Senior Project: GPU Computing

John Kloosterman
john.kloosterman@gmail.com

May 2013

1 Introduction

Graphics Processing Units (GPUs) are data-parallel processors. Although designed for interactive 3D graphics, they can be used for general-purpose computational tasks, which is called General Purpose GPU (GPGPU) computing. GPUs have significant numerical computational power, and are able to perform computationally-heavy tasks more quickly than CPUs while using less power. Some common applications of GPGPU computing are video encoding, scientific computing, and physics simulation in computer games.

However, programming GPUs is a difficult process. The first GPGPU computing was done by writing custom graphics shaders, but once GPU vendors recognized the potential of GPGPU computing, they released frameworks like nVidia's CUDA and the vender-neutral OpenCL to facilitate writing GPGPU programs. However, OpenCL in particular is tedious and difficult to use. This difficulty also gets in the way of testing. As well, there are limitations that GPU architecture places on GPU programs, such as no recursion, no dynamic memory, no global variables, and no function pointers. As a result, many algorithms need substantial reformulations to be run efficiently on a GPU.

The objective of my senior project was to make programming for GPUs using OpenCL simpler for developers, and to explore how to write efficient programs for GPU architectures. The project fell into four segments:

1. Building a test computer system for development and benchmarking.
2. Writing a framework around OpenCL.
3. Designing applications and algorithms that took advantage of GPU architecture, including a raytracer, an artificial intellegence game player, and an economics simulation.
4. Writing a compiler tool to make memory management less complex in OpenCL programs.

Together, these components fulfill the objectives set foward in my proposal of making OpenCL easier to use for other developers and solving an open problem in GPU computing.

Contents

2 OpenCL Overview

OpenCL is an API for compiling and executing code on heterogeneous systems (systems with more than one architecture). The most common scenario is running code on a GPU, but OpenCL backends have been written for other types of accelerator cards, such as Intel's Xeon Phi, as well as devices like dynamically-generated FPGAs. nVidia has a similar API for running code on GPUs called CUDA, but CUDA only supports nVidia GPUs.

2.1 Host and Device Side

In OpenCL, there is a strict separation between the host side and the device side. The "host" must be a CPU; it is responsible for compiling kernels and scheduling data transfers and kernel exections on the device side. The "device" is usually a non-CPU, such as a GPU or FPGA. OpenCL "kernels" are the only code that can be run on the device side. Devices cannot schedule data transfers or new kernel executions. Host-side code can be written in any language the OpenCL host API has been ported to. Kernels for the device side must be written in OpenCL C, which is a subset of C99.

2.2 Threads and Workgroups

An OpenCL kernel describes what a single hardware thread will do. GPU architectures have hundreds of hardware threads executing in parallel, and threads must be scheduled in groups (typically mutiples of 64). This means that efficient algorithms for GPUs will use many hardware threads. This constrasts with the way threads are used on CPUs, where spawning a thread can be an expensive operation.

Threads are grouped into "workgroups", which are limited by the hardware to typically 256 or 1024 threads. OpenCL organizes the threads in a workgroup into up to 3 dimensions. Threads within a workgroup are able to synchronize with each other to avoid race conditions and nondeterministic behaviour, whereas threads in different workgroups have no way to do any synchronization. Therefore, programming for OpenCL typically requires splitting a problem into blocks of the appropriate size for processing by a single workgroup.

2.3 Memory Spaces

CPUs are able to automatically manage a memory hiearchy, because the small number of threads access relatively few new memory locations per cycle. On a data-parallel architecture like a GPU, the thousands of threads can pull in massive amounts of data per cycle, making caching impossible.

Therefore, the burden is shifted to the programmer to organize which data should be stored in which levels of the memory hiearchy.

In OpenCL, video RAM is called `__global` memory. There is a large amount of it (3GB on the Radeon 7970), but it requires several hundred cycles to access. Each workgroup can allocate an amount of much faster `__local` memory as a cache or scratch space. A thread can store data in thread-specific, fast registers that are labeled `__private` memory. In OpenCL, these are completely separate memory spaces; declarations of pointers have to include which memory space they are pointing into.

3 Test Computer

The first part of the project was building a system with GPUs capable of GPGPU computation, for development and testing work. The system was paid for out of Prof. Adams' NSF Grant TUES-2 #1225739, with the nVidia GPUs provided by nVidia.

3.1 Hardware

This system included four different devices that OpenCL supports: an Intel Core i7-3770K CPU, the integrated Intel HD Graphics 4000 GPU, an AMD Radeon 7970 GPU, and two nVidia GTX 480 GPUs.

The theoretical performance of the Radeon 7970 is much higher than all the other devices combined. However, it also had a small maximum workgroup size of 256, whereas the nVidia GPUs supported workgroup sizes of up to 1024.

This system, running 64-bit Ubuntu 12.10, was used for all benchmarks in this report. The GPU used for benchmarks is the AMD Radeon 7970.

3.2 Software Setup

On Linux, it is currently difficult to have multiple OpenCL platforms installed at the same time. GPU platforms will only work if the X.org driver for that GPU is currently being used, which meant that I could not use the nVidia GPUs when the AMD GPU was being used to drive the display. As well, Ubuntu would not boot with the nVidia graphics drivers installed when the AMD graphics card was installed in the system. On top of this, the nVidia and Intel platforms inmplemented OpenCL 1.1 whereas the AMD platform implemented OpenCL 1.2, and the headers are incompatible between versions even when only OpenCL 1.1 features are being used. For this reason, I developed exclusively with the AMD APP SDK 2.8 on Linux.

The situation is far easier on Windows, where both AMD and nVidia GPUs were always accessible with OpenCL. The Intel integrated GPU was only available if the system was booted with a display connected to it. AMD has the most robust set of tools for profiling and debugging OpenCL, but these tools only work on Windows in Visual Studio. Therefore, my workflow was to develop on Linux, but ensure that the code was portable to Windows, should I need to test on more GPUs and work with the profiling tools available there. Windows portability largely consisted of not relying on C++11 features.

3.3 OpenCL Quirks

As of the AMD APP SDK 2.8, for AMD GPUs `printf()` only reliably works when the thread in a workgroup with local ID (0,0,0) calls it. Sometimes, calls from other threads will make it. As well, `printf()` often does not work even then if it is inside any kind of conditional structure.

4 OpenCL Framework

OpenCL is designed to be flexible, but this means that it is unwieldly for developers to use. The simplest OpenCL program that runs code on a GPU is on the order of 20 lines of boilerplate long. The framework is an attempt at making OpenCL kernel calls syntactially as similar as possible to calling a C++ function or method.

4.1 Functions vs. Kernels

OpenCL kernels, code that runs on the device side, are marked with the `__kernel` keyword in an OpenCL C source file. This framework wraps around OpenCL's native host side API by defining the CLKernel class, making it simpler to compile kernels, pass parameters to them, and set the local and global workgroup size of the kernel.

OpenCL does not support directly running functions that are not marked as kernels on the device. However, these functions need some way to be tested. The `CLFunction` class in the framework removes OpenCL's limitation. A kernel to call the function is automatically generated on the host at compile-time, and that kernel is inserted into the source file passed to OpenCL. `CLFunction` will run the function on one hardware thread on the device.

Some functions, particularly those that involve threads cooperating on a task with temporary data stored in `__local` memory, cannot be called by `CLFunction`. One idiom I developed when testing these kinds of functions is to write a shim kernel that copies data into the correct memory space, calls the function to be tested, then copies the results back to `__global` memory. This idiom uses `CLKernel` insead of `CLFunction`.
(See `societies/util/test/max_min_tester.cl.in` for an example.)

4.2 Usage

C++11 constructs allow a class to syntactically behave like a variadic function, by defining an overloaded () operator using a variadic template. At this time, compilers only partically support the features needed to make using variadic templates elegant. With C++11, a `CLKernel` or `CLFunction` can be called like this:

```
#include <CLKernel.h>
#include <string>

std::string src; // some kernel source code
cl_int i, j, k;
CLKernel theKernel( "kernel_name", src );
theKernel( i, j, k );
```

Microsoft Visual Studio 2008 (the version that AMD's OpenCL tools currently target) does not support C++11. This requires a clunkier syntax:

```
#include <CLKernel.h>
#include <vector>
#include <string>

std::string src; // some kernel source code
cl_int i, j, k;
CLKernel theKernel( "kernel_name", src );

std::vector<CLArgument> arguments;
arguments.push_back( i );
arguments.push_back( j );
arguments.push_back( k );

theKernel( arguments );
```

The CLArgument class has constructors for many different types, which means that variables of those types can be passed into a CLKernel or a CLFunction without needing to explicitly create a CLArgument.

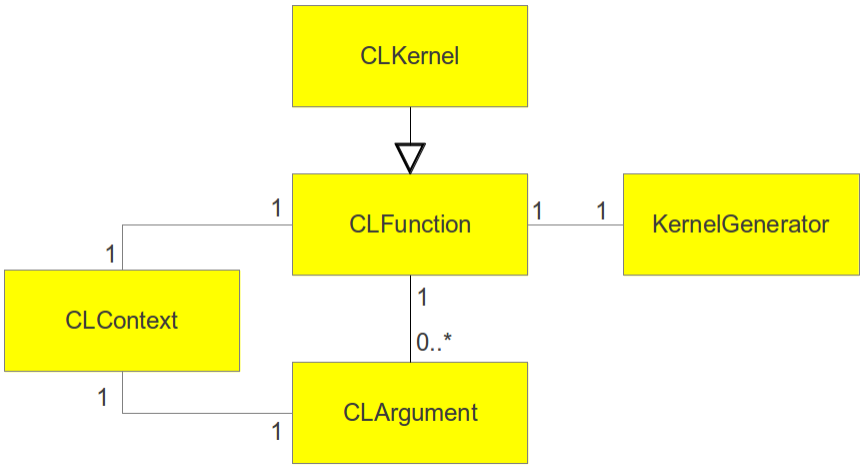

Figure 1: Framework class diagram

4.3 Other Features

I found I was often developing on my laptop, which does not have an OpenCL-supported GPU. The framework automatically falls back to using a CPU if there are no GPUs, so that programs will still run.

If the `CL_DEBUG` environment variable is set to 1, the framework will compile kernels with debugging symbols and run them on the CPU. This allows for debugging kernels using gdb as described in the AMD OpenCL programming guide[2].

4.4 Memory Copies

One of the main bottlenecks for GPU computing is the time to transfer data from RAM to the GPU, so the framework needs to avoid making any unnecessary copies. By default, all buffers are copied to and from the GPU every time a kernel is scheduled. This produces expected behaviour at the expense of being slow. The constructor for a `CLArgument` allows developers to specify whether the host memory backing the buffer should be copied to or from the GPU.

4.5 Lazy Initialization of Buffers

`CLArgument` is a wrapper around the OpenCL C++ API's `cl::Buffer` object. `cl::Buffer` objects are tied to an OpenCL context. However, `CLArguments` are not, because they need to be able to be instantiated implicitly by the C++ compiler. The required OpenCL context is passed later to the `CLArgument` by a `CLFunction` or `CLKernel` when the `cl::Buffer` is needed for the first time.

This causes undesired behaviour if a copy of an original `CLArgument` is what initializes the `cl:Buffer`, because the reference to the `cl::Buffer` does not propagate back to the original. C++ creates copies of `CLArguments` implicitly when they are used as arguments to `CLKernels`

or `CLFunction`s. (These arguments are not defined as references so that temporaries can be used as arguments.) The result is that `CLArgument`s cannot point to data stored on the device between function calls that is not copied back to the host. Code like this then breaks:

```
cl_float array[1024];
CLArgument arrayArg( "float", array, 1024, false, false );
someKernel( arrayArg ); // stores data into arrayArg, doesn't copy
                        // data back to host.
someOtherKernel( arrayArg ); // reads data someKernel() stored.
```

The `CLArgument::makePersistent()` method solves this problem by giving developers a way to create the `cl::Buffer` before any copies of the `CLArgument` are made. For an example of where this is necessary, see `OpenCLPlayer::makeMove()` in `kalah/opencl_player.cpp`.

4.6 Examples

There is an example usage of this framework in the `cl_test/example` directory, attached as Appendix A. All the other components of my project used this library as well, which means they serve as more complicated usage examples.

5 Raytracer

As a simple application to run on top of my framework, I implemented an OpenCL raytracer for honours credit in CS 352 (Computer Graphics). To exploit parallelism, the raytracer maps one pixel onto one hardware thread on the OpenCL device, where hardware threads have no overhead to create. The objective was for the raytracer to support real-time user interaction.

5.1 Capabilities

The raytracer has two geometric primitives: spheres and planes. Geometry can have a solid colour or be reflective. There can be any number of geometric primitives.

The lighting model takes into account ambient and diffuse lighting. There can be any number of diffuse light sources.

5.2 Limitations

Because OpenCL does not support recursion, reflective surfaces do not behave as they do in other raytracers. Reflective surfaces shoot a ray off the reflective surface, and that ray takes the colour of the first object it hits, taking into account only ambient lighting (see Figure 2). Recursive raytracers are able to take into account other types of lighting from the reflected surface, and can simulate rays being reflected more than once. This is not possible with this OpenCL implementation, because it would involve a recursive call from the lighting function of the reflective object to the lighting function of the reflected object.

5.3 User Interface

The user interface was implemented in GTK+ (see figure 3), with the rendered image being displayed in a `GtkImage`. This is inefficient, as the image is rendered on the GPU, copied to the CPU, copied to the `GtkImage`, then pushed back to the GPU. An alternative that trades increased complexity for more performance would be taking advantage of OpenCL's OpenGL interoperability features to draw the image.

