HES-SO // MASTER

MSE | MASTER OF SCIENCE IN ENGINEERING

L

Projet d'approfondissement,
orientation Technologies de l'information et de la communication (TIC)

GPGPU: A look into OpenCL

rédigé par

Frédéric Dubouchet

Sous la direction de

Prof. Paul Albuquerque

de la MRU TIC de l'hepia

Le 8 Juin 2012

Abstract

In the search for better performance in general computing the Graphics Processing Unit (GPU) offers an attractive alternative to custom made hardware, as it is both affordable and standard. The only point that was missing was portability and openness of the software interface. OpenCL (Open Computing Language) is the solution to this problem. As an open standard it allows us to use any GPU, Central Processing Unit (CPU) or custom computing hardware, with the same code. In this paper we will look into this technology by implementing various algorithm and comparing performances with CPU and an other software platform, CUDA, which is the NVIDIA proprietary platform for general purpose GPU programming, and one of the mostly used interfaces.

Introduction

General purpose programming on graphical processors is a new field with a growing community of practitioners. Until recently only proprietary interfaces existed to harness the power of these chips. With the arrival of the Open Computing Language (OpenCL) a new open interface has appeared, and with it a hope for a unified, simple and portable framework for general purpose computing on heterogeneous hardware.

Challenges

OpenCL a quick overview

Definition

The Open Computing Language (OpenCL) is the open standard for General Purpose Graphical Processing Unit programming (GPGPU programming). It is maintained by the Khronos group, and was initially proposed by Apple. Many companies of the industry are members of the OpenCL Working group:

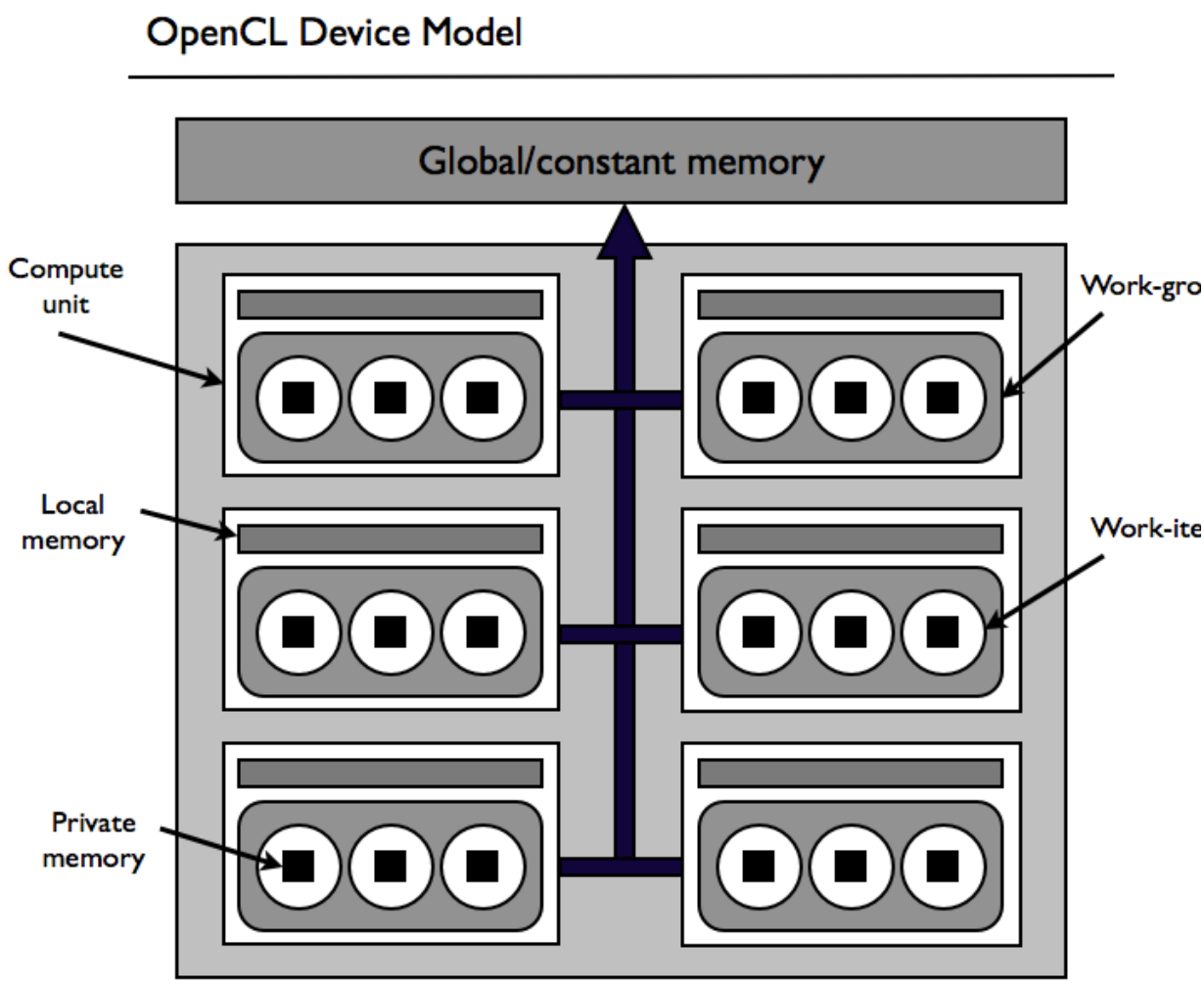
Altera, AMD, Apple, ARM, Broadcom, Codeplay, DMP, EA, Ericsson, Fixstars, Freescale, Hi corp, IBM, Intel, Imagination Technologies, Kestrel Institute, Kishonti, Los Alamos, Motorola, Movidius, Multicoreware, Nokia, NVIDIA, OpenEye, Presagis, Qualcomm, Rightware, Samsung, ST, Symbio, Texas Instruments, The University of West Australia, Vivante and Xilinx.

"OpenCL™ is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices. OpenCL (Open Computing Language) greatly improves speed and responsiveness for a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software."

"The Khronos Group is a not for profit industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics, dynamic media, computer vision and sensor processing on a wide variety of platforms and devices. All Khronos members are able to contribute to the development of Khronos API specifications, are empowered to vote at various stages before public deployment, and are able to accelerate the delivery of their cutting-edge 3D platforms and applications through early access to specification drafts and conformance tests."

Architecture

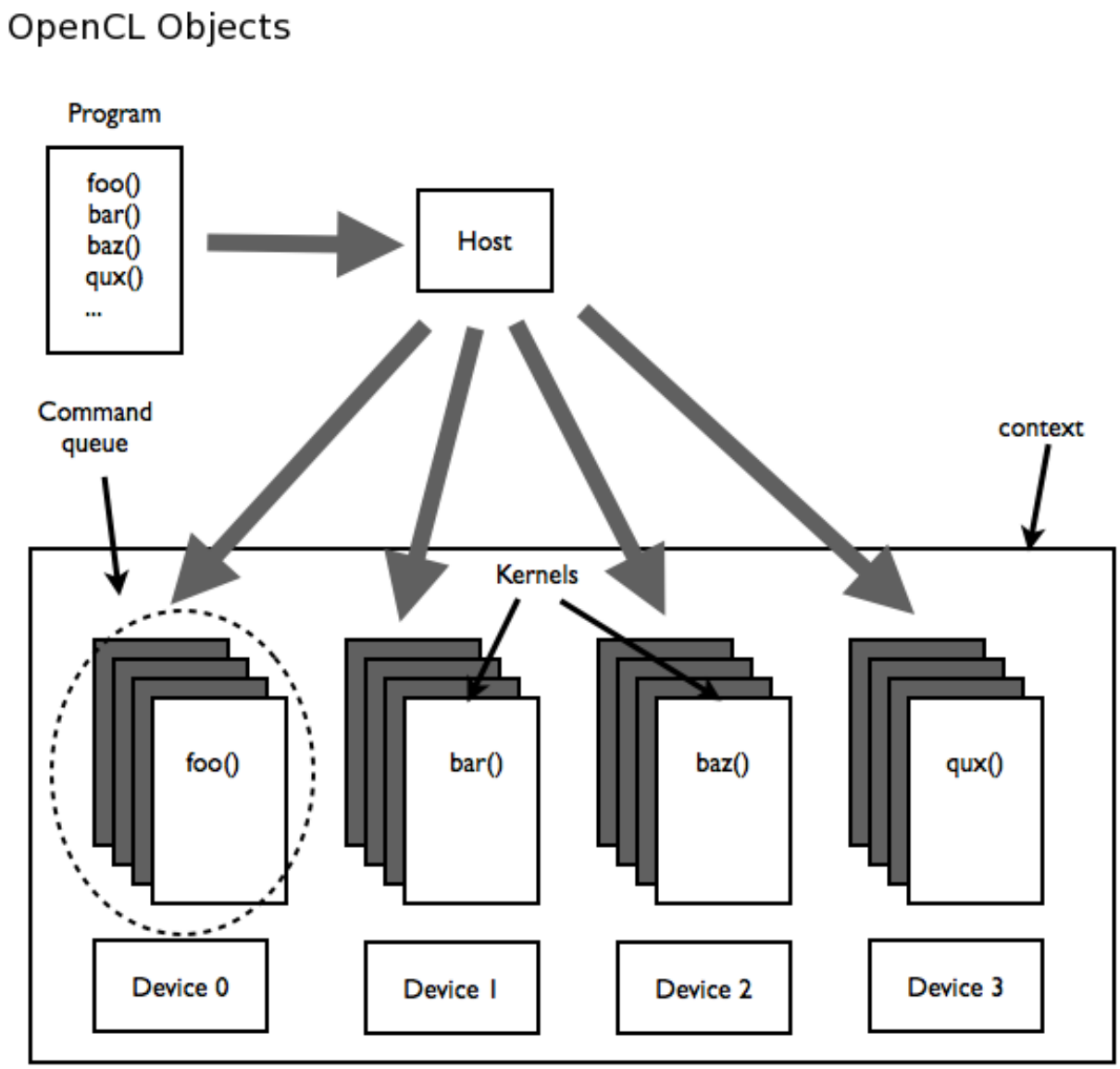
OpenCL is not really a language but an Application Programming Interface (API) and a kernel language derived from C99 that is compiled for, and executed on the device itself. It defines an abstract hardware architecture onto which the real hardware is mapped. Every piece of hardware (CPU, GPU, others) is called a device. This device is divided into work groups, each of which is further subdivided into work items.



API

The OpenCL API is composed of two parts: The host code that runs on the CPU which runs kernel handling event synchronization and manages the memory buffer, and the device code, which is

a kernel language based on the C99 specification.

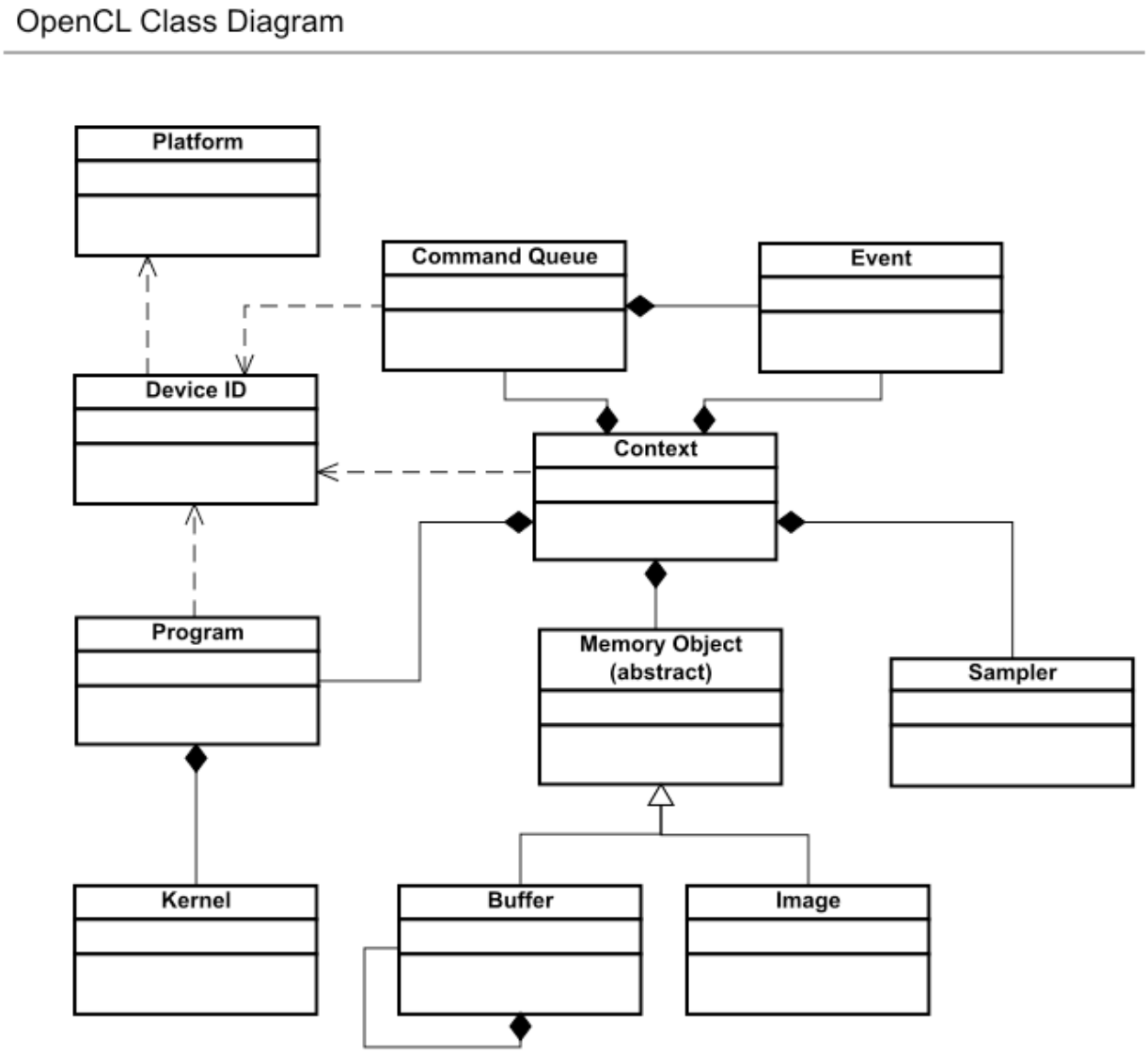


The host API is written in C but there is a C++ interface provided by Khronos, which is the interface used in this paper. Several other wrappers for other languages are also avaiable, such as Java, C#, Python, and even javascript with something called webCL (also specified by the Khronos group).

The device code the OpenCL kernel code is based on is C99, adding some extensions like vectorization, image access, and linear algebra functions.

The workload is itself specified in terms of *context*, *program* and *command queues*. The *context* contains the parameters of the current problem and manages the devices available for computation, as well as the *programs* and the *command queues*. The *program* consists of the set of kernels which are constructed from OpenCL code. The *context* is responsible for dispatching these kernels into *command queues*. Each *command queue* is a set of commands that can be executed in any order, or

simultaneously. A command queue is limited to a single device.



Events fired by devices are also available for triggering specific actions in case a finer synchronization mechanism is needed. Once the device has finished, the host receives and processes the output data via the buffer API.

Simple example

A simple example were we take two vectors in input and deliver a vector in output by just multiplying the component together. This will demonstrate how simple OpenCL syntax is, how the code is separated into the two parts. The host code is done is C++ using the official interface and the device code is in OpenCL device language.

Simple.cpp

```
// simple OpenCL example

#include <iostream>
#include <fstream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>

const unsigned int vector_size = 1024;

int main(int ac, char** av) {
   try {
      // get the device (here the GPU)
      std::vector<cl::Platform> platforms;
      std::vector<cl::Device> devices_;
      cl::Platform::get(&platforms);
      platforms[0].getDevices(CL_DEVICE_TYPE_GPU, &devices_);
      cl_context_properties properties[] = {
         CL_CONTEXT_PLATFORM,
         (cl_context_properties)(platforms[0])(),
         0
      };
      cl::Context context_ = cl::Context(CL_DEVICE_TYPE_GPU,
properties);
      devices_ = context_.getInfo<CL_CONTEXT_DEVICES>();
      cl::CommandQueue queue_(context_, devices_[0]);
      // load the kernel code
      std::ifstream ifs("./simple.cl");
      std::string str(
            (std::istreambuf_iterator<char>(ifs)),
            std::istreambuf_iterator<char>());
      ifs.close();
      // compile
      cl::Program::Sources source(1, std::make_pair(str.c_str(),
str.size()));
      cl::Program program_(context_, source);
      program_.build(devices_);
      // create the kernel
      cl::Kernel kernel_(program_, "simple");
      // prepare the buffers
      std::vector<float> in1(vector_size);
      std::vector<float> in2(vector_size);
      for (std::vector<float>::iterator ite = in1.begin(); ite !=
in1.end(); ++ite)
         (*ite) = (float)random();
      for (std::vector<float>::iterator ite = in2.begin(); ite !=
in2.end(); ++ite)
         (*ite) = (float)random();
      cl::Buffer buf_in1_ = cl::Buffer(
            context_,
            CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
```

```
            sizeof(float) * vector_size,
            (void*)&in1);
        cl::Buffer buf_in2_ = cl::Buffer(
            context_,
            CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
            sizeof(float) * vector_size,
            (void*)&in2);
        cl::Buffer buf_out_ = cl::Buffer(
            context_,
            CL_MEM_WRITE_ONLY,
            sizeof(float) * vector_size);
        // set the arguments
        kernel_.setArg(0, buf_in1_);
        kernel_.setArg(1, buf_in2_);
        kernel_.setArg(2, buf_out_);
        // wait to the command queue to finish before proceeding
        queue_.finish();
        // run the kernel
        std::vector<float> out(vector_size);
        queue_.enqueueNDRangeKernel(
            kernel_,
            cl::NullRange,
            cl::NDRange(vector_size),
            cl::NullRange);
        queue_.finish();
        // get the result out
        queue_.enqueueReadBuffer(
            buf_out_,
            CL_TRUE,
            0,
            vector_size * sizeof(float),
            &out[0]);
        queue_.finish();
        std::cout << "Operation successfull" << std::endl;
    } catch (cl::Error& er) {
        std::cerr << "Exception(CL) : " << er.what() << std::endl;
    } catch (std::exception& ex) {
        std::cerr << "Exception(STL) : " << ex.what() << std::endl;
    }
    return 0;
}
```

Simple.cl

```
// very simple kernel
__kernel void simple(
        __global read_only float* in1,
        __global read_only float* in2,
        __global write_only float* out)
{
    const uint pos = get_global_id(0);
    out[pos] = in1[pos] * in2[pos];
}
```

Compilation

Compilation is down with a normal C/C++ compiler. During this paper clang/clang++ and gcc/g++ were used. The OpenCL kernel code itself is compiled at runtime with the OpenCL API, no external compiler is required. It is possible to compile the code in advance and load the binary executable for the device but this break the compatibility and make your binary only workable on a certain class of device.

Strength

As an open standard OpenCL is not tied to any of the hardware or software manufacturers, and is thereby portable to any platform that implement the standard. It is also usable on other devices, such as CPUs or accelerating cards, and is thus not limited to GPUs.

OpenCL is also cross compatible with OpenGL, a graphic language, which allows you to display buffers and textures directly on screen without having to pass by the CPU.

Other possibilities

CUDA

This is the most used interface for general purpose GPU programming. It was developed by NVIDIA and has been used for many years and is still very much in use. This only works on NVIDIA cards, and will not run on CPUs. Since it is a proprietary language, NVIDIA is in total control of the API and the development kit. It is interesting to note that NVIDIA is talking about releasing the interface of the compiler to academics.

The abstraction level in CUDA is less strong, that is, the language is closer to the architecture. This means it could potentially offer more optimization possibilities to the programmer. We will see that this is not a key element as the compiler is able to reach this level of optimization without having to compromise code readability.

CUDA has its kernel and GPU code directly embedded inside the C/C++ code. In OpenCL the code is separated from the C/C++ host code. This can lead to incompatibilities with C++ as valid C++ code will be flagged as invalid by the CUDA compiler. This problem does not exist for OpenCL as the compiler is a separate entity and the code is fed to it via the API. This design code from the fact that OpenCL can have many target that won't produce a different compiled code. This means that compiled OpenCL code is not portable across platforms and you should always keep the code uncompiled in the executable.

CUDA offers a full suite of libraries and tools that are presently not available in OpenCL. Among these are performance tuning tools and libraries for matrices and fast Fourier transforms. Recently many implementations of common algorithms have started to appear for OpenCL, and unlike with CUDA these are directly available to any computing device.

DirectCompute

This is the Microsoft interface, highly tied to the Windows platform. DirectCompute is part of the DirectX API, the game development tools for Windows. It works with DirectX 10 and 11 under Windows Vista and Windows 7.

DirectCompute shares a lot of concepts with both CUDA and OpenCL, and is presently working with both AMD and NVIDIA graphic cards.

Shader languages

Shaders are the steps that the graphic card has to make before rendering to the screen. There is mainly two steps the vertex shader, also called vertex code and the pixel shader, also called fragment code, vertex shader is manipulating the data as point in space, as vector and the pixel shader the final rendering as pixels on screen.

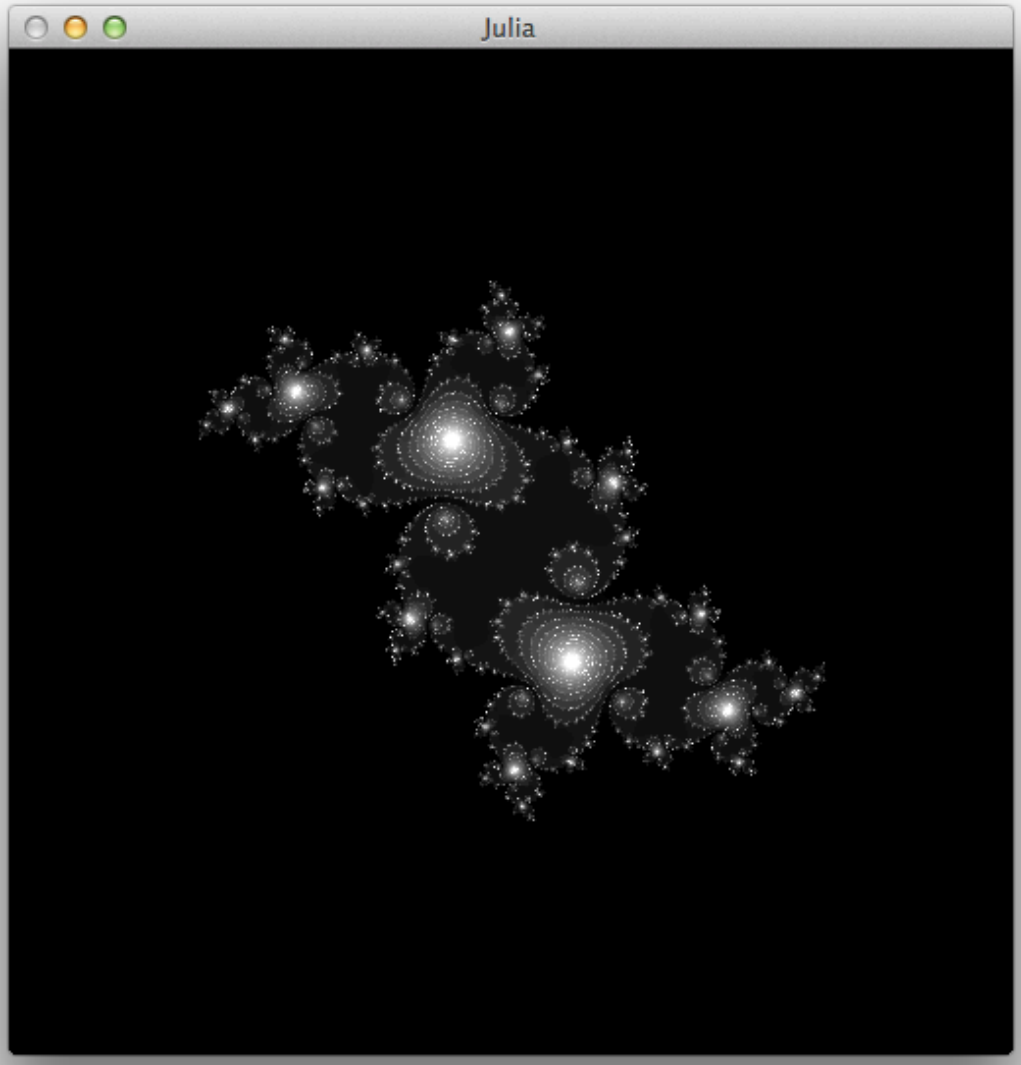
There are many different shader languages. The main ones are: Cg, the NVIDIA "neutral" shading language one of the first GPGPU language widely used. HLSL, the Microsoft version tightly coupled with DirectX, and therefore easier to use on a Microsoft platform (XNA). GLSL, the OpenGL version, that is supported widely but not implemented completely on all platforms.

