UDC 004.021

O.V. Kasitskyj, P.I. Bidyuk, L.O. Korshevnyuk

EFFECTIVE IMPLEMENTATION OF THE EXPECTATION MAXIMIZATION ALGORITHM USING GPGPU

The problem of decreasing of running time for the data processing algorithms is very important especially when they are used in real time. For example, in real time image processing, process control systems, speech recognition, etc. The paper considers the possibility of decreasing running time of the expectation maximization (EM) algorithm using modern computing systems. The proposed modified EM-algorithm is aimed at better parallelism for the general purpose graphical processing unit (GPGPU).The experimental results are obtained with solving of the classical problem of Gaussian random variables mixture separation. The proposed implementation of the algorithm was performed on one and two 8-core processor (CPU) setup, as well as on the general purpose graphical processing unit. The graphics processor, because of its abilities for parallel computations and due to the properties of the EM-algorithm considered, showed substantially higher effectiveness in all the computational experiments. Besides, the modified EM-algorithm showed almost two times faster performance on GPGPU than on one or two CPU using large sample sizes (from 5 million values and higher). The lower price of graphics processor is an additional advantage of the approach proposed for such parallel algorithms and GPGPU usage.

Introduction

The problem of effective implementation of computing algorithms is in the centre of attention for many researchers because decreasing of programs running time provides the users with substantial benefits in multifunctional systems [1, 2]. This is especially important in the case of solving the problems of data clustering, objects/subjects classification, image processing, and optimization where the number of computational operations can be very high and time consuming [3].

An expectation maximization algorithm (EM-algorithm) is widely used in mathematical and applied statistics, optimization theory and its multiple applications for computing unknown model parameters, imputing missed measurements, finding the minima and maxima values for various functions etc. One of its known applications is directed towards maximum likelihood estimation of unknown model parameters for probabilistic models in the cases when some investigated process variables cannot be measured directly.

The algorithm is functioning iteratively in two steps. At the first E-step (expectation step) an expected value of likelihood function is computed using current approximation for non-measurable variables. The M-step is used for computing the model parameter estimates that maximize the expected likelihood generated at the E-step. The EM-algorithm is also often used for data clustering, machine learning, in computer vision systems, and natural language processing (in this area it is known as a special case: Baum-Welch algorithm). Due to the possibility of functioning in the conditions of lost data the EM-algorithm is very useful instrument for portfolio risks estimation in analysis of various financial data [4]. Among other well-known applications is medical image processing: the positron emission tomography and the single-photon emission computed tomography.

We use the EM-algorithm to find the maximum likelihood estimates of parameters of a statistical model in cases where the equations cannot be solved directly. Typically these models involve latent variables in addition to unknown parameters and known data observations. That is, either there are missing values among the data, or the model can be formulated more simply by assuming the existence of additional unobserved data points.

The problem statement

It is necessary to find consistent unbiased estimates of random variables mixture model parameters. The estimation problem can be formulated in the following way: given a set of N points in a D-dimensional space, $x_1, x_2, ..., x_N \in R^D$, and a family of probability densities F on R^D, it is necessary find the probability density $f(x) \in F$, such that probability of generation of the set $x_1, x_2, ..., x_N$, is maximized, using this density.

The common technique for definition of the probability density family is assigning densities to have a common algebraic form with different parameters θ [5, 6].

Problem solution

As shown in [5], this problem has the following solution:

$$m_k = \frac{\sum_{n=1}^{N} p(k|n)x_n}{\sum_{n=1}^{N} p(k|n)}, \qquad (1)$$

$$\sigma_k = \sqrt{\frac{1}{D}\frac{\sum_{n=1}^{N} p(k|n)\|x_n - m_k\|^2}{\sum_{n=1}^{N} p(k|n)}}, \qquad (2)$$

$$p_k = \frac{1}{N}\sum_{n=1}^{N} p(k|n). \qquad (3)$$

The first two equations are easy to understand, because m_k and σ_k are the sample mean and the sample standard deviation, weighted by a probability of a given point to be generated from the k-th model. The third equation for the mixture is not so obvious, but not so hard to interpret, since the values of p_k are obtained as a sample mean of conditional probabilities $p(k|n)$.

An iterative computing procedure

The equations (1)–(3) are strongly connected because $p(k|n)$ on the right hand sides of both equations are dependent to all variables on the left side. This makes it hard to directly solve these equations. Nevertheless, EM-algorithm allows us to construct an iterative procedure for solving this system [7].

