Adaptive parallel implementation of the Combinatorial GMDH algorithm

O.A. Koshulko[1], A.I. Koshulko[1]

[1]*Glushkov Institute of Cybernetics of NAS of Ukraine*

koshulko@opengmdh.org

Abstract. *The combinatorial algorithm of the Group Method of Data Handling is a compute intensive modelling method well proven for analysis and forecast of a variety of complex systems especially of the so-called "black-box" type. The algorithm is highly dependent on computational power that makes the use of multiprocessing reasonable. To exploit efficiently different kinds of multiprocessor computer systems we propose an adaptive parallel implementation of the combinatorial GMDH algorithm and an example of its usage for the forecasting of the "Top500 Supercomputer's List".*

Keywords

Combinatorial GMDH, parallel processing, Parallel COMBI, Top500 Supercomputer's List, multiprocessor architectures.

1 Introduction

The Group Method of Data Handling (GMDH) [1] is a computational modeling method based on a combinatorial search. For many complex systems represented by multi-parametric time series the combinatorial GMDH search of optimal model structure is a well proven and desirable way to analyze an internal structure of the processes within the system and to make a forecast.

The greater combinatorial search we perform the higher quality of approximation and forecast we are able to obtain for investigated process on the assumption of a proper problem posing. So, the efficiency of GMDH approach depends directly on computational capabilities. Therefore GMDH programs must use parallel processing to exploit capabilities of recent multiprocessor computer systems including clusters and grids.

To work efficiently with most kinds of multiprocessor architectures we use the adaptive parallel combinatorial core implemented for the well known COMBI algorithm [2] that performs a full combinatorial search. Also it is shown below how parallel processing could be used to forecast the "Top500 Supercomputer's List".

2 Combinatorial GMDH algorithm

GMDH is orientated on the forecasting of the so-called "black box" objects and systems with noised observations. It searches in a certain class of functions a model structure with *optimal complexity* which is indicated by the minimal value of an *external criterion*. Here *complexity* is the number of terms in a model. *External criterion* is an expression which describes requirements to the best model, for example

precision, level of complexity, etc. External criterion is always calculated on the part of observations which was excluded from the data set used for estimation of model coefficients.

It is important to note that with noised observations external criterion has a global minimum which shows that complexity of the obtained model is optimal. As an example Fig. 1 shows a typical change of criterion value for a number of models with different complexity is shown.

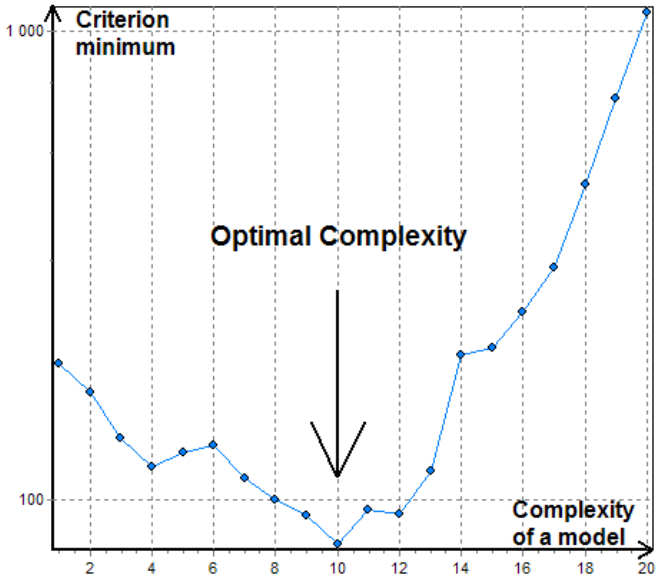

Fig. 1. An example of typical distribution of the criterion minimal values by structures with different complexity. The global minimum of the external criterion shows a model with optimal complexity.

