DOI 10.12694/scpe.v19i4.1468
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 4, pp. 423–432. http://www.scpe.org

A COMPARISON OF MESSAGE PASSING INTERFACE (MPI) AND CO-ARRAY FORTRAN FOR LARGE FINITE ELEMENT VARIABLY SATURATED FLOW SIMULATIONS *

FRED T. TRACY† THOMAS C. OPPE‡ AND MAUREEN K. CORCORAN§

Abstract. The purpose of this research is to determine how well co-array FORTRAN (CAF) performs relative to Message Passing Interface (MPI) on unstructured mesh finite element groundwater modelling applications with large problem sizes and core counts. This research used almost 150 million nodes and 300 million 3-D prism elements. Results for both the Cray XE6 and Cray XC30 are given. A comparison of the ghost-node update algorithms with source code provided for both MPI and CAF is also presented.

Key words: Co-array FORTRAN, MPI, finite element method, variably saturated seepage flow modelling

AMS subject classifications. 35J66, 65Y05, 76S05

1. Introduction. Several parallel programming paradigms have been developed in recent years as alternatives to the popular software Message Passing Interface (MPI) [1] used for passing messages among the processes of a distributed memory parallelized program. One of these new ways is the Partitioned Global Address Space (PGAS) [2] paradigm where arrays partitioned across processes can be referenced by special syntax implemented in the language. A popular PGAS language for FORTRAN users is co-array FORTRAN (CAF) [3], and a CAF specification has been adopted in the FORTRAN 2008 standard. CAF has performed better than MPI for certain applications, and it was found easier to program than MPI. A recent example tested structured-grid partial-differential-equation applications [4].

A recent paper [7] describing the challenges and scalability results of running a large finite element model of variably saturated flow [5] in a three-dimensional (3-D) levee on a large high performance, parallel computer where MPI was used for the communication was published. Using the same levee model, this current research expands that work by using CAF for the communication and comparing these results with the results using MPI. The original finite element model consisted of 3,017,367 nodes and 5,836,072 3-D prism elements running on 32 cores, and the problem and core count were magnified as much as 350 times to achieve 1,044,246,303 nodes and 2,042,625,200 elements.

A traditional partitioning of the mesh achieved approximately the same number of finite element nodes on each core. Thus, the main communication challenge was updating ghost-node information on the different cores for a solution of a system of simultaneous, linear equations at each nonlinear iteration. In both the MPI and CAF versions, the ghost node data are first buffered and then sent to the different cores where they are needed. Details of the FORTRAN coding for both MPI and CAF are described herein.

2. Description of the problem. The problem consists of steady-state seepage flow through a levee as shown in Fig. 2.1 and idealised in Fig. 2.2 where there are several soil layers. A detailed description of this problem is given in [6, 7]. The challenges of parallelization using MPI of the groundwater program used in this research, when the problem size is approximately one billion nodes and two billion elements, are given in [7]. Performance results are also given. Fig. 2.3 shows a portion of the 3-D mesh of the levee system before a tree with its root system was added at the toe. More details of the modelling of the woody vegetation are given in [6]. To model the tree root at the toe of the levee, a 5 ft × 6 ft × 6 ft heterogeneous zone was added in which the mesh was refined using 1 in × 1 in × 1 in 3-D prism elements (Fig. 2.4). To simulate heterogeneities, a randomly generated hydraulic conductivity was assigned to each element in this zone. The resulting mesh consisted of 3,017,367 nodes and 5,836,072 3-D prism elements.

*This work was supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program (HPCMP).

†Information Technology Laboratory (ITL), Engineer Research and Development Center (ERDC), Vicksburg, MS.
‡ITL, ERDC, Vicksburg, MS.
§Geotechnical and Structures Laboratory, ERDC, Vicksburg, MS.

