Michał **BUGAŁA**

AGOGE - AN INTEGRATED DEVELOPMENT ENVIRONMENT FOR OPENCL C

Abstract. The article presents the capabilities and technological solutions of Agoge, an integrated development environment dedicated to OpenCL. The automation of the process of creating an OpenCL context, handling of computational kernel and computational grid are discussed and the potential for using the environment in numerical calculations and image analysis is presented.

Keywords: development environment, OpenCL C programming language, OpenCL technology, parallel processing, image analysis.

1. INTRODUCTION

The Agoge Project was initiated by the author as part of research carried out by the Simulator Department of OBRUM in the area of acceleration of algorithm processing and image analysis. The article is not devoted to the advantages of using OpenCL technology, image processing and analysis algorithms, nor does it describe specific applications of the environment in projects carried out by OBRUM's Simulator Department (the potential of the project will be fully utilised in the future in the creation of efficient physical simulations in the area of aerodynamics, hydrodynamics and classical mechanics). The aim of this article is to present a tool for using and working with OpenCL technology in a fast and user-friendly way. The potential reader of this article and the user of the Agoge environment is a programmer looking for solutions to develop fast algorithms in a dedicated development environment that Agoge, described here, is aspiring to provide. From the perspective of OBRUM, where the project was developed, the reader should treat the Agoge environment as a tool for conducting research at the OBRUM's Simulator Department, and the aim of creating the project was to develop a solution that will enable efficient use of the OpenCL technology. These issues may seem distant from OBRUM's main tasks, but from the point of view of the Simulator Department and programming technologies they are very up to date.

OpenCL (*Open Computing Language*) is a heterogeneous technology[1] that enables creation of highly efficient programs using parallel processing, where many computations can be executed simultaneously. The added advantage of the OpenCL standard is its API[2] which allows the use of this technology in any development environment and programming language (via the *OpenCL C ++ Standard Library*) [1], [2]. While OpenCL API can be used in numerous development environments and programming languages, the computational routine (also called the computational kernel or computational function) must be prepared in the OpenCL C language [3].

1 The founding feature of OpenCL is that it supports various hardware platforms with various operating systems (unlike, for instance, CUDA or C ++ AMP). Currently, OpenCL is used on four types of computation devices (traditional processors, graphics cards, DSPs, FPGA type systems) and on operating systems of the Windows family, Linux and Mac OS X.

2 API (*Application Programming Interface*) - set of rules and descriptions used for communicating between computer programs.

Michał **BUGAŁA** – Ośrodek Badawczo-Rozwojowy Urządzeń Mechanicznych „OBRUM” sp z o.o., Gliwice

The Agoge development environment described in this article is the host of the OpenCL API, the implementation of which is invisible to the user (the user does not have to deal with the cumbersome preparation of the computation context[3] required to execute the OpenCL kernel) and it is an environment dedicated to the OpenCL C language version 1.1 using a data-driven programming model (computational grid)[4].

There are a number of tools that support OpenCL and OpenCL C, but most of them are of the SDK[5] type (AMD Accelerated Parallel Processing SDK, NVIDIA OpenCL SDK, Intel SDK for OpenCL) working in the OpenCL host environment. One exception is OpenCL Studio, which is a stand-alone application dedicated to OpenCL technology. However, also in this case the user has to take care of the OpenCL context itself using the LUA scripts [4].

One feature that should distinguish Agoge from other available OpenCL support tools is the ability to automatically create OpenCL contexts based on the arguments given in the OpenCL C kernel, isolate the source code of the kernel from the OpenCL host development environment[6], and reduce the number of used programming languages to one (from the user's point of view). Automatic generation of the source code of the host is also envisaged (the form of the OpenCL C kernel will determine the character of the OpenCL host API).

The purpose of the Agoge project is to accelerate and facilitate work with the OpenCL technology, which will allow the user to focus on developing parallel algorithms in OpenCL C and to prepare a computational procedure in an unobstructed creative way (less complex, if possible).

An indirect goal of the project was to become more familiar with OpenCL technology and to gain more experience in the creation of development environments [5].

2. IMPLEMENTATION

The Agoge development environment was created in Pascal (Delphi Pascal dialect) and Delphi XE development environment version 10. Pascal is in this case the OpenCL host, but the user is not obliged to be familiar with that programming language. In the Agoge environment the actions for which the host is responsible are automated or provided in the form of simple editors (for example, a list for selecting devices on which parallel calculations will be performed) and are broken down into the following stages:

- creating an object representing the platform and computation device;
- debugging and compilation of the source code into binary form;
- preparation of the context (creation of a data buffer, loading images from files, etc.);

3 The computation context may include: specification of the OpenCL operating environment (e.g. type of computing device), data preparation (e.g. creation of a memory buffer for the computational grid), creation of a command queue (e.g. transfer of data to a computing device) [1].

4 The programming model in OpenCL consists of data model (computational grid addresses data) and task model (command queue). In the task model it is still possible to carry out parallel computation, the kernel, however, uses only one work-item within the work-group [1].

5 SDK (*Software Development Kit*) - a set of software development tools required to create applications that make use of a specific library.

6 It is not possible to create an OpenCL context in OpenCL C. Another programming language (which uses OpenCL API), called OpenCL host, is responsible for the context. In most cases, the source code of the kernel is edited in the OpenCL host environment which is not dedicated to the OpenCL C language.

- preparation of arguments (parameters) and transferring them to the kernel;
- creation of a command queue and running the kernel;
- collecting results;
- presentation of results;
- freeing memory.

2.1. Source code editor and programming model

Agoge has all the features that a good source code editor needs (syntax highlighting, code hinting and completion, text search, list of recent files, etc.), and an integrated development environment (debugging[7] with indication of error source in the source code; code compilation, program execution). Figure 1 shows the main screen which presents an example of environment behaviour when an error in the source code is detected.

