One Approach to Computational Load Balancing within the Node of Hybrid Computing System

Tat'yana Baranova[1[0000-0002-2003-3737]], Alexander Bugerya[1[0000-0002-9698-458X]], Ekaterina Gladkova[1[0000-0001-9536-4867]] and Kirill Efimkin[1[0000-0002-1087-7645]]

[1] Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, 125047, Moscow, Russia
shurabug@yandex.ru

Abstract. The issues of the computations distributing within single node of a hybrid computing system for applied programs with computation-intense operations are considered in this paper. There are two methods proposed: a method for static distribution of computations and a method for automatic balancing of the computational load during program execution, which is based on periodic analyzing the CPU load by the executed program and making decision whether redistribution of computational load is needed. The proposed methods are implemented in an applied program that solves a gas dynamic problem using the computing resources of the multicore central processor and graphics accelerators. The results of program execution with various data distributions were obtained and analyzed, both with and without the mechanism for automatic balancing of the computational load.

Keywords: parallel programming, programming automation, computational load balancing, hybrid computing architectures, NORMA language, automatic program generation.

1 Introduction

In the modern world there are a huge number of different problems to solve and it requires significant computing power. We have computation-intense problems in science and industry, in business and for individual purposes. Typical examples of such resource-intense problems are numerical methods in solving mathematical physics equations (e.g., modeling of processes occurring in a nuclear reactor), modeling of physical, chemical and biological processes. New challenges of this kind are constantly emerging. Modern computing systems for such problems to solve provide the possibility of parallel computing. Therefore if the program is effective and up to date it must be parallel.

There are various methods to automate the development of parallel programs. Monographs [1, 2] are devoted to the subject. There were strictly formulated mathematical basis of joint study of parallel numerical methods and parallel computing systems and investigated the task of mapping the program on the architecture of a

parallel computer. The idea of automatic mapping of given sequential program on parallel computing system is generally stated there as a NP-complete problem. This fact explains the up to date lack of a practical satisfactory universal method for the development of parallel programs.

The theoretical difficulties arising in the development of parallel programs are aggravated by the constant development and complication of computing systems architectures. These possibilities, on the one hand, provide a new potential for accelerating computations, and, on the other hand, arise the problem of utilizing this potential, developing methods and programming tools in the context of these new possibilities.

In addition to general purpose central processor unit (CPU) modern computing systems typically contain additional computing units designed to quick and energy-efficient parallel mass computing operations same for a large amount of data being processed. Examples of such computer units include graphic processor units (GPUs) and Xeon Phi accelerators. To be effective the parallel program must provide all the computer units at its disposal with continuous data loading for calculations. It should also ensure that computing is synchronized where it is necessary when accessing shared data to minimize computer units outages during synchronization and access to other resources both software and hardware. If some computer units process the amount of data allocated to them faster than others and then stand out of action waiting for synchronization there is a need to redistribute the processed data between the computer units while the program is running.

The solution of the problem of data distribution between the computer units is called computational load balancing. In case periodic solution of this problem is needed during program execution it will be called dynamic balancing of the computational load. The effectiveness of the program as a whole depends to a large extent on the successful implementation of this problem.

Research in the field of creating both programming methods for new architectures and the implementation of these methods in language tools for parallel programming is very active, and supported by manufacturers of computer systems. A fairly complete classification of architectures, methods and means of parallel programming is presented on the site [3], which is devoted, in particular, to parallel computing technologies. One approach could be noted of those already implemented. It is based on a perfectly reasonable symbiosis of the parallel compiler and hints from the programmer, made in the form of special software directives, for example [4, 5].

In this case another constructive approach to problems of developing parallel programs is worked out [6]. It determines limits of automatic parallelizing in the particular program and gives the facilities for automatic generation of the effective parallel program. This approach uses the non-procedural NORMA language [7] as a programming language.

This article will propose and consider the method of organizing automatic dynamic computational load balancing within a single node of a hybrid computing system, which has one or more CPUs and one or more additional computer units. The issue of the distribution of computational load between the nodes of the computing system is not considered in this work. The presented method was designed to be used in the NORMA programming system [6] in compiling programs for computer systems with

graphics accelerators (GPU). But this method itself is universal and does not depend on the type of specific hybrid computing system and software used. It seems to the authors it can be successfully applied in the development of parallel programs with computation-intense operations both in manual programming and in the case of automatic approach.

