PARALLEL ALGORITHM TO FIND THE STACKELBERG EQUILIBRIUM PROFILES IN THE THREE STAGE DYNAMIC GAMES WITH DISCRETE PAYOFF FUNCTIONS

Boris HÂNCU

Catedra Informatică şi Optimizare Discretă

În articol este definit un joc dinamic cu trei jucători pe trei niveluri în informaţie completă. Ordinea de alegere a strategiilor este următoarea: jucătorul 1 alege primul strategia sa şi o transmite jucătorului 2 care, la rândul său, va alege strategia sa, după ce îşi alege strategia jucătorul 3. Aceste jocuri pot fi utilizate la modelarea proceselor decizionale în sisteme cu structuri ierarhice. În calitate de soluţie a acestui joc sunt considerate situaţiile Stackelberg de echilibru determinate în baza algoritmului inducţiei recursive. Este descris un algoritm paralel al metodei inducţiei recursive pentru determinarea situaţiilor Stackelberg de echilibru, când funcţiile-scop ale jucătorilor sunt reprezentate în forma unor matrici. Algoritmul paralel este elaborat pentru sisteme paralele de calcul cu memorie distribuită de tip clastere, în care schimbul de date între procesele (procesoarele) de calcul se realizează prin transmiterea de mesaje. Pentru implementarea soft a paralelizării la nivel de date şi la nivel de operaţii a algoritmului paralel se utilizează sistemul de funcţii standardizate Message Passing Interface (MPI). Este formulată şi demonstrată teorema despre corectituninea algoritmului. La fel sunt prezentaţi şi estimatorii unor caracteristici numerice care descriu timpul de calcul al algoritmului paralel.

1. Preliminarie

Let us consider the dynamic game of three players with three stages in the following strategic form

$$\Gamma = \langle Z, Y, X; G : Z \times Y \times X \to \Re, F : Z \times Y \times X \to \Re, H : Z \times Y \times X \to \Re \rangle,$$

where *Z, Y* and *X* are the sets of available strategies and G, F, H are the payoff functions for the player 1, player 2 and player 3 respectively.

The game occurs as follows: in the first stage, the player 1 chooses independently his strategy $z \in Z$, and communicates this strategy to the player 2. In the second stage, the player 2 observes a chosen strategy of player 1 and chooses independently his strategy $y \in Y$, after that communicates this pair of strategies (z, y) to the player 3. In the third stage, the player 3 observes the pair of strategies (z, y) and chooses his strategy $x \in X$. After this, the game is considered finished. Also, suppose that the players want to maximize their payoff functions.

The following backward induction algorithm can find the solution of the dynamic game of three players with three stages:

A) the player 3 observes the chosen strategies $z \in Z$ by player 1 and the strategy $y \in Y$ by player 2 and he will choose his strategy from the optimal reaction set of player 3 to a strategy $z \in Z$ and $y \in Y$. So he will choose the following value of the best response point to set mapping $BR_3 : Z \times Y \to 2^X$

$$x^*(z, y) = BR_3(z, y) = Arg \max_{x \in X} H(z, y, x);$$

B) the player 2 knowing that the player 3 will play the strategy $x^*(z, y)$ chooses the strategy from the optimal reaction set of player 2 for a strategy $z \in Z$ of player 1, so he will choose the following value of the best response point to set mapping $BR_2 : Z \to 2^Y$

$$y^*(z) = BR_2(z) = Arg \max_{y \in Y} F\left(z, y, x^*(z, y)\right);$$

C) the player 1 knowing that the player 2 will choose the strategy $y^*(z)$ and player 3 will choose the strategy $x^*(z,y)$ chooses the strategy from the following set of best response reaction

$$z^* \in BR_1 = Arg \max_{z \in Z} F\left(z, y^*(z), x^*(z, y^*(z)\right).$$

Here $2^X, 2^Y$ denote the set of all subsets of the set X and Y respectively.

Let us introduce the following definition.

Definition. *The strategy profiles*

$$\left(z^*, y^*, x^*\right) = \left(z^*, y^*(z^*), x^*(z^*, y^*(z^*))\right)$$

defined by the steps A)-C) of the backward induction algorithm are called the Stackelberg equilibrium profile in the dynamic game of three players with three stages.