Before the general purpose graphic processing languages appeared this was the only way to access the power of the graphic card, and this is still much in use today. As it was developed over many years, this is a very stable technology and there are many examples and libraries using theses languages.

Shader languages are oriented toward graphics programming and are therefore not well-adapted to some of the algorithms we could want to implement. They are made to handle vectors of four components: position, color, normal. This can be quite inconvenient. This is also their strength as this is what the GPU was originally built for. Communication between host and GPU tend to be messy at best, especially with non-graphical structures.

Julia set

The first experiment with OpenCL was to render fractals of Julia set type. To have a good overview on the result we implemented a full software version. Since it is possible with OpenCL, we also made some performance tests on the CPU with the same code as on the GPU.



Definition

The Julia set is a family of fractal objects in the complex plan. It consist of values with the property that all nearby values behave similarly under repeated iteration of the function. The Julia set function is:

$$f : \mathbb{C} \to \mathbb{C}$$

$$z \to z^2 + c$$

The output is proportional to the number of iteration made. Graphically this is presented in grayscale. The variable c is a complex number that can vary. In the software presented this is a parameter and it can be set at execution.

Implementation

The implementation is straight forward. We need a computing unit per pixel, for which we have to implement the function and some of the basic complex operations.

```
float2 square_c(float2 c) {
    return (float2)(c.x * c.x - c.y * c.y, 2.0f * c.y * c.x);
}

float2 add_c(float2 c1, float2 c2) {
    return (float2)(c1.x + c2.x, c1.y + c2.y);
}

float square_norm_c(float2 c) {
    return c.x * c.x + c.y * c.y;
}

__kernel void julia_buffer(
    __global float* out,
    float2 c,
    uint max_iteration)
{
    const uint2 d = (uint2)(get_global_id(0), get_global_id(1));
    const uint2 m = (uint2)(get_global_size(0), get_global_size(1));
    float2 pos = (float2)(
          (((float)d.x / (float)m.x) * 4.0f) - 2.0f,
          (((float)d.y / (float)m.y) * 4.0f) - 2.0f);
    uint ite = 0;
    while ((square_norm_c(pos) < 4.0f) && (ite < max_iteration)) {
          pos = add_c(square_c(pos), c);
          ite++;
    }
    float val = (float)ite / (float)max_iteration;
    out[d.y * m.x + d.x] = val;
}
```

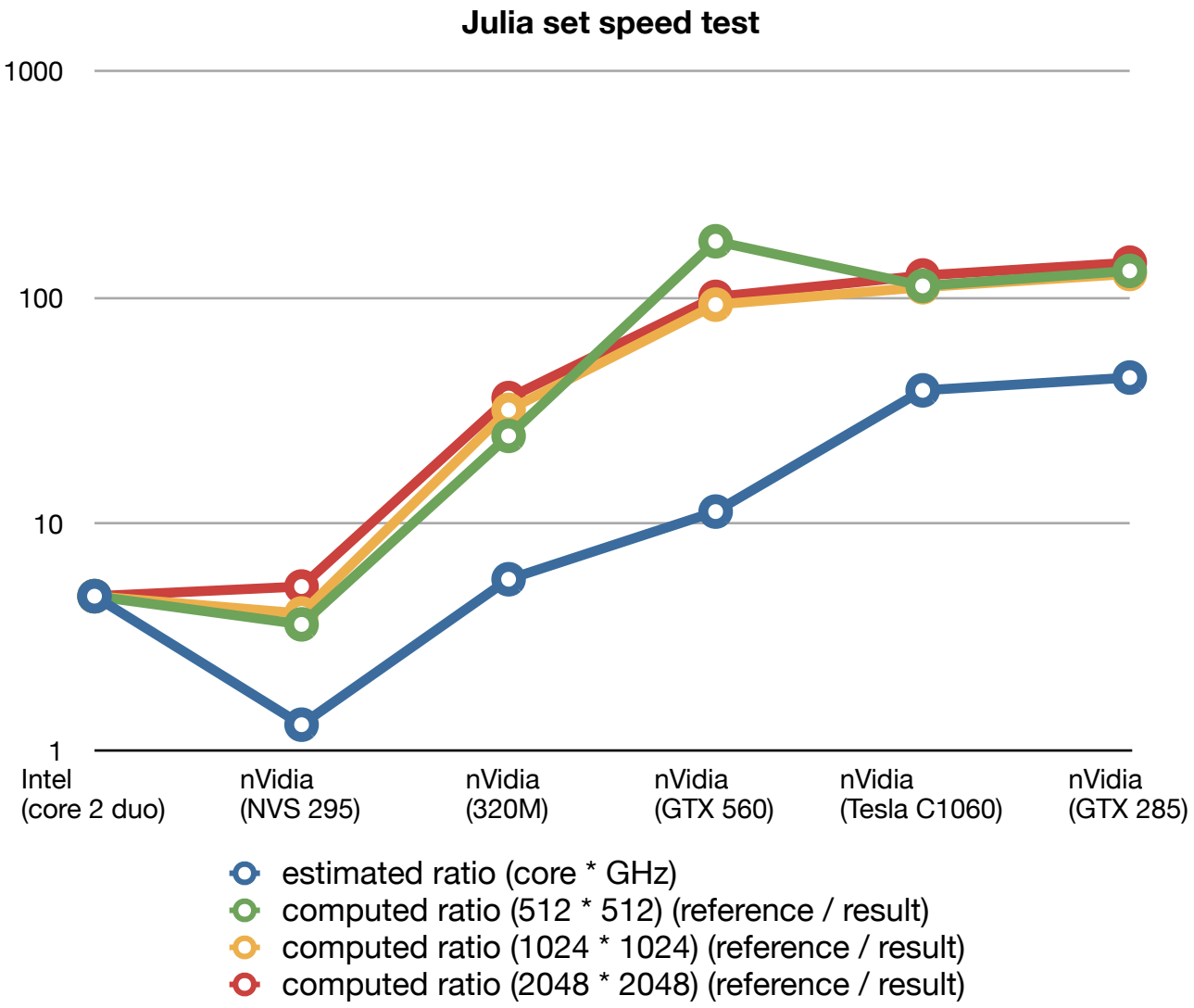
In OpenCL, types like float2, int4, etc. are vectorized version of the standard ones, and the associated operations work on them in a vectorized way.

Results

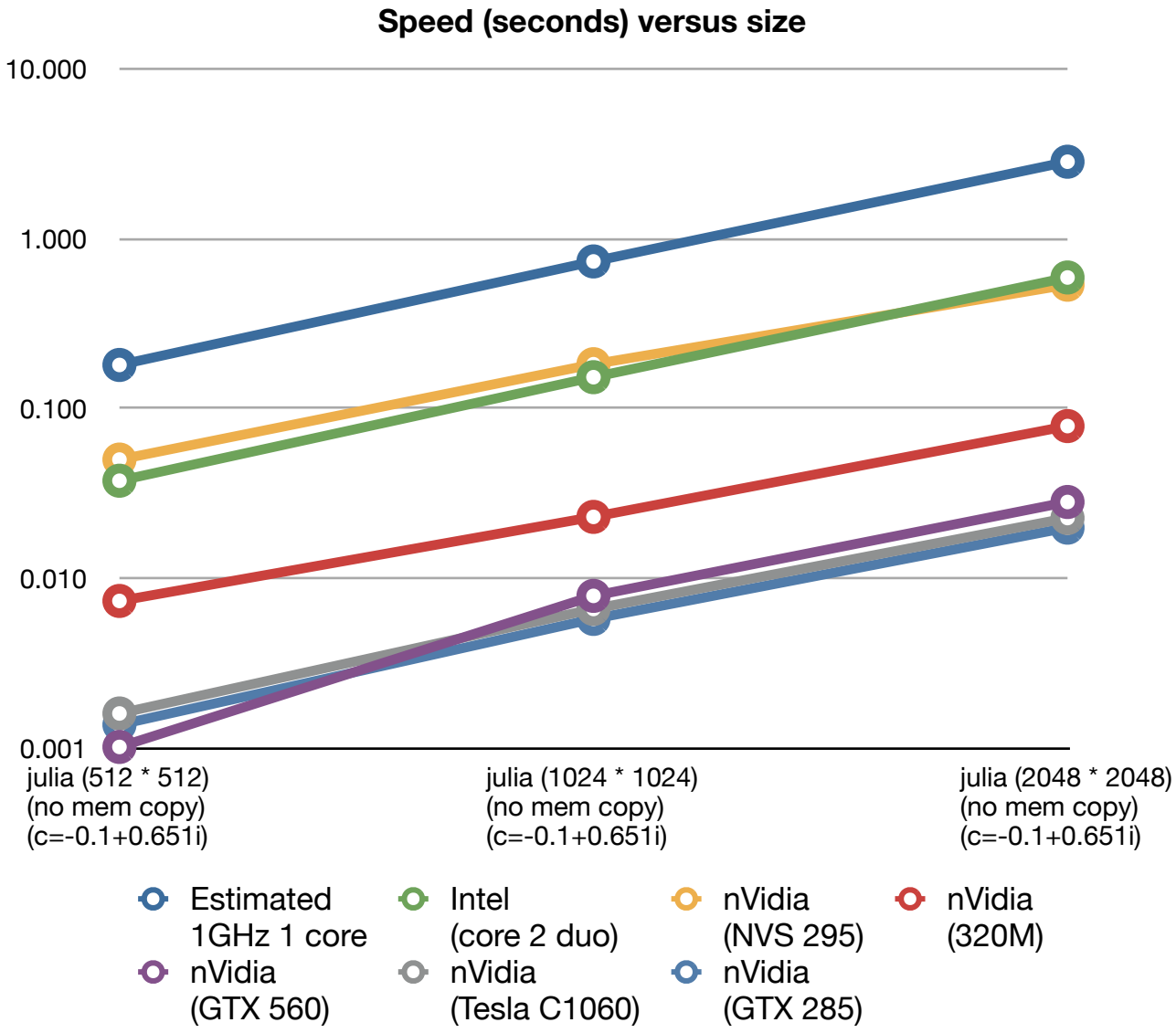
Calculating the ratio of processing unit frequency speed by the number of cores we can estimate the value of a theoretical 1GHz processor and then plot the different performances compared to this

processor. This shows that the Julia set, as it is a highly parallel algorithm, is very efficiently computed on GPUs, even if we normalize to the number of cores they have.

Time calculation where made with the boost library only taking into account the computing time and not the transfer of memory from the CPU to the GPU. The visualization uses GLUT and OpenGL so if you want to see the result in an actual window both need to be supported.



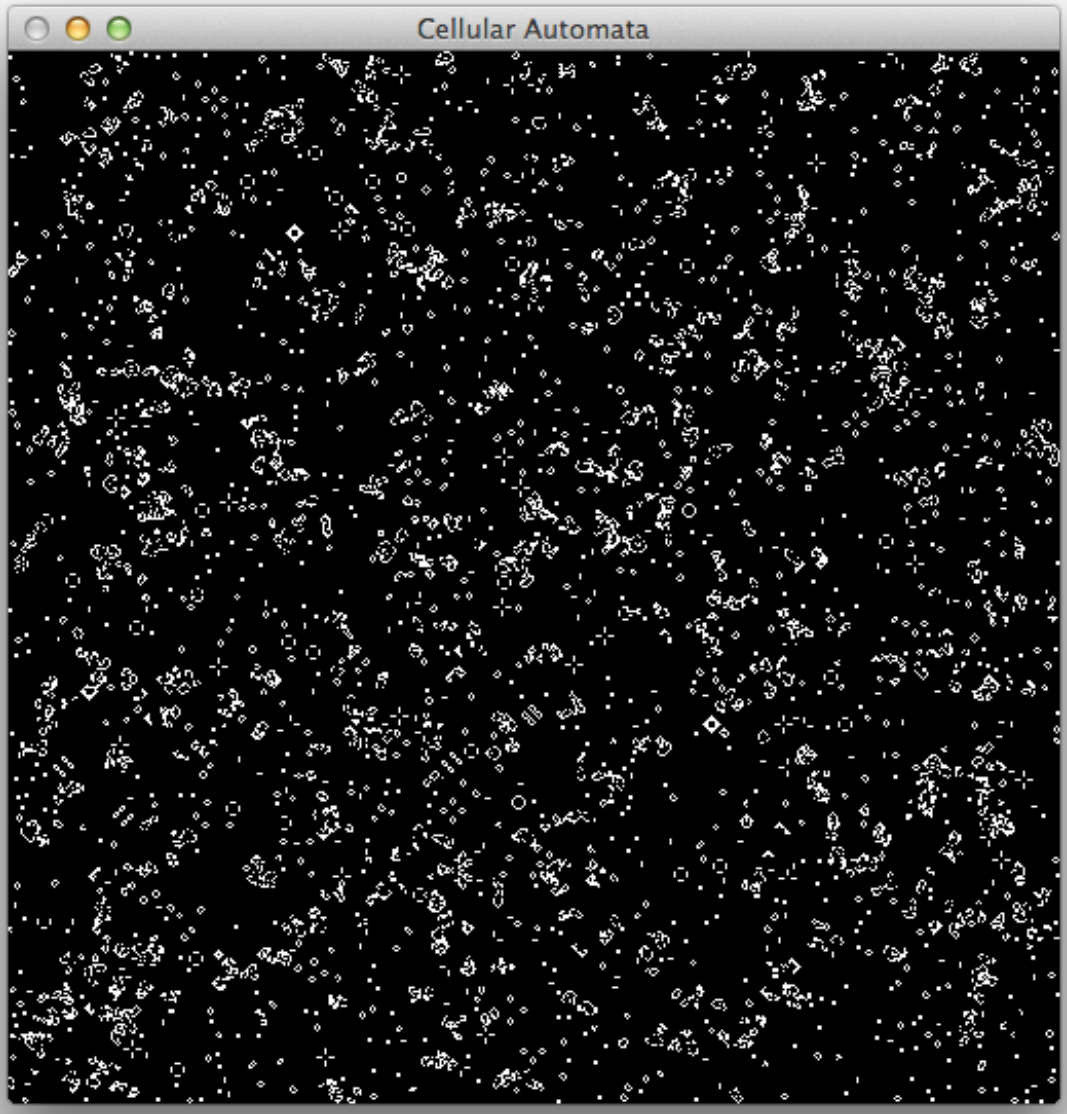
We notice a bump in performances for the GTX 560 around the 512*512 computation, this is most probably due to memory space or special optimization for textures of this size.



Another interesting point is that the software version of the Julia set compiled with optimized flags (-O3) are getting the same results as the CPU based OpenCL version. Based on these results we will optimistically use the OpenCL/CPU version of our code to compare with the GPU version in future implementations.

Cellular automata

A cellular automata consist of a grid of cells in which each cell has a finite number of states. At each iteration the future state of a cell is computer based on present state and on the state of its neighboring cells.



Definition

The cellular automata that was implemented is Conway's Game of Life. The cells in this game have only two states: "alive" and "dead". The cells move from one state to the other based on simple rules.

If the cell is alive and 2 or 3 neighboring cells are alive then the cell stays alive else the cell dies. If The cell is dead and 3 neighboring cells are alive this cell becomes alive, else it stays dead.

Implementation

We use a basic scheme as every cell get as assign core, this is manages by OpenCL the real gain happen between the generations. I tried to have the generation happening in the kernel and having a synchronization after every iteration but as there is too many cells this doesn't work so I used a technique of double-buffering on the graphics card using the OpenCL copy buffer inside the graphic card. This show a direct scaling of the performances with the number of cells and the number of iterations.

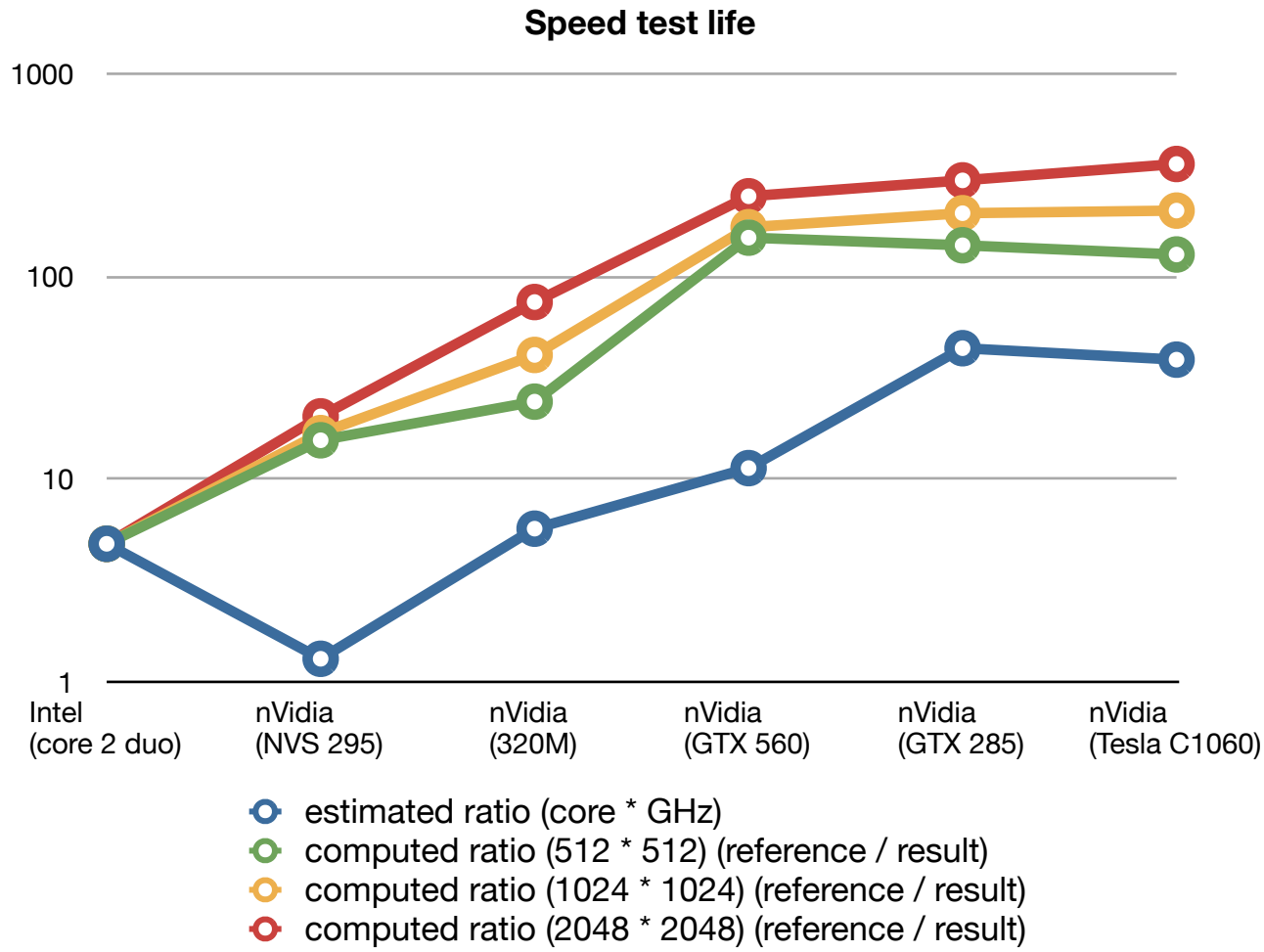
```
for (int i = 0; i < max_iterations_; ++i) {
    // make the computation
    err_ = queue_.enqueueNDRangeKernel(
           kernel_,
           cl::NullRange,
           cl::NDRange(mdx_, mdy_),
           cl::NullRange,
           NULL,
           &event_);
    queue_.finish();
    // swap the buffers (except for the last time)
    if (i < (max_iterations_ - 1)) {
           queue_.enqueueCopyBuffer(
                 cl_buffer_out_,
                 cl_buffer_in_,
                 0,
                 0,
                 total_size_,
                 NULL,
                 &event_);
           queue_.finish();
    }
}
```

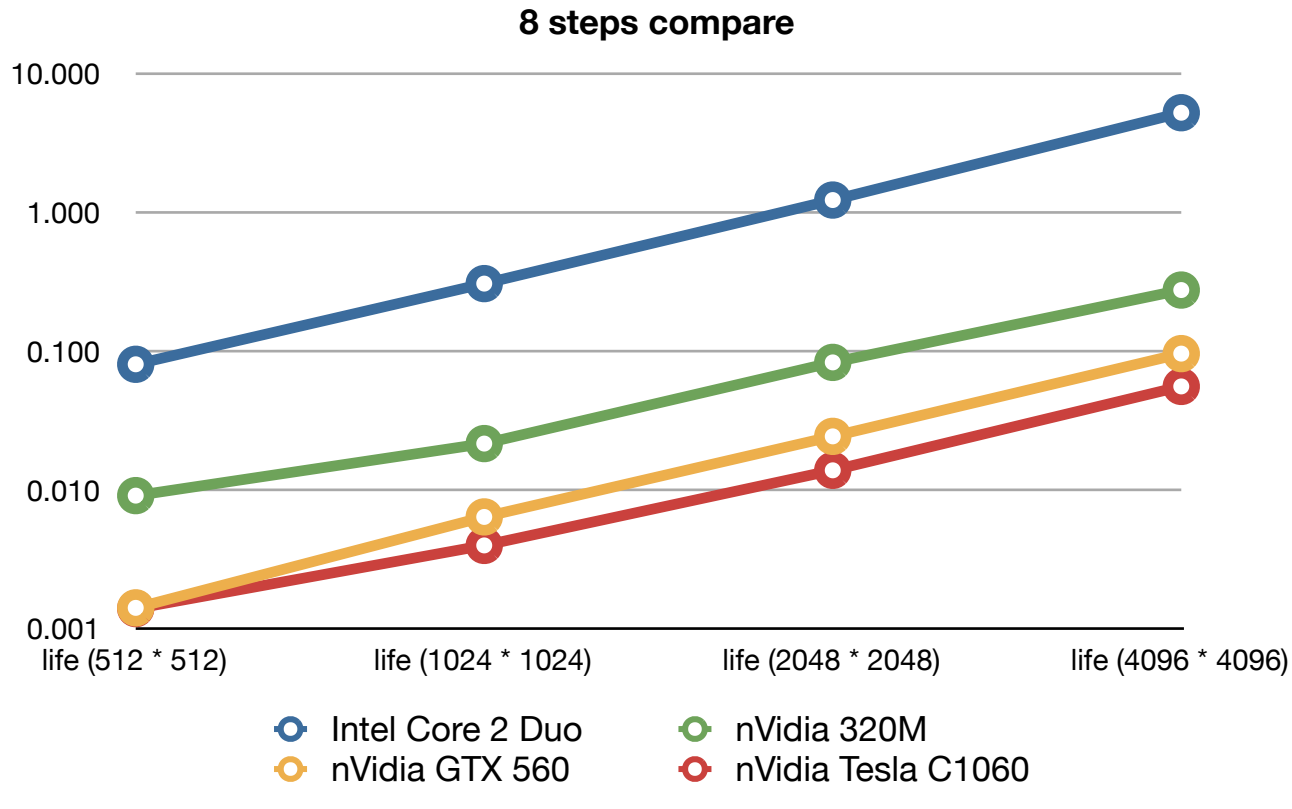
The other technique that was tried was to use a texture instead of a linear buffer. This led to clearer code as the grid is directly mapped to the texture. Each cell is one pixel. This technique has different performance outcomes and is very dependent on the hardware.

Results

We see even better result than with the Julia set as there is very little to compute per cell.

