Acta Informatica Pragensia
2(2), 2013, 59–67, DOI: 10.18267/j.aip.24
Online: aip.vse.cz

Section:
Peer-reviewed papers

Framework for utilizing computational devices within simulation

Miroslav Mintál[1]

[1] Department of Transportation Networks, Faculty of Management Science and Informatics, University of Žilina, Univerzitná 8215/1, 010 26 Žilina, Slovak Republic

Miroslav.Mintal@fri.uniza.sk

Abstract: Nowadays there exist several frameworks to utilize a computation power of graphics cards and other computational devices such as FPGA, ARM and multi-core processors. The best known are either low-level and need a lot of controlling code or are bounded only to special graphic cards. Furthermore there exist more specialized frameworks, mainly aimed to the mathematic field. Described framework is adjusted to use in a multi-agent simulations. Here it provides an option to accelerate computations when preparing simulation and mainly to accelerate a computation of simulation itself.

Keywords: GPGPU, GPU framework, OpenCL, Simulation

1 INTRODUCTION

Different simulation studies are being developed worldwide every day. These simulations often contains complicated models, which computation often last for a long time. In several cases is the computation slower than the modeled system. It happens especially if we want to model larger systems.

A solution may be to buy a new more powerful hardware or to rent a computation power. However these restrictions can be conquered also with utilization of current hardware. A majority of programs is serial and that is why they cannot fully utilize current multi-core processors. After adjustment, these programs could fully utilize the processor. In a computer occurs more powerful computational device when used properly. This device is a graphic card.

To fully use this hardware there exist several frameworks and libraries. Some of them will be described in chapter two. This paper will deal with a designed framework that is able to utilize mentioned hardware with focus placed on the utilization in simulation. Designed framework is integrated into multi-agent simulation architecture ABAsim [1]. This is utilized both on education purposes as well as commercially in the traffic simulations. Designed framework will be described in third chapter. Chapter four defines a practical use of the framework. Last chapter will contain summary and conclusion.

2 EXISTING FRAMEWORKS

Recently there did not exist any frameworks for simple use of computational power of graphic cards. The only option was to convert data into textures and work with them in a complicated way. To utilize a computation power of graphic cards and other computation devices there exist nowadays several frameworks. There is a small amount of basic frameworks. These will be described in subsection 2.1. Then there exist frameworks based on basic frameworks that simplify utilizing of this performance such as automation of some steps, simplifying function calls or implementation of some algorithms. In subsection 2.2 will be some of these frameworks presented.

2.1 LOW-LEVEL FRAMEWORKS

Among basic frameworks that enable to utilize computational devices belong these:

CUDA [2] – is mostly used, especially for good publicity and amount of provided libraries. The disadvantage is that it only works with NVidia graphic cards.

OpenCL [3] – its main advantage is that it can use computational power of almost every recent graphics card, processors and FPGA. Because of the wide spectrum of supported devices, it has a disadvantage in more complicated host code.

DirectCompute – is part of DirectX. It is used in engines built on DirectX. It is rarely used for general computations.

2.2 HIGHER FRAMEWORKS

Then there exist frameworks and libraries based on mentioned low-level frameworks. These utilizes in most cases OpenCL and CUDA. They simplify the work with computation on graphic cards in the way that they make some steps automatically and provides simpler calls of functions. The best example is a graphic card initialization automation before the computation. In the OpenCL an initialization of single graphic card can take more than fifty lines of code.

Now will be described frameworks that are focused on the mathematical computations on the graphic cards:

VexCL **[**4**]** – for initializing of computational devices it is necessary to call initializing call. However it has an advantage that the code is executed over all requested devices. To copy data between graphic card and processor it uses simple call of `copy` command. Data can be also set by direct value assignment, but this is computationally slow. Furthermore it supports simple execution of simple mathematic operations. Operation is written with the graphics card variables and is automatically executed over all element of variables. An example of such a simple operation may look as follows:

```
gpu_vec_C = gpu_vec_A + gpu_vec_B;
gpu_vec_D = gpu_vec_C * 2.0;
```

It also supports more complicated operations however the whole code with parameters and all key words from OpenCL must be written.

ViennaCL [5] – automatically initializes graphic card during the first call of any function from this library. As well as VexCL it supports simple entries in simple mathematical operations. It also supports data copying in the simple function `copy`, or by directly values assignment, but it is slow.

MTL4 – it is similar mathematical framework as previous two with one difference. This framework is based on CUDA, so it can use only NVidia graphic cards for computations.

