Parallelism on Hybrid Metaheuristics for Vector Autoregression Models

Alfonso L. Castaño
Faculty of Computer Science
University of Murcia
Murcia, Spain
alfonsoluis.castanom@um.es

Javier Cuenca
Faculty of Computer Science
University of Murcia
Murcia, Spain
jcuenca@um.es

José-Matías Cutillas-Lozano
Faculty of Computer Science
University of Murcia
Murcia, Spain
josematias.cutillas@um.es

Domingo Giménez
Faculty of Computer Science
University of Murcia
Murcia, Spain
domingo@um.es

Jose J. López-Espín
Center of Operations Research
Miguel Hernández University
Elche, Spain
jlopez@umh.es

Alberto Pérez-Bernabeu
Center of Operations Research
Miguel Hernández University
Elche, Spain
apb74@alu.ua.es

***Abstract*—Vector Autoregression Models are multi-equation models that linearly describe the simultaneous interactions and behavior among a group of variables, using only their own past. They have been traditionally used in finance and econometrics, but, with the arrival of Big Data, huge amounts of data are being collected in numerous fields and their use for other fields is being considered. Tools are available for these models, but the huge amount of data makes it necessary to exploit High-Performance Computing for the acceleration of methods to obtain the models. This paper considers a matrix formulation to represent time dependencies, and the solution of the optimization problem generated is approached through hybrid metaheuristics. The parameterized, parallel implementation of the metaheuristics and the matrix formulation ease the exploitation of multilevel shared-memory parallelism.**

***Index Terms*—vector autoregression models, time series, metaheuristics, shared-memory, GPU**

I. Introduction

Vector Autoregression (VAR) models [1] are multi-equation models that linearly describe the simultaneous interactions and behavior of a group of variables, using only their own past. More specifically, a VAR is a model of simultaneous equations formed by a system of equations in which the contemporary values of model variables do not appear in any explanatory variable in the equations. The set of explanatory variables in each equation is a block consisting of lags of each of the model variables, and the block is the same for all the equations.

VAR models have traditionally been used in finance and econometrics [2], [3]. With the arrival of Big Data, huge amounts of data are being collected in numerous fields. We are studying the application of statistical models in health problems which, conventionally, have been applied in econometrics [4]. We use Vector Autoregression Models (VAR) to model time series. Tools exist to tackle this problem [5], but the large amount of data, along with the availability of computational techniques and high performance systems, advise an in-depth analysis of the computational aspects of VAR, so large models can be solved efficiently with today's computational systems.

A matrix formulation of the time series of VAR is presented. The problem of finding the optimum VAR model for a given series is an optimization problem whose solution can be approached through metaheuristics. The application of parameterized metaheuristic schemes have proved to be a practical approach for the determination of satisfactory metaheuristics for several problems [6], [7]. Additionally, the parameterized scheme facilitates the development and optimization of parallel implementations [8], [9]. This paper analyzes the application of shared-memory parallelism to determine VAR models through hybrid metaheuristics developed on top of a parameterized metaheuristic scheme.

There are different strategies to solve this problem [10]. Ordinary Least Squares (OLS) is used for the fitness of the metaheuristics. Other information criteria could be used, of which the most well-known and used (but not the only ones) are Akaike (AIC) [11], Schwarz (BIC) [12] and Hannan-Quinn (HQC) [13]. All these criteria have similar computational costs, and the study of the goodness of the metaheuristics and their parallelization is not affected by the criterion.

The rest of the paper is organized as follows. Section II describes the matrix formulation of the time series used in this paper. The hybrid metaheuristics considered for approaching the time series model are analyzed in Section III, while their parallelization for shared-memory is studied in Section IV. The results of some experiments are shown in Section V, which provides experimental evidence of the advantages of the parameterized scheme and the matrix formulation for obtaining parallel and efficient versions of the metaheuristics. Some ongoing research lines are outlined in Section VI, and Section VII concludes the paper.

This work was supported by the Spanish MINECO, as well as European Commission FEDER funds, under grants TIN2016-80565 and TIN2015-66972-C5-3-R.

978-1-5386-7879-4/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCS.2018.00134

II. Matrix Formulation for Vector Autoregression Models

There is ample literature on time series [14], most of it devoted to econometrics and finance [2], [3], and there are many types of time series. VAR are based on the idea that the value of a variable at a time point depends linearly on the value of one or more variables at previous instants of time. Since, most of the time, not all the variables will follow this pattern, there will be dependent (endogenous) and independent (exogenous) variables. The latter will not be analyzed by the model, but will be used to model dependent variables. A matrix formulation is used to facilitate application of High-Performance techniques to the problem.