EM-algorithm is an iterative process, with the following steps:

1. E-step: estimate conditional expectation:

$$Q(\theta|\theta_n) = E_{Z|X,\theta_n}\{\ln P(X,z|\theta)\}.$$

2. M-step: maximization of the Q in relation to θ:

$$\theta_{n+1} = \arg\max_\theta Q(\theta|\theta_n).$$

Calculation the sum of current membership probabilities for each point, conditional membership probability calculation loop, expectation estimation loop and variance estimation loop were parallelized using OpenCL to obtain better performance using general purpose graphical processing unit (GPGPU).

Both, central processing unit (CPU) and graphical processing unit (GPU) algorithms were implemented so the GPU parallelization efficiency can be determined. For all tests we used the following formulae to validate the correctness and measure performance:

$$\text{Eps} = \frac{1}{K}\sum_{k=1}^{K}(m_k - m_k^*)^2 + \sum_{k=1}^{K}(\sigma_k - \sigma_k^*)^2,$$

where m_k^* and σ_k^* are estimated mean and standard deviation by the specific algorithm implemented.

The multithreaded CPU versions of the EM-algorithm

Multithreaded CPU version is implemented by means of OpenMP technology [8]. This implementation was tested in detail in [9]. We will only show some basic results achieved.

The tests were executed on a personal computer (PC) with Intel Core i7 2600k CPU. This CPU has 4 cores with Hyper-Threading technology, which makes it possible to execute 8 parallel threads [10]. In this test a mixture of two Gaussians with the same variance, $\sigma_1 = \sigma_2 = 1$, and means $m_1 = -m_2 = a$ were used. The sample size included 1000000 elements. The results are given in table 1.

***Table* 1.** EM-algorithm implementation on various CPU cores

№	Threads#	Eps	Time, sec
1	1	3,5e–8	43,66
2	2	3,5e–8	24,16
3	3	3,5e–8	19,20
4	4	3,5e–8	16,14
5	5	3,5e–8	14,36
6	6	3,5e–8	13,38
7	7	3,5e–8	12,87
8	8	3,5e–8	12,46

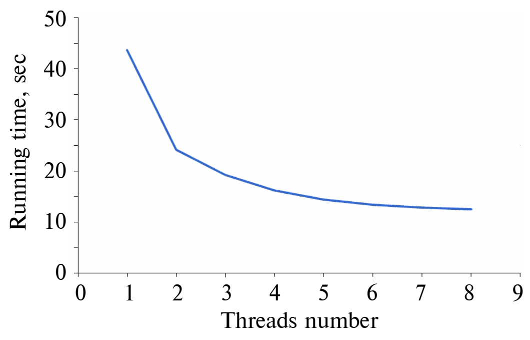

Fig. 1. EM-algorithm running time

The relation between running time and the number of CPU cores shows Fig. 1. As we can see from the Fig. 1, additional increase in CPU cores number will decrease time for a less and less value. That makes it clear that we should investigate other available technologies for EM-algorithm speeding up.

GPGPU versus Multithreaded CPU EM-algorithm comparison

All the tests were executed on a high-end workstation with two Intel Xeon E5-2670 CPUs and NVIDIA Quadro 4000 graphics card. Each of the two CPUs has 8 cores with Hyper-Threading technology, which makes it possible to execute 16 parallel threads on each CPU, and makes it 32 parallel threads in total [10]. The installed GPU has 256 computational units, each capable of executing 32 threads in a warp [11], giving the total of 8,192 active threads.

The CPU version is implemented using OpenMP technology and compiled with Intel C++ Compiler 2011. GPU version is implemented using OpenCL technology, and compiled by the device driver.

The mixtures of three Gaussians with the same variance, $\sigma_1 = \sigma_2 = \sigma_3 = 1$, and means, $m_1 = -2$, $m_2 = 0{,}1$, $m_3 = 1{,}5$, have been generated as the test data. The sample size was from 1000000 to 100000000 elements on different tests. The results are shown in table 2.

***Table* 2.** EM-algorithm implementation comparison: one CPU, two CPUs, and GPU

№	Sample size	Eps	Time, one CPU	Time, two CPUs	Time, GPU
1	100000	2,7E–4	1,76	1,79	1,58
2	200000	2,7E–4	3,45	2,60	2,32
3	400000	1,7E–4	7,24	5,52	3,93
4	800000	1,4E–4	14,7	11,5	7,07
5	1600000	7,2E–5	31,4	20,9	13,6
6	3200000	3,1E–5	73,3	64,3	26,6
7	6400000	1,4E–5	154	120	52,8
8	12800000	4,0E–6	291	226	105
9	25600000	8,3E–7	680	456	210
10	51200000	3,5E–7	1396	902	431

Other results of testing are shown in Figs. 2–4. Fig. 2 shows precision of EM algorithm computing results for the tests 1–10. Figs. 3, 4 show increase of running time for EM-algorithm with one CPU, two CPUs and GPU technology.