FIG. 2.1. *River side of a levee with trees.*

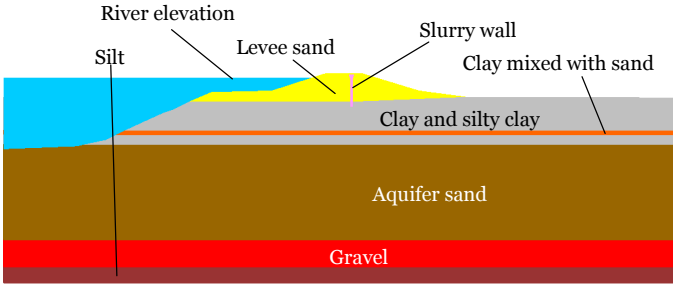

FIG. 2.2. *Cross section of a levee with material types and elevation of the river.*

3. High performance parallel computing. The parallel 3-D groundwater finite element program mentioned above was run on Garnet, the Cray XE6 at the U.S. Army Engineer Research and Development Center (ERDC) [8], and on Lightning, the Cray XC30 at the Air Force Research Laboratory, Aberdeen, MD [9]. At the time of this study, Garnet consisted of 4,716 dual-socket compute nodes with each socket populated with a 2.5 GHz 16-core AMD 6200 Opteron (Interlagos) processor. Each node had 64 GB memory (60 GB user-accessible) or an average of 1.875 GB memory per core. The interconnect type was Cray Gemini in a 3-D torus topology. Garnet was rated at 1.5 peak PFLOPS or 10 GFLOPS per core when these computations were done. Garnet had a large Lustre file system that was tuned for parallel I/O. At the time of this research, Lightning consisted of 2,360 dual-socket compute nodes with each socket populated with a 2.7 GHz 12-core Intel Xeon E5-2697v2 (Ivy Bridge) processor. Each node had 64 GB memory (63 GB user-accessible) or an average of 2.625 GB memory per core. The interconnect type was Cray Aries in a Dragonfly topology. Lightning was rated at 1.2 peak PFLOPS or 21.6 GFLOPS per core when the data in this paper were collected. Lightning had a large Lustre file system that was also tuned for parallel I/O.

The parallelization of the 3-D seepage/groundwater program was separated into four parts or programs. One MPI process or one CAF image was placed on each core of a compute node. The four programs are (1) a *partitioner* using the Parallel Graph Partitioning and Fill-reducing Matrix Ordering program, ParMETIS [10], to divide the mesh into approximately equal pieces among the MPI processes or CAF images, (2) a *preparer* to provide data, such as owned nodes, ghost nodes, owned elements, ghost elements, and communication data, needed for each MPI process or CAF image, (3) a *finite element program* that does the finite element computations with output files containing results for each owned node of that MPI process or CAF image, and (4) a *post processor* that combines all data from each MPI process or CAF image into the final output files.

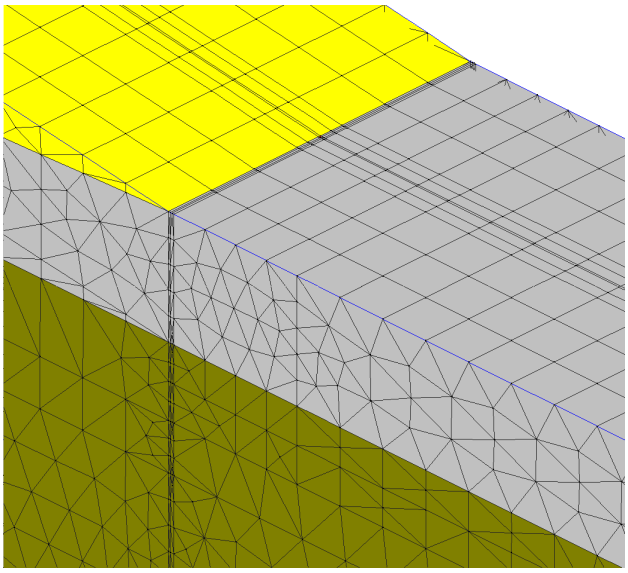

FIG. 2.3. *Portion of the 3-D mesh before the root zone was added.*

The primary communication challenge is ghost node updates in the conjugate gradient or BICG-STAB [11] linear solvers using either a Picard or Newton linearization [12, 13] of the governing nonlinear equation of Richards' equation [14]. As in [7], only times to solution for the finite element part of the program suite were collected for MPI and CAF and reported in this paper. The ghost node update routine for both MPI and CAF is examined in detail in the following section.