This definition is a generalization of the one of equilibrium profile, which is used at the first time by Scalckelberg in [1].

2. Stackelberg equilibrium profile in the three level dynamic games with discrete payoff functions

Consider the following set of the 3-dimensional matrices $\left\{A^k = \left\|\left(a_{ij}^k, b_{ij}^k, c_{ij}^k\right)\right\|\right\}$ and define the dynamic game of three players with three stages, where *k=1,.,l,...,K; i=1,...n,...,I; j=1,...,m,...,J* are the sets of available strategies and $\left\{a_{ij}^k\right\}, \left\{b_{ij}^k\right\}, \left\{c_{ij}^k\right\}$ are the payoffs for the player 1, player 2 and player 3 respectively. The game occurs as follows: in the first stage, the player 1 chooses independently his strategy *k,* and communicates this strategy to the player 2. In the second stage the player 2 observes a chosen strategy of player 1 independently chooses his strategy *i*, after that communicates this pair of strategies *(k,i)* to the player 3. In the third stage the player 3 observes the pair of strategies *(k,i)* and chooses his strategy *j*. After this, the game is considered finished. Also, suppose that the players want to maximize their payoffs. The solution of this dynamic game can be found by the following backward induction algorithm:

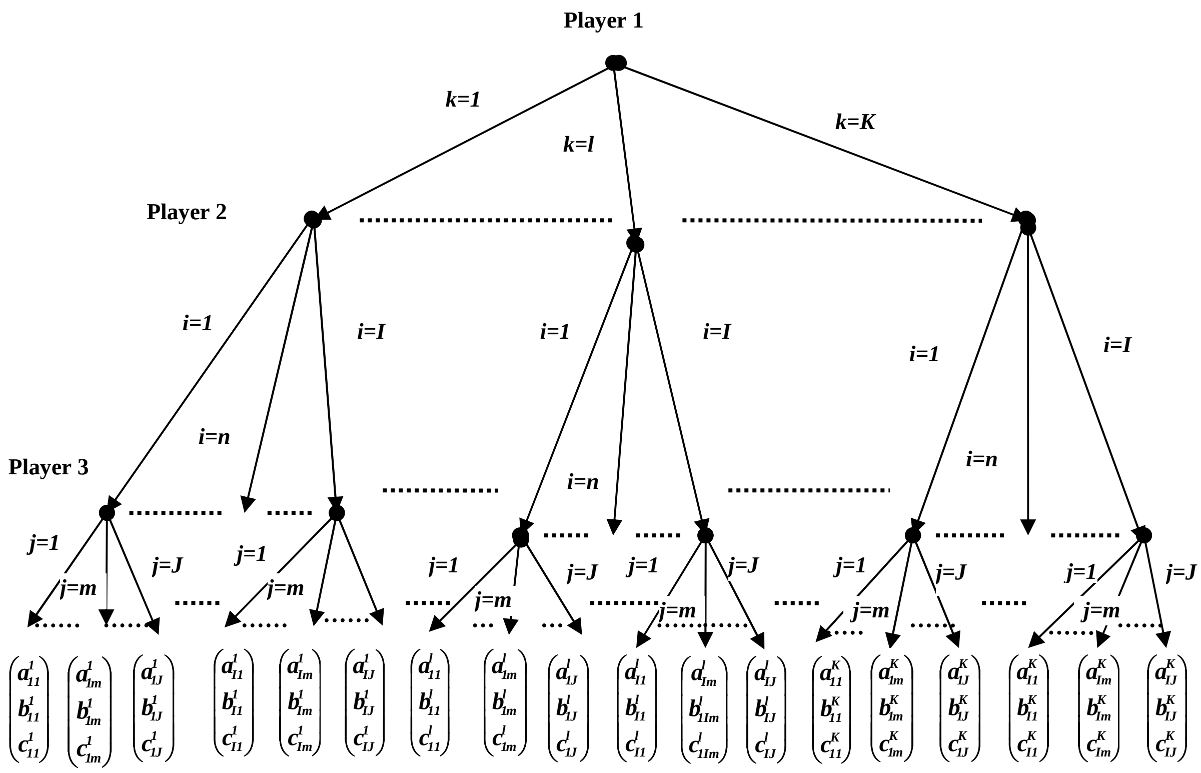
a) for all *k=1,...,l,...,K* and *i=1,...,n,...,I* the player 3 will choose the strategy from the following best response set $J(k,i) = \left\{ j^*(k,i) \middle| j^*(k,i) = \arg\max_j \left\{c_{ij}^k\right\} \right\}$;

