Marcin CEGIELSKI

Technical University of Lodz

doi:10.15199/48.2016.07.16

Parallel computation of transient processes on OpenCL framework

***Abstract**. Parallel execution of calculation of transient analysis is based on a split-level model into sub-systems, which in certain time increments are calculated independently of each other. Each such process has a high computational complexity. The process of implementing the calculation allows the use of parallel systems to calculations based on the use of the GPU, whose dynamic growth has been observed for several years. The article presents a brief description of parallel computing systems based on the OpenCL platform that uses GPUs. There is described the ability to implement the algorithm using this platform. There is also discussed, the timing to perform operations on GPU in relation to the calculations for classic CPU.*

***Streszczenie.** Równoległa realizacja obliczeń analizy stanów przejściowych bazuje na podziale na poziomie modelu na pod-układy, które w określonych krokach czasowych obliczane są niezależnie od siebie. Każdy taki proces charakteryzuje się dużą złożonością obliczeniową. Proces realizacji obliczeń pozwala na zastosowanie do obliczeń systemów równoległych opartych o wykorzystanie GPU, których dynamiczny rozwój jest obserwowany od kilku lat. W artykule przedstawiono krótką charakterystykę równoległych systemów obliczeniowych opartych o platformę OpenCL wykorzystującą procesory GPU. Opisano możliwość implementacji algorytmu z wykorzystaniem tej platformy. Omówiono zależności czasowe realizacji obliczeń na procesorach graficznych w stosunku do obliczeń na klasycznych CPU. (**Obliczenia równoległe na platformie OpenCL w analizie stanów przejściowych**).*

Keywords: parallel computation, transient processes, GPU, diakoptic.
Słowa kluczowe: równoległe obliczenia, procesy przejściowe, procesory graficzne, diakoptyka.

Introduction

Computation of complex systems is always connected with necessity of making a large number of step by step toilsome calculations. Firstly, it causes a huge waste of time, secondly, the possibility that mistakes and inaccuracies will appear is increasing very fast. Moreover, existing novel computer technologies do not eliminate factors mentioned above. Therefore, it is necessary to look for possible variants of changing a particular serial complicated process of computation for a few independent simplest procedures of calculation, which would enable to get the same solution in several computational stages. Mathematical theory which is needed to get this attitude is known and was created and systematized in Kron's work. [1].

Principles of diakoptic method

Principles of parallelization technique of diakoptic method was shown in [2]. The main idea of the diakoptic methods is partitioning of scheme into sub-schemes. The new-created mathematical model is shown in equations (1) i (2):

$$(1)\qquad F_i(\vec{X}_i, \vec{V}) = 0; \qquad i = 1, 2, ..., N$$

$$(2)\qquad G(\vec{X}, \vec{V}) = 0$$

where X_i is a vector of of internal variables of the i-th circuit; $X = X_1, X_2, ..., X_n)^T$; V, Y are vectors of additional input and output variables; $F_i()$ is a set of differential equations that describe dynamic processes in the i-th sub-circuits; $G()$ is a set of algebraic equations that represents the separate sub-schemes interaction.

The essence of the diakoptic approach is an independent solution (numerical integration) of each equation (1), that represents i-th sub-schema, and sequential reconciliation of obtained results by solving equations (2) . It is possible to solve equations (1) both in particular time intervals and in separate iterations. The integration process of these equations in time domain is known as waveform relaxation method and as the practice has shown is the most effective [9] [10] [11].

This approach involves the procedure of reconciliation of external variables between successive steps of computations. Since transient values of external variables come from the previous results of the step of computations of the mathematical model, we must use numerical methods of integrations. Explicit methods are very simple and what is more they require much less time for calculations although the accuracy of the results obtained with their help strongly depends on the choice of the step which should be small enough. Incorrect selection of the interval reconciliation of external variables can cause instability calculations of the whole process. The main important element of computations is the problem of stability estimation. Improving the stability of the computation it is achieved by minimizing the error between the corresponding coefficients of the divided parts of the model.

Diakoptic approach is used in the analysis of large dynamic models. Particularly in areas such as electromagnetism, electronics or large power grids [13].

Open CL framework

OpenCL™ is the first free open standard of parallel programming for different vendors' platform of modern processors which can be found in personal computers i servers equipped with GPU systems. OpenCL (Open Computing Language) significantly improves application's speed and its adaptation ability to different hardware for a wide range of applications in numerous market categories such as gaming and entertainment or scientific and medical programs. One code can be executed as well on the CPU as GPU, DSP, FPGA and other equipment.

