorul nucleului. Fragmentul de cod care realizează cele menționate este prezentat mai jos.

```
ColStart=Table[1,{i,CountEven}];ColEnd=Table[1, {i,CountEven}];
If[Mod[m,CountEven]!=0,ColEnd[[1]]=IntegerPart[m/CountEven]+1,ColEnd[[1]]=IntegerPart[m/ CountEven]];
For[i=2,i<=CountEven,i++,ColStart[[i]]=ColEnd[[i-1]]+1;
If[i<=Mod[m,CountEven],ColEnd[[i]]=ColStart[[i]]+ IntegerPart[m/CountEven], ColEnd[[i]]= ColStart[[i]]
   +IntegerPart[m/CountEven]-1]];
RowStart=Table[1,{i,CountOdd}];RowEnd=Table[1,{i, CountOdd}];
If[Mod[n,CountOdd]!=0,RowEnd[[1]]=IntegerPart[n/ CountOdd]+1,RowEnd[[1]]=IntegerPart[n/ CountOdd]];
For[i=2,i<=CountOdd,i++,RowStart[[i]]=RowEnd[[i-1]]+1;
If[i<=Mod[n,CountOdd],RowEnd[[i]]=RowStart[[i]]+ IntegerPart[n/CountOdd],RowEnd[[i]]= RowStart[[i]]
   +IntegerPart[n/CountOdd]-1]];
```

Acest mod reprezintă „o inflație informațională": nucleul i are acces la toate elementele vectorilor ColStart și ColEnd, pe când este suficient să posede numai elementele i ale acestor vectori. Dar în Wolfram Matematica nu există funcții de transmitere a datelor de tipul „nucleu-nucleu".

Vom realiza o analiză comparativă a timpului de calcul pentru determinarea situaţiilor de echilibru pentru jocurile bimatriceale în strategii pure în cazul calculelor secvențiale și al celor paralele. Deoarece pentru matrici de dimensiuni foarte mari tiparul mulţimii situaţiilor de echilibru va fi foarte „costisitor în timp", vom genera matrici $A = \left\|a_{i,j}\right\|_{i=\overline{1,n}}^{j=\overline{1,m}}$ și $B = \left\|b_{i,j}\right\|_{i=\overline{1,n}}^{j=\overline{1,m}}$ pentru care *NE=(n,m)*. Este evident că pentru matrici de tipul A=Table[10i+j,{i,n},{j,m}]; și B=Table[10i+j,{i,n},{j,m}]; vom avea că *NE=(n,m)*. Pentru soluționarea jocurilor bimatriceale cu matrici de acest tip au fost realizate programe în codul Wolfram Mathemtica, în care se reflectă diferite modalităţi de paralelizare la nivel de date (divizare și distribuire a matricilor) și modalități de distribuire a calculelor pe nuclee.

Pentru **cazul secvenţial** vom folosi urmatorul cod de program:

Programul 3.1

```
GraphMaxIndRow[list_]:=Join@@MapIndexed[Thread[{First/@Position[#,Max@#],First@#2}]
   &,Transpose[list]];
GraphMaxIndCol[list_]:=Join@@MapIndexed[Thread[{First@#2,First/@Position[#,Max@#]}]
   &,list];
                              (* initializarea n și m *)
Print["n=",n," m=",m," Time=", AbsoluteTiming[
A=Table[10i+j,{i,n},{j,m}];B=Table[10i+j,{i,n},{j,m}];
Br1=GraphMaxIndRow[A];Br2= GraphMaxIndCol[B];
```

```
    If[IntersectingQ[Br1,Br2],Print["NE=",Intersection[Br1,Br2]],Print["Nu exista
    situatii de echilibru"]];]]
```

Pentru a lucra cu matrici destul de mari vom diviza matricile A și B în blocuri de submatrici. De exemplu, pentru n=m=5000 matricile A și B se divizează în submatrici de dimensiunea 625x625. Astfel, în cazul în care **matricile A și B sunt divizate în submatrici şi iniţializate de nucleul master**, atunci vom folosi următorul cod de program **secvenţial**[2]**:**

Programul 3.2

```
CountEven=(* se initializeaza *); CountOdd=(* se initializeaza *);
GraphMaxIndRow[list_]:=Join@@MapIndexed[Thread[{First/@Position[#,Max@#],
   First@#2}]&,Transpose[list]];
GraphMaxIndCol[list_]:=Join@@MapIndexed[Thread[{First@#2,First/@Position[#,
   Max@#]}]&,list];
                                  (* initializarea n și m *)
Print["n=",n," m=",m," Time=", AbsoluteTiming[
ColStart=Table[1, {i,CountEven}]; ColEnd=Table[1,{i,CountEven}];
If[Mod[m,CountEven]!=0,ColEnd[[1]]=IntegerPart[m/CountEven]+1,ColEnd[[1]]=
   IntegerPart[m/CountEven]];
For[i=2,i<=CountEven,i++,ColStart[[i]]=ColEnd[[i-1]]+1; If[i<=Mod[m,CountEven],
   ColEnd[[i]]= ColStart[[i]]+ IntegerPart[m/CountEven],ColEnd[[i]]=ColStart[[i]]+
   IntegerPart[m/CountEven]-1]];
RowStart=Table[1,{i,CountOdd}];RowEnd=Table[1,{i,CountOdd}];
If[Mod[n,CountOdd]!=0,RowEnd[[1]]=IntegerPart[n/CountOdd]+1,RowEnd[[1]]=
   IntegerPart[n/CountOdd]];
For[i=2,i<=CountOdd,i++,RowStart[[i]]=RowEnd[[i-1]]+1;If[i<=Mod[n,CountOdd],
   RowEnd[[i]]= RowStart[[i]]+IntegerPart[n/CountOdd], RowEnd[[i]]=RowStart[[i]] +
   IntegerPart[n/CountOdd]-1]];
For[p=1,p<=CountEven,p++,subA= Table[10i+j,{i,n}, {j,ColStart[[p]], ColEnd[[p]]}];
LL=GraphMaxIndRow[subA]; If[p!=1, For[i=1,i<=ColEnd[[p]]-ColStart[[p]]+1,i++,For[k=1,
   k<=First@ Dimensions[LL], k++, While[LL[[k,2]]==i,LL[[k,2]]= ColStart[[p]]+i-
   1;Continue[]]]];]; GraphMaxIndRow[p]=LL;];
For[q=1,q<=CountOdd,q++,subB=Table[10i+j,{i,RowStart[[q]], RowEnd[[q]]},{j,m}];
CC=GraphMaxIndCol[subB]; If[q!=1,For[i=1,i<=RowEnd[[q]]-RowStart[[q]]+1,i++,
   For[k=1,k<=First@Dimensions[CC], k++, While[CC[[k,1]]==i,CC[[k,1]]=RowStart[[q]]+i-1;
   Continue[]]]];];GraphMaxIndCol[q]=CC;];
Br1=Fold[Join, {},Table[GraphMaxIndRow[p], {p,1,CountEven}]];
Br2=Fold[Join, {},Table[GraphMaxIndCol[q], {q,1,CountOdd}]];
If[IntersectingQ[Br1,Br2],Print["NE=", Intersection[Br1,Br2]],Print["Nu exista situatii
   de echilibru"]];]]
```

În cazul în care **matricile A și B sunt divizate în submatrici, fiecare nucleu iniţializeaza submatricea sa** și distribuirea calculelor pe nuclee pare/impare se realizează folosind ciclul `For[p=1,p<=CountEven,p++…` pentru elementele `EvenKernels[[p]]` și ciclul `For[q=1,q<=CountOdd,q++…` pentru elementele `OddKernels[[q]]` (nu se indică identificatorul nucleului și identificatoarele pot fi orice şir crescător de numere întregi), atunci vom folosi următorul cod paralel de program:

Programul 3.3

```
KernelsID=ParallelEvaluate[$KernelID];EvenKernels=Select[KernelsID,EvenQ];
OddKernels=Select[KernelsID,OddQ]; CountOdd=If[OddQ[$KernelCount],
   IntegerPart[$KernelCount/2]+1,IntegerPart[$KernelCount/2]];
CountEven=$KernelCount-CountOdd;
GraphMaxIndRow[list_]:=Join@@MapIndexed[Thread[{First/@Position[#,Max@#], First@#2}]&,
   Transpose[list]];
```

[2]Penrtu executarea acestui program: se copie textul în buferul `clipboard (Ctrl+c)`, se execută

```
[UserName@hpc]$ math
 In[1]:=(Ctrl+v)
```

```
GraphMaxIndCol[list_]:=Join@@MapIndexed[Thread[{First@#2,First/@Position[#,
   Max@#]}]&,list];
SetSharedFunction[GraphMaxIndRow,GraphMaxIndCol];
                              (* initializarea n și m *)
Print["n=",n," m=",m," Time=",AbsoluteTiming[
   ColStart=Table[1,{i,CountEven}];ColEnd=Table[1,{i,CountEven}];
   If[Mod[m,CountEven]!=0,ColEnd[[1]]=IntegerPart[m/CountEven]+1,
      ColEnd[[1]]=IntegerPart[m/CountEven]];
   For[i=2,i<=CountEven,i++,ColStart[[i]]=ColEnd[[i-1]]+1;If[i<=Mod[m,CountEven],
      ColEnd[[i]]= ColStart[[i]]+IntegerPart[m/CountEven],ColEnd[[i]]=ColStart[[i]]+
      IntegerPart[m/CountEven]-1]];
   RowStart=Table[1,{i,CountOdd}];RowEnd=Table[1,{i,CountOdd}];
   If[Mod[n,CountOdd]!=0,RowEnd[[1]]= IntegerPart[n/CountOdd]+1,RowEnd[[1]]=
      IntegerPart[n/CountOdd]];
   For[i=2,i<=CountOdd,i++,RowStart[[i]]=RowEnd[[i-1]]+1; If[i<=Mod[n,CountOdd],
      RowEnd[[i]] = RowStart[[i]]+IntegerPart[n/CountOdd], RowEnd[[i]]=RowStart[[i]]+
      IntegerPart[n/CountOdd]-1]];
   For[p=1,p<=CountEven,p++,ParallelEvaluate[{subA=Table[10i+j,{i,n},{j,
   ColStart[[p]], ColEnd[[p]]}]; LL=GraphMaxIndRow[subA];
      If[p!=1,For[i=1,i<=ColEnd[[p]]-ColStart[[p]]
      +1,i++,For[k=1,k<=First@Dimensions[LL],k++,While[LL[[k,2]]==i,LL[[k,2]]=ColStart[[
      p]]+i-1;Continue[]]]];]; GraphMaxIndRow[p]=LL; }, EvenKernels[[p]]] ;];
   For[q=1,q<=CountOdd,q++,ParallelEvaluate[{subB=Table[10i+j,{i,RowStart[[q]],RowEnd[[q
      ]]},{j,m}];CC=GraphMaxIndCol[subB];If[q!=1,For[i=1,i<=RowEnd[[q]]-RowStart[[q]]+1,
      i++,For[k=1,k<=First@ Dimensions[CC],k++,While[CC[[k,1]]==i,CC[[k,1]]=
      RowStart[[q]]+i-1; Continue[]]]];];GraphMaxIndCol[q]=CC;},OddKernels[[q]]];];
   Br1=Fold[Join, {},Table[GraphMaxIndRow[p], {p,1,CountEven}]];
   Br2=Fold[Join, {},Table[GraphMaxIndCol[q], {q,1,CountOdd}]];
   If[IntersectingQ[Br1,Br2],Print["NE=", Intersection[Br1,Br2]],Print["Nu exista
      situatii de echilibru"]];]]
```

În cazul în care **matricile A și B sunt divizate în submatrici, fiecare nucleu inițializează submatricea sa** și distribuirea calculelor pe nuclee pare/impare se realizează **fără a utiliza** ciclul `For[p=1,p<=CountEven,p++...` pentru elementele `EvenKernels[[p]]` și ciclul `For[q=1,q<=CountOdd,q++...` pentru elementele `OddKernels[[q]]` (se indică identificatorul nucleului și identificatoarele trebuie să reprezinte un șir crescător de numere întregi de la 1 la `$KernelCount`), atunci vom folosi următorul cod modificat al Programului 3.3:

Programul 3.3a

Codul din programul 3.3

```
   ColStartAll=Sort[Join[ColStart,ColStart]];
   ColEndAll=Sort[Join[ColEnd,ColEnd]];
   ParallelEvaluate[{subA=Table[10i+j,{i,n},{j,ColStartAll[[$KernelID]],
      ColEndAll[[$KernelID]]}]; LL=GraphMaxIndRow[subA];
      If[$KernelID!=1,For[i=1,i<=ColEndAll[[$KernelID]]-ColStartAll[[$KernelID]]+1,i++,
      For[k=1,k<=First@Dimensions[LL],
      k++,While[LL[[k,2]]==i,LL[[k,2]]=ColStartAll[[$KernelID]]+i-1; Continue[]]]];];
      GraphMaxIndRow[$KernelID]=LL; }, Select[KernelsID,OddQ]];
   RowStartAll=Sort[Join[RowStart,RowStart]];
   RowEndAll=Sort[Join[RowEnd,RowEnd]];
   ParallelEvaluate[{subB=Table[10i+j,{i,RowStartAll[[$KernelID]],RowEndAll[[$KernelID]]
      },{j,m}];CC=GraphMaxIndCol[subB];
      If[$KernelID!=2,For[i=1,i<=RowEndAll[[$KernelID]]-RowStartAll[[$KernelID]]+1,i++,
      For[k=1,k<=First@ Dimensions[CC],k++,While[CC[[k,1]]==i,CC[[k,1]]=
      RowStartAll[[$KernelID]]+i-1;Continue[]]]];]; GraphMaxIndCol[$KernelID]=CC;},
      Select[KernelsID,EvenQ]];
```

Codul din Programul 3.3

În cazul în care **matricile A și B sunt divizate în submatrici, fiecare nucleu inițializează submatricea sa** și distribuirea calculelor nu se face pe nuclee pare/impare, adică folosim distribuirea pe care o face nucleul `Master`, utilizatorul nu o poate gestiona (identificatoarele trebuie să reprezinte un șir crescător de numere întregi de la `1` la `$KernelCount),` atunci vom folosi următorul cod modificat al Programului 3.3:

