IRSTI 50.07.07

Medetov B.[*], Koishigarin A., Yskak A., Niazaliev K., Naurzbayeva A.

IETP, Al-Farabi Kazakh National University,
Almaty, Kazakhstan [*]e-mail: bm02@mail.ru

A COMPARATIVE ANALYSIS OF OPENMP AND CUDA PERFORMANCE AS EXEMPLIFIED BY THE COMPUTATION OF FOURIER TRANSFORM

A comparative analysis of the performance of the two technologies of parallel computing, OpenMP and nVidia CUDA have been carried out as exemplified by the computation of Fourier transform. It was obtained that the execution time for the Fourier transform on multi-core central processor depends on the number of cores nonlinearly. In addition, the form of this dependence changes because of the number of threads: for the threads whose number is lower than that of hard cores the dependence is powerlike whereas for the threads whose number is higher than the hard cores number the dependence is exponential. The maximum efficiency of computation with the use of OpenMP can be achieved when the number of threads used in the program is twice the number of hard cores. The comparison conducted for this case showed that for a small number of frames OpenMP is more efficient in terms of execution time, otherwise, CUDA offers an advantage.

Key words: parallel computing, Fourier transform, NVIDIA CUDA, OpenMP, digital processing.

Медетов Б.[*], Қойшигарин А., Ысқақ Ә., Ниазалиев Қ., Наурызбаева А.

ЭТФҒЗИ, Әл-Фараби ат. Қазақ ұлттық университеті,
Алматы қ., Қазақстан [*]e-mail: bm02@mail.ru

Фурье түрлендіруін есептеу мысалында OpenMP мен CUDA өнімділіктерін салыстырмалы талдау

Жұмыста кең тараған екі параллель есептеу технологияларының спектральды талдауға негізделген салыстырмалы зерттеуі жүргізілді. Тәжірибелік әдіспен екі технологияның да есептеу өнімділігі бағаланды. Зерттеу барысында, көпядролы орталық процессорде Фурье түрлендіруінің орындалу уақытының ядролар санына тәуелділігі бейсызық заңға бағынатындығы анықталды. Осыған қоса, аталған тәуелділіктің формасы ағындардың санына байланысты өзгереді: ағындар саны процессордың физикалық ядролар санын аз болса тәуелділік дәрежелік түрге ие болса, ағындар саны процессордың физикалық ядролар санынан көп болса, тәуелділік экспоненциалды болады. OpenMP технологиясын қолдану арқылы ең үлкен өнімділікті ағындар саны процессордың физикалық ядролар санынан екі есе артық болғанда қол жеткізуге болады. Жұмыстағы салыстыру нәтижесінде фреймдердің аз санында OpenMP технологиясы тиімдірек болса, фреймдер саны өскен сайын CUDA технологиясының тапсырманы орындау уақыты азырақ болды.

Түйін сөздер: параллель есептеу, Фурье түрлендіруі, NVIDIA CUDA, OpenMP, цифрлы өңдеу.

Медетов Б.[*], Койшигарин А., Ыскак А., Ниазалиев К., Наурызбаева А.

НИИЭТФ, Казахский национальный университет им. аль-Фараби,
г. Алматы, Казахстан [*]e-mail: bm02@mail.ru

Сравнительный анализ производительности OpenMP и CUDA на примере вычисления Фурье преобразования

Сравнительный анализ производительности двух технологий параллельных вычислений – OpenMP и nVidia CUDA – был проведен на примере вычисления преобразования Фурье. Было получено, что время выполнения преобразования Фурье на многоядерном центральном процессоре зависит от количества ядер нелинейно. Кроме того, форма этой зависимости изменяется из-за количества потоков: для потоков, число которых меньше, чем количества физических ядер, зависимость является степенной, тогда как для потоков, число которых больше, чем количество физических ядер, зависимость является экспоненциальной. Максимальная эффективность вычислений с использованием OpenMP может быть достигнута, когда количество потоков, используемых в программе, в два раза больше количество физических ядер. Данное сравнение показало, что в условиях проведенных экспериментов для небольшого количества фреймов наибольший выигрыш по времени дает OpenMP, а в противном случае превосходство получает уже CUDA.

Ключевые слова: параллельное вычисление, преобразование Фурье, NVIDIA CUDA, OpenMP, цифровая обработка.

