One Approach for Parallel Algorithms Representation

A. Bosakova-Ardenska

Abstract— **This paper presents one approach for parallel algorithms representation. The proposed model is practice oriented and its name is AMPA (Agenda Model for Parallel Algorithms) due to basic blocks organization like a schedule. The model uses classical Master/Slave paradigm. One parallel merge sorting algorithm based on quick sort is presented with the discussed AMPA model and also three known representation approaches (description with natural language, pseudo code and PRAM). A survey of professional opinion about AMPA and other approaches is conducted. The results show that most of the interviewed people choose AMPA as the best way to understand the algorithm.**

Index Terms— **Master-slave, Merge sort, Parallel algorithms, PRAM, Programming model, Pseudo code, Quicksort.**

I. INTRODUCTION

DURING THE LAST YEARS the parallel programming becomes one of the most popular techniques in application development. Development of processors architectures (SoC and Multi-core architectures) leads to significant advancement in software technologies. The possibilities lot of us to have multi processors on a small chip leads to the development of parallel applications which could effectively use these hardware resources. The scientific evolution also needs of computational resources and effective parallel programs. The complexity of software also increases and this is the reason that new usage models for program design are wanted. Some new parallel programming models for specific multi-thread architectures were designed to last year's [1,2,3]. They are useful for designing parallel algorithms for specific architectures like NVidia GPU.

The main idea behind this research is to be proposed a practice oriented high-level model for parallel algorithms representation. The proposed model uses well known Master/Slave paradigm.

A.BOSAKOVA-ARDENSKA is with Department of Computer Systems and Technologies University of Food Technologies, Plovdiv, Bulgaria (e-mail: a_bosakova@uft-plovdiv.bg)

Manuscript received June 5, 2016; accepted September 17, 2016.
DOI: 10.17694/bajece.292651

II. AGENDA MODEL FOR PARALLEL ALGORITHMS (AMPA)

AMPA is a simple to practice oriented model for parallel algorithm representation. The name is Agenda Model for Parallel Algorithms due to its structure. According to this model, there are 6 basic elements and traditional Master/Slave code organization logic. The Master/Slave code organization logic is a variant of SPMD (Single Program Multiple Data) models which are successfully applied in parallel algorithms developing [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. The operations are located in their exact positions depends on parallel execution. This organization is like a schedule and that is the reason for the name Agenda in AMPA.

The AMPA defines two types of processes- Master and Slave. Master is always one but Slaves are many. The model consists of six graphical elements:

1) Process block (Master or Slave). If the algorithm contains only Master process this is not a parallel algorithm;

2) Operation block – this block contains some operations: calculations or data exchanging;

3) Vertical arrow – this is a line which presents execution's flow in one process;

4) Horizontal arrow – this is a line which presents communications among processes;

5) Execution type block – this is block which groups other blocks to point sequential or parallel execution part;

6) Parallel steps block – this block groups other blocks whose parallel execution has to be repeated and it shows how many times the execution will be repeated.

The blocks of Master and Slave processes are situated in parallel lines. If two blocks of Master and Slave processes are at the same level, this means that these operations could be executed simultaneously. I.e. the position of every block shows when the block could be executed. Figure 1 shows an example of the parallel algorithm presented with AMPA.

Execution type blocks and Parallel steps block are drawn with dashed line. The AMPA model could be applied for multi-thread application. In this case:

- the Master process is "Process" but "Slave" processes are implemented as threads;

- horizontal arrows will be replaced with "read/write global data" (i.e. threads will work with data of its own process).