Creating applications for heterogeneous parallel processing platforms using traditional approaches to programming CPU and multi-GPU can be difficult. In addition, there are many methods that are very different from each other. Parallel programming models working on CPU processors are usually based on standards, but they assume a shared address space and do not include vectors operations. Parallel programming models designed for GPU must take into consideration complex hierarchical systems of memory and vector operations, but they traditionally are specific for a given system platform, which is provided by the hardware vendor. These restrictions make it difficult for developers to gain access to computing power of various types of general-purpose graphic processors and other types of processors from one common source code base.

The basic idea of OpenCL is based on using a hierarchical model that includes: Platform Model, Memory Model, Execution Model and Programming Model.

Platform model contains one host and one or more computing devices, each of them with one or more computational units consisting of one or more elements of data processing. OpenCL application is carried out as a combination of host code and code of device kernel. Some part of host code OpenCL application is running on the host processor in accordance with its native model of given host's platform. OpenCL application’s host code is executed as commands from the host to the OpenCL device. The device executes computing commands within devices.

An OpenCL application is implemented as both host code and device kernel code. The host code portion of an OpenCL application runs on a host processor according to the models native to the host platform. The OpenCL application host code submits the kernel code as commands from the host to OpenCL devices. An OpenCL device executes the command’s computation on the processing elements within the device.

Developers provides programs as a form of *C++ OpenCL* source text or binary source files IL appropriate for the environment, on which given platform OpenCL is running. OpenCL platform provides a compiler that translates the input program with both figures into executable program objects. The device code compiler can be used online or offline. Online compiler is available during the execution host's program using standard interfaces API. The compiler in offline mode is invoked beyond the control of the host program, using methods specific to a given platform. OpenCL runtime allows developers to load a previously compiled executable program to the device and to run it. [6]

The framework contains the following components:

1. **OpenCL Platform layer**: The platform layer allows the host program to discover OpenCL devices and their capabilities and to create contexts.
2. **OpenCL Runtime**: The runtime allows the host program to manipulate contexts once they have been created.
3. **OpenCL Compiler**: The OpenCL compiler creates program executables that contain OpenCL kernels. The OpenCL C programming language implemented by the compiler supports a subset of the ISO C99 language with extensions for parallelism.

GPU system

One of the new methods of parallel computations is the usage of GPU processors. A GPU consists of a set of cores with a hierarchical shared memory. Differences between SMP parallel systems (*Symmetric Multi-Processor)* and GPU systems are clear.

Firstly, the number of the computational cores. For example GPU system is connected by devices. One device consists of 256 graphical cores. Multi-Core system have 4 or 8 independent threads units.

Secondly, GPU works in batch mode. Computation process needs extra time for loading and saving data between system memory and GPU system memory.

Thirdly, the management of SMP system is easier than GPU system.

CPU and GPU processors in the computational system can be used in the same time.

A GPU is a massively parallel SPMD (single program, multiple data) processor. It is generally embedded in a graphics card (the device) connected to a computer (the host) via a PCI bus interface. The device also embeds local memory, available for graphics or GPU use. The device memory contains the GPU program code, and all the data that the GPU programs manipulate. The host can allocate blocks in the device memory, transfer data blocks from the host memory to the device memory and vice versa, as well as execute kernels (GPU programs) to manipulate the data in the device memory.

GPU systems are oriented on parallel computation with single precision floats. They can also use double with less speed of computation.

In the next paragraphs elements of operating system of GPU system are discussed.

Principles of parallel computation program

The parallel algorithm of computing transient processes in electrical circuits or in other dynamic systems according to diakoptic approach can be shown with block scheme in figure 1 and [3].