We consider d dependent parameters at t time instants. The vector of parameters at an instant i is represented by $y^{(i)}$. So, the time series Y is a matrix of dimension $t \times d$. Similarly, vectors $z^{(i)}$, $1 \leq i \leq t$, represent the values of independent variables; each vector $z^{(i)}$ is of size e and the series of independent variables is a matrix Z of dimension $t \times e$. So, the matrices which determine the time series are

$$\begin{pmatrix} y_1^{(1)} & \dots & y_d^{(1)} \\ \vdots & \ddots & \vdots \\ y_1^{(t)} & \dots & y_d^{(t)} \end{pmatrix} \quad \begin{pmatrix} z_1^{(1)} & \dots & z_e^{(1)} \\ \vdots & \ddots & \vdots \\ z_1^{(t)} & \dots & z_e^{(t)} \end{pmatrix} \tag{1}$$

The values of the endogenous variables at instant i depend on those of t_y and t_z previous instants for the endogenous and exogenous variables. The dependencies on the endogenous variables are represented by matrices $A_i \in R^{d \times d}$, $1 \leq i \leq t_y$, and those on the exogenous ones by matrices $B_i \in R^{e \times d}$, $1 \leq i \leq t_z$. The dependencies can be represented as

$$\begin{aligned} y^{(j)} \approx y^{(j-1)}A_1 + y^{(j-2)}A_2 + \ldots + y^{(j-t_y)}A_{t_y} + \\ z^{(j-1)}B_1 + z^{(j-2)}B_2 + \ldots + z^{(j-t_e)}B_{t_e} + C \end{aligned} \tag{2}$$

where C is a vector $1 \times d$ of constant values.

The dependencies can be represented in matrix form, with the values of the series at instants $t_y + 1$ to t depending on the previous ones:

$$\begin{aligned} Y(t_y + 1 : t, :) \approx \\ Y(t_y : t-1, :)A_1 + Y(t_y - 1 : t - 2, :)A_2 \\ + \ldots + Y(1 : t - t_y, :)A_{t_y} + \\ Z(t_y : t-1, :)B_1 + Z(t_y - 1 : t - 2, :)B_2 \\ + \ldots + Z(t_y - t_z + 1 : t - t_z, :)B_{t_z} + C \end{aligned} \tag{3}$$

Alternatively, if the vectors at previous instants are represented in a matrix X of dimension $(t - t_y) \times (d \cdot t_y + e \cdot t_z + 1)$

$$\begin{pmatrix} y^{(t_y)} & \dots & y^{(1)} & z^{(t_y)} & \dots & z^{(t_y - t_z + 1)} & 1 \\ & \vdots & & & \vdots & & \vdots \\ y^{(t-1)} & \dots & y^{(t-ty)} & z^{(t-1)} & \dots & z^{(t-t_z)} & 1 \end{pmatrix} \tag{4}$$

Equation (3) can be represented as

$$\begin{pmatrix} y^{(t_y+1)} \\ \vdots \\ y^{(t)} \end{pmatrix} \approx X * \begin{pmatrix} A_1 \\ \vdots \\ A_{t_y} \\ B_1 \\ \vdots \\ B_{t_z} \\ C \end{pmatrix} \tag{5}$$

where the matrix on the right (built from matrices A_i, B_i and C) represents the model of the time series. We call this matrix A, of dimension $(d \cdot t_y + e \cdot t_z + 1) \times d$.

With this formulation, the optimization problem to be solved consists of the determination of the model (matrix A) which best represents the dependencies on the time series:

$$\min_A \|Y(t_y + 1 : t, :) - X * A\| \tag{6}$$

The problem can be approached by Ordinary Least Squares (OLS). Here, local search and distributed metaheuristic algorithms and hybridations are considered. The basic distributed metaheuristics considered are Genetic Algorithms [15] and Scatter Search [16], in combination with Local Search [17] and Tabu Search [18].

For each possible model (matrix A, an element of the metaheuristic) the norm in (6) represents the fitness, whose computation has a cost of order $O((t - t_y) \cdot (d \cdot t_y + e \cdot t_z + 1) \cdot d)$, which means a high computational cost in the application of metaheuristics for large time series with large dependencies. So, matrix computation techniques should be used to reduce the execution time [19]. These techniques are not considered in this paper, which is devoted to the analysis of the application of hybrid metaheuristics and their parallelization. If optimized matrix operations were used, the execution time would be reduced, with no modifications to the metaheuristic and parallel scheme.

III. Basic and Hybrid Metaheuristics

The representation of the candidate elements and their fitness are common to all the metaheuristics considered:

- A candidate solution is a matrix A (A_i, B_i and C in (5)) of size $(d \cdot t_y + e \cdot t_z + 1) \times d$. Different types of problems can be stated by imposing restrictions on the values in A. For example, the values can be in an interval or a set of integers or real numbers. LAPACK [20] can be used for the solution of (6) for the real, continuous problem. The metaheuristic approach, generally with a larger execution time, can be used for the different versions of the problem. The results in the experimental section were obtained for solutions in an interval of real numbers, but this work focuses on the exploitation of parallelism, and the conclusions do not change significantly for the different versions.
- The fitness for a candidate solution A is a measure of how the time series Y is approached with the model A.

The norm $\|Y - X * A\|$ is used in the experiments. Other statistical criteria (AIC, BIC, HQC, etc.) can be used, with different costs for the computation of the fitness. The differences in the solutions with the different criteria are residual, and now again the parallelism is not influenced by the fitness function.

The main characteristics of the basic metaheuristics considered are commented on. Local search and distributed (population-based) metaheuristics are combined in a hybrid approach.