b) for all *k=1,...,l,...,K* `the player 2 knowing that the player 3 will play the strategy $j^*(k,i) \in J(k,i)$ chooses the strategy from the optimal reaction set of player 2 for a strategy of player 1 $I(k) = \left\{ i^*(k) \middle| i^*(k) = \arg\max_i \left\{ b_{ij^*(k,i)}^k \right\} \right\}$;

c) the player 1 knowing that the player 2 will choose the strategy $i^*(k) \in I(k)$ and player 3 will choose the strategy $j^*(k,i) \in J^*(k,i)$ chooses the strategy from the set of best response reaction $K^* = \left\{ k^* \middle| k^* = \arg\max_k \left\{ a_{i^*(k) j^*(k, i^*(k))}^k \right\} \right\}$.

Therefore, the strategy profiles $\left(k^*, i^*(k^*), j^*(k^*, i^*(k^*))\right)$ defined by the steps a)-c) of the backward induction algorithm *are called the Stackelberg equilibrium profile in the dynamic game of three players with three stages and discrete payoff functions* [2].

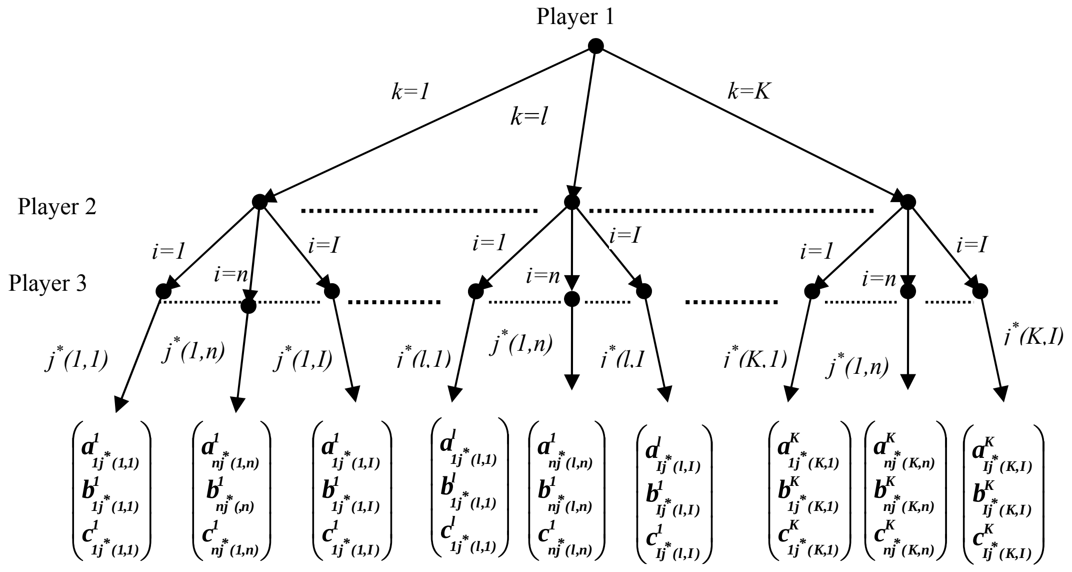
In the extensive form, the described above dynamic game is presented in the Picture 1.



