Selection algorithm of graphic accelerators in heterogeneous cluster for optimization computing

A. M. Lavreniuk, M. S. Lavreniuk

The paper highlights the question of the optimal GPU computers selection for kernels in OpenCL when they are starting on heterogeneous clusters where different types of GPU are used. The authors propose optimal GPU selection algorithm that helps to get the best efficiency while program execution using GPU.

Introduction

The accelerators, especially graphic (GPU) are commonly used while constructing large and small clusters. A lot of such clusters already are available as grid nodes in Grid [1]. To increase the productivity of calculations on these clusters and Grid it is recommended to use many GPU that are available on the node simultaneously, for task execution [2]. Accordingly, the situation may occur when a program is executed in GPU of different architecture and different generations [3].

We can define two leaders in the production of graphic cards that support parallel computing. They are NVIDIA, AMD and have some differences. For example, vector operations are not supported in the GPU NVIDIA but are supported in the AMD, besides at different GPU the lengths of certain types vectors are different.

Kernel written in OpenCL [4] works with the different productivity on GPU with diverse architecture and different generations [2]. Even GPU computers that where produced with one manufacturer but belong to different generations have various productivity with almost identical parameters. This is confirmed by an experiments that have been carried out. When the execution time of the program in very short (up to several minutes), then this problem is not significant. But if the program runs for many hours or even days, for example, while solving the problem of rapid synthesis of 3D seismograms in 2.5D model [5], then the non-optimal choice of GPU for kernels execution in OpenCL on cluster total execution time can increase almost in 1.5 times. It can be several hours or even several days depending on the problem.

The problem of OpenCL kernels distribution between several GPU

Within software development SDK (software development kit) from the company's products GPU NVIDIA proposed next mechanism for GPU selection as the C++ function:

```
cl_device_id oclGetMaxFlopsDev(cl_context cxGPUContext)
{...
     max_flops = compute_units * clock_frequency;
...}
```

So the maximum GPU productivity is equal to the product of the number of parallel compute units by maximum clock frequency of the device:

$$max\_flops = compute\_units * clock\_frequency, \tag{1}$$

where $compute\_units$ - number of parallel compute units, $clock\_frequency$ - maximum clock frequency. Below in table 1 the main GPU parameters are presented for different manufacturers and different generations that have been software derived (function oclGetMaxFlopsDev).

As seen from the table, the value $clock\_frequency$ for ATI RV770 is equal 0, and, consequently, to the formula (1) $max\_flops = 0$, which is contrary to reality.

Table 1. Main GPU parameters

Parameter	Value			
CL_DEVICE_NAME	Tesla M2050	GeForce GTX 260	Cayman	ATI RV770
CL_DEVICE_MAX_COMPUTE_UNITS	14	27	24	10
CL_DEVICE_MAX_ CLOCK_FREQUENCY	1147 MHz	1242 MHz	830 MHz	0 MHz

We must note that the parallel compute units consist can contain different number of unified processors depending on GPU generation and as it is shown at fig. 1 and 2, GPU Tesla M2050 is faster than GeForce GTX 260 especially with a large data amounts. However, according to table,

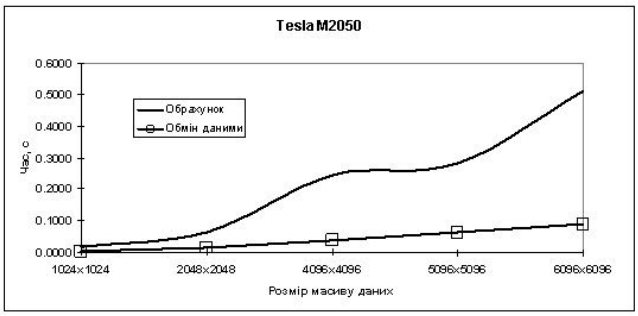

Figure 1. Dependence of computation time and time data exchange on the amount of data for GPU NVIDIA Tesla M2050.

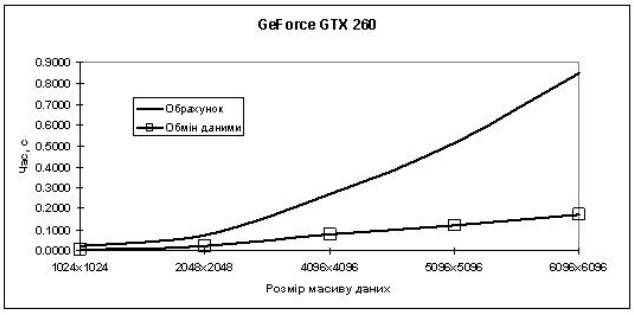

Figure 2. Dependence of computation time and time data exchange on the amount of data for GPU NVIDIA GeForce 260.

compute_units Tesla M2050 is less than GeForce GTX 260 and *clock_frequency* for Tesla M2050¡ GeForce GTX 260. And we can see the contradiction again. The same situation is with GPU Cayman, which is the part of last generation AMD Radeon HD 6990 - *compute_units* is less that GeForce GTX 260. So, equation (1) will provide us with incorrect result for the optimal GPU selection for task execution on cluster with GPU.

