Modification of self-organizing migration algorithm for OpenCL framework

MICHAL PAVLECH, JAN SECKAR
Faculty of Applied Informatics
Tomas Bata University in Zlin
nam. T.G.Masaryka 5555, 760 01 Zlin
Czech Republic
pavlech@fai.utb.cz, seckar@fai.utb.cz

Abstract: - This paper deals with modification of self-organizing migration algorithm using the OpenCL framework. This modification allows the algorithm to exploit modern parallel devices, like central processing units and graphics processing units. The main aim was to create algorithm which shows significant speedup when compared to sequential variant. Second aim was to create the algorithm robust and portable to allow its execution on a range of different devices. Results of tests presented in this paper show that both aims were successfully fulfilled.

*Key-Words:*OpenCL, GPGPU, SOMA, optimization, parallel

1 Introduction

General purpose computing on graphics processing units (GPGPU) offers the advantages of massively parallel computing with relatively low cost hardware. Various scientific fields have already adopted the GPGPU, including the research in evolutionary computing. There are three different frameworks which allow the usage of GPUs for scientific computation: Open Computing Language (OpenCL), Compute Unified Device Architecture (CUDA) and DirectCompute. While there have been quite a lot of research conducted on the matter of evolutionary algorithms (EAs) for CUDA framework the other two frameworks are a lot less common amongst the scientific papers.

The CUDA framework works only on GPUs made by Nvidia but supports several operating systems (OS), DirectCompute supports GPUs from both major manufacturers (Nvidia and AMD) but only supports Microsoft OS (Windows 7 and newer). Great advantage of OpenCL is its ability to run on GPUs from both major manufacturers as well as on central processing units (CPUs) from both Intel and AMD, supported operating systems include 32 and 64bit varieties of MS Window, Linux, and OS X. Therefore it is our belief that OpenCL applications may posses higher added value and do not carry the possibility of vendor lock-in for neither hardware nor OS.

Previous uses of OpenCL include several evolutionary algorithms including the genetic algorithm [1], evolutionary strategies [2] and particle swarm optimization [3].

Self-organizing migration algorithm (SOMA) is an optimization method belonging to the group of evolutionary algorithms. Its nature makes it especially suitable for parallel implementations as it has very low requirements for inter-thread communication.

The paper is divided as follows: the methods part describes OpenCL framework, SOMA algorithm, the modifications of SOMA algorithm which were necessary for its implementation in OpenCL and methodology of testing. Second part presents results of tests in graphical format and is followed by the discussion.

2 Methods

2.1 OpenCL

OpenCL is an open, free framework for programming of modern parallel computing devices. The framework is maintained by Khronos group [4] which is responsible for development and approval of new features. The current stable version 1.2 is supported by all relevant hardware vendors and is available for a broad range of operating systems.
The main difference between OpenCL and CUDA framework is that OpenCL is not only aiming at GPUs but at a wider range of devices. Supported devices include GPUs (Nvidia and AMD), CPUs

ISBN: 978-1-61804-117-3

(Intel and AMD) and mobile processors (ARM). The developers (and consequently the users) are not bound to specific hardware or vendor and thus the programs using OpenCL have larger target audience.

Because OpenCL programs must be able to run on high number of different devices (which are frequently incompatible at the binary level) it was necessary to employ the online compilation. This means that programs which have OpenCL components have to load the source code and compile it specifically for the device which will then run the component. The newer versions of framework make it possible to use offline compilers and compile the source code into libraries and use them without compilation each time the application is launched, but these libraries are device specific and programs loose their portability.

The execution of OpenCL programs is divided into two parts based on the hardware which runs the software component: device and host. The device is a collection of compute units which run special functions called the kernels which are executed in parallel on the device. A compute unit is composed of one or more processing elements and local Memory. The host is a hardware which interacts with the device using the OpenCL context.

The kernels are run on the device using the single instruction multiple data (SIMD) programming model. This means that high number of kernels is executed concurrently on numerous computing units each one with a distinct data but their instruction set is equal.