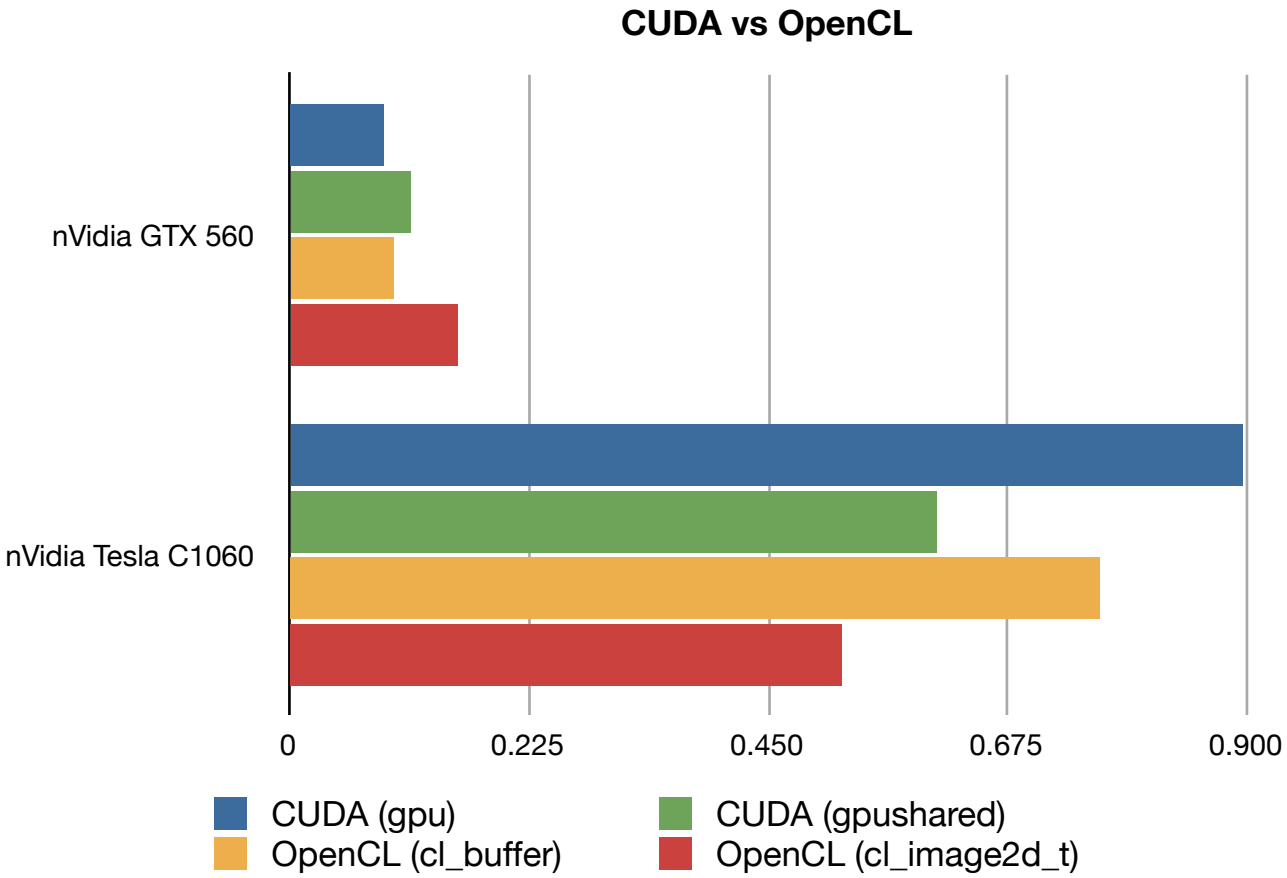
The only important performance hit that this algorithm could suffer is random memory access. But, as we are accessing neighboring elements and the GPU is made for this kind of actions on textures, this is not really affecting the results.


Speed test life
1000
100
10
1
Intel (core 2 duo)
nVidia (NVS 295)
nVidia (320M)
nVidia (GTX 560)
nVidia (GTX 285)
nVidia (Tesla C1060)
estimated ratio (core * GHz)
computed ratio (512 * 512) (reference / result)
computed ratio (1024 * 1024) (reference / result)
computed ratio (2048 * 2048) (reference / result)


8 steps compare
10.000
1.000
0.100
0.010
0.001
life (512 * 512)
life (1024 * 1024)
life (2048 * 2048)
life (4096 * 4096)
Intel Core 2 Duo
nVidia 320M
nVidia GTX 560
nVidia Tesla C1060

In this graph we see a direct correlation between the size of the grid and the computing speed. This is also true for the number of steps we compute. This test was made with 8 steps, and the values on the side are the speed.

We used code written by Pierre Küntzli to compare performances between CUDA and OpenCL with a grid 192 * 192 and a 1000 iterations.



The horizontal axe is the speed so shorter bar are faster. As expected the result are very close, but depend on the machine and the technique used. Similar results can be seen in the literature, where it also emerges that the differences in performance are getting smaller with time.

Floyd-Warshall

Floyd-Warshall is an algorithm for computing the shortest path between all pair of vertices in a weighted graph. The edges can be weighted either positive or negative.

Definition

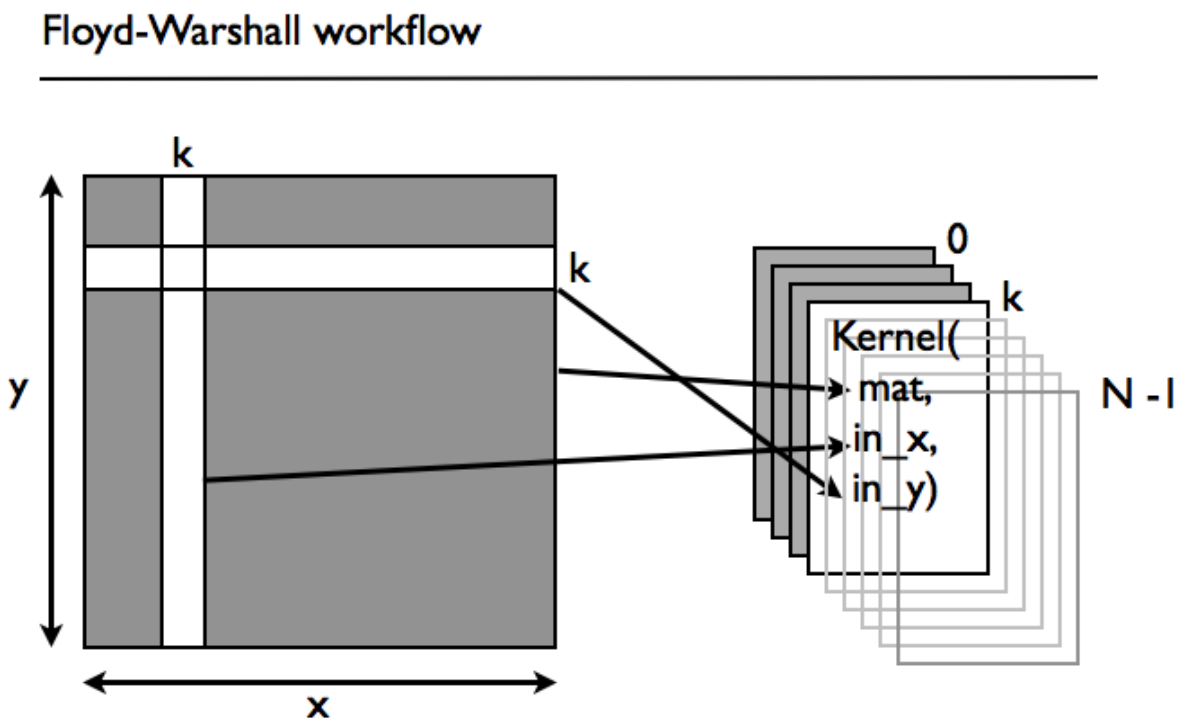
Floyd-Warshall is able to find the shortest paths in a graph in the order of V^3, where V is the number of vertices in the graph and V^2 is the number of edges. It works by iteratively checking every possible combination of edges.

$$W^0_{(V,V)} : \ initial\ weight\ connection\ matrix\ and\ k, i, j = 1, ..., V$$

$$W^k_{i,j} = min(W^{k-1}_{i,j}, W^{k-1}_{i,k} + W^{k-1}_{k,j})$$

Implementation

The algorithm is using a line k and a column k in which you fetch the values add and then compare and replace it in. The idea is to get the matrix and the line and column corresponding to the iteration in the Floyd-Warshall algorithm. Doing so we avoid any lookup inside the matrix and speed up the process drastically. Getting the column an line corresponding to the iteration is done by copying from inside the matrix to a buffer.



The kernel is quite simple since preparation of the line and column k is done outside the kernel. We can use the ternary operator in order to avoid having a branch.

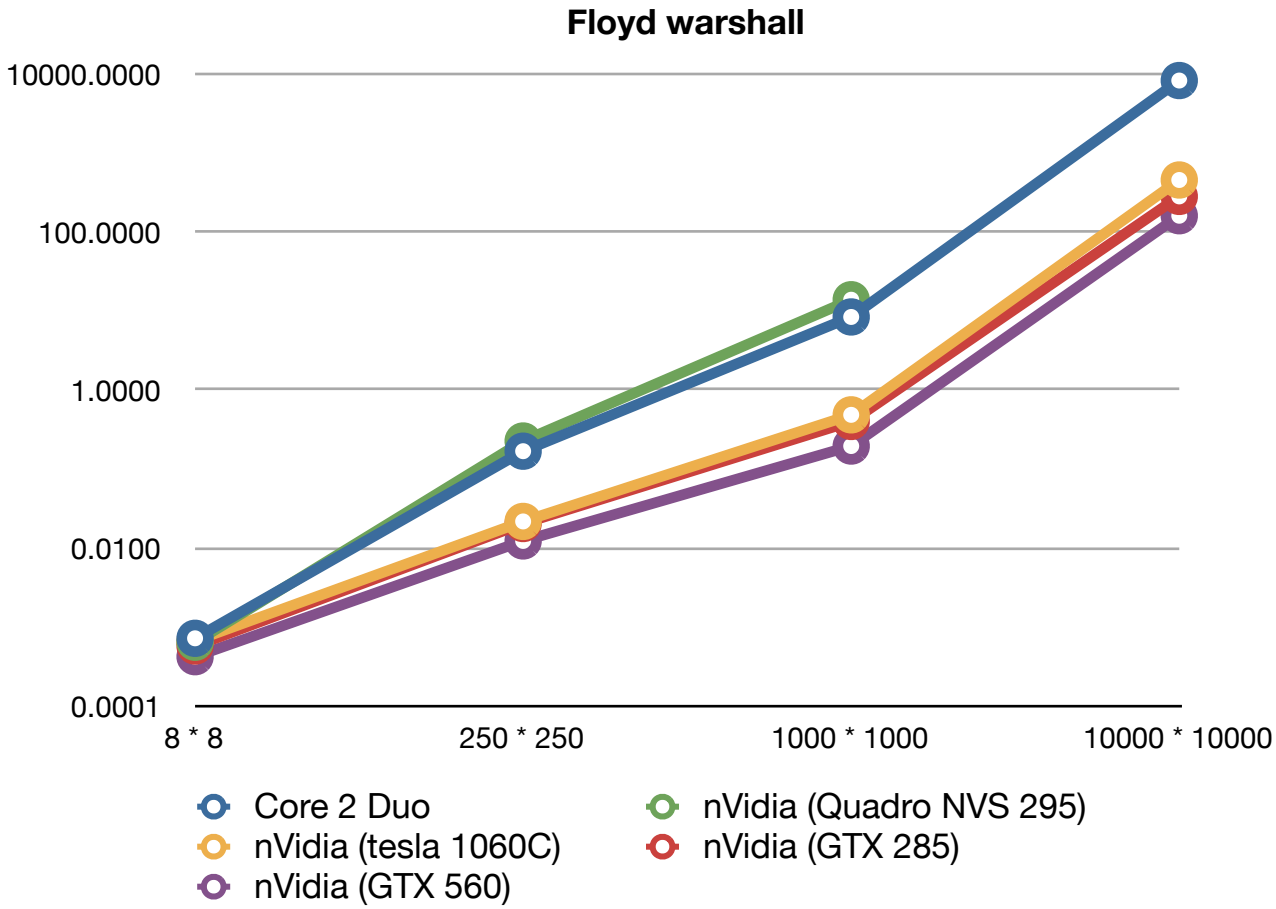
```
__kernel void floyd_warshall_buffer(
    __global float* mat,
    __global float* in_x,
    __global float* in_y)
{
    const int2 d = (int2)(get_global_id(0), get_global_id(1));
    const int2 m = (int2)(get_global_size(0), get_global_size(1));

    int position = d.y * m.x + d.x
    float val1 = mat[position];
    float val2 = in_x[d.x] + in_y[d.y];
    mat[position] = (val1 < val2) ? val1 : val2;
}
```

Results

As the following graph shows we have performances that are around one or two order of magnitude faster for a GPU than a CPU. The non-linearity is probably due to cache miss or some similar phenomenon.

This implementation uses a single adjacency matrix that can get quite big. It is known that one can divide the matrix into smaller chunks to avoid reaching the memory limitation. This could be achieved outside the kernel while still using the same kernel as the one we provide, and making the memory swap in the host code.



Fast Fourier transform

The Fourier transform is a mathematical operation that moves a function from a temporal domain to a frequency domain. In our case we are talking about Discreet Fourier Transform (DFT), and in particular, the Fast Fourier Transform (FFT).

Definition

The Discrete Fourier Transform (DFT) is a discrete transform. It transform a function into another, which is called frequency domain, or simply DFT, of the original function.

$$x_0, ..., x_{N-1} \in \mathbb{C}$$

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}} \quad k = 1, ..., N-1$$

In order to compute the DFT one must compute N number of values N times. The complexity is thus

$$\mathcal{O}(N^2).$$

The most commonly used FFT algorithms are based on a divide-and-conquer approach similar to the algorithm of Cooley and Turkey (1965). The computation of a DFT of length N is done by splitting the input sequence into a fixed small number of subsequences, compute their DFT, and assemble the outputs to build the final sequence. If the split and assembly are linear in time the complexity becomes

$$\mathcal{O}(N \log(N))$$

Implementation

I used the implementation from Eric Brainville with radix power of 2. It is supposed to be equivalent to the CUDA FFT library, and it served as a base to the Apple implementation. The idea is to call the kernel once per subdivision.

```
// Compute T x DFT-2.
// T is the number of threads.
// N = 2*T is the size of input vectors.
// X[N], Y[N]
// P is the length of input sub-sequences: 1,2,4,...,T.
// Each DFT-2 has input (X[I],X[I+T]), I=0..T-1,
// and output Y[J],Y|J+P], J = I with one 0 bit inserted at position P.
__kernel void fftRadix2Kernel(
     __global const real2_t * x,
     __global real2_t * y,
     int p)
{
     int t = get_global_size(0); // thread count
     int i = get_global_id(0);   // thread index
     int k = i & (p - 1);        // index in input sequence, in 0..P-1
     int j = ((i - k) << 1) + k; // output index
```

```
    real_t alpha = -FFT_PI * (real_t)k / (real_t)p;
    // Read and twiddle input
    x += i;
    real2_t u0 = x[0];
    real2_t u1 = twiddle(x[t], 1, alpha);
    // In-place DFT-2
    DFT2(u0, u1);
    // Write output
    y += j;
    y[0] = u0;
    y[p] = u1;
}
```

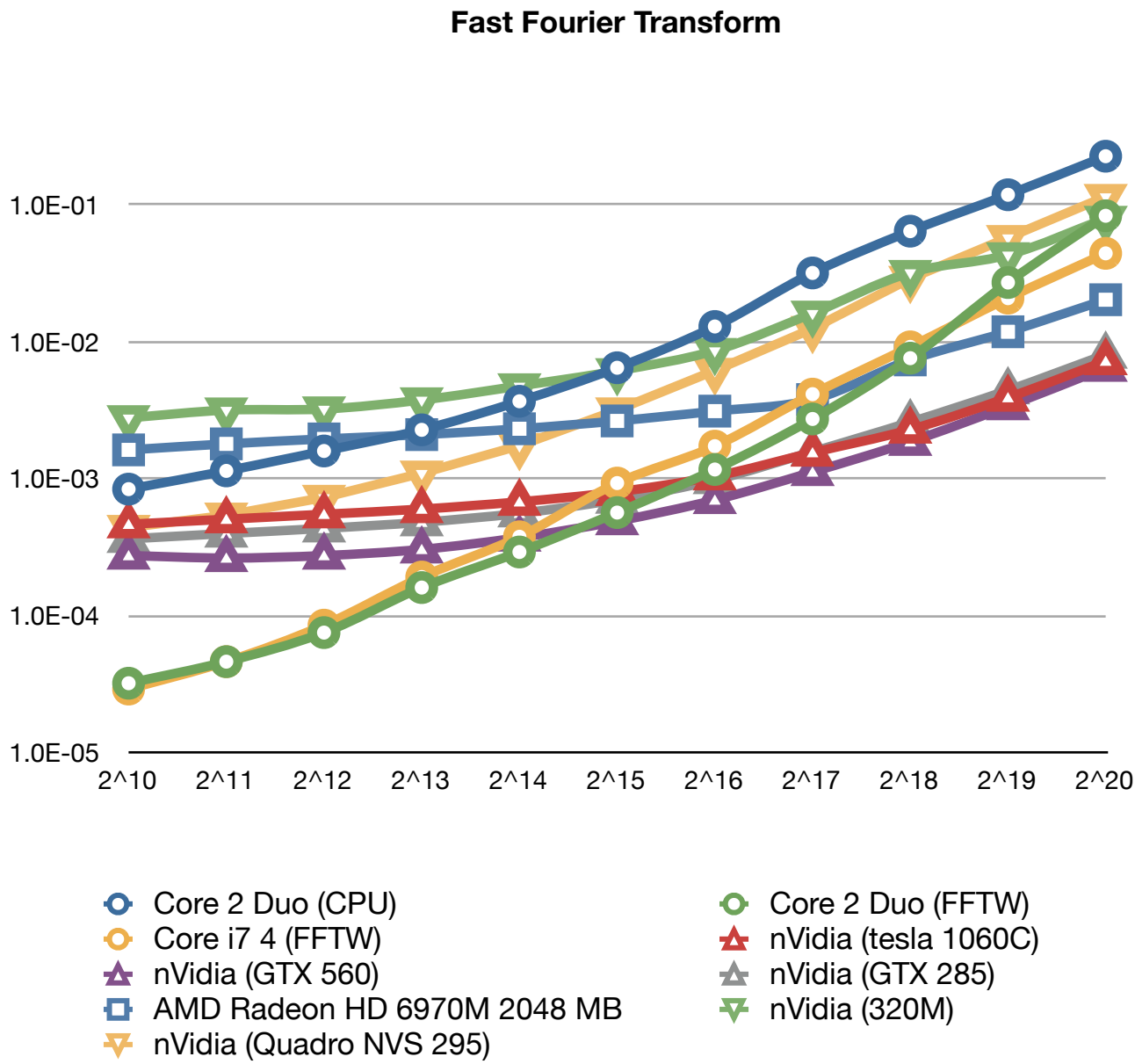
The macros real_t and real2_t are there to be able to switch from double to single precision floating values. Some of the subroutines are also missing but the full source is available either from my examples or from Eric Bainville's website.

For comparing we used the standard and widely used FFTW library. As a reference we also had a look into the GPUFFTW library and our measures match theirs.

Results

The implementation match the result from GPUFFTW. FFTW is faster for small datasets, probably due to overhead in loading and initiating the GPU kernel. As soon as we reach 2^{15} points the GPU start to be better, and the increase of time compare to the increase of data size is a lower than with the CPU.

It is probably possible to pipeline the data into different steps, like loading the next buffer while calculating one iteration of the FFT with a pipeline depth equal to the logarithm of 2 of the data length.


Fast Fourier Transform
1.0E-01
1.0E-02
1.0E-03
1.0E-04
1.0E-05
2^10
2^11
2^12
2^13
2^14
2^15
2^16
2^17
2^18
2^19
2^20
Core 2 Duo (CPU)
Core i7 4 (FFTW)
nVidia (GTX 560)
AMD Radeon HD 6970M 2048 MB
nVidia (Quadro NVS 295)
Core 2 Duo (FFTW)
nVidia (tesla 1060C)
nVidia (GTX 285)
nVidia (320M)

Optimizations

In this part we will have a look at what speed optimizations techniques have been used and what their benefits are in term of performances. Most of these techniques are tied to a specific problem, or to specific hardware.

Comparison with CPU

Performances of GPU is at least one order of magnitude better than the CPU, depending on the algorithm. Some of the functionality that we use to have on CPU are not available on GPU, mainly recursive function calls. Some instructions that are frequent on CPU are not present or very slow on GPU like modulus and branches.

Parameter vectorization

With experimentation, it became clear that vectorizing parameters offered some improvement in performances. This is probably something that may disappeared in future implementations but seams true for now. To clarify, it is better to send parameters as a vector to a kernel than sending parameters separately.

Modulus

As said previously, GPU miss the modulus operator. The following formula allows to compute simple and efficient moduli. It is not usable on negative values unless we do some sign preserving or sign extension.

$$a \ mod \ 2^b = a \ \& \ (2^b - 1)$$

This formula is generalizable to any radix if we take into account the fact that any number can be decomposed into a sum of powers of 2, and that the modulus has the following property:

$$a_1 \mod b_1 \equiv c_1 \ and \ a_2 \mod b_2 \equiv c_2$$

$$a_1 + a_2 \mod b_1 + b_2 \equiv c_1 + c_2$$

Buffers versus images

Something that is commonly proposed as an optimization on certain platforms is to replace buffers by images, as GPU are made to handle textures rather than arrays. This is in fact efficient on older or portable platforms but inefficient on more recent and high end cards.

Memory spatial distribution

The GPU is made for texture lookup and texture manipulation. Random access to memory should be avoided where possible. This became evident in the Floyd-Warshall naive implementation as the lookup of the current column and line was crippling the performance. When the column and the line were passed as separate parameters, and the memory management was done outside the kernel, the speed went up by an order of magnitude.

As a general rule of thumb, as long as memory access is restricted to close-by locations it is rapid, but random access is very slow. This may be one of the reasons that the FFT, and in particular

the butterfly part of it, gives good but not outstanding results. This may be improved with new GPU architectures that are suppose to address these issues.

Memory management

In the case where the algorithm manipulating memory by chunks and then doing simple computations for each iteration, it is simpler and more efficient to make the memory manipulation outside of the kernel and then calling the kernel.

This strategy was used in the Floyd-Warshall implementation and for the cellular automata. For Floyd-Warshall the column and line treaded at each iteration where prepared outside the kernel and then the kernel called for one iteration.

OpenCL with OpenGL

OpenGL interoperation was tested and seams to work as long as you have access to the console. This mean that in case you are tunneling your results trough X, it is not working! In case you want to test it, in the software, you have to make a modification in the source of *cl_<name>.cpp*. Just comment out the following line in the constructor.

```
throw cl::Error(0, "cheat pass");
```

If after that you get the message:

```
Warning         : could not attach GL and CL together!
```

It means your card doesn't support it.

Features and limitations

Hardware platforms used

The test and implementation where made on mainly Linux (Fedora 16 and Ubuntu) and Mac OSX (10.7). The was almost no portability issues other than different header/library path to be included. The OpenCL library is installed by default on Max OSX and come with the proprietary drivers on Linux. As for SDK, OpenCL SDK is installed with Xcode (the standard Apple development tools) on Mac OSX and with the CUDA development kit on Linux. It can also be downloaded from the Khronos group. The library comes with the driver, you just need the headers and these are specified by the Khronos group and therefore standard.

Hardware wise many different computing units where tried :

Manufacturer	**Model**	**Memory**	**OS**
Intel	Core Duo 2	8GB	Mac OSX 10.7
NVIDIA	320M	256MB	Mac OSX 10.7
AMD	Radeon HD 6970M	2GB	Mac OSX 10.7
NVIDIA	Quadro NVS 295	256MB	Linux/Fedora 16
NVIDIA	Tesla 1060C	4GB	Linux/Ubuntu
NVIDIA	GTX 560	1GB	Linux/Debian 6.0.4
NVIDIA	GTX 285	1GB	Linux/Fedora 16

Problems encountered

I have tried and implemented Julia set and Game of Life using both *cl_image2d_t* and *cl_buffer*. The result were mixed. Usually it can be a big improvement, up to 5X, or a small setback, less than 10%. This is completely depending on the device. This is probably due to different generation of hardware or to the fact that some hardware are more gaming related.

The function *clEnqueueCopyBufferRect*, in the OpenCL 1.1 specification works under most of the hardware I have tried it on, but there are some exceptions. In particular when you try to copy a chunk of non continuous memory.

Acknowledgment

I'd like to thanks people at CERN, that let me make this possible: Daniel Valuch, Andy Butterworth, Wolfgang Hofle, Michael Jaussi, and the HES-SO/hepia with Pr. Paul Albuquerque

Conclusion

OpenCL seams like a mature technology and all the test that were made show decent to good results. Comparing to CUDA show that the critics that say OpenCL is somewhat slower than CUDA are rather wrong and that the drivers are on the same level. Comparing it to CPU show excellent results for highly parallel algorithms and decent result with others.

The complexity of the language and the ease of porting algorithms show a good potential. The interface is simple and clear enough and the fact that you have a separation between host and device code make the implementation a lot simpler by not having to rely on an external compiler. The abundance of bindings and the portability is an impressive and important point for the further acceptance of the platform.

The development cycle for such solutions is therefore short and the platform global cost is low compare to custom hardware solutions. Adding the portability that is offered by OpenCL opens the road to easier maintenance and upgrades.

References

http://www.khronos.org - Khronos Group

http://www.nvidia.com/object/cuda_home_new.html - CUDA

http://algs4.cs.princeton.edu/44sp/ - dataset for Floyd-Warshall

http://www.bealto.com/gpu-fft2.html - OpenCL FFT

http://developer.apple.com/library/mac/samplecode/OpenCL_FFT/Introduction/Intro.html - Apple FFT on OpenCL

http://www.fftw.org/ - FFTW library

http://gamma.cs.unc.edu/GPUFFTW/results.html - GPUFFTW results

OpenCL in Action - Matthew Scarpino - 2011

OpenCL Programming Guide - Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung , Dan Ginsburg - 2012

http://www.boost.org - the boost library

http://enja.org/category/tutorial/advcl/index.html - OpenCL examples (C/C++)

http://en.wikipedia.org/wiki/Fast_Fourier_transform - Wikipedia FFT page

Annexes

Software notes

The software used for these test is available on github at :

https://github.com/anirul/OpenCL_PA_2012

Compiling

Go to the directory you're interested in and make! This has been tested on various kind of Linux and Mac OSX. You must have installed the dependencies and have valid OpenCL and OpenGL drivers.