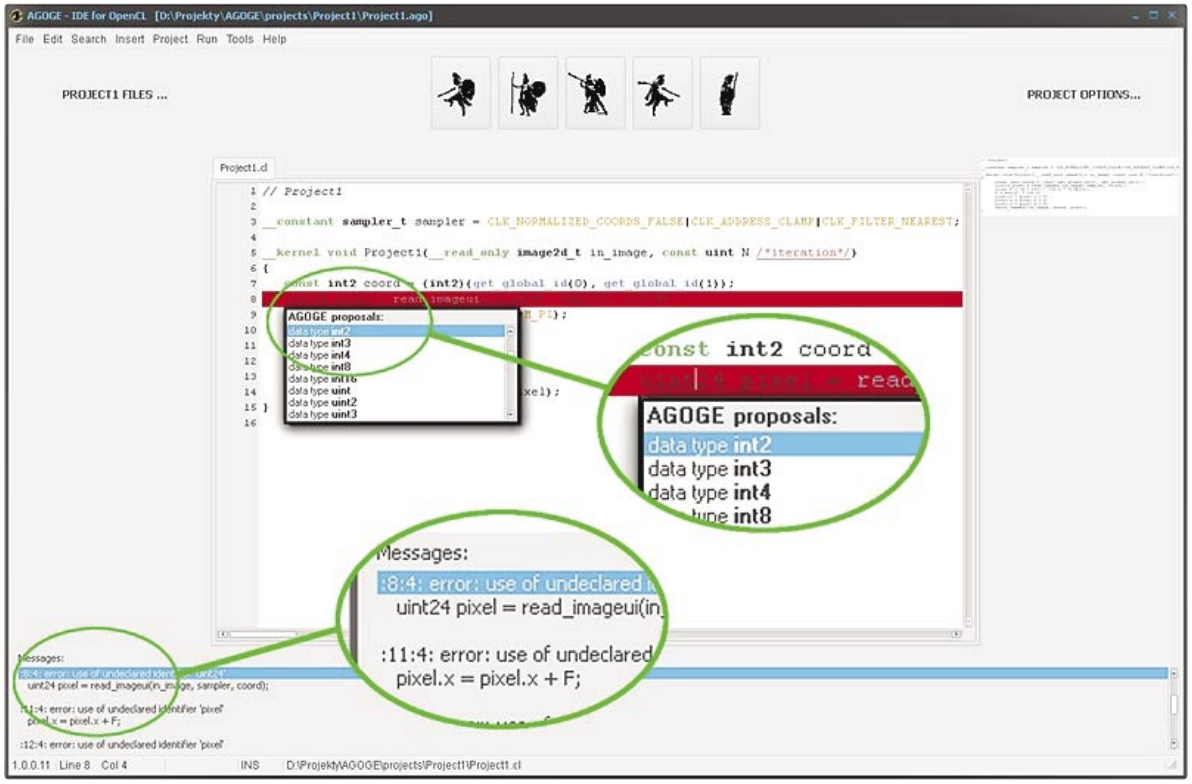

Fig. 1. Main screen of the Agoge environment showing an example of response to an error encountered in the source code (along with hints)

A programming model based on a computational grid (indexed space) does not allow for task allocation (task model prepared in OpenCL API), but these tasks do not use parallel processing [1], and are therefore not dealt with in more detail here. The programming model in the Agoge environment is focused on parallel processing with the use of a computational grid.

7 Debugging and compilation of the OpenCL C source code is effected by means of OpenCL-compatible drivers of the hardware that performs calculations. When an error is encountered, the OpenCL API returns the error message (tests performed on three different devices, two of which were of the same make, showed lack of standards for this type of messages).

2.2. Arguments of the computation procedure

Creating an OpenCL context based on arguments provided by the OpenCL C kernel requires the user to press the *Create Arguments...* (or *Update Arguments*[8]) button in options of the project in the Agoge environment (Fig. 2).

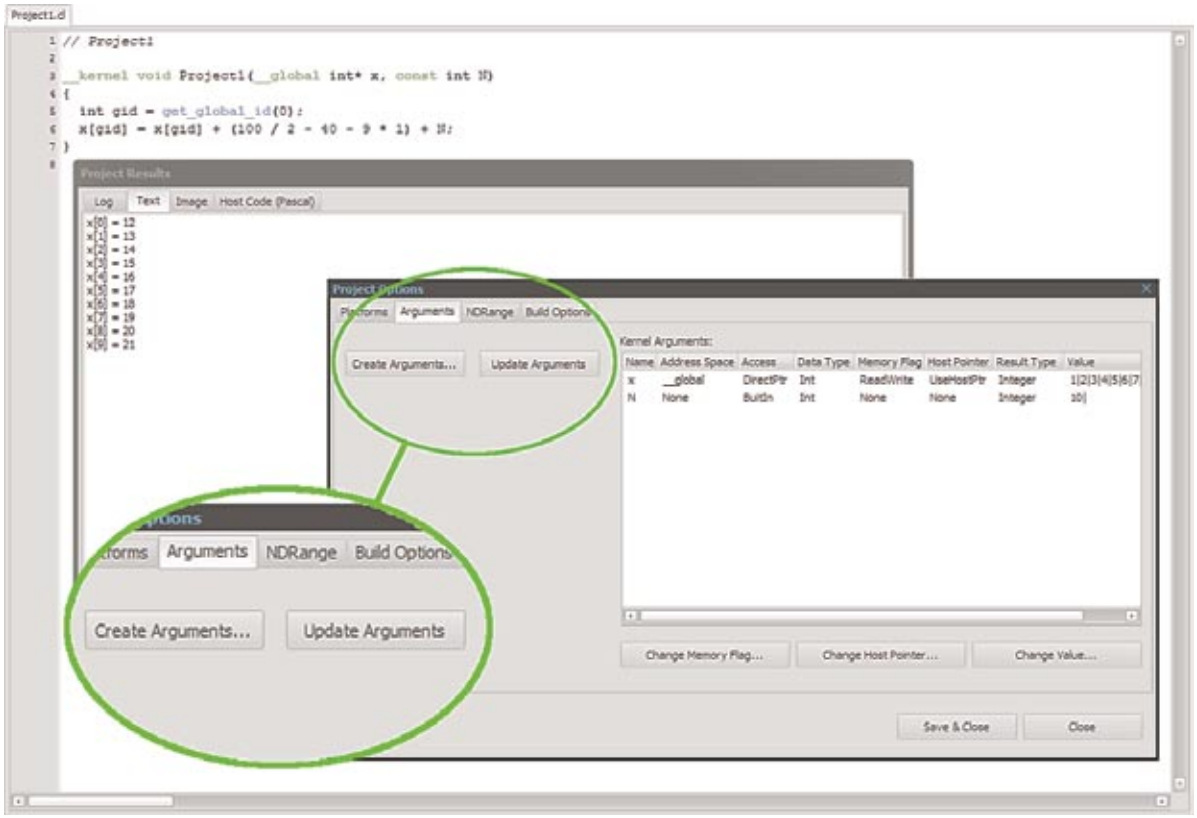

Fig. 2. An illustrative set of windows with results and project options (Arguments tab) after execution of the computation procedure (the Kernel Arguments list was created automatically)

Lexical analysis of kernel arguments in the OpenCL C code enabled the creation of a list of arguments with default values for the address space of the argument (*Address Space*), data access (*Access*), data type (*Data Type*), memory flag (*Memory Flag*), type of host pointer (*Host Pointer*), type of result data[9] (*Result Type*), data entered or read from the kernel (*Value*). The user can change the memory flag, type of host pointer and data (*Value*). Figure 3 shows examples of windows for changing data values for the *int* type (scalar value) and the

8 The difference between *Create Arguments ...* and *Update Arguments* is that the creation of arguments (*Create Arguments ...*) fills in the fields in the argument list with default values, while *Update Arguments* modifies the list of arguments without interfering with the changes made by the user in the argument list - it is recommended to use the *Update Arguments ...* button after each modification of the arguments in the OpenCL C kernel (it is assumed that the *Update Arguments ...* function will be automatically run after each change of arguments in the source code in subsequent versions of the environment).