Frameworks MTL4, ViennaCL and VexCL were compared in the paper from Demidov et. al. [**6**]. In the paper they are compared on chosen mathematical problems, where they achieve similar results on sufficient big problem. In the small problems a time of initialization was the factor. When comparing CUDA and OpenCL, were the frameworks based on CUDA a little bit faster. However practically this difference was insignificant. In spite of CUDA, OpenCL supports more hardware.

Flame GPU **[**7**]** **[**8**]** – is computational framework assigned for simulation. One can see two fundamental disadvantages. The first is, that it is based on CUDA. Therefore it can use only the NVidia graphic cards. As NVidia form only a part of current graphic cards and do not support other computational devices, its utilization is limited. Other possible disadvantage is the use of XML based code when programming. This means a longer studying before work with the framework.

3 DEVELOPED FRAMEWORK

Designed framework is built on the OpenCL. Due to this it can work on different computation hardware and not only on the graphic cards NVidia, as it is in the case of CUDA framework. It is aimed on the computations during simulations. Because of this it does not support simple mathematical operations in the simple calls. During simulation are usually utilized complicated computations. These can be created in a simpler way within the framework. Framework supports also other benefit properties of mentioned frameworks, such as: automatic initialization, easier copying of data between the computational device and computer memory and other, that will be described later.

Framework is kernel oriented not a data oriented. That means that for every computation there is created a kernel object with a code and data can assigned into this. This kernel can then simply be called several times with different data. It is also possible to assign data output from one kernel as a data input of another kernel. Then simply a computation over the kernels is called and no commands to transfer data between kernels have to be entered.

In the next parts will be the framework described in parts and its advantages will be pointed out.

3.1 INITIALIZING

Framework can automatically initialize itself. It is initialized during a first call that will have to work with the graphic card. If no graphic cards are found the computations will be parallel executed on the processor or other device supporting OpenCL. If necessary it is possible to manually define which device should be initialized. For example if it is wanted to execute the computations only on the specific graphic card.

3.2 MEMORY

When creating a kernel it is necessary to define the types of input and output data (parameters of the kernel). This is done by the next procedure:

```
procedure addParamArr<TType>(paName: String = '';
    paMemoryType: TECMemoryType = mtReadWrite;
    paLength: Cardinal = 0);
```

The only obligatory parameter is `TType`, `TType` is converted into appropriate type in OpenCL. If `TType` is a complicated `record`, then a `struct` with the same name is created in OpenCL. Then every element of the `record` is converted into an appropriate type in OpenCL and is inserted into the created `struct` under the same name as it has in the `record`. It is better to set a name that is later easy to use when copying data and during the pointing on the data in the kernel.

Then `setParam` and `getParam` methods can be called over the kernel to set data and to obtain data from the graphic card. In the calls a parameter name of kernel can be used. Framework than copies correct size of data between graphic card and the computer memory.

3.3 KERNEL CODE

Kernel code can be entered directly as a string or as a name of a file where the code is. If the code is in file and it would contain a syntax error, framework announces this error and will wait for the error repair in the file. If the code would be repaired, framework is able to continue in the computation without crash or shutting down the application.

Another advantage of the framework is, that it is not necessary to write a declaration of kernel function. Name of the kernel together with the correct parameter types and surrounding OpenCL code is generated automatically. It is needed to write only the computation code for the kernel that will use the names of the parameters identified during kernel creation. If the names were not defined, generated names will be created according to the type of used memory.

Sometimes it is more useful not to have a lot of kernels, but call the functions from one kernel. For example for the reason of speed or code readability. It is possible to assign these functions into the kernel again as a string or a file name. When there is an error in the function code in the file, it is again possible to repair it by announced notification and continue in the application run.

3.4 TEMPLATE FUNCTIONS

As was mentioned, framework does not support simple notation for simple mathematic operations. However it supports template functions for more complicated functions utilizing computational device. To these operations belongs for example `prefix sum (scan)` or `sort`.

It is because these operations are mostly specific for parallel computations or more difficult to code and optimization. That is why a user can use them with benefit and is not forced to concern in detail of how they work on graphic cards or other computational device.

3.5 SIMULATION

Framework is integrated into the multi-agent simulation architecture ABAsim. This architecture is divided into two layers: controlling agents and dynamic agents. Controlling agents are responsible for larger parts and work as a controllers. Dynamic agents represent intelligent entities presented in the model. These can be neurons in brain, ants in anthill or cars, trains and pedestrians in the traffic simulation.