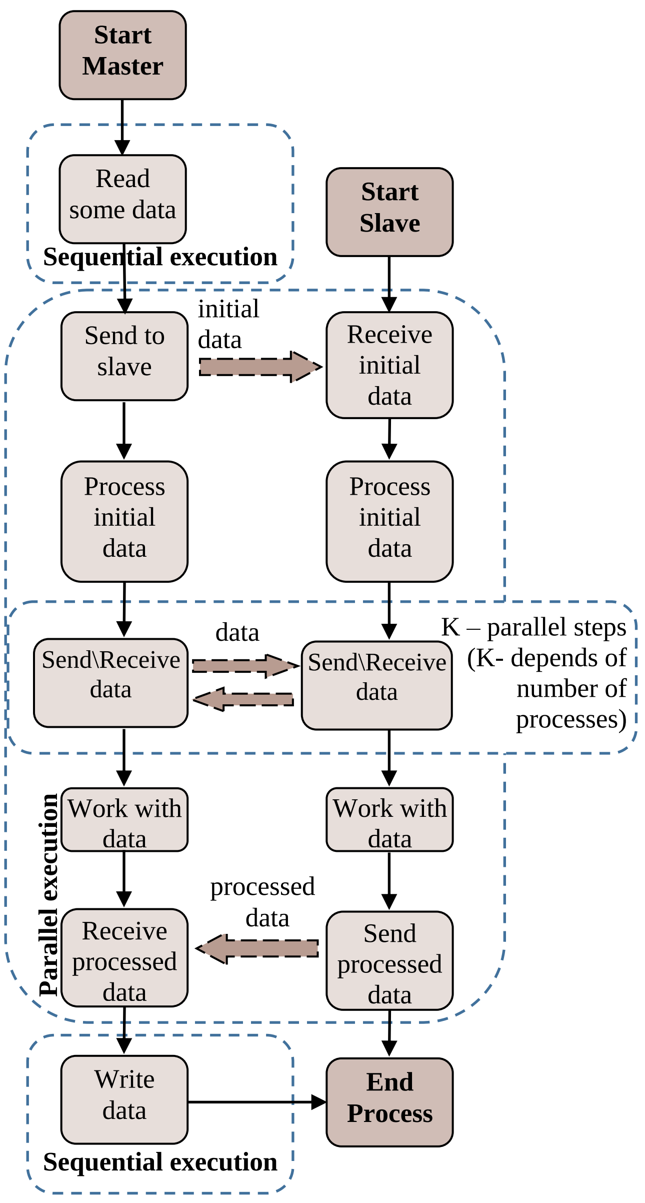

Fig.1. Sample algorithm presented with AMPA

III. APPLICATION OF AMPA

Parallel merge sort uses a "divide and conquers" approach and data distribution maps into a binary tree [6]. Data are divided into sub-lists and the process continues while lists reach size one. The proposed model is used for the representation of one modification of parallel merge sort algorithm. This modification uses quicksort algorithm [17] to sort sub-lists. The number of sub-lists is equal to the number of parallel processes (processors). The sub-lists are the same size. After their sorting with quicksort sub-lists are merged. The next figures (fig. 2, fig. 3, fig. 4 and fig. 5) present this algorithm using respectively a description of the natural language, pseudo code, PRAM model [18] and proposed AMPA model.

The numbers that need to be sorted are distributed equally to the parallel processes (processors). Each process sorts its part of the numbers using the quicksort algorithm. Finally, the sorted parts are merged.

Fig.2. Parallel merge sort – described by natural language

```
for i=1 to M-1 do in parallel
    myarr <= P0 (arr[i*n/m])
qsort(myarr)
    merge (myarr => P0 (arr[i*n/m]))
end parallel
```

Fig.3. Parallel merge sort – described by pseudo code

```
begin
    global read(arr[i*n/m], myarr);
    qsort(myarr);
    merge (global write(myarr, arr[i*n/m]))
end
```

Fig.4. Parallel merge sort – described by PRAM

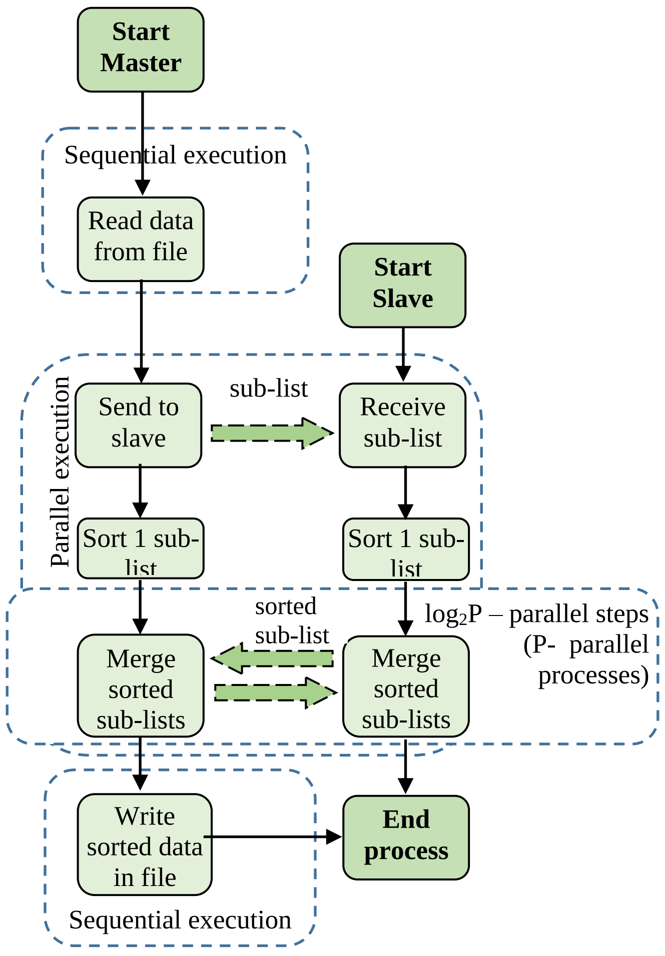

Fig.5. Parallel merge sort – AMPA described

The variables and operations which are used in fig. 3 and fig. 4 are:

M – number of parallel processes (processors);

N – size of the array for sorting (count of all numbers);

arr – array for sorting;

myarr – local array for sub-list;

global read() – operation for global memory reading;

global write() – operation for global memory writing;

=> and <= - operations for data reading/writing.