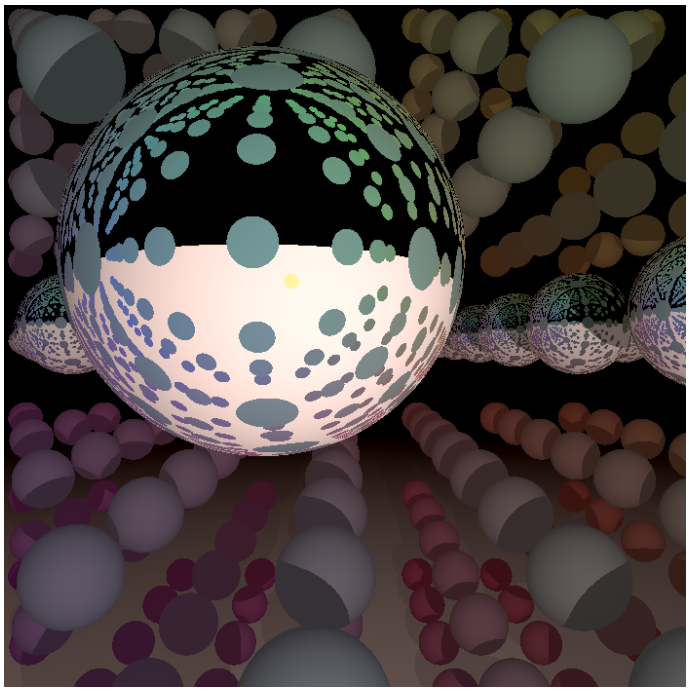

Figure 2: Reflections that only take into account ambient lighting

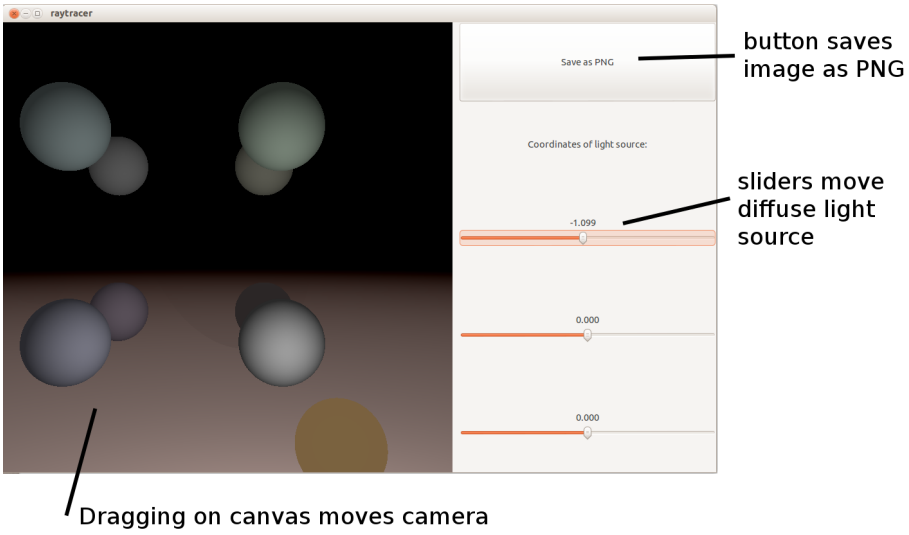

Figure 3: Raytracer user interface

5.4 Performance

The raytracer is able to render a 700x700 pixel test scene with 1000 spheres and a moveable diffuse light source at speeds that make it interactive (see figure 4). Using the CPU, this scene takes 1.28 seconds per frame (0.78 frames per second). Using the Radeon 7970, the scene takes 0.055 seconds per frame (18 frames per second). If the number of spheres is reduced to 216, the Radeon 7970 can render the scene at 60 frames per second.

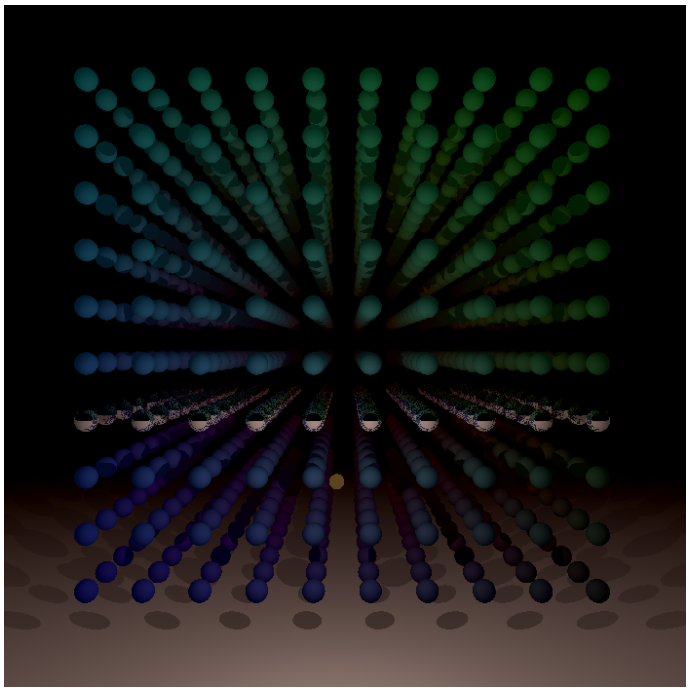

Figure 4: Test scene, featuring 1000 spheres and ground plane

5.5 Discarding Frames

Moving the light source or camera generates GTK+ events, which are processed in callback functions like `motion_notify()`. These callbacks call `run_kernel()`, which renders the frame on the GPU. This blocks the main program thread, but mouse drag events on the image still get queued. The effect is that the program can get more and more behind when rendering frames if events that render frames are added to the queue faster than frames can be rendered. This makes the raytracer feel unresponsive.

To prevent this from happening, after rendering a frame, the callbacks discard all the intermediate events on the GTK+ event queue that could cause a new frame to be rendered. This is done by having an `in_handler` flag that is set when a callback that is rendering a frame is running. All callbacks that can render a frame check to see if `in_handler` is set, and if so, return without rendering the frame. Immediately after rendering a frame, a callback runs this loop:

```
while ( gtk_events_pending() )
  gtk_main_iteration();
```

This causes all the queued GTK+ events to be processed; because the `in_handler` flag is set, any other callbacks in the queue do not render a frame. The end result is that all the events that could render a new frame are flushed from the queue.

6 Mankalah Minimax AI

This part of the project implemented a minimax player for the Mankalah game introduced in CS 212. Minimax is a much harder algorithm to implement on a GPU than a raytracer, because the minimax tree has dependencies between nodes, and the parallelism is less obvious.

6.1 Strategy

Following previous work on another minimax player implemented in CUDA[4], the minimax tree is broken up into layers. (See figure 5.) This implementation has 3. There are two layers of minimax computed sequentially on the CPU. The leaf nodes of the first layer serve as inputs for the second layer. The bottom-level leaf nodes of the second layer are copied over to the GPU, where 4 more levels of minimax are computed in parallel. Because the Mankalah minimax tree has a branching factor of 6, the overwhelming majority of the work is done in the bottom 4 levels of the tree.

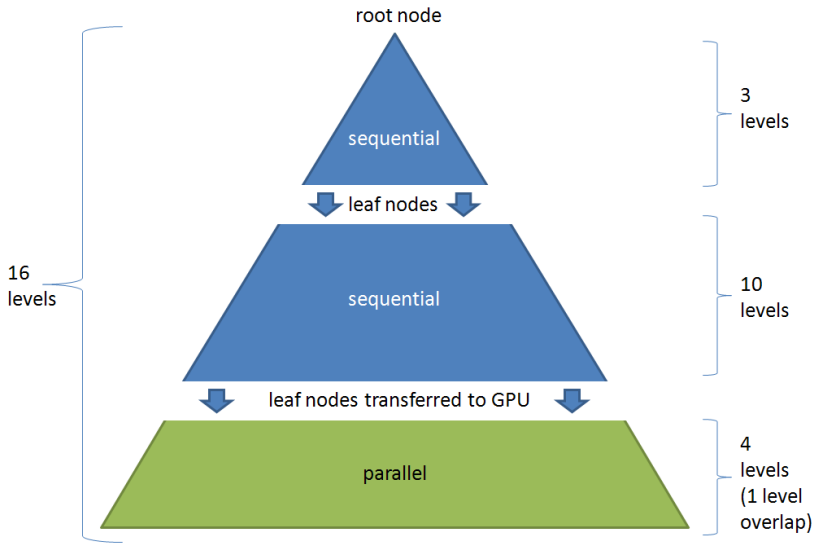

Figure 5: An example of how a minimax tree of depth 16 would be built out of the two sequential layers and one parallel layer of minimax.

The algorithm run on the GPU for evaluating the bottom layers of the minimax tree is as follows:

6.1.1 Generating boards

In this step, the game boards that represent the game tree for the next number of moves are generated from the start board. The tree is stored in `__local` memory in an array, where the child of the node stored at location n in the array is at location 6n + 1, as the minimax tree for Mankalah has a branching factor of 6. This can be computed in O(log(n)) time with b^d threads (see figure 6), by having the first thread generate the first node, 6 threads generate the first level of child nodes, 36 threads generate the second level, and so on.

6.1.2 Evaluating boards

In the standard minimax algorithm, the evaluate function needs to be computed only for boards that are leaf nodes of the game tree. Because all threads execute the same instruction counter, however, it is not slower to run the function on all the generated boards. This saves time in the next step, as should the game end at a stage before the bottom level of the minimax tree, the board is already evaluated.

6.1.3 Minimax

The minimax scores at a node can be computed in $O(log(n), b^d)$ time by using b^{d-1} threads to compute the minimax values for the nodes one level above the leaf nodes, b^{d-2} threads to compute the minimax values for the nodes 2 levels about the leaf nodes, and so on. After this process, the parent node for the game tree stores the minimax value for the tree.

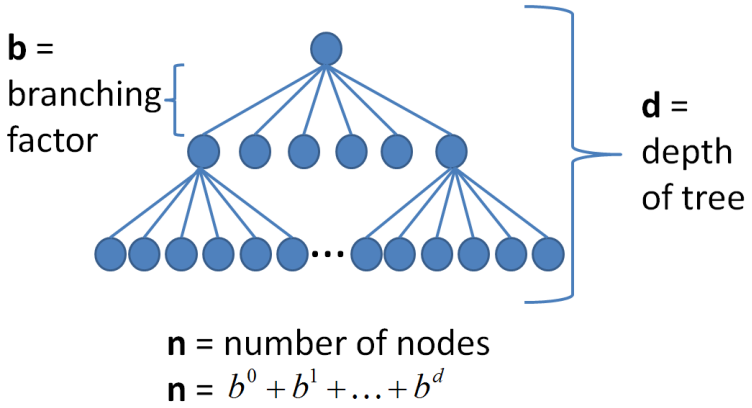

Figure 6: Variable letter meanings

6.1.4 Limitations

Because the boards are stored in local memory, and there is sequential dependency between the 3 steps of the algorithm, all threads for one of these minimax trees must be in the same workgroup. On the Radeon 7970, the maximum work group size is 256. Therefore, for Kalah, with a branching factor of 6, a minimax tree of 4 levels ($6^0 + 6^1 + 6^2 + 6^3 = 259$ nodes) was used, with a workgroup size of $6^3 = 216$ threads.

Since there were more nodes in the tree than threads in the workgroup, some threads had to evaluate 2 boards. This was judged to be a better design than using a tree of depth 3 (where no thread would have to evaluate 2 boards), because each workgroup could do 6 times as much work in the tree-based stages with one additional step, while the cost of running the evaluate function is relatively small. As well, on the Radeon 7970, threads are scheduled on the hardware in multiples of 64 (see the AMD documentation[2], pg. 5-21), so a tree of depth 4 has $256 - 216 = 40$ unused threads per workgroup (16%), whereas a tree of depth 3 has $64 - 43 = 21$ unused threads per workgroup (32%).

6.2 Increasing Search Depth

Because of memory bottlenecks, there is a limit to how many GPU minimax instances can be dispatched efficiently at one time. On the test system, 10 levels of sequential minimax was optimal for performance. As well, as Rocki and Suda[4] noted, the search tree has to be traversed twice at this level, once to determine what the leaf nodes are, and second to run the minimax reduction on the nodes after the parallel level has run.

Therefore, to gain more search depth, another level of sequential minimax was run above this first round of sequential minimax. Adding more levels at the topmost level of minimax increased the depth of the search tree but not the number of GPU instances dispatched at one time. Only one traversal of the search tree is necessary inside this layer.

6.3 Performance

6.3.1 Optimal Sequential Depth

One optimization question is for a minimax tree of a given depth how many levels shuld be computed in each of the 3 minimax stages. The GPU portion is fixed at 4 levels, so the distribution needs to be between the two sequential minimax stages. On the test system, the optimal allocation was to do 10 levels on the second sequential level before moving the boards to the GPU. This was determined by timing tests on different distributions of depths between the the first and second sequential levels.

6.3.2 Speedup

A recursive sequential algorithm was implemented for performance comparisons. As the table below demonstrates, the using the GPU yielded about a 10x speedup once the problem became sufficiently large. A 10x performance increase for a game with a branching factor of 6 means that the minimax player can look about one more move ahead in the game in the same amount of time. This table compares the wall-clock time to run a minimax search of a given depth using the sequential CPU implementation and the 3-level OpenCL parallel algorithm.

Depth	Sequential (s)	GPU OpenCL (s)	Speedup
4	0.000068	0.000919	0.07x
5	0.000343	0.000907	0.37x
6	0.001658	0.001054	1.57x
7	0.008301	0.001382	6.00x
8	0.040561	0.005487	7.39x
9	0.203477	0.021644	9.40x
10	0.980497	0.1048	9.35x
11	4.80143	0.438745	10.9x
12	23.4349	2.23596	10.5x
13	113.512	9.41622	12.1x

Since minimax is an embarrasingly parallel problem, a parallel CPU implementation should be able to achieve a speed of $\frac{sequential}{\#cores}$, or 8x on the test system. The GPU was able to best that.

The time it took to run the OpenCL parallel implementation on the CPU was also measured. It took several times longer to run than the sequential algorithm. This demonstrates how optimizing algorithms for CPUs and GPUs requires completely different assumptions, even if OpenCL is able to run the same code on both.

6.4 Testing

Along with testing of each of the GPU minimax algorithm's components, I used fuzz testing to ensure that the output of the GPU minimax algorithm was the same as the output of the sequential CPU minimax implementation. This was done by generating random valid boards then running the sequential and GPU OpenCL algorithms on them and comparing their output.

6.5 Sharing Code Between Host and Device

In order to avoid code duplication for the Mankalah board logic, the code in `board.c` and a few other files is used both on the host and device side. To consolidate all a kernel's dependencies into one file, I used the preprocessor to `#include` many `.c` files into one source file. I have tried consistently use the extension `.cl.in` for OpenCL source files that need to be preprocessed before they will compile.

For data structures passed between the host and device, a shared header file can be used. However, it is important to ensure that the layout of the structures are exactly the same. Notably, fundamental types like `int` on the device side are not guaranteed to be the same size as `int` on the host side. OpenCL provides types on the host prefaced with `cl_` that are guaranteed to be the same size as the type without the prefix on the client side (for an `int` on the device side, a `cl_int` is necessary on the host side). I used preprocessor macros to use the correct names for the types depending on where the shared code is compiled.

6.6 Tree Structures in OpenCL

Because there is no dynamic memory allocation in OpenCL, I have stored all trees in arrays. It is common to store binary trees in arrays, where the first child of the node at index n is at index $2n$, and the second at $2n + 1$. This can be generalized to trees of other branching factors. Support functions for this data structure can be found in `kalah/tree_array.c`.

7 Economics Simulation

As a larger, more complex problem, I worked on implementing a GPU version of an economics simulation used for research in Calvin's economics department.[3] The simulation already exists in Python, but it takes on the order of weeks to run, making it a good candidate for technology to speed it up. The simulation consists of a number of agents, which each hold a number of resources, that can harvest resources, invent machines to make harvesting resources more efficent, and trade resources and machines with each other.

This problem has promise for a speedup with a GPU's massive parallelism. Each of the agents in the simulation is largely independent, and most of the decisions that agents have to make need to evaluate the relative worth of their resources. Therefore, there is a natural mapping of one hardware thread to one resource, with each agent mapped to its own workgroup.