Programul 3.3b

Codul din Programul 3.3

```
Print["n=",n," m=",m," Time=",AbsoluteTiming[
ColStart=Table[1,{i,$KernelCount}];ColEnd=Table[1,{i,$KernelCount}];
If[Mod[m,$KernelCount]!=0,ColEnd[[1]]=IntegerPart[m/$KernelCount]+1,
ColEnd[[1]]=IntegerPart[m/$KernelCount]];
For[i=2,i<=$KernelCount,i++,ColStart[[i]]=ColEnd[[i-1]]+1;
If[i<=Mod[m,$KernelCount],ColEnd[[i]]=ColStart[[i]]+
IntegerPart[m/$KernelCount],ColEnd[[i]]=ColStart[[i]]+  IntegerPart[m/$KernelCount]-1]];
RowStart=Table[1,{i,$KernelCount}];RowEnd=Table[1,{i,$KernelCount}];
If[Mod[n,$KernelCount]!=0,RowEnd[[1]]= IntegerPart[n/$KernelCount]+1,
RowEnd[[1]]=IntegerPart[n/$KernelCount]];For[i=2,i<=$KernelCount,i++,
RowStart[[i]]=RowEnd[[i-1]]+1;
If[i<=Mod[n,$KernelCount],RowEnd[[i]]= RowStart[[i]]+ IntegerPart[n/$KernelCount],
RowEnd[[i]]=RowStart[[i]]+IntegerPart[n/$KernelCount]-1]];
ParallelEvaluate[{subA=Table[10i+j,{i,n},{j,ColStart[[$KernelID]],
   ColEnd[[$KernelID]]}];LL=GraphMaxIndRow[subA];
   If[$KernelID!=1,For[i=1,i<=ColEnd[[$KernelID]]-ColStart[[$KernelID]]+1,
   i++,For[k=1,k<=First@Dimensions[LL],k++,
   While[LL[[k,2]]==i,LL[[k,2]]=ColStart[[$KernelID]]+i-1;Continue[]]]];];
   GraphMaxIndRow[$KernelID]=LL; }];
ParallelEvaluate[{subB=Table[10i+j,{i,RowStart[[$KernelID]], RowEnd[[$KernelID]]},
   {j,m}];CC=GraphMaxIndCol[subB]; If[$KernelID!=1,For[i=1,i<=RowEnd[[$KernelID]]-
   RowStart[[$KernelID]]+1,i++,For[k=1,k<=First@ Dimensions[CC],k++,
   While[CC[[k,1]]==i,CC[[k,1]]=RowStart[[$KernelID]]+i-1;
   Continue[]]]];];GraphMaxIndCol[$KernelID]=CC;}];
Br1=Fold[Join,{},Table[GraphMaxIndRow[p], {p,1,$KernelCount}]];
Br2=Fold[Join,{},Table[GraphMaxIndCol[q], {q,1,$KernelCount}]];
If[IntersectingQ[Br1,Br2],Print["NE=", Intersection[Br1,Br2]],Print["Nu exista situatii
   de echilibru"]];]]
```

În cazul în care **matricile A și B sunt initializate de nucleul Master și fiecare nucleu „citeste" submatricea sa,** atunci vom folosi următorul cod paralel de program:

Programul 3.4

```
KernelsID=ParallelEvaluate[$KernelID];
EvenKernels=Select[KernelsID,EvenQ];
OddKernels=Select[KernelsID,OddQ];
CountOdd=If[OddQ[$KernelCount],OdsN=IntegerPart[$KernelCount/2]+1,OdsN=
   IntegerPart[$KernelCount/2]];
CountEven=$KernelCount-CountOdd;
GraphMaxIndRow[list_]:=Join@@MapIndexed[Thread[{First/@Position[#,Max@#], First@#2}]&,
   Transpose[list]];
GraphMaxIndCol[list_]:=Join@@MapIndexed[Thread[{First@#2,First/@Position[#,
   Max@#]}]&,list];
SetSharedFunction[GraphMaxIndRow,GraphMaxIndCol];
                              (* initializarea n și m *)
Print["n=",n," m=",m," Time=", AbsoluteTiming[
   A=ParallelTable[10i+j,{i,n},{j,m}];  B=ParallelTable[10i+j,{i,n},{j,m}];
ColStart=Table[1,{i,CountEven}];ColEnd=Table[1,{i,CountEven}];
If[Mod[m,CountEven]!=0,ColEnd[[1]]=IntegerPart[m/CountEven]+1,ColEnd[[1]]=
   IntegerPart[m/CountEven]];
For[i=2,i<=CountEven,i++,
```

```
ColStart[[i]]=ColEnd[[i-1]]+1;
If[i<=Mod[m,CountEven],ColEnd[[i]]=ColStart[[i]]+IntegerPart[m/CountEven],
ColEnd[[i]]=ColStart[[i]]+IntegerPart[m/CountEven]-1]];
RowStart=Table[1,{i,CountOdd}];RowEnd=Table[1,{i,CountOdd}];
If[Mod[n,CountOdd]!=0,RowEnd[[1]]=IntegerPart[n/CountOdd]+1,RowEnd[[1]]=
   IntegerPart[n/CountOdd]]; For[i=2,i<=CountOdd,i++,RowStart[[i]]=RowEnd[[i-1]]+1;
If[i<=Mod[n,CountOdd],RowEnd[[i]]=RowStart[[i]]+IntegerPart[n/CountOdd],
RowEnd[[i]]=RowStart[[i]]+IntegerPart[n/CountOdd]-1]];
For[p=1,p<=CountEven,p++,ParallelEvaluate[{subA=A[[1;;n,ColStart[[p]];;ColEnd[[p]]]];
LL=GraphMaxIndRow[subA];
If[p!=1,For[i=1,i<=ColEnd[[p]]-ColStart[[p]]+1,i++, For[k=1,k<=First@Dimensions[LL],
   k++, While[LL[[k,2]]==i,LL[[k,2]]=ColStart[[p]]+i-1;Continue[]]]];];
   GraphMaxIndRow[p]=LL;},EvenKernels[[p]]];];
For[q=1,q<=CountOdd,q++,ParallelEvaluate[{subB=B[[RowStart[[q]];;RowEnd[[q]],1;;m]];
CC=GraphMaxIndCol[subB];If[q!=1,For[i=1,i<=RowEnd[[q]]-RowStart[[q]]+1,i++,
   For[k=1,k<=First@Dimensions[CC],k++,While[CC[[k,1]]==i,CC[[k,1]]=RowStart[[q]]+i-
   1;Continue[]]]];];GraphMaxIndCol[q]=CC;},OddKernels[[q]]];];
Br1=Fold[Join, {},Table[GraphMaxIndRow[p], {p,1,CountEven}]];
Br2=Fold[Join, {},Table[GraphMaxIndCol[q], {q,1,CountOdd}]];
If[IntersectingQ[Br1,Br2],Print["NE=", Intersection[Br1,Br2]],Print["Nu exista situatii
   de echilibru"]];]]
```

În cazul în care **matricile A și B sunt initializate de nucleul Master, fiecare nucleu „citeste" submatricea sa** și distribuirea calculelor pe nuclee pare/impare se realizează **fără a utiliza** ciclul `For[p=1,p<=CountEven,p++...` pentru elementele `EvenKernels[[p]]` și ciclul `For[q=1,q<=CountOdd,q++...` pentru elementele `OddKernels[[q]]` (se indică identificatorul nucleului și identificatoarele trebuie să reprezinte un șir crescător de numere întregi de la 1 la `$KernelCount`), atunci vom folosi următorul cod modificat al Programului 3.4:

Programul 3.4a

```
KernelsID=ParallelEvaluate[$KernelID]
EvenKernels=Select[KernelsID,EvenQ];
OddKernels=Select[KernelsID,OddQ];
CountOdd=If[OddQ[$KernelCount], IntegerPart[$KernelCount/2]+1,
   IntegerPart[$KernelCount/2]];
CountEven=$KernelCount-CountOdd;
GraphMaxIndRow[list_]:=Join@@MapIndexed[Thread[{First/@Position[#,Max@#],First@#2}]&,Tr
   anspose[list]];
GraphMaxIndCol[list_]:=Join@@MapIndexed[Thread[{First@#2,First/@Position[#,Max@#]}]&,li
   st];
SetSharedFunction[GraphMaxIndRow,GraphMaxIndCol];
                                (* initializar n si m *)
Print["n=",n," m=",m," Time=", AbsoluteTiming[
A=ParallelTable[10i+j,{i,n},{j,m}];
B=ParallelTable[10i+j,{i,n},{j,m}];
ColStart=Table[1,{i,CountEven}];ColEnd=Table[1,{i,CountEven}];
If[Mod[m,CountEven]!=0,ColEnd[[1]]=IntegerPart[m/CountEven]+1,
   ColEnd[[1]]=IntegerPart[m/CountEven]];
For[i=2,i<=CountEven,i++,ColStart[[i]]=ColEnd[[i-1]]+1;
If[i<=Mod[m,CountEven],ColEnd[[i]]=ColStart[[i]]+IntegerPart[m/CountEven],
ColEnd[[i]]=ColStart[[i]]+IntegerPart[m/CountEven]-1]];
RowStart=Table[1,{i,CountOdd}];RowEnd=Table[1,{i,CountOdd}];
If[Mod[n,CountOdd]!=0,RowEnd[[1]]=IntegerPart[n/CountOdd]+1,
   RowEnd[[1]]=IntegerPart[n/CountOdd]];
For[i=2,i<=CountOdd,i++,RowStart[[i]]=RowEnd[[i-1]]+1;
   If[i<=Mod[n,CountOdd],RowEnd[[i]]= RowStart[[i]]+IntegerPart[n/CountOdd],
   RowEnd[[i]]=RowStart[[i]]+IntegerPart[n/CountOdd]-1]];
 ColStartAll=Sort[Join[ColStart,ColStart]];
```

```
   ColStartAll=Sort[Join[ColStart,ColStart]];
   ColEndAll=Sort[Join[ColEnd,ColEnd]];
   ParallelEvaluate[{subA=A[[1;;n,ColStartAll[[$KernelID]];;ColEndAll[[$KernelID]]]];
   LL=GraphMaxIndRow[subA];If[$KernelID!=1,For[i=1,i<=ColEndAll[[$KernelID]]-
   ColStartAll[[$KernelID]]+1,i++, For[k=1,k<=First@Dimensions[LL],k++,
   While[LL[[k,2]]==i,LL[[k,2]]=ColStartAll[[$KernelID]]+i-1;Continue[]]]];];
   GraphMaxIndRow[$KernelID]=LL;},Select[KernelsID,OddQ]];
   RowStartAll=Sort[Join[RowStart,RowStart]]; RowEndAll=Sort[Join[RowEnd,RowEnd]];
   ParallelEvaluate[{subB=B[[RowStartAll[[$KernelID]];;RowEndAll[[$KernelID]],1;;m]];
   CC=GraphMaxIndCol[subB];If[$KernelID!=2,For[i=1,i<=RowEndAll[[$KernelID]]-
   RowStartAll[[$KernelID]]+1,i++,For[k=1,k<=First@ Dimensions[CC],k++,
       While[CC[[k,1]]==i,CC[[k,1]]=RowStartAll[[$KernelID]]+i-1;Continue[]]]];];
       GraphMaxIndCol[$KernelID]=CC;},Select[KernelsID,EvenQ]];
Br1=Fold[Join, {},Table[GraphMaxIndRow[OddKernels[[p]]], {p,1,CountEven}]];
Br2=Fold[Join, {},Table[GraphMaxIndCol[EvenKernels[[q]]], {q,1,CountOdd}]];
If[IntersectingQ[Br1,Br2],Print["NE=", Intersection[Br1,Br2]],Print["Nu exista situatii
   de echilibru"]];]]
```

În cazul în care **matricile A și B sunt initializate de nucleul Master, fiecare nucleu „citește" submatricea sa** și distribuirea calculelor nu se face pe nuclee pare/impare, adică folosim distribuirea pe care o face nucleul `Master`, utilizatorul nu o poate gestiona (identificatoarele trebuie să reprezinte un șir crescător de numere întregi de la 1 la `$KernelCount`, atunci vom folosi următorul cod modificat al Programului 3.4:

Programul 3.4b

```
KernelsID=ParallelEvaluate[$KernelID]
EvenKernels=Select[KernelsID,EvenQ];
OddKernels=Select[KernelsID,OddQ];
CountOdd=If[OddQ[$KernelCount],OdsN=IntegerPart[$KernelCount/2]+1,OdsN=IntegerPart[$Ker
   nelCount/2]];
CountEven=$KernelCount-CountOdd;
GraphMaxIndRow[list_]:=Join@@MapIndexed[Thread[{First/@Position[#,Max@#],First@#2}]&,Tr
   anspose[list]];
GraphMaxIndCol[list_]:=Join@@MapIndexed[Thread[{First@#2,First/@Position[#,Max@#]}]&,li
   st];
SetSharedFunction[GraphMaxIndRow,GraphMaxIndCol];
                                (* initializare n si m *)
Print["n=",n," m=",m," Time=", AbsoluteTiming[
A=ParallelTable[10i+j,{i,n},{j,m}];B=ParallelTable[10i+j,{i,n},{j,m}];
ColStart=Table[1,{i,CountEven}];ColEnd=Table[1,{i,CountEven}];
If[Mod[m,CountEven]!=0,ColEnd[[1]]= IntegerPart[m/CountEven]+1,ColEnd[[1]]=
   IntegerPart[m/CountEven]];
For[i=2,i<=CountEven,i++,ColStart[[i]]=ColEnd[[i-1]]+1;
   If[i<=Mod[m,CountEven],ColEnd[[i]]=ColStart[[i]]+IntegerPart[m/CountEven],
   ColEnd[[i]]=ColStart[[i]]+IntegerPart[m/CountEven]-1]];
   RowStart=Table[1,{i,CountOdd}];RowEnd=Table[1,{i,CountOdd}];
   If[Mod[n,CountOdd]!=0,RowEnd[[1]]=
   IntegerPart[n/CountOdd]+1,RowEnd[[1]]=IntegerPart[n/CountOdd]];
   For[i=2,i<=CountOdd,i++,RowStart[[i]]=RowEnd[[i-1]]+1;
   If[i<=Mod[n,CountOdd],RowEnd[[i]]= RowStart[[i]]+IntegerPart[n/CountOdd],
   RowEnd[[i]]=RowStart[[i]]+IntegerPart[n/CountOdd]-1]];
ColStartAll=Sort[Join[ColStart,ColStart]];
   ParallelEvaluate[{subA=A[[1;;n,ColStart[[$KernelID]];;ColEnd[[$KernelID]]]];
   LL=GraphMaxIndRow[subA]; If[$KernelCount!=1,For[i=1,i<=ColEnd[[$KernelCount]]-
   ColStart[[$KernelCount]]+1,i++,For[k=1,k<=First@Dimensions[LL],k++,
   While[LL[[k,2]]==i,LL[[k,2]]=ColStart[[$KernelCount]]+i-1; Continue[]]]];];
   GraphMaxIndRow[$KernelCount]=LL;}];
ParallelEvaluate[{subB=B[[RowStart[[$KernelCount]];;RowEnd[[$KernelCount]],1;;m]];
   CC=GraphMaxIndCol[subB];
```

```
   If[$KernelCount!=1,For[i=1,i<=RowEnd[[$KernelCount]]-RowStart[[$KernelCount]]+1,i++,
   For[k=1,k<=First@Dimensions[CC],k++,While[CC[[k,1]]==i,CC[[k,1]]=
   RowStart[[$KernelCount]]+i-1; Continue[]]]];];
   GraphMaxIndCol[$KernelCount]=CC;}];
Br1=Fold[Join, {},Table[GraphMaxIndRow[p], {p,1,$KernelCount}]];
Br2=Fold[Join, {},Table[GraphMaxIndCol[q], {q,1,$KernelCount}]];
If[IntersectingQ[Br1,Br2],Print["NE=", Intersection[Br1,Br2]],Print["Nu exista situatii
   de echilibru"]];]]
```

Programele 3.1, 3.2, 3.3, 3.3a, 3.3b, 3.4, 3.4a și 3.4b au fost testate pe diferite exemple de jocuri bimatriceale și rezultatele testărilor coincid cu rezultatele teoretice.