Dependencies

The software is using Boost, for program_option and date_time, FFTW for reference implementation of the FFTs, and of course OpenCL and OpenGL. You will also need GLUT for the window management.

Executions

There is help on the command line for each of the software, but here is the list of parameters and possibilities for each of them.

Test

Take no parameter. This is a test program that just output the capabilities of the devices available on the platform. It is always good to check if this work as it is the simplest OpenCL software.

Julia

-h [--help]	output the help message, list the available commands
-c [--opencl-cpu]	compute using openCL on CPU
-g [--opencl-gpu]	compute using openCL on GPU this is the default
-m [--image]	use OpenCL Image2D instead of buffer
-w [--width] arg	image width
-t [--height] arg	image height
-i [--iterations] arg	Julia maximum iterations
-l [--loop] arg	number of loops (only in no-window mode)
-x [--cx] arg	value of c(x) real part
-y [--cy] arg	value of c(y) imaginary part
-n [--no-window]	no window output

Cellular_automata

-h [--help]	output the help message, list the available commands
-c [--opencl-cpu]	compute using openCL on CPU

-g [--opencl-gpu]	compute using openCL on GPU (default)
-m [--image]	use OpenCL Image2D instead of buffer
-w [--width] arg	image width
-t [--height] arg	image height
-i [--iterations] arg	Julia iterations
-f [--cl-file] arg	OpenCL file to be compiled (default : life.cl)
-l [--loop] arg	number of loops (only in no-window mode)
-n [--no-window]	no window output

Floyd_warshall

-h [--help]	produce help message
-c [--opencl-cpu]	compute using openCL on CPU
-g [--opencl-gpu]	compute using openCL on GPU (default)
-f [--file-in] arg	Input graph file in EWD format
-o [--file-out] arg	Output graph file in EWD format
-l [--loop] arg	number of loops (only in no-window mode)

The file format accepted is EWD as the file found on princeton dataset. This is a pretty simple file format. It consists of the parameters of the graph: number of nodes, number of edges, and all the edges, consisting of beginning node, end node and the weight.

Fft

-h [--help]	produce help message
-c [--opencl-cpu]	compute using openCL on CPU
-g [--opencl-gpu]	compute using openCL on GPU (default)
-w [--fftw]	compute using the FFTW3 library
-f [--file-in] arg	Input file in TXT format
-o [--file-out] arg	Output file in TXT format
-l [--loop] arg	number of loops

The file format used is just the real part output one after the other. There is a simple matlab script, *generate_data.m,* to produce the test set in the directory. The output is done in a similar manner.

Source code

The source code is suppose to be organized in directories and have a directory called CL with the header for OpenCL (provided on the Khronos website). The structure is given at the same time than the sources (except for the Khronos code where I only provide the directory structure).

CL

This is the directory that contain the Khronos headers, here is the list of files that should be included.

```
├── CL
│   ├── cl.h
│   ├── cl.hpp
│   ├── cl_d3d10.h
│   ├── cl_ext.h
│   ├── cl_gl.h
│   ├── cl_gl_ext.h
│   ├── cl_platform.h
│   └── opencl.h
```

Here is included the general sources files and headers that are used in many different projects I include them here but if you want to compile a specific project don't forget to include them in the appropriate directory

Cl_util.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <iostream>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>

#include "cl_util.h"

std::ostream& operator<<(std::ostream& os, cl::Error err) {
     switch (err.err()) {
             case CL_BUILD_PROGRAM_FAILURE :
                    os << "CL_BUILD_PROGRAM_FAILURE";
                    break;
             case CL_COMPILER_NOT_AVAILABLE :
                    os << "CL_COMPILER_NOT_AVAILABLE";
                    break;
             case CL_DEVICE_NOT_AVAILABLE :
                    os << "CL_DEVICE_NOT_AVAILABLE";
                    break;
             case CL_DEVICE_NOT_FOUND :
                    os << "CL_DEVICE_NOT_FOUND";
                    break;
             case CL_IMAGE_FORMAT_MISMATCH :
                    os << "CL_IMAGE_FORMAT_MISMATCH";
                    break;
```

```
case CL_IMAGE_FORMAT_NOT_SUPPORTED :
       os << "CL_IMAGE_FORMAT_NOT_SUPPORTED";
       break;
case CL_INVALID_ARG_SIZE :
       os << "CL_INVALID_ARG_SIZE";
       break;
case CL_INVALID_ARG_VALUE :
       os << "CL_INVALID_ARG_VALUE";
       break;
case CL_INVALID_BINARY :
       os << "CL_INVALID_BINARY";
       break;
case CL_INVALID_BUFFER_SIZE :
       os << "CL_INVALID_BUFFER_SIZE";
       break;
case CL_INVALID_BUILD_OPTIONS :
       os << "CL_INVALID_BUILD_OPTIONS";
       break;
case CL_INVALID_COMMAND_QUEUE :
       os << "CL_INVALID_COMMAND_QUEUE";
       break;
case CL_INVALID_CONTEXT :
       os << "CL_INVALID_CONTEXT";
       break;
case CL_INVALID_DEVICE :
       os << "CL_INVALID_DEVICE";
       break;
case CL_INVALID_DEVICE_TYPE :
       os << "CL_INVALID_DEVICE_TYPE";
       break;
case CL_INVALID_EVENT :
       os << "CL_INVALID_EVENT";
       break;
case CL_INVALID_EVENT_WAIT_LIST :
       os << "CL_INVALID_EVENT_WAIT_LIST";
       break;
case CL_INVALID_GL_OBJECT :
       os << "CL_INVALID_GL_OBJECT";
       break;
case CL_INVALID_GLOBAL_OFFSET :
       os << "CL_INVALID_GLOBAL_OFFSET";
       break;
case CL_INVALID_HOST_PTR :
       os << "CL_INVALID_HOST_PTR";
       break;
case CL_INVALID_IMAGE_FORMAT_DESCRIPTOR :
       os << "CL_INVALID_IMAGE_FORMAT_DESCRIPTOR";
       break;
case CL_INVALID_IMAGE_SIZE :
       os << "CL_INVALID_IMAGE_SIZE";
       break;
case CL_INVALID_KERNEL_NAME :
       os << "CL_INVALID_KERNEL_NAME";
       break;
case CL_INVALID_KERNEL :
       os << "CL_INVALID_KERNEL";
       break;
case CL_INVALID_KERNEL_ARGS :
       os << "CL_INVALID_KERNEL_ARGS";
       break;
case CL_INVALID_KERNEL_DEFINITION :
       os << "CL_INVALID_KERNEL_DEFINITION";
       break;
case CL_INVALID_MEM_OBJECT :
       os << "CL_INVALID_MEM_OBJECT";
       break;
case CL_INVALID_OPERATION :
       os << "CL_INVALID_OPERATION";
       break;
case CL_INVALID_PLATFORM :
```

```
                os << "CL_INVALID_PLATFORM";
                break;
            case CL_INVALID_PROGRAM :
                os << "CL_INVALID_PROGRAM";
                break;
            case CL_INVALID_PROGRAM_EXECUTABLE :
                os << "CL_INVALID_PROGRAM_EXECUTABLE";
                break;
            case CL_INVALID_QUEUE_PROPERTIES :
                os << "CL_INVALID_QUEUE_PROPERTIES";
                break;
            case CL_INVALID_SAMPLER :
                os << "CL_INVALID_SAMPLER";
                break;
            case CL_INVALID_VALUE :
                os << "CL_INVALID_VALUE";
                break;
            case CL_INVALID_WORK_ITEM_SIZE :
                os << "CL_INVALID_WORK_ITEM_SIZE";
                break;
            case CL_MAP_FAILURE :
                os << "CL_MAP_FAILURE";
                break;
            case CL_MEM_OBJECT_ALLOCATION_FAILURE :
                os << "CL_MEM_OBJECT_ALLOCATION_FAILURE";
                break;
            case CL_MEM_COPY_OVERLAP :
                os << "CL_MEM_COPY_OVERLAP";
                break;
            case CL_OUT_OF_HOST_MEMORY :
                os << "CL_OUT_OF_HOST_MEMORY";
                break;
            case CL_OUT_OF_RESOURCES :
                os << "CL_OUT_OF_RESOURCES";
                break;
            case CL_PROFILING_INFO_NOT_AVAILABLE :
                os << "CL_PROFILING_INFO_NOT_AVAILABLE";
                break;
            case CL_SUCCESS :
                os << "CL_SUCCESS";
                break;
            default :
                os << err.err();
                break;
        }
        return os;
}
```

Cl_util.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
```

```
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef CL_UTIL_HEADER_DEFINED
#define CL_UTIL_HEADER_DEFINED

std::ostream& operator<<(std::ostream& os, cl::Error err);

#endif // CL_UTIL_HEADER_DEFINED
```

Glut_win.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdexcept>
#include <string>
#ifdef __linux__
#include <GL/glut.h>
#endif
#ifdef __APPLE__
#include <glut/glut.h>
#endif
#include <boost/bind.hpp>

#include "glut_win.h"

static void init() {
     glut_win::instance()->pwin_->init();
}

static void finish() {
     glut_win::instance()->pwin_->finish();
}

static void display() {
     static bool s_first = true;
     if (s_first) {
```

```
                s_first = false;
                init();
        } else {
                glut_win::instance()->pwin_->display();
        }
        glutSwapBuffers();
}

static void keyboard(unsigned char key, int x, int y) {
        switch(key) {
                case 27 :
                        finish();
                        exit(0);
                        break;
        }
        glut_win::instance()->pwin_->keyboard(key, x, y);
}

static void reshape(int w, int h) {
        glut_win::instance()->pwin_->reshape(w, h);
}

static void idle() {
        glut_win::instance()->pwin_->idle();
        glutPostRedisplay();
}

static void mouse_move(int x, int y) {
        glut_win::instance()->pwin_->mouse_move(x, y);
}

static void mouse_event(int button, int state, int x, int y) {
        glut_win::instance()->pwin_->mouse_event(button, state, x, y);
}

glut_win* glut_win::instance_ = NULL;

glut_win* glut_win::instance(
        const std::string& name,
        const std::pair<unsigned int, unsigned int>& range,
        i_win* windesc,
        bool fullscreen)
                throw(std::exception)
{
        if (!instance_)
                instance_ = new glut_win(name, range, windesc, fullscreen);
        return instance_;
}

glut_win* glut_win::instance() throw(std::exception) {
        if (instance_) return instance_;
        else throw std::runtime_error(std::string("glut_win was not initialized!"));
}

glut_win::glut_win(
        const std::string& name,
        const std::pair<unsigned int, unsigned int>& range,
        i_win* windesc,
        bool fullscreen)
                throw(std::exception)
{
        if (!windesc) throw std::runtime_error(std::string("Window class == NULL"));
        pwin_ = windesc;
        fullscreen_ = fullscreen;
        int ac = 0;
        glutInit(&ac, NULL);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
        if (!fullscreen_)
                glutInitWindowSize(range.first, range.second);
        glutCreateWindow(name.c_str());
```

```
	if (fullscreen_)
		glutFullScreen();
}

glut_win::~glut_win() { }

void glut_win::run()
	throw(std::exception)
{
	glutDisplayFunc(display);
	glutKeyboardFunc(keyboard);
	glutReshapeFunc(reshape);
	glutMotionFunc(mouse_move);
	glutMouseFunc(mouse_event);
	glutIdleFunc(idle);
	glutMainLoop();
}
```

Glut_win.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef GLUT_WIN_HEADER_DEFINED
#define GLUT_WIN_HEADER_DEFINED

class i_win {
public :
	virtual void init() = 0;
	virtual void display() = 0;
	virtual void idle() = 0;
	virtual void reshape(int w, int h) = 0;
	virtual void mouse_event(int button, int state, int x, int y) = 0;
	virtual void mouse_move(int x, int y) = 0;
	virtual void keyboard(unsigned char key, int x, int y) = 0;
	virtual void finish() = 0;
};

// window class
class glut_win {
	static glut_win* instance_;
	bool fullscreen_;
	glut_win(
		const std::string& name,
		const std::pair<unsigned int, unsigned int>& range,
```

```
                i_win* windesc,
                bool fullscreen = false)
                        throw(std::exception);
public :
        i_win* pwin_;
        static glut_win* instance(
                const std::string& name,
                const std::pair<unsigned int, unsigned int>& range,
                i_win* windesc,
                bool fullscreen = false)
                        throw(std::exception);
        static glut_win* instance()
                        throw(std::exception);
        virtual ~glut_win();
        void run()
                        throw(std::exception);
};

#endif // GLUT_WIN_HEADER_DEFINED
```

Simple

This is the simple example of the beginning, only the Makefile will be added here as the source is in the text already.

```
├── simple
│   ├── Makefile
│   ├── simple.cl
│   └── simple.cpp
```

Makefile

```
# a simple makefile for a simple example


ifeq ($(OSTYPE), darwin)
CC = clang
CXX = clang++
CFLAGS = -O3 -I/opt/local/include -I/usr/local/include -I..
LIBS = -L/opt/local/lib/ \
          -framework GLUT -framework Cocoa \
          -framework OpenCL -framework OpenGL\
          -lboost_program_options -lboost_system
else
CC = gcc
CXX = g++
CFLAGS = -O3 -I/usr/include -I/usr/local/include -I..
LIBS = -L/usr/lib64 -L/usr/local/lib \
          -lglut -lGL -lGLU -lOpenCL \
          -lboost_program_options -lboost_system
endif
ALL=simple


all: $(ALL)
```

```
simple.o : simple.cpp
	$(CXX) -o simple.o -c simple.cpp $(CFLAGS)

simple : simple.o
	$(CXX) -o simple simple.o $(LIBS)

clean:
	rm -f $(ALL) *.o
```

Julia

```
├── julia
│   ├── Makefile
│   ├── cl_julia.cpp
│   ├── cl_julia.h
│   ├── cl_util.cpp
│   ├── cl_util.h
│   ├── glut_win.cpp
│   ├── glut_win.h
│   ├── julia.cl
│   ├── main.cpp
│   ├── win_julia.cpp
│   └── win_julia.h
```

Makefile

```
# a simple makefile for julia

ifeq ($(OSTYPE), darwin)
CC = clang
CXX = clang++
CFLAGS = -O3 -I/opt/local/include -I/usr/local/include -I..
LIBS = -L/opt/local/lib/ \
	-framework GLUT -framework Cocoa -framework OpenCL -framework OpenGL \
	-lboost_program_options -lboost_system
else
CC = gcc
CXX = g++
CFLAGS = -O3 -I/usr/include -I/usr/local/include -I..
LIBS = -L/usr/lib64 -L/usr/local/lib \
	-lglut -lGL -lGLU -lOpenCL \
	-lboost_program_options -lboost_system
endif
ALL=julia

all: $(ALL)
```

```
cl_util.o : cl_util.cpp cl_util.h
	$(CXX) -o cl_util.o -c cl_util.cpp $(CFLAGS)
cl_julia.o : cl_julia.cpp cl_julia.h
	$(CXX) -o cl_julia.o -c cl_julia.cpp $(CFLAGS)
glut_win.o : glut_win.cpp glut_win.h
	$(CXX) -o glut_win.o -c glut_win.cpp $(CFLAGS)
win_julia.o : win_julia.cpp win_julia.h
	$(CXX) -o win_julia.o -c win_julia.cpp $(CFLAGS)
main.o : main.cpp cl_julia.h win_julia.h
	$(CXX) -o main.o -c main.cpp $(CFLAGS)

julia : cl_julia.o main.o cl_util.o glut_win.o win_julia.o
	$(CXX) -o julia cl_julia.o cl_util.o main.o glut_win.o win_julia.o $(LIBS)

clean:
	rm -f $(ALL) *.o
```

Cl_julia.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <iostream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#ifdef __linux__
#include <GL/glx.h>
#endif
#include <boost/date_time/posix_time/posix_time.hpp>

#include "cl_julia.h"
```

```
using namespace boost::posix_time;

cl_julia::cl_julia(bool gpu) {
	//setup devices and context
	std::vector<cl::Platform> platforms;
	err_ = cl::Platform::get(&platforms);
	device_used_ = 0;
	err_ = platforms[0].getDevices((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU,
&devices_);
	int t = devices_.front().getInfo<CL_DEVICE_TYPE>();
	try {
		throw cl::Error(0, "cheat pass");
		#if defined (__APPLE__) || defined(MACOSX)
			CGLContextObj kCGLContext = CGLGetCurrentContext();
			CGLShareGroupObj kCGLShareGroup = CGLGetShareGroup(kCGLContext);
			cl_context_properties props[] =
			{
				CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
(cl_context_properties)kCGLShareGroup,
				0
			};
			context_ = cl::Context(props);
		#else
			#if defined WIN32 // Win32
				cl_context_properties props[] =
				{
					CL_GL_CONTEXT_KHR,
(cl_context_properties)wglGetCurrentContext(),
					CL_WGL_HDC_KHR,
(cl_context_properties)wglGetCurrentDC(),
					CL_CONTEXT_PLATFORM, (cl_context_properties)
(platforms[0])(),
					0
				};
				context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
			#else
				throw cl::Error(0, "cheat pass");
				cl_context_properties props[] =
				{
					CL_GL_CONTEXT_KHR,
(cl_context_properties)glXGetCurrentContext(),
					CL_GLX_DISPLAY_KHR,
(cl_context_properties)glXGetCurrentDisplay(),
					CL_CONTEXT_PLATFORM, (cl_context_properties)
(platforms[0])(),
					0
				};
				context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
			#endif
		#endif
	} catch (cl::Error er) {
		std::cerr << "Warning         : could not attach GL and CL toghether!" <<
std::endl;
		cl_context_properties properties[] = { CL_CONTEXT_PLATFORM,
(cl_context_properties)(platforms[0])(), 0};
		context_ = cl::Context((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU ,
properties);
		devices_ = context_.getInfo<CL_CONTEXT_DEVICES>();
	}
	queue_ = cl::CommandQueue(context_, devices_[device_used_], 0, &err_);
}

void cl_julia::init() {
	std::string file_name = "./julia.cl";
	FILE* file = fopen(file_name.c_str(), "rt");
	if (!file) throw std::runtime_error("could not open file " + file_name);
	const unsigned int BUFFER_SIZE = 4096;
	size_t bytes_read = 0;
	std::string kernel_source = "";
```

```
	do {
		char temp[BUFFER_SIZE];
		memset(temp, 0, BUFFER_SIZE);
		bytes_read = fread(temp, sizeof(char), BUFFER_SIZE, file);
		kernel_source += temp;
	} while (bytes_read != 0);
	fclose(file);
	cl::Program::Sources source(
		1,
		std::make_pair(kernel_source.c_str(),
		kernel_source.size()));
	program_ = cl::Program(context_, source);
	try {
		err_ = program_.build(devices_);
	} catch (cl::Error er) {
		std::cout << "build status    : "
			<< program_.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(devices_[0]) <<
std::endl;
		std::cout << "build options   : "
			<< program_.getBuildInfo<CL_PROGRAM_BUILD_OPTIONS>(devices_[0]) <<
std::endl;
		std::cout << "build log       : "
			<< program_.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices_[0]) <<
std::endl;
		throw er;
	}
}

void cl_julia::setup(
	const std::pair<float, float>& c,
	const std::pair<unsigned int, unsigned int>& s,
	unsigned int max_iterations)
{
	cl_c_.x = c.first;
	cl_c_.y = c.second;
	max_iterations_ = max_iterations;
	mdx_ = s.first;
	mdy_ = s.second;
}

void cl_julia::prepare_buffer() {
	total_size_ = mdx_ * mdy_;
	kernel_ = cl::Kernel(program_, "julia_buffer", &err_);
	//initialize our CPU memory arrays, send them to the device and set the kernel
arguements
	cl_buffer_out_ = cl::Buffer(
		context_,
		CL_MEM_WRITE_ONLY,
		sizeof(float) * total_size_,
		NULL,
		&err_);
	queue_.finish();
	//set the arguements of our kernel
	err_ = kernel_.setArg(0, cl_buffer_out_);
	err_ = kernel_.setArg(1, cl_c_);
	err_ = kernel_.setArg(2, max_iterations_);
	//Wait for the command queue to finish these commands before proceeding
	queue_.finish();
}

void cl_julia::prepare_image() {
	total_size_ = mdx_ * mdy_;
	origin_.push_back(0);
	origin_.push_back(0);
	origin_.push_back(0);
	region_.push_back(mdx_);
	region_.push_back(mdy_);
	region_.push_back(1);
	kernel_ = cl::Kernel(program_, "julia_image", &err_);
```

```
	//initialize our CPU memory arrays, send them to the device and set the kernel_
arguements
	cl::ImageFormat format = cl::ImageFormat(CL_INTENSITY, CL_UNORM_INT8);
	cl_image_out_ = cl::Image2D(
			context_,
			CL_MEM_WRITE_ONLY,
			format,
			mdx_,
			mdy_,
			0,
			NULL,
			&err_);
	queue_.finish();
	//set the arguements of our kernel_
	err_ = kernel_.setArg(0, cl_image_out_);
	err_ = kernel_.setArg(1, cl_c_);
	err_ = kernel_.setArg(2, max_iterations_);
	//Wait for the command queue to finish these commands before proceeding
	queue_.finish();
}

time_duration cl_julia::run_buffer(std::vector<float>& out) {
	ptime before;
	ptime after;
	if (out.size() != total_size_)
			out.resize(total_size_);
	before = microsec_clock::universal_time();
	err_ = queue_.enqueueNDRangeKernel(
			kernel_,
			cl::NullRange,
			cl::NDRange(mdx_, mdy_),
			cl::NullRange,
			NULL,
			&event_);
	queue_.finish();
	after = microsec_clock::universal_time();
	err_ = queue_.enqueueReadBuffer(
			cl_buffer_out_,
			CL_TRUE,
			0,
			sizeof(float) * total_size_,
			&out[0],
			NULL,
			&event_);
	queue_.finish();
	return (after - before);
}

time_duration cl_julia::run_image(std::vector<char>& out) {
	ptime before;
	ptime after;
	if (out.size() != total_size_)
			out.resize(total_size_);
	before = microsec_clock::universal_time();
	err_ = queue_.enqueueNDRangeKernel(
			kernel_,
			cl::NullRange,
			cl::NDRange(mdx_, mdy_),
			cl::NullRange,
			NULL,
			&event_);
	queue_.finish();
	after = microsec_clock::universal_time();
	err_ = queue_.enqueueReadImage(
			cl_image_out_,
			CL_TRUE,
			origin_,
			region_,
			mdx_ * sizeof(char),
			0,
```

```
            (void*)&out[0]);
    queue_.finish();
    return (after - before);
}
```

Cl_julia.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef CL_JULIA_HEADER_DEFINED
#define CL_JULIA_HEADER_DEFINED

class cl_julia {
private :
    cl::Buffer cl_buffer_out_;
    cl::Image2D cl_image_out_;
    size_t buffer_size_;
    size_t total_size_;
    cl::size_t<3> origin_;
    cl::size_t<3> region_;
    cl_uint mdx_;
    cl_uint mdy_;
    cl_float2 cl_c_;
    cl_uint max_iterations_;
    unsigned int device_used_;
    std::vector<cl::Device> devices_;
    cl::Context context_;
    cl::CommandQueue queue_;
    cl::Program program_;
    cl::Kernel kernel_;
    // debugging variables
    cl_int err_;
    cl::Event event_;
public :
    cl_julia(bool gpu);
    void init();
    void setup(
            const std::pair<float, float>& c,
            const std::pair<unsigned int, unsigned int>& s,
            unsigned int max_iterations);
    void prepare_buffer();
    void prepare_image();
    boost::posix_time::time_duration run_buffer(std::vector<float>& out);
    boost::posix_time::time_duration run_image(std::vector<char>& out);
```

```
};

#endif // CL_JULIA_HEADER_DEFINED
```

Julia.cl

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

float2 square_c(float2 c) {
     return (float2)(c.x * c.x - c.y * c.y, 2.0f * c.y * c.x);
}

float2 mult_c(float2 c1, float2 c2) {
     return (float2)(c1.x * c2.x - c1.y * c2.y, c1.y * c2.x + c1.x * c2.y);
}

float2 add_c(float2 c1, float2 c2) {
     return (float2)(c1.x + c2.x, c1.y + c2.y);
}

float square_norm_c(float2 c) {
     return c.x * c.x + c.y * c.y;
}

__kernel void julia_buffer(
     __global float* out,
     float2 c,
     uint max_iteration)
{
     const uint2 d = (uint2)(get_global_id(0), get_global_id(1));
     const uint2 m = (uint2)(get_global_size(0), get_global_size(1));
     float2 pos = (float2)(
             (((float)d.x / (float)m.x) * 4.0f) - 2.0f,
             (((float)d.y / (float)m.y) * 4.0f) - 2.0f);
     uint ite = 0;
     while ((square_norm_c(pos) < 4.0f) && (ite < max_iteration)) {
             pos = add_c(square_c(pos), c);
             ite++;
     }
     float val = (float)ite / (float)max_iteration;
     out[d.y * m.x + d.x] = val;
}