The kernels can be further merged into work-groups which enable more delicate control over their execution, with each work-group being executed on a single computing unit and kernels in a work-group having access to the shared memory. Kernel in a work-group is called a work-item.

2.2 SOMA

Self-organizing migration algorithm, SOMA, was first introduced by Zelinka in 2004 [5]. It is an evolutionary optimization technique which mimics the behavior of a pack of semi-intelligent individuals which cooperate in their search for resources. In this case the resource is represented by the minima of the function. SOMA maintains population of solutions like most of the EAs but due to its different background it uses slightly different nomenclature, one optimization round is called migration instead of generation. The basic version of SOMA, called AllToOne, finds better solutions by moving individuals in the solution space towards the individual with the best fitness (the leader). The movement distance and coarseness of the search is controlled by several variables:

- *PathLength* controls how far from the leader will active individual stop its movement
- *Step* controls the length of each discrete step.
- *PRT* variable defines the stochastic element of movement

PRT controls the creation of *PRTVector*, which defines if a dimension of solution can or cannot be changed during this migration. *PRTVector* is generated for each individual separately according to equation:

$$\begin{aligned} \mathbf{PRTVector}_j &= 1 \text{ if } \text{rand}_j(0,1) < PRT \\ &= 0 \text{ otherwise} \end{aligned} \quad (1)$$

The movement is then performed using equation:

$$\mathbf{x}_{i,j,t}^{ML+1} = \mathbf{x}_{i,j,start}^{ML} + \left(\mathbf{x}_{L,j}^{ML} - \mathbf{x}_{i,j,start}^{ML}\right).t.\mathbf{PRTVector}_j \quad (2)$$

where *ML* is the number of current migration round, $x^{ML}_{i,j,start}$ is position of active individual at the beginning of current migration, $x^{ML}_{L,j}$ is the position of the leader, $t \in [0; pathLength]$, *t = 0, Step, 2*Step,...* The cost function for each step is evaluated and the position of an individual for next migration round is set to the position with the best value of cost function found during the movement.

2.3 OCLSOMA

Modification of SOMA for OpenCL framework is implemented using the official C++ bindings for the OpenCL. The host portion of the source code is implemented as a standard C++ class with critical parts calling the OpenCL API. Most of the OpenCL functions use one thread per individual.

Great disadvantage of OpenCL over the CUDA is the lack of native package for generation of random number on the device, like CURAND. Previous studies overcame this problem by various methods, for example by generation of random numbers on the host and copying to the device [1].

OCLSOMA employs random number generator (RNG) proposed in [6] and is based on source code provided by Wes Devauld [7]. The generation of random numbers is implemented as a kernel and is run entirely on the device. To ensure that different sequence of random numbers is generated for each thread (individual) a different state, which consists of 6 integer numbers, is maintained for each thread.

The original state is initialized using the C++ rand() function and copied to device. Only one number per thread is copied to device, consecutive kernel call copies each random number into 5 other numbers and thus creating RNG state.
The population is stored on the device and is copied to the host only after a given number of migration rounds finishes. The population consists of float values and is stored in a one dimensional array, where genotype values are followed by fitness value. Access to particular individuals employs index translation.
Prior to first migration it is necessary to create an initial population which is implemented on the device with a separate kernel. The number of kernels launched is equivalent to population size, each kernel obtains random values from RNG and multiplies them into a range defined for each test function. After the genotype of whole individual is generated a cost function is computed and all values are stored in population array.
Before any individual can perform migration it must know the position of the leader. Search for the leader is performed on the device before each migration round and employs the parallel reduction technique. The index of the leader is stored in global memory where it can be accessed by all threads.
A separate kernel is employed for migration, with migration of each individual being run in a separate thread. In order to store temporary positions of individuals and temporary best found value there are 2 more arrays in global memory besides the population. Both arrays have the same size and dimensions as population. First array is used for storing the result of each step during the migration, second stores best found value during this migration. After all steps are performed the best value is copied into population. The leader is not performing migration, so there is no threat of rewriting the data of individual which is currently in use by another thread.
OCLSOMA records its runtimes in 3 separate areas:

- Kernel time – time needed for kernels to finish.
- Data transfer time – time needed to copy the data to and from device.
- Initialization time – initialization of OpenCL and arrays.