The combinatorial considered here GMDH is a version of the COMBI algorithm published in the collected source codes [2]. The COMBI uses typical for GMDH class of functions (a base function) which is a multi-parametric polynomial that can be given in the form:

$$y = a_0 + \sum_{i=1}^{s} a_i x_1^{j_1} x_2^{j_2} ... x_k^{j_k}, \tag{1}$$

where a_i - coefficients, s – number of all vectors $(j_1, .., j_k)$ where $j \in N_0$, $j \leq P$, P – power of the polynomial (1), k – the number of input parameters of a complex system. The form (1) enables us to alternatively define $p_{\max} \geq j \geq p_{\min}$, where $p_{\min},\ p_{\max} \in Z$ (i.e. power of the base function can be defined as an interval of integers, for example [-1, 3]).

The COMBI algorithm tries to extract from data a model structure with optimal complexity combining different components of the polynomial function (1) and estimating coefficients of each structure with the least-squares method. The number of different structures that appear during the combinatorial search is

$$q_n = \sum_{i=0}^{n} C_n^i = 2^n, \tag{2}$$

where n – the number of components in the polynomial (1),

$$n = \frac{(P+k)!}{P!\,k!}\ . \tag{3}$$

The number of different structures q_n originally depends on P and k, therefore lets mark the corresponding class of problems as Q_k^P, and the greater class as Q^P (equal to Q_k) to mark the infinite union of classes $Q_1^P, Q_2^P, Q_3^P, ...$

3 Compute intensity of the combinatorial algorithm

One of the main problems in applying a full combinatorial search is a compute intensity. For example if maximum of model complexity n in a certain simulation equals 40 than approximately 10^{12} structures have to be considered. If a teraflop computer is used the problem takes approximately 12 hours to solve with our GMDH implementation. The increase of n by one doubles the number of structures and therefore computational time also doubles. In practice it often turns out that the computational power necessary to solve the formulated problem is not available and the problem must be simplified manually to reduce computational time.

One of the ways of approximate estimation of compute intensity of a problem is to calculate the number of all possible structures, see Tab. 1. Practically compute intensity of classes of problems Q_1 and Q^1 is quite suitable for single-processor computers if n is about 25 or less. However from Tab. 1 we can see that if $P > 1$ and $k > 1$ the number of problems solvable with a single processor is only 8. The number of problems solvable in parallel is also 8. It is significant that parallel processing doubles the number of compute intensive classes of problems solvable by the combinatorial algorithm.

Tab. 1. The number of structures q_n in different classes of problems Q_k^P where $k, P \leq 8$. Q_k^P rounded values are placed on the crossing of rows Q_k and columns Q_P . Problem classes solvable with parallel processing are in bold font. Too complex problems are marked '–'.

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_7
Q^1	4	8	16	32	64	128	256	512
Q^2	8	64	1024	32769	$2 \cdot 10^6$	$\mathbf{3 \cdot 10^8}$	$\mathbf{7 \cdot 10^{10}}$	$\mathbf{4 \cdot 10^{13}}$
Q^3	16	1024	10^6	$\mathbf{3 \cdot 10^{10}}$	–	–	–	–
Q^4	32	32769	$\mathbf{3 \cdot 10^{10}}$	–	–	–	–	–
Q^5	64	$2 \cdot 10^6$	–	–	–	–	–	–
Q^6	128	$\mathbf{3 \cdot 10^8}$	–	–	–	–	–	–
Q^7	256	$\mathbf{7 \cdot 10^{10}}$	–	–	–	–	–	–
Q^8	512	$\mathbf{4 \cdot 10^{13}}$	–	–	–	–	–	–

4 Adaptive parallel processing

Obviously combinatorial search could work in parallel very efficiently and is able to use not only symmetric multiprocessors but also computer systems with low-speed inter-processor communication or distributed systems.

To determine architectures of interest for GMDH computing we classify them by the following properties.

1. Software environment type:

– Message passing library (for example MPI - Message Passing Interface, PVM - Parallel Virtual Machine, etc). Such systems are Massively Parallel Processors (MPP) including compute clusters;

– Automatic process migration (based on SSI - Single System Image, MOSIX, etc). These are MOSIX clusters, SMP and NUMA computers with Single System Image and to some extent distributed computing environments like BONIC.

2. Hardware environment type:

– Uniform (all CPU have the same performance);

– Nonuniform (containing different CPUs).

The simplest case is the uniform system and automatic process migration. It requires to divide the full task by the number of processors in the computer system and distribute obtained parts of the task to all processors of the system using available programming interface.

