UDK 624.04

Eduard Bondarenko

MODERN PARALLEL PROGRAMMING TOOLS FOR SOLVING SYSTEM OF THE LINEAR EQUATIONS BY CONJUGATE GRADIENT METHODS

Annotation. The work is devoted to comparison the performance of programs which are executed on the CPU and GPU with shared memory space. OpenMP technology was used for increasing performance by parallelizing the code on the CPU. CUBLAS and OpenACC technologies were used for computing on the GPU. For performance tests several conjugate gradient methods for solving system of linear equations with the dense matrix were implemented. Within OpenMP technology, this paper contains a performance comparison of programs which were compiled using two compilers: PGCC and GCC. Also is viewed the performance boost through the use of the accelerator and various types of technologies for program acceleration. The primary purpose of this paper is to show the distinct advantages of using GPU-accelerator and make the conclusions associated with the use of various parallelization approaches.
Keywords: Accelerators; GPU; OpenACC; OpenMP; CUBLAS.

Introduction

In the development process of computer technology the idea of increasing the productivity of computer systems by increasing the number of processors was implemented more than one times. No exaggeration to say, that all history of the development of highly productive systems this history of implementations the ideas of parallel processing at a particular stage in the development of computer technology, of course, coupled with the increasing frequency of electronic circuits.

Today there are several approaches to increase the speed of data processing. They are classified by the type of devices, on which parallel computing are held.

The most common approach is to perform parts of code simultaneously on the multiple CPUs. OpenMP is the one of the most popular means of programming parallel applications for computers with shared memory. A sequential program is taken for basis, and for creating her

 ISSN 1562-9945

parallel version the user is given a set of directives, functions and environment variables. OpenMP technology is aimed at the opportunity to have one version of the program for parallel and sequential execution. Also an important advantage of this technology is the ability to incremental programming. Thus, not parallel part of the program is gradually decreased. That approach significantly facilitates the adaptation process of sequential programs for parallel computing as well as debugging and optimization.

But, restrictions on power consumption, heat release and fast approaching the physical limit of the transistor size are forcing researchers and manufacturers to seek a solution elsewhere.

Appearance on the market GPU with the programmable conveyor attracted the attention of many researchers to the possibility of using graphic devices not only for image rendering using OpenGL and DirectX. Because speed of performance of the arithmetic operations on the GPU was very high, results of the first experiments had predicted a bright future for GPU-computations. But the need to work within strict constraints on resources and programming methods was too serious obstacle to widespread acceptance of the technology.

Only with the advent of graphic devices from NVIDIA, which were built on the CUDA architecture and CUDA C compiler, programming on the GPU for the general purpose computing become much more affordable and accessible and thus covers the maximum number of developers.

This method of paralleling programs is less comfortable than OpenMP technology, but allows you to get a big performance boost by executing code on the video adapter. In this approach programmer needs to support both single-threaded and multithreaded versions of the codes.

OpenACC standard is a logical sequel of the development of heterogeneous software. Like OpenMP, he is based on the directive programming and therefore requires much less structural changes in the code, than low-level programming model on the display adapters (CUDA). In this approach, a developer can quickly determine the benefits of using the accelerator.

Today standard OpenACC has support in such commercial compilers as: PGI Accelerator C/C++ and Fortran [13], Cray CCE compilers for Cray systems [14], CAPS compilers [15]. There is a free version of the

OpenACC compiler – accULL [16]. Unfortunately the latter is not fully support standard OpenACC 1.0 [9], therefore not considered in this paper.

Problem Definition

Currently, parallelism is the main trend of the development of computer technology: any modern computer consists of several cores on the central processors and several hundred cores on the graphics processors. Development of the technologies is determines the development of numerical algorithms: more time is devoted to parallel numerical methods. Now, any algorithm is viewed through the prism of his ability to parallelization.

Partial differential equations is a widely used mathematical apparatus in different areas of science and technology. Unfortunately, the explicit solution of these equations in analytical form is only possible in special simple cases. Thus the possibility to analyze the mathematical models which are based on a large number of differential equations is provided by means of approximate numerical methods. The volume of the computation in this area of computational mathematics usually is very significant so the traditional approach in this case is to use high performance parallel computer systems.

Conjugate gradient methods are among the most efficient methods for solving the linear systems. The aim of this work is a consideration of several types of gradient methods for solving SLAE and consideration the range of issues, related to their parallelization on the CPU and GPU with usage of two compilers.

During the development of a parallel version of the conjugate gradient method for solving SLAE keep in mind, that the execution of the iterations performed sequentially, therefore, the most appropriate approach is to parallelize computations, implemented in the course of iterations.