I did not complete the reimplementation of the simulation, but have met my objective of applying GPU computing to a complex, real-world problem. The Societies paper[3] breaks the simulation into 6 phases, of which the first 3 are non-trivial, the fourth is very complex, and the last 2 are trivial. I implemented the first 2 phases. This was enough to encounter difficult problems that required very different algorithms to efficiently solve on a GPU.

7.1 Phase 1: Resource Extraction

In this phase, agents harvest resources while there is time left in the "day". In each round, agents choose one of the resources that would be the most valuable for them to extract. Agents gain experience extracting resources, which reduces the amount of time needed to extract that resource.

The challenges implementing this phase were implementing the maximum and minimum algorithms, creating a mechanism for threads to create a variable-size array of options, and finding a random number generator suitable for OpenCL.

7.1.1 Array Maximum and Minimum

Several times in this simulation, I needed to find the maximum or minimum in an array without modifying the values in the array. The most efficient way to do this with at least 1 thread for every 2 elements is to build a max/min tree, since this allows the maximum or minimum to be found in O(log(n),n) time. Unfortunately, for an array with n elements, this requires a scratch array in scarce `__local` memory of $\frac{n}{2}$ elements.

The algorithm has two phases. In the first phase, $\frac{n}{2}$ threads each compare 2 elements of the original array, and put the largest in the scratch array. In the second phase, $\frac{n}{4}$ threads compare 2 elements of the scratch array, and put the maximum/minimum of the two values in the location of the first value. The second phase iterates, each time with half as many threads, until there is one value left, which is the maximum/minimum of the entire array.

The implementation of this algorithm is in `societies/util/max_min.cl`. This implementation also includes the ability to pass in a mask array, which allows the algorithm to ignore certain values in the array. This permits using the algorithm multiple times to find sets of arbitary numbers of maximum/minimum elements in a given array. (See the `max_n_indices()` function.)

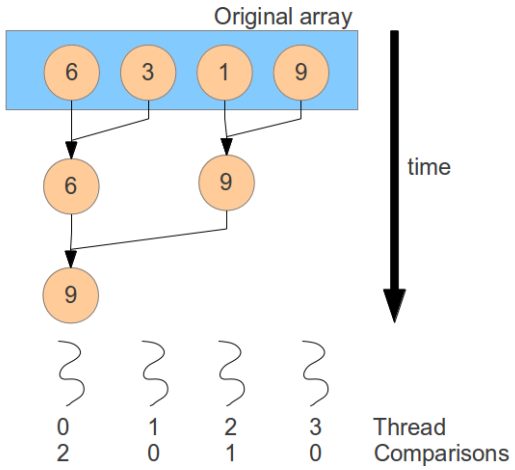

Figure 7: Thread to comparison mappings for maximum/minimum algorithm

7.1.2 Variable-length Arrays

There are cases where one thread needs to make a choice between different values on behalf of the workgroup. One implementation of this idiom can be found in `societies/util/choose_thread.cl`, where several threads can register their ability to be chosen, then one thread randomly makes a choice between them. Since not all threads want to be chosen, there needs to be a data structure that can hold a variable number of elements and that all threads can add elements too.

This data structure can be implemented in OpenCL with an atomic counter variable in `__local` memory initialized to 0, with an array in `__local` memory that is large enough for the maximum possible number of elements. When a thread wants to add an element, it calls the OpenCL-builtin `atomic_inc()` function to increment the counter, and puts a value in the array at the position that `atomic_inc()` returns, which is the previous value of the counter variable. After all threads have finished adding values to the array, the counter variable holds the number of items in the array.

7.1.3 Random Numbers

Many of the Societies algorithms required a source of random numbers. On a GPU, this is difficult because there is no hardware source of random numbers, and algorithmic pseudorandom number generators require a unique seed per workgroup so that each workgroup does not generate the same sequence of pseudorandom numbers. I made use of the MWC64X random number generator, which is ideal for GPUs because it requires very little state to be preserved across uses. I found an OpenCL implementation which was verified against statistical tests for randomness.[5]

7.2 Phase 2: Resource Trading

The agents are able to trade resources in pairs. The algorithms from the previous phase were sufficient for implementing this phase, except that there had to be an efficient way to randomly pair agents while ensuring that each agent was only in one pair. My solution was to generate a random permutation of the numbers $0 \cdots n-1$ using the "inside-out" Fisher-Yates shuffle, where n is the number of agents, on the CPU. This permutation is copied to the GPU, where one workgroup is assigned to each pair of agents.

7.3 Testing

One weakness of the Python Societies code is that it is not written in a way that makes it easy to test. I made sure to make my code very clean and wrote unit tests for all my functions, so that my code will be useful to the Societies project in the future. The algorithms I developed to find the maximum and minimum elements in an array I was also careful to test; this code should be useful as a reference for others doing similar work in OpenCL.

7.4 Kernel Size

The OpenCL kernels for Societies were relatively large. This is because `__local` memory cannot be preserved across kernel runs, so splitting up complex tasks into several kernels would require storing intermediate values in sloe `__global` memory. The Mankalah player has 3 stages that are independent except that the values in `__local` memory needed to be kept. This is a limitation of the hardware that leaks through as complexity for programmers.

My strategy was to test each of the segments of the kernels independently, so that it was still possible to debug one section of code at a time. However, certain kinds of bugs like race conditions due to missing barriers can only be tested in the context of the entire kernel.[1]

7.5 Configuration Values

There were several configuration values that are known only at run-time for the host, but are available at compilation time for the deveice-side kernels. Some of them, like the number of agents in the simulation, needed to be accessible as preprocessor macros. Another advantage of passing configuration values from the host to the device using preprocessor macros rather than inside a configuration object is that memory accesses can be avoided.

7.6 Partial Completion

As I was starting work on the third phase of the simulation, it was getting unwieldly to program the large and complex kernels. As well, the method by which agents traded higher-order devices in the simulation was complicated, only partially documented in the Societies paper, and the Python source code was obtuse. I found that managing buffers of `__local` memory were the largest source of complexity in the kernels, and thought addressing that problem would be ultimately more fruitful than finishing the increasingly complex parts of the Societies simulation.

[1] As an aside, CPUs are much more vulnerable to synchronization errors in OpenCL code than GPUs, because on a GPU, all hardware threads in a workgroup share an instruction pointer. In fact, the AMD OpenCL backend removes barriers that are made unnecessary by the GPU's architecture.

8 __local Memory malloc()

One problem I ran across when implementing the Societies code was that most functions required many parameters to pass around scratch `__local` memory that was needed by some algorithm several function calls deep. For instance, many of the Societies kernels needed to find maximums in arrays at some point. The algorithm that I used required scratch space in `__local` memory. This meant that I had to pass pointers to scratch buffers for the maximum algorithm as parameters to every function that could later use that algorithm.

This problem emerged because scratch buffers needed to be declared at the beginning of a kernel rather then when they were actually used. I had to keep track of exactly how large they needed to be every time they were used. Also, because `__local` memory usage restricts the number of hardware threads that can be concurrently running on the GPU, it is important to minimize the amount used, which means reusing the same buffers whenever possible. The result is that the complexity of managing buffers of `__local` memory became a bottleneck for my productivity.

Because the sizes of all these buffers are known at compile-time, and OpenCL does not support recursion or function pointers, relatively simple static analysis is enough to determine the maximum amount of `__local` memory a kernel needs. This makes it easy to shift the accounting burden from the programmer onto a tool.

As well, OpenCL does not support global variables, so any state used by a `malloc()`-like tool needs to be passed as a parameter to every function. This is inconvenient for the programmer, but should be easy for a tool to insert into a program.

8.1 Integration with Clang

The Clang C-family compiler [1] is written to be extensible. Clang's abstract syntax tree (AST) includes references to the text that every AST node is built from, which makes it easy for tools to analyze and rewrite source code. Clang's `Rewriter` class allows for sections of source code to be added, deleted, and moved around. Clang also is used in many OpenCL vendors' backends, which means it natively supports the OpenCL keywords like `__kernel` and adds these annotations to its AST.

This tool links against Clang, and creates an instance of an OpenCL compiler that does semantic analysis but stops before code generation. By using Clang's AST nodes and its `Rewriter` class, this tool offloads the heavy lifting of parsing and rewriting C source code to Clang.

Because of the tight integration with Clang, the tool is vulnerable to modifications to Clang's internals. The tool was developed against Clang 3.2 and LLVM 3.2 (available at `http://llvm.org/releases`).

8.2 API

This tool adds two new built-in functions to the OpenCL C language: `local_malloc()` and `local_free()`. The semantics of these functions is the same as `malloc()` and `free()` in C. `local_malloc()` allocates a buffer of `__local` memory, and `local_free()` deallocates a buffer.

Unlike usual `malloc()` and `free()`, the most recently freed buffer must be the most recently allocated buffer. As well, the amounts of memory allocated and freed have to be computable at compile-time by Clang; this allows expansions of preprocessor macros and simple arithmetic expressions as well as literals.

8.3 Phase 1: Computing Maximum Allocation

The tool registers hooks into Clang's AST processing methods, which are called as the AST is constructed. Whenever a function declaration is encountered, the tool generates a new node in a call

graph. When a function call is encountered, the tool creates an edge in the call graph. When a call to `local_malloc()` or `local_free()` is encountered, these are also added as actions to the call graph.

After all nodes in the AST have been visited, this call graph is guaranteed to have no cycles, because OpenCL does not support recursion. Therefore, in this call tree, the maximum amount of memory allocated by a function is the maximum amount allocated by any of a function's children plus any memory the function has allocated at the same time.

8.4 Phase 2: Program Rewriting

The tool then revisits all the nodes in the AST and rewrites function declarations and calls. To the kernel function, it inserts initialization code and allocates a `__local` memory buffer of the size computed in the previous phase. The tool adds a state object to every function call and declaration, so that that state object is accessible anywhere in the program.

8.4.1 Code Before Rewriting

```
#define SIZEOF_INT 4
#define NUM_THREADS 256

int function( int i )
{
    int ret;

    __local int *useless_ptr = local_malloc( SIZEOF_INT );
    *useless_ptr = i * 2;
    ret = *useless_ptr;
    local_free( SIZEOF_INT );

    return ret;
}

__kernel
void
function_sum(
    __global int *range_start,
    __global int *sum
)
{
    size_t num_threads = get_local_size( 0 );
    size_t local_id = get_local_id( 0 );
    __local int local_sum;

    // Initialize the sum.
    if ( local_id == 0 )
        local_sum = 0;
    barrier( CLK_LOCAL_MEM_FENCE );

    // Allocate scratch __local memory.
    __local int *values = local_malloc( NUM_THREADS * SIZEOF_INT );

    // Run the function at our thread's location.
    values[local_id] = function( *range_start + local_id );

    // Find the sum using atomics.
    atomic_add( &local_sum, values[local_id] );

    // Free scratch memory.
    local_free( NUM_THREADS * SIZEOF_INT );
```

```
    // Copy the result to global memory.
    if ( local_id == 0 )
        *sum = local_sum;
}
```

8.4.2 Code After Rewriting

```
/* Snip: embedded local_malloc implementation code */

#define SIZEOF_INT 4
#define NUM_THREADS 256

int function( int i, LocalMallocState *__local_malloc_state )
{
    int ret;

    __local int *useless_ptr = local_malloc( SIZEOF_INT, __local_malloc_state );
    *useless_ptr = i * 2;
    ret = *useless_ptr;
    local_free( SIZEOF_INT, __local_malloc_state );

    return ret;
}

__kernel
void
function_sum(
    __global int *range_start,
    __global int *sum
)
{
    __local char __local_malloc_buffer[1028];
    LocalMallocState __local_malloc_state_backing;
    LocalMallocState *__local_malloc_state = &__local_malloc_state_backing;
    local_malloc_init( __local_malloc_buffer, 1028, __local_malloc_state );

    size_t num_threads = get_local_size( 0 );
    size_t local_id = get_local_id( 0 );
    __local int local_sum;

    // Initialize the sum.
    if ( local_id == 0 )
        local_sum = 0;
    barrier( CLK_LOCAL_MEM_FENCE );

    // Allocate scratch __local memory.
    __local int *values = local_malloc( NUM_THREADS * SIZEOF_INT, __local_malloc_state
        );

    // Run the function at our thread's location.
    values[local_id] = function( *range_start + local_id, __local_malloc_state );

    // Find the sum using atomics.
    atomic_add( &local_sum, values[local_id] );

    // Free scratch memory.
    local_free( NUM_THREADS * SIZEOF_INT, __local_malloc_state );

    // Copy the result to global memory.
    if ( local_id == 0 )
        *sum = local_sum;
}
```

8.5 Including Runtime Code

The actual code for the `local_malloc()` and `local_free()` functions needs to be bundled with the rewritten code. One difficulty is that these functions have a different signature before and after rewriting (because the state object is added as a parameter). Also, OpenCL does not support including header files in its compiler. Therefore, the rewriter injects the needed run-time code at the beginning of the program, with the signatures of `local_malloc()` and `local_free()` as the programmer expects them defined as prototypes. These prototypes are visible only when a preprocessor macro is defined, which is defined for the first pass. This allows Clang to parse the code without errors. This code is left in after rewriting, but the preprocessor macro is no longer defined. This exposes the run-time signatures of `local_malloc()` and `local_free()` along with their code.

I wrote a small tool that allowed me to embed the `local_malloc()` support code into the executable of the tool, which means that the tool does not need to find a source code file at runtime.

8.6 Results

An example program using `local_malloc()` is in `local_malloc/example`. This example uses the rewriting tool to rewrite a kernel using `local_malloc()`, then runs it using the OpenCL framework. The overhead of using this tool to rewrite the program source code is only 2 lines.

9 Conclusions

OpenCL was difficult to work with, and the limitations it imposed made it difficult to do otherwise-simple tasks like find minimum values in an array. There is a long way to go before GPUs are accessible to non-specialist developers with OpenCL. OpenACC, which is built into a C/C++ compiler and uses `#pragma` directives similar to OpenMP, is a promising route. However, I do not think OpenACC would have reduced the complexity of writing larger applications like the Societies simulation. The framework I developed was successful at making OpenCL easier to use on the host side, but the OpenCL language created complexity on the device side. It is on the device side, rather than the host side, where the most work is needed, with tools along the lines of `local_malloc()` in this project.

The Mankalah player and Societies simulation were excellent exercises at designing data-parallel algorithms to solve problems that are not obviously data-parallel. Because of the performance and power benefits of data-parallel architectures, the kind of thinking needed to design these algorithms will only become more relevant. The Mankalah player was also a useful exercise at designing an application that made use of the strengths of all the parts of a heterogeneous architecture.

10 Future Work

Some kind of standard library for OpenCL kernels that could perform tasks like sorting arrays in `__local` memory efficiently, generating pseudorandom numbers, or finding the minimum n values in an array would make complex kernels much easier to write. In general, OpenCL's API encourages writing small kernels that do one thing, but this conflicts with GPU architecture where it is faster to keep data in `__local` memory while a large kernel does a lot of processing work on it.