9 The result data type and data, as opposed to the preceding fields, are parameters specific to the Agoge environment - their presence is not determined by OpenCL API, but by the environment itself, which requires entering data into arguments and determining appropriate presentation of results. The form of presented results depends on the result type (text, number, image) and is determined automatically and cannot be changed.

image2d_t type (image) - *Change Value...* button in the project options.

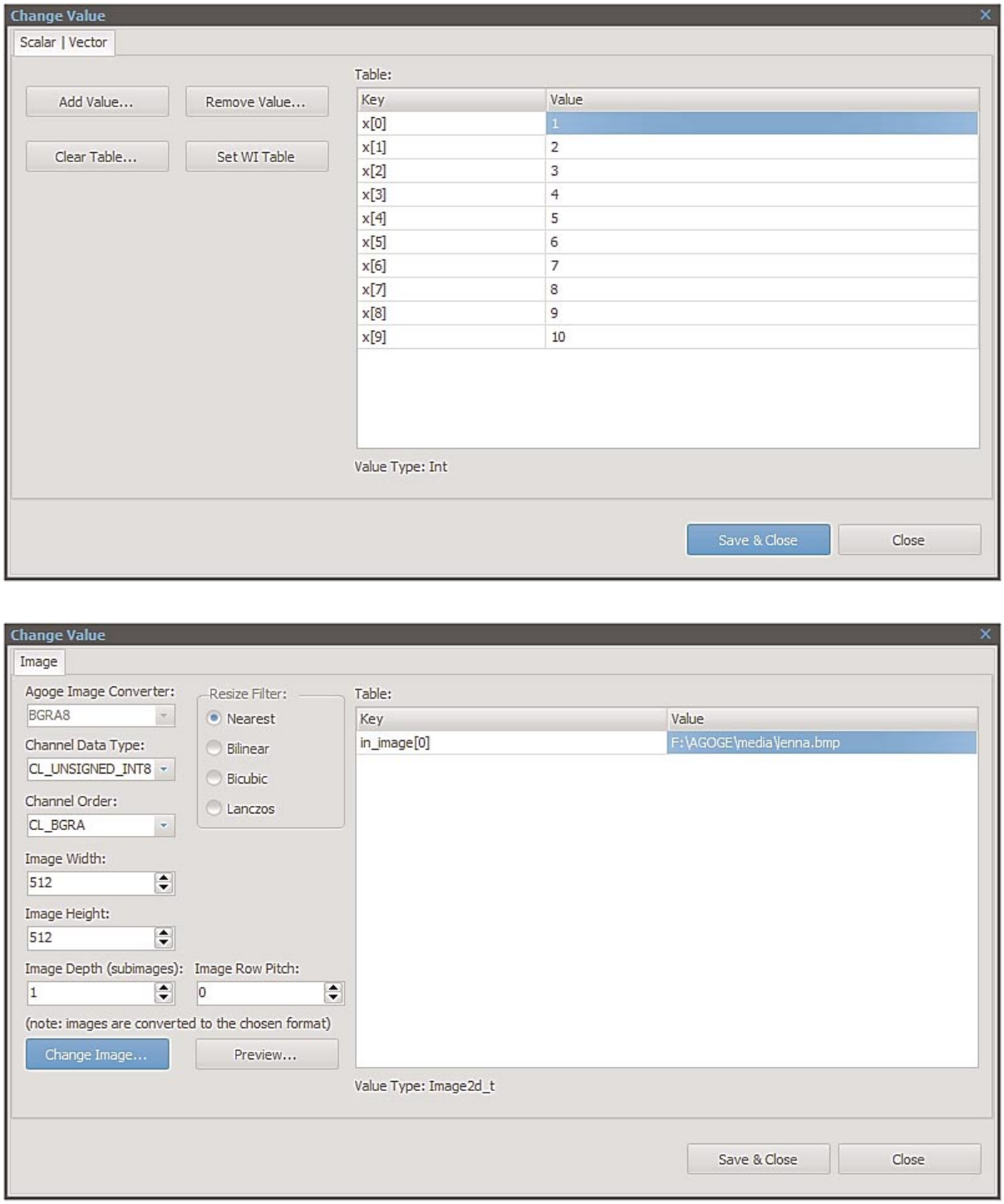

Fig. 3. Examples of windows for editing values for the Int and Image2d_t data types

The complete list of scalar and vector values that can be used in calculations and can be modified in the *Change Value* window (Fig. 3.) is presented in Table 1. Graphic types supported by Agoge are limited to two-dimensional images[10].

10 Support for one- and three-dimensional images is planned in future versions of the environment.

Table 1. OpenCL C data types handled by the kernel

Scalar type	Vector type	Equivalents in OpenCL API	Equivalents used in the Agoge environment (host: Pascal/Delphi)
char	charN[11]	cl_char, cl_charN	N-element table of ShortInt type
unsigned char, uchar	ucharN	cl_uchar, cl_ucharN	N-element table of Byte type
short	shortN	cl_short, cl_shortN	N-element table of SamllInt type
unsigned short, ushort	ushortN	cl_ushort, cl_ushortN	N-element table of Word type
int	intN	cl_int, cl_intN	N-element table of LongInt type
unsigned int, uint	uintN	cl_uint, cl_uintN	N-element table of LongWord type
long	longN	cl_long, cl_longN	N-element table of Int64 type
unsigned long, ulong	ulongN	cl_ulong, cl_ulongN	N-element table of UInt64 type
float	floatN	cl_float, cl_floatN	N-element table of Single type
double	doubleN	cl_double, cl_doubleN	N-element table of Double type

Two-dimensional images can be scaled at the environment level in accordance with the width and height values given in the *Change Value* window (Fig. 3). After changing a 2D image (graphics file selection window after clicking *Change Image ...*), the environment asks the user for scaling[12] - if there is no confirmation the image is loaded without scaling, while the image size values are set in the *Change Value* window according to the size of the loaded image[13]. Major graphics file formats supported by Agoge: BMP, JPEG, JPEG2000, PNG, MNG, PPM, GIF, TGA, etc.

2.3. Computational grid

The computational grid (referred to as NDRange, also called an index space) is the most important concept in OpenCL computation procedures. It determines the manner in which data are distributed among the computation cores and it may have a one-, two- or three-dimensional form [1]. Figure 4 shows a computational grid property editor in the options of the project effected in the Agoge environment.

11 Permissible N values: 2, 3, 4, 8, 16 (OpenCL C standard). Single-element tables were used in the host as an equivalent to scalar values (N=1). This enabled the application of the same mechanism to both scalar and vector values (N = 1, 2, 3, 4, 8 or 16).

12 Interpolation be means of the nearest neighbour method, bilinear interpolation, bicubic interpolation, Lanczos interpolation.