3.2. Utilizarea modelului MPI de programare paralelă pentru soluționarea jocurilor bimatriceale

Folosind modelul de programare paralelă MPI [6] pe sistemul paralel de calcul cu memoria distribuită, vom elabora programe paralele pentru determinarea situaţiilor Nash de echilibru în jocurile bimatriceale. Vom prezenta următoarele explicații despre algoritmul și funcţiile utilizate în programele elaborate. În acest program matricea A este divizată în submatrici linii și matricea B la fel în submatrici linii. Deci, modul de divizare se deosebeşte de cele utilizate mai sus. Acest mod de difizare duce la mai multe calcule. Mai jos vom explica funcțiile și procedurile utilizate.

- Funcția prepare_sendcounts_and_displs determină dimensiunile submatricilor distribuite pe procese (numărul de rânduri), astfel încât să se verifice principiul „load balansing" în cazul paralelizării la nivel de date.
- Funcția get_max_locs determiă valoarea elementului maximal și poziţiile sale, adică indicele liniilor sau al coloanelor.
- În cazul când doar procesul cu rankul 0 iniţializează cu valori matricile A și B, utilizăm funcția MPI_Scatterv, cu parametrii determinaţi de funcția prepare_sendcounts_and_displs, pentru a distribui matricile proceselor din mediul de comunicare.
- Fiecare proces determină vectorul col_max care conține elementele maximale de pe coloanele submatricii matricii A distribuită în baza funcției MPI_Scatterv. După ce se utilizează funcția MPI_Allreduce, care în vectorul global_col_max stochează elementul maximal de pe coloanele matricii „globale" A.
- Fiecare proces în matricea col_max_locs salvează 0 dacă elementul (i,j) din submatricea sa nu este egal cu maximul „global" de pe coloana j a matricii globale și 1 dacă este egal. Acesta va fi utilizat pentru determinarea soluţiilor Nash ca intersecția graficelor aplicaţiilor multivoce de tipul „best response".
- Fiecare proces pentru determinarea elementului maximal de pe liniile și poziţia lor, adică indicele liniilor, în submatricile sale va utiliza funcția get_max_locs, care va determina matricea row_max_locs elementul (i,j) al careia este egal cu 0 dacă elementul (i,j) al submatricii sale nu este maximal pe linia i a submatricii, și egal cu 1 dacă elementul (i,j) al submatricii sale este maximal pe linia i a submatricii. După ce se va folosi funcția MPI_Gatherv, care în vectorul global_row_max va păstra deja elementul maximal de pe liniile matricii „globale" B. Acest vector poartă un caracter „decorativ", el nu se utilizează mai departe. Deci, și funcția MPI_Gatherv poate fi exclusă, ceea ce va micșora timpul de transmitere a datelor.
- Situaţiile Nash de echilibru se calculează astfel: fiecare proces verifică dacă elementele cu valoarea 1 din vectorii row_max_locs și col_max_locs se află pe aceeași poziție și în baza indicilor locali se recalculează indicii globali.

În prgramul ce urmează **procesul cu rancul 0 iniţializează matricile A și B și distribuie submatricile corespunzatoare fiecarui proces** (procesor, nucleu) folosind funcția MPI_Scaterv. Acest program realizează calculele și modul de paralelizare a datelor similar Pogramului 3.4b.

Programul 3.5MPI

```
#include <stdlib.h>
#include "mpi.h"
using namespace std;
void prepare_sendcounts_and_displs(int rows, int cols, int size,  int  *sendcounts_elem,
     int *displs_elem,int *sendcounts_row, int *displs_row)
    { int rows_per_process_1 = rows / size + 1;
    int rows_per_process_2 = rows_per_process_1 - 1;
    int elem_per_process_1 = rows_per_process_1 * cols;
```

```
    int elem_per_process_2 = rows_per_process_2 * cols;
    int rank_max_for_group_1 = rows - rows_per_process_2 * size;
    for (int i = 0; i < rank_max_for_group_1; ++i){        sendcounts_row[i] =
        rows_per_process_1;            sendcounts_elem[i] = elem_per_process_1;  }
    for (int i = rank_max_for_group_1; i < size; ++i){           sendcounts_row[i] =
        rows_per_process_2; sendcounts_elem[i] = elem_per_process_2;     }
    displs_elem[0] = 0; displs_row[0] = 0;
    for (int i = 1; i < size; ++i) {          displs_row[i] = displs_row[i - 1] +
        sendcounts_row[i - 1];         displs_elem[i] = displs_elem[i - 1] +
        sendcounts_elem[i - 1];} }
void get_max_locs(double *arr, int n,double *value, int *indices)
    { double max = arr[0]; for (int i = 1; i < n; ++i)
    if (arr[i] > max) max = arr[i];    *value = max; for (int i = 0; i < n; ++i)
        indices[i] = arr[i] == max ? 1 : 0; }
int main(int argc, char *argv[])
{
int size, rank, t, namelen; double startwtime, endwtime; MPI_Status status;
  MPI_Init(&argc, &argv); MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
int rows, cols,Number_Nash=0,Total_Number_Nash=0;
if(rank == 0)
  {startwtime = MPI_Wtime();    cout<<"Introduceti numarul de randuri: "; cin>>rows;
  cout<<"Introduceti numarul de coloane: "; cin>>cols;
  if (size > rows) {if (rank == 0) printf("ERROR: number of processes > number of
  matrix rows!"); MPI_Finalize(); return 1;   }}
MPI_Bcast(&rows,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&cols,1,MPI_INT,0,MPI_COMM_WORLD);
double *A = NULL, *B = NULL;
if (rank == 0)
  {A = new double[rows * cols]; B = new double[rows * cols]; srand(time(0)); for (int i
    = 0; i <rows; ++i) { for (int j = 0; j < cols; ++j)    A[i * cols + j]=10*i+j;}
    for (int i=0; i<rows;++i) {for (int j=0; j<cols;++j) B[i*cols+j]=10*i+j;}}
int *sendcounts_elem = new int[size]; int *displs_elem = new int[size];
int *sendcounts_row = new int[size]; int *displs_row = new int[size];
prepare_sendcounts_and_displs(rows, cols, size,sendcounts_elem, displs_elem,
  sendcounts_row, displs_row);
int elem_to_send = sendcounts_elem[rank];
int rows_to_send = sendcounts_row[rank];
double *subarr = new double[elem_to_send];
MPI_Scatterv(A,sendcounts_elem,    displs_elem,    MPI_DOUBLE,subarr,    elem_to_send,
  MPI_DOUBLE, 0, MPI_COMM_WORLD);
double *col_max = new double[cols];
for (int i = 0; i < cols; ++i) col_max[i] = subarr[i];
for (int i = 1; i < rows_to_send; ++i)
for (int j=0;j<cols;++j if(subarr[i*cols+j]>col_max[j]) col_max[j]=subarr[i*cols+j];
double *global_col_max = new double[cols];
int *col_max_locs = new int[rows_to_send * cols];
MPI_Allreduce(col_max, global_col_max, cols, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
for (int i=0; i<rows_to_send;++i for (int j=0; j<cols;++j) col_max_locs[i*cols+j]
  =(int)(subarr[i * cols + j] == global_col_max[j]);
double *row_max = new double[rows_to_send]; int *row_max_locs = new int[elem_to_send];
double *global_row_max = new double[rows];
MPI_Scatterv(B, sendcounts_elem, displs_elem, MPI_DOUBLE,subarr, elem_to_send,
  MPI_DOUBLE, 0, MPI_COMM_WORLD);
for (int i=0;i<rows_to_send;++i) get_max_locs(subarr+i*cols,cols,row_max+i,
  row_max_locs + i * cols);
```

```
MPI_Gatherv(row_max, sendcounts_row[rank], MPI_DOUBLE,global_row_max, sendcounts_row,
   displs_row, MPI_DOUBLE,0, MPI_COMM_WORLD);
for (int i = 0; i < rows_to_send; ++i) { for (int j = 0; j < cols; ++j) if
   (row_max_locs[i * cols + j] && col_max_locs[i * cols + j]) { int global_i =
   displs_row[rank] + i; Number_Nash=Number_Nash+1; printf("Nash equilibrium: (%d,
   %d)\n", global_i, j);} }
printf("Numbers of Nash equilibrium profiles= %d determined by the process %d
   \n",Number_Nash, rank);
MPI_Reduce(&Number_Nash,&Total_Number_Nash,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
if(rank==0) printf("Total numbers of Nash equilibrium profiles= %d \n",
Total_Number_Nash);
delete[] subarr; delete[] row_max; delete[] global_row_max; delete[] row_max_locs;
delete[] col_max; delete[] global_col_max;    delete[] col_max_locs; delete[]
displs_row; delete[] sendcounts_row; delete[] displs_elem; delete[] sendcounts_elem;
if (rank==0){ endwtime = MPI_Wtime(); printf("wall clock time = %f seconds \n",
   endwtime-startwtime);}
MPI_Finalize(); return 0;}
```