Generation of every new structure in COMBI is the result of binary register shift so each structure has a binary sequential code that could be converted into a decimal and back. We just have to inform the parallel process about its unique shift range i.e. a 'start' and 'stop' position.

So if N_{CPU} processors are available then we divide the full task into the same number of groups $G = N_{CPU}$. Each of the groups will have N_G models

$$N_G = \left[\frac{q_n}{G}\right] \tag{4}$$

where [ ] – integer part of the number.

So, the process numbered i have the 'Start' and the 'End' (or stop) positions of shift register marked as S_i and E_i:

$$S_i = \left[\frac{q_n}{G} \cdot (i-1)\right], \quad E_i = \left[\frac{q_n}{G} \cdot i\right]. \tag{5}$$

One of the obstacles here is that the processors of a computer system could have different performances. However if we have a list of performances of CPUs or just information about the system architecture class we can automatically adapt the mechanism of task dividing to fit the performance of every processor.

If we have a vector of CPU performance $R = (r_1, ..., r_{N_{CPU}})$ and $\exists\ i,\ j\ :\ r_i \neq r_j$ then

$$G = \sum_{i=1}^{N_{CPU}} r_i \Big/ LCD(r_1, ..., r_{N_{CPU}}), \tag{6}$$

where LCD – largest common divisor. Now we can formulate special expressions for different multiprocessor architectures.

I. For SMP, NUMA, uniform MOSIX-clusters:

$$S_i = \left[\frac{(q_n}{N_{CPU}} \cdot (i-1)\right], \quad E_i = \left[\frac{q_n}{N_{CPU}} \cdot i\right]. \tag{7}$$

II. The same formulas (7) have to be used in case of any MPP-system including uniform MPI-clusters but the software must use a message passing library.

III. Nonuniform MOSIX-clusters:

$$S_i = \left[\frac{q_n \cdot (i-1)}{\sum\limits_{j=1}^{N_{CPU}} r_j \Big/ LCD(r_1, ..., r_{N_{CPU}})}\right], \quad E_i = \left[\frac{(q_n \cdot i}{\sum\limits_{j=1}^{N_{CPU}} r_j \Big/ LCD(r_1, ..., r_{N_{CPU}})}\right]. \tag{8}$$

IV. Distributed networks and grids:

$$S_i = \left[\frac{q_n}{G} \cdot \sum_{j=1}^{i-1} r_j \Big/ LCD(r_1, ..., r_{N_{CPU}})\right], \quad E_i = \left[\frac{q_n}{G} \cdot \sum_{j=1}^{i} r_j \Big/ LCD(r_1, ..., r_{N_{CPU}})\right]. \tag{9}$$

The alternative way of parallel processing that could be used with some types of computer architectures, for example Massively Parallel Processors, is to dedicate one CPU to manage the process of

numerously repeated distribution of small tasks on demand. It is the only way if an algorithm could not be divided once into the number of independent parts which does not concern combinatorial GMDH. The consequence of such an approach would be a low efficiency on the systems with a small number of processors, for example, if one of only four processors is reserved for managing then we have only 75% of resources used in processing. But most important is that such a way is not suitable for distributed networks and grids that are very cost-effective and widely used.

So if we know the architecture class we can divide the full task for only one time. However another important obstacle here is that the time necessary for consideration of a model structure highly depends on model complexity and can be significantly different. Therefore tasks with equal number of structures are not equal in terms of computational time. An example of influence of the second obstacle is shown in Fig. 2.