__kernel void julia_image(
```

```
        __write_only image2d_t out,
        float2 c,
        uint max_iteration)
{
        const sampler_t format =
                CLK_NORMALIZED_COORDS_FALSE |
                CLK_FILTER_NEAREST |
                CLK_ADDRESS_CLAMP;
        const int2 d = (int2)(get_global_id(0), get_global_id(1));
        const int2 m = (int2)(get_global_size(0), get_global_size(1));
        float2 pos = (float2)(
                (((float)d.x / (float)m.x) * 4.0f) - 2.0f,
                (((float)d.y / (float)m.y) * 4.0f) - 2.0f);
        uint ite = 0;
        while ((square_norm_c(pos) < 4.0f) && (ite < max_iteration)) {
                pos = add_c(square_c(pos), c);
                ite++;
        }
        float val = (float)ite / (float)max_iteration;
        write_imagef(out, d, (float4)(val, val, val, val));
}
```

Main.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <iostream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/program_options.hpp>
#include <boost/filesystem.hpp>

#include "cl_julia.h"
#include "cl_util.h"
#include "glut_win.h"
#include "win_julia.h"

using namespace cl;
using namespace boost::posix_time;
using namespace boost::program_options;

int main(int ac, char** av) {
```

```
bool no_window = false;
bool enable_gpu = true;
bool enable_image = false;
unsigned int max_height = 512;
unsigned int max_width = 512;
unsigned int max_iterations = 1000;
unsigned int nb_loops = 1000;
float cx = -0.1f;
float cy = 0.651f;
try {
        options_description desc("Allowed options");
        desc.add_options()
               ("help,h", "produce help message")
               ("opencl-cpu,c", "compute using openCL on CPU")
               ("opencl-gpu,g", "compute using openCL on GPU")
               ("image,m", "use OpenCL Image2D instead of buffer")
               ("width,w", value<unsigned int>(), "image width")
               ("height,t", value<unsigned int>(), "image height")
               ("iterations,i", value<unsigned int>(), "julia iterations")
               ("loop,l", value<unsigned int>(),
                      "number of loops (only in no-window mode)")
               ("cx,x", value<float>(), "value of c(x)")
               ("cy,y", value<float>(), "value of c(y)")
               ("no-window,n", "no window output")
        ;
        variables_map vm;
        store(command_line_parser(ac, av).options(desc).run(), vm);
        if (vm.count("help")) {
               std::cout << desc << std::endl;
               return 1;
        }
        if (vm.count("opencl-cpu")) {
               enable_gpu = false;
               std::cout << "OpenCL (CPU)    : enable" << std::endl;
        } else if (vm.count("opencl-gpu")) {
               enable_gpu = true;
               std::cout << "OpenCL (GPU)    : enable" << std::endl;
        }
        if (vm.count("width")) {
               max_width = vm["width"].as<unsigned int>();
        }
        std::cout << "Width           : " << max_width << std::endl;
        if (vm.count("height")) {
               max_height = vm["height"].as<unsigned int>();
        }
        std::cout << "Height          : " << max_height << std::endl;
        if (vm.count("iterations")) {
               max_iterations = vm["iterations"].as<unsigned int>();
        }
        std::cout << "Iterations      : " << max_iterations << std::endl;
        if (vm.count("loop")) {
               nb_loops = vm["loop"].as<unsigned int>();
        }
        std::cout << "Loops           : " << nb_loops << std::endl;
               if (vm.count("cx")) {
               cx = vm["cx"].as<float>();
        }
        std::cout << "Cx              : " << cx << std::endl;
        if (vm.count("cy")) {
               cy = vm["cy"].as<float>();
        }
        std::cout << "Cy              : " << cy << std::endl;
        if (vm.count("no-window")) {
               no_window = true;
               std::cout << "Window          : no" << std::endl;
        } else {
               std::cout << "Window          : yes" << std::endl;
        }
        if (vm.count("image")) {
               enable_image = true;
```

```
                        std::cout << "OpenCL Image2D  : enable" << std::endl;
                } else {
                        enable_image = false;
                        std::cout << "OpenCL Image2D  : disable" << std::endl;
                }
        } catch (std::exception& e) {
                std::cerr << "exception     : " << e.what() << std::endl;
                return -1;
        }
        try {
                if (!no_window) {
                        win_julia wjulia(
                                std::make_pair<float, float>(cx, cy),
                                std::make_pair<unsigned int, unsigned int>(max_width,
max_height),
                                enable_gpu,
                                enable_image,
                                max_iterations);
                        glut_win* pwin = glut_win::instance(
                                std::string("Julia"),
                                std::make_pair<unsigned int, unsigned int>(max_width,
max_height),
                                &wjulia);
                        pwin->run();
                } else {
                        std::vector<float> vec_buffer(max_width * max_height);
                        std::vector<char> vec_image(max_width * max_height);
                        cl_julia* pjulia = new cl_julia(enable_gpu);
                        pjulia->init();
                        pjulia->setup(
                                std::make_pair<float, float>(cx, cy),
                                std::make_pair<unsigned int, unsigned int>(max_height,
max_width),
                                max_iterations);
                        time_duration best_time = minutes(60);
                        for (int i = 0; i < nb_loops; ++i) {
                                time_duration actual_time;
                                if (enable_image) {
                                        pjulia->prepare_image();
                                        actual_time = pjulia->run_image(vec_image);
                                } else {
                                                pjulia->prepare_buffer();
                                        actual_time = pjulia->run_buffer(vec_buffer);
                                }
                                if (actual_time < best_time) best_time = actual_time;
                                std::cout
                                        << "\riteration [" << i + 1 << "/" << nb_loops << "] :
"
                                        << "actual_time : " << actual_time;
                                std::cout.flush();
                        }
                        std::cout << std::endl << "Finished : [" << nb_loops << "] Best
time : "
                                << best_time << std::endl;
                        delete pjulia;
                }
        } catch (cl::Error er) {
                std::cerr << "exception (CL)  : " << er.what() << "(" << er << ")" <<
std::endl;
                return -1;
        } catch (std::exception& ex) {
                std::cerr << "exception (STL) : " << ex.what() << std::endl;
                return -2;
        }
        return 0;
}
```

Win_julia.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdexcept>
#include <string>
#include <vector>
#ifdef __linux__
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#endif
#ifdef __APPLE__
#include <glut/glut.h>
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#endif
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

#include "cl_julia.h"
#include "glut_win.h"
#include "win_julia.h"

using namespace boost::posix_time;

win_julia::win_julia(
     const std::pair<float, float>& c,
     const std::pair<unsigned int, unsigned int>& range,
     bool gpu,
     bool image,
     unsigned int max_iterations)
     :       c_(c),
             range_(range),
             max_iterations_(max_iterations),
             p_julia_(NULL),
             gpu_(gpu),
             image_(image)
{
     best_time_ = minutes(60);
}

win_julia::~win_julia() {}

void win_julia::init() {
     glClearColor(0, 0, 0, 0);
```

```
	gluOrtho2D(-1, 1, -1, 1);
	glGenTextures(1, &texture_id_);
	p_julia_ = new cl_julia(gpu_);
	p_julia_->init();
	p_julia_->setup(c_, range_, max_iterations_);
}

void win_julia::display() {
	glClear(GL_COLOR_BUFFER_BIT);
	glEnable(GL_TEXTURE_2D);
	glBindTexture(GL_TEXTURE_2D, texture_id_);
	glPushMatrix();
		glBegin(GL_QUADS);
			glTexCoord2f(0, 1);
			glVertex2f(-1, 1);
			glTexCoord2f(1, 1);
			glVertex2f(1, 1);
			glTexCoord2f(1, 0);
			glVertex2f(1, -1);
			glTexCoord2f(0, 0);
			glVertex2f(-1, -1);
		glEnd();
	glPopMatrix();
	glDisable(GL_TEXTURE_2D);
	glFlush();
	glutPostRedisplay();
}

void win_julia::idle() {
	glFinish();
	if (p_julia_) {
		time_duration actual_time;
		if (image_) {
			p_julia_->prepare_image();
			actual_time = p_julia_->run_image(current_image_);
		} else {
			p_julia_->prepare_buffer();
			actual_time = p_julia_->run_buffer(current_buffer_);
		}
		if (actual_time < best_time_) best_time_ = actual_time;
		std::cout << "\rCompute time    : " << actual_time << " best " <<
best_time_;
		std::cout.flush();
	}
	glBindTexture(GL_TEXTURE_2D, texture_id_);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	if (image_) {
		glTexImage2D(
			GL_TEXTURE_2D,
			0,
			1,
			range_.first,
			range_.second,
			0,
			GL_LUMINANCE,
			GL_UNSIGNED_BYTE,
			&current_image_[0]);
	} else {
		glTexImage2D(
			GL_TEXTURE_2D,
			0,
			1,
			range_.first,
			range_.second,
			0,
			GL_LUMINANCE,
			GL_FLOAT,
			&current_buffer_[0]);
	}
```

```
	glFinish();
}

void win_julia::reshape(int w, int h) {
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glViewport(0, 0, w, h);
	glMatrixMode(GL_MODELVIEW);
	glFinish();
}

void win_julia::mouse_event(int button, int state, int x, int y) {}

void win_julia::mouse_move(int x, int y) {}

void win_julia::keyboard(unsigned char key, int x, int y) {}

void win_julia::finish() {
	if (p_julia_) {
		delete p_julia_;
		p_julia_ = NULL;
	}
	std::cout << std::endl;
}
```

Win_julia.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef WIN_JULIA_HEADER_DEFINED
#define WIN_JULIA_HEADER_DEFINED

class win_julia : public i_win {
private :
	std::pair<float, float> c_;
	std::pair<unsigned int, unsigned int> range_;
	unsigned int max_iterations_;
	std::vector<float> current_buffer_;
	std::vector<char> current_image_;
	cl_julia* p_julia_;
	bool gpu_;
	bool image_;
	unsigned int texture_id_;
	boost::posix_time::time_duration best_time_;
```

```
public :
     // constructor size of image and image pointer
     win_julia(
             const std::pair<float, float>& c,
             const std::pair<unsigned int, unsigned int>& range,
             bool gpu,
             bool image,
             unsigned int max_iterations);
     virtual ~win_julia();
public :
     // inherited from the i_win interface
     virtual void init();
     virtual void display();
     virtual void idle();
     virtual void reshape(int w, int h);
     virtual void mouse_event(int button, int state, int x, int y);
     virtual void mouse_move(int x, int y);
     virtual void keyboard(unsigned char key, int x, int y);
     virtual void finish();
};

#endif // WIN_JULIA_HEADER_DEFINED
```

Cellular automata

```
├── cellular_automata
│   ├── Makefile
│   ├── cl_cellular_automata.cpp
│   ├── cl_cellular_automata.h
│   ├── cl_util.cpp
│   ├── cl_util.h
│   ├── glut_win.cpp
│   ├── glut_win.h
│   ├── life.cl
│   ├── main.cpp
│   ├── win_cellular_automata.cpp
│   └── win_cellular_automata.h
```

Makefile

```
# a simple makefile for cellular automata


ifeq ($(OSTYPE), darwin)
CC = clang
CXX = clang++
CFLAGS = -g -O0 -I/opt/local/include -I/usr/local/include -I..
LIBS = -L/opt/local/lib/ \
          -framework GLUT -framework Cocoa -framework OpenCL -framework OpenGL \
          -lboost_program_options -lboost_system
else
CC = gcc
CXX = g++
CFLAGS = -g -O0 -I/usr/include -I/usr/local/include -I..
LIBS = -L/usr/lib64 -L/usr/local/lib \
          -lglut -lGL -lGLU -lOpenCL \
          -lboost_program_options -lboost_system
```

```
    endif
    ALL=cellular_automata


    all: $(ALL)


    cl_util.o : cl_util.cpp cl_util.h
              $(CXX) -o cl_util.o -c cl_util.cpp $(CFLAGS)
    glut_win.o : glut_win.cpp glut_win.h
              $(CXX) -o glut_win.o -c glut_win.cpp $(CFLAGS)
    cl_cellular_automata.o : cl_cellular_automata.cpp cl_cellular_automata.h
              $(CXX) -o cl_cellular_automata.o -c cl_cellular_automata.cpp $(CFLAGS)
    win_cellular_automata.o : win_cellular_automata.cpp win_cellular_automata.h
              $(CXX) -o win_cellular_automata.o -c win_cellular_automata.cpp $(CFLAGS)
    main.o : main.cpp win_cellular_automata.h cl_cellular_automata.h
              $(CXX) -o main.o -c main.cpp $(CFLAGS)


    cellular_automata : main.o cl_util.o glut_win.o cl_cellular_automata.o
win_cellular_automata.o
              $(CXX) -o cellular_automata main.o cl_util.o glut_win.o
cl_cellular_automata.o win_cellular_automata.o $(LIBS)


    clean:
              rm -f $(ALL) *.o
```

Cl_cellular_automata.cpp

```
/*
 * Copyright (c) 2012, Frederic DUBOUCHET
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic DUBOUCHET ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <iostream>
#include <vector>
```

```
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#ifdef __linux__
#include <GL/glx.h>
#endif
#include <boost/date_time/posix_time/posix_time.hpp>

#include "cl_cellular_automata.h"

using namespace boost::posix_time;

cl_cellular_automata::cl_cellular_automata(bool gpu) {
	//setup devices_ and context_
	std::vector<cl::Platform> platforms;
	err_ = cl::Platform::get(&platforms);
	device_used_ = 0;
	err_ = platforms[0].getDevices((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU,
&devices_);
	int t = devices_.front().getInfo<CL_DEVICE_TYPE>();
	try {
		throw cl::Error(0, "cheat pass");
		#if defined (__APPLE__) || defined(MACOSX)
			CGLContextObj kCGLContext = CGLGetCurrentContext();
			CGLShareGroupObj kCGLShareGroup = CGLGetShareGroup(kCGLContext);
			cl_context_properties props[] =
			{
				CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
(cl_context_properties)kCGLShareGroup,
				0
			};
			context_ = cl::Context(props);
		#else
			#if defined WIN32 // Win32
				cl_context_properties props[] =
				{
					CL_GL_CONTEXT_KHR,
(cl_context_properties)wglGetCurrentContext(),
					CL_WGL_HDC_KHR,
(cl_context_properties)wglGetCurrentDC(),
					CL_CONTEXT_PLATFORM, (cl_context_properties)
(platforms[0])(),
					0
				};
				context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
			#else
				throw cl::Error(0, "cheat pass");
				cl_context_properties props[] =
				{
					CL_GL_CONTEXT_KHR,
(cl_context_properties)glXGetCurrentContext(),
					CL_GLX_DISPLAY_KHR,
(cl_context_properties)glXGetCurrentDisplay(),
					CL_CONTEXT_PLATFORM, (cl_context_properties)
(platforms[0])(),
					0
				};
				context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
			#endif
		#endif
	} catch (cl::Error er) {
		std::cerr << "Warning         : could not attach GL and CL together!" <<
std::endl;
		cl_context_properties properties[] = { CL_CONTEXT_PLATFORM,
(cl_context_properties)(platforms[0])(), 0};
		context_ = cl::Context((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU ,
properties);
		devices_ = context_.getInfo<CL_CONTEXT_DEVICES>();
	}
	queue_ = cl::CommandQueue(context_, devices_[device_used_], 0, &err_);
}
```

```
void cl_cellular_automata::init(const std::string& cl_file) {
	FILE* file = fopen(cl_file.c_str(), "rt");
	if (!file) throw std::runtime_error("could not open file " + cl_file);
	const unsigned int BUFFER_SIZE = 4096;
	size_t bytes_read = 0;
	std::string kernel__source = "";
	do {
		char temp[BUFFER_SIZE];
		memset(temp, 0, BUFFER_SIZE);
		bytes_read = fread(temp, sizeof(char), BUFFER_SIZE, file);
		kernel__source += temp;
	} while (bytes_read != 0);
	fclose(file);
	cl::Program::Sources source(
		1,
		std::make_pair(kernel__source.c_str(),
		kernel__source.size()));
	program_ = cl::Program(context_, source);
	try {
		err_ = program_.build(devices_);
	} catch (cl::Error er) {
		std::cerr << "build status    : "
			<< program_.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(devices_[0]) <<
std::endl;
		std::cerr << "build options   : "
			<< program_.getBuildInfo<CL_PROGRAM_BUILD_OPTIONS>(devices_[0]) <<
std::endl;
		std::cerr << "build log       : "  << std::endl
			<< program_.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices_[0]) <<
std::endl;
		throw er;
	}
}

void cl_cellular_automata::setup(
	const std::pair<unsigned int, unsigned int>& s,
	unsigned int max_iterations)
{
	max_iterations_ = max_iterations;
	mdx_ = s.first;
	mdy_ = s.second;
}

void cl_cellular_automata::prepare_buffer(const std::vector<float>& in) {
	total_size_ = mdx_ * mdy_;
	if (in.size() != (total_size_))
		throw std::runtime_error("input buffer size != image size!");
	kernel_ = cl::Kernel(program_, "life_buffer", &err_);
	//initialize our CPU memory arrays, send them to the device and set the kernel_
arguements
	cl_buffer_in_ = cl::Buffer(
		context_,
		CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
		sizeof(float) * total_size_,
		(void*)&in[0],
		&err_);
	cl_buffer_out_ = cl::Buffer(
		context_,
		CL_MEM_WRITE_ONLY,
		sizeof(float) * total_size_,
		NULL,
		&err_);
	queue_.finish();
	//set the arguements of our kernel_
	err_ = kernel_.setArg(0, cl_buffer_in_);
	err_ = kernel_.setArg(1, cl_buffer_out_);
	err_ = kernel_.setArg(2, max_iterations_);
	//Wait for the command queue_ to finish these commands before proceeding
	queue_.finish();
```

```
}

void cl_cellular_automata::prepare_image(const std::vector<char>& in) {
	total_size_ = mdx_ * mdy_;
	origin_.push_back(0);
	origin_.push_back(0);
	origin_.push_back(0);
	region_.push_back(mdx_);
	region_.push_back(mdy_);
	region_.push_back(1);
	if (in.size() != (total_size_))
		throw std::runtime_error("input buffer size != image size!");
	kernel_ = cl::Kernel(program_, "life_image", &err_);
	//initialize our CPU memory arrays, send them to the device and set the kernel_
arguements
	cl::ImageFormat format = cl::ImageFormat(CL_INTENSITY, CL_UNORM_INT8);
	cl_image_in_ = cl::Image2D(
		context_,
		CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
		format,
		mdx_,
		mdy_,
		0,
		(void*)&in[0],
		&err_);
	cl_image_out_ = cl::Image2D(
		context_,
		CL_MEM_WRITE_ONLY,
		format,
		mdx_,
		mdy_,
		0,
		NULL,
		&err_);
	queue_.finish();
	//set the arguements of our kernel_
	err_ = kernel_.setArg(0, cl_image_in_);
	err_ = kernel_.setArg(1, cl_image_out_);
	err_ = kernel_.setArg(2, max_iterations_);
	//Wait for the command queue_ to finish these commands before proceeding
	queue_.finish();
}

time_duration cl_cellular_automata::run_buffer(std::vector<float>& out) {
	ptime before;
	ptime after;
	if (out.size() != total_size_)
		out.resize(total_size_);
	before = microsec_clock::universal_time();
	for (int i = 0; i < max_iterations_; ++i) {
		// make the computation
		err_ = queue_.enqueueNDRangeKernel(
			kernel_,
			cl::NullRange,
				cl::NDRange(mdx_, mdy_),
			cl::NullRange,
			NULL,
			&event_);
		queue_.finish();
		// swap the buffers (except for the last time)
		if (i < (max_iterations_ - 1)) {
			queue_.enqueueCopyBuffer(
				cl_buffer_out_,
				cl_buffer_in_,
				0,
				0,
				total_size_,
				NULL,
				&event_);
			queue_.finish();
```

```
                }
        }
        after = microsec_clock::universal_time();
        err_ = queue_.enqueueReadBuffer(
                cl_buffer_out_,
                CL_TRUE,
                0,
                sizeof(float) * total_size_,
                &out[0],
                NULL,
                &event_);
        queue_.finish();
        return (after - before);
}

time_duration cl_cellular_automata::run_image(std::vector<char>& out) {
        ptime before;
        ptime after;
        if (out.size() != total_size_)
                out.resize(total_size_);
        before = microsec_clock::universal_time();
        for (int i = 0; i < max_iterations_; ++i) {
                // make the computation
                err_ = queue_.enqueueNDRangeKernel(
                        kernel_,
                        cl::NullRange,
                        cl::NDRange(mdx_, mdy_),
                        cl::NullRange,
                        NULL,
                        &event_);
                queue_.finish();
                // copy the in out in the in buffer (except for the last time)
                if (i < (max_iterations_ - 1)) {
                        queue_.enqueueCopyImage(
                                cl_image_out_,
                                cl_image_in_,
                                origin_,
                                origin_,
                                region_,
                                NULL,
                                &event_);
                        queue_.finish();
                }
        }
        after = microsec_clock::universal_time();
        err_ = queue_.enqueueReadImage(
                cl_image_out_,
                CL_TRUE,
                origin_,
                region_,
                mdx_ * sizeof(char),
                0,
                (void*)&out[0]);
        queue_.finish();
        return (after - before);
}
```

Cl_cellular_automata.h

```
/*
 * Copyright (c) 2012, Frederic DUBOUCHET
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
```

```
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic DUBOUCHET ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef CL_cellular_automata_HEADER_DEFINED
#define CL_cellular_automata_HEADER_DEFINED

class cl_cellular_automata {
private :
     cl::Buffer cl_buffer_out_;
     cl::Buffer cl_buffer_in_;
     cl::Image2D cl_image_out_;
     cl::Image2D cl_image_in_;
     cl::size_t<3> region_;
     cl::size_t<3> origin_;
     size_t buffer_size_;
     size_t total_size_;
     cl_uint mdx_;
     cl_uint mdy_;
     cl_uint max_iterations_;
     unsigned int device_used_;
     std::vector<cl::Device> devices_;
     cl::Context context_;
     cl::CommandQueue queue_;
     cl::Program program_;
     cl::Kernel kernel_;
     // debugging variables
     cl_int err_;
     cl::Event event_;
public :
     cl_cellular_automata(bool gpu);
     void init(const std::string& cl_file);
     // if change size
     void setup(
             const std::pair<unsigned int, unsigned int>& s,
             unsigned int max_iterations = 1);
     // then before the next iteration
     void prepare_image(const std::vector<char>& in);
     void prepare_buffer(const std::vector<float>& in);
     // run the actual next step
     boost::posix_time::time_duration run_image(std::vector<char>& out);
     boost::posix_time::time_duration run_buffer(std::vector<float>& out);
};

#endif // CL_cellular_automata_HEADER_DEFINED
```

Life.cl

```
/*
 * Copyright (c) 2012, Frederic DUBOUCHET
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
```

```
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic DUBOUCHET ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

int pos2linear(
      const int2 position,
      const int2 m)
{
      return position.y * m.x + position.x;
}

int add(__global const float* in, const int2 position, const int2 m) {
      // empty border
      return
              ((position.x < 0 || position.x >= m.x) || (position.y < 0 || position.y >=
m.y))
                    ? 0.0f
                    : (in[pos2linear(position, m)]) ? 1.0f : 0.0f;
}

__kernel void life_buffer(
      __global float* in,
      __global float* out,
      unsigned int max_iterations)
{
      const int2 d = (int2)(get_global_id(0), get_global_id(1));
      const int2 m = (int2)(get_global_size(0), get_global_size(1));

      int position = pos2linear(d, m);
      int sum = 0;
      sum += add(in, (int2)(d.x - 1, d.y + 1), m);
      sum += add(in, (int2)(d.x    , d.y + 1), m);
      sum += add(in, (int2)(d.x + 1, d.y + 1), m);
      sum += add(in, (int2)(d.x - 1, d.y    ), m);
      sum += add(in, (int2)(d.x + 1, d.y    ), m);
      sum += add(in, (int2)(d.x - 1, d.y - 1), m);
      sum += add(in, (int2)(d.x    , d.y - 1), m);
      sum += add(in, (int2)(d.x + 1, d.y - 1), m);
      // wait for the other task to finish calculating
      barrier(CLK_GLOBAL_MEM_FENCE);
      out[position] = ((sum == 3) || (in[position] && (sum == 2))) ? 1.0f : 0.0f;
}

__kernel void life_image(
      __read_only image2d_t in,
      __write_only image2d_t out,
      unsigned int max_iterations)
{
      const sampler_t format =
              CLK_NORMALIZED_COORDS_FALSE |
              CLK_FILTER_NEAREST |
              CLK_ADDRESS_CLAMP;
      const int2 d = (int2)(get_global_id(0), get_global_id(1));
      const float4 white = (float4)(1.0f, 1.0f, 1.0f, 1.0f);
      const float4 black = (float4)(0.0f, 0.0f, 0.0f, 0.0f);
```

```
	float4 center_col = read_imagef(in, format, d);
	float4 acc = (float4)(0.0f, 0.0f, 0.0f, 0.0f);
	acc += read_imagef(in, format, (int2)(d.x - 1, d.y + 1));
	acc += read_imagef(in, format, (int2)(d.x    , d.y + 1));
	acc += read_imagef(in, format, (int2)(d.x + 1, d.y + 1));
	acc += read_imagef(in, format, (int2)(d.x - 1, d.y    ));
	acc += read_imagef(in, format, (int2)(d.x + 1, d.y    ));
	acc += read_imagef(in, format, (int2)(d.x - 1, d.y - 1));
	acc += read_imagef(in, format, (int2)(d.x    , d.y - 1));
	acc += read_imagef(in, format, (int2)(d.x + 1, d.y - 1));
	// wait for the other task to finish calculating
	barrier(CLK_GLOBAL_MEM_FENCE);
	// compute futur state
	float4 col = ((acc.x == 3.0f) || ((center_col.x != 0.0f) && (acc.x == 2.0f)))
		? white
		: black;
	write_imagef(out, d, col);
}
```