În baza Programului 3.5MPI vom elabora un program în care deja **fiecare proces** (procesor, nucleu) **inițializează submatricea sa proprie**, deci nu folosim funcția MPI_Scaterv. Acest program realizează calculele și modul de paralelizare a datelor similar Pogramului 3.3b.

Programul 3.6MPI

/* Urmează codul de program din Programul 3.5MPI */

```
double *A = NULL, *B = NULL;
int *sendcounts_elem = new int[size]; int *displs_elem = new int[size];
int *sendcounts_row = new int[size]; int *displs_row = new int[size];
prepare_sendcounts_and_displs(rows, cols, size,sendcounts_elem, displs_elem,
      sendcounts_row, displs_row);
int elem_to_send = sendcounts_elem[rank];
int rows_to_send = sendcounts_row[rank];
double *subarr = new double[elem_to_send];
for (int i = 0; i < sendcounts_row[rank]; ++i) {
for (int j = 0; j < cols; ++j)
subarr[i * cols + j]=10*(i+displs_row[rank])+j; }
double *col_max = new double[cols];
for (int i = 0; i < cols; ++i) col_max[i] = subarr[i];
for (int i = 1; i < rows_to_send; ++i) for (int j = 0; j < cols; ++j)
if (subarr[i * cols + j] > col_max[j]) col_max[j] = subarr[i * cols + j];
double *global_col_max = new double[cols];
int *col_max_locs = new int[rows_to_send * cols];
MPI_Allreduce(col_max, global_col_max, cols, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
for  (int  i=0;i<rows_to_send;++i)  for(int  j=0;j<cols;++j)  col_max_locs[i*cols+j]
=(int)(subarr[i * cols + j] == global_col_max[j]);
double *row_max = new double[rows_to_send];
int *row_max_locs = new int[elem_to_send];
double *global_row_max = new double[rows];
for (int i=0;i<rows_to_send;++i) {for (int j=0;j<cols;++j) subarr[i*cols+j]=10*
   (i+displs_row[rank])+j; };
for (int i=0;i<rows_to_send; ++i) get_max_locs(subarr+i*cols,cols,row_max+i,
row_max_locs + i * cols);
MPI_Gatherv(row_max, sendcounts_row[rank], MPI_DOUBLE, global_row_max, sendcounts_row,
    displs_row, MPI_DOUBLE,      0, MPI_COMM_WORLD);
for (int i = 0; i < rows_to_send; ++i) { for (int j = 0; j < cols; ++j)
if  (row_max_locs[i*cols+j]&&col_max_locs[i*cols+j])  {int  global_i=displs_row[rank]+i;
    Number_Nash=Number_Nash+1; printf("Nash equilibrium: (%d, %d)\n", global_i, j); }}
MPI_Reduce(&Number_Nash,&Total_Number_Nash,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
```

```
if(rank==0) printf("Total numbers of Nash equilibrium profiles= %d
    \n",Total_Number_Nash);
```

/* Urmeza codul de program din Programul 3.5MPI */

Remarcă. *Nu au fost utilizate operaţiile de reducere, adică funcția MPI_Reduce cu operaţia MPI_MAXLOC, pentru a exclude necesitatea de a transmite date între procese.*

Programele 3.5MPI și 3.6MPI au fost testate pe diferite exemple de jocuri bimatriceale și rezultatele testărilor coincid cu rezultatele teoretice.

3.3. Analiza comparativă a timpului de calcul pentru rezolvarea în paralel a jocurilor bimatriceale

În cele ce urmează vom realiza o analiză comparativă a timpului de calcul al programelor 3.1, 3.3, 3.3a, 3.3b, 3.4, 3.4a, 3.4b, 3.5MPI și 3.6MPI. Pentru aceasta programele 3.1, 3.3, 3.3a, 3.3b, 3.4, 3.4a, 3.4b au fost executate pe clasterul paralel al Universitatii de Stat din Moldova folosind Wolfram Mathematica Kernel 11.1. Dependența timpului de executare de dimensiunea matricilor (pentru n=m=1000, 2000,…,10000) și de numarul de nuclee (8, 12 și 16) este prezentată în urmatoarele grafice.

În graficele din Figura 1 sunt prezentate rezultatele testării programelor elaborate în sistemul Matematica executate pe 8 nuclee configurate în baza urmatorului fragment de program:

```
Needs["SubKernels`RemoteKernels`"]
LaunchKernels[RemoteMachine["compute-0-1", "ssh -x -f -l `3` `1`
    /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
    -linkname ``2`' -subkernel -noinit", 4]]
LaunchKernels[RemoteMachine["compute-0-2", "ssh -x -f -l `3` `1`
    /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
    -linkname ``2`' -subkernel -noinit", 4]]
```

Graficele au fost construite utilizand funcția `ListLinePlot` și următoarele liste de date experimentale:

```
TimeProgram31={5.960,6.407,14.697,26.076,40.696,58.366,79.619,128.528,130.861,160.297, 195.799}
TimeProgram33Kernels8={20.637,141.232,453.8,1051.05,2025.56,3475.15,5484.36}
TimeProgram33aKernels8={7.119,47.845,152.506,352.495,681.094,1164.21,1831.16,2724.42, 3847.95}
TimeProgram33bKernels8={1.972,9.134,26.604,56.797,104.37,171.902,265.238,388.637,544.52, 734.387}
TimeProgram34Kernels8={20.5678,141.653,454.005,1057.43,2072.08}
TimeProgram34aKernels8={8.580,53.563,165.209,379.064,931.462}
TimeProgram34bKernels8={11.934,37.656,77.505,138.16,166.933,2072.08}
```

