

Parallel computing of GAME models

Pavel Kordik, Jakub Spirk, Ivan Simecek

Dept. of Computer Science and Engineering, Karlovo nam. 13, 121 35 Praha 2, Czech Republic

kordikp@fel.cvut.cz

Abstract. *With recent development of multi-core and multi-processor computers, single thread algorithms use just fraction of possible computing resources available on single personal computer. The trend is to develop distributed versions of algorithms so they can run on several cores in parallel efficiently using all resources available. In this paper we present an efficient distributed version of the GAME algorithm for inductive models evolution. We also discuss the possibilities and assets of parallelizing evolution of inductive models. Our experimental results demonstrate that for two core processors the distributed GAME algorithm achieves 1.7 speedup in average against the serial version. For eight cores, the speedup is 3.5 in average.*

Keywords

Inductive modelling, distributed computing, parallel processing, FAKE GAME.

1 Introduction

Nowadays, standard personal computer has multi-core processor(s). When you run single thread application written e.g. in Java language, it is executed on single core, thus using just a fraction of resources available, depending on how many cores the computer has.

Most common multiprocessor systems today use an SMP architecture [1]. Symmetric multiprocessing or SMP involves a multiprocessor computer-architecture where two or more identical processors can connect to a single shared main memory. Most common multiprocessor systems today use the SMP architecture. In case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors.

SMP systems allow any processor to work on any task no matter where the data for that task are located in memory; with proper operating system support, SMP systems can easily move tasks between processors to balance the workload efficiently.

The limitation is that single-thread applications can be run on one core while other cores are idle. The GAME algorithm described in [2] was implemented as single-thread Java application.

The goal of this paper is to explore possible speedup of the GAME algorithm by distributing the load over several cores. Methods for distributed computing are discussed below, later experiments are performed to measure the speedup achieved.

2 Methods for distributed computing of inductive models

The most important task, when parallelizing algorithms, is to identify sections of the algorithm, that can run in parallel and are most time demanding at the same time.

In the case of the GAME algorithm, the most time consuming section is the learning of units (neurons). The learning repeats several times for each unit and it can be run in parallel (at least for units in single layer). The GMDH MIA and COMBI algorithms were successfully migrated to multiprocessor systems [3]. The GAME algorithm proceeds from GMDH MIA, but units are evolved by niching genetic algorithm [5].

Genetic algorithms are suitable for parallelization and there exists many techniques for this problem [7]. We decided to take advantage from the fact that the GAME algorithm is implemented in Java. This programming language supports distributed parallel computing by the Thread technology [6]. Furthermore, it makes it fully transparent for programmer. That can be disadvantage when one needs to have all processes and their placement under control, but it is not our case.

Units can learn independently on different cores and they do not need to communicate during this process. The only thing we need to resolve is the synchronization of the learning process (all units should finish learning before the selection) and how many threads should be initialized.

Our experiments showed that the time needed to initialize new tread (0.1 ms) is insignificant with respect to learning and therefore we decided that each unit will learn in a separate thread. Also the difference in time needed for thread synchronization using class Semaphore compared to method join () of the java. lang. Thread class was statistically insignificant.

3 Methodology of experiments

We performed our experiments on the following hardware: Intel Core 2 Duo E6550 on 2,33GHz and 2x Intel Quad Core XEON E5430 on 2,66GHz.

The performance of the parallel implementation can be measured as the speedup related to the sequential implementation

$$S = \frac{T_{seq}}{T_{par}} \quad (1)$$

where T_{seq} is the average duration of the sequential version of the GAME algorithm and T_{par} is the duration of its parallel version.

Speedup for one core can be defined as an efficiency:

$$E = \frac{S}{P} \quad (2)$$

where P is the number of cores.

We run 4 experiments (A,B,C,D) with different configuration of the GAME algorithm and different data set. Each experiment was repeated 20 times.

Tab. 1. Settings of experiments

Experiment	A	B	C	D
Data set	ecoli-raw.txt	antro-age.txt	buildingraw.txt	motol-brain-pressure.txt
GAME configuration	user defined	standard	quick	standard
Cross validation	5 folds	10 folds	10 folds	10 folds
Models in ensemble	5	5	single	7

Description of data sets can be found in [2]. A configuration of experiments varies significantly from fast, delivering models in seconds, to extremely slow, where models are constructed for several hours.