4. Ghost node update.

4.1. MPI. Table 4.1 gives the ghost node update subroutine for MPI and a description of the important variables. The subroutine has three steps: (1) a set-up phase in which data are to be received from the different cores, (2) send data to the different cores, and (3) wait until all the MPI messages have been processed. The arrays, *nstngh*, *ighost*, and *nodgh* have all been supplied by the preparer program. There are no global arrays in the parallel versions of the finite element program, i.e., no arrays with sizes of the total number of nodes or the total number of elements. The elimination of global arrays allowed for much larger finite element meshes to be run.

4.2. CAF. Table 4.2 gives the ghost node update subroutine for CAF and a description of the important variables. The same data provided in the MPI version were also provided to the CAF subroutine. The CAF version of ghost node updating is simpler than the MPI version in that for CAF, data are first placed in a buffer and then directly "put" into the different cores by the statement,

```
vc(nst :  nst + num - 1)[i] = buff(:)
```

While in the MPI case, the efficiency of the data transfer is dependent on how well `MPI_IRECV`, `MPI_SEND`, and `MPI_WAIT` are implemented, the efficiency of the CAF routine is dependent on the quality of the FORTRAN compiler implementation and internal data transfer capability. It is also important to note that two explicit barrier calls,

```
call sync_all
```

were required in the CAF implementation, whereas none were required in the MPI version. Also, the huge page option described by

```
module load craype-hugepages2M
```

was required to run the CAF version.

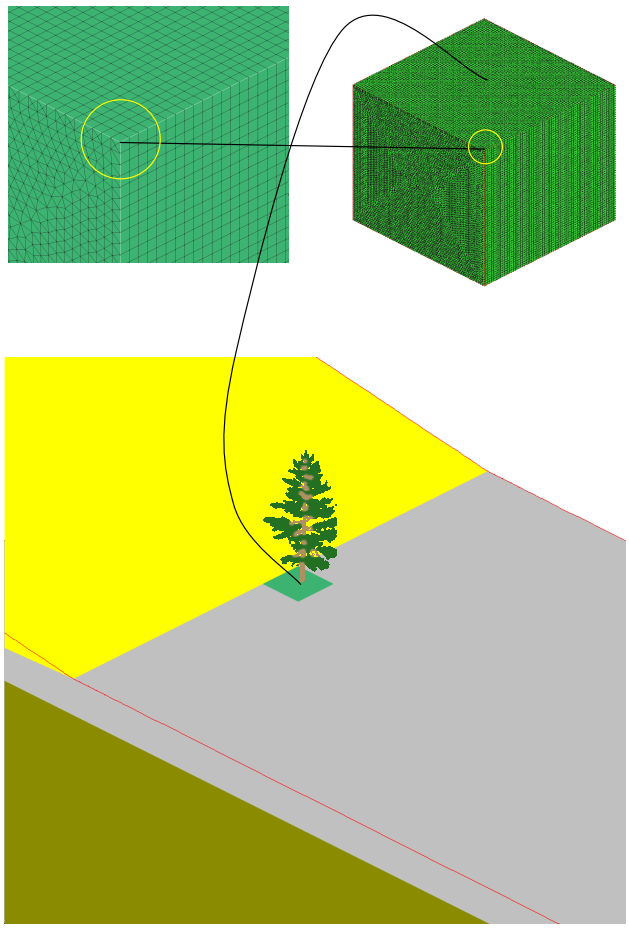

FIG. 2.4. *Heterogeneous zone representing the roots of a tree.*

5. Results and Analysis. All runs on both computers were done using the Cray compiler with -O1 optimisation.

5.1. Results. Tables 5.1 and 5.2 give the time to solution for the finite element program on the Cray XE6 and XC30 for different problem sizes and core counts. The m represents how much the original problem is magnified to produce larger problem sizes. When $m = 2$, for instance, two original meshes are created and joined such that the number of elements is exactly doubled, and the number of nodes is doubled less one set of the nodes common to the two pieces. The original problem was run with 96 cores. Although the MPI version of the finite element program could run this problem on 32 cores, the first multiple of 32 where the CAF version would run was 96. Values of $m = 1$, 2, 5, 10, 20, and 50 were run. The running times for MPI and CAF for three runs, their respective averages, and the ratio of MPI to CAF running times were tabulated for each m value and core count.