The proposed method was successfully tested on a gas dynamics program with computation-intense operations. It was tested on parallel systems with hybrid architecture (with NVIDIA GPUs).

2 Static computational load distribution method

The issue of the distribution of computational load within the single node of the hybrid computing system is considered. Each of such nodes has one or more central processor units, CPUs. Since all modern CPUs are multi-core and have access to all the RAM of the node, it doesn't matter how many of them are in the node – their entire totality is always considered by the application program and operating system as a single multi-core processor. There is also one or more special computer units in the hybrid computing system (accelerator, GPU or Xeon Phi, or perhaps some other). Their number is already important as each such computer unit has access and can process data only from its own memory.

An effective parallel program should use all available computing power. Accordingly, in the node of a hybrid computing system, the entire amount of computational output must be somehow distributed between the CPU and the accelerators. Calculations done on CPU should be performed using multi-thread programming technologies, such as OpenMP. Calculations performed on the accelerator should be done using technology available for this type of accelerator, such as NVIDIA CUDA for NVIDIA GPU.

The process of automatic static distribution of computation between CPU and GPU when compiling programs written in the NORMA language is detailed in the papers [8, 9]. The methods and the ideas outlined in these articles can be applied to any parallel program and any other type of accelerators. In short, these methods are as follows. In the NORMA language, an operator describing some calculations can be done on the domain. The domain is an analogous to the mathematical concept of the grid. Thus the operator describes a set of identical calculations produced at points (grid nodes) of domain of any type. As a rule calculations at each point of the domain do not depend on the values at other points calculated in the same operator at the same iteration step. Then the calculations done by one operator are independent at each domain point and can be processed in parallel.

To distribute such calculations between CPU and accelerators it is proposed that each such operator is performed on both the CPU and each accelerator available in the system. But the entire domain of such an operator (or rather, the array of points of this domain) is distributed among the computer units and each computer unit performs the operator only for the points distributed to it. To allow the computer unit to perform the operator for its points it is also necessary that the arrays of the variables defined

on these domain points both required for calculations and those that are calculated as a result of the performed operator are also physically distributed between the memory of the CPU and the memory of each accelerator. That is, in fact, the process of computational distribution comes down to the process of data distribution and then to the synchronized performing calculations by each computer unit over the intended part of the common data.

The following approach is proposed for the distribution of such variables. At first, the entire amount of data processed is divided into two unequal parts: the area processed by the CPU and the area processed by the accelerator (accelerators). The size of the areas is chosen according to the expected ratio of the CPU performance and the total performance of the accelerators. Then, if there are several accelerators their areas are divided into the corresponding number of subareas and distributed equally among them. There is an example of such distribution on Fig. 1.

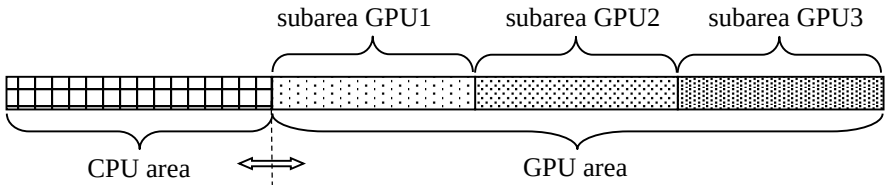

Fig. 1. Distribution of the processed data among the areas.

Each computer unit performs calculations with the data that has been occurred in its area (subarea). Additionally, the problem of data transmission between areas and subareas could be solved if necessary.

3 Dynamic computational load distribution method

The boundary between the CPU area and the accelerator area can be fixed or may be being modified while the program runs. By changing the area boundary periodically the program performs a dynamic balancing of the computational load which will be discussed further. While changing the size of the GPU area the size of each accelerator's subareas is recalculated accordingly.

In order to assess the need to adjust the position of the area boundary the process of estimating the overall efficiency of calculations in the current distribution of the computational load is started periodically (for example, at each *n* step of the iteration). As a result of this process it is decided whether to maintain the current position of the area boundary or to shift it to some value in one direction or another. In case of a shift it is necessary to redistribute the data occurred in the other area or subarea and this data starts to be processed by another computer unit.