A. Local Search

Local Search methods work by analyzing the neighborhood of the candidate elements. The neighborhood considered for a vector with $(d \cdot t_y + e \cdot t_z + 1) \cdot d$ entries consists of $2 \cdot (d \cdot t_y + e \cdot t_z + 1) \cdot d$ elements, which are obtained by adding and subtracting a certain value to each position of the vector. In the continuous version of the problem, any quantity could be selected to be added or subtracted. Each time an entry with value v is modified, the value to modify it is randomly selected in the interval $[0.001, v/10]$, which means the minimum modification is 0.001, and the entry is not modified by more than a 10% of its value. Other intervals were considered, but this interval gave satisfactory experimental results.

The fitness is calculated for all the elements generated in the neighborhood of an element. The cost of the computation of the fitnesses is of order $O\left((t - t_y) \cdot (d \cdot t_y + e \cdot t_z + 1)^2 \cdot d^2\right)$. So, the computation for the best neighbor can be very costly for large neighborhoods and time series, and if needed the number of elements in the neighborhoods could be established at a fixed value, with random selection of the entries to be modified.

If the best element in the neighborhood is better than the active element, it becomes the new active element. If not, the search continues with the same element, and the new neighborhood to be explored changes due to the random generation of the neighbors. A maximum number of iterations from an initial value can be established, and after this number a new active element can be generated for a new search, so giving a GRASP method [21].

B. Tabu Search

Tabu Search [18] is used to direct the search. A list of tabu decisions is established, and the same decision can not be taken while it is in the tabu list. The idea is to diversify the search in different directions to avoid local optima. In our implementation, the last positions modified in the vector v are maintained in the list. Short lists gave satisfactory experimental results, maybe due to the random selection of the amplitude of the modification of the entries in the active candidate.

A long-term tabu strategy is used with a tabu element which is the mean of the best elements in a number of the last iterations. It is used in the selection of elements to be explored. Elements close to this tabu element are discarded. The idea is to avoid concentrating the search in the same area of the solution space.

C. Genetic Algorithms

The previous methods are local search metaheuristics, which can be hybridized with distributed metaheuristics. The most popular distributed (or population based) metaheuristic is Genetic Algorithm [15]. The way in which the basic functions of the GA have been implemented is briefly explained:

- A population is initially generated, with the values of each entry of each individual of the population randomly generated in a given interval. The population generated can be improved with a Local or Tabu Search, so obtaining a hybrid metaheuristic.
- A number of iterations are performed on the population, with the combination of some elements to generate descending elements which substitute the ascending elements if their fitness is better. GA select elements to be combined with more probability for the best elements (roulette selection). In out implementation, the best elements are always selected, and some elements from the worst ones are randomly selected. Combinations of best elements, of best and worst and of worst elements are generated. When the number of combinations of best elements is large, the method is closer to a pure GA.
- The combination on a GA can also be carried out in different ways. A typical approach is to combine two ascendants by selecting a middle point at which they are crossed: two descendants are generated, each with the initial half of one ascendant and the final half of the other. This typical combination does not make sense for our problem, because it can generate combinations of elements in which the whole part of the vector corresponding to the endogenous or exogenous elements is unchanged. So, a simplified Path Relinking [22] approach is used: given two ascendants v_1 and v_2, the two descendants are in the path connecting them, $\frac{1}{3}v_1 + \frac{2}{3}v_2$ and $\frac{2}{3}v_1 + \frac{1}{3}v_2$.
- The mutation contributes to the diversification of the search. A low percentage of individuals are selected to be mutated. An entry v of the individual is randomly selected, and it is updated by randomly adding or subtracting a random value in the interval $[0, v]$. The modification is in general larger than when analyzing the neighborhood, and so mutation can contribute to avoid local optima. The elements obtained by mutation are likely to die, and to prevent their early death, they can be improved with a few iterations of Local Search.
- The best elements from those in the original population and those generated with combination and mutation are included in the population for the next iteration. The inclusion of a few of the nonbest elements can contribute to diversify the search, as could the use of the long-term tabu strategy, which can be hybridized with GA.

D. Scatter Search

Scatter Search (SS) is another popular distributed metaheuristic [22]. It differs from GA in the systematic application of intensification and in the way in which diversification is achieved. The main differences are:

- An initial set of elements is randomly generated as in GA, but it is normally smaller than for GA. The elements are improved before starting the iterations.
- Typically, all the elements are combined, in pairs or bigger groups. In our implementation, the combination is like that for GA, but the percentages of best and worst elements selected for combination are similar, and a large number of combinations is carried out, and the elements so obtained are improved.
- The diversification is carried out through the selection and combination of nonpromising elements, and so mutation is not applied.
- The percentage of nonpromising elements to be included in the reference set is now higher, again to help diversification.

E. Hybrid metaheuristics

Metaheuristics can be hybridized in many ways [23], and hybridation with exact methods is also possible [24]. Some hybridation possibilities have been mentioned for the basic methods in the previous subsections. Metaheuristics can be developed with a unified scheme [25], which can in turn be parameterized [6]. The parameterized scheme used here for hybrid metaheuristics is shown in Algorithm 1. Each basic function includes some parameters whose value is selected to obtain basic metaheuristics or hybridations. The hybridation of local search and distributed methods is achieved with the inclusion of improvements at two parts of the scheme.