5.1.1. Analysis. The following observations are made:

- When $m = 1$ and the number of cores is 96, the ratio of MPI / CAF running times was almost equal.
- As m increased, this ratio got significantly smaller. The ratios become so small that m was not increased further than 50. Apparently, the global synchronisations required by the CAF implementation became increasingly costly as the partition size grew.
- The MPI/CAF ratio is larger for the XE6 than the XC30.
- The XC30 running times are approximately half of those of the XE6.

TABLE 4.1
Ghost node update for MPI.

Receive FORTRAN code

```
do i = 1, noproc
  num = numngh(i)
  if (num .ne. 0) then
    itag = 100
    nst = nstngh(i)
    call MPI_IRECV (v(nst), num, MPI_REAL8, i - 1, itag, MPI_COMM_WORLD, &
      ireq(i), ierror)
  end if
end do
```

Send FORTRAN code

```
allocate (buff(num_max))

do i = 1, noproc
  num = ighost(i + 1) - ighost(i)
  if (num .ne. 0) then
    do j = 1, num
      jloc = nodgh(ighost(i) + j)
      buff(j) = v(jloc)
    end do
    itag = 100
    call MPI_SEND (buff, num, MPI_REAL8, i - 1, itag, MPI_COMM_WORLD, ierror)
  end if
end do

deallocate (buff)
```

Wait FORTRAN code

```
do i = 1, noproc
  if (numngh(i) .ne. 0) then
    call MPI_WAIT (ireq(i), istat, ierror)
  end if
end do
```

```
    noproc = number of cores or MPI processes
 num_max = maximum number of ghost node data to send
       v = variable to be updated
  nstngh(i) = the starting address of v where data are to be received from core, i - 1
numngh(i) = the number of values be received in v from core, i - 1
     nodgh = array containing local node numbers whose data are to be sent to other cores
     ighost = array containing the accumulated number of ghost nodes whose data
              are to be sent
```

TABLE 4.2
Ghost node update for CAF.

Special CAF variables

common / caf / nstnghc(npmx)[*], vc(ndlmx)[*]

CAF put FORTRAN code

```
call sync_all
allocate (buff(num_max))

do i = 1, noproc
  num = ighost(i + 1) - ighost(i)
  if (num .ne. 0) then
    nst = nstnghc(image)[i]
    do j = 1, num
      jloc = nodgh(ighost(i) + j)
      buff(j) = v(jloc)
    end do
    vc(nst : nst + num - 1)[i] = buff(:)
  end if
end do

call sync_all
deallocate (buff)

do i = nnpown + 1, nnpl
  v(i) = vc(i)
end do
```

noproc = number of cores or CAF images
image = CAF image number
npmx = maximum number of CAF images
nnpl = number of local nodes
ndlmx = maximum number of local nodes
v = variable to be updated
vc = CAF array containing the updated ghost node data
nstnghc(i) = CAF array containing the starting address of v where data are to be received from core, i
nodgh = array containing local node numbers whose data are to be sent to other cores
ighost = array containing the accumulated number of ghost nodes whose data are to be sent

TABLE 5.1
Time (sec) for MPI and CAF on the Cray XE6 and XC30 for $m = 1, 2, and\ 5$.