There is space for adding layers of abstraction to the OpenCL C language. The most complex kernels I wrote involved lots of coordination across threads, like what was needed for the maximum/minimum or minimax algorithms. It would be more natural to program at the workgroup level rather than at the thread level for these kernels.

References

[1] clang: a C language family frontend for LLVM. http://clang.llvm.org.

[2] Advanced Micro Devices. AMD Accelerated Parallel Processing Programming Guide. http://developer.amd.com/download/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf, July 2012.

[3] Anthony J. Ditta, Loren Haarsma, and Rebecca Roselius Haney. Societies: A Model of a Complex Technological Evolving Economy. *Journal of Artificial Societies and Social Simulations*. Working paper submitted to journal.

[4] Kamil Rocki and Reji Suda. Parallel Minimax Tree Searching on GPU. In *PPAM 2009, Part I, LNCS 6067*, pages 449–456. Springer-Verlag Berlin Heidelberg, 2009.

[5] David B. Thomas. The MWC64X Random Number Generator. http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.html.

A Source Code Listings

README.markdown

```
This is the code for [John Kloosterman's](http://jkloosterman.net) senior project at [
    Calvin College](http://cs.calvin.edu).

* cl_test: A C++ framework around OpenCL that removes some of OpenCL's tedium and makes
     it possible to call non-kernels directly.
* raytracer: An OpenCL raytracer with a GTK+ UI.
* kalah: An OpenCL implementation of the minimax AI algorithm for [Kalah](http://en.
    wikipedia.org/wiki/Kalah).
* societies: An OpenCL reimplementation of parts of the economics simulator [SugarPy](
    http://abs.calvin.edu/hg/sugarpy/).
* local_malloc: A Clang-based tool to rewrite OpenCL kernels to support a malloc()-like
     dynamic allocation of workgroup-local memory.
* reports: the LaTeX source and output for the final report about the project.

Building
--------
The top-level Makefile will build cl_test, kalah, and the raytracer.

There are references to the OpenCL library and include paths in most Makefiles. You
    will need to edit these if you are not using the AMD APP SDK on 64-bit Linux.

The Makefiles use Clang (clang++) as the default C++ compiler. If Clang is not
    installed, this will need to be changed to use GCC (g++).

Dependencies
------------
* cl_test: OpenCL 1.1 or 1.2
* raytracer: GTK+ 2, libpng 1.2.49
* local_malloc: Source and a build of LLVM and Clang version 3.2
```

Makefile

```
all:
    $(MAKE) -C cl_test
    $(MAKE) -C raytracer
    $(MAKE) -C kalah
```

```
clean:
    $(MAKE) -C cl_test clean
    $(MAKE) -C raytracer clean
    $(MAKE) -C kalah clean
```

A.1 cl_test

cl_test/CLArgument.cpp

```
/**
 * CLArgument encapsulates data passed to CLKernel
 *  and CLFunction.
 *
 * @author John Kloosterman
 * @date December 2012
 */

/*
 * The cl::Buffer object is initialized only right before
 *  it is needed. The reason: it is important to be able
 *  to initialize a CLArgument without passing in a
 *  CLContext so that type coersion can work (e.g. passing
 *  a cl_int to a CLFunction automatically creates the correct
 *  CLArgument). cl::Buffers need a cl::Context for initialization.
 *
 * However, this causes unexpected behaviour if there are copies of a
 *  CLArgument being made before the buffer is initialized.
 *  Each copy will then create their own cl::Buffer. This breaks code
 *  like this:
 *
 *   cl_float array[1024];
 *   CLArgument arrayArg( "float", array, 1024, false, false );
 *   someKernel( arrayArg ); // stores data into arrayArg, doesn't copy
 *                           // data back to host.
 *   someOtherKernel( arrayArg ); // reads data someKernel() stored.
 *
 * This is because C++ made a copy of arrayArg when its cl::Buffer was
 *  not initialized, so each copy created its own buffer and the cl::Buffer
 *  passed to someKernel() and someOtherKernel() was not the same one.
 * The solution is to call CLArgument::makePersistent() on the buffer
 *  first.
 */

#include "CLArgument.h"
#include <iostream>
#include <cassert>
using namespace std;

CLArgument::CLArgument(
    string name,
    size_t size,
    void *ptr,
    bool copy,
    bool isArray,
    bool copyTo,
    bool copyBack
    )
{
    initialize(
        name,
        size,
        ptr,
        copy,
        isArray,
```

```
        copyTo,
        copyBack
        );
}

void CLArgument::initialize(
    string name,
    size_t size,
    void *ptr,
    bool copy,
    bool isArray,
    bool copyTo,
    bool copyBack
    )
{
    mySize = size;
    myName = name;
    myCopy = copy;
    myIsArray = isArray;
    myCopyTo = copyTo;
    myCopyBack = copyBack;

    if ( copy )
        copy_data( size, ptr );
    else
        myPtr = ptr;

    myBufferInitialized = false;
}

void CLArgument::copy_data(
    size_t size,
    void *ptr
    )
{
    myPtr = malloc( size );
    memcpy( myPtr, ptr, size );
}

CLArgument::CLArgument( const CLArgument &other )
    : myBufferInitialized( other.myBufferInitialized ),
      mySize( other.mySize ),
      myName( other.myName ),
      myCopy( other.myCopy ),
      myIsArray( other.myIsArray ),
      myCopyTo( other.myCopyTo ),
      myCopyBack( other.myCopyBack )
{
    if ( other.myCopy )
    {
        copy_data( other.mySize, other.myPtr );
        myBufferInitialized = false;
    }
    else
    {
        myPtr = other.myPtr;
        myBuffer = other.myBuffer;
    }
}

CLArgument::~CLArgument()
{
    if ( myCopy )
    {
```

```
        free( myPtr );
    }
}

cl::Buffer *CLArgument::getBuffer( CLContext &context )
{
    if ( !myBufferInitialized )
    {
        myBuffer = cl::Buffer(
            context.getContext(),
            // If we don't supply memory, allocate some for us.
            ( myPtr == NULL ) ? CL_MEM_HOST_NO_ACCESS : CL_MEM_USE_HOST_PTR,
            mySize,
            myPtr
            );

        myBufferInitialized = true;
    }

    return &myBuffer;
}

std::string CLArgument::getType()
{
    return myName;
}

void CLArgument::copyToDevice( cl::CommandQueue &queue )
{
    assert( myBufferInitialized );

    if ( !myCopyTo )
        return;

    queue.enqueueWriteBuffer(
        myBuffer,
        CL_TRUE,
        0,
        mySize,
        myPtr
    );
}

/**
 * See note in CLArgument.h
 */
void CLArgument::makePersistent( CLContext &context )
{
    myBuffer = cl::Buffer(
        context.getContext(),
        // If we don't supply memory, allocate some for us.
        ( myPtr == NULL ) ? CL_MEM_ALLOC_HOST_PTR : CL_MEM_USE_HOST_PTR,
        mySize,
        myPtr
        );

    myBufferInitialized = true;
}

void CLArgument::copyFromDevice( cl::CommandQueue &queue )
{
    assert( myBufferInitialized );

    // If we own the memory, nobody else can read it anyways.
```

```
    if ( myCopy )
        return;
    if ( !myCopyBack )
        return;

    queue.enqueueReadBuffer(
        myBuffer,
        CL_TRUE,
        0,
        mySize,
        myPtr
    );
}

bool CLArgument::isArray()
{
    return myIsArray;
}
```

cl_test/CLArgument.h

```
/**
 * CLArgument encapsulates data to pass to
 *  a CLKernel or a CLFunction.
 *
 * There are lots of automatically generated
 *  constructors so that type coersion
 *  will allow for calls like:
 *
 *  cl_int i;
 *  cl_float j;
 *  CLKernel a_kernel( ... );
 *  a_kernel( i, j ); // this will only work in C++11
 *
 * @author John Kloosterman
 * @date December 2012
 */

#ifndef _CL_UNIT_ARGUMENT
#define _CL_UNIT_ARGUMENT

#include "CLIncludes.h"
#include "CLContext.h"
#include <string>

#include <iostream>

// Don't worry about ___3 types for the time being.
//  and half* types, which are also typedefed with something
//  else that makes it impossible for C++ to tell their types apart.

/**
 * Macro to define constructors for a OpenCL host type,
 *  its vector types, and arrays of both of those.
 */
#define CONSTRUCTORS( type )                    \
  CTR( cl_##type, type );                       \
  PTR_CTR( cl_##type, type );                   \
  CTR( cl_##type##2, type##2 );                 \
  PTR_CTR( cl_##type##2, type##2 );             \
  CTR( cl_##type##4, type##4 );                 \
  PTR_CTR( cl_##type##4, type##4 );             \
  CTR( cl_##type##8, type##8 );                 \
  PTR_CTR( cl_##type##8, type##8 );             \
```

```
  CTR( cl_##type##16, type##16 );               \
  PTR_CTR( cl_##type##16, type##16 );

#define CTR( host, kernel )                                 \
    CLArgument( host val )                                  \
    { initialize( #kernel, sizeof( host ), &val ); }

#define PTR_CTR( host, kernel )                                         \
    CLArgument( host *array, size_t elements, bool copyTo = true, bool copyBack = true
        ) \
    { initialize( #kernel, sizeof( host ) * elements, array, false, true, copyTo,
        copyBack ); }

class CLArgument
{
public:
    /**
     * @param name
     *  The name of the type in OpenCL code.
     * @param size
     *  How large the object or array is
     * @param ptr
     *  A pointer to the object or array
     * @param copy
     *  Whether the CLArgument should keep a private copy of
     *   the object.
     * @param isArray
     *  Whether the object is an array or not.
     * @param copyTo
     *  Whether to copy the contents of the buffer to the device
     *   when a kernel using it is queued
     * @param copyTo
     *  Whether to copy the contents of the buffer from the device
     *   after a kernel using it is done executing.
     */
    CLArgument(
        std::string name,
        size_t size,
        void *ptr,
        bool copy = true,
        bool isArray = false,
        bool copyTo = true,
        bool copyBack = true );
    CLArgument( const CLArgument &other );
    ~CLArgument();

    // Add constructors for the value types and
    //  arrays for all the OpenCL types.
    CONSTRUCTORS( int );
    CONSTRUCTORS( uint );
    CONSTRUCTORS( long );
    CONSTRUCTORS( ulong );
    CONSTRUCTORS( short );
    CONSTRUCTORS( ushort );
    CONSTRUCTORS( char );
    CONSTRUCTORS( uchar );
    CONSTRUCTORS( float );
    CONSTRUCTORS( double );

    /**
     * Value constructor for user-defined types. This
     *  value cannot be changed once created. For values
     *  that need to be changed, use the array constructor
     *  with an array of size 1.
```

```
     *
     * @param name
     *  The device-side type of the value.
     * @param value
     *  The value.
     */
    template<class T>
    CLArgument( std::string name, T value );

    /**
     * Array constructor for user-defined types. Arrays
     *  use the memory passed into them to back the OpenCL
     *  buffer, meaning that any changes copied back from
     *  the device modify that array.
     *
     * @param name
     *  The device-side type of the elements of the array.
     * @param array
     *  A pointer to the array.
     * @param elements
     *  The number of elements in the array
     * @param copyTo, copyFrom
     *  Whether to copy the contents of this buffer to/from
     *  the device.
     */
    template<class T>
    CLArgument( std::string name, T *array, size_t elements, bool copyTo = true, bool
        copyBack = true );

    /**
     * Call this if the same CLArgument will be passed
     *  to more than one kernel or to the same kernel twice,
     *  and you want to reuse the same OpenCL buffer.
     * For instance, this is needed if one kernel stores
     *  data in a buffer, the buffer is not copied back to the host,
     *  and another kernel reads from the same buffer.
     */
    void makePersistent( CLContext &context );

    cl::Buffer *getBuffer( CLContext &context );
    std::string getType();
    void copyToDevice( cl::CommandQueue &queue );
    void copyFromDevice( cl::CommandQueue &queue );
    bool isArray();

private:
    void copy_data( size_t size, void *ptr );
    void initialize(
        std::string name,
        size_t size,
        void *ptr,
        bool copy = true,
        bool isArray = false,
        bool copyTo = true,
        bool copyBack = true
        );

    bool myBufferInitialized;
    void *myPtr;
    size_t mySize;
    std::string myName;
    cl::Buffer myBuffer;
    bool myCopy;
    bool myIsArray;
```

```
    bool myCopyTo;
    bool myCopyBack;
};

template<class T>
CLArgument::CLArgument( std::string name, T value )
{
    initialize( name, sizeof( T ), &value );
}

template<class T>
CLArgument::CLArgument( std::string name, T *array, size_t elements, bool copyTo, bool
    copyBack )
{
    initialize( name, sizeof( T ) * elements, array, false, true, copyTo, copyBack );
}

#endif
```

cl_test/CLContext.cpp

```
/**
 * An abstraction of cl::Context.
 *
 * @author John Kloosterman
 * @date December 2012
 */

#include "CLContext.h"
#include <iostream>
#include <cstdlib>
using namespace std;

CLContext::CLContext()
{
    /// @todo: allow some way to change this default using
    ///  environment variables or the like?
    initialize( 0, 0 );
}

CLContext::CLContext( int platform, int device )
{
    initialize( platform, device );
}

/**
 * Intialize this CLContext with a given platform
 *  and device.
 *
 * @TODO: device is not actually selectable?
 *
 * @param platform, device
 *  The OpenCL platform and device numbers
 *  to use for this context.
 */
void CLContext::initialize( int platform, int device )
{
    char *cl_debug = getenv( "CL_DEBUG" );
    if ( cl_debug
         && strcmp( cl_debug , "1" ) == 0 )
    {
        cout << "CL_DEBUG set: using CPU and outputting compiler output." << endl;
        myDebug = true;
    }
```

```
    else
        myDebug = false;

    vector<cl::Platform> platforms;
    cl::Platform::get( &platforms );

    if ( myDebug )
        platforms[platform].getDevices( CL_DEVICE_TYPE_CPU, &myDevices );
    else
    {
        try {
            platforms[platform].getDevices( CL_DEVICE_TYPE_GPU, &myDevices );
        } catch ( cl::Error e ) {
            // If we got no GPU devices, get a CPU.
            cout << "No GPU, so using a CPU." << endl;
            platforms[platform].getDevices( CL_DEVICE_TYPE_CPU, &myDevices );
        }
    }

    myContext = cl::Context( myDevices, NULL, NULL, NULL );
    myCommandQueue = cl::CommandQueue( myContext, myDevices[device], 0 );
}

cl::Context &CLContext::getContext()
{
    return myContext;
}

cl::CommandQueue &CLContext::getCommandQueue()
{
    return myCommandQueue;
}

/**
 * Compile OpenCL source for this platform.
 *
 * @param src
 *  The source code to compile.
 * @param compiler_flage
 *  Flags to pass to the OpenCL compiler.
 *
 * @return
 *  A cl::Program
 */
cl::Program CLContext::buildProgram( string &src, string &compiler_flags ) const
{
    cl::Program::Sources sources(
        1,
        std::pair<const char *, int>( src.c_str(), (int) src.length() )
        );

    cl::Program program( myContext, sources );

    try {
        if ( myDebug )
        {
            program.build(
                myDevices,
                ( "-g -O0 " + compiler_flags ).c_str()
                );

            // In debug mode, show all status
            std::cout << "Build Status: "
                      << program.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(myDevices[0])
```

```
                    << std::endl;
            std::cout << "Build Options:\t"
                    << program.getBuildInfo<CL_PROGRAM_BUILD_OPTIONS>(myDevices[0])
                    << std::endl;
            std::cout << "Build Log:\t "
                    << program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(myDevices[0])
                    << std::endl;
        }
        else
        {
            program.build( myDevices, compiler_flags.c_str() );
        }
    }
    catch ( cl::Error err )
    {
        // If there was a compiler error, spit out the errors.
        std::cout << "Build Status: "
                  << program.getBuildInfo<CL_PROGRAM_BUILD_STATUS>(myDevices[0])
                  << std::endl;
        std::cout << "Build Options:\t"
                  << program.getBuildInfo<CL_PROGRAM_BUILD_OPTIONS>(myDevices[0])
                  << std::endl;
        std::cout << "Build Log:\t "
                  << program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(myDevices[0])
                  << std::endl;

        throw new exception();
    }

    return program;
}
```