Algorithm 1 Parameterized scheme for hybrid metaheuristics

```
Initialize(Sini,ParamIni)
Improve(Sini,Sref,ParamImpIni)
while not EndCondition(Sref,ParamEndCon) do
   Select(Sref,Ssel,ParamSel)
   Combine(Ssel,Scom,ParamCom)
   Diversify(Sref,Scom,Sdiv,ParamDiv)
   Improve(Scom,Sdiv,ParamImp)
   Include(Scom,Sdiv,Sref,ParamInc)
end while
```

The general ideas for each function and the meaning of their sets and parameters are commented on. The parameters are summarized in Table I.

- The only parameter for the initialization is the number of elements which are randomly generated for the set `Sini` (ParamIni=$\{INEIni\}$). When only one value is generated we obtain a local search method, and the value is normally smaller for SS than for GA.
- A percentage $PEIIni$ of the elements of `Sini` are improved, with an intensification of the improvement ($IIEIni$), and the best $FNEIni$ elements are selected for the reference set (`Sref`) to work with in the successive iterations. For Tabu Search, $LTLIni$ is the length of the tabu list.
- The iterations finish after $MNIEnd$ steps or after $MIREnd$ steps without improving the best solution. A time limit can also be established.
- Some elements of the reference set are selected (set `Ssel`). A large number of best elements ($NBESel$) would give a method close to a GA, while a large number of elements from the worst ones ($NWESel$) contributes to diversification.
- A number of combinations between best elements, best-worst elements and worst elements are made ($NBBCom$, $NBWCom$ and $NWWCom$), with more combinations between best elements for more intensification of the search and more combinations with worst elements for diversification.
- A percentage of elements ($PEDDiv$) from the reference set and the elements obtained by combination are diversified.
- The same improvement routine used after initialization is called again, now to improve some of the elements obtained by combination (a percentage $PEIImp$ of elements to be improved, with intensity $IIEImp$) and the elements obtained by diversification, which are improved with intensity $IIDImp$. $LTLImp$ and $LTDImp$ are the lengths of the corresponding tabu lists.
- The best $NBEInc$ elements (from the three sets `Sref`, `Scom`, `Sdiv`) are included in the reference set for the next iteration, and the other $FNEIni$-$NBEInc$ elements are selected from the remaining ones. The parameter $LTMInc$ indicates the number of previous iterations for the computation of the mean solution of the long-term tabu memory.

IV. Shared-memory Hybrid Metaheuristics

Metaheuristics can be parallelized in a number of ways [26], [27], and there are works on parallelization of each of the basic metaheuristics considered (Local Search [28], Tabu Search [29], GA [30]) and for different types of computational systems (GPU [31]). The unified, parameterized scheme enables the simultaneous implementation of parallel versions of different basic metaheuristics and their hybridations for different types of computational systems (e.g., shared-memory [8] and heterogeneous clusters [9]).

Here, we discuss and analyze the possibilities of exploitation of shared-memory parallelism in nodes with multicore CPUs. The parallelism is implemented with independent parallelization of each basic routine in Algorithm 1. The parallelization in the routines `Initialize`, `Combine`, `Diversify` and `Include` consists of just parallelization of a loop, with dynamic assignation of the steps of the loop to a pool of threads. The improvement function has higher computational cost, and two levels of parallelism are used, the first to distribute the set of elements to be improved, and the second to assign different areas of the neighborhood to different threads.

TABLE I
METAHEURISTIC PARAMETERS IN THE PARAMETERIZED METAHEURISTIC SCHEME

$INEIni$	Initial Number of Elements
$PEIIni$	Percentage of Elements to be Improved in the initialization
$IIEIni$	Intensification in the Improvement of Elements
$LTLIni$	Length of the Tabu List for local search
$FNEIni$	Final Number of Elements for successive iterations
$MNIEnd$	Maximum Number of Iterations
$MIREnd$	Maximum number of Iterations with Repetition
$NBESel$	Number of Best Elements selected for combination
$NWESel$	Number of Worst Elements selected for combination
$NBBCom$	Number of Best-Best elements combinations
$NBWCom$	Number of Best-Worst elements combinations
$NWWCom$	Number of Worst-Worst elements combinations
$PEDDiv$	Percentage of Elements to be Diversified
$PEIImp$	Percentage of Elements obtained by combination to be Improved
$IIEImp$	Intensification of the Improvement of Elements obtained by combination
$IIDImp$	Intensification of the Improvement of elements obtained by Diversification
$LTLImp$	Length of the Tabu List for local search on elements obtained by combination
$LTDImp$	Length of the Tabu list for local search on elements obtained by Diversification
$NBEInc$	Number of Best Elements included in the reference set for the next iteration
$LTMInc$	Long-Term Memory size for the selection of elements to be included in the reference set

Additionally, for our approximation to the time series problem, the matrix operations (matrix multiplication and computation of the norm) can be done in parallel, and there are three parallelism levels in total.

The three levels are exploited in shared-memory with multilevel parallelism in OpenMP [32], with a different number of threads at each level depending on the amount of work at each level and the characteristics of the computational system (number of cores and computational capacity).