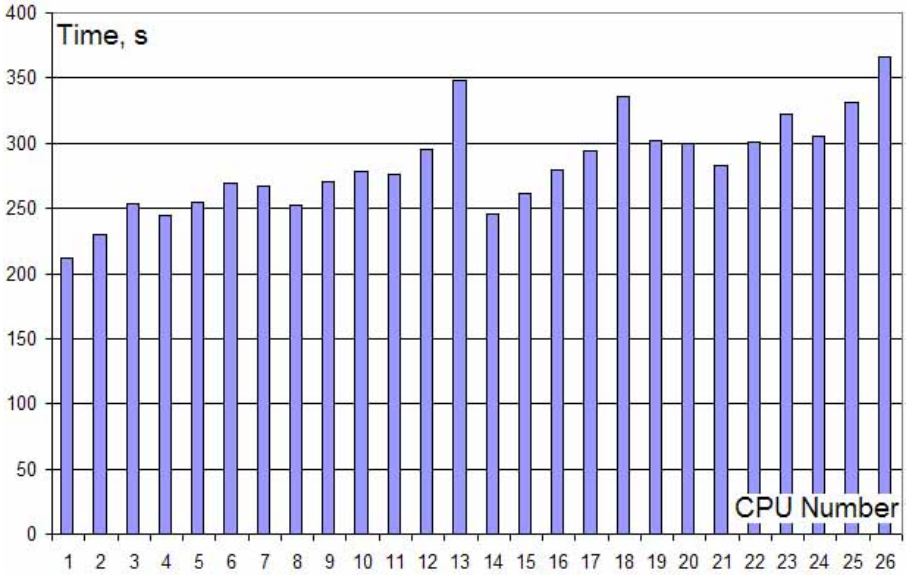

Fig. 2. Distribution of computational times for 26 similar CPUs in case of dividing the Q_6^2 problem into equal parts.

To obtain the final structure we have to compare the results of all CPUs. Therefore the longest computational time will be the time of problem solving. Instead of 26 time speed-up in Fig. 2 we see only 71% of it. Simulation of the use of the system with 100 CPUs shows that we will have not more than 32% of full speed-up that is a very low efficiency.

The solution to this problem is to consider couples of structures with opposite binary register position. Each iteration of the combinatorial GMDH search is defined by the unique state of binary vector where units and their positions define the set of parameters to consider during the iteration. Coupling the first half of vector states with their bitwise negated representations (the couples like 000101 and 111010) covers all possible vector states. Fig. 3 shows that under the same conditions than as in Fig. 2 such an approach gives 96% of maximal speed-up. Simulation of partitioning into a thousand tasks shows that the efficiency of proposed approach remains on the same level.

5 Parallel COMBI application

The above ideas of parallel processing have been implemented in the algorithm core of software called Parallel COMBI (www.opengmdh.net). One of the objects that can be successfully predicted with the use of the Parallel COMBI is the Top500 List of Supercomputers.

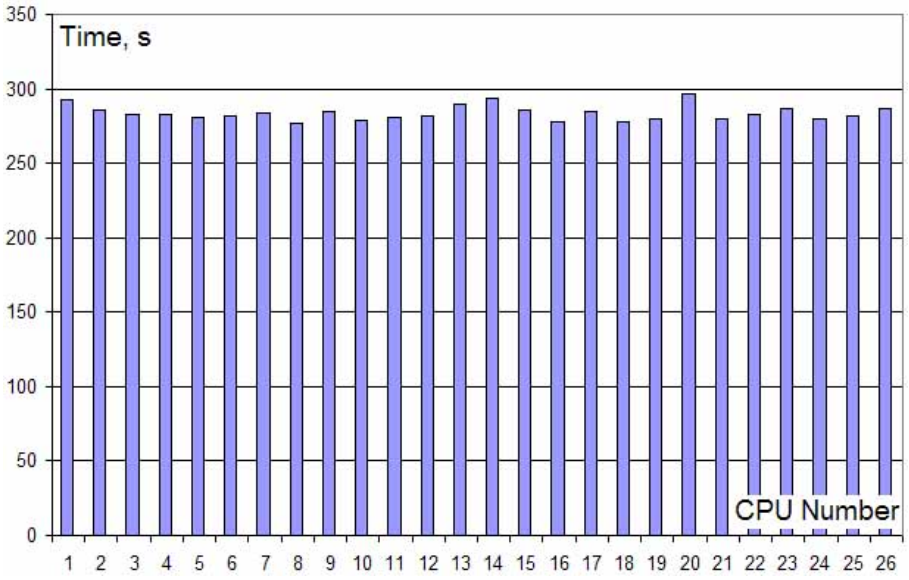

Fig. 3. Distribution of computational times for 26 similar CPUs with use of opposite register state.

