The methodology of application development for hybrid architectures

Alexander Bogdanov
Saint-Petersburg State Electrotechnical University "LETI"
Saint-Petersburg, Russia
e-mail: bogdanov@csa.ru

Vladimir Orekhov
Saint-Petersburg State Electrotechnical University "LETI"
Saint-Petersburg, Russia
e-mail: orekhov.volodya@gmail.com

Vladimir Gaiduchok
Saint-Petersburg State Electrotechnical University "LETI"
Saint-Petersburg, Russia
e-mail: gvladimiru@gmail.com

ABSTRACT
This paper provides an overview of the main recommendations and approaches of the methodology of parallel computation application development for hybrid structures.

Keywords
Hybrid architectures, parallel computation, methodology

1. INTRODUCTION
Avoiding all the problems during the research and development is virtually impossible, that is why one of the main goals of modern application development methodologies is to detect potential problems as early, as possible, and to eliminate them.

There are several great methodologies concerning the application and product development in general [1,2]; however, the application development for hybrid architectures is a very specific branch and has its own tricky problems. The following methodology was developed within the master's thesis project "Optimization of complex tasks' computation on hybrid distributed computational structures" accomplished by V. Orekhov. This methodology both covers the well-known problems of applications based on parallel computations and describes the process of its development in details, offering easy ways of avoiding potential crucial problems.

2. PRELIMINARY ALGORITHM ANALYSIS AND TASK SIMULATION
Before the actual development of an application based on parallel computing, developers should go through several steps, which are illustrated in figure 1.

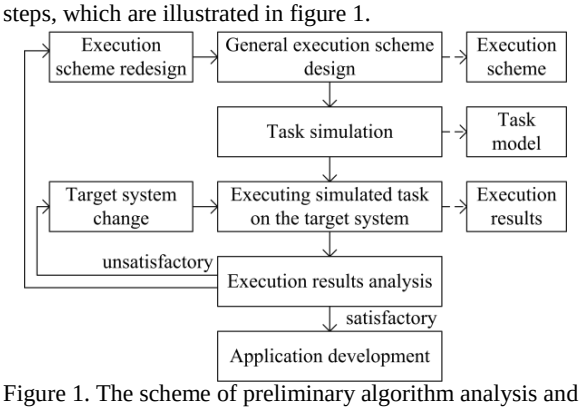

Figure 1. The scheme of preliminary algorithm analysis and task simulation

2.1. Execution scheme design
This step includes the design of a general execution scheme, which will on a high level of abstraction show the main elements: blocks of code, which will be executed in parallel, blocks of code, which will be executed in series and data transfers between the host and computational devices.

An example of an execution scheme for the K-Means clustering algorithm is represented in figure 2.

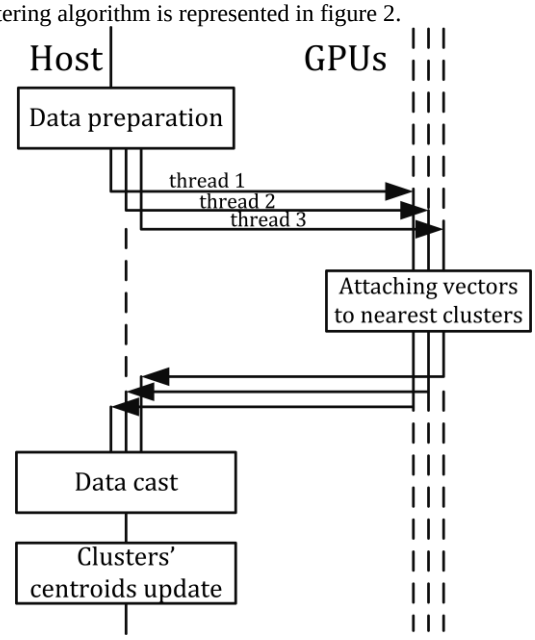

Figure 2. Sample execution scheme for the K-Means clustering algorithm

The process of the execution plan design is iterative, on each iteration developers should try to maximize two main parameters:

- parallelization coefficient;
- amount of stand-alone computations (meaning the amount of computations, during which the "host – computational devices" channel is not used to transfer data).