4 The method to determine whether you need to adjust the distribution of the load

To determine the overall effectiveness of the calculations it is proposed to use the method based on the evaluation of the program's use of the CPU resource. Developing this method we assume that the program that solves the computational problem should be constantly engaged in computing and completely consume the resource of the computer units. There may be sure synchronization points when individual processes and threads can wait for other processes and threads but the waiting time should be quite short comparing to the computing time. If the program periodically waits for some external events and spends considerable time in stand by, then this method couldn't be applied to such a program.

So, ideally, a computing program should create a 100% CPU load. If the program is hybrid and uses calculations on the accelerator along with the calculations on the CPU, then, if the accelerator processes the data allocated to it more slowly than the CPU its ones, the program will stand by in synchronization points waiting for the accelerator to execute. And, as a result, the CPU load will be less than 100%. It's easy and quick for the program to get the information about its consumption of CPU's resource. In UNIX family OS, for example, it is done by a system call **clock_gettime(...)**. A call with a **CLOCK_REALTIME** parameter gives a total system time and with the **CLOCK_PROCESS_CPUTIME_ID** parameter gives the processor time consumed by the running program. If these two parameters are detected over a period, then the rate of CPU load by this program during this period can be calculated according to the following formula:

$$CPU_{load} = t_{CLOCK\_PROCESS\_CPUTIME\_ID} / N_{thread} / t_{CLOCK\_REALTIME} * 100\%,$$

where t is an appropriate time, and N_{thread} is the number of CPU threads running.

The next issue is how to interpret the resulting CPU load. We resume that ideally a computing program should have a CPU load 100%. But it will also be 100% if the accelerator processes its data faster than the CPU and the program doesn't stand by waiting for the accelerator to terminate. To be able to diagnose such a situation and to allow the program to spend some time in synchronization points with other processes it is suggested that the eligible CPU load is considered an empirical value of 95%. In other words it is allowed that the CPU stands by waiting for the accelerator but for a short period of time no more than 5% of the total CPU's load. If the value of the CPU load is more than that is considered eligible, then it is necessary to reduce its area (and, accordingly, to increase the accelerator area). If, on the contrary, less than eligible – then to increase the CPU area.

But shifting the area boundaries entails starting data redistribution process and it can take considerable time to complete. Therefore, it is highly desirable to avoid the situation of constant changes from decreasing to increasing areas and vice versa. Thus an eligible CPU load is proposed to consider not a specific value but a small range, for example, from 85% to 95%.

If the CPU load has been estimated periodically (for example, at each *n* step of the iteration) for the time interval expired since the previous estimation, one can decide whether to leave the current distribution of the data (if the CPU load is in the eligible range) or increase the CPU area (if the CPU load is below the range) or reduce the CPU area (if the CPU load is above the range). It is also important to determine how much it is necessary to shift the boundary of areas. It is obvious that the more the CPU load rate differs from the needed the more the area boundary should be changed. On the other hand, moving the areas boundary should be careful when the size of one area highly exceeds the size of the other. As even a small boundary shift can significantly alter the amount of data being processed which in turn can fundamentally change the balance of computational load between the CPU and the accelerator. A sudden change in the ratio of areas size may cause the reverse changes in return at the next step. And if the response is also sharp it will cause constant changes which should be avoided as it was explained earlier. Therefore, near the extreme values of the relative size of the areas the algorithm for moving the areas boundary should be carefully implemented to shift the boundary to small extent.

As a result, the algorithm that determines the magnitude of the area boundary can be described by a function of two variables – a deviation from the eligible CPU load and the current relative size of the areas.

The proposed method can be algorithmized and is suitable for a wide range of computational tasks and does not depend on the characteristics of a particular computing system. Therefore, it can be used for automatic balancing of the computational load. It is planned to implement it in the NORMA compiler [10]. The main goal is that the compiler would automatically generate all the necessary code to determine the CPU load, to decide whether to shift the boundary of the areas, and to redistribute the data being processed. In the meantime, the method is "manually" implemented in some gas dynamics solving applied program, and the next chapter gives the results of its application.