Picture 1

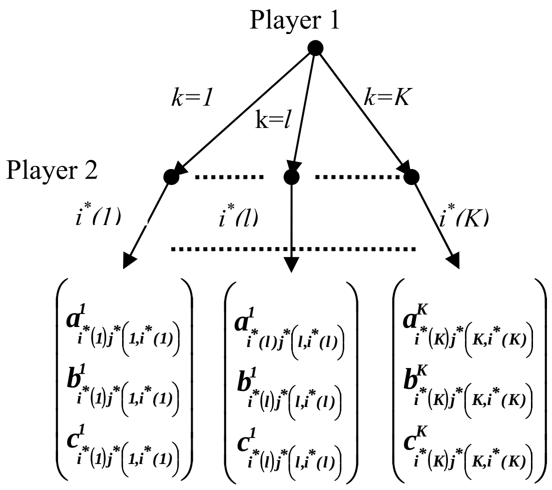
In the graphically form the steps a)-c) of the backward induction algorithm should be represented as follows.

a') The step a) of the backward induction algorithm, i.e. for all $k=1,...,l,...,K$ and $i=1,...,n,...,I$ the subgame obtained after the player 3 chooses the strategy $j^*(k,i)=\arg\max_j\{c_{ij}^k\}$ is represented in Picture 2;



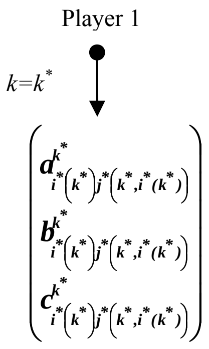
Picture 2

b') The step b) of the backward induction algorithm, i.e. for all $k=1,...,l,...,K$ the subgame obtained after the player 2 chooses the strategy $i^*(k)=\arg\max_i\{b_{ij^*(k,i)}^k\}$ is represented in Picture 3;

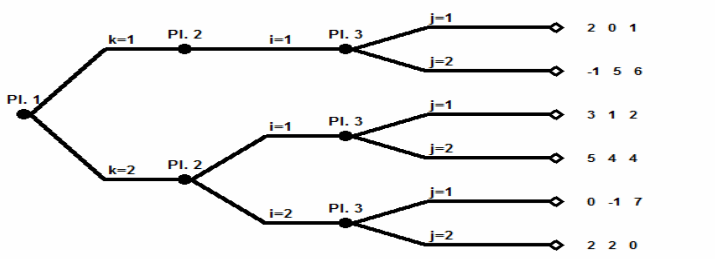


Picture 3

c') The step c) of the backward induction algorithm i.e. the "subgame" obtained after the player 1 chooses the strategy $k^* = \arg\max\limits_k \left\{a^k_{i^*(k)j^*(k,i^*(k))}\right\}$ is represented in Picture 4.



Picture 4



Picture 5

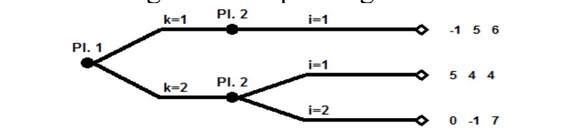
Let us consider the following example.

Example. Let $k=1,2,\ i=1,2,\ j=1,2$ and the matrices are the following $A^1 = \big((2,0,1)\ \ (-1,5,6)\big)$, $A^2 = \begin{pmatrix} (3,1,2) & (5,4,4) \\ (0,-1,7) & (2,2,0) \end{pmatrix}$. So according to the step a) of the backward induction algorithm we obtain that $j^*(1,1)=2,\ \ j^*(2,1)=2,\ \ j^*(2,2)=1.$ According to the step b) we obtain that $i^*(1)=1,\ \ i^*(2)=1.$ According the step c) we obtain that $k^*=2.$ In this way $i^*(k^*)=1,\ \ j^*(k^*,i^*(k^*))=2,$ and the Stackelberg equilibrium profile of the game is (2,1,2).

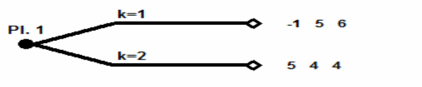
Using the *Gambit*[1] (Software Tools for Game Theory) Version 0.2007.01.30 we can represent in the extensive form and the solution of the game from Example 1 in the following pictures[2]. The initial game is presented in the Picture 5:

[1] This program is free software, distributed under the term of the GNU General Public License.
[2] The labels of picture are the following.
- Nodes labelling: *display the node's label above each node; display the payoff of reaching the node below each node*;
- Action labelling: display the name of the action above each action (branches); display the probability the action is player below each action (branches).