These times are used to evaluate performance of OCLSOMA.

2.4 Performance tests

Performance of OCLSOMA was evaluated on 3 standard test functions with aim to measure the benefits and flexibility of OpenCL modification. All test functions are implemented as device functions. These functions are:
De Jong's function:

$$f(x) = \sum_{i=1}^{n} x_i^2$$
$$-5.12 < x_i < 5.12 \qquad (3)$$

Schwefel's function

$$f(x) = \sum_{i=1}^{n} [-x_i \sin(\sqrt{|x_i|})]$$
$$-500 < x_i < 500 \qquad (4)$$

Griewangk's function

$$f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1, \qquad (5)$$
$$-600 < x_i < 600$$

All the tests were repeated 10 times and their results were averaged in order to avoid random glitches. The tests were run on 3 different devices to show the ability of OpenCL solution to run on a range of devices. First two devices were GPUs from both major manufacturers: Nvidia Tesla C2075 and AMD Radeon HD 7850. Third device was multi-core CPU Intel Xeon E5607. Serial implementation of SOMA was run on the Xeon E5607 CPU to provide a initial values to which OpenCL variant can be compared.
It was noticed during the development that the size of work-group has significant influence on the performance. To test this issue a test was conducted with different number of work-items in a work-group on all 3 devices. The population size was set to a fixed value of 3072 (number divisible by all tested work-group sizes), the dimension of test functions was set to 100. The work-group size was changed in powers of 2 and time needed to finish 10 migration rounds was measured.
The purpose of the next test was to determine how OCLSOMA scales to an increasing number of work-items in this case represented by individuals in population. The dimension of cost function was set to 100 and population size was changed from 100 to 2100 in steps of 200. The time needed for 10 migration rounds was measured and the results were compared to serial implementation.
Last set of tests is targeting the ability of the algorithm to scale to an increase in computational complexity which is represented by increase in cost function's dimension. The population size was set to

a fixed value of 1000, the dimension was changing from 25 to 250 with a step of 25. And as in previous cases time needed for 10 migration rounds was measured.

3 Test results

3.1 Work-group test

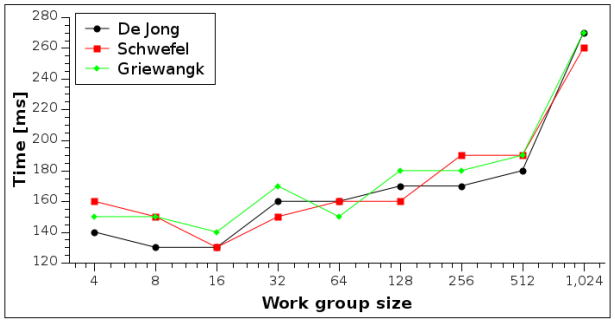

Fig. 1 *Work-group test on Nvidia Tesla C2075*

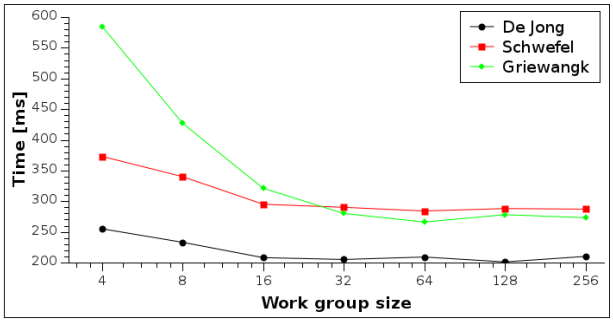

Fig. 2 *Work-group test on ATI Radeon 7850*

Results showed (Fig. 1 – Fig. 3) that work-group size has substantial impact on the performance of OCLSOMA. It can be seen that optimal work-group size depends more on the cost function than on the device used. All remaining tests in this paper used work group size of 32 in order to keep the consistency across tests however in order to achieve the top performance it is necessary to run tests for specific [device; cost function] pairs.