Framework is integrated into dynamic agents. It is because there is a large amount of these agents and they execute the same activity (the same code). This is preferable for graphic cards. For optimal utilization it is requested that the executed code is the same for all threads and is executed in as big as possible number of threads. Algorithm executed on big amount of threads is advantageous also from the point of future computation acceleration. The trend heads to the computational devices with higher number of cores that can faster process more threads.

This way it is possible to create a kernel for dynamic agent and register it in the framework. This kernel is self-executed during simulation for every agent. Of course, this framework is able to utilize in other parts of simulation. It is enough to create kernel and then execute it in desired places.

4 USING OF PROPOSED FRAMEWORK

Described framework can be used during simulation in several ways and also it can be used outside the simulation. Framework has been tested in the pedestrian simulation. Before simulation itself it can be used to accelerate pre-simulation computations. For example to compute gradient maps [9] used for pedestrian navigation. Firstly we will describe a simple example of vectors addition. Secondly we will outline computations before and during simulation of pedestrians on graphic cards with their advantages.

4.1 VECTORS ADDITION

An example of vectors addition (see below) is composed of three blocks. The first block creates reusable kernel. The second one copies data from RAM to graphic card and starts the computation. The third block deallocates kernel. More comprehensive description of the example is located below.

```
// Part 1
length := 20;
FArrayAdd := TECKernel.Create;
FArrayAdd.Code := 'c[id] = a[id] + b[id];';
FArrayAdd.addParamArr<Single>('a', mtReadOnly);
FArrayAdd.addParamArr<Single>('b', mtReadOnly);
FArrayAdd.addParamArr<Single>('c', mtWriteOnly);
// Part 2
FArrayAdd.setParamLength('c', length);
FArrayAdd.setParam('a', a_cpu);
FArrayAdd.setParam('b', b_cpu);
FArrayAdd.Compute(length);
FArrayAdd.getParam('c', c_cpu);
// Part 3
FArrayAdd.Free;
```

In the first block, the kernel is created and code for graphic card is assigned. Afterwards the parameters with their types are defined. The second block executes the computation using `Compute` method. Result is retrieved by `getParam` into variable `c_cpu`. Another values for addition can be set using `setParam`. Call `setParamLength` is required only for output parameters, which size changes. The third block comprises just a single command, which deallocates the whole kernel with all parameters (if they are not used in another kernel).

4.2 COMPUTATION OF GRADIENT MAP

Computation comprises two phases. Firstly a distance matrix is calculated. Gradient map is then created from this distance matrix (see Figure 1).

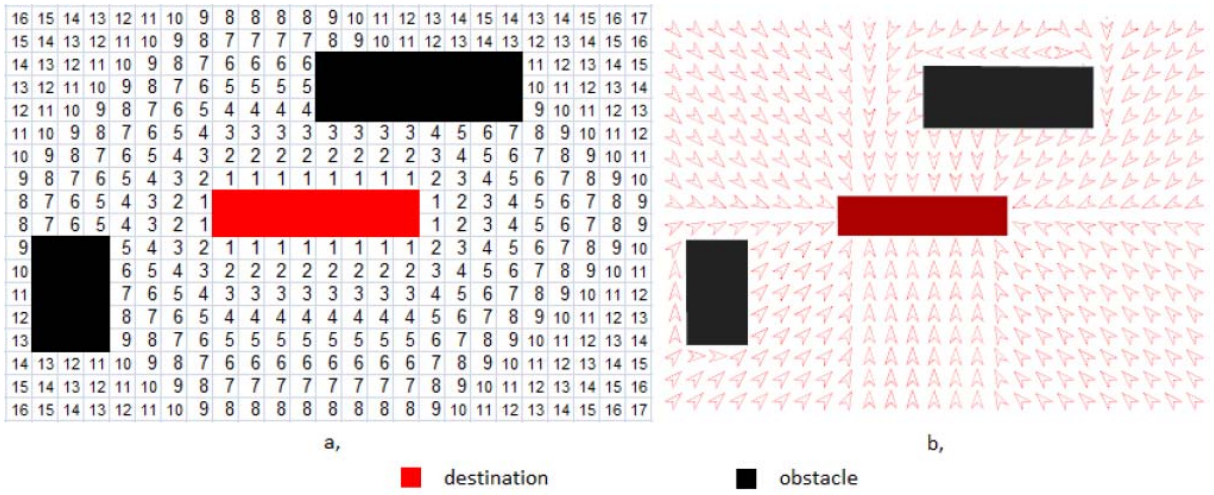

Fig. 1. Distance matrix (a,) and gradient map (b,) with two obstacles [9]