Optimization of kernel's distribution in OpenCL for few GPU

We suggest an approach for the optimal GPU selection, which is based on a set of computational tests of different complexity which are necessary to execute on GPU. These tests contain the basic operations used in the program. Operating time of the test should be substantially less than the operating time of the entire program, for example, no more than $0.01 * T_p$, where T_p - approximate operating time of the entire program (the control program and the kernels). The algorithm is as follows:

- check the types of available GPU with CL_DEVICE_NAME;
- if CL_DEVICE_NAME in all GPU are the same, then we choose necessary amount of computing devises and start kernels on them;
- if CL_DEVICE_NAME are different, then we make a test and calculate the value of isCalc using formulas (2) and (3);
- if $isCalc = true$, then we use chosen GPU, otherwise we check for the other GPU computing devise.

Using suggested formula 2, we determine whether it is necessary to put on particular GPU kernels in OpenCL.

$$isCalc = \begin{cases} true, & \text{if } P_i <= c * max(P_{1..n}); \\ false, & \text{if } P_i > c * max(P_{1..n}). \end{cases} \quad (2)$$

where P_i - total time of making test on GPU i computing devise, c - coefficient, which can take it's values from 0..1, it depends on the complexity of the task, optimally $0.8, i = 1..n, n$ - the amount of GPU computing devises.

Using formula (3), P_i - can be presented as the sum of operating time of kernel and time spent on exchange operations of big data amounts between main program and kernel.

$$P_i = k_1 * T_i + k_2 * T_i' \tag{3}$$

where T_i - execution time of one iteration of test on GPU i computing devise, k_1- the amount of iterations in one test, T_i'- data exchange time with GPU i computing devise for test, k_2 - the amount of data exchange operations between control program and kernel in one test. In our experiments $k_1 = 100, k_2 = 2$. The results are shown on fig. 1 and 2.

At first sight, the data exchange time is negligible, if before the computing data is loaded on GPU, and after the end of long calculations results are loaded into RAM. However, when during a long time of calculations extra data are downloaded in few iterations to GPU or reading of intermediate results in GPU, then the data exchange time between PC and GPU becomes an important indicator, that significantly affects the overall computation time.

Conclusions

When you run tasks on heterogeneous GPU clusters, architecture of which you do not know, as it often happens in the case of calculations in Grid, the proposed approach enables:

- to conduct rapid testing of productivity of GPU;
- optimally select GPU computing devises, excluding not productive enough for a particular problem;
- to execute the task in the optimum time.

The future work will be focused on structuring the tests according to their complexity and operating time. Find right minimal sufficient set of tests especially with regard for all types of different GPU architectures and generations. We plan to make the following tests in the form of program library with subsequent placement on clusters of GPU. This will make their practical use more simple and effective for clusters and grid nodes with GPU calculations optimization.

References

[1] Lizandro Solano-Quinde, Zhi Jian Wang, Brett Bode, and Arun K. Somani. Unstructured grid applications on GPU: performance analysis and improvement —In Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-4). ACM, New York, NY, USA, Article 13, 8 pages

[2] Chris Jang. OpenCL™ Optimization Case Study: GATLAS - Designing Kernels with Auto-Tuning, http://golem5.org/gatlas/CaseStudyGATLAS.htm

[3] Marcus Hinders. GPU Computations in Heterogeneous Grid Environments, Joint Research Report, http://www.techila.fi/technology/technology-docs/

[4] OpenCL - The open standard for parallel programming of heterogeneous systems, http://www.khronos.org/opencl/

[5] Marmalevski N.Ya, Merschyy V.V, Roganov Yu.V, Tulchinsky V.G, Yushchenko R.A CUDA Application for rapid synthesis of 3D seismograms in 2.5D model —Calculations in geology, Moscow-2011, N3, pages. 8-12.

Authors

Alla Mykolaivna Lavreniuk — PhD, Associate Professor, National Technical University of Ukraine "Kyiv Polytechnical Institute", Institute of Physics and Technology, Kiev, Ukraine; E-mail: *alla.lavrenyuk@gmail.com*

Mykola Serhiiovych Lavreniuk — the 3d year bachelor, faculty of cybernetics, Taras Shevchenko national university of Kiev, Kiev, Ukraine; E-mail: *nick_93@ukr.net*