The main assumption for pseudo code description is that unsorted array belongs to process (processor) P0. The main assumption for PRAM description is that unsorted array is allocated into global memory.

The number of merge operations is equal to $\log_2 P$, where P is a number of parallel processes, i.e. the number of sub-lists. This means that after first merge operation the number of processors which execute merge operation will decrease twice. For example:

P = 8, number of parallel merge operations = 3

1 parallel merge operation: 4 processes will receive sorted sub-lists of other 4 processes and will execute merge operation;

2 parallel merge operation: 2 processes will receive sorted sub-lists of other 2 processes and will execute merge operation;

3 parallel merge operation: 1 process will receive sorted sub-lists of other process and will execute merge operation. After this step, a final sorted list will be reached.

IV. RESULTS

Discussed parallel sorting algorithm and its four representations are used for the short survey of opinion among:

- students which study course Supercomputers, part of Computer Systems and Technologies speciality at University of Food Technologies, Plovdiv (Bulgaria);

Centre for Supercomputing Applications) in assistance with

- participants of training school “Practical Programming Models and Skills on INTEL Xeon Phi for Scientific Research Engineers”. This course was organized by NCSA (National Science and Technology Facilities Council (STFC) and Bayncore (U.K.).

More than fourteen people were included in the survey. The questions listed in current survey are:

1) Which of the four representations of the parallel algorithm helps you best to understand its idea?

(a) Description with natural language

(b) pseudo code

(c) PRAM

(d) AMPA

2) Which of the models for presentation of the parallel algorithm would you use if you need to implement it? Why?

The figures six and seven present results of the survey. Some of the answers to question “Why?” of question 2 (Which of the models for presentation of the parallel algorithm would you use if you need to implement it? Why?) are presented in table 1.

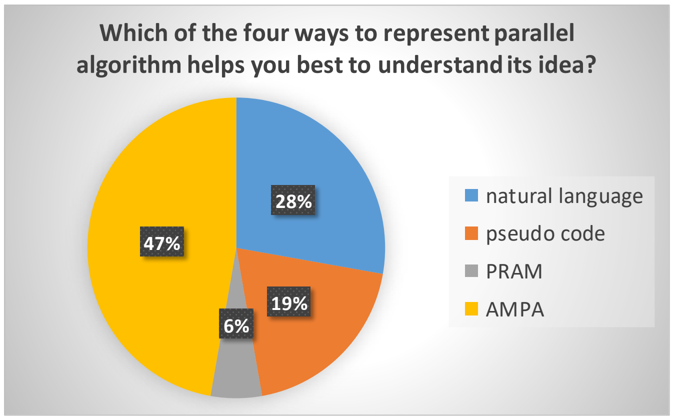

Fig.6 Results for question 1 of conducted survey

Preferred model for implementation	Reasons
natural language	This representation tells me just what needs to be done.
pseudo code	This representation is most understandable for me.
	This representation is “universal” code and could be used as a basic for a parallel program.
	This representation is shortest and clearly described.
PRAM	The source code in this representation could be used for the skeleton of a program.
	This representation is short.
AMPA	This representation is the best for idea understanding.
	The detailed description of the parallel algorithm is suitable for its precise implementation.
	This model gives a good visual idea and thus it will decrease the count of the logical errors in implementation.

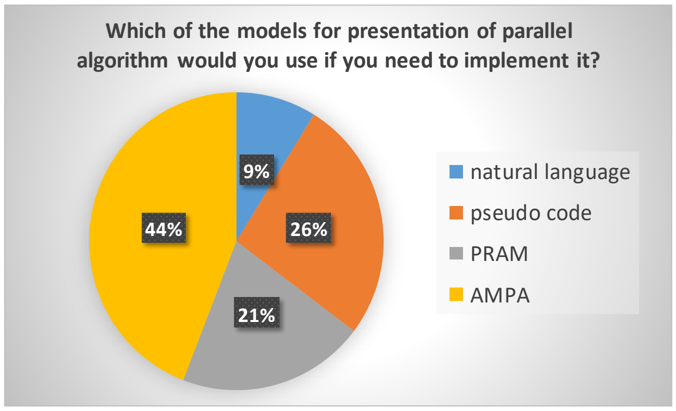

Fig.7. Results for question 2 of conducted survey

V. CONCLUSIONS AND FUTURE WORK

A novel approach for parallel algorithms representation with graphical elements is presented in this paper. One parallel merge sort algorithm is described using natural language, pseudo code, PRAM and AMPA. These four presentations were evaluated by students and participants of professional course for parallel programming. The results show that:

- Preferred model is AMPA because it gives is good visual idea about algorithm (47% of interviewed people choose AMPA as the best way to understand the algorithm);

- When the algorithm has to be implemented the AMPA and pseudo code models are most preferred (44%- AMPA and 26%- pseudo code).