cl_test/CLContext.h

```
/**
 * Abstraction of a cl::Context. CLFunctions and
 *  CLKernels can take custom contexts, if you want
 *  to run on a platform and device other than the default.
 *
 * @author John Kloosterman
 * @date December 2012
 */

#ifndef _CL_CONTEXT_H
#define _CL_CONTEXT_H

#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>

class CLContext
{
public:
    CLContext();
    CLContext( int platform, int device );
    cl::Context &getContext();
    cl::CommandQueue &getCommandQueue();
    cl::Program buildProgram( std::string &src, std::string &compiler_flags ) const;
private:
    void initialize( int platform, int device );
    cl::Context myContext;
    cl::CommandQueue myCommandQueue;
    std::vector<cl::Device> myDevices;
    bool myDebug;
};
```

```
#endif
```

cl_test/CLFunction.cpp

```
#include "CLFunction.h"

/*
 * These need to correspond to the calls
 *  to _GEN_CL_FUNCTION_RUN_P at the end
 *  of CLFunction.h.
 */
#define _GEN_CL_FUNCTION_RUN( host, kernel ) \
    template<>                               \
    host CLFunction<host>::run()             \
    {                                   \
        return run( #kernel );          \
    }

_GEN_CL_FUNCTION_RUN( cl_int, int )
_GEN_CL_FUNCTION_RUN( cl_float, float )
_GEN_CL_FUNCTION_RUN( cl_double, double )
_GEN_CL_FUNCTION_RUN( cl_float3, float3 )

/**
 * Special case for functions that return void.
 */
template<>
void CLFunction<void>::run()
{
    std::string src;
    std::string kernelFunction;

    generateKernelSource( "void", src, kernelFunction );

    copyBuffersToDevice();

    cl::Kernel kernel = generateKernel( src, kernelFunction );

    std::vector<int> globalDimensions;
    globalDimensions.push_back( 1 );
    std::vector<int> localDimensions;
    std::vector<int> globalOffset;
    enqueueKernel( kernel, globalDimensions, globalOffset, localDimensions );

    copyBuffersFromDevice();
}
```

cl_test/CLFunction.h

```
/**
 * CLFunction encapsulates a non-kernel OpenCL function
 *  and allows it to be called as if it were a kernel.
 * It does this by generating a kernel on the fly
 *  that calls that function.
 *
 * @author John Kloosterman
 * @date December 2012
 */

#ifndef _CL_FUNCTION_H
#define _CL_FUNCTION_H

#include "CLContext.h"
```

```
#include "CLArgument.h"
#include "KernelGenerator.h"
#include <iostream>
#include <cassert>

template<class T>
class CLFunction
{
public:
    /**
     * @param function
     *  The name of the function to call.
     * @param kernel
     *  Source code the function is in.
     * @param context
     *  The CLContext in which to run the function.
     */
    CLFunction( std::string function,
                std::string kernel,
                CLContext context = CLContext() )
        : myContext( context ),
          myFunction( function),
          myKernel( kernel ),
          kernelBuilt( false )
        {
        }
    virtual ~CLFunction() { }

#ifdef _CPP_11_
    /*
     * If using C++11, you can call a CLFunction like
     *  a normal function, since this variadic macro
     *  and operator() together allow code like this:
     *
     * CLFunction<int> someFunction( ... );
     * int i = someFunction();
     */

    template<class ...Arguments>
    void setArguments( Arguments... params )
        {
            // This has to be done with an array, instead of
            //  the more elegant creating the array with an
            //  initializer list, in order to be compatible with
            //  my compiler.
            CLArgument args[] = { params... };
            size_t size = sizeof( args ) / sizeof( CLArgument );
            myArguments = std::vector<CLArgument>( args, args + size );

            generateBuffers();
        }

    template<class ...Arguments>
    T operator()( Arguments... params )
        {
            setArguments( params... );
            return run();
        }
#endif

    /*
     * If not using C++11, you will have to create a
     *  std::vector<CLArgument> of arguments,
     *  and call the function using the run() method.
```

```
     */
    virtual T run();
    T run( std::string type );

    T operator()( std::vector<CLArgument> arguments )
        {
            myArguments = arguments;
            generateBuffers();
            return run();
        }

protected:
    CLContext myContext;
    std::vector<CLArgument> myArguments;
    std::vector<cl::Buffer*> myBuffers;
    std::string myFunction;
    std::string myKernel;
    cl::Kernel myCLKernel;
    bool kernelBuilt;

    virtual void generateKernelSource( const std::string type, std::string &source, std
        ::string &kernel_name );
    void generateBuffers();
    void copyBuffersToDevice();
    void copyBuffersFromDevice();
    cl::Kernel generateKernel( std::string src, std::string kernel_name, std::string
        compiler_flags="" );
    void enqueueKernel( cl::Kernel &kernel, std::vector<int> &globalDimensions, std::
        vector<int> &globalOffset, std::vector<int> &localDimensions );
};

/**
 * Generate the kernel that will be called
 *  to run the OpenCL function.
 *
 * @param type
 *  The return type of the OpenCL function
 * @param source
 *  A string in which to put the OpenCL code
 *   along with the generated kernel.
 * @param kernel_name
 *  A string in which to put the name of the generated kernel
 */
template<class T>
void CLFunction<T>::generateKernelSource( const std::string type, std::string &source,
    std::string &kernel_name )
{
    KernelGenerator generator( myFunction, myArguments, type );
    source = myKernel + "\n\n" + generator.generate();
    kernel_name = generator.getKernelFunction();
}

/**
 * Ask all the CLArguments passed to the kernel
 *  to generate a cl::Buffer to pass to OpenCL.
 */
template<class T>
void CLFunction<T>::generateBuffers()
{
    myBuffers.clear();

    for ( unsigned i = 0; i < myArguments.size(); i++ )
    {
        myBuffers.push_back( myArguments[i].getBuffer( myContext ) );
```

```
    }
}

/**
 * Ask all the OpenCL buffers to copy themselves to the
 *  device.
 */
template<class T>
void CLFunction<T>::copyBuffersToDevice()
{
    // Queue up copying those buffers.
    cl::CommandQueue queue = myContext.getCommandQueue();
    for ( unsigned i = 0; i < myArguments.size(); i++ )
    {
        myArguments[i].copyToDevice( queue );
    }
}

/**
 * Create a cl::Kernel from given inputs.
 *
 * @param src
 *  The complete source code to compile.
 * @param kernel_name
 *  The name of the kernel we are interested in inside the source cdoe.
 * @param compiler_flags
 *  Flags to send to the compiler.
 *
 * @return
 *  A cl::Kernel.
 */
template<class T>
cl::Kernel CLFunction<T>::generateKernel( std::string src, std::string kernel_name, std
    ::string compiler_flags )
{
    if ( !kernelBuilt )
    {
        cl::Program program = myContext.buildProgram( src, compiler_flags );
        myCLKernel = cl::Kernel(
            program,
            kernel_name.c_str()
        );

        kernelBuilt = true;
    }

    // Make those buffers arguments for the kernel.
    for ( unsigned i = 0; i < myBuffers.size(); i++ )
    {
        myCLKernel.setArg( i, *myBuffers[i] );
    }

    return myCLKernel;
}

/**
 * Ask all the OpenCL buffers to copy themselves
 *  back from device.
 */
template<class T>
void CLFunction<T>::copyBuffersFromDevice()
{
    cl::CommandQueue queue = myContext.getCommandQueue();
```

```
    for ( unsigned i = 0; i < myArguments.size(); i++ )
    {
        myArguments[i].copyFromDevice( queue );
    }
}

/**
 * Run a kernel on the device.
 *
 * @param kernel
 *  The kernel to run.
 * @param globalDimensions, localDimensions
 *  OpenCL global and local dimensions for the kernel.
 * @param globalOffset
 *  OpenCL global offset.
 */
template<class T>
void CLFunction<T>::enqueueKernel(
    cl::Kernel &kernel,
    std::vector<int> &globalDimensions,
    std::vector<int> &globalOffset,
    std::vector<int> &localDimensions )
{
    // OpenCL only allows up to 3 dimensions.
    cl::NDRange globalWorkSize;
    if ( globalDimensions.size() == 1 )
    {
        globalWorkSize = cl::NDRange( globalDimensions[0] );
    }
    else if ( globalDimensions.size() == 2 )
    {
        globalWorkSize = cl::NDRange(
            globalDimensions[0],
            globalDimensions[1]
        );
    }
    else if ( globalDimensions.size() == 3 )
    {
        globalWorkSize = cl::NDRange(
            globalDimensions[0],
            globalDimensions[1],
            globalDimensions[2]
        );
    }
    else
    {
        assert( false );
    }

    // Global offset
    cl::NDRange globalOffsetRange;
    if ( globalOffset.size() == 1 )
    {
        globalOffsetRange = cl::NDRange( globalOffset[0] );
    }
    else if ( globalOffset.size() == 2 )
    {
        globalOffsetRange = cl::NDRange(
            globalOffset[0],
            globalOffset[1]
        );
    }
    else if ( globalOffset.size() == 3 )
    {
```

```
        globalOffsetRange = cl::NDRange(
            globalOffset[0],
            globalOffset[1],
            globalOffset[2]
        );
    }
    else
    {
        globalOffsetRange = cl::NullRange;
    }

    // Local dimensions
    cl::NDRange localWorkSize;
    if ( localDimensions.size() == 1 )
    {
        localWorkSize = cl::NDRange( localDimensions[0] );
    }
    else if ( localDimensions.size() == 2 )
    {
        localWorkSize = cl::NDRange(
            localDimensions[0],
            localDimensions[1]
        );
    }
    else if ( localDimensions.size() == 3 )
    {
        localWorkSize = cl::NDRange(
            localDimensions[0],
            localDimensions[1],
            localDimensions[2]
        );
    }
    else
    {
        localWorkSize = cl::NullRange;
    }

    // Queue up the kernel.
    cl::CommandQueue &queue = myContext.getCommandQueue();
    queue.enqueueNDRangeKernel(
        kernel,
        globalOffsetRange,
        globalWorkSize,
        localWorkSize,
        NULL,
        NULL
    );

    // This is needed in case the queue supports running things
    //  out of order, the next kernel depends on this one being done,
    //  and no buffers are read back synchronously to the device.
//  queue.enqueueBarrier();
}

/**
 * Run the OpenCL function.
 *
 * There is a specialized version for <void> in CLFunction.cpp.
 */
template<class T>
T CLFunction<T>::run( std::string type )
{
    std::string src;
    std::string kernelFunction;
```

```
    cl::CommandQueue queue = myContext.getCommandQueue();

    generateKernelSource( type, src, kernelFunction );

/*
    std::cout << "FULL SOURCE" << std::endl;
    std::cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << std::endl;
    std::cout << src << std::endl;
    std::cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << std::endl;
*/

    copyBuffersToDevice();
    cl::Kernel kernel = generateKernel( src, kernelFunction );

    T result;
    cl::Buffer resultBuffer;
    // Create the buffer for the result.
    resultBuffer = cl::Buffer(
        myContext.getContext(),
        CL_MEM_WRITE_ONLY,
        sizeof( T ),
        NULL
        );

    // Make it the next kernel argument.
    kernel.setArg( (cl_uint) myArguments.size(), resultBuffer );

    std::vector<int> globalDimensions;
    globalDimensions.push_back( 1 );
    std::vector<int> localDimensions;
    std::vector<int> globalOffset;
    enqueueKernel( kernel, globalDimensions, globalOffset, localDimensions );

    // Enqueue reading the result.
    queue.enqueueReadBuffer(
        resultBuffer,
        CL_TRUE,
        0,
        sizeof( T ),
        &result,
        NULL,
        NULL );

    // Enqueue reading all the results back.
    copyBuffersFromDevice();

    return result;
}

/**
 * Specialization for void.
 */
template<>
void CLFunction<void>::run();

/*
 * Generate run() methods for every standard type.
 */
#define _GEN_CL_FUNCTION_RUN_P( host, kernel ) \
    template<>                                 \
    host CLFunction<host>::run();

_GEN_CL_FUNCTION_RUN_P( cl_int, int )
_GEN_CL_FUNCTION_RUN_P( cl_float, float )
```

```
_GEN_CL_FUNCTION_RUN_P( cl_double, double )
_GEN_CL_FUNCTION_RUN_P( cl_float3, float3 )

#endif
```

cl_test/CLIncludes.h

```
#ifndef _CL_INCLUDES
#define _CL_INCLUDES

#define _CL_USE_EXCEPTIONS
#include <CL/cl.hpp>

#endif
```