The Top500 project represents a relatively long history of supercomputer industry observations releasing twice a year a list of the world's most powerful supercomputers (www.top500.org). A general trend on all Top500 supercomputer lists is an exponential growth of performance of most of Top500 ranks. On logarithmic scales this process can be represented by a line which after extrapolation will show us the approximate performance of future supercomputers. Traditionally this kind of forecast is available directly from the Top500 website representing the entry-level performance, the largest system, and the total performance sum of the List (www.top500.org/lists/2007/06/performance_development).

Such an analysis of the first and the last positions of the Top500 shows that actual Top500 values fluctuate around the trend line. Taking into account that we are considering the process on logarithmic scales we can't expect the forecast to be very precise with this approach. However for the really long-time forecast the use of a simple model such as an exponential trend is quite a reliable way.

Detection of other dependences that can be hidden in the Top500 statistics requires consideration of more complex models which structures are originally undefined. The problem of choosing the correct model structure is even more significant when considering the process as a multi-parametric function. With Parallel COMBI we can choose important parameters of the process among the large number of characteristics available from the Top500 List.

The use of polynomial GMDH models improves short term forecasting of entry-level performance of the Supercomputers' Top500 List in comparison with exponential trend

$$Y(x) = a \cdot \exp(bx). \tag{10}$$

Fig. 4 shows that a half-year forecast of the R_{max} – performance measured by HPL benchmark [3] for the entry-level system of the Top500 List of Supercomputers can be made more precise than with an exponential trend.

Polynomial GMDH forecasts in Fig. 4 are calculated without changing internal algorithm parameters except the input data length. The input of the multi-parametric simulation was time series of: R_{max} sum of all Top500 systems, R_{peak} sum of the Top500, the number of all Top500 processors, the number of systems excluded from the Top500, R_{max} of system #400, $R_{max}$of system #300.

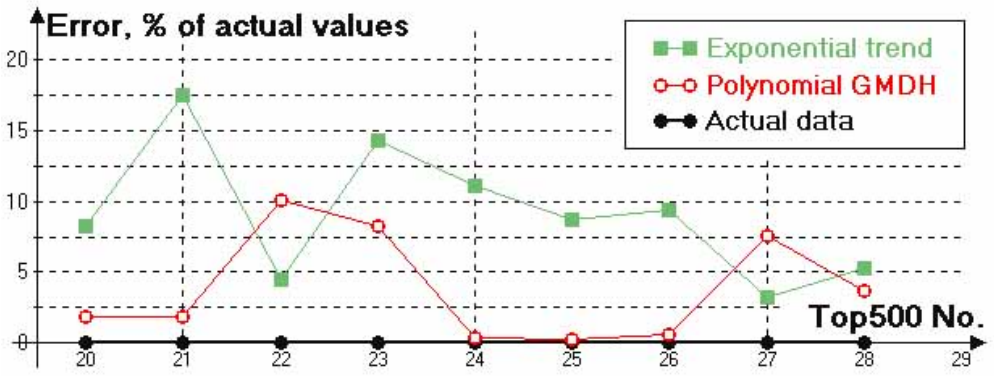

Fig. 4. Half-year forecasts by the Parallel COMBI in comparison with exponential trends.

6 Conclusion

It is shown that adaptive parallel implementation of GMDH algorithm is capable of the efficient exploiting different multiprocessor architecture classes. The mechanism of opposite register state is proposed to improve efficiency of parallel processing. Application of Parallel COMBI is demonstrated by the example of forecasting the "Top500 Supercomputer's List" that gives better results than exponential trends widely used for this purpose.

References

[1] Madala H.R., Ivakhnenko A.G.: Inductive Learning Algorithms for Complex Systems Modeling. – CRC Press, 1994. – 368 p.

[2] Ivakhnenko A.G., Koppa Y.V., Stepashko V.S.: *S*pravochnik po tipovim programmam modeluvannya. Kiev: Technika, 1980. – 184 p.

[3] Dongarra J., Luszczek P., Petitet A.: The LINPACK Benchmark: Past, Present, and Future *C*oncurrency and Computation: Practice and Experience – 2003. – Volume 15. – P. 1-18.