Although these two parameters seem to mean the same thing, it is very important to realize the difference between them.

The parallelization coefficient shows the general amount of computations, which are executed in parallel:

$$_c = \frac{C_{parallel}}{C_{overall}},$$

where $C_{parallel}$ – the amount of computations, which are executed in parallel;

$C_{overall}$ – the overall amount of computations (both parallel and serial).

The amount of stand-alone computations, however, is a more specific parameter, and basically means the amount of computations, during execution of which the "host – computational devices" channel is not used to transfer data. This parameter takes into consideration the data transfer costs and maximization of this parameter means maximization of task execution solely on computational devices.

2.2. Task simulation

This step includes the design of a task on the basis of the designed execution scheme, which will simulate the behavior of the target task.

As far as at this step we focus on time complexity, the actual implementation of the algorithm's steps is not important; what is important is to simulate the same amount of computations via something basic – like simple arithmetic operations

In this simulated task all the computations should be replaced with the equivalent amount of simple operations. That will allow achieving a model with the same computational difficulty as in the target task, extremely cutting the development time.

2.3. Executing the simulated task

At this step the simulated task should be executed on the target computational system. After that it is time to analyze the execution results. The results can include such base parameters, as:

- Execution time;
- Execution speed in iterations per second, if the number of iterations is not constant (as it is for the clustering algorithms).

These base parameters show the performance of a particular task, executed on a particular computational system. At the same time, it appears that knowing simple fact like "*X executes faster, than Y*" is not enough – we need a more detailed comparison of X and Y to organize a proper analysis, for that purpose derivative parameters can be used:

- Scaling efficiency

$$SE_{16-32} = \frac{Sp_{32}}{Sp_{16}},$$

  where SE – scaling efficiency,
  Sp_i – execution speed or other base parameter for i computational devices.
- Coefficient showing the proximity to the ideal scaling efficiency.

The scaling efficiency chart for the K-Means clustering algorithm can illustrate the meaning of the scaling efficiency parameter and is represented in figure 3.

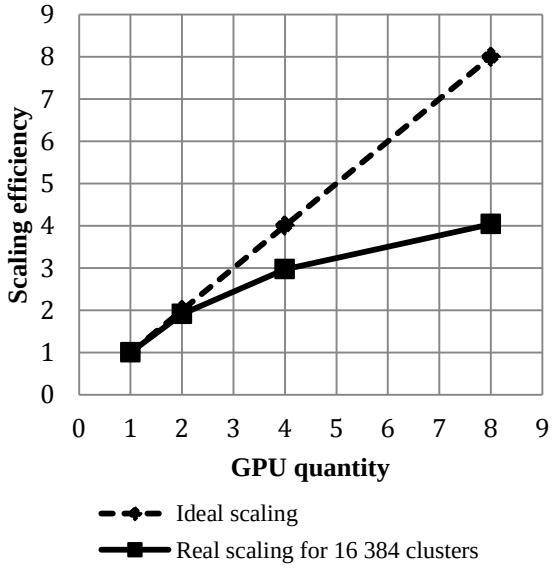

Figure 3. Scaling efficiency analysis

If the results are satisfactory, the execution scheme is proven to be appropriate and the application should be developed based on it.

If the results are not satisfactory, either the execution scheme should be redesigned or the target computational system should be changed.

Quite frequently it is impossible to execute the simulated task on the target computational system just because it doesn't exist – for example, all parts for the hybrid cluster are bought but yet not set to work. In that case the simulated task should be executed on the closest available computational system (here "closest" corresponds to system's structure and size).

3. DETERMENATION OF SCALING EFFICIENCY

When we speak about effective scaling in terms of hybrid clusters, one of the main trends is the following one:

Scaling can be efficient up to some critical number of computational devices, and then it becomes inefficient.

This number is a crucially important constant as it describes the relations between the specific task (usually an implementation of some algorithm) and the specific computational system. However, it can take a lot of efforts to detect this number and it turns out that in most cases we don't need it.

All we need to know is:

- The upper bound for the computational devices number, which are supposed to be used for this task;
- Whether this number is smaller, than the critical number, after which we lose scaling efficiency, or not.