Main Results

Common Details

Three conjugate gradient methods for solving SLAE with 10 000 double-precision variables were used for getting results in this work:

- Conjugate Gradient [2]
- Conjugate Gradient Squared [3]
- Generalized Product Bi-Conjugate Gradient [4]

The values presented in the matrix were generated symmetrically in range from 0 to 1. During computing two criteria were used for stopping: by accuracy (1) with parameter $E = 10^{-6}$ and by count of iterations.

$$\left\| x^{(s)} - x^{(s-1)} \right\| < E \quad (1)$$

The second criterion is determined by the maximum number of iterations N, to which the researcher is ready to go. If the number of iteration, which must be performed, is greater than N, iteration will not be executed, method finishes the work, vector X(N) is interpreted as an approximate solution of the system.

Calculations were performed using infrastructure which is specified in Table I.

The simple Conjugate Gradient method requires 13 iterations for solving system of the linear equations. Stabilized Conjugate Gradient method requires 8 iterations. Generalized Product Bi-Conjugate Gradient method is more efficient than other methods, thus requires only 7 iterations.

Results of experiments are given in Table II, Table III, Table IV, Table V (running time of the algorithms is presented in seconds).

Table I

Infrastructure of the computer

Processor	Intel i5-3350P CPU @ 3.10GHz
RAM	8Gb
Graphic adapter	NVIDIA GeForce GTX 650 Ti 1Gb
Operating system	Windows 7
Compiler, debugger	cuda compilation tools 5.5 pgcc 13.10-0 64-bit gcc version 4.8.1
Math library	CUBLAS

Several compilers have been used at the work for completeness of the tests.

Comparison of PGCC and GCC Compilers

At the time of writing article, GCC was not support OpenACC standard, so it was used for comparing with PGCC compiler exclusively within the OpenMP technology.

From the data, specified in the Table II, Table III, we can conclude that the PGCC compiler much better optimizes the code of the program, which compiled without optimization, thereby increasing his productivity.

Using the PGCC compiler by developer, allows up to three times increasing the speed of the program, without optimization.

Table IV and Table V show that PGCC compiler optimizes the code two times better, than GCC in case of using optimization.

Table II

PGCC Compiler Without Optimization (seconds)

Method	1	2	3	4	5
CG	1.9562	1.0139	0.7051	0.6645	4.5490
CGS	2.4788	1.3258	1.0261	0.9558	4.5873
GPBiCG	2.2527	1.1407	0.7958	0.7053	6.0232

Table III

GCC Compiler Without Optimization (seconds)

Method	1	2	3	4	5
CG	5.7391	3.2139	2.3024	1.8460	1.8702
CGS	7.2587	3.9506	2.6844	2.1637	2.2199
GPBiCG	6.3615	3.4349	2.3929	1.9034	1.9436

Table IV

PGCC Compiler High Level Speed Optimization (seconds)

Method	1	2	3	4	5
CG	0.9439	0.6086	0.5702	0.5748	1.1577
CGS	1.1968	0.7517	0.7148	0.7263	1.3382
GPBiCG	1.0511	0.6751	0.6387	0.6453	1.4470

Table V

GCC Compiler High Level Speed Optimization (seconds)

Method	1	2	3	4	5
CG	1.6130	1.1045	0.9156	0.8798	0.8971
CGS	1.9874	1.2199	0.9593	0.9469	0.9813
GPBiCG	1.7658	1.0856	0.8765	0.8767	0.8766

Working time of program which uses five threads is increased. This explained by the insufficient computational load, which falls on each processor. It's disadvantageous to use more than four threads for this task on quad-core processor.

Despite the fact that PGCC compiler is generating faster code, execution of the program which was compiled by this compiler, on the five threads takes more time, than execution of program which was compiled using GCC. Perhaps, PGCC adapts the code to the physical number of processors in the system, thereby increasing speed of his work.

Comparison of OpenMP and OpenACC Technologies

The diagram Fig. 1 shows the work of the programs that are running on the CPU with usage several cores and the work of program that

is running on the graphic adapter. The diagram clearly shows, how program, written by using OpenACC is working faster, than her analogues, which are running on the central processor. Transferred into hardware thread scheduler, faster memory and more homogeneous processor provide an opportunity to get a big performance boost by using such type of accelerator. The bottleneck in this approach is the copying the data for processing to the device and back into memory. Thus, to get the maximum performance necessary to reduce the number of copy operation.

In the program, the results of which are shown above at the diagram, copying data occurs two times, at the beginning of work, to download initial data and upon completion, to obtain the result.

Unfortunately, memory of graphic adapter allows carrying out calculations only with 10 000 double-precision variables. When solving linear algebraic equations with more number of unknowns, increment of productivity in computations on the accelerator will be more explicit.