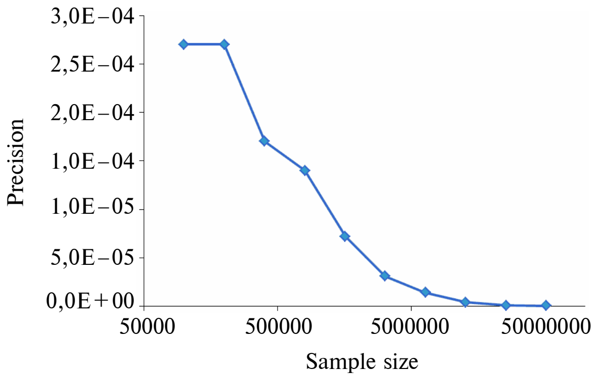

Fig. 2. EM-algorithm precision related to sample size

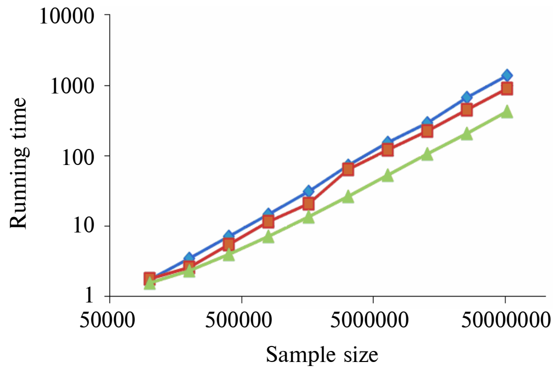

Fig. 3. Comparing EM-algorithm running times for alternative implementations

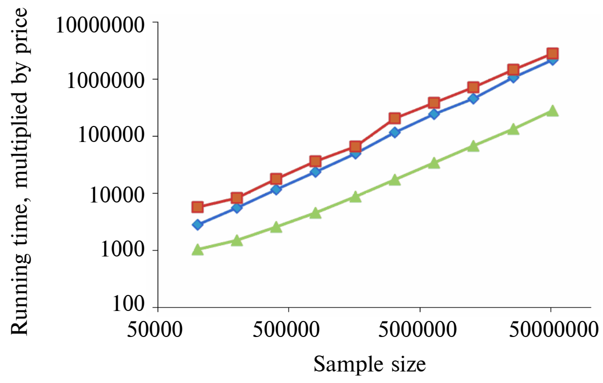

Fig. 4. Comparing normalized by price running times

As we can see from Fig. 3, GPU version is about two times faster on big sample sizes. That advantage becomes even more significant if we take the price into comparison. As of June 2013, one Intel Xeon E5-2670 CPU costs about 1600 USD, while Nvidia Quadro 4000 GPU costs about 650 USD.

In Fig. 4 a comparison is given for the chart of Fig. 3 normalized by the price.

As one can see, moving from one CPU to two CPUs increases the normalized running time, while using the GPU is not only faster, but it is much more cost effective approach.

The general effectiveness of the different processor setups of one CPU, two CPUs and GPU used for the implementation of the EM-algorithm is appropriate to compare with the integrated performance indicator (Setup Value). The proposed SV indicator is a relative dimensionless value that shows the effectiveness estimation of a certain algorithm implementation with processor setup on the variety of samples:

$$\mathrm{SV} = \frac{\mathrm{SM}}{\mathrm{Price}}, \qquad (4)$$

where SM – Setup Mark and Price is the cost of processor setup. Setup Mark is calculated as sum over all tests of the performance mark P_s of s-th test (sample) multiplied by a weighting coefficient w_s of this test:

$$\mathrm{SM} = \sum_{s=1}^{S} P_s \cdot w_s, \qquad (5)$$

where S is number of tests or in fact number of samples;

$$P_s = \frac{N_s}{t_s},\ w_s = \frac{N_s}{\sum_{i=1}^{S} N_i}, \qquad (6)$$

where N_s is size of s-th sample and t_s is running time for s-th test or sample. So the final formula for Setup Value over all samples for certain processor setup with (4)–(6) is as follows:

$$\mathrm{SV} = \frac{\sum_{s=1}^{S}\left(\frac{N_s}{t_s}\cdot\frac{N_s}{\sum_{i=1}^{S} N_i}\right)}{\mathrm{Price}}. \qquad (7)$$

In general, SV indicator is versatile and useful to compare effectiveness of the parallelized algorithm implementation on different types of processor setups. In fact, it is not always possible to make visual valid comparison of effectiveness with the running time graphs, because they may be overlapped several times on the whole interval of available sample sizes.

This phenomenon depends on features of specific algorithm or processor setup and can occur because of the nature of certain implementation of parallel calculations when a full effective usage of threads and decreasing of running time can only take place with an increasing sample to a certain size.