Main.cpp

```
/*
 * Copyright (c) 2012, Frederic DUBOUCHET
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic DUBOUCHET ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <iostream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/program_options.hpp>
#include <boost/filesystem.hpp>

#include "cl_util.h"
#include "glut_win.h"
#include "cl_cellular_automata.h"
#include "win_cellular_automata.h"

using namespace cl;
using namespace boost::program_options;
using namespace boost::posix_time;

int main(int ac, char** av) {
	bool no_window = false;
	bool enable_gpu = true;
	bool enable_image = false;
```

```
unsigned int max_height = 512;
unsigned int max_width = 512;
unsigned int max_iterations = 10;
unsigned int nb_loops = 100;
std::string cl_file = "life.cl";
try {
        // parse command line
        options_description desc("Allowed options");
        desc.add_options()
               ("help,h", "produce help message")
               ("opencl-cpu,c", "compute using openCL on CPU")
               ("opencl-gpu,g", "compute using openCL on GPU (default)")
               ("image,m", "use OpenCL Image2D instead of buffer")
               ("width,w", value<unsigned int>(), "image width")
               ("height,t", value<unsigned int>(), "image height")
               ("iterations,i", value<unsigned int>(), "julia iterations")
               ("cl-file,f", value<std::string>(),
                      "OpenCL file to be compiled (default : life.cl)")
               ("loop,l", value<unsigned int>(),
                      "number of loops (only in no-window mode)")
               ("no-window,n", "no window output")
        ;
        variables_map vm;
        store(command_line_parser(ac, av).options(desc).run(), vm);
        if (vm.count("help")) {
               std::cout << desc << std::endl;
               return 1;
        }
        if (vm.count("opencl-cpu")) {
               enable_gpu = false;
               std::cout << "OpenCL (CPU)    : enable" << std::endl;
        }
        if (vm.count("opencl-gpu")) {
               enable_gpu = true;
               std::cout << "OpenCL (GPU)    : enable" << std::endl;
        }
        if (vm.count("cl-file")) {
               cl_file = vm["cl-file"].as<std::string>();
               std::cout << "OpenCL file     : " << cl_file << std::endl;
        }
        if (vm.count("width")) {
               max_width = vm["width"].as<unsigned int>();
        }
        std::cout << "Width           : " << max_width << std::endl;
        if (vm.count("height")) {
               max_height = vm["height"].as<unsigned int>();
        }
        std::cout << "Height          : " << max_height << std::endl;
        if (vm.count("iterations")) {
               max_iterations = vm["iterations"].as<unsigned int>();
        }
        std::cout << "Iterations      : " << max_iterations << std::endl;
        if (vm.count("no-window")) {
               no_window = true;
               std::cout << "Window          : no" << std::endl;
        } else {
               std::cout << "Window          : yes" << std::endl;
        }
        if (vm.count("loop")) {
               nb_loops = vm["loop"].as<unsigned int>();
        }
        if (vm.count("image")) {
               enable_image = true;
               std::cout << "OpenCL Image2D  : enable" << std::endl;
        } else {
               enable_image = false;
               std::cout << "OpenCL Image2D  : disable" << std::endl;
        }
        // create the initial state (random)
        std::vector<float> initial_buffer;
```

```
		std::vector<char> initial_image;
		if (enable_image) {
			initial_image.resize(max_width * max_height);
			for (int i = 0; i < max_height * max_width; ++i) {
				initial_image[i] = (char)(((random() % 2) == 0) ? 0xff :
0x0000);
			}
		} else {
			initial_buffer.resize(max_width * max_height);
			for (int i = 0; i < max_height * max_width; ++i)
				initial_buffer[i] = (float)(((random() % 2) == 0) ? 1.0f :
0.0f);
		}
		// do the actual job
		if (!no_window) {
			win_cellular_automata* pwca = NULL;
			if (enable_image)
				pwca = new win_cellular_automata(
					std::make_pair<unsigned int, unsigned int>(max_width,
max_height),
					initial_image,
					cl_file,
					enable_gpu,
					max_iterations);
			else
				pwca = new win_cellular_automata(
					std::make_pair<unsigned int, unsigned int>(max_width,
max_height),
					initial_buffer,
					cl_file,
					enable_gpu,
					max_iterations);
			glut_win* pwin = glut_win::instance(
				std::string("Cellular Automata"),
				std::make_pair<unsigned int, unsigned int>(max_width,
max_height),
				pwca);
			pwin->run();
			delete pwca;
		} else {
			cl_cellular_automata cca(enable_gpu);
			cca.init("life.cl");
			cca.setup(
				std::make_pair<unsigned int, unsigned int>(max_width,
max_height),
				max_iterations);
			time_duration best_time = minutes(60);
			for (int i = 0; i < nb_loops; ++i) {
				time_duration actual_time;
				if (enable_image) {
					cca.prepare_image(initial_image);
					actual_time = cca.run_image(initial_image);
				} else {
					cca.prepare_buffer(initial_buffer);
					actual_time = cca.run_buffer(initial_buffer);
				}
				if (actual_time < best_time) best_time = actual_time;
				std::cout
					<< "\riteration [" << i + 1 << "/" << nb_loops << "] :
"
					<< "actual_time : " << actual_time;
				std::cout.flush();
			}
			std::cout << std::endl
				<< "Finished : [" << nb_loops << "] Best time : " << best_time
				<< std::endl;
		}
	// error handling
	} catch (cl::Error er) {
		std::cerr
```

```
                << "exception (CL)  : " << er.what()
                << "(" << er << ")" << std::endl;
            return -2;
    } catch (std::exception& ex) {
            std::cerr << "exception (std) : " << ex.what() << std::endl;
            return -1;
    }
    return 0;
}
```

Win_cellular_automata.cpp

```
/*
 * Copyright (c) 2012, Frederic DUBOUCHET
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic DUBOUCHET ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdexcept>
#include <string>
#include <vector>
#ifdef __linux__
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#endif
#ifdef __APPLE__
#include <glut/glut.h>
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#endif
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

#include "cl_cellular_automata.h"
#include "glut_win.h"
#include "win_cellular_automata.h"

using namespace boost::posix_time;

win_cellular_automata::win_cellular_automata(
      const std::pair<unsigned int, unsigned int>& range,
      const std::vector<float>& initial_buffer,
      const std::string& cl_file,
      bool gpu,
      unsigned int max_iterations)
      :       range_(range),
```

```
                max_iterations_(max_iterations),
                gpu_(gpu),
                image_(false),
                p_cellular_automata_(NULL),
                cl_file_(cl_file)
{
      current_buffer_ = initial_buffer;
      best_time_ = minutes(60);
}

win_cellular_automata::win_cellular_automata(
      const std::pair<unsigned int, unsigned int>& range,
      const std::vector<char>& initial_image,
      const std::string& cl_file,
      bool gpu,
      unsigned int max_iterations)
      :       range_(range),
                max_iterations_(max_iterations),
                gpu_(gpu),
                image_(true),
                p_cellular_automata_(NULL),
                cl_file_(cl_file)
{
      current_image_ = initial_image;
      best_time_ = minutes(60);
}


win_cellular_automata::~win_cellular_automata() {}

void win_cellular_automata::init() {
      glClearColor(0, 0, 0, 0);
      gluOrtho2D(-1, 1, -1, 1);
      glGenTextures(1, &texture_id_);
      p_cellular_automata_ = new cl_cellular_automata(gpu_);
      p_cellular_automata_->init(cl_file_);
      p_cellular_automata_->setup(range_, max_iterations_);
}

void win_cellular_automata::display() {
      glClear(GL_COLOR_BUFFER_BIT);
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, texture_id_);
      glPushMatrix();
                glBegin(GL_QUADS);
                        glTexCoord2f(0, 1);
                        glVertex2f(-1, 1);
                        glTexCoord2f(1, 1);
                        glVertex2f(1, 1);
                        glTexCoord2f(1, 0);
                        glVertex2f(1, -1);
                        glTexCoord2f(0, 0);
                        glVertex2f(-1, -1);
                glEnd();
      glPopMatrix();
      glDisable(GL_TEXTURE_2D);
      glFlush();
      glutPostRedisplay();
}

void win_cellular_automata::idle() {
      glFinish();
      if (p_cellular_automata_) {
                time_duration actual_time;
                if (image_) {
                        p_cellular_automata_->prepare_image(current_image_);
                        actual_time = p_cellular_automata_->run_image(current_image_);
                } else {
                        p_cellular_automata_->prepare_buffer(current_buffer_);
                        actual_time = p_cellular_automata_->run_buffer(current_buffer_);
```

```
                }
                if (actual_time < best_time_) best_time_ = actual_time;
                std::cout << "\rCompute time    : " << actual_time << " best " <<
best_time_;
                std::cout.flush();
        }
        glBindTexture(GL_TEXTURE_2D, texture_id_);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        if (image_) {
                glTexImage2D(
                        GL_TEXTURE_2D,
                        0,
                        1,
                        range_.first,
                        range_.second,
                        0,
                        GL_LUMINANCE,
                        GL_UNSIGNED_BYTE,
                        &current_image_[0]);
        } else {
                glTexImage2D(
                        GL_TEXTURE_2D,
                        0,
                        1,
                        range_.first,
                        range_.second,
                        0,
                        GL_LUMINANCE,
                        GL_FLOAT,
                        &current_buffer_[0]);
        }
        glFinish();
}

void win_cellular_automata::reshape(int w, int h) {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glViewport(0, 0, w, h);
        glMatrixMode(GL_MODELVIEW);
        glFinish();
}

void win_cellular_automata::mouse_event(int button, int state, int x, int y) {}

void win_cellular_automata::mouse_move(int x, int y) {}

void win_cellular_automata::keyboard(unsigned char key, int x, int y) {}

void win_cellular_automata::finish() {
        if (p_cellular_automata_) {
                delete p_cellular_automata_;
                p_cellular_automata_ = NULL;
        }
        std::cout << std::endl;
}
```

Win_cellular_automata.h

```
/*
 * Copyright (c) 2012, Frederic DUBOUCHET
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
```

```
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic DUBOUCHET ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef WIN_cellular_automata_HEADER_DEFINED
#define WIN_cellular_automata_HEADER_DEFINED

class win_cellular_automata : public i_win {
private :
     std::pair<unsigned int, unsigned int> range_;
     unsigned int max_iterations_;
     std::vector<float> current_buffer_;
     std::vector<char> current_image_;
     cl_cellular_automata* p_cellular_automata_;
     bool gpu_;
     bool image_;
     unsigned int texture_id_;
     std::string cl_file_;
     boost::posix_time::time_duration best_time_;
public :
     // constructor size and vector image
     win_cellular_automata(
             const std::pair<unsigned int, unsigned int>& range,
             const std::vector<float>& initial_buffer,
             const std::string& cl_file,
             bool gpu,
             unsigned int max_iterations);
     win_cellular_automata(
             const std::pair<unsigned int, unsigned int>& range,
             const std::vector<char>& initial_image,
             const std::string& cl_file,
             bool gpu,
             unsigned int max_iterations);
     virtual ~win_cellular_automata();
public :
     // inherited from the i_win interface
     virtual void init();
     virtual void display();
     virtual void idle();
     virtual void reshape(int w, int h);
     virtual void mouse_event(int button, int state, int x, int y);
     virtual void mouse_move(int x, int y);
     virtual void keyboard(unsigned char key, int x, int y);
     virtual void finish();
};

#endif // WIN_cellular_automata_HEADER_DEFINED
```

Floyd-Warshall

```
    ├── floyd_warshall
    │   ├── Makefile
    │   ├── cl_floyd_warshall.cpp
    │   ├── cl_floyd_warshall.h
    │   ├── cl_util.cpp
```

```
    │   ├── cl_util.h
    │   ├── ewd_file.cpp
    │   ├── ewd_file.h
    │   ├── floyd_warshall.cl
    │   └── main.cpp
```

Makefile

```
    # a simple makefile for Floyd-Warshall

    ifeq ($(OSTYPE), darwin)
    CC = clang
    CXX = clang++
    CFLAGS = -g -O0 -I/opt/local/include -I/usr/local/include -I..
    LIBS = -L/opt/local/lib/ \
              -framework GLUT -framework Cocoa -framework OpenCL -framework OpenGL \
              -lboost_program_options -lboost_system
    else
    CC = gcc
    CXX = g++
    CFLAGS = -g -O0 -I/usr/include -I/usr/local/include -I..
    LIBS = -L/usr/lib64 -L/usr/local/lib \
              -lglut -lGL -lGLU -lOpenCL \
              -lboost_program_options -lboost_system
    endif
    ALL=floyd_warshall

    all: $(ALL)

    cl_util.o : cl_util.cpp cl_util.h
              $(CXX) -o cl_util.o -c cl_util.cpp $(CFLAGS)
    cl_floyd_warshall.o : cl_floyd_warshall.cpp cl_floyd_warshall.h
              $(CXX) -o cl_floyd_warshall.o -c cl_floyd_warshall.cpp $(CFLAGS)
    ewd_file.o : ewd_file.cpp ewd_file.h
              $(CXX) -o ewd_file.o -c ewd_file.cpp $(FLAGS)
    main.o : main.cpp cl_floyd_warshall.h ewd_file.h cl_util.h
              $(CXX) -o main.o -c main.cpp $(CFLAGS)

    floyd_warshall : main.o cl_util.o cl_floyd_warshall.o ewd_file.o
              $(CXX) -o floyd_warshall main.o cl_util.o ewd_file.o cl_floyd_warshall.o $
(LIBS)

    clean:
              rm -f $(ALL) *.o
```

Cl_floyd_warshall.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <iostream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#ifdef __linux__
#include <GL/glx.h>
#endif
#include <boost/date_time/posix_time/posix_time.hpp>

#include "cl_floyd_warshall.h"

using namespace boost::posix_time;

cl_floyd_warshall::cl_floyd_warshall(bool gpu) {
   //setup devices_ and context_
   std::vector<cl::Platform> platforms;
   err_ = cl::Platform::get(&platforms);
   device_used_ = 0;
   err_ = platforms[0].getDevices((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU,
&devices_);
   int t = devices_.front().getInfo<CL_DEVICE_TYPE>();
   try {
      throw cl::Error(0, "cheat pass");
#if defined (__APPLE__) || defined(MACOSX)
      CGLContextObj kCGLContext = CGLGetCurrentContext();
      CGLShareGroupObj kCGLShareGroup = CGLGetShareGroup(kCGLContext);
      cl_context_properties props[] =
      {
         CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
(cl_context_properties)kCGLShareGroup,
         0
      };
      context_ = cl::Context(props);
#else
#if defined WIN32 // Win32
      cl_context_properties props[] =
      {
         CL_GL_CONTEXT_KHR, (cl_context_properties)wglGetCurrentContext(),
         CL_WGL_HDC_KHR, (cl_context_properties)wglGetCurrentDC(),
```

```
         CL_CONTEXT_PLATFORM, (cl_context_properties)(platforms[0])(),
         0
      };
      context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
#else
      throw cl::Error(0, "cheat pass");
      cl_context_properties props[] =
      {
         CL_GL_CONTEXT_KHR, (cl_context_properties)glXGetCurrentContext(),
         CL_GLX_DISPLAY_KHR, (cl_context_properties)glXGetCurrentDisplay(),
         CL_CONTEXT_PLATFORM, (cl_context_properties)(platforms[0])(),
         0
      };
      context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
#endif
#endif
   } catch (cl::Error er) {
      std::cerr << "Warning         : could not attach GL and CL toghether!" <<
std::endl;
      cl_context_properties properties[] = { CL_CONTEXT_PLATFORM,
(cl_context_properties)(platforms[0])(), 0};
      context_ = cl::Context((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU ,
properties);
      devices_ = context_.getInfo<CL_CONTEXT_DEVICES>();
   }
   queue_ = cl::CommandQueue(context_, devices_[device_used_], 0, &err_);
}

void cl_floyd_warshall::init(const std::string& cl_file) {
   FILE* file = fopen(cl_file.c_str(), "rt");
   if (!file) throw std::runtime_error("could not open file " + cl_file);
   const unsigned int BUFFER_SIZE = 4096;
   size_t bytes_read = 0;
   std::string kernel__source = "";
   do {
      char temp[BUFFER_SIZE];
      memset(temp, 0, BUFFER_SIZE);
      bytes_read = fread(temp, sizeof(char), BUFFER_SIZE, file);
      kernel__source += temp;
   } while (bytes_read != 0);
   fclose(file);
   cl::Program::Sources source(
         1,
         std::make_pair(kernel__source.c_str(),
            kernel__source.size()));
   program_ = cl::Program(context_, source);
   try {
      err_ = program_.build(devices_);
   } catch (cl::Error er) {
      std::cerr << "build status    : "
         << program_.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(devices_[0]) << std::endl;
      std::cerr << "build options   : "
         << program_.getBuildInfo<CL_PROGRAM_BUILD_OPTIONS>(devices_[0]) << std::endl;
      std::cerr << "build log       : "  << std::endl
         << program_.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices_[0]) << std::endl;
      throw er;
   }
}

void cl_floyd_warshall::setup(const std::pair<unsigned int, unsigned int>& s) {
   mdx_ = s.first;
   mdy_ = s.second;
}

void cl_floyd_warshall::prepare(const std::vector<float>& in) {
   total_size_ = mdx_ * mdy_;
   if (in.size() != (total_size_))
      throw std::runtime_error("input buffer size != image size!");
   kernel_ = cl::Kernel(program_, "floyd_warshall_buffer", &err_);
```

```
   //initialize our CPU memory arrays, send them to the device and set the kernel_
arguements
   cl_buffer_mat_ = cl::Buffer(
         context_,
         CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
         sizeof(float) * total_size_,
         (void*)&in[0],
         &err_);
   cl_buffer_in_x_ = cl::Buffer(
         context_,
         CL_MEM_READ_ONLY,
         sizeof(float) * mdx_,
         NULL,
         &err_);
   cl_buffer_in_y_ = cl::Buffer(
         context_,
         CL_MEM_READ_ONLY,
         sizeof(float) * mdy_,
         NULL,
         &err_);
   queue_.finish();
   //set the arguements of our kernel_
   err_ = kernel_.setArg(0, cl_buffer_mat_);
   err_ = kernel_.setArg(1, cl_buffer_in_x_);
   err_ = kernel_.setArg(2, cl_buffer_in_y_);
   //Wait for the command queue_ to finish these commands before proceeding
   queue_.finish();
}

time_duration cl_floyd_warshall::run(std::vector<float>& mat) {
   ptime before;
   ptime after;
   time_duration total = seconds(0);
   if (mat.size() != total_size_)
      mat.resize(total_size_);
   cl::size_t<3> dst_origin;
   dst_origin.push_back(0);
   dst_origin.push_back(0);
   dst_origin.push_back(0);
   for (int i = 0; i < mdx_; ++i) {
      std::cout << "Compute generation [" << i + 1 << "/" << mdx_ << "]\r";
      std::cout.flush();
      before = microsec_clock::universal_time();
      // create the cl_buffer_in_x_ and cl_buffer_in_y_
      { // cl_buffer_x_
         cl::size_t<3> src_origin;
         cl::size_t<3> region;
         src_origin.push_back(i * sizeof(float));
         src_origin.push_back(0);
         src_origin.push_back(0);
         region.push_back(sizeof(float));
         region.push_back(mdx_);
         region.push_back(1);
         queue_.enqueueCopyBufferRect(
               cl_buffer_mat_,
               cl_buffer_in_x_,
               src_origin,
               dst_origin,
               region,
               mdx_ * sizeof(float),
               0,
               sizeof(float),
               0,
               NULL,
               &event_);
      }
      { // cl_buffer_y_
         cl::size_t<3> src_origin;
         cl::size_t<3> region;
         src_origin.push_back(0);
```

```
            src_origin.push_back(i * sizeof(float));
            src_origin.push_back(0);
            region.push_back(mdy_ * sizeof(float));
            region.push_back(1);
            region.push_back(1);
            queue_.enqueueCopyBufferRect(
                  cl_buffer_mat_,
                  cl_buffer_in_y_,
                  src_origin,
                  dst_origin,
                  region,
                  mdy_,
                  0,
                  sizeof(float),
                  0,
                  NULL,
                  &event_);
         }
         queue_.finish();
         // make the computation
         err_ = queue_.enqueueNDRangeKernel(
               kernel_,
               cl::NullRange,
               cl::NDRange(mdx_, mdy_),
               cl::NullRange,
               NULL,
               &event_);
         queue_.finish();
         after = microsec_clock::universal_time();
         total += (after - before);
      }
      std::cout << std::endl;
      err_ = queue_.enqueueReadBuffer(
            cl_buffer_mat_,
            CL_TRUE,
            0,
            sizeof(float) * total_size_,
            &mat[0],
            NULL,
            &event_);
      queue_.finish();
      return total;
}
```

Cl_floyd_warshall.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef CL_floyd_warshall_HEADER_DEFINED
#define CL_floyd_warshall_HEADER_DEFINED

class cl_floyd_warshall {
private :
     cl::Buffer cl_buffer_mat_;
     cl::Buffer cl_buffer_in_x_;
   cl::Buffer cl_buffer_in_y_;
     size_t buffer_size_;
     size_t total_size_;
     cl_uint mdx_;
     cl_uint mdy_;
     unsigned int device_used_;
     std::vector<cl::Device> devices_;
     cl::Context context_;
     cl::CommandQueue queue_;
     cl::Program program_;
     cl::Kernel kernel_;
     // debugging variables
     cl_int err_;
     cl::Event event_;
public :
     cl_floyd_warshall(bool gpu);
     void init(const std::string& cl_file);
     // if change size
     void setup(const std::pair<unsigned int, unsigned int>& s);
     // then before the next iteration
     void prepare(const std::vector<float>& in);
     // run the actual next step
     boost::posix_time::time_duration run(std::vector<float>& mat);
};

#endif // CL_floyd_warshall_HEADER_DEFINED
```

Ewd_file.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <limits>
#include <stdexcept>
```