cl_test/CLKernel.cpp

```
#include "CLKernel.h"

CLKernel::CLKernel( std::string function,
                    std::string kernel,
                    std::string compilerFlags,
                    const CLContext context )
    : CLFunction( function, kernel, context ),
      globalDimensionsSet( false ),
      globalOffsetSet( false ),
      localDimensionsSet( false ),
      myCompilerFlags( compilerFlags )
{
    myCLKernel = generateKernel( myKernel, myFunction, compilerFlags );
};


void CLKernel::setGlobalDimensions( int dim1 )
{
    globalDimensions.clear();
    globalDimensions.push_back( dim1 );
    globalDimensionsSet = true;
}

void CLKernel::setGlobalDimensions( int dim1, int dim2 )
{
    globalDimensions.clear();
    globalDimensions.push_back( dim1 );
    globalDimensions.push_back( dim2 );
    globalDimensionsSet = true;
}

void CLKernel::setGlobalDimensions( int dim1, int dim2, int dim3 )
{
    globalDimensions.clear();
    globalDimensions.push_back( dim1 );
    globalDimensions.push_back( dim2 );
    globalDimensions.push_back( dim3 );
    globalDimensionsSet = true;
}

void CLKernel::setLocalDimensions( int dim1 )
{
    localDimensions.clear();
    localDimensions.push_back( dim1 );
    localDimensionsSet = true;
}
```

```
void CLKernel::setLocalDimensions( int dim1, int dim2 )
{
    localDimensions.clear();
    localDimensions.push_back( dim1 );
    localDimensions.push_back( dim2 );
    localDimensionsSet = true;
}

void CLKernel::setLocalDimensions( int dim1, int dim2, int dim3 )
{
    localDimensions.clear();
    localDimensions.push_back( dim1 );
    localDimensions.push_back( dim2 );
    localDimensions.push_back( dim3 );
    localDimensionsSet = true;
}

void CLKernel::setGlobalOffset( int dim1 )
{
    globalOffset.clear();
    globalOffset.push_back( dim1 );
    globalOffsetSet = true;
}

void CLKernel::setGlobalOffset( int dim1, int dim2 )
{
    globalOffset.clear();
    globalOffset.push_back( dim1 );
    globalOffset.push_back( dim2 );
    globalOffsetSet = true;
}

void CLKernel::setGlobalOffset( int dim1, int dim2, int dim3 )
{
    globalOffset.clear();
    globalOffset.push_back( dim1 );
    globalOffset.push_back( dim2 );
    globalOffset.push_back( dim3 );
    globalOffsetSet = true;
}

void CLKernel::setLocalArgument( int arg, size_t size )
{
    myCLKernel.setArg( arg, cl::__local( size ) );
}

void CLKernel::run()
{
    // Be nicer, later.
    assert( globalDimensionsSet );

    copyBuffersToDevice();

    for ( unsigned i = 0; i < myBuffers.size(); i++ )
    {
        myCLKernel.setArg( i, *myBuffers[i] );
    }
    enqueueKernel( myCLKernel, globalDimensions, globalOffset, localDimensions );

    copyBuffersFromDevice();
}
```

cl_test/CLKernel.h

```
/**
 * CLKernel encapsulates an OpenCL kernel that is
 *  already in a source file.
 *
 * @author John Kloosterman
 * @date December 2012
 */

#ifndef _CL_KERNEL
#define _CL_KERNEL

#include "CLFunction.h"

class CLKernel : public CLFunction<void>
{
public:

    /**
     * @param function
     *  The name of the kernel in the soruce code.
     * @param kernel
     *  The source code of the kernel.
     * @param compilerFlags
     *  Flags to pass to the OpenCL compiler.
     * @param context
     *  The CLContext to use.
     */
    CLKernel( std::string function,
              std::string kernel,
              std::string compilerFlags = "",
              const CLContext context = CLContext() );
    virtual ~CLKernel() { };

    void setGlobalDimensions( int dim1 );
    void setGlobalDimensions( int dim1, int dim2 );
    void setGlobalDimensions( int dim1, int dim2, int dim3 );

    void setGlobalOffset( int dim1 );
    void setGlobalOffset( int dim1, int dim2 );
    void setGlobalOffset( int dim1, int dim2, int dim3 );

    void setLocalDimensions( int dim1 );
    void setLocalDimensions( int dim1, int dim2 );
    void setLocalDimensions( int dim1, int dim2, int dim3 );

    /**
     * Tell the CLKernel that this kernel has a parameter
     *  declared in __local memory. OpenCL uses this to allow
     *  the host to dynamically determine the size of __local
     *  buffers.
     */
    void setLocalArgument( int arg, size_t size );
    void setCompilerFlags( std::string flags );

    virtual void run();

private:
    bool globalDimensionsSet;
    bool globalOffsetSet;
    bool localDimensionsSet;
    std::vector<int> globalDimensions;
    std::vector<int> globalOffset;
    std::vector<int> localDimensions;
    std::string myCompilerFlags;
```

```
};

#endif
```

cl_test/examples/Makefile

```
OPENCL_LIB = /opt/AMDAPP/lib/x86_64
OPENCL_INCLUDE = /opt/AMDAPP/include

vector_add: vector_add.cpp
    clang++ -std=c++11 -g -L .. -L $(OPENCL_LIB) -I $(OPENCL_INCLUDE) -I .. vector_add.
        cpp -o vector_add -lCLTest -lOpenCL

clean:
    rm -f vector_add
```

cl_test/examples/README

```
This folder contains example OpenCL programs and tests that use the clTest framework.
```

cl_test/examples/vector_add.cl

```
/**
 * vector_add.cl: add two arrays of floats together
 *  into a third array.
 *
 * @author John Kloosterman
 * @date April 9, 2013
 */

__kernel void
vector_add(
    __global float *a,
    __global float *b,
    __global float *result
    )
{
    size_t global_id = get_global_id( 0 );

    result[global_id] = a[global_id] + b[global_id];
}
```

cl_test/examples/vector_add.cpp

```
/**
 * vector_add is a simple example program that uses the clTest
 *  framework to add two arrays of floats using OpenCL.
 */

#include <CLKernel.h>
#include <cassert>
#include <iostream>
#include <fstream>
using namespace std;

int main ( void )
{
    // Declare the input arrays and initialize them.
    const int ARRAY_SIZE = 16384;
    cl_float host_a[ARRAY_SIZE];
    cl_float host_b[ARRAY_SIZE];

    for ( int i = 0; i < ARRAY_SIZE; i++ )
```

```
    {
        host_a[i] = i;
        host_b[i] = i;
    }

    // Declare the output array.
    cl_float host_result[ARRAY_SIZE];

    // Load the source code of the OpenCL kernel into a string.
    ifstream t( "vector_add.cl" );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());

    // Create a CLKernel for the OpenCL kernel.
    CLKernel vector_add( "vector_add", src );

    // Set the global dimensions (i.e. how many instances of
    //  the kernel will be run). We do not set a local (workgroup)
    //  dimension, which means the runtime will choose an optimal
    //  size for us.
    vector_add.setGlobalDimensions( ARRAY_SIZE );

    // Create CLArguments that encapsulate the data in the input
    //  and output arrays.
    //
    // The framework has templates for cl_float, so it is able to infer
    //  the size of the data in the arrays automatically.
    // The default is to copy data both to and from the device. In this
    //  case, copying it from the device is unnecessary.
    CLArgument a( host_a, ARRAY_SIZE );
    CLArgument b( host_b, ARRAY_SIZE );

    // This CLArgument could have been declared like the other two,
    //  but this is how to declare CLArguments for arbitrary
    //  types.
    CLArgument result(
        "float",                          // type on device side
        host_result,                      // pointer to data
        sizeof( cl_float ) * ARRAY_SIZE,  // size of data
        false,                            // copy to device
        true                              // copy from device
        );

    // Call the kernel.
    vector<CLArgument> args;
    args.push_back( a );
    args.push_back( b );
    args.push_back( result );
    vector_add( args );

    // Output results.
    for ( int i = 0; i < ARRAY_SIZE; i++ )
    {
        cout << host_a[i] << " + " << host_b[i] << " = " << host_result[i] << endl;
        assert( host_a[i] + host_b[i] == host_result[i] );
    }

    return 0;
}
```

cl_test/KernelGenerator.cpp

```
#include "KernelGenerator.h"
#include <iostream>
```

```
#include <sstream>
using namespace std;

KernelGenerator::KernelGenerator(
    std::string function,
    std::vector<CLArgument> &arguments,
    std::string returnType )
    : myFunction(function), myReturnType( returnType ), myArguments( arguments )
{

}

string KernelGenerator::intToString( int i )
{
    return static_cast<ostringstream*>( &(ostringstream() << i) )->str();
}

/**
 * This does a bunch of text processing to turn
 *  a signature like
 *
 *   int some_function ( int a, int b ) { body }
 *
 *  into a kernel like
 *
 *   __kernel void _autogen_run_some_function(
 *       __global int *a,
 *       __global int *b,
 *       __global int *ret
 *     )
 *   {
 *       *ret = some_function( *a, *b );
 *   }
 */
string KernelGenerator::generate()
{
    string ret =
        "/** Automatically generated by KernelGenerator **/\n" \
        "__kernel void _autogen_run_" + myFunction;
    ret += "( ";

    for ( unsigned i = 0; i < myArguments.size(); i++ )
    {
        // global *someType arg3,
        ret += "__global " + myArguments[i].getType() + " *arg" + intToString( i ) + ",
             ";
    }

    if ( myReturnType != "void" )
        ret += "__global " + myReturnType + " *ret ";
    else
    {
        // Take off the comma
        ret.erase( ret.end() - 2 );
    }

    ret +=")\n";

    ret += "{";
    ret += "\n";

    ret += "\t";
    if ( myReturnType != "void" )
        ret += "*ret = ";
```

```
    ret += myFunction + "( ";
    for ( unsigned i = 0; i < myArguments.size(); i++  )
    {
        if ( !myArguments[i].isArray() )
            ret += "*";

        ret += "arg" + intToString( i ) + ", ";
    }

    // take off last comma.
    ret.erase( ret.length() - 2, 1 );

    ret += ");";
    ret += "\n";

    ret += "}";

    return ret;
}

string KernelGenerator::getKernelFunction()
{
    return "_autogen_run_" + myFunction;
}
```

cl_test/KernelGenerator.h

```
/**
 * Class to generate an OpenCL kernel
 *  that calls an non-kernel OpenCL function.
 *
 * @author John Kloosterman
 * @date December 2012
 */

#ifndef _KERNEL_GENERATOR_H
#define _KERNEL_GENERATOR_H

#include <string>
#include <vector>

#include "CLArgument.h"

class KernelGenerator
{
public:
    KernelGenerator(
        std::string function,
        std::vector<CLArgument> &arguments,
        std::string returnType );
    std::string generate();
    std::string getKernelFunction();

private:
    std::string myFunction;
    std::string myReturnType;
    std::vector<CLArgument> &myArguments;

    std::string intToString( int i );
};

#endif
```

cl_test/Makefile

```
OPENCL_INCLUDE = /opt/AMDAPP/include

all: KernelGenerator.o CLArgument.o CLContext.o CLKernel.o CLFunction.o *.cpp *.h
    ar rcs libCLTest.a KernelGenerator.o CLArgument.o CLContext.o CLKernel.o CLFunction
        .o

%.o: %.cpp *.h
    clang++ -I $(OPENCL_INCLUDE) -Wall -g -c $< -o $@

clean:
    rm -f *.a *.o
```

cl_test/README

```
cl_test: A framework for running and unit testing OpenCL code.
=============================================================

See the code examples in examples/ for how to use this framework.
All of the other components in my senior project use the framework
as well, which can serve as more complex, real-world examples.
```

A.2 kalah

kalah/board.c

```
/**
 * C interface for Mankalah boards.
 *
 * @author John Kloosterman
 * @date Dec. 24, 2012
 *
 * Based off of Prof. Plantinga's C# code for CS212.
 */

#include "board.h"

#ifndef _OPENCL_
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#endif

/**
 * Initialize a board with positions for the beginning
 *  of a game.
 *
 * @param b
 *  A board to initialize
 * @param to_move
 *  The player who will be going first.
 */
void board_initialize( Board *b, PlayerPosition to_move )
{
    int i;

    for ( i = 0; i < 13; i++ )
        b->board[i] = 4;
    b->board[6] = 0;
    b->board[13] = 0;
    b->player_to_move = to_move;
    b->score = 0;
```

```
    b->legal_move = TRUE;
}

/**
 * Determine whether a move is legal for the current
 *  player on a board.
 *
 * @param b
 *  The board in question
 * @param move
 *  The cell in the board whose stones to move.
 *
 * @return
 *  TRUE if a legal move, FALSE otherwise.
 */
int board_legal_move( Board *b, int move )
{
    if ( b->player_to_move == TOP
         && move >= 7
         && move <= 12
         && b->board[move] != 0)
    {
        return TRUE;
    }
    else if (
        b->player_to_move == BOTTOM
        && move >= 0
        && move <= 5
        && b->board[move] != 0 )
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

/**
 * Determine whether the game is over, given
 *  a board from a game.
 *
 * @param b
 *  The board in question
 */
int board_game_over( Board *b )
{
    if ( b->board[0] == 0 && b->board[1] == 0 && b->board[2] == 0 &&
         b->board[3] == 0 && b->board[4] == 0 && b->board[5] == 0 )
    {
        return TRUE;
    }
    else if ( b->board[7] == 0 && b->board[8] == 0 && b->board[9] == 0 &&
              b->board[10] == 0 && b->board[11] == 0 && b->board[12] == 0 )
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

/**
```

```
 * Determine who won the game, given a board for
 *  which the game is over.
 *
 * @param b
 *  A board for which the game is over.
 */
PlayerPosition board_winner( Board *b )
{
    int top = board_top_score( b );
    int bottom = board_bottom_score( b );

    if ( top > bottom )
        return TOP;
    else if ( bottom > top )
        return BOTTOM;
    else
        return NOBODY;
}

/**
 * Return the score of the player in TOP
 *  position.
 */
int board_top_score( Board *b )
{
    int i;

    int score = 0;
    for ( i = 7; i <= 13; i++ )
        score += b->board[i];

    return score;
}

/**
 * Return the score of the player in
 *  BOTTOM position.
 */
int board_bottom_score( Board *b )
{
    int i;

    int score = 0;
    for ( i = 0; i <= 6; i++ )
        score += b->board[i];

    return score;
}

/**
 * Perform a move on a board.
 *
 * @param b
 *  The board to make the move on. This board
 *   will be modified by this function.
 * @param move
 *  The index of the move to make.
 *
 * @return
 *  The number of stones captured in the move,
 *   if any.
 */
int board_make_move( Board *b, int move )
{
```

```
    int stones, capture, pos;

    // Check if a legal move.
    if ( !board_legal_move( b, move ) )
    {
#ifndef _OPENCL_
        printf( "Illegal move: %d; terminating.", move );
        abort();
#endif

        return -1;
    }

    // Pick up the stones
    stones = b->board[move];
    b->board[move] = 0;
    capture = 0;

    // distribute the stones
    for ( pos = move + 1; stones > 0; pos++ )
    {
        // Don't add stones to opponent's kalah
        if ( b->player_to_move == TOP && pos == 6 )
            pos++;
        if ( b->player_to_move == BOTTOM && pos == 13 )
            pos++;
        if ( pos==14 )
            pos=0;

        b->board[pos]++;
        stones--;
    }
    pos--;

    // was there a capture by TOP?
    if ( b->player_to_move == TOP && pos > 6 && pos < 13 && b->board[pos]==1 && b->
        board[12-pos] > 0 )
    {
        capture = b->board[12-pos] + 1;
        b->board[13] += b->board[12-pos];
        b->board[12-pos] = 0;
        b->board[13]++;
        b->board[pos]=0;
    }

    // was there a capture by BOTTOM?
    if ( b->player_to_move == BOTTOM && pos >= 0 && pos < 6 && b->board[pos] == 1 && b
        ->board[12-pos] > 0 )
    {
        capture = b->board[12-pos] + 1;
        b->board[6] += b->board[12-pos];
        b->board[12-pos] = 0;
        b->board[6]++;
        b->board[pos] = 0;
    }

    // who gets the next move?
    if ( b->player_to_move == TOP )
    {
        if ( pos != 13 )
            b->player_to_move = BOTTOM;
    }
    else
    {
```

```
        if ( pos != 6 )
            b->player_to_move = TOP;
    }

    return capture;
}

/**
 * Output a board to standard output.
 */
void board_print( Board *b )
{
#ifndef _OPENCL_
    int i;

    printf("\n    ");
    for (i=12; i>=7; i--)
        printf("%d  ", b->board[i] );
    printf("\n");
    printf("%d                    %d\n", b->board[13], b->board[6]);
    printf("    ");
    for (i=0; i<=5; i++)
        printf("%d  ", b->board[i]);
    printf("\n");
    printf( "Score: %d\n", b->score );
    printf( "Legal move: %d\n", b->legal_move );
    printf( "Player to move: %d\n", b->player_to_move );
#endif
}

/**
 * Determine whether two boards are equal. This doesn't
 *  check the score field.
 *
 * @param a, b
 *  The boards to compare.
 */
int board_equal( Board *a, Board *b )
{
    int i;

    if ( !(a->legal_move) && !(b->legal_move) )
        return TRUE;

    if ( a->legal_move != b->legal_move )
        return FALSE;

    if ( a->player_to_move != b->player_to_move )
        return FALSE;

    for ( i = 0; i < 14; i++ )
    {
        if ( a->board[i] != b->board[i] )
            return FALSE;
    }

    return TRUE;
}
```