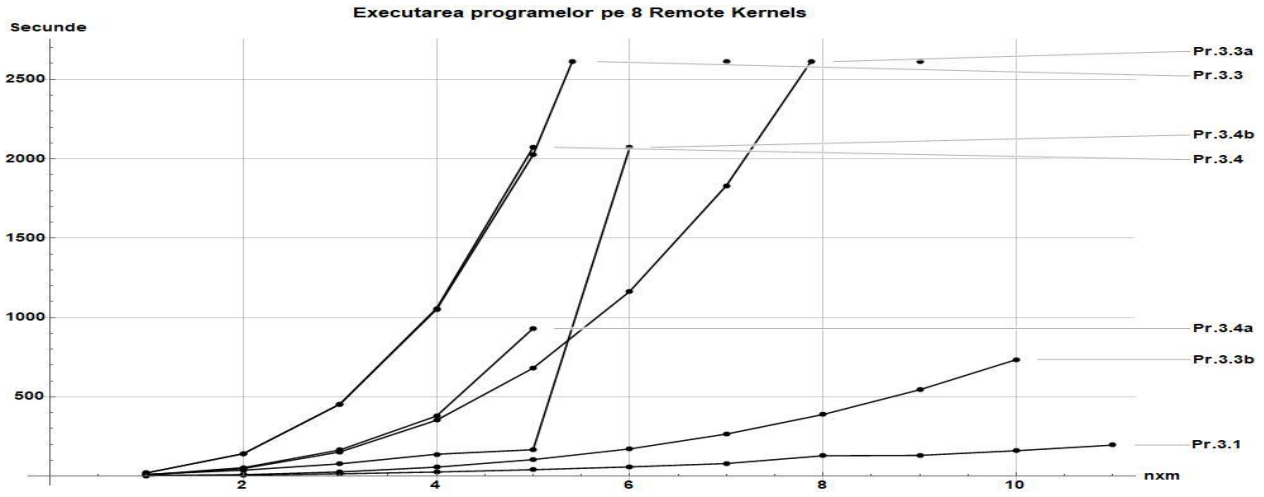

Fig.1

În graficele din Figura 2 sunt prezentate rezultatele testării programelor elaborate în sistemul Matematica executate pe 12 nuclee configurate în baza următorului fragment de program:

```
Needs["SubKernels`RemoteKernels`"]
LaunchKernels[RemoteMachine["compute-0-1", "ssh -x -f -l `3` `1`
    /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
    -linkname ``2`' -subkernel -noinit", 4]]
```

```
LaunchKernels[RemoteMachine["compute-0-2", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
   -linkname ``2`' -subkernel -noinit", 4]]
LaunchKernels[RemoteMachine["compute-0-3", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
   -linkname ``2`' -subkernel -noinit", 4]]
```

Graficele au fost construite utilizand funcția `ListLinePlot` și următoarele liste de date experimentale:

```
TimeProgram31={5.960,6.407,14.697,26.076,40.696,58.366,79.619,128.528,130.861,160.297, 195.799}
TimeProgram33Kernels12={12.876,78.16,242.742,550.009,1048.71,1782.46,2798.29}
TimeProgram33aKernels12={2.932,17.167,51.233,115.886,218.871,370.058,576.612,849.117, 1194.49,
            1627.99}
TimeProgram33bKernels12={1.374,5.219,12.95,25.171,44.121,70.615,105.487,149.306,205.103, 271.831}
TimeProgram34kernels12={13.049,80.7464,248.428,563.051,1009.78}
TimeProgram34aKernels12={5.6,25.823,69.4372,150.639,646.035,665.539}
TimeProgram34bKernels12={11.732,29.572,54.947,99.029,398.723}
```

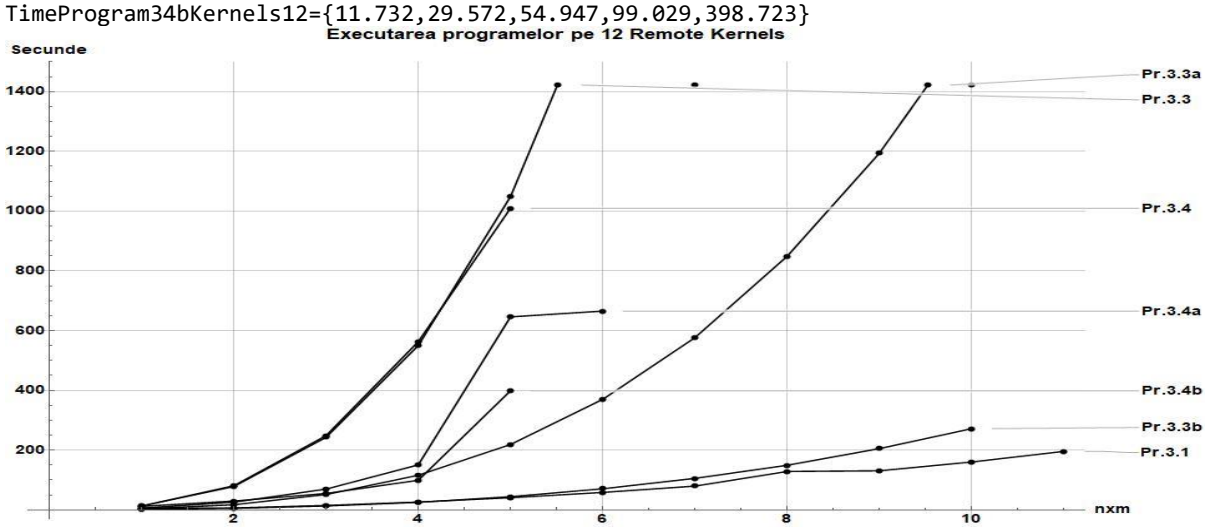

Fig.2

În graficele din Figura 3 sunt prezentate rezultatele testării programelor elaborate în sistemul Matematica executate pe 16 nuclee configurate în baza următorului fragment de program:

```
Needs["SubKernels`RemoteKernels`"]
LaunchKernels[RemoteMachine["compute-0-1", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
   -linkname ``2`' -subkernel -noinit", 4]]
LaunchKernels[RemoteMachine["compute-0-2", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
   -linkname ``2`' -subkernel -noinit", 4]]
LaunchKernels[RemoteMachine["compute-0-3", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
   -linkname ``2`' -subkernel -noinit", 4]]
LaunchKernels[RemoteMachine["compute-0-4", "ssh -x -f -l `3` `1`
   /share/apps/Wolfram/Mathematica/11.1/Executables/wolfram -wstp -linkmode Connect `4`
   -linkname ``2`' -subkernel -noinit", 4]]
```

Graficele au fost construite utilizand funcția `ListLinePlot` și urmatoarele liste de date experimentale:

```
TimeProgram31={5.960,6.407,14.697,26.076,40.696,58.366,79.619,128.528,130.861,160.297, 195.799}
TimeProgram33Kernels16={9.838,53.509,158.503,347.441,656.05,1104.23,1721.81,2531.98,3573.8,
            4847.68}
TimeProgram33aKernels12={2.932,17.167,51.235,115.886,218.871,370.058,576.612,849.117,
            1194.49,1627.99}
TimeProgram33bKernels16={1.12,3.912,9.26,17.454,28.574,44.503,64.25,87.286,117.971,155.42,
            195.296}
TimeProgram34Kernels16={10.56,57.93,168.629,363.764,753.647}
TimeProgram34aKernels16={5.247,20.98,50.938,103.002,216.102}
TimeProgram34bKernels16={10.658,27.924,51.672,93.735,267.369}
```