13 The parameters of a two-dimensional computational grid are also set by default.

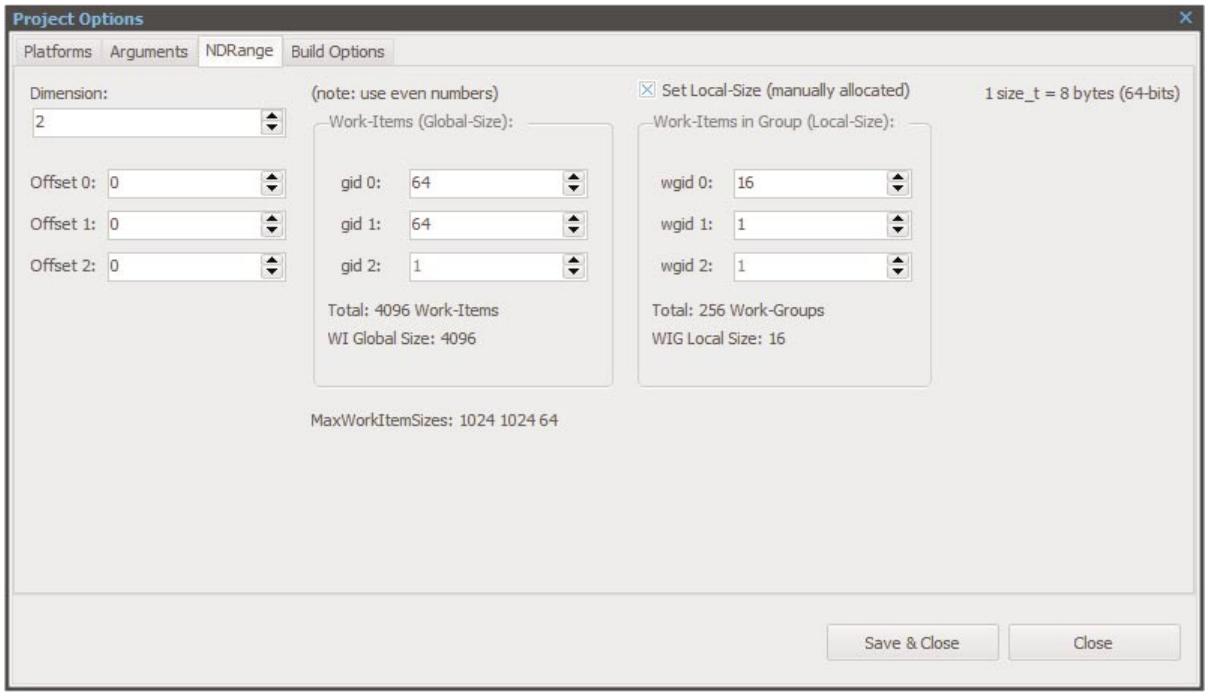

Fig. 4. Project options window in the Agoge environment (*NDRange* tab) enables changing the properties of the computational grid.

The article is not devoted to explaining the OpenCL architecture, and thus the classification of data in accordance to computational grid parameters will not be discussed in depth, that is in a way that allows for a clear understanding of the relations between the parameters of the computational grid and for determination of their values. For details of all issues of OpenCL technology readers are referred to literature [2], [3], [4].

The computational grid includes the following parameters: size of the global grid (*Work-Items*), size of local grid (*Work-Items in Group*), offset (*Offset*) and dimension (*Dimension*). Less experienced users should uncheck the *Set Local-Size* box, then the local grid size will be set automatically (by OpenCL drivers).

The window shown in Figure 4 defines a two-dimensional computational grid in which the total value of threads (work-items) performing one instance of the computation procedure is 4096 (64x64). Work-items are divided into 256 work-groups, where individual work-items can communicate by means of shared memory (also referred to as local memory). Maximum values of parameters should be adapted to the capabilities of the specific computation device. Parameters of the available hardware can be viewed on the *Platforms* tab in the project options, as shown in Figure 5.

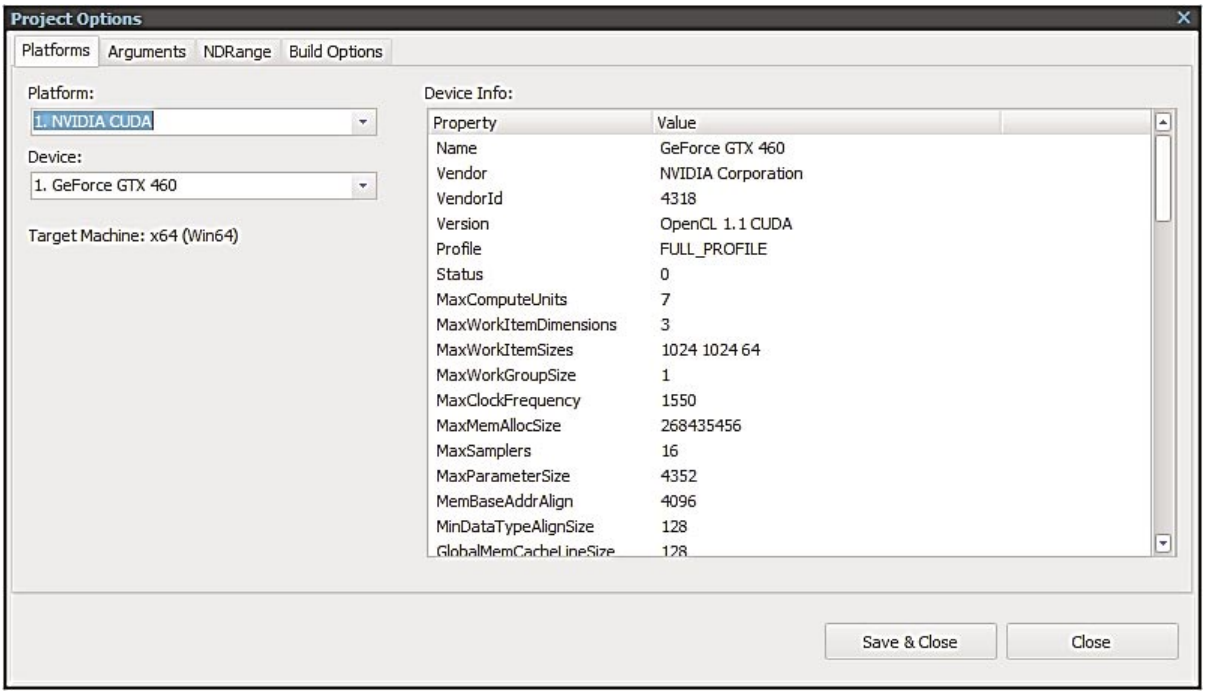

Fig. 5. Project Options window (Platforms tab), where computation device can be selected and its parameters can be viewed

2.4. Special types and compilation of the computation procedure

Special data types were introduced to fulfil the needs of the Agoge environment. The introduction of special types in the source code has no effect on the manner of running kernel in another potential environment (the kernel created in Agoge may be launched by other host). Adding a special type is effected by typing an appropriate lexeme when declaring an argument in the kernel, hidden in the comment[14], as shown in Figure 6. Currently, there are three lexemes implemented representing the special token of the Agoge environment:

- /*out*/,
- /*symbol*/,
- /*iteration*/.