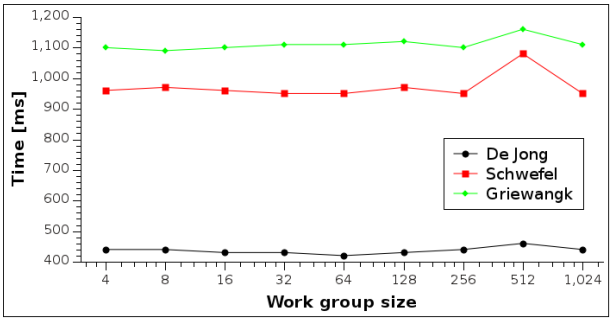

Fig. 3 *Work-group test on Intel Xon E5607*

3.2 Population size test

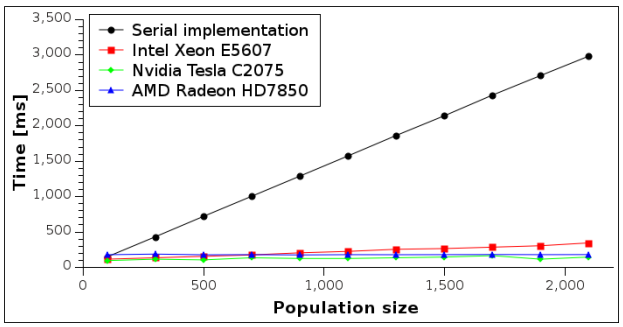

Fig. 4 *Population size test – De'Jong's function*

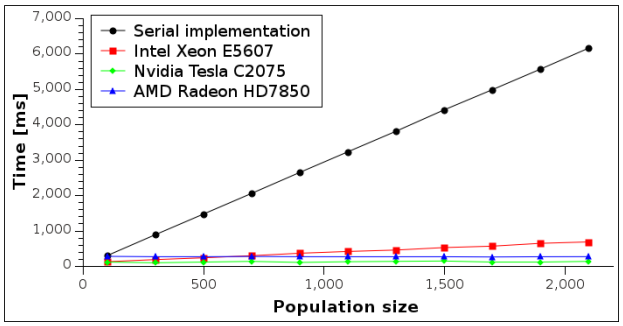

Fig. 5 *Population size test – Schwefel's function*

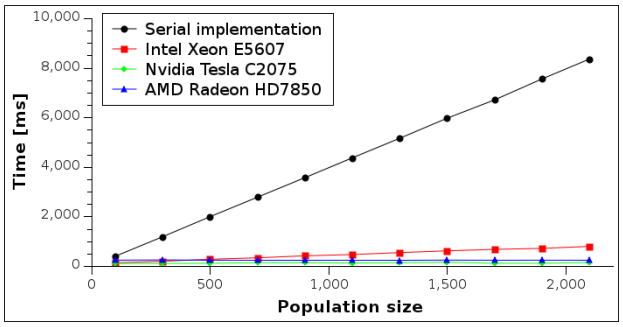

Fig. 6 *Population size test – Griewangk's function*

The population tests showed that OCLSOMA can scale very well to increase in population size. As can be seen from comparison to sequential version (Fig. 4 - 6) the OpenCL version of algorithm is considerably faster that the sequential variant.
Table 1 shows the speedup of OpenCL variants (population size 2100) on different devices compared to sequential implementation.
Instances running on GPUs show the highest speedup but also an instance running on multi-core CPU outperforms sequential version by a factor of at least 8.