V. EXPERIMENTAL RESULTS

Experiments were carried out in nodes of a cluster with five nodes of four types:

- **marte** (and its twin node **mercurio**): hexa-core CPU AMD Phenom II X6 1075T at 3.00 GHz, with architecture x86-64, 16 GB of RAM, private L1 and L2 caches of 64 KB and 512 KB, and a L3 cache of 6 MB, shared by all the cores.
- **saturno**: 4 CPU Intel Xeon E7530 (hexa-core) at 1.87 GHz, with architecture x86-64, 32 GB of RAM, private L1 and L2 caches of 32 KB and 256 KB per core, and a L3 cache of 12 MB shared by all the cores in each socket.
- **jupiter**: 2 CPU Intel Xeon E5-2620 (hexa-core) at 2.00 GHz, with architecture x86-64, 32 GB of RAM memory, private L1 and L2 caches of 32 KB and 256 KB per node, and a L3 cache of 15 MB shared by all the cores of a socket.
- **venus**: 2 CPU Intel Xeon E5-2620 (hexa-core) at 2.40 GHz, architecture x86-64, 64 GB of RAM memory, private L1 and L2 caches of 32 KB and 256 KB per node, and L3 cache of 15 MB shared by all the cores in a socket.

Parallelism can be exploited at three levels:

- The highest level corresponds to the parallelization of the loops for treating the elements in each basic function of Algorithm 1. The parallelization at this level is preferable when the number of elements of the sets is large enough; but there are few possibilities of parallelism for local search methods or for medium size sets in computational systems with many cores.
- Medium level parallelism can be exploited in the analysis of the neighborhood in the improvement functions. In our implementation the neighborhood has $2 \cdot (d \cdot t_y + e \cdot t_z + 1) \cdot d$ elements, so there are enough elements for the exploitation of parallelism when the number of data and time dependencies are not too small.
- At the lowest level is the computation of the fitness function, (6). To take advantage of shared-memory parallelism at this level the size of the problem should be large to compensate the cost of thread management.

The exploitation of parallelism at the three levels is initially analyzed in the four nodes where experiments were carried out. The vector of metaheuristic parameters is $(80, 20, 50, 10, 1, 10, 5, 10, 10, 5, 50, 10, 1, 10, 20, 1, 15, 3)$: 80 elements are randomly generated, 50% of them are improved, with 10 steps of exploration of the neighborhood and a tabu list of only one element; the reference set for the successive iterations comprises 20 elements; the best 10 elements and 5 elements from the remaining ones are selected for combinations, with 10, 10 and 5 combinations between best, best-worst and worst elements; 50% of the elements in the reference set and those obtained by combination are improved, with intensity 10 and tabu list with 1 element; 20% of the elements are mutated, and the elements generated are improved with 20 steps of improvement and tabu list with 1 element; the 15 best elements and five elements from the remaining ones form the reference set for the next iteration; and information of the last 3 iterations is used to diversify the nonbest elements selected. Experiments are conducted for two problems: a small problem (SP) with $t = 200$, $d = 4$, $e = 2$, $t_y = 3$ and $t_z = 2$; and a large problem (LP) with $t = 600$, $d = 8$, $e = 6$, $t_y = 7$ and $t_z = 6$. Fig. 1 shows the speed-up for SP (left) and LP (right), for the four computational nodes considered. A similar behavior is observed in the four nodes, and its use allows us to draw system-independent conclusions:

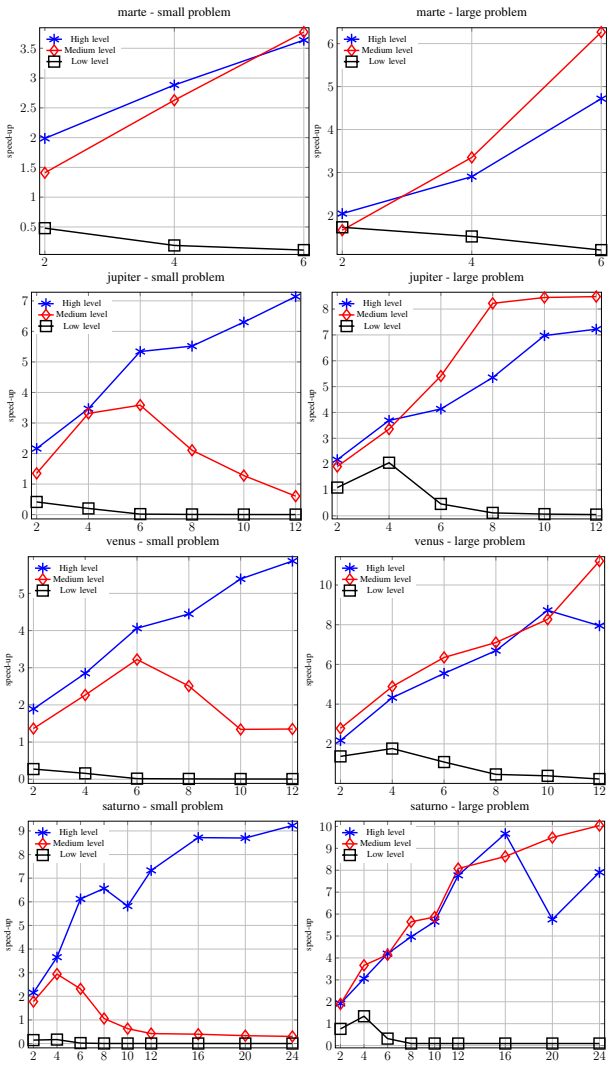

Fig. 1. Speed-up in four computational nodes with parallelism at three levels, varying the number of threads, for small (left) and large (right) problems.