kalah/board.h

```
#ifndef _BOARD_H
#define _BOARD_H
```

```
#include "types.h"

/*
 * C interface for Mankalah boards.
 *
 * @author John Kloosterman
 * @date Dec. 24, 2012
 */

#define BOARD_SIZE 14
#define TOP_KALAH 13
#define BOTTOM_KALAH 6

// Whose turn it is to move.
typedef CL_CHAR PlayerPosition;
#define TOP 0
#define BOTTOM 1
#define NOBODY 2

// The state of a board.
typedef struct {
    CL_CHAR board[BOARD_SIZE];
    PlayerPosition player_to_move;
    CL_INT score;
    CL_CHAR legal_move;
} Board;

void board_initialize( Board *b, PlayerPosition to_move );
int board_legal_move( Board *b, int move );
int board_game_over( Board *b );
int board_make_move( Board *b, int move );
PlayerPosition board_winner( Board *b );
int board_top_score( Board *b );
int board_bottom_score( Board *b );
void board_print( Board *b );
int board_equal( Board *a, Board *b );

#endif
```

kalah/depths.h

```
#ifndef _DEPTHS_H

/**
 * Constants that determine how deep minimax
 *  searching goes for minimax players.
 */

// The number of parallel trees that are evaluated in a batch
//  in the OpenCL player.
#define WORKGROUP_SIZE 500000

// Depth of first level of sequential Minimax. Increase this value to
//  go deeper without changing the performance tuning for the GPU.
#define PRE_DEPTH 0
// Depth of second level of sequential Minimax. The larger this is,
//  the more parallel workgroups are spawned at the same time, and
//  the more memory is needed and transferred. The optimal size depends
//  on the type of OpenCL device being used.
#define SEQUENTIAL_DEPTH 9
// Depth of the parallel minimax component. This is limited to 4 on
//  my test system because the maximum local workgroup size on AMD
//  GPUs is 256, and 2^4 is 216.
#define PARALLEL_DEPTH 4
```

```
// One of the sequential levels overlaps with the parallel level,
//  which is why we have to subtract 1.
#define MINIMAX_DEPTH (PRE_DEPTH + SEQUENTIAL_DEPTH + PARALLEL_DEPTH - 1)

#endif
```

kalah/evaluate.c

```
#include "evaluate.h"

int minimax_eval( Board *b )
{
    int i;

    // Kalah counts 5 times more, but stones count too.
    int score = 5 * ( b->board[13] - b->board[6] );

    for ( i = 0; i <= 5; i++ )
        score -= b->board[i];

    for ( i = 7; i <= 12; i++ )
        score += b->board[i];

    return score;
}
```

kalah/evaluate.h

```
#ifndef _EVALUATE_H
#define _EVALUATE_H

/**
 * The minimax evaluate function,
 *   f: Board -> int
 *  that is used by all the minimax players.
 *
 * @author John Kloosterman
 * @date Jan. 5, 2012
 */

#include "board.h"

int minimax_eval( Board *b );

#endif
```

kalah/kalah_player.h

```
#ifndef _KALAH_PLAYER_H
#define _KALAH_PLAYER_H

#include "board.h"

typedef int (*PlayerMoveFunction)( Board *b );
typedef const char *(*PlayerNameFunction)( void );

typedef struct
{
    PlayerMoveFunction make_move;
    PlayerNameFunction get_name;
} KalahPlayer;

#endif
```

kalah/main.cpp

```
extern "C" {
#include "board.h"
#include "kalah_player.h"
#include "simple_players.h"
}

#include "opencl_player.h"

#include <cstdio>

#define MAX_SCORE 96

int play_game( KalahPlayer top, KalahPlayer bottom, PlayerPosition first_player )
{
    Board b_data;
    Board *b = &b_data;  // as shorthand

    board_initialize( b, first_player );

    if ( first_player == TOP )
        printf( "Player %s begins.\n", top.get_name() );
    else
        printf( "Player %s begins.\n", bottom.get_name() );

    while ( !board_game_over( b ) )
    {
        int move;

        board_print( b );
        printf( "\n" );

        if ( b->player_to_move == TOP )
        {
            move = top.make_move( b );
            printf( "TOP (%s) chooses move %d.\n", top.get_name(), move );
        }
        else
        {
            move = bottom.make_move( b );
            printf( "BOTTOM (%s) chooses move %d.\n", bottom.get_name(), move );
        }

        board_make_move( b, move );

        if ( board_game_over( b ) )
        {
            if ( board_winner( b ) == TOP )
            {
                printf( "TOP (%s) wins, %d to %d.\n",
                        top.get_name(),
                        board_top_score( b ),
                        board_bottom_score( b ) );
            }
            else if ( board_winner( b ) == BOTTOM )
            {
                printf( "BOTTOM (%s) wins, %d to %d.\n",
                        bottom.get_name(),
                        board_bottom_score( b ),
                        board_top_score( b ) );
            }
            else
            {
```

```
                printf( "TIE!\n" );
            }
        }
        else
        {
            if ( b->player_to_move == TOP )
                printf( "TOP (%s) to move.\n", top.get_name() );
            else
                printf( "BOTTOM (%s) to move.\n", bottom.get_name() );
        }
    }

    return board_top_score( b );
}

void match ( KalahPlayer top, KalahPlayer bottom )
{
    printf( "============ Game 1 ============\n" );
    int game1 = play_game( top, bottom, TOP );

    printf( "============ Game 2 ============\n" );
    int game2 = play_game( top, bottom, BOTTOM );

    printf( "================================\n\n" );

    int top_score = game1 + game2;
    if ( top_score > ( MAX_SCORE / 2 ) )
        printf( "TOP (%s) wins, %d to %d.\n", top.get_name(), top_score, MAX_SCORE -
            top_score );
    else if ( top_score < ( MAX_SCORE / 2 ) )
        printf( "BOTTOM (%s) wins, %d to %d.\n", bottom.get_name(), MAX_SCORE -
            top_score, top_score );
    else
        printf( "TIE!\n" );
}

int main ( void )
{
    KalahPlayer bottom = bonzo_player();
    KalahPlayer top = opencl_minimax_player();
    match( top, bottom );

    return 0;
}
```

kalah/Makefile

```
OPENCL_LIB = /opt/AMDAPP/lib/x86_64
OPENCL_INCLUDE = /opt/AMDAPP/include
CL_TEST = ../cl_test

all: opencl_player_tests kalah

opencl_player_tests: opencl_player.cl opencl_player_tests.cpp opencl_player.cpp board.o
     tree_array.o simple_players.o evaluate.o
    clang++ -Wall -g -I $(OPENCL_INCLUDE) -I $(CL_TEST) -L $(OPENCL_LIB) -L $(CL_TEST)
        \
     evaluate.o opencl_player.cpp opencl_player_tests.cpp tree_array.o board.o
         simple_players.o \
     -o opencl_player_tests -lCLTest -lOpenCL

kalah: main.o board.o simple_players.o opencl_player.o tree_array.o opencl_player.cl
    evaluate.o
    clang++ main.o board.o simple_players.o opencl_player.o tree_array.o evaluate.o \
```

```
        -L $(OPENCL_LIB) -L $(CL_TEST) -Wall -g -o kalah -lCLTest -lOpenCL

%.o: %.cpp *.h
    clang++ -Wall -I /opt/AMDAPP/include -I ../cl_test/ -g -c $< -o $@

%.o: %.c *.h
    clang -Wall -g -c $< -o $@

opencl_player.cl: opencl_player.cl.in board.c depths.h tree_array.c evaluate.c
    cpp opencl_player.cl.in > opencl_player.cl

clean:
    rm -f kalah opencl_player.cl *.o opencl_player_tests
```

kalah/opencl_player.cl.in

```
/**
 * Parallel OpenCL minimax component. This corresponds to
 *  level (3) in header of opencl_player.cpp.
 *
 * @author John Kloosterman
 * @date Dec. 2012 - Jan. 2013
 */

#define _OPENCL_
#include "board.c"
#include "tree_array.c"
#include "depths.h"
#include "evaluate.c"

/**
 * Determine the index of the first leaf node
 *  in the tree array.
 */
int leaf_start ( void )
{
    return tree_array_size( 6, PARALLEL_DEPTH - 2 );
}

// (a) generate the boards in log(n) time.
// Note that to do this in parallel, every thread must be in the same local
//  group for the synchronization to work.
// AMD OpenCL allows only 256 work-items per work-group, so we are limited
//  to a depth of 3 with this approach (when there will be 216 leaf nodes).

/**
 * Evaluate the score of a given board
 *  and put it in board_array.
 */
void evaluate_board( int board, __local Board *board_array )
{
    Board b = board_array[board];
    board_array[board].score = minimax_eval( &b );
}

/**
 * Generate the tree of possible boards resulting
 *  from this workgroup's start board.
 *
 * This algorithm uses
 *   1 thread for the root board
 *   6 threads for the second level
 *   36 threads for the third level
 *   216 threads for the fourth level.
```

```
 * Therefore, it can generate the boards in O(lg(n), n) time.
 */
void generate_boards( __global Board *host_start, __local Board *board_array )
{
    int tree = get_global_id( 0 );
    int local_id = get_local_id( 1 );
    int move = local_id % 6;
    int move_offset;

    // Put in the root node.
    if ( local_id == 0 )
    {
        board_array[0] = host_start[tree];
    }
    barrier( CLK_LOCAL_MEM_FENCE );

    Board m_board;
    int power_of_six = 6;
    int level_start = 1;
    for ( int i = 1; i < PARALLEL_DEPTH; i++ )
    {
        if ( local_id < power_of_six )
        {
            // Integer division rounds down.
            int parent = tree_array_parent( 6, level_start + local_id );
            m_board = board_array[parent];

            // Calculate the move offset based on whose turn it is.
            if ( m_board.player_to_move == TOP )
                move_offset = 7;
            else
                move_offset = 0;

            // Make the move.
            if ( m_board.legal_move
                 && board_legal_move( &m_board, move + move_offset ) )
            {
                board_make_move( &m_board, move + move_offset );
            }
            else
            {
                m_board.legal_move = FALSE;
            }

            board_array[level_start + local_id] = m_board;
        }

        level_start += power_of_six;
        power_of_six *= 6;

        barrier( CLK_LOCAL_MEM_FENCE );
    }
}

/**
 * Evaluate all the boards in the move tree.
 */
void evaluate_boards( __local Board *board_array )
{
    int local_id = get_local_id( 1 );

    // Evaluate boards.
    evaluate_board( local_id, board_array);
```

```
    // Some threads have to do a second one, because
    //  there are more boards than threads.
    if ( local_id < 43 )
        evaluate_board( local_id + 216, board_array );
}

/**
 * Run minimax on the board tree, using the same log(n)
 *  tree strategy as for generate_boards(), but in reverse:
 *  starting from the second-to-bottom level with 36 threads
 *  and working upwards.
 */
void minimax( __local Board *board_array )
{
    // Minimax
    int power_of_six = (int) pow( 6.0f, PARALLEL_DEPTH - 2 );
    int level_start = leaf_start();
    int local_id = get_local_id( 1 );

    for ( int i = PARALLEL_DEPTH - 2; i >= 0; i-- )
    {
        level_start -= power_of_six;

        if ( local_id < power_of_six )
        {
            // The board in board_array.
            int board_offset = level_start + local_id;
            // The location in board_array of the children of this board.
            int child_offset = tree_array_first_child( 6, board_offset );
            PlayerPosition playerToMove;
            int score;

            Board board = board_array[board_offset];

            if ( board.legal_move && !board_game_over( &board ) )
            {
                playerToMove = board.player_to_move;
                if ( playerToMove == TOP )
                    score = INT_MIN;
                else
                    score = INT_MAX;

                for ( int i = 0; i < 6; i++ )
                {
                    Board child = board_array[child_offset + i];
                    if ( child.legal_move
                         && ( ( playerToMove == TOP && child.score > score )
                              || ( playerToMove == BOTTOM && child.score < score ) ) )
                    {
                        score = child.score;
                    }
                }

                board_array[board_offset].score = score;
            }
        }

        power_of_six /= 6;
        barrier( CLK_LOCAL_MEM_FENCE );
    }
}

/**
 * Kernel to perform 4 levels of minimax on a game board.
```

```
 *
 * Dimensions:
 *   global: (boards to do minimax on)x216 threads
 *   local: 1x216
 *
 * The board to do minimax on is at host_start[get_global_id(0)]
 * The minimax value of that board is stored in that board's
 *   @c score field.
 */
__kernel void opencl_player( __global Board *host_start )
{
    int tree = get_global_id( 0 );

    // does this need a literal size? Probably.
    __local Board board_array[259];

    // If this whole tree isn't meaningful, don't do anything.
    if ( !host_start[tree].legal_move )
        return;

    generate_boards( host_start, board_array );
    barrier( CLK_LOCAL_MEM_FENCE );

    evaluate_boards( board_array );
    barrier( CLK_LOCAL_MEM_FENCE );

    minimax( board_array );
    barrier( CLK_LOCAL_MEM_FENCE ); // Some of these might be redundant

    // Store back in global memory.
    if ( get_local_id( 1 ) == 0 )
        host_start[tree].score = board_array[0].score;
}

/**************************************/

/**
 * Testing stub for generate_boards.
 */
__kernel void generate_boards_test( __global Board *start_board, __global Board *
    host_board_array )
{
    __local Board board_array[259];
    int local_id = get_local_id( 1 );

    generate_boards( start_board, board_array );
    barrier( CLK_LOCAL_MEM_FENCE );

    // Copy from local to global mem.
    host_board_array[local_id] = board_array[local_id];

    if ( local_id < 43 )
        host_board_array[local_id + 216] = board_array[local_id + 216];
}

/**
 * Testing stub for evaluate_boards().
 */
__kernel void evaluate_boards_test( __global Board *host_board_array )
{
    __local Board board_array[259];
    int local_id = get_local_id( 1 );

    board_array[local_id] = host_board_array[local_id];
```

```
    if ( local_id < 43 )
        board_array[local_id + 216] = host_board_array[local_id + 216];
    barrier( CLK_LOCAL_MEM_FENCE );

    evaluate_boards( board_array );
    barrier( CLK_LOCAL_MEM_FENCE );

    host_board_array[local_id] = board_array[local_id];

    if ( local_id < 43 )
        host_board_array[local_id + 216] = board_array[local_id + 216];
}

/**
 * Testing stub for minimax()
 */
__kernel void minimax_test( __global Board *host_board_array )
{
    __local Board board_array[259];
    int local_id = get_local_id( 1 );

    board_array[local_id] = host_board_array[local_id];
    if ( local_id < 43 )
        board_array[local_id + 216] = host_board_array[local_id + 216];
    barrier( CLK_LOCAL_MEM_FENCE );

    minimax( board_array );
    barrier( CLK_LOCAL_MEM_FENCE );

    host_board_array[local_id] = board_array[local_id];

    if ( local_id < 43 )
        host_board_array[local_id + 216] = board_array[local_id + 216];
}
```