Table 1 *Speedup values of OCLSOMA – population test*

	De'Jong	Schwefel	Griewangk
Radeon HD7850	17.0	23.4	36.0

ISBN: 978-1-61804-117-3

Tesla C2075	21.3	47.4	59.6
Xeon E5607	8.8	9.1	10.6

3.3 Cost function's dimension test

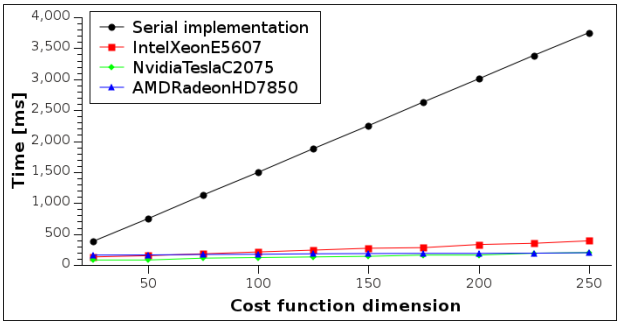

Fig. 7 *Dimension test De'Jong's function*

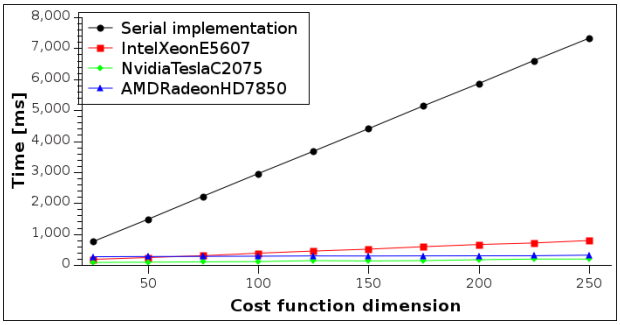

Fig. 8 *Dimension test Schwefel's function*

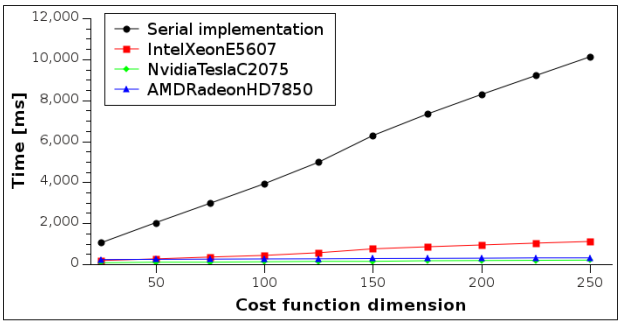

Fig. 9 *Dimension test Griewangk's function*

Table 2 *Speedup values of OCLSOMA – dimension test*

	De'Jong	Schwefel	Griewangk
Radeon HD7850	18.9	23.0	32.6
Tesla C2075	18.8	38.5	50.6
Xeon E5607	9.6	9.3	9.1

Tests with increase in cost function's dimension (Fig. 7 – Fig. 9) show that OCLSOMA is capable of scaling to computationally more complex problems. As with the population tests, the highest values of speedup were measured for GPU devices, but multi-core CPU is also a preferable alternative to sequential variant.

3.4 Composition of time

As a consequence of online compilation feature of OpenCL programs the runtime of OCLSOMA is longer than is needed for actual run of SOMA algorithm and necessary data transfers. Time measurements used 3 variables for 3 aspects of algorithm: initialization time (acquisition of available context, devices compilation, loading of kernels and setting their arguments), data transfer time (copying of arrays to and from the device) and kernel time (all kernel functions – generation of initial population, generation of random numbers, cost functions, evolutionary operators). Fig. 10 and Fig. 11 show total and relative values for these 3 times on Griewangks function, dimension set to 100 running on Xeon E5607.

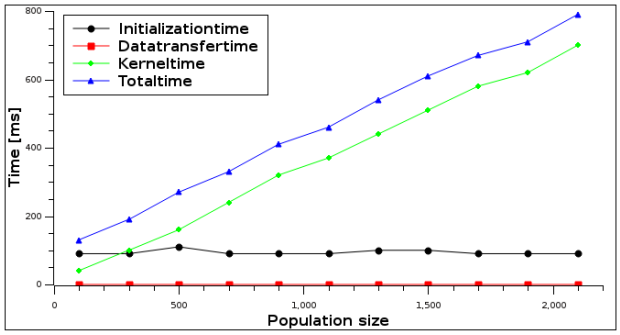

Fig. 10 *Duration of 3 distinct parts of OCLSOMA*

The initialization time remained almost constant without being influenced by population size or cost function dimension and changed only with regard to cost function and device (Table 3). The high relative amount of initialization time for small population sizes is much less influential for higher sizes. Data transfer times were so low that they fell bellow the recognition threshold of standard C++ time functions.