Introduction

With advancement of science and technology the researchers encounter increasingly complex problems which require an enormous number of computations to be solved. At present the best available solution to provide the required computation capacity is parallel computing. There are several options, the most common of them being OpenMP for parallel computing on a multi-core central processor and nVidia CUDA on the basis of a graphic processor. With such diversity the question is which technology is the most appropriate for solving a specific problem? This issue has been discussed and studied quite a while.

Thus, a comparison [1] was made of the open parallel computing systems on different hardware platforms. In another study [2] parallel computing was applied to the neural network modelling and performance comparison of central and graphic processors made. In the work [3] OpenACC, OpenMP and CUDA technologies are compared for the computation of various tasks such as matrix multiplication, Mandelbrot set calculation etc. A comparison of three parallel computing technologies OpenMP, nVidia CUDA and StarPU was made by the example of matrix multiplication [4]. In the research [5] an experiment was conducted to evaluate a cluster of two graphic processors. Calculations were performed using a “hybrid” method: two technologies of parallel computing were used simultaneously.

In our research we investigated certain problems of parallel audio signal processing. Our primary focus was on comparative analysis of OpenMP and CUDA performance in the computation of Fourier transform.

The complete audio signal processing cycle consists usually of the following main phases:

1. Data preparation
2. Parameter computation (vectorization);
3. Codebook compilation

At the first phase of audio signal processing silence and very noisy lengths are removed, the signal is segmented into quasi-stationary lengths and so on.

At the second phase certain parameters of an audio signal are calculated, e. g. base frequency, cepstral coefficients, formants and others. Many of these parameters are identified with the help of the fast Fourier transform (FFT). At the last phase, i. e. during codebook compilation, such methods as Gaussian mixtures, hidden Markov models and others are often used [6].

To assess the execution time for each phase we used a program which performs all three phases of the audio signal processing. It was found that Fourier transform at the second phase takes 70-75% of the total execution time. Therefore, a faster signal spectrum computation could significantly accelerate audio signal processing as a whole. Thereby, it is possible to shorten the processing time of phone calls and other audio signals in telecommunication centers, call-centers in various organizations etc. Therefore, a faster signal spectrum computation is quite a topical problem. With this purpose in mind we performed a comparative performance analysis of two parallel computing technologies, OpenMP и nVidia CUDA.

OpenMP technology (Open Multiprocessing) is an applied programming interface (API) for the parallel programming with the use of shared memory. C, C++ and Fortran programming languages as well as Solaris, AIX, HP-UX, Linux, Max OS X, Windows operating systems are supported. OpenMP is developed with the participation of big IT companies, such as AMD, Intel, IBM, Cray and others [7].

CUDA (Compute Unified Device Architecture) is the hardware-software platform for parallel computing using nVidia graphic processor resources for non-graphic computations [8]. CUDA development started in 2006, C, C++ and Fortran programming languages as well as Windows 8, Windows XP, Windows Vista, Linux, Mac OS X operating systems are supported.

Experiment

An experiment was conducted on the measurement of Fourier transform execution time on a multi-core processor (OpenMP) and a graphic processor (CUDA). The experiments were carried out on the hardware-software platform with the following characteristics:

1. Processor: Intel Xeon E5-2620, CPU clock 2GHz (2.5 GHz with Turbo Boost technology), number of cores/threads – 6/12;
2. Graphic processor: NVIDIA Tesla C2075;
3. Operating system: Windows 8, 64 bit, RAM: DDR3, 16Gb;

Experimental results with OpenMP

Below is a fragment of the code written in C++ programming language and designed for the experimental evaluation of Fourier transform execution time with the application of OpenMP technology:

```
for (int p = 12; p >= 1; p --)
{
 clock_t t_beg, t_end;
 int idxFr = 0;
 float Z_Cnt = 50.0;
 for(int FrNum = 500; FrNum <= 200000;
FrNum += ((FrNum < 5000) ? 500 : 5000))
{
 t_beg = clock();
 for(int i = 0; i < Z_Cnt; i++)
 {
 #pragma omp parallel for num_threads(p)
 for(idxFr = 0; idxFr < FrNum; idxFr++)
 {
 CalcFourier(data_in, data_out, idxFr);
 };
 };
 t_end = clock();
 cout << "Cores: " << p << " FrCnt = " <<
FrNum << " Time: " << (float)(t_end-
t_beg)/(Z_Cnt) << endl;
};
};
```

This fragment consists of four *for() cycles.* In the first, i. e. the outermost, cycle the number of threads is specified through *p* variable. In the second cycle the number of segments (frames) is changed and in the third one the number of repeated experimental measurements of the code execution time is specified. The experiment was conducted 50 times and finally the average computation time of Fourier transform was determined. Fourier transform itself is calculated using *CalcFourier(data_in, data_out, idxFr)* function. Parallelizing is performed with the application of *#pragma omp parallel for num_threads(p)* directive. The last line of the code displays the number of threads, the number of frames and corresponding Fourier transform execution time for every iteration of the outermost cycle.