Using the step a) of the backward induction algorithm we will obtain $j^*(1,1)=2,\quad j^*(2,1)=2,\ j^*(2,2)=1$ and so excluding the corresponding subtrees the following subgame is obtained (see the Picture 6):



Picture 6



Picture 7

Using the step b) of the backward induction algorithm we will obtain $i^*(1)=1,\ i^*(2)=1$ and so excluding the corresponding subtrees the following subgame is obtained (see the Picture 7).

Finally, using the step c) of the backward induction algorithm, we will obtain $k^* = 2$ and then the Stackelberg equilibrium profile of the game is (2,1,2).

3. Parallel algorithm to find the Stackelberg equilibrium profile in the three level dynamic games with discrete payoff functions

Parallel computers have two basic architectures: distributed memory and shared memory. Distributed memory parallel computers are essentially a collection of serial computers (nodes) working together to solve a problem. Each node has rapid access to its own local memory and access to the memory of other nodes via some sort of communications network, usually a proprietary high-speed communications network. Data are exchanged between nodes as messages over the network. In a shared memory computer, multiple processor units share access to a global memory space via a high-speed memory bus. This global memory space allows the processors to efficiently exchange or share access to data.

The first step in designing of a parallel algorithm is to decompose the problem into smaller problems. Then, the smaller problems are assigned to processors to work on simultaneously. Roughly speaking, there are two kinds of decompositions:

1) domain decomposition;
2) functional decomposition.

In domain decomposition or "data parallelism", data are divided into pieces of approximately the same size and then mapped to different processors. Each processor then works only on the portion of the data that is assigned to it. Of course, the processes may need to communicate periodically in order to exchange data. Data parallelism provides the advantage of maintaining a single flow of control. A data parallel algorithm consists of a sequence of elementary instructions applied to the data: an instruction is initiated only if the previous instruction is ended. Single-Program-Multiple-Data (SPMD) follows this model where the code is identical on all processors. Such strategies are commonly employed in finite differencing algorithms where processors can operate independently on large portions of data, communicating only the much smaller shared border data at each iteration.

Frequently, the domain decomposition strategy turns out not to be the most efficient algorithm for a parallel program. This is the case when the pieces of data assigned to the different processes require greatly different lengths of time to process. The performance of the code is then limited by the speed of the slowest process. The remaining idle processes do no useful work. In this case, functional decomposition or "task parallelism" makes more sense than domain decomposition. In task parallelism, the problem is decomposed into a large number of smaller tasks and then, the tasks are assigned to the processors as they become available. Processors that finish quickly are simply assigned more work. Task parallelism is implemented in a client-server paradigm. The tasks are allocated to a group of slave processes by a master process that may also perform some of the tasks. The client-server paradigm can be implemented at virtually any level in a program. Historically, there have been two approaches to writing parallel programs. They are

- use of a directives-based data-parallel language, *and*
- explicit message passing via library calls from standard programming languages.

In a directives-based data-parallel language, such as High Performance Fortran (HPF) or OpenMP, a serial code is made parallel by adding directives (which appear as comments in the serial code) that tell the compiler how to distribute data and work across the processors. The details of how data distribution, computation, and communications are to be done are left to the compiler. Data parallel languages are usually implemented on shared memory architectures because the global memory space greatly simplifies the writing of compilers. In the message passing approach, it is left up to the programmer to explicitly divide data and work across the processors as well as manage the communications among them. This approach is very flexible.