The /*out*/ lexeme informs the OpenCL context creation system that the argument will be write-only (memory flag set to *WriteOnly*) and the default values will not be passed to the kernel[15]. The /*symbol*/ lexeme is reserved for the *uchar* and *char* data types (eight-bit integer representing one of 256 characters) - it enables entering and reading symbols (characters) of the ASCII code table (example in Figure 6). Explanation of the /*iteration*/ lexeme requires discussing the methods of running procedures, which is done further in this paper.

14 Agoge lexemes will be ignored by other hosts: they will be treated as meaningless comments.

15 If the */*out*/* lexeme is not entered, the memory flag is set to *ReadWrite*. The *ReadOnly* flag can be obtained by typing *const* after declaring the global address space *__global* (standard OpenCL C solution for read-only data).

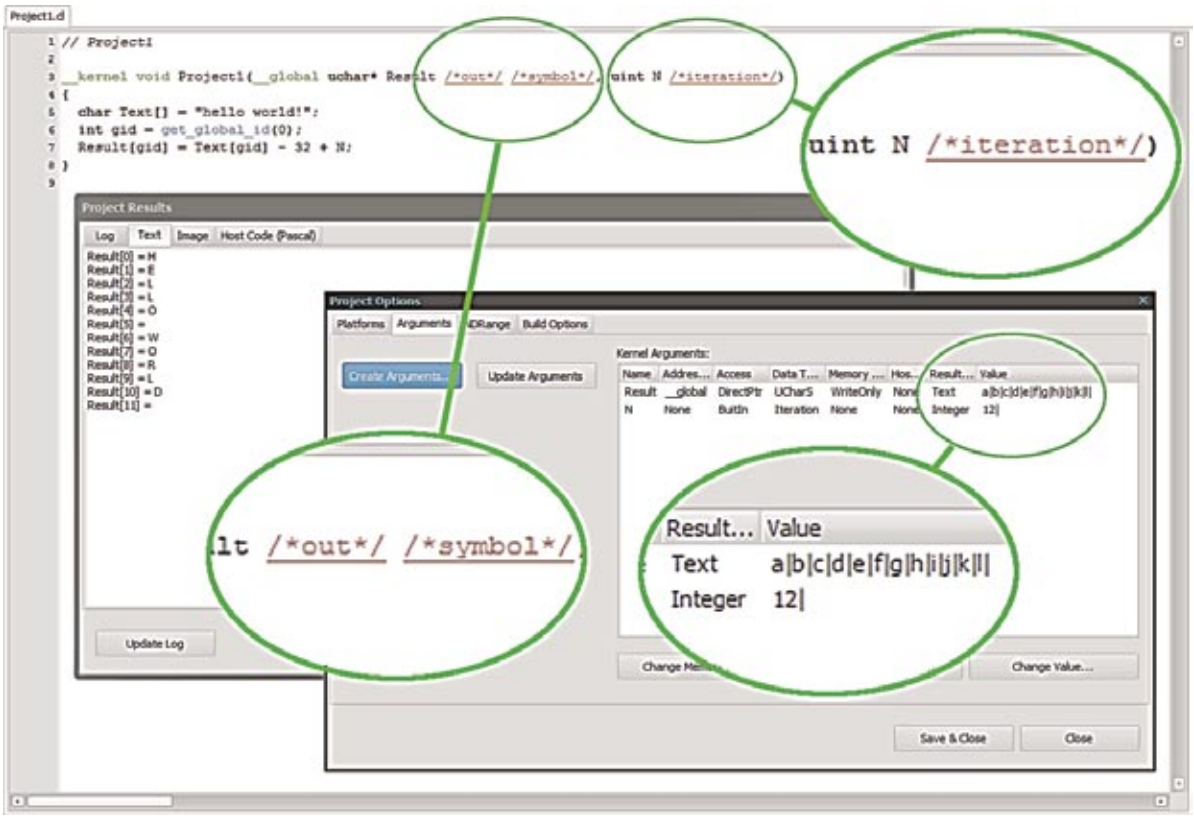

Fig. 6. An illustrative set of windows after executing an example in which special Agoge types are used (12 work-items on a unidimensional computational grid)

Buttons for running (starting) or stopping the computation procedure are located in the top part of the main window (Figs. 1 and 7). Use of the */*iteration*/* lexeme is justified when launching the computation procedure in the *Build and Run* mode. The value of the current iteration, starting from zero, is passed onto the argument - the results shown in Figure 6 (*Project Results* window) are the results for the first iteration (N=0). The */*iteration*/* lexeme can be used for *uint* type data.

Fig. 7. Control buttons: Build and Run; Stop; Build and Execute; Build Only; Close Application

3. AGOGE ENVIRONMENT IN PRACTICE

The Agoge development environment can be used to implement OpenCL parallel processing algorithms in numerical computations (Fig. 2), cryptography (Fig. 6), image

processing and analysis. This section is devoted to more advanced examples of using the environment[16].

OpenCL and environment tests were carried out on two stations, comprising, in total, three OpenCL platforms:

- Station 1:
  - Device: GeForce GTX 460
    - Platform: NVIDIA CUDA
    - OpenCL version: OpenCL 1.1 CUDA
- Station 2 (two platforms on one computer):
  - Device 1: GeForce GTX 680
    - Platform: NVIDIA CUDA
    - OpenCL version: OpenCL 1.1 CUDA
  - Device 2: Intel Core i7-3930K CPU @ 3.20GHz
    - Platform: Intel OpenCL
    - OpenCL version: OpenCL 1.2

Figure 8 shows the test graphics [6].

Fig. 8. Lena at 512x512 px resolution

3.1. Image processing - example of image blur

Figure 8 shows the image blur source code (averaging filter based on Gaussian distribution with 5x5 matrix) with the computation procedure and result presentation. The average computation time on Station 2 (Device 1) was 188 μs. Information on the kernel execution time[17] is displayed in the bottom part of the Agoge window.

16 A more in-depth justification for using the Agoge environment and OpenCL, along with professional examples, will be presented in future articles on subsequent projects implemented by the OBRUM Simulator Department.