Table 1 – The dependence of Fourier transform execution time on the number of core/threads

Number of core/threads	Execution time (ms)
1	3792
2	1989
3	1399
4	1050
5	843.9
6	705.1
7	883
8	817.4
9	750.3
10	695.8
11	634.9
12	589.7

Illustrated in Fig 1 is the experimental relationship between Fourier transform execution time and the number of threads in the case of 100,000 (one hundred thousand) frames.

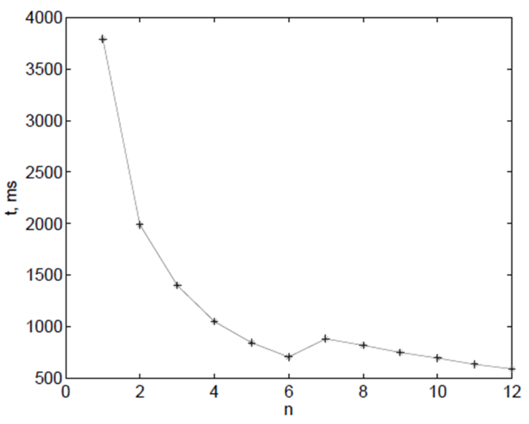

Figure 1 – The dependence of Fourier transform execution time (t) on the number of core/threads (n) (OpenMP). Number of frames 100,000

It is seen that at n=7 there is a jump on the curve of the execution time dependence on the number of threads. Taking this into consideration we examined the two segments separately in order to obtain the analytical dependence of execution time on the number of threads. The first segment corresponds to $0 < n \leq 6$ and the second one to $6 < n \leq 12$. It can be seen that in each of the segments the execution time decreases monotonically. In that context, the following two functions were selected for modelling this dependence:

$$T_1(n) = a * n^b \text{ и } T_2(n) = a * \exp(b * n). \quad (1)$$

Where a and b are several constant coefficients, n is the number of threads. Then, based on experimental data with the use of the least square method a and b coefficients in the formula (1) were defined and computational error estimated. *T*₁*(n)* function can be represented logarithmically as:

$$log(T_1(n)) = log(a) + b * lo\,g(n). \quad (2)$$

If the following notation is introduced: $y = log(T_1(n))$, $x = log(n)$, $A = log(a)$, then on a logarithmic scale $T_1(n)$ function will represent linear function of the next form:

$$y = A + b * x. \quad (3)$$

By introducing the following notation: $y = log(T_2(n))$, $A = log(a)$, $x=n$, for $T_2(n)$ function we can derive corresponding straight-line equation of the form (3).

A and *b* coefficients in the equation (3) were defined from experimental data using the least square method as follows:

$$b = \frac{\sum[(x_i - \bar{x})y_i]}{\sum(x_i - \bar{x})^2}, \quad (4)$$

$$A = \bar{y} - b * \bar{x}, \quad (5)$$

where $\bar{x}$ и $\bar{y}$ are mean values of x and y respectively that are calculated using the following formulas:

$$\bar{x} = \frac{1}{N}\sum x_i, \quad (6)$$

where *N* is the total number of points, in this case *N*=6.

Mean square root errors of determination of *A* and *b* are calculated as:

$$S_b = \sqrt{\frac{\sum(y_i - b * x_i - A)^2}{(n-2)\sum(x_i - \bar{x})^2}} \quad (7)$$

$$S_A = \sqrt{\left(\frac{\sum(y_i - b * x_i - A)^2}{(n-2)}\right)\left(\frac{1}{n} + \frac{\bar{x}^2}{\sum(x_i - \bar{x})^2}\right)}. \quad (8)$$

Relative error of *A* and *b* coefficients determination is calculated from the formulas:

$$\varepsilon_A = \frac{S_A * t_\alpha}{A} * 100\%, \quad (9)$$

$$\varepsilon_b = \frac{S_b * t_\beta}{b} * 100\%, \quad (10)$$

where t_α and t_β are Student's coefficients, for the number of measurements 6: $t_\alpha = t_\beta$=2,45.