Message Passing Interface (MPI) is a widely used standard library for writing massage passing programs, especially on parallel machines with distributed memory. For the distributed memory parallel computers architectures we designed a parallel algorithm to determinate the Stackelberg equilibrium profile in the above described game. We are using the following MPI functions to software implementation of the backward induction algorithm described in the steps a)-c). The MPI_comm_group routine determines the group handle of a communicator. The **MPI_Group_incl** routine creates a new group from an existing group and specifies member processes. The **MPI_Comm_create** routine creates a new communicator to include specific processes from an existing communicator. The MPI function **MPI_Type_struct** is the most general way to construct an MPI derived type because it allows the length, location, and type of each component to be specified independently. Less general procedures are available to describe common patterns of access, primarily within arrays. The **MPI_Bcast** routine enables you to copy data from the memory of the root processor to the same memory locations for other processors in the communicator. The **MPI_Scatter** routine is a one-to-all communication. Different data are sent from the root process to each process (in rank order). When **MPI_Scatter** is called, the root process breaks up a set of contiguous memory locations into equal chunks and sends one chunk to each processor. The **MPI_Reduce** routine enables collect data from each processor, reduce these data to a single value (such as a *sum* or *max*) and store the reduced result on the root processor. The **MPI_Allreduce** is used to combine the elements of each s input buffer of process and stores the combined value on the received buffer of all group members. In addition, the function **MPI_Reduce** combines the elements provided in the send buffer, applies the specified operation (*sum, min, max,* etc), and returns the result to the received buffer of the root process.

The basic steps of the parallel algorithm to find the Stackelberg equilibrium profile in the three level dynamic games are described as following.

1) Using the MPI functions **MPI_comm_group**, **MPI_Group_incl** and **MPI_Comm_create** we create the new communicator, for example *MPI_Comm_Slave* and new group of processes *MPI_Goup_Slave*. The total number of processors of these communicator is equal to *K*I*. Only the process of this communicator will participate on the soft implementation of the backward induction algorithm.

For "data parallelism" of the algorithm we use the message passing approach, implemented by the MPI functions and consist of the following basic steps.

2) One of the process of the communicator *MPI_Comm_Slave* (for example process with the rank equal to *root*) initialized the matrices

$$A=\begin{pmatrix} \left(a^{1}_{11},b^{1}_{11},c^{1}_{11}\right) & \cdots & \left(a^{1}_{1m},b^{1}_{m1},c^{1}_{1m}\right) & \cdots & \left(a^{1}_{1J},b^{1}_{1J},c^{1}_{1J}\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \left(a^{1}_{n1},b^{1}_{n1},c^{1}_{n1}\right) & \cdots & \left(a^{1}_{nm},b^{1}_{nm},c^{1}_{nm}\right) & \cdots & \left(a^{1}_{nJ},b^{1}_{nJ},c^{1}_{nJ}\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \left(a^{K}_{I1},b^{K}_{I1},c^{K}_{I1}\right) & \cdots & \left(a^{K}_{Im},b^{K}_{Im},c^{K}_{Im}\right) & \cdots & \left(a^{K}_{IJ},b^{K}_{IJ},c^{K}_{IJ}\right) \end{pmatrix}$$

and using MPI functions distributed the rows of these matrices (one row per one processes) to all processes of the communicator *MPI_Comm_Slave*. These could be done as follows:

2.1. Use the functions **MPI_Type_struct**, **MPI_Type_commit** and **MPI_Address** to construct the matrice, every element of which is a tuple of the form $\left(a^{k}_{ij},b^{k}_{ij},c^{k}_{ij}\right)$ where *k=1,...,K*, *i=1,...,I* and *j=1,...,J*. This new MPI derived type data reduces the latency[1] of inter-process communication.

2.2. Used the **MPI_Bcast** function for distributed to all process the value *K, I* and *J*.

2.3. Using the **MPI_Scatter** functions we distribute the rows of the matrix constructing in step 2.1 to process of the group *MPI_Goup_Slave*. So, the every process with *rank=(k,i)* (if it is used the communicator with Decartes topology) or rank=*k*i* of the *K*I* total number of processes will receive from the process with rank *root* the following vector $\left(\left(a^{k}_{i1},b^{k}_{i1},c^{k}_{i1}\right) \cdots \left(a^{k}_{im},b^{k}_{im},c^{k}_{im}\right) \cdots \left(a^{k}_{iJ},b^{k}_{iJ},c^{k}_{iJ}\right)\right)$ for *k=1,...,K, i=1,...,I.*

[1] Latency is the time required to send an 8-byte message from on node to another of network communications using basic MPI routines.