- The exploitation of parallelism at the highest level is the preferred option for SP.
- The performance with medium level parallelism decreases for SP when the number of threads increases.
- For LP the speed-up with medium and large parallelism is similar, with medium-level parallelism being slightly better.
- The exploitation of low level parallelism is far worse than with the other types of parallelism. Only for LP and a small number of threads is the speed-up larger than one.

The preferred number of threads at each of the three levels can be different depending on the values of the metaheuristic parameters. So, a different number of threads can be fixed for each level in each of the basic functions in Algorithm 1. Fig. 2 analyses three-level parallelism; it shows the speed-up with different combinations of threads at each level, with a total of six and twelve threads, and for two metaheuristics working with small sets (the small metaheuristic (SM) corresponds to the vector of metaheuristic parameters $(10, 5, 50, 5, 1, 2, 1, 2, 1, 0, 50, 5, 1, 10, 5, 1, 3, 1)$, and the large metaheuristic (LM) to $(20, 10, 50, 5, 1, 4, 2, 4, 2, 0, 50, 5, 1, 10, 5, 1, 6, 1)$). The size of the time series are those of LP. Comparing the speed-up with the different combinations the conclusions are:

- Working with small sets (SM) propitiates the use of parallelism at a low level. For six threads, the combination $1 \times 6 \times 1$ (parallelism with 6 threads in the middle level) gives the highest speed-up, with a mean of 5.11, while for the combination $2 \times 3 \times 1$ the mean is 4.62. For twelve threads the preferred combination changes, with a mean speed-up of 5.58 for $1 \times 6 \times 1$ and of 5.85 for $1 \times 3 \times 4$. In one case (**saturno** with 12 threads for SM) the best combination is $1 \times 6 \times 2$.
- The speed-up with 12 threads is far from the optimum, which is due to the size of the sets with which the metaheuristics work, which means the amount of computation is not enough to fully exploit the computational capacity of the computational systems.
- For LM, six threads give better performance than twelve, which may be due to a larger amount of memory being accessed by more threads.

VI. Ongoing Work

Nowadays computational nodes comprise multicore CPUs plus one or more GPUs, so we are working in the exploitation of the heterogeneous CPU+GPU parallelism. GPUs parallelism can be exploited at different levels. In any case, the data of the problem (matrices Y and X and the dimensions) are transfered from CPU to the GPUs before starting the computation, so the transferences during the computation correspond to elements of the sets and their fitness.

- The highest level corresponds to an island scheme, with the reference set divided in subsets which are each assigned to a different GPU. If only one GPU is available, all the computation should be carried out in that GPU.
- The highest level of the shared-memory version corresponds to the parallelization of the loops for the treatment of elements. The steps of a loop are assigned to different threads, with one thread per GPU, and each thread calls its GPU to work with the corresponding element, which is transfered from CPU to GPU for the computation, and the required results (the elements generated together with the fitness) are transfered back from GPU to CPU. In this way the GPU computes the fitness of the element or explores its neighborhood in the improvement function.
- GPUs can work at the second level of the shared-memory version: the analysis of the neighborhood of each element would be done in parallel, with each GPU working in one area of the neighborhood, obtaining the best neighbor in its area, which is transfered together with its fitness back to the CPU, which computes the best neighbor before the next step of the improvement of the active element.