```
#include <sstream>
#include <math.h>
#include "ewd_file.h"

ewd_file::ewd_file() : nb_vector_(0), nb_edges_(0) {
   edges_.clear();
}

ewd_file::~ewd_file() {
   nb_vector_ = 0;
   nb_edges_ = 0;
   edges_.clear();
}

void ewd_file::import_file(const std::string& name) throw(std::exception)
{
   std::ifstream ifs;
   ifs.open(name.c_str());
   if (!ifs.is_open())
      throw std::runtime_error("Could not open file : " + name);
   ifs >> nb_vector_;
   if (nb_vector_ == 0)
      throw std::runtime_error("Parse error invalid vectors size");
   ifs >> nb_edges_;
   if (nb_edges_ == 0)
      throw std::runtime_error("Parse error invalid edges size");
   std::cout << "Graph size      : " << nb_vector_ << std::endl;
   for (int i = 0; i < nb_edges_; ++i) {
      std::cout << "Graph edge [" << (i + 1) << "/" << nb_edges_ << "]\r";
      std::cout.flush();
      std::stringstream ss("");
      ss << "Parse error at edge : " << i;
      unsigned int v1;
      unsigned int v2;
      float d;
      ifs >> v1;
      ifs >> v2;
      ifs >> d;
      if (v1 >= nb_vector_) {
         ss << " invalid first vector : " << v1;
         throw std::runtime_error(ss.str());
      }
      if (v2 >= nb_vector_) {
         ss << " invalid second vector : " << v2;
         throw std::runtime_error(ss.str());
      }
      if (d <= 0.0f) {
         ss << " invalid distance : " << d;
         throw std::runtime_error(ss.str());
      }
      edges_.insert(
            std::make_pair<std::pair<unsigned int , unsigned int>, float>(
               std::make_pair<unsigned int, unsigned int>(v1, v2),
               d));
   }
   std::cout << std::endl;
   ifs.close();
}

void ewd_file::export_file(const std::string& name) throw(std::exception)
{
   std::ofstream ofs;
   ofs.open(name.c_str());
   if (!ofs.is_open())
      throw std::runtime_error("Could not open file : " + name);
   this->operator<<(ofs);
   ofs.close();
}

float ewd_file::dist(unsigned int v1, unsigned int v2) const {
```

```
   if (v1 == v2)
      return 0.0f;
   std::map<std::pair<unsigned int, unsigned int>, float>::const_iterator ite;
   ite = edges_.find(std::make_pair<unsigned int, unsigned int>(v1, v2));
   if (ite == edges_.end())
      return huge_float;
   return ite->second;
}

size_t ewd_file::size() const {
   return nb_vector_;
}

void ewd_file::import_matrix(float* p, size_t size) throw(std::exception)
{
   nb_vector_ = sqrt(size);
   for (int x = 0; x < nb_vector_; ++x) {
      for (int y = 0; y < nb_vector_; ++y) {
         float distance = p[x + (y * nb_vector_)];
         if (distance < huge_float) {
            edges_.insert(
                  std::make_pair<std::pair<unsigned int , unsigned int>, float>(
                     std::make_pair<unsigned int, unsigned int>(x, y),
                     distance));
         }
      }
   }
}

void ewd_file::export_matrix(float* p, size_t size) throw(std::exception)
{
   if ((nb_vector_ * nb_vector_) != size)
      throw std::runtime_error("Unmatched size!");
   for (int x = 0; x < nb_vector_; ++x) {
      std::cout
         << "Export matrix line [" << x + 1 << "/" << nb_vector_ << "]\r";
      std::cout.flush();
      for (int y = 0; y < nb_vector_; ++y) {
         p[x + (y * nb_vector_)] = dist(x, y);
      }
   }
   std::cout << std::endl;
}

void ewd_file::print_matrix(std::ostream& os) {
   for (int x = 0; x < nb_vector_; ++x) {
      for (int y = 0; y < nb_vector_; ++y) {
         std::stringstream ss("");
         ss << dist(x, y);
         int line_left = 8 - ss.str().size();
         os << ss.str();
         for (int i = 0; i < line_left; ++i)
            os << " ";
      }
      std::cout << std::endl;
   }
}

std::ostream& ewd_file::operator<<(std::ostream& os) {
   os << nb_vector_ << std::endl;
   os << nb_edges_ << std::endl;
   std::map<std::pair<unsigned int, unsigned int>, float>::const_iterator ite;
   for (ite = edges_.begin(); ite != edges_.end(); ++ite) {
      os
         << ite->first.first << "\t"
         << ite->first.second << "\t"
         << ite->second << std::endl;
   }
   return os;
}
```

Ewd_file.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef EWD_FILE_HEADER_DEFINED
#define EWD_FILE_HEADER_DEFINED

#include <iostream>
#include <fstream>
#include <map>
#include <string>

const float huge_float = 1e30f;

class ewd_file {
protected :
     size_t nb_vector_;
     size_t nb_edges_;
     std::map<std::pair<unsigned int, unsigned int>, float> edges_;
public :
     ewd_file();
     virtual ~ewd_file();
     void import_file(const std::string& name) throw(std::exception);
     void export_file(const std::string& name) throw(std::exception);
     void import_matrix(float* p, size_t size) throw(std::exception);
     void export_matrix(float* p, size_t size) throw(std::exception);
     size_t size() const;
     float dist(unsigned int v1, unsigned int v2) const;
   void print_matrix(std::ostream& os);
     std::ostream& operator<<(std::ostream& os);
};

#endif
```

Floyd_warshall.cl

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
```

```
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

int pos2linear(
      const int2 position,
      const int2 m)
{
   return position.y * m.x + position.x;
}

// mat the matrix
// in_x column
// in_y line
__kernel void floyd_warshall_buffer(
      __global float* mat,
      __global float* in_x,
      __global float* in_y)
{
   const int2 d = (int2)(get_global_id(0), get_global_id(1));
   const int2 m = (int2)(get_global_size(0), get_global_size(1));

   int position = d.y * m.x + d.x
   float val1 = mat[position];
   float val2 = in_x[d.x] + in_y[d.y];
   mat[position] = (val1 < val2) ? val1 : val2;
}
```

Main.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <iostream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/program_options.hpp>
#include <boost/filesystem.hpp>

#include "cl_util.h"
#include "cl_floyd_warshall.h"
#include "ewd_file.h"

using namespace cl;
using namespace boost::program_options;
using namespace boost::posix_time;

int main(int ac, char** av) {
   bool enable_gpu = true;
   bool enable_image = false;
   unsigned int nb_loops = 1;
   std::string cl_file = "floyd_warshall.cl";
   std::string graph_in = "tinyEWD.txt";
   std::string graph_out = "";
   try {
      // parse command line
      options_description desc("Allowed options");
      desc.add_options()
         ("help,h", "produce help message")
         ("opencl-cpu,c", "compute using openCL on CPU")
         ("opencl-gpu,g", "compute using openCL on GPU (default)")
         ("file-in,f", value<std::string>(),
          "Input graph file in EWD format")
         ("file-out,o", value<std::string>(),
          "Output graph file in EWD format")
         ("loop,l", value<unsigned int>(),
          "number of loops (only in no-window mode)")
         ;
      variables_map vm;
      store(command_line_parser(ac, av).options(desc).run(), vm);
      if (vm.count("help")) {
         std::cout << desc << std::endl;
         return 1;
      }
      if (vm.count("opencl-cpu")) {
         enable_gpu = false;
         std::cout << "OpenCL (CPU)    : enable" << std::endl;
      }
      if (vm.count("opencl-gpu")) {
         enable_gpu = true;
         std::cout << "OpenCL (GPU)    : enable" << std::endl;
      }
      if (vm.count("file-in")) {
         graph_in = vm["file-in"].as<std::string>();
      }
      std::cout << "Graph file (in) : " << graph_in << std::endl;
      if (vm.count("file-out")) {
         graph_out = vm["file-out"].as<std::string>();
         std::cout << "Graph file (out): " << graph_out << std::endl;
      }
      if (vm.count("loop")) {
         nb_loops = vm["loop"].as<unsigned int>();
      }
      std::cout << "Loop            : " << nb_loops << std::endl;
      {
         // read the file with the data
         ewd_file ef;
```

```
            ef.import_file(graph_in);
            std::vector<float> initial_state(ef.size() * ef.size());
            ef.export_matrix(&initial_state[0], initial_state.size());
            if (initial_state.size() <= 256) ef.print_matrix(std::cout);
            // do the actual job
            cl_floyd_warshall cfw(enable_gpu);
            cfw.init("floyd_warshall.cl");
            std::cout
               << "Setup           : (" << ef.size() << ", "
               << ef.size() << ")" << std::endl;
            cfw.setup(std::make_pair<unsigned int, unsigned int>(
                     ef.size(),
                     ef.size()));
            time_duration best_time = minutes(60);
            for (int i = 0; i < nb_loops; ++i) {
               time_duration actual_time;
                  std::cout
                     << "Prepare (buffer): " << initial_state.size() << std::endl;
                  cfw.prepare(initial_state);
                  std::cout
                     << "Run (buffer)    : " << initial_state.size() << std::endl;
                  actual_time = cfw.run(initial_state);
               if (actual_time < best_time) best_time = actual_time;
               std::cout
                  << "\riteration [" << i + 1 << "/" << nb_loops << "] : "
                  << "actual_time : " << actual_time;
               std::cout.flush();
            }
            std::cout << std::endl
               << "Finished : [" << nb_loops << "] Best time : " << best_time
               << std::endl;
            if (graph_out.size() || initial_state.size() <= 256)
               ef.import_matrix(&initial_state[0], initial_state.size());
            // export to file out (in case exist)
            if (graph_out.size()) {
               ef.export_file(graph_out);
            }
            if (initial_state.size() <= 256) ef.print_matrix(std::cout);
         }
         // error handling
      } catch (cl::Error er) {
         std::cerr
            << "exception (CL)  : " << er.what()
            << "(" << er << ")" << std::endl;
         return -2;
      } catch (std::exception& ex) {
         std::cerr << "exception (std) : " << ex.what() << std::endl;
         return -1;
      }
      return 0;
   }
```

FFT

```
    ├── fft
    │   ├── Makefile
    │   ├── cl_fft.cpp
    │   ├── cl_fft.h
    │   ├── cl_util.cpp
    │   ├── cl_util.h
    │   ├── fft.cl
    │   ├── fftw_fft.h
    │   ├── generate_data.m
    │   ├── main.cpp
    │   └── txt_file.h
    │
```

Makefile

```
# simple makefile for fft

ifeq ($(OSTYPE), darwin)
CC = clang
CXX = clang++
CFLAGS = -g -O0 -I/opt/local/include -I/usr/local/include -I.. -DWITH_FFTW
LIBS = -L/opt/local/lib/ \
          -framework GLUT -framework Cocoa -framework OpenCL -framework OpenGL \
          -lboost_program_options -lboost_system \
          -lfftw3 -lfftw3f -lm
else
CC = gcc
CXX = g++
CFLAGS = -g -O0 -I/usr/include -I/usr/local/include -I.. -DWITH_FFTW
LIBS = -L/usr/lib64 -L/usr/local/lib \
          -lglut -lGL -lGLU -lOpenCL \
          -lboost_program_options -lboost_system \
          /usr/lib64/libfftw3f.so.3 -lm
endif
ALL=fft

all: $(ALL)

cl_util.o : cl_util.cpp cl_util.h
          $(CXX) -o cl_util.o -c cl_util.cpp $(CFLAGS)
cl_fft.o : cl_fft.cpp cl_fft.h
          $(CXX) -o cl_fft.o -c cl_fft.cpp $(CFLAGS)
main.o : main.cpp cl_fft.h cl_util.h fftw_fft.h txt_file.h
          $(CXX) -o main.o -c main.cpp $(CFLAGS)

fft : main.o cl_util.o cl_fft.o
          $(CXX) -o fft main.o cl_util.o cl_fft.o $(LIBS)

clean:
          rm -f $(ALL) *.o
```

Cl_fft.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
```

```
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <iostream>
#include <vector>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#ifdef __linux__
#include <GL/glx.h>
#endif
#include <boost/date_time/posix_time/posix_time.hpp>

#include "cl_fft.h"

using namespace boost::posix_time;

cl_fft::cl_fft(bool gpu) {
   //setup devices_ and context_
   std::vector<cl::Platform> platforms;
   err_ = cl::Platform::get(&platforms);
   device_used_ = 0;
   err_ = platforms[0].getDevices((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU,
&devices_);
   int t = devices_.front().getInfo<CL_DEVICE_TYPE>();
   try {
      throw cl::Error(0, "cheat pass");
#if defined (__APPLE__) || defined(MACOSX)
      CGLContextObj kCGLContext = CGLGetCurrentContext();
      CGLShareGroupObj kCGLShareGroup = CGLGetShareGroup(kCGLContext);
      cl_context_properties props[] =
      {
         CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
(cl_context_properties)kCGLShareGroup,
         0
      };
      context_ = cl::Context(props);
#else
#if defined WIN32 // Win32
      cl_context_properties props[] =
      {
         CL_GL_CONTEXT_KHR, (cl_context_properties)wglGetCurrentContext(),
         CL_WGL_HDC_KHR, (cl_context_properties)wglGetCurrentDC(),
         CL_CONTEXT_PLATFORM, (cl_context_properties)(platforms[0])(),
         0
      };
      context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
#else
      throw cl::Error(0, "cheat pass");
      cl_context_properties props[] =
      {
         CL_GL_CONTEXT_KHR, (cl_context_properties)glXGetCurrentContext(),
         CL_GLX_DISPLAY_KHR, (cl_context_properties)glXGetCurrentDisplay(),
```

```
        CL_CONTEXT_PLATFORM, (cl_context_properties)(platforms[0])(),
        0
      };
      context_ = cl::Context(CL_DEVICE_TYPE_GPU, props);
#endif
#endif
   } catch (cl::Error er) {
      std::cerr << "Warning         : could not attach GL and CL toghether!" <<
std::endl;
      cl_context_properties properties[] = { CL_CONTEXT_PLATFORM,
(cl_context_properties)(platforms[0])(), 0};
      context_ = cl::Context((gpu) ? CL_DEVICE_TYPE_GPU : CL_DEVICE_TYPE_CPU ,
properties);
      devices_ = context_.getInfo<CL_CONTEXT_DEVICES>();
   }
   queue_ = cl::CommandQueue(context_, devices_[device_used_], 0, &err_);
}

void cl_fft::init(const std::string& cl_file) {
   FILE* file = fopen(cl_file.c_str(), "rt");
   if (!file) throw std::runtime_error("could not open file " + cl_file);
   const unsigned int BUFFER_SIZE = 4096;
   size_t bytes_read = 0;
   std::string kernel__source = "";
   do {
      char temp[BUFFER_SIZE];
      memset(temp, 0, BUFFER_SIZE);
      bytes_read = fread(temp, sizeof(char), BUFFER_SIZE, file);
      kernel__source += temp;
   } while (bytes_read != 0);
   fclose(file);
   cl::Program::Sources source(
         1,
         std::make_pair(kernel__source.c_str(),
            kernel__source.size()));
   program_ = cl::Program(context_, source);
   try {
      err_ = program_.build(devices_);
   } catch (cl::Error er) {
      std::cerr << "build status    : "
         << program_.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(devices_[0]) << std::endl;
      std::cerr << "build options   : "
         << program_.getBuildInfo<CL_PROGRAM_BUILD_OPTIONS>(devices_[0]) << std::endl;
      std::cerr << "build log       : "  << std::endl
         << program_.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices_[0]) << std::endl;
      throw er;
   }
}

void cl_fft::prepare(const std::vector<cl_float2>& in) {
   data_size_ = in.size();
   kernel_ = cl::Kernel(program_, "fftRadix2Kernel", &err_);
   std::vector<cl_float2>::const_iterator ite;
   //initialize our CPU memory arrays, send them to the device and set the kernel_
arguements
   cl_buffer_in_x_ = cl::Buffer(
         context_,
         CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
         sizeof(cl_float2) * data_size_,
         (void*)&in[0],
         &err_);
   cl_buffer_out_y_ = cl::Buffer(
         context_,
         CL_MEM_READ_WRITE,
         sizeof(cl_float2) * data_size_,
         NULL,
         &err_);
   queue_.finish();
   //set the arguements of our kernel_
   err_ = kernel_.setArg(0, cl_buffer_in_x_);
```

```
   err_ = kernel_.setArg(1, cl_buffer_out_y_);
   //Wait for the command queue_ to finish these commands before proceeding
   queue_.finish();
}

time_duration cl_fft::run(std::vector<cl_float2>& out) {
   ptime before;
   ptime after;
   time_duration total = seconds(0);
   if (out.size() != data_size_)
      out.resize(data_size_);
   double p = log2(data_size_);
   for (int i = 1; i <= (data_size_ >> 1); i *= 2) {
      before = microsec_clock::universal_time();
      // enqueue the new parameter p
      err_ = kernel_.setArg(2, i);
      // make the computation
      err_ = queue_.enqueueNDRangeKernel(
            kernel_,
            cl::NullRange,
            cl::NDRange(data_size_ >> 1),
            cl::NullRange,
            NULL,
            &event_);
      queue_.finish();
      after = microsec_clock::universal_time();
      if (i != (data_size_ >> 1)) {
         err_ = queue_.enqueueCopyBuffer(
               cl_buffer_out_y_,
               cl_buffer_in_x_,
               0,
               0,
               data_size_ * sizeof(cl_float2),
               NULL,
               &event_);
         queue_.finish();
      }
      total += (after - before);
   }
   err_ = queue_.enqueueReadBuffer(
         cl_buffer_out_y_,
         CL_TRUE,
         0,
         data_size_ * sizeof(cl_float2),
         &out[0],
         NULL,
         &event_);
   queue_.finish();
   return total;
}
```

Cl_fft.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
```

```
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef CL_fft_HEADER_DEFINED
#define CL_fft_HEADER_DEFINED

class cl_fft {
private :
     cl::Buffer cl_buffer_in_x_;
   cl::Buffer cl_buffer_out_y_;
     size_t data_size_;
     unsigned int device_used_;
     std::vector<cl::Device> devices_;
     cl::Context context_;
     cl::CommandQueue queue_;
     cl::Program program_;
     cl::Kernel kernel_;
     // debugging variables
     cl_int err_;
     cl::Event event_;
public :
     cl_fft(bool gpu);
     void init(const std::string& cl_file);
     // then before the next iteration
     void prepare(const std::vector<cl_float2>& in);
//   void prepare(const std::vector<double>& in);
     // run the actual next step
     boost::posix_time::time_duration run(std::vector<cl_float2>& out);
//   boost::posix_time::time_duration run(std::vector<double>& out);
};

#endif // CL_fft_HEADER_DEFINED
```

Fft.cl

```
// from bealto.com free of use!

#define USE_MAD 1

#if CONFIG_USE_DOUBLE

#if defined(cl_khr_fp64)  // Khronos extension available?
#pragma OPENCL EXTENSION cl_khr_fp64 : enable
#elif defined(cl_amd_fp64)  // AMD extension available?
#pragma OPENCL EXTENSION cl_amd_fp64 : enable
#endif

// double
typedef double real_t;
typedef double2 real2_t;
#define FFT_PI 3.14159265358979323846
#define FFT_SQRT_1_2 0.70710678118654752440

#else

// float
typedef float real_t;
typedef float2 real2_t;
#define FFT_PI       3.14159265359f
#define FFT_SQRT_1_2 0.707106781187f
```

```
#endif

// Return A*B
real2_t mul(real2_t a, real2_t b)
{
#if USE_MAD
  return (real2_t)(mad(a.x, b.x, -a.y * b.y), mad(a.x, b.y, a.y * b.x)); // mad
#else
  return (real2_t)(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); // no mad
#endif
}

// Return A * exp(K*ALPHA*i)
real2_t twiddle(real2_t a, int k, real_t alpha)
{
  real_t cs,sn;
  sn = sincos((real_t)k * alpha, &cs);
  return mul(a, (real2_t)(cs, sn));
}

// In-place DFT-2, output is (a,b). Arguments must be variables.
#define DFT2(a,b) { real2_t tmp = a - b; a += b; b = tmp; }

// Compute T x DFT-2.
// T is the number of threads.
// N = 2*T is the size of input vectors.
// X[N], Y[N]
// P is the length of input sub-sequences: 1,2,4,...,T.
// Each DFT-2 has input (X[I],X[I+T]), I=0..T-1,
// and output Y[J],Y[J+P], J = I with one 0 bit inserted at postion P. */
__kernel void fftRadix2Kernel(__global const real2_t * x,__global real2_t * y,int p)
{
  int t = get_global_size(0); // thread count
  int i = get_global_id(0);   // thread index
  int k = i & (p - 1);            // index in input sequence, in 0..P-1
  int j = ((i - k) << 1) + k;     // output index
  real_t alpha = -FFT_PI * (real_t)k / (real_t)p;

  // Read and twiddle input
  x += i;
  real2_t u0 = x[0];
  real2_t u1 = twiddle(x[t], 1, alpha);

  // In-place DFT-2
  DFT2(u0, u1);

  // Write output
  y += j;
  y[0] = u0;
  y[p] = u1;
}
```

Fftw_fft.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef fftw_fft_HEADER_DEFINED
#define fftw_fft_HEADER_DEFINED

#include <complex>
#include <boost/date_time/posix_time/posix_time.hpp>
#ifdef WITH_FFTW
#include <fftw3.h>
#endif

using namespace boost::posix_time;

template <class T>
class fftw_fft {
   private :
      std::vector<std::complex<T> > fftw_buffer_in_;
      std::vector<std::complex<T> > fftw_buffer_out_;
      size_t data_size_;
   public :
      void prepare(const std::vector<std::complex<T> >& in);
      boost::posix_time::time_duration run(std::vector<std::complex<T> >& out);
};

template <>
void fftw_fft<float>::prepare(const std::vector<std::complex<float> >& in) {
   fftw_buffer_in_ = in;
   fftw_buffer_out_.resize(in.size());
}

template <>
boost::posix_time::time_duration fftw_fft<float>::run(
      std::vector<std::complex<float> >& out)
{
   ptime before;
   ptime after;
   fftwf_plan plan;
   before = microsec_clock::universal_time();
   plan = fftwf_plan_dft_1d(
         fftw_buffer_in_.size(),
         (fftwf_complex*)&fftw_buffer_in_[0],
         (fftwf_complex*)&fftw_buffer_out_[0],
         FFTW_FORWARD,
         FFTW_ESTIMATE);
   fftwf_execute(plan);
   after = microsec_clock::universal_time();
   fftwf_destroy_plan(plan);
   out = fftw_buffer_out_;
   return after - before;
}

#endif // fftw_fft_HEADER_DEFINED
```

Generate_data.m

```
% generate data for fft

clear;
```

```
for i = 10:20;
    val = zeros(2^i, 1);
    for j=1:(2^i);
        val(j) = sin(1 + j * pi / (2^(i - 8))) + 4 * sin(2.1 + j * pi / (2^(i - 6)));
    end;
    file_name = sprintf('input-%02d.txt', i);
    save(['~/Desktop/PA/fft/' file_name], 'val', '-ascii');
end;

system('ls -lah ~/Desktop/PA/fft/*.txt', '-echo');
```

Main.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <iostream>
#include <vector>
#include <complex>
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/program_options.hpp>
#include <boost/filesystem.hpp>

#include "cl_util.h"
#include "cl_fft.h"
#ifdef WITH_FFTW
#include "fftw_fft.h"
#endif

#include "txt_file.h"

using namespace cl;
using namespace boost::program_options;
using namespace boost::posix_time;

int main(int ac, char** av) {
   bool enable_gpu = true;
   bool enable_fftw = false;
   unsigned int nb_loops = 1;
   std::string cl_file = "fft.cl";
   std::string graph_in = "input.csv";
   std::string graph_out = "";
   try {
```

```
      // parse command line
      options_description desc("Allowed options");
      desc.add_options()
         ("help,h", "produce help message")
         ("opencl-cpu,c", "compute using openCL on CPU")
         ("opencl-gpu,g", "compute using openCL on GPU (default)")
#ifdef WITH_FFTW
         ("fftw,w", "compute using the FFTW3 library")
#endif // WITH_FFTW
         ("file-in,f", value<std::string>(),
          "Input file in TXT format")
         ("file-out,o", value<std::string>(),
          "Output file in TXT format")
         ("loop,l", value<unsigned int>(),
          "number of loops")
         ;
      variables_map vm;
      store(command_line_parser(ac, av).options(desc).run(), vm);
      if (vm.count("help")) {
         std::cout << desc << std::endl;
         return 1;
      }
      if (vm.count("opencl-cpu")) {
         enable_gpu = false;
         std::cout << "OpenCL (CPU)    : enable" << std::endl;
      }
      if (vm.count("opencl-gpu")) {
         enable_gpu = true;
         std::cout << "OpenCL (GPU)    : enable" << std::endl;
      }
      if (vm.count("file-in")) {
         graph_in = vm["file-in"].as<std::string>();
      }
      std::cout << "Graph file (in) : " << graph_in << std::endl;
      if (vm.count("file-out")) {
         graph_out = vm["file-out"].as<std::string>();
         std::cout << "Graph file (out): " << graph_out << std::endl;
      }
      if (vm.count("loop")) {
         nb_loops = vm["loop"].as<unsigned int>();
      }
#ifdef WITH_FFTW
      if (vm.count("fftw")) {
         enable_fftw = true;
      } else {
         enable_fftw = false;
      }
#endif // WITH_FFTW
      std::cout << "Loop            : " << nb_loops << std::endl;
      if (!enable_fftw) { // OpenCL
         // read the file with the data
         txt_file<cl_float2> ef;
         std::vector<cl_float2> out;
         ef.import_file(graph_in);
         // do the actual job
         cl_fft cfw(enable_gpu);
         cfw.init("fft.cl");
         std::cout
            << "Setup           : (" << ef.size() << ")" << std::endl;
         time_duration best_time = minutes(60);
         time_duration sum_time = seconds(0);
         for (int i = 0; i < nb_loops; ++i) {
            time_duration actual_time;
            cfw.prepare(ef);
            actual_time = cfw.run(out);
            if (actual_time < best_time) best_time = actual_time;
            sum_time += actual_time;
            std::cout
               << "\riteration [" << i + 1 << "/" << nb_loops << "] : "
               << "actual_time : " << actual_time;
```

```
            std::cout.flush();
         }
         std::cout << std::endl
            << "Finished        : [" << nb_loops << "]"
            << std::endl;
         std::cout << "Best time       : " << best_time << std::endl;
         std::cout << "Average time    : " << sum_time / nb_loops << std::endl;
         // export to file out (in case exist)
         if (graph_out.size()) {
            ef = out;
            ef.export_file(graph_out);
         }
         if (ef.size() <= 256) std::cout << ef;
      }
#ifdef WITH_FFTW
      else { // FFTW
         // read the fiel with the data
         txt_file<std::complex<float> > ef;
         std::vector<std::complex<float> > out;
         ef.import_file(graph_in);
         fftw_fft<float> cfw;
         std::cout
            << "Setup           : (" << ef.size() << ")" << std::endl;
         time_duration best_time = minutes(60);
         time_duration sum_time = seconds(0);
         for (int i = 0; i < nb_loops; ++i) {
            time_duration actual_time;
            cfw.prepare(ef);
            actual_time = cfw.run(out);
            if (actual_time < best_time) best_time = actual_time;
            sum_time += actual_time;
            std::cout
               << "\riteration [" << i + 1 << "/" << nb_loops << "] : "
               << "actual_time : " << actual_time;
            std::cout.flush();
         }
         std::cout << std::endl
            << "Finished        : [" << nb_loops << "]"
            << std::endl;
         std::cout << "Best time       : " << best_time << std::endl;
         std::cout << "Average time    : " << sum_time / nb_loops << std::endl;
         // export to file out (in case exist)
         if (graph_out.size()) {
            ef = out;
            ef.export_file(graph_out);
         }
         if (ef.size() <= 256) std::cout << ef;
      }
#endif // WITH_FFTW
      // error handling
   } catch (cl::Error er) {
      std::cerr
         << "exception (CL)  : " << er.what()
         << "(" << er << ")" << std::endl;
      return -2;
   } catch (std::exception& ex) {
      std::cerr << "exception (std) : " << ex.what() << std::endl;
      return -1;
   }
   return 0;
}
```