17 It is important to note that the specified execution time of the computation procedure is not the time between pressing the *Build and Execute* button and acquiring the results - it is the time of the OpenCL kernel execution only. The kernel execution time does not take into account the time of OpenCL context creation

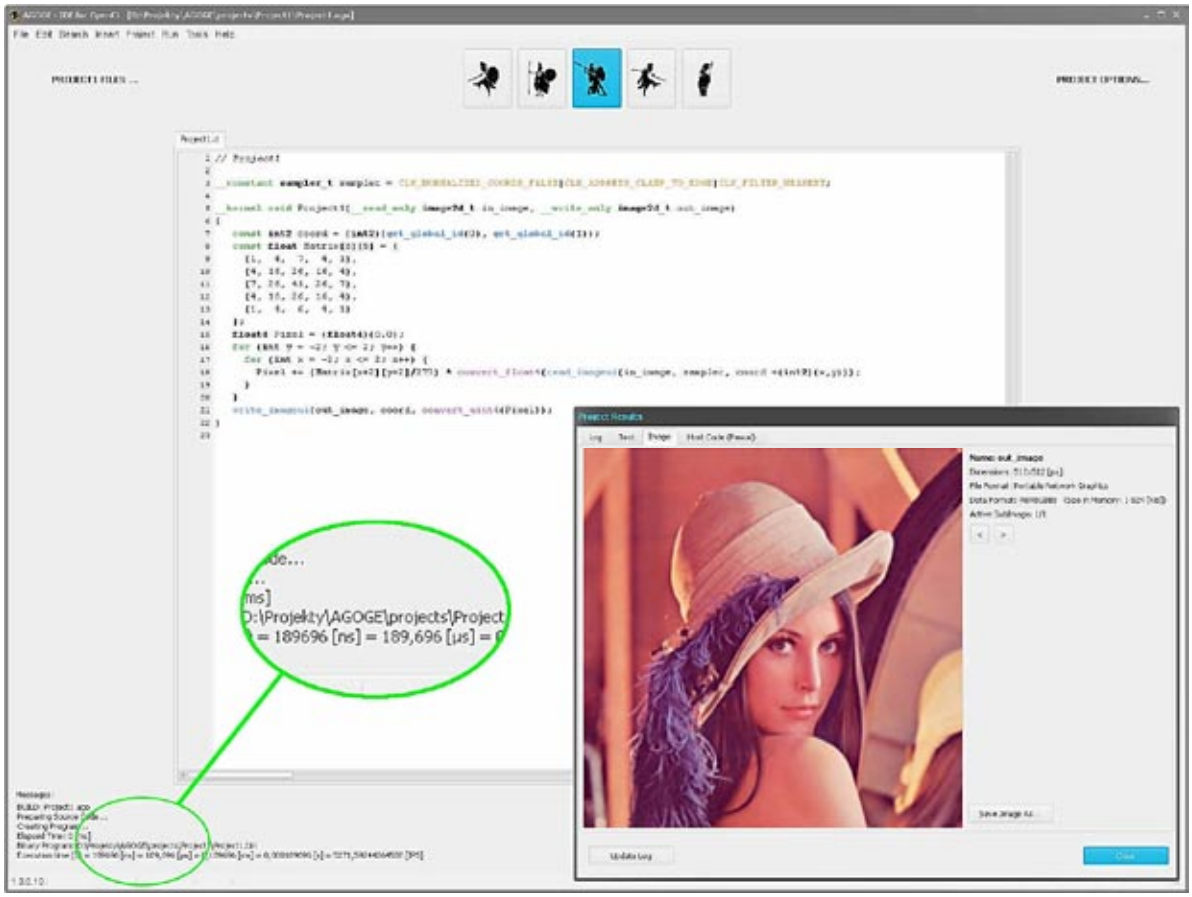

Fig. 9. Image blurring using the Agoge environment

In the case of Device 2 (Station 1), the computation procedure was not executed (the OpenCL driver returned an error code of -10) and the Agoge event log recorded the message: *image format not supported*[18]. The event log of the environment can be viewed in the Log tab in the *Project Results* window or in the AGOGE.log file in the main folder of the application.

3.2. Image processing - animation

Animation of the change of image brightness (change of results over time, presented in the *Project Results* window using double buffering[19]) can be obtained by adding the current iteration value (/**iteration*/) to the kernel and running the program in a loop (Figure 10). The example source code determines the amplification and attenuation parameters of the RGB channels based on the sinus function curve. The example is trivial, nevertheless it shows how the /**iteration*/ lexeme can be used in the Agoge environment.

The average computation time (280 iterations) on Station 2 (Device 1) was 51 μs.

which may include loading an image from a file, transfer of data to the kernel, and preparation of results presentation (drawing an image in the *Project Results* window).