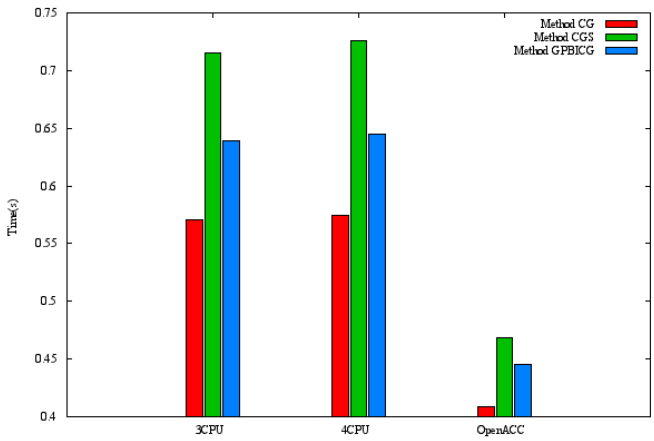

Figure 1 - Comparison of the computation time on the CPUs and GPU with usage OpenMP and OpenACC technologies

Comparison of OpenACC and CUBLAS Technologies

Testing several approaches of the programming on the graphic cards is concluded in comparing the performance of the program written using the mathematical library CUBLAS with the program, which uses standard OpenACC.

The results are shown in the diagram Fig. 2.

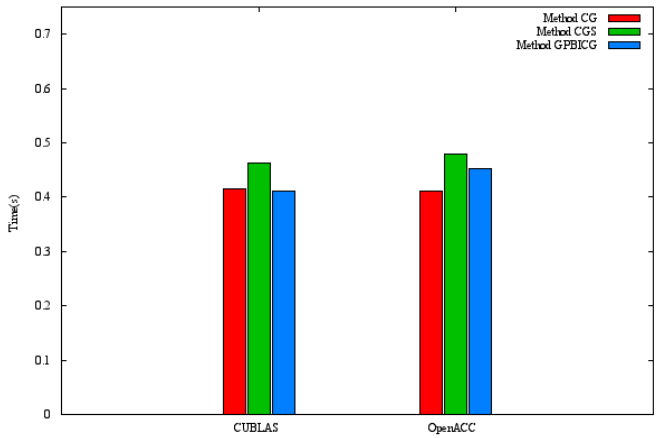

Figure 2. - Comparison of the computation time on the GPU with usage OpenACC and CUBLAS technologies

As expected, the work time of programs is approximately the same. Slight increase in operating time when using OpenACC is that the program is adapted for implementation in single-threaded mode on the CPU, and for multithreaded execution on the graphics card. In the latter case, the memory for working on the CPU will be allocated even if you use the version of the program, which running on the accelerator. If desired, the code of memory allocation can be enclosed in preprocessor directives, thereby to leave from unclaimed calls of allocator.

Using CUBLAS technology is time-consuming to change the code and to debug the application, therefore less justified than usage a high-level directive-based approach OpenACC.

CONCLUSIONS

The results of the study were shown that by count of iterations the generalized product bi-conjugate gradient method is the most effective method of solving the system of linear equations with dense matrix. But, by performance time, the usual method of conjugate gradients proved to be the fastest. Despite the fact that he is solving SLAE, using more number of iterations, than other methods, unlike them he is simpler and contains fewer operations with matrices and vectors. This leads to better optimization and increases the speed of work. In this paper was observed that the application of OpenMP for parallelization of computation allows not only create and support parallel versions of programs, with great convenience, but also allows increasing their performance. In this case, the performance boost was 40 percent. Modification of the existing program on OpenACC technology required minimum cost and gave

30 percent increase in productivity in comparison with the program on OpenMP.

Usage of OpenACC allows getting away from low-level API, thereby focusing developer's attention on the program's logic and saving time at debugging. In addition, the programmer receives the adapted program for various accelerators, whose support is included in the standard.

Importantly note that the use of various types of accelerators gives a performance boost only in case, if the work with the data takes much longer than copying them from the host to the device.

REFERENCES

1. Richard Barrett, Michael W. Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, Henk van der Vorst, "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods" SIAM, (1994).
2. M. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Stand., 49 (1952).
3. P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput., 10 (1989).
4. S. L. Zhang, GPBi-CG: Generalized Product-type Methods Based on Bi-CG for Solving, Nonsymmetric Linear Systems. SIAM J. Sci. Comput., 18 (1997).
5. The Portland Group. PGI Accelerator Programming Model for Fortran & C, v1.3 (2010).
6. NVIDIA. CUDA C Programming Guide, v4.6 (2012).
7. PGI® Compiler User's Guide Parallel Fortran, C and C++ for Scientists and Engineers (2013).
8. PGI Accelerator Compilers OpenACC Getting Started Guide, v13.2 (2012-2013).
9. The OpenACC Application Programming Interface, v1.0 (2011). Available: http://www.openacc-standard.org
10. OpenMP Application Program Interface, v3.0 (2008).
11. Tutorial for studying courses "Parallel processing" and "Computing languages and cluster systems" M. M. Yas'ko – DNU, 2010.
12. CUDA CUBLAS Library (2007).
13. PGI Compilers & Tools http://www.pgroup.com/index.htm
14. Cray compilers http://www.cray.com/Home.aspx
15. CAPS compilers http://www.caps-entreprise.com
16. Acc ULL compiler http://accull.wordpress.com