The table 2 lists error values calculated from the formulas (9) and (10) for each segment and each function type.

Table 2 – Relative errors of A and b parameter calculation

Segment	Function	ε_A	ε_b
1	$T_1(n)$	0,37	2,58
	$T_2(n)$	4,49	29,88
2	$T_1(n)$	2,14	10,47
	$T_2(n)$	0,32	2,96

It follows from the table 2 that the first segment is very well approximated by $T_1(n)$ function and the second segment, on the contrary, by $T_2(n)$ function. Thereby, execution time dependence on the number of threads is piecewise nonlinear. For the threads whose number is lower than that of hard cores this dependence is powerlike whereas for the threads whose number is higher than the number of hard cores the dependence is exponential.

Fig 2 illustrates $T_1(n)$ and $T_2(n)$ function graphs for the first segment (at $0 < n \leq 6$). Based on experimental evidence these functions are of the following form:

$$T_1(n) = 3821 * n^{-0.94}, \quad (11)$$

$$T_2(n) = 4227 * e^{-0.32*n}. \quad (12)$$

Fig 3 illustrates $T_1(n)$ and $T_2(n)$ function graphs for the second segment (at $6 < n \leq 12$). Based on experimental evidence these functions are of the following form:

$$T_1(n) = 3895 * n^{-0.75}, \qquad (13)$$

$$T_2(n) = 1565 * e^{-0.08*n}. \qquad (14)$$

Thereby, general function of Fourier transform execution time dependence on the number of threads on a multi-core processor using OpenMP technology is of the following form:

$$T(n) = \begin{cases} 3821 * n^{-0.94}, \text{at } 0 < n \leq 7 \\ 1565 * e^{-0.08*n}, \text{at } 6 < n \leq 12 \end{cases} \qquad (15)$$

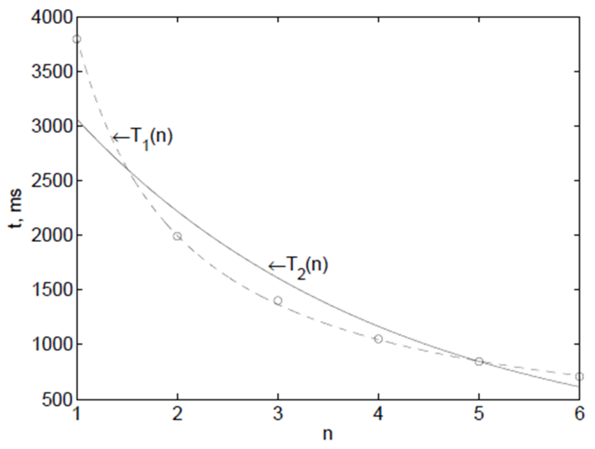

Figure 2 – Computation time (*t*) dependence on the number of threads (*n*) on the central processor (solid line – exponential function, dash line – power function)

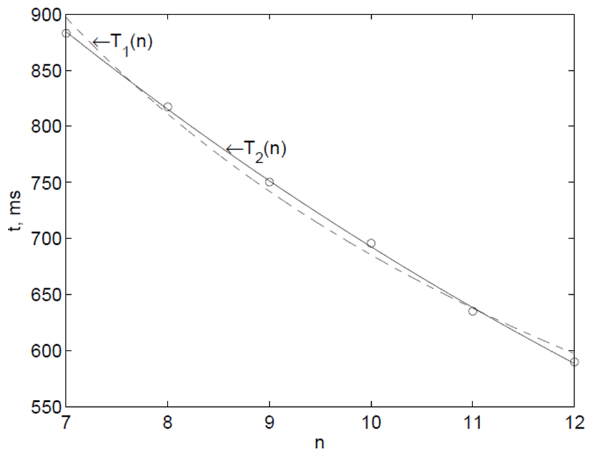

Figure 3 – Computation time (*t*) dependence on the number of cores/threads (*n*) on the central processor (solid line – exponential function, dash line – power function)