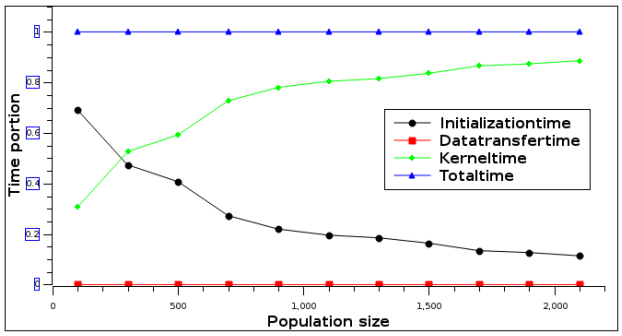

Fig. 11 *The time ratios of 3 distinct parts of OCLSOMA*

ISBN: 978-1-61804-117-3

Table 3 *Average initialization times*

	Radeon HD7850	Tesla C2075	Xeon E5607
Initialization time [ms]	203.9 ± 42.17	64.7 ± 28.00	284 ± 3.40

4 Conclusion

Great advantage of OpenCL framework when compared to CUDA is its ability to run on broader range of devices. Thus we were able to test OCLSOMA on devices from 3 different vendors and 2 hardware groups.

Tests with 3 artificial functions showed the performance of OCLSOMA with comparison to sequential SOMA.

We examined the influence of work-group size on the algorithm's performance prior to other tests.

OCLSOMA scaled extremely well to the increase in population size with the highest recorded speedup of 59.6. The increase in cost function's dimension yielded similar, although slightly worse speedup values.

OpenCL showed promising results on all 3 devices. It is of no surprise that best speedups were achieved on a dedicated compute card - Tesla C2075, but the Radeon offered satisfying results for only a fraction of priors card's price. The only CPU in our tests – Xeon E5607 shows that OpenCL programs can also utilize multi-core CPUs with profit.

The only notable drawback of OpenCL framework is the need for online compilation. We examined the influence of this feature on runtime and conclude that initialization time remains the roughly the same for all observed population sizes and thus its influence decreases as the computational complexity grows.

Our further work will examine the performance of SOMA algorithm under both CUDA and OpenCL frameworks in order to compare their benefits for parallel evolutionary computation.

Acknowledgements

The research described in this paper was supported by funding from European Regional Development Fund under project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089 and by the internal grant agency project IGA/FAI/2012/033.

References:

[1] Puzniakowski T, Bednarczyk M. A., Towards an OpenCL Implementation of Genetic Algorithms on GPUs, Lecture Notes in Computer Science, 2012, Volume 7053, Security and Intelligent Information Systems, Pages 190-203.

[2] Lőrentz I, Andonie R, Malita M, An Implementation of Evolutionary Computation Operators in OpenCL, Studies in Computational Intelligence, 2012, Volume 382/2012, pp. 103-113.

[3] Cagnoni S, Bacchini A, Mussi L, OpenCL Implementation of Particle Swarm Optimization: A Comparison between Multi-core CPU and GPU Performances, Lecture Notes in Computer Science, 2012, Volume 7248/2012, pp. 406-415.

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

[5] Zelinka I, SOMA—self organizing migrating algorithm, In: New optimization techniques in engineering, Berlin: Springer 2004.

[6] L'ecuyer P, Simard R, Chen EJ, Kelton WD, An object-oriented random-number package with many long streams and substreams, INFORMS, 2002, Vol. 50, No. 6, pp. 1073-1075.

[7] Devauld W, Heston-opencl-monte-carlo, [online] Available at: https://github.com/wdevauld/heston-opencl-monte-carlo