m	Nodes	Elements	Cores	Cray	Time MPI		Time CAF		Ratio MPI/CAF
1	3017367	5836072	96	XE6	788.0		816.0		
				XC30	323.6		376.6		
				XE6	786.6		826.7		
				XC30	324.8		379.2		
				XE6	767.7		820.7		
				XC30	322.1		375.3		
				XE6	Avg.	780.8	Avg.	820.8	0.95
				XC30		323.5		377.0	0.86
			128	XE6	601.1		658.8		
				XC30	252.1		322.6		
				XE6	597.3		723.7		
				XC30	258.4		325.4		
				XE6	596.6		628.9		
				XC30	257.0		343.3		
				XE6	Avg.	598.3	Avg.	670.5	0.89
				XC30		255.8		330.4	0.77
2	6000831	11672144	192	XE6	804.1		1028.7		
				XC30	283.0		393.6		
				XE6	788.8		850.2		
				XC30	281.5		363.7		
				XE6	786.3		896.4		
				XC30	281.4		397.8		
				XE6	Avg.	793.1	Avg.	925.1	0.86
				XC30		282.0		385.0	0.73
			256	XE6	642.1		914.4		
				XC30	255.8		408.5		
				XE6	604.0		692.1		
				XC30	258.1		383.5		
				XE6	659.4		917.5		
				XC30	256.0		386.7		
				XE6	Avg.	645.2	Avg.	841.3	0.76
				XC30		256.6		392.9	0.65
5	14951223	29180360	480	XE6	878.3		1361.8		
				XC30	344.0		636.8		
				XE6	819.0		1292.5		
				XC30	347.3		630.1		
				XE6	829.4		1287.3		
				XC30	347.1		635.3		
				XE6	Avg.	842.2	Avg.	1313.9	0.64
				XC30		346.1		634.1	0.55
			640	XE6	708.8		1491.8		
				XC30	260.5		576.3		
				XE6	599.6		1254.0		
				XC30	264.4		671.2		
				XE6	663.4		1241.4		
				XC30	267.5		595.9		
				XE6	Avg.	657.3	Avg.	1328.1	0.49
				XC30		264.1		614.5	0.43

TABLE 5.2
Time (sec) for MPI and CAF on the Cray XE6 and XC30 for $m = 10, 20,$ and 50.

m	Nodes	Elements	Cores	Cray	Time MPI		Time CAF		Ratio MPI/CAF
10	3017367	58360720	960	XE6	862.0		1691.6		
				XC30	344.7		910.7		
				XE6	881.2		1730.2		
				XC30	353.1		912.2		
				XE6	867.5		1473.4		
				XC30	349.5		907.5		
				XE6	Avg.	870.2	Avg.	1631.7	0.53
				XC30		349.1		910.1	0.38
			1280	XE6	632.9		1309.0		
				XC30	266.9		995.1		
				XE6	624.6		1584.0		
				XC30	267.6		999.3		
				XE6	609.4		1768.6		
				XC30	267.6		998.2		
				XE6	Avg.	622.3	Avg.	1553.9	0.40
				XC30		267.4		997.5	0.27
20	59703183	116721440	1920	XE6	957.7		2685.8		
				XC30	351.0		1330.3		
				XE6	852.6		3031.6		
				XC30	356.3		1465.0		
				XE6	874.4		2760.2		
				XC30	363.9		1326.6		
				XE6	Avg.	894.9	Avg.	2825.9	0.32
				XC30		357.1		1374.0	0.26
			2560	XE6	659.8		2609.2		
				XC30	276.5		1496.1		
				XE6	651.8		2641.7		
				XC30	276.1		1755.5		
				XE6	651.2		2283.0		
				XC30	274.9		1473.7		
				XE6	Avg.	654.3	Avg.	2511.3	0.26
				XC30		275.8		1575.1	0.18
50	149207103	291803600	4800	XE6	882.9		5242.1		
				XC30	374.6		2990.1		
				XE6	883.0		5704.6		
				XC30	356.7		3133.1		
				XE6	923.0		5255.3		
				XC30	359.4		3140.2		
				XE6	Avg.	896.3	Avg.	5400.7	0.17
				XC30		363.6		3087.8	0.12
			6400	XE6	703.2		5792.8		
				XC30	306.9		3801.3		
				XE6	739.6		6902.2		
				XC30	302.3		3833.0		
				XE6	749.1		6256.4		
				XC30	300.3		3796.7		
				XE6	Avg.	730.6	Avg.	6317.1	0.12
				XC30		303.2		3810.3	0.08

TABLE 6.1
Consistency check comparing values of pressure head from the original mesh and the mesh for $m = 50$ for the first 10 nodes and last 6 nodes of each mesh.