4 Experimental results

Figure 1 shows boxplots for each configuration executed for serial and parallel version of the GAME algorithm. Its apparent, that the parallel processing is faster for all experiments. The speedup computed according to the Equation 1 ranges from 1.65 to 1.78 and the efficiency of cores is from 0.83 up to 0.89 according to the Amdahl's law [4]:

$$sp(N, P) = \frac{1}{(1 - P) + P / N} \quad (3)$$

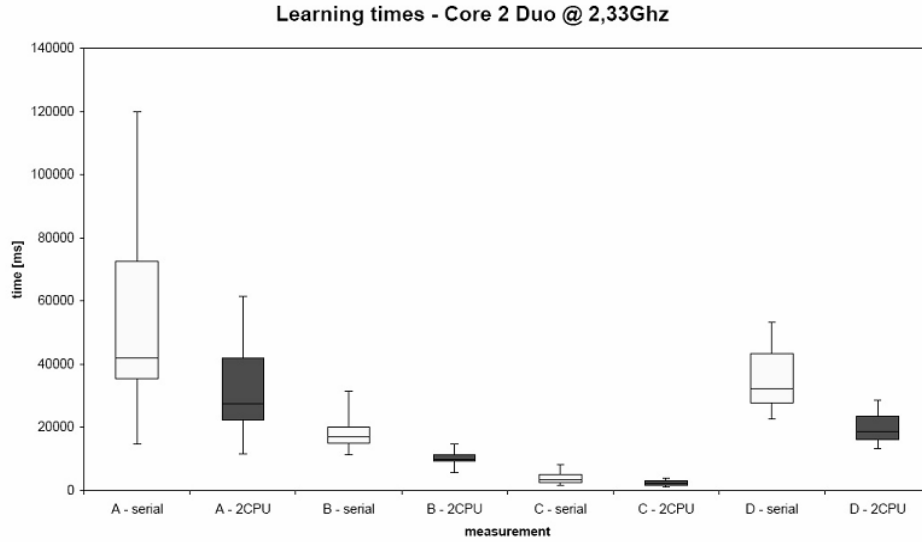


Fig. 1. Experiments of sequential and parallel GAME computed on Core 2 Duo

which corresponds to measured speedup of parallelized part of the algorithm equal to 83.58%, 85.1%, 78.84% and 87.52% respectively.

The Figure 2 is analogous with the previous Figure, only number of cores increased from 2 to 8. For eight cores, the speedup ranged from 3.36 to 3.62. The efficiency compared to two cores computer decreases significantly (around 0.45). Amdahl's law predicts consistent sizes of parallelized part of the code 82,27%, 82, 23%, 80,29% and 82,73% - very similar to values obtained for 2 cores processor.

We expected that for shorter computations, speedup will be significantly smaller, but experimental results show just small decrease for same cases that cannot be generalized.

4.1 Discussion

From results of experiments performed we can use Amdahl's law [4] to predict speedup for the number of processors used. For our implementation of parallel GAME algorithm the speedup will be in average

$$GAME_{sp}(N) = \frac{1}{0,1718 + \frac{0,8282}{N}} = \frac{N}{0,1718N + 0,8282} \quad (4)$$

where N is the number of processors.

Figures 1 and 2 also show that for parallel mode of computation, number of extremely slow computation values decreases. This fact however can be caused by reducing influence of other applications running in parallel with the serial version GAME possibly increasing time of computation.