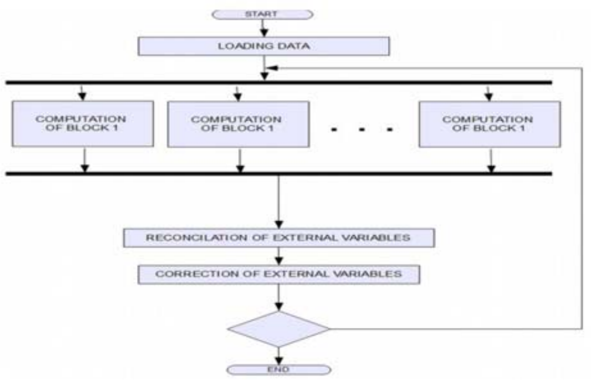

Fig. 1. Schema of diakoptic computations of dynamic system

As can be noticed, the parallelism of the process of the particular algorithm is well specified while the loss for making the computation of the single sub-block is much bigger than time which is needed for computing and negotiating external variables. It is important to notice a certain asynchrony of the described process, which lies in the fact that the external variables are calculated with a time lag in relation to the internal variables.

The main feature of the diakopitc's approach is to divide variables of set of equations into two parts, one of which used an explicit numerical integration method, for which the improper selection integration step can be a source of instability in the calculation.

Among these disadvantages the main one is the potential instability, which can be controlled on a stage of external variables reconciliation [9].

Nowadays there are known methods that allow significantly improving the stability of the calculations and also allows to predict potential instability[7].

We assume that the number of GPU devices is less than the number of separated computation tasks. This implies the need for queuing different tasks and their dynamic separation between the GPU's. After calculating all the tasks the procedure for reconciliation the variables is performed (2), which is performed as a single process. When the reconciliation is done the system allocates the next task with the new step.

The key element of the system which affects the calculation time is the optimum allocation of tasks (1) between the GPU devices and computational complexity of tasks.

Distribution of tasks in computation system

Suppose that we have any but a fixed number of independent computation tasks which has constant number of internal variables.

Diakoptic division of the model into parts allows to split external and internal variables in sub-models, but the number of variables (the size of sub-models) can be different. It comes from the assumption that the division is made not on the level of mathematical model but on the level of model scheme.

For the optimal tasks allocation the knowledge about the computation power and number of GPU devices is needed, It must be known before starting of computation. The model should be divided in equal parts with possibly the same number of variables.

Schema of computational system

System of the threads management is based on the barrier.
The main program controls the whole process, allocates the tasks, stores and connects the results from the particular steps of computation and makes the changes of the computation parameters before the next step.
The course of the whole process can be shown by means of the scheme from figures 2.

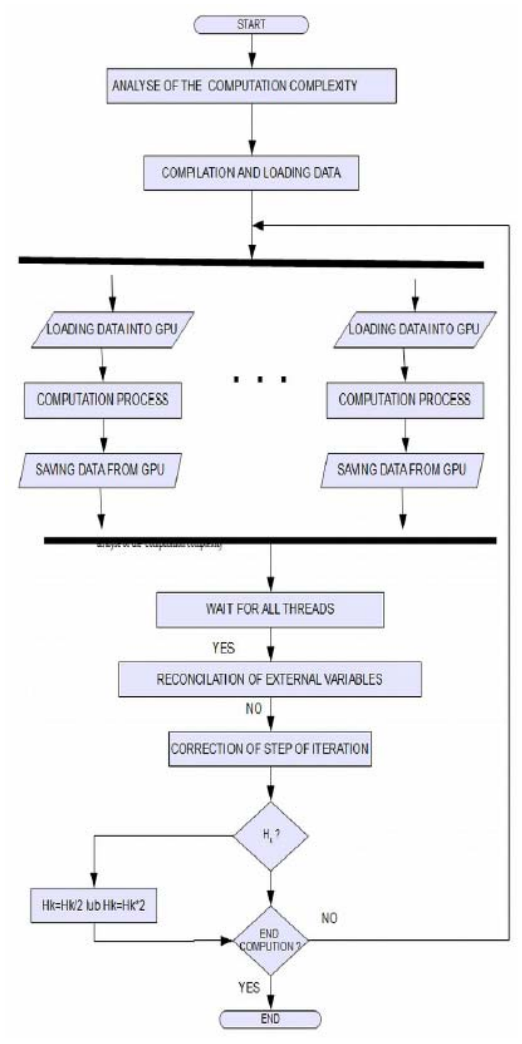

Fig. 2. Diagram of GPU computational system

The beginning of the computation is an analyse of the computation complexity, quantity of tasks and available GPU hosts and devices. The next step is the compilation of the code of the tasks and preparing of data buffers in the following order:

1. Query host for OpenCL devices
2. Create a context to associate OpenCL device
3. Create programs for execution on one or more associated devices
4. Select kernels to execute from the programs
5. Create memory objects accessible from the host and/or the device
6. Copy memory data to the device as needed
7. Provide kernels to command queue for execution
8. Copy results from the device to the host

Step from 6 to 8 are processed in GPU hosts.
The main program waits for the end of the all tasks (GPU hosts). After receiving the solution the phase of the external variables, reconciliation is performed and also the setting of the new integration parameters, which are the base for the beginning of next tasks realization. Strategy of computation of a new step of integration was presented in [8].