Two kernels are needed for distance matrix computation. The first contains code [9] for distance matrix calculation in local group. The second one synchronizes groups. Data about obstacles are then sent to graphic card. This is accomplished by assignment of attributes to the first kernel. Both kernels share the same part of memory on graphic card. Output parameter of the first kernel is assigned to an input parameter of the second kernel and vice versa. Kernels are executed alternately until the whole distance matrix is calculated.

Finally, a gradient map is computed from the distance matrix. For this we need another kernel, which for each cell finds the smallest value in neighborhood. Input of this kernel is distance matrix and output is gradient map.

This solution has several advantages. Processor can execute another pre-simulation computations during computation of gradient maps on graphic card. Thanks to it, we can completely utilize the time needed for creation of gradient maps. Moreover, resulting maps are already located on graphic card and can be used for computation of pedestrians' movement.

4.3 PEDESTRIAN MOVEMENT

During simulation it is able to use a connection of framework with simulation architecture ABAsim. This solution is described in [10]. Described solution uses all properties of framework to implement a pedestrian movement based on social forces [11]. Firstly, a kernel is created. Computation is similar to processor alternative except finding of obstacles and pedestrians in surroundings. Static parameters are set to kernel, such as obstacles or gradient maps. This kernel registers itself into ABAsim architecture. We have to create a function for actualization of pedestrians attributes, for new pedestrian arrival and for pedestrian departure from the simulation.

Before the computation of movement, the pedestrian asks, if there is available kernel for its type of movement. If there is one, ABAsim architecture calls function for actualization of pedestrians attributes, starts computation on graphic card and copies results from the card (if they are needed e.g. for statistics).

The presented solution was tested on two simulation models. Examined models of the pedestrian movement are depicted in figure 2.

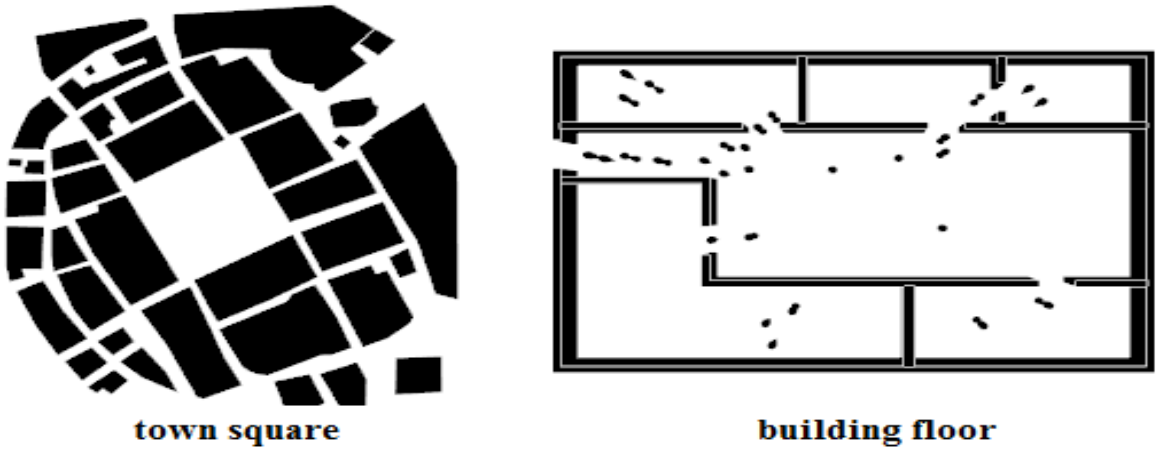

Fig. 2. Models of tested infrastructures [10]

The results of accelerating of movement of pedestrians on the graphics cards opposing to the serial implementation are depicted in figure 3. The computation for higher amount of pedestrians in simulated model was on the graphic card significantly faster than original serial implementation. Only for very small amount of pedestrians (up to the ten pedestrians) was the computation on the graphic cards slower. However in practical use it is needed to accelerate a computation for large amount of pedestrians. That is why results reached by this framework are very good.