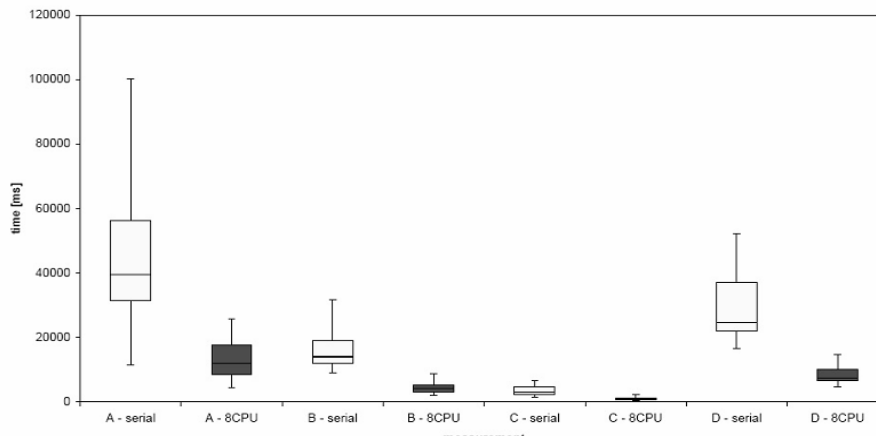
Consistency of measured data can be also verified by variation coefficient, which is defined as

$$V = \frac{\sigma}{\bar{x}} \quad (5)$$

where σ is standard deviation and \bar{x} is average. In Table 3 there are maximal, average values for each experiment, their ratio, standard deviation and the variation coefficient.

From the Table 3 we can draw the conclusion that for more cores, variation coefficients decrease. Computation is less influenced by other processes and we can better estimate time needed to construct inductive model and the real speedup

Learning times - 2x Quad Core XEON @ 2,66GHz



		σ	max	avg	max/avg	sd/avg		σ	max	avg	max/avg	sd/avg
A	serial	30338,37	120115	54950,1	2,185892	0,552108	serial	26547,31	100218	47068,8	2,129181	0,564011
	C2D	14768,35	61680	31985,55	1,928371	0,461719	QCX	6213,12	25642	13183,95	1,944941	0,471264
B	serial	5108,651	31564	17977,2	1,75578	0,284174	serial	5562,112	31645	15677,65	2,018479	0,35478
	C2D	2137,231	14625	10328,25	1,416019	0,206931	QCX	1292,2	8779	4396,95	1,996611	0,293886
C	serial	1644,827	8238	3846,75	2,141548	0,427589	serial	1462,519	6555	3437	1,907186	0,425522
	C2D	809,655	3891	2330,35	1,669706	0,347439	QCX	403,18	2199	1022,2	2,151242	0,394424
D	serial	9246,815	53337	34958,55	1,525721	0,264508	serial	7259,334	52315	30269,35	1,728316	0,239825
	C2D	4558,214	28688	19659,95	1,45921	0,231853	QCX	1856,224	14562	8355,8	1,742742	0,222148

Fig. 3. Variation coefficients for each experiment

5 Conclusion

We designed and developed parallel version of the GAME algorithm generating inductive models on multiple cores simultaneously. The speedup is considerable for up to four cores. For more cores, it is better to run several instances of the serial GAME algorithm with different configuration.

6 Acknowledgements

This research is partially supported by the grant Automated Knowledge Extraction (KJB201210701) of the Grant Agency of the Academy of Science of the Czech Republic and the research program "Transdisciplinary Research in the Area of Biomedical Engineering II" (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic.

References

- [1] <http://en.wikipedia.org/wiki/symmetricmultiprocessing>, August 2008.
- [2] P. Kordik. *Fully Automated Knowledge Extraction using Group of Adaptive Models Evolution*. PhD thesis, Czech Technical University in Prague, FEE, Dep. of Comp. Sci. and Computers, FEE, CTU Prague, Czech Republic, September 2006.
- [3] O. Koshulko and A. Koshulko. Adaptive parallel implementation of the combinatorial gmdh algorithm. In *IWIM, Prague, 2007*.
- [4] S. Krishnaprasad. Uses and abuses of amdahl's law. *J. Comput. Small Coll.*, 17(2):288-293, 2001.
- [5] S. W. Mahfoud. Niching methods for genetic algorithms. Technical Report 95001, Illinois Genetic Algorithms Laboratory (IlligAL), University of Illinois at Urbana-Champaign, May 1995.
- [6] S. Oaks and H. Wong. *Java Threads, Second Edition*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.
- [7] G. G. Robertson. Parallel implementation of genetic algorithms in a classifier system. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 140-147, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.