Process with rank *p=k*i* for *k=1,...,K, i=1,...,I* using the received vector can construct the vectors $C_i^k=\left(c_{i1}^k,...,c_{im}^k,...,c_{iJ}^k\right)$, $B_i^k=\left(b_{i1}^k,...,b_{im}^k,...,b_{iJ}^k\right)$, $A_i^k=\left(a_{i1}^k,...,a_{im}^k,...,a_{iJ}^k\right)$ used for reduction operations.

Thus, the data parallelization of the algorithm to find Stackelberg equilibrium profile in the three level dynamic games with discrete payoff functions are finished.

The "task parallelism" of the backward induction algorithm is soft implemented using the reductions operation. This is presented of the following steps.

3) Using the reduction operation *MPI_MAXLOC* we construct a new reduction operation named *MPI_ALLMAXLOC* to compute all global maximum and all index attached to the rank of the process containing the maximum value. So, all the process with rank *p=k*i*, k=*1,...,K, i=1,...,I*, using the function **MPI_Reduce** with reduction operation *MPI_ALLMAXLOC* and the address of sent buffer represent by the vector C_i^k determine all of the values $j^*(k,i)=\arg\max_j\left\{c_{nj}^k\right\}$ and store these values on the buffer of the process with rank *root*. On the other hand using the reduction operation *MPI_ALLMAXLOC* we will determine the maximal element and its column indexes of the lines in the following matrice with *K*I* rows and *J* columns:

$$\begin{pmatrix} c_{11}^1 & \cdots & c_{1m}^1 & \cdots & c_{1J}^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{n1}^1 & \cdots & c_{nm}^1 & \cdots & c_{nJ}^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{I1}^K & \cdots & c_{Im}^K & \cdots & c_{IJ}^K \end{pmatrix}.$$

Therefore, simultaneously we solve the optimization problems of the third level of extensive form of a game (see Picture 2). Thus, using the function **MPI_Reduce** with reduction operation *MPI_ALLMAXLOC* the process with rank *root* will be obtained the vector of the length *K*I* with elements $J^*=\left(j^*(1,1),..,j^*(1,n),..,j^*(1,I),\cdots,j^*(l,1),..,j^*(l,n),..j^*(l,I),\cdots,j^*(K,1),..,j^*(K,n),..j^*(K,I)\right)$. After this using the MPI function **MPI_Scatter** each processes with rank *p=k, k=1,...,K* obtain from process with rank *root* the vector $J^*(k)=\left(j^*(k,1),..,j^*(k,n),..j^*(k,I)\right)$ which are using in the next step.

4) For all process with rank rank *p=k*, k=*1,...,K* using the function **MPI_Reduce** with reduction operation *MPI_ALLMAXLOC* and the address of sent buffer represent by the vector $\left(b_{1j^*(k,1)}^k,\ \cdots,\ b_{nj^*(k,n)}^k,\ \cdots,\ b_{Ij^*(k,I)}^k\right)$ all the process of the communicator *MPI_Comm_Slave* determine all of the values $i^*(k)=\arg\max_i\left\{b_{ij^*(k,i)}^k\right\}$ and store these values on the buffer of the process with rank *root*. In the other hand using the reduction operation *MPI_ALLMAXLOC* we will determine the maximal element and its column indexes of the lines in the following matrice with *K* rows and *I* columns:

$$\begin{pmatrix} b_{1j^*(1,1)}^1 & \cdots & b_{nj^*(1,n)}^1 & \cdots & b_{1j^*(1,I)}^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{1j^*(l,1)}^l & \cdots & b_{nj^*(l,n)}^l & \cdots & b_{Ij^*(l,I)}^l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{1j^*(K,1)}^K & \cdots & b_{nj^*(K,n)}^K & \cdots & b_{Ij^*(K,I)}^K \end{pmatrix}^T.$$

In other words, concomitantly we solve the optimization problems of the second level of extensive form of a game (see Picture 3). The total number of using processes is equal to *K*. Thus, using the function **MPI_Reduce** with reduction operation *MPI_ALLMAXLOC* it will be determined the vector of the length *K* with elements $I^*=\left(i^*(1),\cdots,i^*(l),\cdots,i^*(K)\right)$, which are stored in the memory of process *root*.