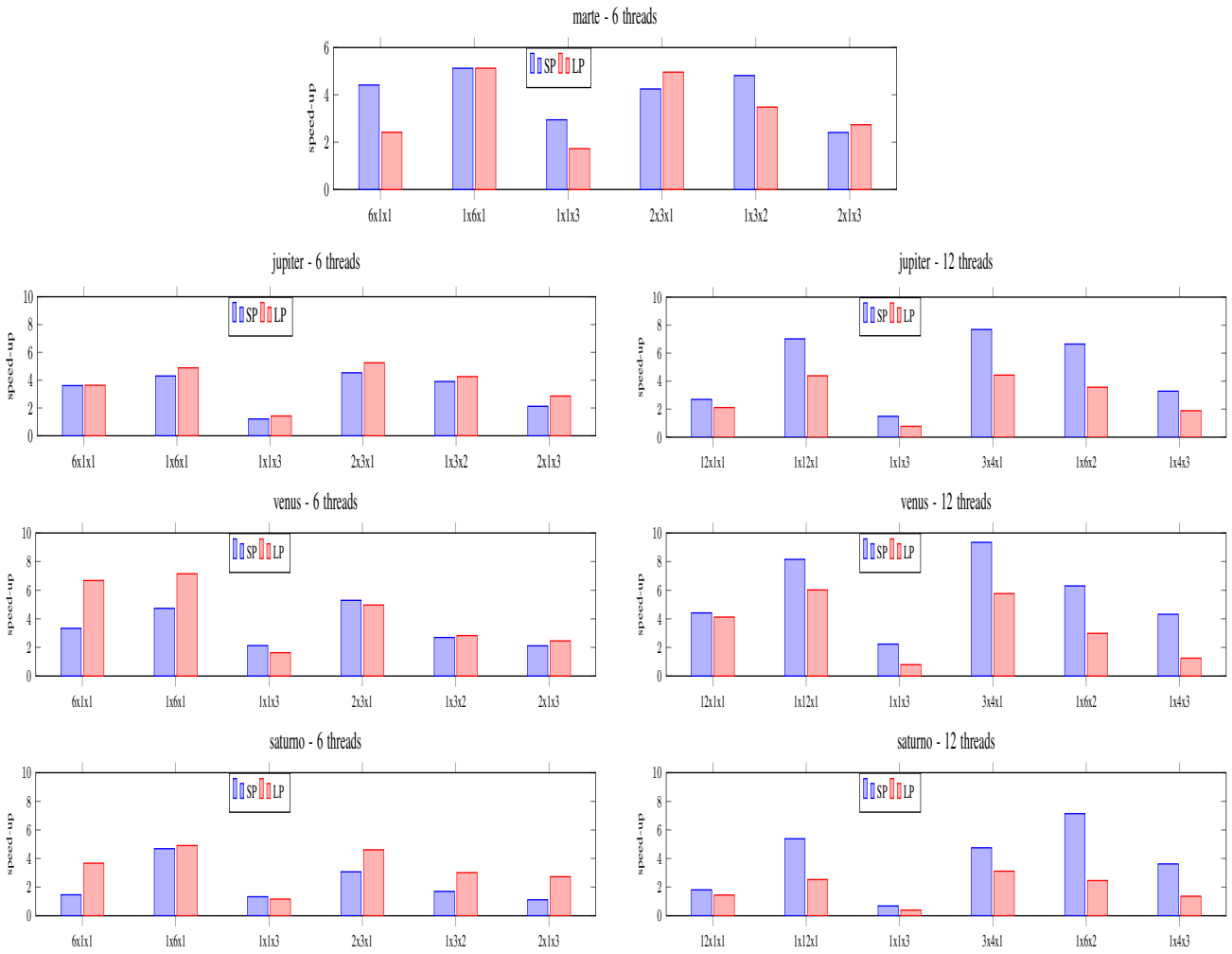

Fig. 2. Speed-up in four computational nodes with exploitation of three-level parallelism and different combinations of threads at each level.

- The parallelization at the lowest level would delegate the computation of the fitness to GPU.

The first version (island model) does not seem to be the best option for the exploitation of GPUs, whose parallelism model is SIMD, and the lowest level version needs large matrices for GPU parallelism to compensate the cost of CPU-GPU transferences. So, the best option seems a combination of the two middle-level versions, maybe with GPUs at the first or second level of the shared-memory version depending on the basic routine and the number of elements to be treated at each level. The points at which the GPUs work are shown in Algorithm 2. The routines are labeled CPU or GPU depending on where they are applied. CPU routines work with OpenMP parallelism. The GPUs work in the computation of the fitness after initialization, combination and diversification, and in the improvement of elements, where the parallelism at two levels might be explored.

The inclusion of GPU parallelism in the different levels is being analyzed. Preliminary results show that the use of GPU can be advantageous for large matrices. For example, in **jupiter**, with siz GPUs, when the computation of the fitness is delegated to one GPU (lowest level parallelism), with the metaheuristic parameters of the experiments in Fig. 1 and with

Algorithm 2 CPU-GPU work partition in the parameterized scheme of metaheuristics

```
InitializeCPU(Sini,ParamIni)
ComputefitnessGPU(Sini,ParamIni)
ImproveGPU(Sini,Sref,ParamImpIni) //two possible levels
while not EndConditionCPU(Sref,ParamEndCon) do
    SelectCPU(Sref,Ssel,ParamSel)
    CombineCPU(Ssel,Scom,ParamCom)
    ComputefitnessGPU(Scom,ParamCom)
    DiversifyCPU(Sref,Scom,Sdiv,ParamDiv)
    ComputefitnessGPU(Sdiv,ParamDiv)
    ImproveGPU(Scom,Sdiv,ParamImp) //two levels
    IncludeCPU(Scom,Sdiv,Sref,ParamInc)
end while
```

the sizes of SP ($t = 200$, $d = 4$, $e = 2$, $t_y = 3$ and $t_z = 2$), the speed-up of GPU parallelism in relation to that of CPU is 0.83 (CPU is faster than GPU), but the speed-up increases for larger problems: for $d = 5$, $e = 3$, $t_y = 3$ and $t_z = 2$ it is 0.92, and for $d = 5$, $e = 3$, $t_y = 4$ and $t_z = 2$ it is 1.25. Higher speed-up is achieved with six GPUs than with just one only for very large problems, so the GPU implementation needs further optimizations.

VII. Conclusions and Future Work

A matrix formulation for Vector Autoregression Models has been stated together with basic and hybrid metaheuristics for the determination of satisfactory models. The three-level parallelism obtained with parallelism in the metaheuristic scheme and in the matrix operations allows us to experiment with different combinations so that the best configuration can be obtained depending on the problem size, the values of the metaheuristic parameters (the metaheuristic being applied) and the computational system.