It is important to know these things as early, as possible, to avoid the problems of inefficient scaling.

Let's assume that the upper bound of scaling is known in the very beginning of the development process. Then the goal is to determine whether or not the scaling is effective for this number of computational devices, for a particular algorithm's implementation and on a particular computational system.

In these terms, if scaling for an exact upper bound of computational devices is not effective (see Case 2), either the execution scheme should be redesigned or the target computational system should be changed, and this approach absolutely fits the scheme, illustrated in figure 1.

The deeper understanding of scaling efficiency can be obtained by simulating and testing tasks on the target systems and building scaling efficiency charts as shown in figure 3. This chart basically shows how much of the hardware potential is used, when scaling on more computational devices. For instance, figure 3 shows nearly 100% scaling efficiency on 2 GPUs and only about 50% scaling efficiency on 8 GPUs. It is important to understand the bottom bound of scaling efficiency, after which we don't want to continue the scaling process (it might be, for instance, 20%).

Several real-life cases will help to better understand the subject:

- *Case 1*: describes the necessity of knowing the upper bound of scaling before the beginning of the development process.
  Let's say the application was simulated, developed and tested for an upper bound of 20 computational devices. After a while, the target hybrid cluster grows wider and gets extra 10 computational devices, so the application executes on a total of 30 computational devices. In worst-case scenario, on 30 devices it will execute slower, than on 20 devices, which is, obviously, an extremely inefficient use of computational capabilities.
- *Case 2*: describes how the scaling efficiency problem can be detected and eliminated in the early stage.

Let's assume that the upper bound is known to be 40 devices but the results of an iterative execution have clearly shown, that the task mapped on 16 devices executes faster, than the task mapped on 32 devices. That means that this combination of an algorithm's implementation and a target computational system are not able to provide an effective scalability up to 40 devices.

4. OPTIMAL TARGET SYSTEM CHOICE

An important question during the parallel application development process is the choice of a target computational system (meaning hardware platform).

The first thing to think about is whether the application can be scaled efficiently or not. If it can be scaled efficiently, the target computational platform should be a hybrid computational system, containing several computational devices. Both scientists and developers often set quite ambitious goals implying efficient scaling on up to hundreds of computational devices. In this case, poor scalability is fatal.

At the same time, if the goal is not that ambitious (for instance, achieving a 40x speedup), it is good to remember the following trend:

Inefficient scaling doesn't mean, that the task can't be effectively executed in parallel.

If the task can't be scaled efficiently, it always can be executed on one computational device. Nowadays practically every PC is a simplest hybrid system with a CPU representing host device and GPU representing computational device and can cope with this task.

When we speak about parallel computations on hybrid systems, it is very important to choose between AMD OpenCL and NVidia CUDA technologies, and this choice affects the target computational system choice heavily.

Here one of the paradigms should be chosen. OpenCL is a free standard, supported by many hardware developers, including AMD, Intel and NVidia; the basic idea behind OpenCL is to create a unified framework for application development on hybrid structures. OpenCL hides all the implementation details and represents complex hybrid systems as a single system with a number of computational devices.

CUDA is focused on the massively parallel computing around NVidia GPUs solely; as they choose such a narrow area, obviously, CUDA shows a better performance on NVidia GPUs comparing to OpenCL.

If NVidia CUDA is used as the dominant technology, there are no other options than using a GPU cluster, which includes numerous NVidia GPUs. If we choose OpenCL, practically every hybrid system can be a target system.

5. CONCLUSION

The described methodology covers the whole development process of a parallel application including preliminary algorithm analysis, task simulation, scalability questions, dominant technology and target system choice.

The methodology is based on the in-depth researches of algorithms' implementation for hybrid distributed structures, one of the center aspects of which was scaling efficiency.

All the main principles are covered by real-life examples.

6. ACKNOWLEDGEMENT

Work partially supported by the Russian Foundation for Basic Research, project no. 13-07-00747-a and St.Petersburg State University, project no. 9.38.674.2013.

REFERENCES

[1] Martin, R. C. "Agile Software Development, Principles, Patterns, and Practices". Prentice Hall/Pearson Education, 2003.

[2] Ries, E. "The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses". Crown Business, 2011.