Experimental results with CUDA

The experiment was conducted using the following code:

```
clock_t t1,t2;
cufftHandle plan;
cufftComplex *dev_out;
float * dev_in;
int n[1] = {NX};
float Exp_Cnt = 50.0;
for (int FrCnt = 500; FrCnt <= 500000; FrCnt += ((FrCnt < 5000) ? 500 : 5000))
{
t1 = clock();
for(int m = 0; m < Exp_Cnt; m++)
{
cudaMalloc((void**)&dev_out, (NX/2+1) * FrCnt * sizeof(cufftComplex));
cudaMalloc((void**)&dev_in, FrCnt * NX * sizeof(float));
cudaMemcpy(dev_in, host_in, FrCnt * NX * sizeof(float), cudaMemcpyHostToDevice);
cufftPlanMany(&plan, 1, n,
 NULL, 1, 0, //advanced data layout, NULL shuts it off
 NULL, 1, 0, //advanced data layout, NULL shuts it off
 CUFFT_R2C, FrCnt);
cufftExecR2C(plan, dev_in, dev_out);
cudaMemcpy(host_out, dev_out, (NX/2+1) * FrCnt * sizeof(cufftComplex), cudaMemcpyDeviceToHost);
cufftDestroy(plan);
cudaFree(dev_out);
cudaFree(dev_in);
};
t2 = clock();
cout << "FrCnt = " << FrCnt << " time = " << (t2 – t1)/Exp_Cnt << endl;
};
```

CUDA has a built-in function “cufft” which enables fast Fourier transform in parallel mode. In this function *cufftPlanMany(...)* transformation plan is created and then implemented with *cufftExecR2C(...)* command. Similar to OpenMP technology in this experiment the measurement is performed 50 times and the mean transformation time is calculated.

Fig. 4 displays transformation time dependence on the number of frames.

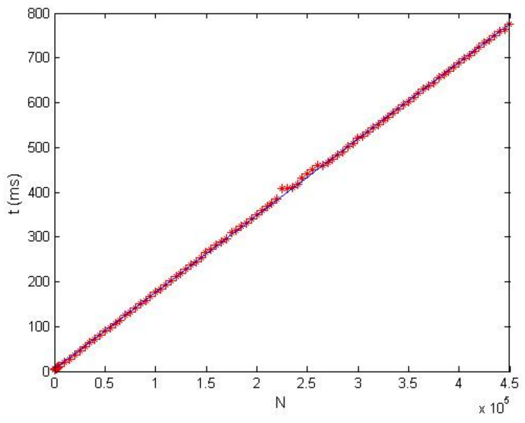

Figure 4 – Fourier transform execution time (t) dependence on the number of frames (N) using CUDA technology

Comparison of experimental results

We made a comparative analysis of Fourier transform execution time dependence on the number of frames. This dependence for OpenMP is linear as is the case for CUDA. Fig. 5 displays these dependences in one chart.

The chart indicates that for a large number of frames Fourier transform execution time with the use of CUDA technology is much shorter than with OpenMP. However, for a relatively small number of frames the execution time with OpenMP proves to be shorter than with CUDA as shown on Fig. 6. From the Fig. 6 it follows that, if the number of frames does not exceed 300 the OpenMP technology is more efficient for computation of Fourier transform, otherwise, i. e. with a large number of frames, CUDA is a better option.

Conclusion

As was shown in our experimental research the execution time for the Fourier transform on multi-core central processor depends on the number of cores nonlinearly. In addition, the dependence is not continuous, it changes because of the number of threads. The general form of function corresponding to this dependence follows the formula (15). It is possible that this form of dependence also applies to any other similar tasks and not only to the computation of Fourier transform.

The maximum efficiency of computation with the use of OpenMP can be achieved when the number of threads used in the program is twice the number of hard cores (see table 1).

Therefore, in our experiment we made a comparison between OpenMP and CUDA for the case in which the number of threads was 12 on a multi-core processor. The comparison showed that under the conditions of the experiments for a small number of frames OpenMP is more efficient in terms of execution time, otherwise, CUDA offers an advantage.