18 Status of the invoking of OpenCL API methods (error codes) is described in English.

19 Double buffering is a technique that eliminates image flickering.

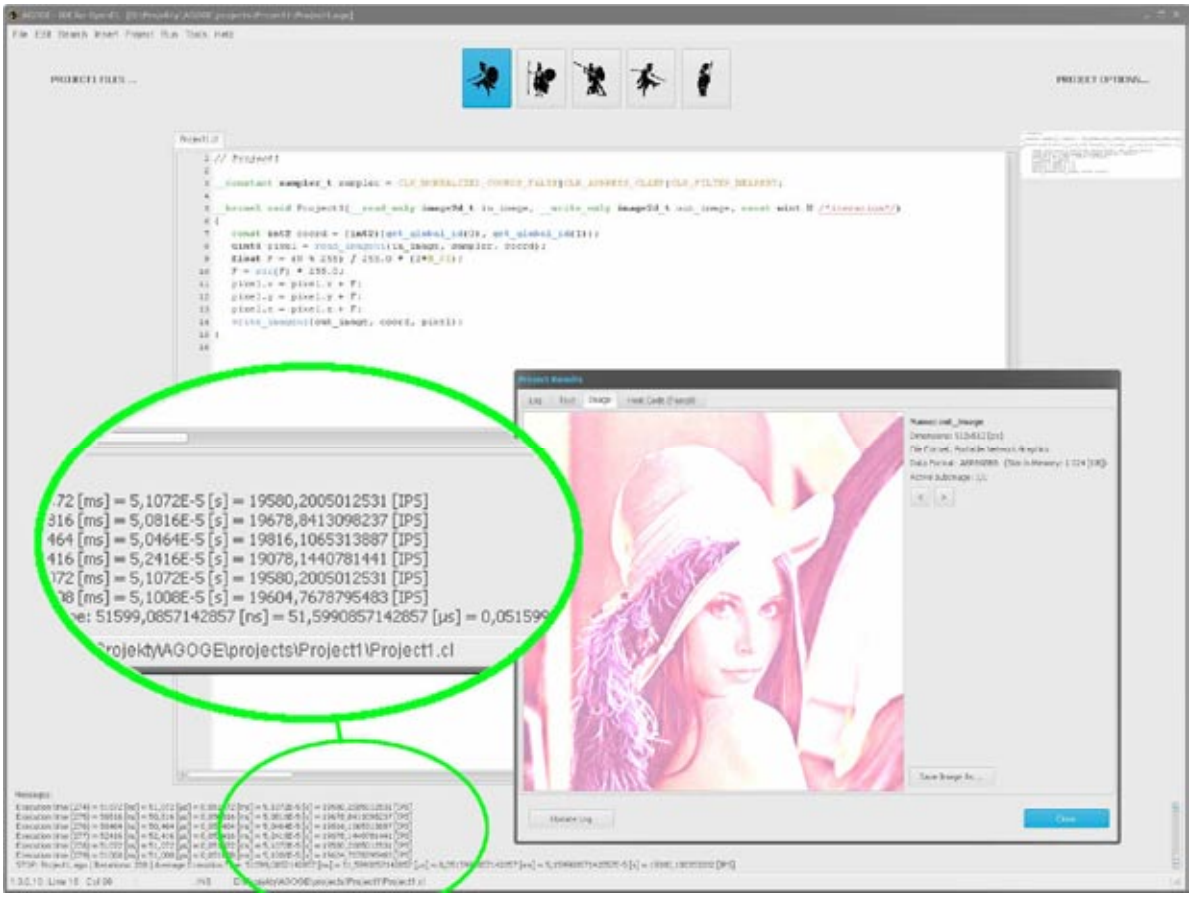

Fig. 10. Change of colour brightness running in a loop (*Build and Run*)

Of course, it is possible to count successive iterations in the kernel itself (Fig. 11), but one must remember the number of code calls that depend on the size of the computational grid.

Project2.cl

```
__kernel void Project2(__global uint * result)
{
  int gid = get_global_id(0);
  if (gid == 0) {
    result[gid] = result[gid] + 1;
  }
}
```

Fig. 11. OpenCL C source code for calculating subsequent iterations

3.3. Numerical calculations – Mandelbrot set

The Mandelbrot set [7] constitutes an example with which the computational capabilities of OpenCL can be demonstrated in a clear and fast manner (Fig. 12).

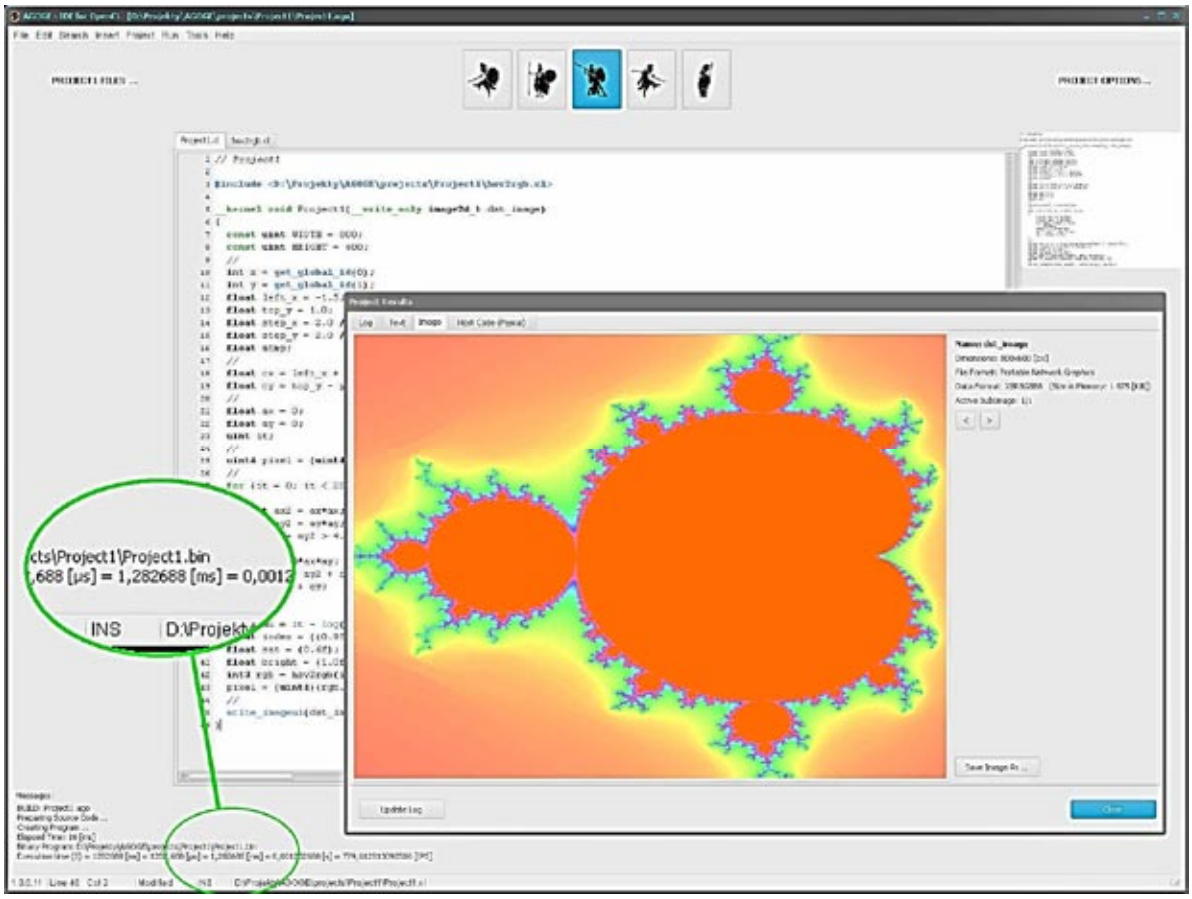

Fig. 12. Mandelbrot set in the Agoge environment

The advantages of using parallel processing with OpenCL are not discussed here, but the following list is a clear proof of the significant acceleration of computation.

- Hardware benchmarking with the application of parallel processing:
  - Station 2, Device 1: 1.3 ms,
  - Station 2, Device 2: 13 ms,
  - Station 1, Device 1: 2.8 ms,
- Benchmark of the drawing algorithm of a Mandelbrot set using OpenCL with no parallel computing:
  - algorithm with parallel processing[20]: 5.7 ms,
  - series algorithm with no parallel computing[21]: 390 ms,

3.4. Numerical calculations – Julia set

The algorithm for drawing Julia set [9], [10] is similar to the implementation of the Mandelbrot set of the previous example, but this time the height and width of the fractal were specified by defining the preprocessor constants (*pp_width, pp_height*) in the *Build Options* tab (Fig. 13). The example also demonstrates the ability to change the standard OpenCL options in the Agoge environment.

20 Arithmetic mean of hardware tests.

21 Algorithm for drawing a Mandelbrot set [8] implemented in the host environment on Station 2 (no use of OpenCL and no fractal colouring).

The average execution time of the computation procedure with the *-cl-fast-relaxed-math* optimisation enabled on Station 1 (Device 1) was 1.27 ms. The average execution time with default OpenCL settings (no optimisation) on the same Station was 1.31 ms.