5) The process with rank *root* using the vectors $I^*=\left(i^*(1),...,i^*(l),...,i^*(K)\right)$ from step 4) and $\left(j^*(1,i^*(1)),...,j^*(l,i^*(l)),...,j^*(K,i^*(I))\right)$, which is obtained from the vector J^* from the step 3), realize

the step c) of the backward induction algorithm and prints the Stackelberg equilibrium profile $\left(k^*, i^*, j^*\right)$ of the three level dynamic game with discrete payoff functions. On the other hand, the *root* process determine the maximal element and its indexes of the following vector (see Picture 4)

$$\left(a^1_{i^*(1)j^*\left(1,i^*(1)\right)} \quad \cdots \quad a^l_{i^*(l)j^*\left(l,i^*(l)\right)} \quad \cdots \quad a^K_{i^*(K)j^*\left(K,i^*(1)\right)} \right).$$

By this, the description of the algorithm ends.

For evaluate the time performance of the parallel algorithm we use the following two numerical characteristics: total time to solve on a parallel computer with *P* processor and speedup achieved by parallel algorithm [4]. Speedup $S_P(K) = \frac{T(K)}{T_P(K)}$, where *T(K)* denote the sequential complexity of the algorithm, $T_P(K)$ denote the time to solve the problem using a parallel algorithm, measures the speedup factor obtained by parallel algorithm when *P* processors are available. Here *K* denote the input size of the problem.

Let us proof the following theorem about the correctness of the parallel algorithm described by the steps 1) - 5) and estimate the total time and the speedup to find the Stackelberg equilibrium profile in the three level dynamic game with discrete payoff functions using this algorithm.

Theorem. *Parallel algorithm, described by the steps 1)-5), correctly computes the Stackelberg equilibrium profile in the three level dynamic games with discrete payoff function. This algorithm runs in* $O(K+I+J))$ *time using a total of* $O(K \cdot I)$ *processors. The speedup of the parallel algorithm is equal to* $\frac{O(K \cdot I \cdot J) + O(K \cdot I) + O(K)}{O(K+I+J)}$.

Proof. The correctness proof of algorithm is straightforward. Suppose that the sequential algorithm to find the maximal element in a list $C_i^k = \left(c_{i1}^k, c_{i2}^k, \dots, c_{im}^k, \dots, c_{iJ}^k\right)$ is $O(J)$. Then the step a) of the backward induction algorithm for the dynamic game request $O(K \cdot I \cdot J)$ operations. Using the parallel system with *K*I* processors (processes or threads) the parallel variant of the algorithm, i.e. step 3), takes $O(J)$ operations. Similar, the step b) implemented on the sequential system takes $O(K \cdot I)$ operations. The parallel system with *K* processor the step 4) of the parallel algorithm take $O(I)$. Finally, the step c) takes $O(K)$ which is equal to the operations request of the step 5). Therefore, the parallel algorithm described by the steps 1)-5) takes $O(K+I+J))$ operations using $O(K \cdot I)$ processes (processors). Speedup of this algorithm is equal to

$$S_{O(K \cdot I)}(K \cdot I \cdot J) = \frac{O(K \cdot I \cdot J) + O(K \cdot I) + O(K)}{O(K+I+J)}.$$

References:

1. Von Stackelberg H. The Theory of the Market Economy. - Oxford: Oxford University Press, 1952.
2. Hancu Boris The full set of Stackelberg equilibrium profiles in the multilevel dynamic games. The 33-rd Annual Congress of the American Romanian Academy of Arts and Sciences (ARA). Sibiu, Romania. June 02-07, 2009. Proceedings, Volume II. Montreal: Polytechnic International Press, 2009, p.301-303.
3. MPI-2: Extensions to the Message-Passing Interface. Message Passing Interface Forum, 2003, November 15.
4. Joseph Jaja. An Introduction to Parallel Algorithms. - New York: Addison-Wesley Publishing Company, 1992.

Prezentat la 28.06.2010