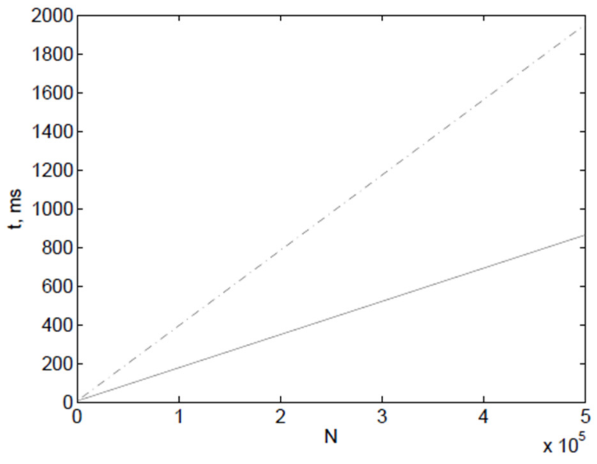

Figure 5 – Execution time (t) dependence on the number of frames (N). Dash line – OpenMP, solid line – CUDA

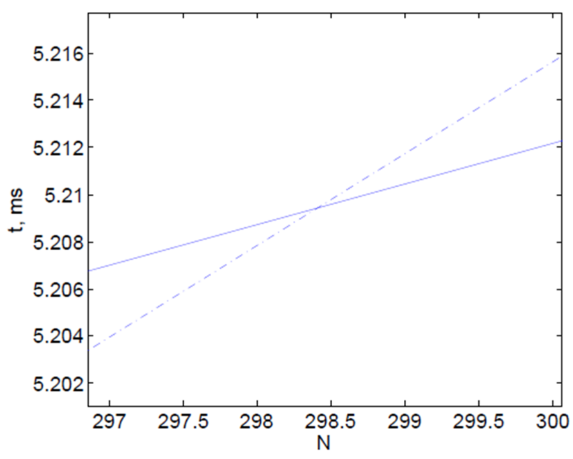

Figure 6 – The initial section of Fourier transform execution time (t) dependence graph on the number of frames (N). Dash line – OpenMP, solid line – CUDA

References

1 Chu S.L., Hsiao C.C. The Comparisons of OpenCL and OpenMP Computing Paradigm, International // Journal of Applied Mathematics & Information Sciences, Apr. – 2014. – P.333-340.

2 Dinkelbach H.Ü., Vitay J., Beuth F. and Hamker Fred H. Comparison of GPU-and CPU-implementations of mean-firing rate neural networks on parallel hardware // Computation in Neural Systems. – 2012. – Vol.23(4). – P.212-236.

3 Ledur C.L., Zeve C.M.D., C.S. dos Anjos J. Comparative Analysis of OpenACC, OpenMP and CUDA using Sequential and Parallel Algorithms // 11th Workshop on Parallel and Distributed Processing (WSPPD), 2013.

4 Khankin K.M. Efficiency comparison of OpenMP, nVidia CUDA and StarPU technologies by the example of matrix multiplication // Messenger of SUSU. Computer technology, management, electronics series. – 2013. – Vol.13, № 1. – P.34-41.

5 Yang C.-T., Huang C.-L., Lin C.-F. Hybrid CUDA, OpenMP, and MPI parallel programming on multi-core GPU clusters // Computer Physics Communications. – 2011. – Vol.182. – P.266–269.

6 Rabiner L.R., Schafer R.W. Digital processing of speech signals. – Prentice-Hall, 1978.

7 OpenMP Application Program Interface. Version 3.1 July 2011. – http://www.openmp.org/mp-documents/OpenMP3.1.pdf

8 What is CUDA. – http://developer.nvidia.com/what-cuda

9 Hastie, Tibshirani and Friedman: The Elements of Statistical Learning (2nd edition). – Springer-Verlag, 2009. – 763 p.

References

1 S.L. Chu, and C.C. Hsiao, Journal of Applied Mathematics & Information Sciences, Apr., 340, (2014).

2 H.Ü. Dinkelbach, J. Vitay, F. Beuth and Hamker Fred H., Computation in Neural Systems, 23(4), 212-236, (2012).

3 Cleverson Lopes Ledur, Carlos M. D. Zeve, Julio C. S. dos Anjos, 11th Workshop on Parallel and Distributed Processing (WSPPD), 2013.

4 K.M. Khankin, Messenger of SUSU, Computer technology, management, electronics series, 13(1), 34-41, (2013).

5 Yang C.-T., Huang C.-L., Lin C.-F. Hybrid, Computer Physics Communications, 182, 266–269, (2011).

6 L.R. Rabiner and R.W. “Schafer Digital processing of speech signals”, (Prentice-Hall, 1978).

7 OpenMP Application Program Interface. Version 3.1 July 2011. http://www.openmp.org/mp-documents/OpenMP3.1.pdf

8 What is CUDA. – http://developer.nvidia.com/what-cuda

9 Hastie, Tibshirani and Friedman, “The Elements of Statistical Learning” (2nd edition). (Springer-Verlag, 2009, 763 p.)