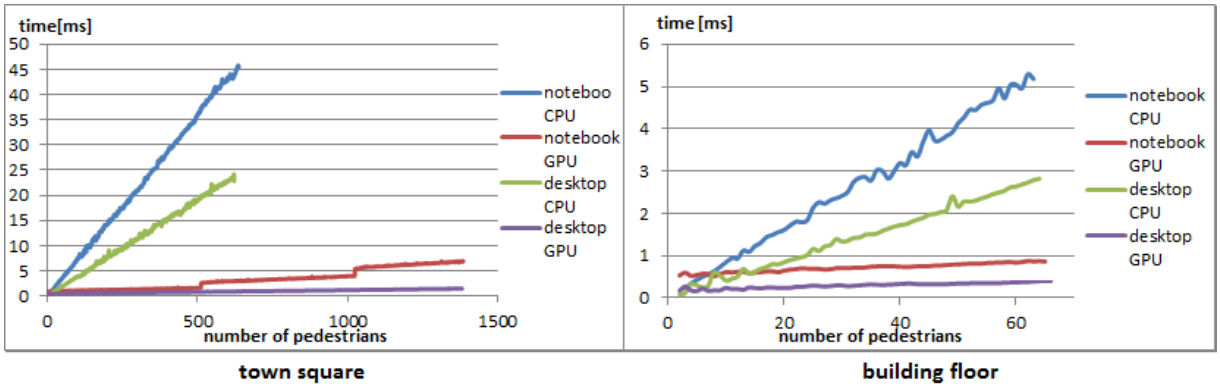

Fig. 3. Results of comparison of framework with original implementation [10]

5 CONCLUSION

The paper presented a framework for utilization of computational power of graphic cards, multi-core processors and other computational devices supporting OpenCL. This framework simplifies initialization and application against basic OpenCL framework. Due to the integration into the multi-agent simulation architecture ABAsim it is possible to utilize it for accelerating of simulation computations. This way it is possible to create larger simulation models which computation will last shorten time.

The framework was utilized to implement various algorithms. It was used also in the pedestrian movement simulation. This implementation significantly accelerated a computation of pedestrians in several models. Algorithms implemented in this framework can in future better utilize new hardware. This is because algorithms are parallel and are able to use a potential of new multi-core computational devices compared to current serial algorithms.

6 REFERENCES

[1] ADAMKO, N. and V. KLIMA. Agent based simulation architeture argumented by actors. In: *SM'2006 - The 2006 European Simulation and Modeling Conference*. Tolouse: 2006, pp. 305-09. ISBN 90-77381-30-9.

[2] KIRK, D. B. and W.M. W. HWU. *Programming Massively Parallel Processors*. 2010. ISBN: 978-0123814722.

[3] GASTER, B. et al. *Heterogenous Computing OpenCL*. 2011. ISBN: 978-0123877666.

[4] DEMIDOV, D. In: *Vector expression template library for OpenCL* [online]. 2013 [cit. 2013-10-15]. Available at: https://speakerdeck.com/ddemidov/vexcl-at-cse13

[5] RUPP, K. F. RUDOLF and J. WEINBUB. ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In: *International Workshop on GPUs and Scientific Applications*. Vienna: 2010, pp. 51-56.

[6] DEMIDOV, D. et al. *Programming CUDA and OpenCL: A Case Study Using Modern C++* Libraries. 2012, p. 21.

[7] RIHMOND, P. and D. ROMANO. Template driven agent based modelling and simulation with CUDA. In: *GPU Computing Gems Emerald Edition*. 2011, pp. 313-24. ISBN 978-0-12-384988-5.

[8] KARMAKHARM, T. P. RICHMOND and D. ROMANO. Agent-based Large Scale Simulation of Pedestrians With Adaptive Realistic Navigation Vector Fields. In: *Theory and Practice of Computer Graphics*. Sheffield, 2010, pp. 67-74.

[9] MINTÁL, M. Accelerating distance matrix calculations utilizing GPU. In *Journal of Information, Control and Management Systems*, Vol. 10, No.1, 2012, pp. 71-79. ISSN 1336-1716.

[10] MINTÁL, M. Social forces pedestrian simulation utilizing graphics card within simulation tool PedSim. In *IMEA 2013*, Pardubice: University of Pardubice, 2013. ISBN 978-80-7395-696-7.

[11] KORMANOVÁ, A. Combining social forces and cellular automata models in pedestrians' movement simulation. In *Journal of Information, Control and Management Systems*, Vol. 10, No.1, 2012, pp. 61-70. ISSN 1336-1716.