5 The results of the proposed method applying

The method of automatic balancing of computational load was implemented in a hybrid applied program that solves gas dynamics problem. The program use MPI technology to engage several nodes of distributed computer system, OpenMP technology for multicore CPU computing within a single node and NVIDIA CUDA technology for GPU computing.

The results below were obtained from K-100 computational cluster [11] using Intel 15.0.0, nvcc compiler version 6.5, and Intel MPI Library 5.0 Update 1. The program starts 4 MPI processes, each runs on its own computational cluster node with 12 CPU cores and 3 GPUs. Tests were conducted with different number of GPUs, from 1 up to 3, and the method worked properly regardless of the number of GPUs. But because this program is well-suited to GPU calculations, the share of CPU calculations was very small. Therefore, further data for starts using only 1 GPU is given so that the CPU's contribution to the overall computation gets more noticeable and the running

processes become more visible. The program also ran on K-60 computational cluster [11] with more powerful GPUs. The method of automatic balancing of the computational load also worked well there, but the share of calculations on the CPU was even less - 4% with only 1 GPU.

The diagram of the program's execution time at different values of the GPU area size is presented on Fig. 2. The first 10 columns correspond to program starts with fixed-boundaries areas and without the automatic balancing of the computational load. In this case the GPU area size is set from 100% (when the CPU is not used at all) to 83%. The last 4 columns are starts using automatic computational load balancing, with different initial GPU area size: 100%, 75%, 50% and 0%.

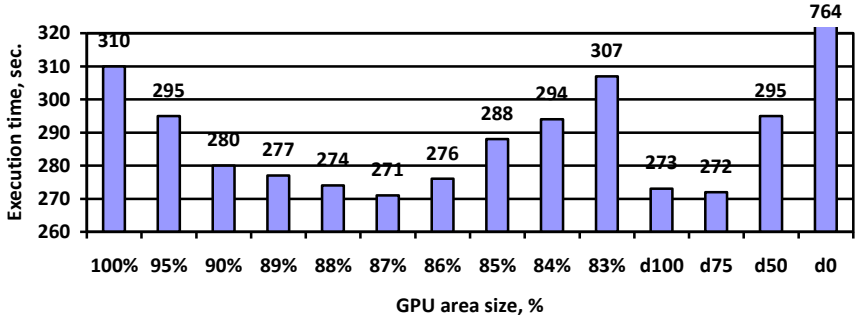

Fig. 2. Program execution time correlated to GPU area size.

The diagram shows that the least execution time is achieved when GPU area size is set to 87% (and 13% CPU area size respectively). Then with small enlargement of CPU area size the program's execution time begins to grow rapidly – in fact, as much as the CPU area size grows, because it is time of the CPU work that begins to determine the operation time of the entire program.

Of particular interest there are columns that correspond to the starts with the use of automatic computational load balancing. They show that if the initial distribution of the computational load has been chosen roughly correct (d100 – the initial GPU area size is 100% and d75 – the initial GPU area size is 75%), then total program's execution time is close to the ideal. But if the initial distribution has not been chosen correctly (d50 and, in particular, d0), the program spends considerable time to get to its proper distribution.

There are graphs of areas size changes corresponding to the iteration step for program starts with automatic computational load balancing on Fig. 3. The values of computational load have been analyzed and adjusted at each 100th step of the iteration.

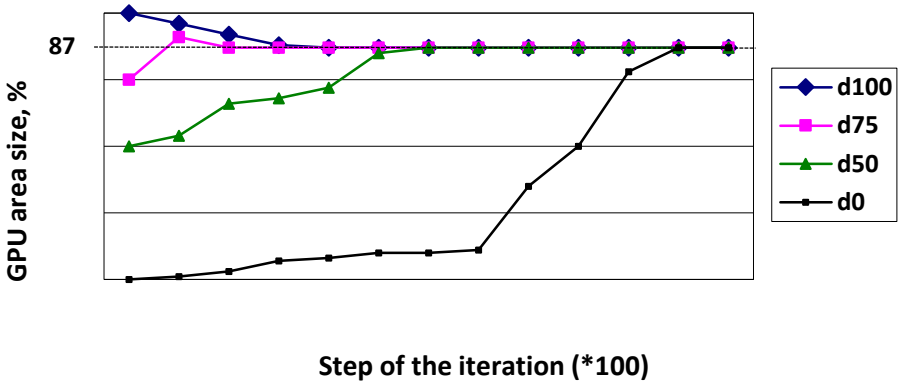

Fig. 3. Changing the GPU area size corresponding to the iteration step.