The experiments show some drawbacks in the implementation, and we are working to improve them:

- Matrix computation techniques are being analyzed to improve the application of linear algebra routines [19]. QR or LQ decompositions can be applied to simplify the model. The Toeplitz-type structure in (5) advises the adaptation of algorithms for structured matrices.
- The reduction in the speed-up with twelve threads may be due to the memory access pattern. Affinity should be considered and message-passing parallelism with an island scheme is being implemented.
- The inclusion of GPU parallelism at the three levels is being implemented.

The determination of the combination of threads at the three levels of parallelism which gives the best performance is a hard problem. So, a systematic approach to obtain satisfactory combinations is needed. We are working on the adaptation to this three-level scheme of auto-tuning techniques used for a two-level scheme [9].

Our work centers on the parallelization of the parameterized schema. We have not yet compared this approach with available software, like STATA [5] and MATLAB [33]. The comparison should be made in terms of goodness of the solution and execution time.

A simulator is being developed [34]. At present it is in a preliminary version which does not include the metaheuristic approach here presented. When the tool is mature enough it will be made freely available.

References

[1] C.A. Sims. Macroeconomics and reality. *Econometrica*, 48(1):1–48, 1980.

[2] J.H. Cochrane. Time series for macroeconomics and finance. Graduate School of Business, University of Chicago, 2005.

[3] R. S. Tsay. *Analysys of financial time series*. John Wiley & Sons, second edition, 2005.

[4] J.J. López-Espín. Development and analysis of algorithm for the best econometric model in health problems (in Spanish). University Miguel Hernández of Elche, 2016.

[5] Stata: Data Analysis and Statistical Software. `https://www.stata.com/`.

[6] F. Almeida, D. Giménez, J.J. López-Espín, and M. Pérez-Pérez. Parameterised schemes of metaheuristics: basic ideas and applications with Genetic algorithms, Scatter Search and GRASP. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 43(3):570–586, 2013.

[7] J.M. Cutillas-Lozano, D. Giménez, and F. Almeida. Hyperheuristics based on parametrized metaheuristic schemes. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015*, pages 361–368, 2015.

[8] F. Almeida, D. Giménez, and J.J. López-Espín. A parameterized shared-memory scheme for parameterized metaheuristics. *The Journal of Supercomputing*, 58(3):292–301, 2011.

[9] J.M. Cutillas-Lozano and D. Giménez. Optimizing a parameterized message-passing metaheuristic scheme on a heterogeneous cluster. *Soft Comput.*, 21(19):5557–5572, 2017.

[10] H. Lütkepohl. Econometric analysis with vector autoregressive models. *Handbook of Computational Econometrics*, pages 281–319, 2009.

[11] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Proc. 2nd Int. Symp. on Information Theory*, pages 267–281, 1973.

[12] G. Schwarz. Estimating the dimension of a model. *Ann. Statist*, 6:461–464, 1978.

[13] E.J. Hannan and B.G. Quinn. The determination of the order of an autoregression. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41:190–195, 1979.

[14] H. Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[15] J.H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):88–105, 1973.

[16] F. Glover and G.A. Kochenberger. Handbook of metaheuristics. *Kluwer Academic*, 2003.

[17] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adpative search procedures. *Kluwer Academic*, 2003.

[18] F. Glover and M. Laguna. Tabu search. *Kluwer Academic*, 1997.

[19] G. Golub and C.F. Van Loan. *Matrix computations*. The John Hopkins University Press, fourth edition, 2013.

[20] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J.J. Dongarra, J. Du Croz, A. Grenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.

[21] M. Resende. Greedy randomized adaptative search procedures (GRASP). Technical Report TR 98.41.1, AT&T Labs-Research, 2000.

[22] F. Glover. A template for scatter search and path relinking. In *Artificial Evolution, Third European Conference, AE'97, Nîmes, France, 22-24 October 1997, Selected Papers*, pages 1–51, 1997.

[23] C. Blum, J. Puchinger, G.R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Appl. Soft Comput.*, 11(6):4135–4151, 2011.

[24] L. Jourdan, M. Basseur, and E.-G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.

[25] G.R. Raidl. A unified view on hybrid metaheuristics. In *Hybrid Metaheuristics, Third International Workshop, LNCS*, volume 4030, pages 1–12, October 2006.

[26] E. Alba. *Parallel metaheuristics: a new class of algorithms*. Wiley-Interscience, New York, 2005.

[27] P. Borovska. Efficiency of parallel metaheuristics for solving combinatorial problems. In *CompSysTech*, page 15, 2007.

[28] A. Kumar and A. Nareyek. Scalable local search on multicore computers. In *Proceedings of the Eighth Metaheuristics International Conference*, pages 146.1–146.10, 2009.

[29] T.G. Crainic, M. Gendreau, and J.Y. Potvin. Parallel tabu search. *Parallel Metaheuristics, E. Alba (ed.)*, 2005.

[30] G. Luque and E. Alba. Parallel genetic algorithms. *Parallel Metaheuristics, E. Alba (ed.)*, 2005.

[31] E.-G. Talbi and G. Hasle. Metaheuristics on GPUs. *J. Parallel Distrib. Comput.*, 73(1):1–3, 2013.

[32] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP*. Morgan Kauffman, 2001.

[33] MathWorks, VAR. `https://www.mathworks.com/help/econ/vector-autoregression-models.html`.

[34] `https://github.com/fylux/tsf_var`.