Txt_file.h

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
```

```
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef txt_file_HEADER_DEFINED
#define txt_file_HEADER_DEFINED

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

template <class T = cl_float2>
class txt_file : public std::vector<T> {
   public :

      txt_file& operator=(const std::vector<T>& in) {
         this->clear();
         this->assign(in.begin(), in.end());
         return (*this);
      }

      void import_file(const std::string& name) throw(std::exception) {
         std::ifstream ifs;
         ifs.open(name.c_str());
         if (!ifs.is_open())
            throw std::runtime_error("Could not open file : " + name);
         std::string line = "";
         while (std::getline(ifs, line)) {
            if (line.size() == 0) continue;
            std::stringstream ss(line);
            std::string item;
            while (std::getline(ss, item, ' ')) {
               if (item.size() == 0) continue;
               T value;
               value.x = atof(item.c_str());
               value.y = 0.0;
               this->push_back(value);
            }
         }
         ifs.close();
      }

      void export_file(const std::string& name) throw(std::exception) {
         std::ofstream ofs;
         ofs.open(name.c_str());
         if (!ofs.is_open())
            throw std::runtime_error("Could not open file : " + name);
         ofs << (*this);
         ofs.close();
      }
```

```
};

std::ostream& operator<<(std::ostream& os, const txt_file<cl_float2>& in) {
   std::vector<cl_float2>::const_iterator ite;
   for (ite = in.begin(); ite != in.end(); ++ite) {
      os << (*ite).x << std::endl;
   }
   return os;
}

std::ostream& operator<<(std::ostream& os, const txt_file<cl_double2>& in) {
   std::vector<cl_double2>::const_iterator ite;
   for (ite = in.begin(); ite != in.end(); ++ite) {
      os << (*ite).x << std::endl;
   }
   return os;
}

template <class T>
class txt_file<std::complex<T> > : public std::vector<std::complex<T> > {
   public :

      txt_file& operator=(const std::vector<std::complex<T> >& in) {
         this->clear();
         this->assign(in.begin(), in.end());
         return (*this);
      }


      std::ostream& operator<<(std::ostream& os) {
         typename std::vector<std::complex<T> >::const_iterator ite;
         for (ite = this->begin(); ite != this->end(); ++ite) {
            os << (*ite).real() << std::endl;
         }
         return os;
      }


      void import_file(const std::string& name) throw(std::exception) {
         std::ifstream ifs;
         ifs.open(name.c_str());
         if (!ifs.is_open())
            throw std::runtime_error("Could not open file : " + name);
         std::string line = "";
         while (std::getline(ifs, line)) {
            if (line.size() == 0) continue;
            std::stringstream ss(line);
            std::string item;
            while (std::getline(ss, item, ' ')) {
               if (item.size() == 0) continue;
               std::complex<T> value;
               value.real() = (T)atof(item.c_str());
               value.imag() = 0.0f;
               this->push_back(value);
            }
         }
         ifs.close();
      }

      void export_file(const std::string& name) throw(std::exception) {
         std::ofstream ofs;
         ofs.open(name.c_str());
         if (!ofs.is_open())
            throw std::runtime_error("Could not open file : " + name);
         this->operator<<(ofs);
         ofs.close();
      }
};

template <class T>
```

```
std::ostream& operator<<(std::ostream& os, const txt_file<std::complex<T> >& in) {
   typename std::vector<std::complex<T> >::const_iterator ite;
   for (ite = in.begin(); ite != in.end(); ++ite) {
      os << (*ite).real() << std::endl;
   }
   return os;
}

#endif
```

Test

```
└── test
    ├── Makefile
    └── test.cpp
```

Makefile

```
# simple makefile for a test program


ifeq ($(OSTYPE), darwin)
CC=clang
CXX=clang++
CFLAGS = -g -O0 -I/opt/local/include -I/opt/local/include/opencv2 -I..
LIBS= -L/opt/local/lib/ \
          -framework OpenCL \
          -lboost_program_options -lboost_system
else
CC=gcc
CXX=g++
CFLAGS = -g -O0 -I/opt/local/include -I..
LIBS= -L/usr/local/lib \
          -lOpenCL \
          -lboost_program_options -lboost_system
endif
ALL=test


all: $(ALL)


test.o : test.cpp
          $(CXX) -o test.o -c test.cpp $(CFLAGS)


test : test.o
          $(CXX) -o test test.o $(LIBS)


clean:
          rm -f $(ALL) *.o
```

Test.cpp

```
/*
 * Copyright (c) 2012, Frederic Dubouchet
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the CERN nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY Frederic Dubouchet ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Frederic DUBOUCHET BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#ifdef __APPLE__
     #include <OpenCL/cl.h>
#else
     #include <CL/cl.h>
#endif // __APPLE__
#include <string>
#include <sstream>
#include <iostream>

std::string GetDeviceInfoString(
     cl_device_id device_id,
     cl_device_info device_info)
{
     char string_temp[512];
     size_t ret_size = 0;
     memset(string_temp, 0, 512);
     cl_int err = clGetDeviceInfo(
             device_id,
             device_info,
             512,
             string_temp,
             &ret_size);
     return std::string(string_temp);
}

std::string GetPlatformInfoString(
     cl_platform_id platform_id,
     cl_platform_info platform_info)
{
     char string_temp[512];
     size_t ret_size = 0;
     memset(string_temp, 0, 512);
     cl_int err = clGetPlatformInfo(
             platform_id,
             platform_info,
```

```
                512,
                string_temp,
                &ret_size);
        return std::string(string_temp);
}

bool GetDeviceInfoBool(
        cl_device_id device_id,
        cl_device_info device_info)
{
        cl_bool ret_bool;
        size_t ret_size = 0;
        cl_int err = clGetDeviceInfo(
                device_id,
                device_info,
                sizeof(cl_bool),
                &ret_bool,
                &ret_size);
        return ((ret_bool) ? true : false);
}

unsigned int GetDeviceInfoUInt(
        cl_device_id device_id,
        cl_device_info device_info)
{
        cl_uint ret_uint;
        size_t ret_size = 0;
        cl_int err = clGetDeviceInfo(
                device_id,
                device_info,
                sizeof(cl_uint),
                &ret_uint,
                &ret_size);
        return (unsigned int)ret_uint;
}

unsigned long GetDeviceInfoULong(
        cl_device_id device_id,
        cl_device_info device_info)
{
        cl_ulong ret_ulong;
        size_t ret_size = 0;
        cl_int err = clGetDeviceInfo(
                device_id,
                device_info,
                sizeof(cl_ulong),
                &ret_ulong,
                &ret_size);
        return (unsigned long)ret_ulong;
}

size_t GetDeviceInfoSizeT(
        cl_device_id device_id,
        cl_device_info device_info)
{
        size_t ret_size1;
        size_t ret_size2;
        cl_int err = clGetDeviceInfo(
                device_id,
                device_info,
                sizeof(size_t),
                &ret_size1,
                &ret_size2);
        return (size_t)ret_size1;
}
/*
std::string GetDeviceInfoFPConfig(
        cl_device_id device_id,
        cl_device_info device_info)
{
```

```
	cl_device_fp_config device_fp_config;
	size_t ret_size = 0;
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_DOUBLE_FP_CONFIG,
		sizeof(cl_device_fp_config),
		&device_fp_config,
		&ret_size);
	std::string flags = "";
	if (device_fp_config & CL_FP_DENORM)
		flags += std::string("CL_FP_DENORM ");
	if (device_fp_config & CL_FP_INF_NAN)
		flags += std::string("CL_FP_INF_NAN ");
	if (device_fp_config & CL_FP_ROUND_TO_NEAREST)
		flags += std::string("CL_ROUND_TO_NEAREST ");
	if (device_fp_config & CL_FP_ROUND_TO_ZERO)
		flags += std::string("CL_ROUND_TO_ZERO ");
	if (device_fp_config & CL_FP_ROUND_TO_INF)
		flags += std::string("CL_ROUND_TO_INF ");
	if (device_fp_config & CL_FP_FMA)
		flags += std::string("CL_FP_FMA ");
	return flags;
}
*/
std::string GetDeviceInfoExecutionCapabilities(
	cl_device_id device_id)
{
	cl_device_exec_capabilities device_exec_capabilities;
	size_t ret_size = 0;
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_EXECUTION_CAPABILITIES,
		sizeof(cl_device_exec_capabilities),
		&device_exec_capabilities,
		&ret_size);
	std::string flags = "";
	if (device_exec_capabilities & CL_EXEC_KERNEL)
		flags += std::string("CL_EXEC_KERNEL");
	if (device_exec_capabilities & CL_EXEC_NATIVE_KERNEL)
		flags += std::string("CL_EXEC_NATIVE_KERNEL");
	return flags;
}

std::string GetDeviceInfoGlobalMemCacheType(
	cl_device_id device_id)
{
	cl_device_mem_cache_type device_mem_cache_type;
	size_t ret_size = 0;
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_GLOBAL_MEM_CACHE_TYPE,
		sizeof(cl_device_mem_cache_type),
		&device_mem_cache_type,
		&ret_size);
	switch (device_mem_cache_type) {
		case CL_NONE :
			return std::string("CL_NONE");
		case CL_READ_ONLY_CACHE :
			return std::string("CL_READ_ONLY_CACHE");
		case CL_READ_WRITE_CACHE :
			return std::string("CL_READ_WRITE_CACHE");
		default :
			return std::string("default");
	}
}

std::string GetDeviceInfoLocalMemType(
	cl_device_id device_id)
{
	cl_device_local_mem_type device_local_mem_type;
```

```
	size_t ret_size = 0;
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_GLOBAL_MEM_CACHE_TYPE,
		sizeof(cl_device_local_mem_type),
		&device_local_mem_type,
		&ret_size);
	switch (device_local_mem_type) {
		case CL_LOCAL :
			return std::string("CL_LOCAL");
		case CL_GLOBAL :
			return std::string("CL_GLOBAL");
		default :
			return std::string("default");
	}
}

std::string GetDeviceInfoMaxWorkItemSizes(
	cl_device_id device_id)
{
	cl_uint dim = GetDeviceInfoUInt(
		device_id,
		CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS);
	size_t ret_size = 0;
	size_t* p = new size_t[dim];
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_MAX_WORK_ITEM_SIZES,
		sizeof(size_t) * dim,
		p,
		&ret_size);
	std::stringstream ss("");
	ss << "(";
	for (int i = 0; i < dim; ++i) {
		if (i != 0) ss << ", ";
		ss << p[i];
	}
	ss << ")";
	return ss.str();
}

cl_platform_id GetDeviceInfoPlatformID(
	cl_device_id device_id)
{
	cl_platform_id platform_id;
	size_t ret_size = 0;
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_PLATFORM,
		sizeof(cl_platform_id),
		&platform_id,
		&ret_size);
	return platform_id;
}

std::string GetDeviceInfoQueueProperties(
	cl_device_id device_id)
{
	cl_command_queue_properties command_queue_properties;
	size_t ret_size = 0;
	cl_int err = clGetDeviceInfo(
		device_id,
		CL_DEVICE_QUEUE_PROPERTIES,
		sizeof(cl_command_queue_properties),
		&command_queue_properties,
		&ret_size);
	switch (command_queue_properties) {
		case CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE :
			return std::string("CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE");
		case CL_QUEUE_PROFILING_ENABLE :
```

```
                return std::string("CL_QUEUE_PROFILING_ENABLE");
            default :
                return std::string("default?");
        }
}

std::string GetDeviceInfoSingleFPConfig(
        cl_device_id device_id)
{
        cl_device_fp_config device_fp_config;
        size_t ret_size = 0;
        cl_int err = clGetDeviceInfo(
                device_id,
                CL_DEVICE_SINGLE_FP_CONFIG,
                sizeof(cl_device_fp_config),
                &device_fp_config,
                &ret_size);
        std::string flags = "";
        if (device_fp_config & CL_FP_DENORM)
                flags += "CL_FP_DENORM ";
        if (device_fp_config & CL_FP_INF_NAN)
                flags += "CL_FP_INF_NAN ";
        if (device_fp_config & CL_FP_ROUND_TO_NEAREST)
                flags += "CL_FP_ROUND_TO_NEAREST ";
        if (device_fp_config & CL_FP_ROUND_TO_ZERO)
                flags += "CL_FP_ROUND_TO_ZERO ";
        if (device_fp_config & CL_FP_ROUND_TO_INF)
                flags += "CL_FP_ROUND_TO_INF ";
        if (device_fp_config & CL_FP_FMA)
                flags += "CL_FP_FMA ";
        if (device_fp_config & CL_FP_INF_NAN)
                flags += "CL_FP_INF_NAN ";
        return flags;
}

std::string GetDeviceInfoType(
        cl_device_id device_id)
{
        cl_device_type device_type;
        size_t ret_size = 0;
        cl_int err = clGetDeviceInfo(
                device_id,
                CL_DEVICE_TYPE,
                sizeof(cl_device_type),
                &device_type,
                &ret_size);
        switch (device_type) {
                case CL_DEVICE_TYPE_CPU :
                        return std::string("CL_DEVICE_TYPE_CPU");
                case CL_DEVICE_TYPE_GPU :
                        return std::string("CL_DEVICE_TYPE_GPU");
                case CL_DEVICE_TYPE_ACCELERATOR :
                        return std::string("CL_DEVICE_TYPE_ACCELERATOR");
                case CL_DEVICE_TYPE_DEFAULT :
                        return std::string("CL_DEVICE_TYPE_DEFAULT");
                default :
                        return std::string("default?");
        }
}

#define PRINT_DEVICE_INFO_STRING(id, info) { \
        std::cout << #info << " \t:\t" \
                << GetDeviceInfoString(id, info) << std::endl; \
}

#define PRINT_PLATFORM_INFO_STRING(id, info) { \
        std::cout << "\t" << #info << " \t:\t" \
                << GetPlatformInfoString(id, info) << std::endl; \
}
```

```
#define PRINT_DEVICE_INFO_BOOL(id, info) { \
     std::cout << #info << " \t:\t" \
             << ((GetDeviceInfoBool(id, info)) ? "true" : "false") \
             << std::endl; \
}

#define PRINT_DEVICE_INFO_UINT(id, info) { \
     std::cout << #info << " \t:\t" \
             << GetDeviceInfoUInt(id, info) << std::endl; \
}

#define PRINT_DEVICE_INFO_ULONG(id, info) { \
     std::cout << #info << " \t:\t" \
             << GetDeviceInfoULong(id, info) << std::endl; \
}

#define PRINT_DEVICE_INFO_SIZE_T(id, info) { \
     std::cout << #info << " \t:\t" \
             << GetDeviceInfoSizeT(id, info) << std::endl; \
}
/*
void PRINT_DEVICE_DOUBLE_FP_CONFIG(cl_device_id id) {
     std::cout << "CL_DEVICE_DOUBLE_FP_CONFIG" << " \t:\t"
             << GetDeviceInfoFPConfig(id, CL_DEVICE_DOUBLE_FP_CONFIG)
             << std::endl;
}
*/
void PRINT_DEVICE_EXECUTION_CAPABILITIES(cl_device_id id) {
     std::cout << "CL_DEVICE_ECECUTION_CAPABILITIES" << " \t:\t"
             << GetDeviceInfoExecutionCapabilities(id) << std::endl;
}

void PRINT_DEVICE_GLOBAL_MEM_CACHE_TYPE(cl_device_id id) {
     std::cout << "CL_DEVICE_GLOBAL_MEM_CACHE_TYPE" << " \t:\t"
             << GetDeviceInfoGlobalMemCacheType(id) << std::endl;
}
/*
void PRINT_DEVICE_HALF_FP_CONFIG(cl_device_id id) {
     std::cout << "CL_DEVICE_HALF_FP_CONFIG" << " \t:\t"
             << GetDeviceInfoFPConfig(id, CL_DEVICE_HALF_FP_CONFIG)
             << std::endl;
}
*/
void PRINT_DEVICE_LOCAL_MEM_TYPE(cl_device_id id) {
     std::cout << "CL_DEVICE_LOCAL_MEM_TYPE" << " \t:\t"
             << GetDeviceInfoLocalMemType(id) << std::endl;
}

void PRINT_DEVICE_MAX_WORK_ITEM_SIZES(cl_device_id id) {
     std::cout << "CL_DEVICE_MAX_WORK_ITEM_SIZES" << " \t:\t"
             << GetDeviceInfoMaxWorkItemSizes(id) << std::endl;
}

void PRINT_DEVICE_PLATFORM(cl_device_id id) {
     std::cout << "CL_DEVICE_PLATFORM" << std::endl;
     cl_platform_id platform_id = GetDeviceInfoPlatformID(id);
     PRINT_PLATFORM_INFO_STRING(platform_id, CL_PLATFORM_PROFILE);
     PRINT_PLATFORM_INFO_STRING(platform_id, CL_PLATFORM_VERSION);
     PRINT_PLATFORM_INFO_STRING(platform_id, CL_PLATFORM_NAME);
     PRINT_PLATFORM_INFO_STRING(platform_id, CL_PLATFORM_VENDOR);
     PRINT_PLATFORM_INFO_STRING(platform_id, CL_PLATFORM_EXTENSIONS);
}

void PRINT_DEVICE_QUEUE_PROPERTIES(cl_device_id id) {
     std::cout << "CL_DEVICE_QUEUE_PROPERTIES" << " \t:\t"
             << GetDeviceInfoQueueProperties(id) << std::endl;
}

void PRINT_DEVICE_SINGLE_FP_CONFIG(cl_device_id id) {
     std::cout << "CL_DEVICE_SINGLE_FP_CONFIG" << " \t:\t"
```

```
            << GetDeviceInfoSingleFPConfig(id) << std::endl;
}
void PRINT_DEVICE_TYPE(cl_device_id id) {
      std::cout << "CL_DEVICE_TYPE" << " \t:\t"
            << GetDeviceInfoType(id) << std::endl;
}

void print_device_info(cl_device_id device_id) {
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_ADDRESS_BITS);
      PRINT_DEVICE_INFO_BOOL(device_id, CL_DEVICE_AVAILABLE);
      PRINT_DEVICE_INFO_BOOL(device_id, CL_DEVICE_COMPILER_AVAILABLE);
//    PRINT_DEVICE_DOUBLE_FP_CONFIG(device_id);
      PRINT_DEVICE_INFO_BOOL(device_id, CL_DEVICE_ENDIAN_LITTLE);
      PRINT_DEVICE_INFO_BOOL(device_id, CL_DEVICE_ERROR_CORRECTION_SUPPORT);
      PRINT_DEVICE_EXECUTION_CAPABILITIES(device_id);
      PRINT_DEVICE_INFO_STRING(device_id, CL_DEVICE_EXTENSIONS);
      PRINT_DEVICE_INFO_ULONG(device_id, CL_DEVICE_GLOBAL_MEM_CACHE_SIZE);
      PRINT_DEVICE_GLOBAL_MEM_CACHE_TYPE(device_id);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE);
      PRINT_DEVICE_INFO_ULONG(device_id, CL_DEVICE_GLOBAL_MEM_SIZE);
//    PRINT_DEVICE_HALF_FP_CONFIG(device_id);
      PRINT_DEVICE_INFO_BOOL(device_id, CL_DEVICE_IMAGE_SUPPORT);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_IMAGE2D_MAX_HEIGHT);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_IMAGE2D_MAX_WIDTH);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_IMAGE3D_MAX_DEPTH);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_IMAGE3D_MAX_HEIGHT);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_IMAGE3D_MAX_WIDTH);
      PRINT_DEVICE_INFO_ULONG(device_id, CL_DEVICE_LOCAL_MEM_SIZE);
      PRINT_DEVICE_LOCAL_MEM_TYPE(device_id);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_CLOCK_FREQUENCY);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_COMPUTE_UNITS);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_CONSTANT_ARGS);
      PRINT_DEVICE_INFO_ULONG(device_id, CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE);
      PRINT_DEVICE_INFO_ULONG(device_id, CL_DEVICE_MAX_MEM_ALLOC_SIZE);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_MAX_PARAMETER_SIZE);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_READ_IMAGE_ARGS);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_SAMPLERS);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_MAX_WORK_GROUP_SIZE);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS);
      PRINT_DEVICE_MAX_WORK_ITEM_SIZES(device_id);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MAX_WRITE_IMAGE_ARGS);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MEM_BASE_ADDR_ALIGN);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE);
      PRINT_DEVICE_INFO_STRING(device_id, CL_DEVICE_NAME);
      PRINT_DEVICE_PLATFORM(device_id);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE);
      PRINT_DEVICE_INFO_STRING(device_id, CL_DEVICE_PROFILE);
      PRINT_DEVICE_INFO_SIZE_T(device_id, CL_DEVICE_PROFILING_TIMER_RESOLUTION);
      PRINT_DEVICE_QUEUE_PROPERTIES(device_id);
      PRINT_DEVICE_SINGLE_FP_CONFIG(device_id);
      PRINT_DEVICE_TYPE(device_id);
      PRINT_DEVICE_INFO_STRING(device_id, CL_DEVICE_VENDOR);
      PRINT_DEVICE_INFO_UINT(device_id, CL_DEVICE_VENDOR_ID);
      PRINT_DEVICE_INFO_STRING(device_id, CL_DEVICE_VERSION);
      PRINT_DEVICE_INFO_STRING(device_id, CL_DRIVER_VERSION);
}

int enum_platform(cl_platform_id platform_id) {

      cl_uint num_devices = 0;
      {
            // check the number of devices
            cl_int err = clGetDeviceIDs(
                  platform_id,
                  CL_DEVICE_TYPE_ALL,
```

```
                0,
                NULL,
                &num_devices);
        }
        {
                printf("found(%d) devices.\n", num_devices);
                // declare memory to hold the device id
                cl_device_id* device_ids = new cl_device_id[num_devices];

                // ask OpenCL for exactly 1 GPU
                cl_int err = clGetDeviceIDs(
                        platform_id,
                        CL_DEVICE_TYPE_ALL,
                        num_devices,
                        device_ids,
                        NULL);

                // make sure nothing went wrong
                if (err != CL_SUCCESS) {
                        printf("Error(%d) in creating device!\n", err);
                        switch (err) {
                                case CL_INVALID_PLATFORM :
                                        printf("\tCL_INVALID_PLATFORM\n");
                                        break;
                                case CL_INVALID_DEVICE_TYPE :
                                        printf("\tCL_INVALID_DEVICE_TYPE\n");
                                        break;
                                case CL_INVALID_VALUE :
                                        printf("\tCL_INVALID_VALUE\n");
                                        break;
                                case CL_DEVICE_NOT_FOUND :
                                        printf("\tCL_DEVICE_NOT_FOUND\n");
                                        break;
                        }
                        return EXIT_FAILURE;
                }
                for (int i = 0; i < num_devices; ++i)
                        print_device_info(device_ids[i]);
        }
        return EXIT_SUCCESS;
}

int main(int ac, char** av) {
        cl_uint num_platforms = 0;
        {
                cl_int err = clGetPlatformIDs(
                        0,
                        NULL,
                        &num_platforms);
                if (err != CL_SUCCESS) {
                        printf("error(%d) in enumerating platform!\n", err);
                        return EXIT_FAILURE;
                }
        }
        {
                printf("found(%d) platforms.\n", num_platforms);
                cl_platform_id* platform_ids = new cl_platform_id[num_platforms];
                cl_int err = clGetPlatformIDs(
                        num_platforms,
                        platform_ids,
                        NULL);
                for (int i = 0; i < num_platforms; ++i)
                        if (enum_platform(platform_ids[i]) != EXIT_SUCCESS)
                                return EXIT_FAILURE;
        }
        return EXIT_SUCCESS;
}
```