$m = 1$			$m = 50$		
Node	MPI	CAF	Node	MPI	CAF
1	129.00000	129.00000	1	129.00000	129.00000
2	128.99678	128.99678	2	128.99678	128.99678
3	128.99345	128.99345	3	128.99345	128.99345
4	119.00000	119.00000	4	119.00000	119.00000
5	118.99735	118.99735	5	118.99735	118.99735
6	124.23808	124.23808	6	124.23808	124.23808
7	118.99464	118.99464	7	118.99464	118.99464
8	123.54520	123.54520	8	123.54520	123.54520
9	128.98993	128.98993	9	128.98993	128.98993
10	110.50000	110.50000	10	110.50000	110.50000
6000826	0.0000000	0.0000000	149207098	0.0000000	0.0000000
6000827	0.041718483	0.041718484	149207099	0.041718479	0.041718479
6000828	3.0365778	3.0365778	149207100	3.0365779	3.0365779
6000829	0.036494873	0.036494870	149207101	0.036494875	0.036494875
6000830	0.0000000	0.0000000	149207102	0.0000000	0.0000000
6000831	0.0000000	0.0000000	149207103	0.0000000	0.0000000

6. Consistency check. A check for consistency for the first 10 nodes and last 6 nodes of the meshes for $m = 1$ and $m = 50$ was done with pressure head results placed in Table 6.1. The MPI and CAF results should be the same and because of symmetry, the values for $m = 1$ and $m = 50$ should also be the same. A comparison of the MPI and CAF results in Table 6.1 shows excellent consistency.

7. Conclusions. The following conclusions can be drawn:

1. Both MPI and CAF versions ran successfully and gave the same results.
2. As the problem size and process count increased, the results remained consistent.
3. The update routine for CAF was simpler than that for MPI.
4. CAF required more memory than MPI to run the same size mesh.
5. CAF required huge pages, but MPI did not.
6. CAF required explicit barriers, but MPI did not.
7. For the original problem, CAF and MPI performed almost the same when using 96 cores.
8. As the problem size and process count grew, MPI performed much better than CAF.
9. The MPI/CAF ratio is larger for the XE6 than the XC30.
10. The XC30 running times are approximately half of those of the XE6.

REFERENCES

[1] MESSAGE PASSING INTERFACE FORUM, *MPI: A Message-Passing Interface Standard, Version 3.0*, http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf, 2012.
[2] PGAS, *Partitioned Global Address Space*, http://pgas.org, 2015.
[3] R. W. NUMRICH AND J. K. REID, *Co-arrays in the next FORTRAN Standard*, Scientific Programming, 14 (2006), pp. 1-18.
[4] S. GARAIN, D. S. BALSARA, AND J. REID, *Comparing Co-array FORTRAN (CAF) with MPI for several structured mesh PDE applications*, J. of Comp. Physics, 297 (2015), pp. 237-253.
[5] J. ISTOK, *Groundwater modelling by the finite element method*, AGU, 1989.
[6] M. CORCORAN, J. PETERS, J. DUNBAR, J. LLOPIS, F. TRACY, J. WIBOWO, J. SIMMS, C. KEES, S. MCKAY, J. FISCHENICH, M. FARTHING, M. GLYNN, B. ROBBINS, R. STRANGE, M. SCHULTZ, J. CLARKE, T. BERRY, C. LITTLE, AND L. LEE, *Initial research into the effects of woody vegetation on levees, volume I of IV: project overview, volume II of IV: field data collection, volume III of IV: numerical model simulation, and volume IV of IV: summary of results and conclusions*, U.S. Army Engineer Research and Development Center, Vicksburg, MS, 2011.
[7] F. T. TRACY, T. C. OPPE, W. A. WARD, AND M. K. CORCORAN, *A scalability study using supercomputers for huge finite element variably saturated flow simulations*, Scalable Computing: Practice and Experience, 16 (2015), pp. 153-162.

[8] ERDC DSRC, http://www.erdc.hpc.mil/hardware/index.html, Department of Defense Supercomputing Resource Center, Vicksburg, MS, 2014.
[9] AFRL DSRC, http://www.afrl.hpc.mil/index.html, Department of Defense Supercomputing Resource Center, Aberdeen, MD, 2014.
[10] G. Karypis, *ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering*, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview, 2014.
[11] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003.
[12] S. Mehl, *Use of Picard and Newton iteration for solving nonlinear ground water flow equations*, Ground Water, 44 (2006), pp. 583-594.
[13] C. T. Kelley, *Solving nonlinear equations with Newton's method*, SIAM, 2003.
[14] L. A. Richards, *Capillary conduction of liquids through porous mediums*, J. of Physics, 1 (1931), pp. 318-333.

Edited by: Dana Petcu
Received: Nov 23, 2018
Accepted: Dec 23, 2018