In the future, the research will continue with developing a software tool for AMPA modelling. This tool will facilitate the use of the model.

ACKNOWLEDGMENT

The author would like to thanks to NCSA and Prof. Stoyan Markov for their assistance in the survey of opinions about the proposed model.

REFERENCES

[1] Kirtzic J. S., O. Daescu, A parallel algorithm development model for the GPU architecture, Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2012
[2] Valiant L., A bridging model for multi-core computing, Journal of Computer and System Sciences, vol. 77, no. 1, pp. 154–166, 2011.
[3] Luebke D., CUDA: Scalable parallel programming for high-performance scientific computing, 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Paris, pp. 836-838, 2008
[4] Yukiya Aoyama, Jun Nakano, "RS/6000 SP: Practical MPI Programming", International Technical Support Organization, IBM, 1999
[5] Seyed H. Roosta, Parallel Processing and Parallel Algorithms: theory and computation, Springer, ISBN 0-387-98716-9, 2000
[6] Wilkinson B. and Allen M., Sorting Algorithms, Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice-Hall, 1999
[7] Sahni S. and G. Vairaktarakis, The master-slave paradigm in parallel computer and industrial settings, Journal of Global Optimization, 9, pp. 357–377, 1996
[8] Baldo L., L. Brenner, L. G. Fernandes, P. Fernandes, A. Sales, Performance Models For Master/Slave Parallel Programs, Electronic Notes in Theoretical Computer Science, 2004
[9] Mostaghim S., J. Branke, A. Lewis, H. Schmeck, Parallel Multi-objective Optimization using Master-Slave Model on Heterogeneous Resources, Proceedings of the IEEE Congress on Evolutionary Computation, 2008
[10] Cazenave T., Nicolas Jouandeau, A Parallel Monte-Carlo Tree Search Algorithm, Computers and Games, 2008
[11] Shuping LIU, Yanliu CHENG, The Design and Implementation of MPI Master-Slave Parallel Genetic Algorithm, International Conference on Education Technology and Computer (ICETC2012), 2012
[12] Depolli M., R. Trobec, B. Filipi`c, Asynchronous Master-Slave Parallelization of Differential Evolution for Multiobjective Optimization, Evolutionary Computation 21 (2), pp. 261–291, 2013
[13] Krichene H., M. Baklouti, Jean-Luc Dekeyser, Ph. Marquet, M. Abid, Master-Slave Control structure for massively parallel System on Chip, DSD SEAA - 16th Euromicro Conference on Digital System Design, Sep 2013, Santander, Spain. 2013
[14] Scrucca L., On some extensions to GA package: Hybrid optimisation, parallelisation and islands evolution, The R Journal 9(1), pp.187-206, 2017
[15] Jiaxing Qu, Guoyin Zhang, Zhou Fang, Jiahui Liu, A Parallel Algorithm of String Matching Based on Message Passing Interface for Multicore Processors, International Journal of Hybrid Information Technology, Vol.9, No.3, pp. 31-38, 2016
[16] Jiahui Liu, Dahua Song, Yiqiu Xu, A Parallel Encryption Algorithm for Dual-core Processor Based on Chaotic Map, Proceedings of SPIE - The International Society for Optical Engineering SPIE Proceedings, 2012
[17] Hoare C.A.R., Quicksort, The Computer Journal, vol. 5, pp 10-16, 1962
[18] JaJa Joseph, An Introduction to Parallel Algorithms, Addison-Wesley publishing company, 1992

BIOGRAPHIES

ATANASKA D. BOSAKOVA-ARDENSKA was born in 1980. She received the M.Sc. degree of Computer Systems and Technologies at Technical University of Sofia, Plovdiv branch 2004. She receives Ph.D. in 2009 with thesis "Parallel information processing in image processing systems". From 2010 she is assistant in the department of Computer Systems and Technologies in University of Food Technologies. From 2014 she is associated professor by "Synthesis and Analysis of Algorithms" in Department of Computer Systems and Technologies in the University of Food Technologies in Plovdiv, Bulgaria. She is a member of USB (The Union of Scientist in Bulgaria) and head of Club of Young Scientists in Plovdiv (USB – Plovdiv in Bulgaria). Her research interests include: parallel algorithms, sorting algorithms, image processing, MPI (Message Passing Interface), C/C++ programming.