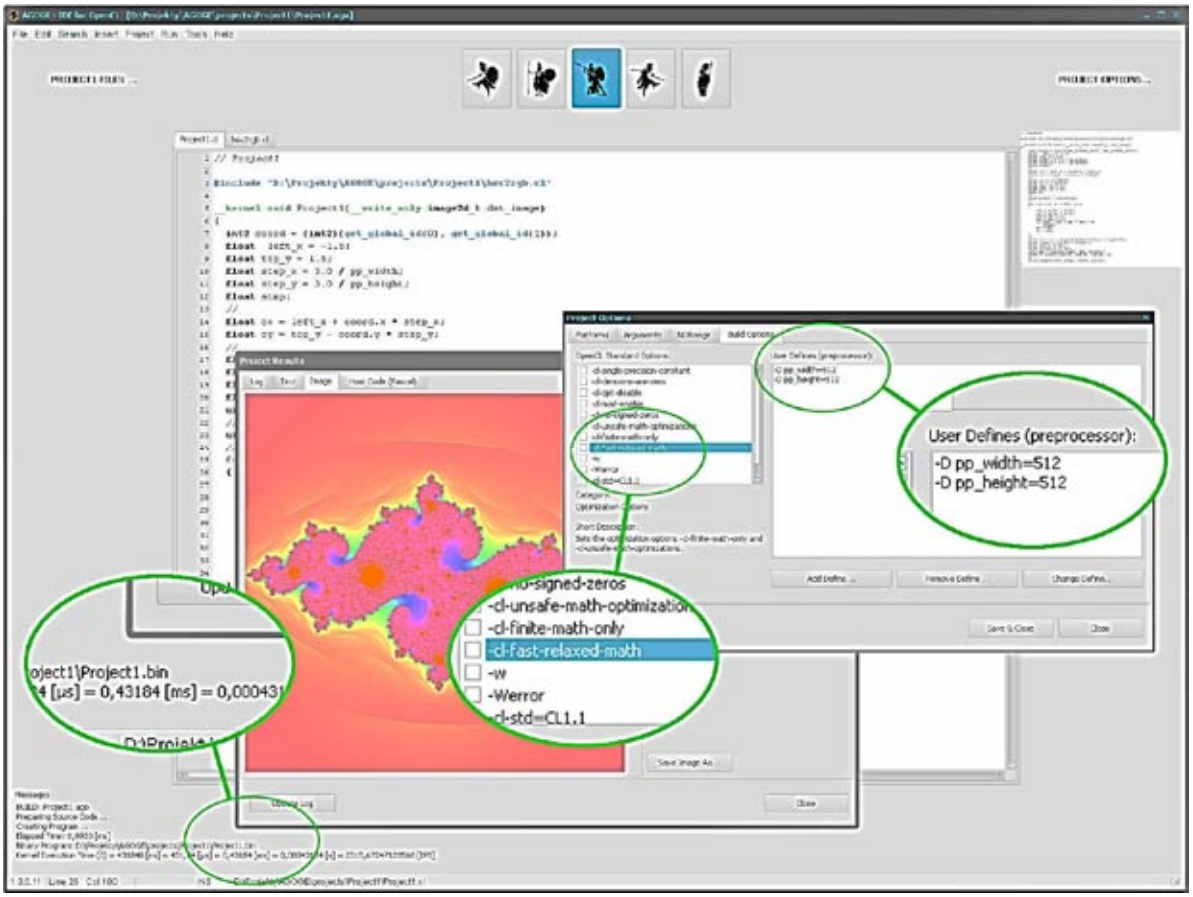

Fig. 13. Julia set

4. CONCLUSIONS

The examples presented in this article may not fully reflect the potential of OpenCL technology, but their main purpose is to present an opportunity to use the current version of the Agoge development environment to develop powerful OpenCL C algorithms and demonstrate the efficiency of the environment itself. Knowledge of the OpenCL API and the implementation of the source code of the host is most appropriate for potential programmers wishing to use OpenCL. However, the constant repetition of actions associated with the handling and resources of the kernel (when using the OpenCL technology on its own, without a dedicated environment) is impractical, cumbersome and prone to generate many peripheral errors[22]. The effort that would have to be expended to develop a source code for the host, for instance for the examples presented here, is much greater than that put in the development of algorithms in OpenCL C.

22 The author is referring to errors that are not related to the OpenCL API itself or to the OpenCL C source code, but to errors that result from, for instance, the need to use additional sources for decoding different graphics file formats, or just to ordinary errors (the amount of memory declared after changing the computational grid parameters, etc.). .

OpenCL and Agoge can be very useful in implementing and performing efficient computations in any scientific area that can be described using parallel algorithms. Algorithms in which multiple instructions can be executed simultaneously are (or should be) applied in image processing and analysis, fluid and gas dynamics simulation, ray tracing, chemistry and computational biology, in finite elements methods and in many others.

The Agoge development environment is constantly evolving, and the direction of development depends on the tasks performed at OBRUM's Simulator Department. Plans for the near future include implementation of the option to introduce data structures into the kernel, generation of the source code for the host[23], implementation and integration with the OpenGL standard[24], support for one-dimensional and three-dimensional images.

5. REFERENCES

[1] Sawerwain M.: OpenCL – Akceleracja GPU w praktyce, PWN, Warszawa 2014, ISBN 978-83-01-18012-6.
[2] Khronos Group Inc.: The open standard for parallel programming of heterogeneous systems, https://www.khronos.org/opencl [Retrieved: 25.09.2016].
[3] Khronos OpenCL Working Group: The OpenCL C Specification, https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf [Retrieved: 26.09.2016].
[4] PUC-RIO, Lua, https://www.lua.org/ [Retrieved: 11.10.2016].
[5] Bugała M.: Hydrogen – Środowisko programistyczne dla systemu Virtual BattleSpace, Szybkobieżne Pojazdy Gąsienicowe (43) no. 1/2017, pp. 55-67. ISSN 0860-8369. OBRUM sp. z o.o. Gliwice, January 2017.
[6] USC, The USC-SIPI Image Database, http://sipi.usc.edu/database/ [Retrieved: 29.09.2016].
[7] Mandelbrot B.: The Fractal Geometry of Nature, W. H. Freeman and Company, New York 1982, ISBN 0-7167-1186-9.
[8] Rosetta Code, Mandelbrot Set, https://rosettacode.org/wiki/Mandelbrot_set [Retrieved: 10.10.2016].
[9] Rosetta Code, Julia Set, https://rosettacode.org/wiki/Julia_set [Retrieved: 11.10.2016].
[10] Julia G.: Mémoire sur l'itération des fonctions rationnelles, Journal de mathématiques pures et appliquées 8e série, tome 1 (1918), p. 47-246., http://sites.mathdoc.fr/JMPA/PDF/JMPA_1918_8_1_A2_0.pdf [Retrieved: 11.10.2016].

23 The generated host source codes will be saved as functions that can be attached to and used in more complex applications. The amount of host source code required to complete the OpenCL computation procedure may be several times greater than that of the procedure code itself, and for this reason, the acceleration of this process in the Agoge environment (automatic creation of the OpenCL context) is justified.

24 The main reason being making the results presentation more attractive and faster, rather than increasing the computational capacity.