Process of computation

The main idea of the diakoptic method is a division onto parallel tasks, which can be running independently. The computation process is asynchronous and the expense of a particular computation task can be diverse. Therefore full exploitation of the computation power can be impossible. The estimation of the computation speed can be complicated by the necessity of reconciliation of the external sub-circuit variables values and hard to estimate quantity of the repetition of computation steps which has to be repeated. The value of the repetition can be estimated according to carried out computation experiments in which the proper criteria of the estimation of the stability were applied. [4] [5]. Using of the proper criteria allows to accept the estimated increasing integration step value. The time of the calculation of the i-th task is defined by formula (3).

(3) $$t_i(n) = h_i \cdot t_{int}\left(n\right) + t_{comp} + t_{load} + t_{save}$$

where h mean the average step of the integration of the i-th sub-circuit, t_{int} the time of the computation (integration) of one step, dependent on the size of differential equations i-th sub-circuit, t_{comp} time of loading and preparing program to running, t_{load} and t_{save} time of the transfer of data to and from the GPU host.
The time of computation of the main step with alignment of external variables (2) is given by (4)

(4) $$t_s = max(t_i(n)) + t_{recon}$$

where $max(t_i(n))$ maximum computation time of equation (3), t_{recon} time of reconciliation and correction of external variables.

The total time of computations is defined by formula (5)

(5) $$t_{TCO} = \sum_{s=1}^{N} t_s + t_{init} + t_{close}$$

where t_{init} and t_{close} times initialization and closing of *OpenCL* environment (t_{init}).
You will notice that there is a correlation between the amount of the main steps and the time of calculations tasks in one step. Increasing the pace of integration accelerates the calculations, but can potentially lead to instability calculations after several stages of approval.
The criteria of stability [5] can be used in the same time then computations of the sub-models. Paired external variables can be tested in parallel with the process of their calculations in tasks. We can decide the quantity of iteration step independent in all tasks. This means that you can speed up calculations by changing the integration step only in certain sub-systems, and not all.

Times in formula 3 was estimated in computational experiments. We can notice, time of building GPU program is bigger then time of transfer of data between host operational memory and GPU internal memory. It is recommended to use the same compiled program in all steps of computation equations (2). The method of correction of external variables can be computed by resources of host computer.
The time of the calculation of the i-th task is reduced to (6).

(6) $$t_i(n) = h_i \cdot t_{int}\left(n\right) + t_{load} + t_{save}$$

Times of initialization and closing OpenCL framework are small according time $t_i(n)$. We can assume, $t_{init} \approx 0$ and $t_{close} \approx 0$. Simplify the total time of computation is given by formula (7).

(7) $$t_{TCO} = \sum_{s=1}^{N} t_s(n) + t_{comp}$$

where t_{comp} is the maximum time of compilation of all OpenCL programs to binary source files IL, used in computations (6).

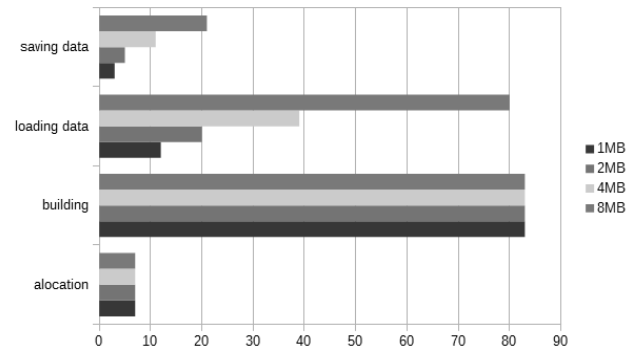

Fig. 3. Times of transferring data between host and OpenCL framework using GPU system

Times of transferring data of sub-circuits represented by (3) between host and OpenCL framework on GPU device is shown on fig. 3 and CPU device on fig. 4.