Another advantage of Setup Value (SV) is a usage of weighted values of Setup Mark (SM) depending on the size of a sample. On one side it is important to increase the efficiency of the algorithm implementation on all samples, and on the other side it is most important to increase the efficiency on big size samples.

It was established that decreasing of running time by 10 % on a big size sample in absolute values may be much more useful and beneficial than decreasing the running time on a small size sample by 15 or 20 %. That is especially significant factor for real-time multifunctional systems.

The calculated values of *SV* indicator (7) for one CPU, two CPUs, and GPU for considered ten samples and proposed modified EM-algorithm are shown in table 3.

***Table* 3.** Setup Values of effectiveness for one CPU, two CPUs and GPU

Value	One CPU	Two CPUs	GPU
Setup Mark	38853.63	56667.5	119913.30
Price, USD	1600	3200	650
Setup Value	**24,28**	**17,71**	**184,48**

The graphical representation of the ratio of the Setup Values of one CPU, two CPUs and GPU is shown in the Fig. 5.

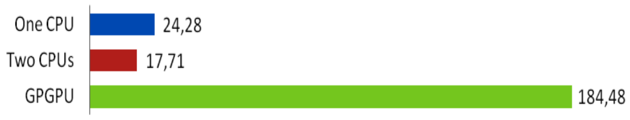

Fig. 5. Comparison of Setup Values for alternative implementations: one CPU, two CPUs, GPU

As one can see in the table 3 and Fig. 5, the GPU is significantly more efficient with running modified EM algorithm than others. And the setup of two CPUs is not so much more efficient, as far as more expensive. It has 2 times higher price and provides decrease in the complex efficiency of 27 %.

Conclusions

Theoretical and practical running time estimations were presented in the paper. The Gaussian mixture separation quality was measured for different sizes of mixture samples. In the mixture separation problem EM-algorithm performance degrades when the distance between mean values is less than three standard deviations, what is totally in the spirit of three sigma laws. In such cases, it is very important to have an efficient EM-algorithm implementation to be able to process such test cases in a reasonable time.

In our implementation the GPU version is about two times faster the two high-end Intel CPUs on big mixture samples. This advantage is very significant since the GPU costs two and half times less than one such CPU, and five times faster than the two CPU setup.

The implementation of the modified EM-algorithm on the setup of two CPUs does not provide the expected productivity growth. The proposed integrated effectiveness indicator Setup Value shows decrease of 27 % at the price higher in two times.

The normalized by price comparison of two CPU setup versus the GPU yields about ten times increase of execution speed per dollar what justifies the approach proposed.

The further researches should be focused on the development of new algorithms for large data sets processing that would be able to provide high degree of parallelism and thus show less running time and more practically needed efficiency.

1. *K. Mehlhorn and S. Thiel,* "Faster algorithms for bounds-consistency of the sortedness and the alldifferent constraint", in Proc. Principles and practice of constraint programming Int. Conf., San Francisco, 2000, pp. 306–319.
2. *R. Dechter,* Constraint processing. San Francisco: Elsevier Science, 2003, 481 p.
3. *Measurements*, models, systems and design, J. Korbicz, Ed. Warszawa: Wydawnictwa Komunikacjii Lacznosci, 2007, 507 p.
4. *W. Hu,* "Calibration of multivariate generalized hyperbolic distributions using the EM algorithm", Ph.D thesis, The Florida State University, 2005, 115 p.
5. *F. Dellaert,* "The expectation maximization algorithm", College of Computing, Georgia Institute of Technology, 2002, 7 p.
6. *Yan Shuicheng et al.,* "Sift-bag kernel for video event analysis", in Proc. 16th ACM Int. Conf. on Multimedia, 2008, pp. 229–238.
7. *S. Borman.* The expectation maximization algorithm – a short tutorial [Online]. Avaliable: http://www.seanborman.com/publications/EM_algorithm.pdf
8. *Ben-AriM.* Principles of Concurrent and Distributed Programming. New York: Addison-Wesley, 2006, 384 p.
9. Kasitskyi O.V., Bidyuk P.I. Effective EM algorithm implementation for multicore CPU systems // Системні науки і кібернетика. – 2013. – **3**, № 1. – С. 47–58.
10. *B. Chapman et al.,* Using OpenMP: Portable Shared Memory Parallel Programming. Boston: The MIT Press, 2007, 378 p.
11. *OpenCL* Programming Guide for the CUDA Architecture (2009) [Online]. Avaliable: http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf

Рекомендована Радою
Навчально-наукового комплексу
Інститут прикладного системного
аналізу НТУУ "КПІ"

Надійшла до редакції
17 червня 2013 року