The diagram shows that d100 and d75 starts have already got to the ideal distribution at the 4th adjustment. The ideal distribution for the given program on the set hardware could be seen on Fig. 2. and considered 87% GPU area size. D50 start has already spent more steps on adjustment and as a result the total time of its execution is noticeably longer. D0 start has been moving cautiously away from zero size of the GPU area for a very long time, the average values of areas distribution have been being already passed much faster and finally it has also got to the same ideal distribution, 87% for the GPU area size. But it has taken 150 adjustments and considerable time has been wasted.

6 Conclusion

The proposed method of automatic computational load balancing, despite of its simplicity, can be successfully used when solve computation-intense problems on hybrid computing systems. Tests have shown that the described method implementation gives the program its ideal distribution of the computational load and in the case of a small change in the load the method gives the opportunity to cope with such changes quickly. This method is independent neither from the hardware of the hybrid computing system nor from the software chosen for solving the applied problem.

References

1. Voevodin, V.V.: Matematicheskie modeli i metody v parallelnykh protsessakh. Nauka, Moscow (1986).
2. Voevodin, V.V., Voevodin, Vl.V.: Parallelnye vychisleniia. BKhV-Peterburg, S. Peterburg (2002).
3. Informational Analytical Center, http://parallel.ru/index_eng.html, last accessed 2020/11/25.

4. OpenACC, http://openacc.org, last accessed 2020/11/25.
5. DVM-system, http://www.keldysh.ru/dvm, last accessed 2020/11/25.
6. Sistema NORMA, http://www.keldysh.ru/pages/norma, last accessed 2020/11/25.
7. Andrianov, A.N., Baranova, T.P., Bugerya, A.B., Gladkova, E.N., Efimkin, K.N.: Iazyk NORMA. Preprinty IPM im. M.V.Keldysha (2019), ISSN 2071-2898 (Print), ISSN 2071-2901 (Online), No 132, 48 p., doi:10.20948/prepr-2019-132., http://library.keldysh.ru/preprint.asp?id=2019-132, last accessed 2020/11/25.
8. Andrianov, A.N., Baranova, T.P., Bugerya, A.B., Efimkin, K.N.: Raspredelenie vychislenii v gibridnykh vychislitelnykh sistemakh pri transliatsii programm na iazyke NORMA. Vychislitelnye metody i programmirovanie (2019), ISSN 1726-3522, M.: NIVTs MGU im. M.V. Lomonosova, Vol. 20, № 3, P. 224–236, DOI: 10.26089/NumMet.v20r321, http://num-meth.srcc.msu.ru/zhurnal/tom_2019/pdf/v20r321.pdf, last accessed 2020/11/25.
9. Andrianov, A.N., Baranova, T.P., Bugerya, A.B., Efimkin, K.N.: Metody raspredeleniia vychislenii pri avtomaticheskom rasparallelivanii neprotsedurnykh spetsifikatsii. Superkompiuternye dni v Rossii: Trudy mezhdunarodnoi konferentsii. 23–24 sentiabria 2019 g., g. Moskva. Pod. red. Vl.V. Voevodina. M.: MAKS Press (2019), ISBN 978-5-317-06007-7, e-ISBN 978-5-317-06244-6, P. 59–70, DOI: 10.29003/m680.RussianSCDays, URL: http://russianscdays.org/files/2019/pdf/59.pdf, last accessed 2020/11/25.
10. Andrianov, A.N., Bugerya, A.B., Efimkin, K.N, Koludarov, P.I.: Modulnaia arkhitektura kompiliatora iazyka Norma+. M.: Preprint IPM im. M.V. Keldysha RAN (2011), No 64, 16 p., http://keldysh.ru/papers/2011/prep64/prep2011_64.pdf, last accessed 2020/11/25.
11. Tsentr kollektivnogo polzovaniia IPM im. M.V. Keldysha RAN, http://ckp.kiam.ru/?hard, last accessed 2020/11/25.