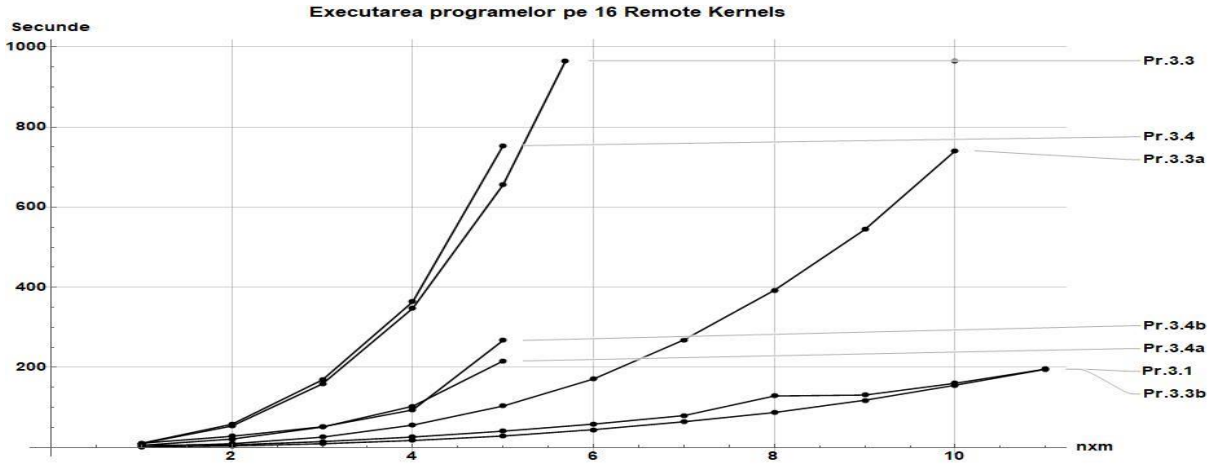

Fig.3

Graficele din Figura 4 au fost construite utilizand funcția `ListLinePlot` și urmatoarele liste de date experimentale:

```
TimeProgram31={5.960,6.407,14.697,26.076,40.696,58.366,79.619,128.528,130.861,160.297, 195.799}
TimeProgram35MPIKernels8={8.509,7.879,7.548,15.030,12.225,11.279,13.347,15.173,24.635, 21.768}
TimeProgram35MPIKernels12={6.622,6.561,8.631,10.905,11.025,12.161,14.985,17.322,17.029, 25.354}
TimeProgram35MPIKernels16={9.236,10.651,7.275,8.535,13.168,13.686,23.581,18.239,18.429, 26.282}
TimeProgram36MPIKernels8={6.290,6.236,8.272,7.084,8.348,8.078,9.049,8.526,10.743,8.386}
TimeProgram36MPIKernels12={6.195,5.514,6.682,6.675,8.338,6.836,9.034,7.968,6.585,7.645}
TimeProgram36MPIKernels16={7.091,7.184,5.923,6.531,8.030,6.297,6.737,8.211,10.105,11.871}
```

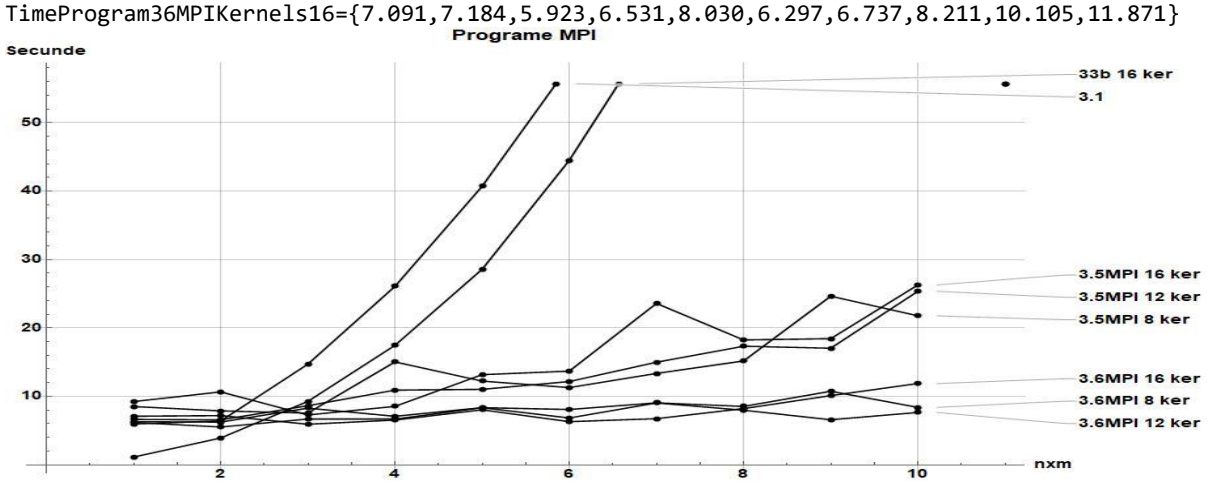

Fig.4

Pentru Programul 3.6MPI s-au mai efectuat calcule suplimentare prezentate în urmatorul tabel:

Dimensiune matrice	16 procesoare	28 procesoare	32 procesoare	48 procesoare
n=30000 m=30000	11.665889	11.187384	14.855256	14.361466
n=35000 m=35000	18.748501	12.359752	13.841478	10.827413
n=40000 m=40000	17.021988	12.191731	13.195590	11.202775
n=45000 m=45000	33.997708	14.808254	13.028965	12.560618
n=48000 m=48000	69.756550	20.810968	18.946320	15.717546
n=49000 m=49000	110.552440	32.880968	16.161673	15.140105
n=49500 m=49500	186.125029	40.321999	22.643520	17.640198
n=49900 m=49900	eroare	55.035883	17.819622	19.017421
n=50000 m=50000	eroare	46.226555	16.388920	11.839626
n=55000 m=55000	eroare	eroare	57.292023	38.964008
n=60000 m=60000	eroare	eroare	eroare	156.085372
n=65000 m=65000	eroare	eroare	eroare	eroare

Concluzii

Menționam că programele 3.3-3.4 dau un timp mai mare de calcul, deoarece „distribuirea pe nuclee" se face secvențial, adică dacă executăm programul pe 8 nuclee pe nodurile de pe compute-0-1 și compute-0-2, atunci mai întâi se execută programul pe nodurile de pe compute-0-1 și abia după pe compute-0-2.

Graficele prezentate permit realizarea unei analize comparative a timpului de executare a programelor în dependenţă de: a) modaliățile de distribuire a calculelor pe nuclee; b) modalitățile de paralelizare la nivel de date. Din analiza graficelor din figurile 1-3 putem face următoarele concluzii la implementarea algoritmilor paraleli în sistemul Wolfram Mathematica:

- evitarea necesităților schimbului de date între nuclee folosind date partajate și încărcarea datelor necesare nucleului o singură dată;
- evitarea repetării calculelor identice pe nucleele individuale, distribuirea omogenă a calculelor pe nuclee.

Analizând graficele din Figural 4 și calculele suplimentare prezentate în tabel, putem concluziona că pentru implementarea soft a algoritmilor paraleli la soluționarea jocurilor bimatriceale în strategii pure pe siteme paralele de calcul de tip HPC cluster, utilizarea modelului de programare bazat pe funcțiile MPI este mult mai eficientă decât utilizarea sistemului Matematica.

Referinţe:

1. Parallel Computing. URL: https://reference.wolfram.com/ language/guide/ ParallelComputing.html
2. STEPHEN, W. *Mathematica book.* Co-published by Wolfram Media and Cambridge University Press. 2003. 1300 p.
3. MANGANO, S. *Mathematica Cookbook*. Published by O'Reilly Media, Inc. 2010. 801 p.
4. JAJA, J. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, Inc., 1992.
5. HANCU**,** B., NOVAC, L. *Bazele teoriei jocurilor*: Note de curs. Chisinau: CEP USM, 2018. 172 p.
6. HÎNCU, B., CALMÎŞ, E. *Modele de programare paralelă pe clustere*. Partea I. *Programare MPI.* Note de curs. Chişinău: CEP USM, 2016. 129 p.

Date despre autori:
Boris HÂNCU, doctor în ştiinţe fizico-matematice, conferenţiar universitar, Universitatea de Stat din Moldova.
E-mail: boris.hancu@gmail.com
Ionel ANTOHI, doctorand, Şcoala doctorală *Matematică şi Ştiinţa Informaţiei*, Universitatea de Stat din Moldova.

Prezentat la 10.07.2020