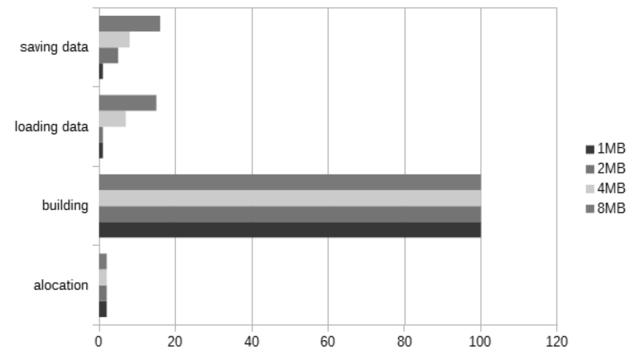

Fig. 4. Times of transferring data between host and OpenCL framework using CPU system

Computational experiments were performed for different sizes of the data system of equations (2) from 1MB to 8MB. “allocation” is the time of booking a data buffer in the GPU device. The “loading data” is the time of data transfer from host into a buffer in GPU device. The “saving data” is a time of receiving computed data to the host. The “building” is a time of compiling and transferring base code of the program. Time of transferring compiled code into GPU device is similar to the time of allocation. The cost of added elements of communication between host and GPU device is short. Value of the time $t_s(n)$ and number of reconciliation determines the speed of the whole method of calculation of transient processes in equation (7).

Conclusion

In this article the model of computations in CPU,GPU systems is presented, which is optimized for parallel execution of tasks of analyses of transient processes. The schema of distributions of tasks in heterogeneous and steering of processes were shown. Proposed concept of the computation system with *OpenCL* framework was analysed according to the time of compilation, loading and saving data in GPU system. The time of building application in the *OpenCL* framework has a big cost against the times of transferring data. Times of buffering and copping data are acceptable.

The attempt of the estimation of the total time of computation was made for the diakoptic model of the analyses of transient processes.

Author: *dr inż. Marcin Cegielski, Politechnika Łódzka, Instytut informatyki, ul. Wólczańska 215, 93-005 Łódź, E-mail: marcin.cegielski@p.lodz.pl.*

REFERENCES

[1] Kron G., A set of principles to interconnect the solution of physical systems, *Jurnal of applied Physics*, vol 24, no. 8, August 1953, page. 980
[2] Stakhiv P., Rendzinyak S., Krupskyy B., Parallelization of Diacoptic Methods for Multiprocessor Computing Systems, *Bulletin of The Polish Academy of Sciences Technical Sciences* Vol. 51, No. 4, 2003
[3] Byczkowska-Lipińska L., Cegielski M., Parallel Computations of Transient Processes in The Electric Circuits in Cluster Systems, *Przegląd elektrotechniczny*, 2/2007, nr 5
[4] Cegielski M., Stachiw P., Stability of parallel computations of transient processes in the electric circuits, *Przegląd elektrotechniczny*, 2/2007, nr 5
[5] Stakhiv P., Cegielski M., New approach to stability of parallel parallel computations of transient processes, *Przegląd elektrotechniczny*, 12/2008, nr 84
[6] Howes L., Munshi A., The OpenCL Specification, Version: 2.1, Khronos OpenCL Working Group, 29 January 2015
[7] Cegielski M., Stachiw P., Stability of parallel computations of transient processes in the electric circuits, *Przegląd elektrotechniczny*, 2/2007, nr 5
[8] Cegielski M., Stachiw P., Evaluation of criteria New approach to stability of parallel computations of transient processes, *Przegląd elektrotechniczny*, 2008, nr 6
[9] Stakhiv P., Rendzinyak S., Krupskyy B., Parallelization of Diacoptic Methods for Multiprocessor Computing Systems, Bull. Pol. Ac.: Tech., Vol. 51, No.3 (2003), pp. 381-394
[10] Uriarte F. M., On Kron's diakoptics, *Electric Power Systems Research,* 2012, Vol. 88, pp. 146-150
[11] Lelarasmee E., Ruehli E., A.L. Sangiovanni-Vincentelli, The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits, *Computer-Aided Design of Integrated Circuits and Systems*, *IEEE*, Vol. 1, Issue: 3, pp. 131-145
[13] Notaros B., Ilic M., Olcan D., Djordjevic M., Manic A., Chobanyan E., Hybrid higher order numerical methods in electromagnetic, *International Conference on Electromagnetics in Advanced Applications (ICEAA)*, 2014, pp. 411-414