kalah/opencl_player.cpp

```
/**
 * OpenCL minimax player.
 *
 * Minimax trees have the property that every subtree of a
 *  minimax tree is itself a minimax tree. We use that to
 *  split up the computation into 3 stages, whose depths are
 *  defined in depths.h:
 *
 * (1) PRE_DEPTH levels of sequential minimax on the CPU, whose
 *     leaf nodes are the level (2) trees.
 * (2) SEQUENTIAL_DEPTH levels of sequential minimax on the CPU.
 *     This is done in 3 stages:
 *       a) All the leaf nodes of this tree for which the game is
 *          not over are put into an array.
 *       b) The parallel minimax implementation, level (3) is called
 *          on each of those leaf nodes.
 *       c) Minimax is run on the leaf nodes.
 *     The leaf nodes have to remain in memory in order to be passed
 *      off to the parallel level, which limits how deep this level
 *      can search. If it's too small, the GPU is under-utilized, and if
 *      it's too big, we use and transfer around too much memory.
 * (3) PARALLEL_DEPTH levels of minimax are run on the GPU.
 *
 * @author John Kloosterman
 * @date Dec. 26, 2012 - Jan. 5, 2013
 */
```

```
#include "opencl_player.h"

extern "C" {
#include "board.h"
#include "tree_array.h"
#include "simple_players.h"
#include "evaluate.h"
}

#include "depths.h"

#include <iostream>
#include <cmath>
#include <climits>
#include <fstream>
#include <string>
#include <list>
using namespace std;

/**
 * Set the board that the OpenCLPlayer should evaluate
 *  when makeMove() is called.
 */
void OpenCLPlayer::set_board( Board b )
{
    myStartBoard = b;
}

/**
 * Run levels (2) and (3) of minimax on the current board.
 */
MinimaxResult OpenCLPlayer::makeMove()
{
    // Handle the case when the game ends at our root.
    if ( board_game_over( &myStartBoard ) )
    {
        MinimaxResult mr;

        mr.move = -30;
        mr.score = minimax_eval( &myStartBoard );

        return mr;
    }

    // Create the start boards
    generate_start_boards();

    if ( !myStartBoards.size() )
    {
        return run_minimax( myStartBoard, mySequentialDepth );
    }

    // C++ guarantees vector elements are stored contiguously.
    CLArgument start_boards(
        "Board",
        &myStartBoards[0],
        myStartBoards.size(),
        false, true );
    start_boards.makePersistent( myContext );
    int num_leaf_nodes = (int) myStartBoards.size();

    if ( num_leaf_nodes )
    {
        int offset = 0;
```

```
        int items;
        int iterations = 0;

        // We have to batch the work to the GPU, because if one batch
        //  takes too long, the system gets unreponsive, and after a
        //  certain point, the watchdog timer goes off and the system
        //  locks up.
        do {
            if ( ( num_leaf_nodes - offset ) < WORKGROUP_SIZE )
                items = num_leaf_nodes - offset;
            else
                items = WORKGROUP_SIZE;

            vector<CLArgument> args;
            args.push_back( start_boards );
            opencl_player.setGlobalDimensions( items, 216 );
            opencl_player.setGlobalOffset( offset, 0 );
            opencl_player.setLocalDimensions( 1, 216 );
            opencl_player( args );

            offset += WORKGROUP_SIZE;
            iterations++;
        } while ( offset < num_leaf_nodes );
    }

    // Run minimax on the start boards.
    minimax_idx = 0;
    MinimaxResult move = run_minimax( myStartBoard, mySequentialDepth );
    assert( board_legal_move( &myStartBoard, move.move ) );

    return move;
}

/**
 * Recursively find the leaf nodes of the level (2) minimax tree,
 *  and put then in the myStartBoards vector.
 */
void OpenCLPlayer::generate_board( Board parent, int depth )
{
    if ( depth == 0 )
    {
        myStartBoards.push_back( parent );
        return;
    }
    else if ( board_game_over( &parent ) )
    {
        return;
    }

    for ( int i = 0; i < 6; i++ )
    {
        int move_offset = 0;
        Board m_board = parent;

        if ( m_board.player_to_move == TOP )
            move_offset = 7;

        if ( board_legal_move( &m_board, i + move_offset ) )
        {
            board_make_move( &m_board, i + move_offset );
            generate_board( m_board, depth - 1 );
        }
    }
}
```

```
/**
 * Generate all the level (2) leaf nodes for the current
 *  board.
 */
void OpenCLPlayer::generate_start_boards()
{
    myStartBoards.clear();
    generate_board( myStartBoard, mySequentialDepth );
};

/**
 * Run minimax on the level (2) leaf nodes in myStartBoards,
 *  which have had their scores populated by the level (3)
 *  GPU component.
 */
MinimaxResult OpenCLPlayer::run_minimax( Board &parent, int depth )
{
    MinimaxResult best_result;
    best_result.move = -1;

    if ( depth == 0 )
    {
        MinimaxResult mr;

        // Make sure we are reading everything in the same order.
        assert( board_equal( &myStartBoards[minimax_idx], &parent ) );

        mr.score = myStartBoards[minimax_idx].score;
        minimax_idx++;

        mr.move = -1;

        return mr;
    }
    else if ( board_game_over( &parent ) )
    {
        MinimaxResult mr;

        mr.score = minimax_eval( &parent );
        mr.move = -1;

        return mr;
    }

    int move_offset;
    if ( parent.player_to_move == TOP )
    {
        move_offset = 7;
        best_result.score = INT_MIN;
    }
    else
    {
        move_offset = 0;
        best_result.score = INT_MAX;
    }

    MinimaxResult rec_result;
    for ( int i = 0; i < 6; i++ )
    {
        if ( board_legal_move( &parent, i + move_offset ) )
        {
            Board moved_board = parent;
```

```
            board_make_move( &moved_board, i + move_offset );
            rec_result = run_minimax( moved_board, depth - 1 );

            if ( ( parent.player_to_move == TOP && rec_result.score > best_result.score
                 )
                 || ( parent.player_to_move == BOTTOM && rec_result.score < best_result
                     .score ) )
            {
                best_result.move = i + move_offset;
                best_result.score = rec_result.score;
            }
        }
    }

    return best_result;
}

/**
 * Recursively run level (1) minimax on a board.
 *
 * @param player
 *  The OpenCLPlayer to use to evaluate leaf nodes.
 * @param b
 *  The parent of the minimax subtree
 * @param depth
 *  The current minimax tree depth, which increases
 *   as we go down levels.
 */
MinimaxResult opencl_player_pre_minimax( OpenCLPlayer &player, Board &b, int depth )
{
    MinimaxResult ret;

    // Use the OpenCLPlayer object to go deeper.
    if ( depth == PRE_DEPTH )
    {
        player.set_board( b );
        ret = player.makeMove();
        return ret;
    }
    else if ( board_game_over( &b ) )
    {
        ret.score = minimax_eval( &b );
        ret.move = -10;

        return ret;
    }

    MinimaxResult rec_result;
    MinimaxResult best_result;
    best_result.move = -20;

    if ( b.player_to_move == TOP )
    {
        // MAX
        best_result.score = INT_MIN;

        for ( int i = 7; i < 13; i++ )
        {
            if ( board_legal_move( &b, i ) )
            {
                Board moved_board;
                moved_board = b;
                board_make_move( &moved_board, i );
```

```
                rec_result = opencl_player_pre_minimax( player, moved_board, depth + 1
                    );

                if ( rec_result.score > best_result.score )
                {
                    best_result.move = i;
                    best_result.score = rec_result.score;
                }
            }
        }
    }
    else
    {
        // MIN
        best_result.score = INT_MAX;

        for ( int i = 0; i < 6; i++ )
        {
            if ( board_legal_move( &b, i ) )
            {
                Board moved_board;
                moved_board = b;
                board_make_move( &moved_board, i );

                rec_result = opencl_player_pre_minimax( player, moved_board, depth + 1
                    );

                if ( rec_result.score < best_result.score )
                {
                    best_result.move = i;
                    best_result.score = rec_result.score;
                }
            }
        }
    }

    return best_result;
}

/**
 * Bindings from the OpenCL player object to the
 *  C world.
 */

// Instantiating an OpenCLPlayer involves recompiling
//  the kernel, which is slow, so instiantiate only one.
ifstream t("opencl_player.cl");
string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
OpenCLPlayer player( SEQUENTIAL_DEPTH, src );

extern "C" const char *opencl_player_name( void )
{
    return "OpenCL Minimax";
}

extern "C" int opencl_player_move( Board *b )
{
    MinimaxResult ret = opencl_player_pre_minimax( player, *b, 0 );

    return ret.move;
}

KalahPlayer opencl_minimax_player( void )
```

```
{
    KalahPlayer k;

    k.get_name = opencl_player_name;
    k.make_move = opencl_player_move;

    return k;
}
```

kalah/opencl_player.h

```
#ifndef _CL_PLAYER
#define _CL_PLAYER

#include <CLKernel.h>

extern "C" {
#include "board.h"
#include "kalah_player.h"
}

/*
 * OpenCL Mankalah Player.
 */

KalahPlayer opencl_minimax_player( void );

class OpenCLPlayer
{
public:
    OpenCLPlayer( int sequentialDepth, std::string src )
        : mySequentialDepth( sequentialDepth ),
          opencl_player( "opencl_player", src, "", myContext )
        {
        };

    void set_board( Board b );
    MinimaxResult makeMove();

private:
    void generate_start_boards();
    void generate_board( Board parent, int depth );
    MinimaxResult run_minimax( Board &parent, int depth );

    int minimax_idx;
    CLContext myContext;
    int mySequentialDepth;
    int myBoardsSize;
    Board myStartBoard;
    CLKernel opencl_player;

    std::vector<Board> myStartBoards;
};

// Exposed for testing.
MinimaxResult opencl_player_pre_minimax( OpenCLPlayer &player, Board &b, int depth );

#endif
```

kalah/opencl_player_tests.cpp

```
/**
 * Tests to ensure that the components of the OpenCL minimax
 *  player work properly and that its output matches the
```

```
 *  sequential version.
 *
 * @author John Kloosterman
 * @date Jan. 6, 2013
 */

extern "C" {
#include "board.h"
#include "tree_array.h"
#include "simple_players.h"
#include "depths.h"
#include "evaluate.h"
}

#include <sys/time.h>

#include "opencl_player.h"

#include <CLKernel.h>
#include <fstream>
#include <iostream>
#include <cassert>
#include <cstdlib>
#include <climits>
#include <ctime>
using namespace std;

// The number of iterations of each test to run.
#define NUM_ITERATIONS 512

/**
 * Determine whether a given board is a valid
 *  randomly generated board. It doesn't have to be
 *  a possible one, just one that isn't unreasonable.
 *
 * @param b
 *  A board.
 */
bool valid_board( Board b )
{
    if ( board_game_over( &b ) )
        return FALSE;

    for ( int i = 0; i < 14; i++ )
    {
        if ( b.board[i] < 0 || b.board[i] > 96 )
            return false;
    }

    return true;
}

/**
 * Randomly generate a valid board.
 *
 * @return
 *  A random board.
 */
Board generate_valid_board()
{
    Board board;

    board.score = -1;
    board.legal_move = TRUE;
```

```
    do {
        if ( rand() % 2 )
            board.player_to_move = TOP;
        else
            board.player_to_move = BOTTOM;

        int stones_left = 12 * 4;

        // This isn't uniform, but works.
        for ( int i = 0; i < 14; i++ )
        {
            if ( stones_left < 8 )
                break;

            int st = rand() % 8;
            stones_left -= st;

            board.board[i] = st;
        }

    } while ( !valid_board( board ) );

    return board;
}

/**
 * Helper method for @c test_generate boards that recursively
 *  determines whether the OpenCL kernel generated a correct
 *  tree of boards based on a start board.
 *
 * @param board
 *  The start board for this subtree.
 * @param opencl_boards
 *  The array of boards generated by the OpenCL player.
 * @param idx
 *  The index in the tree array of the root of this subtree.
 */
void check_boards( Board *board, Board *opencl_boards, int idx )
{
    assert( board_equal( board, &opencl_boards[idx] ) );

    int move_offset;
    if ( board->player_to_move == TOP )
        move_offset = 7;
    else
        move_offset = 0;

    int child_offset = tree_array_first_child( 6, idx );
    // If our child would be deeper than the OpenCL tree is (4 levels)
    //  we are a leaf node.
    if ( child_offset > 258 )
        return;

    for ( int i = 0; i < 6; i++ )
    {
        if ( board_legal_move( board, i + move_offset ) )
        {
            Board m_board = *board;
            board_make_move( &m_board, i + move_offset );
            check_boards( &m_board, opencl_boards, child_offset + i );
        }
    }
}
```

```
/**
 * Test to determine whether the OpenCL player
 *  correctly generates a tree of possible boards.
 *
 * @param src
 *  The source code for the OpenCL player.
 */
void test_generate_boards( string src )
{
    cout << "Testing generate_boards..." << endl;

    Board boards[259];
    Board start_board;

    CLKernel generate_boards_test( "generate_boards_test", src );

    CLArgument host_boards( "Board", boards, 259 );
    CLArgument start_boards( "Board", &start_board, 1 );
    vector<CLArgument> args;
    args.push_back( start_boards );
    args.push_back( host_boards );
    generate_boards_test.setGlobalDimensions( 1, 216 );
    generate_boards_test.setLocalDimensions( 1, 216 );

    for ( int i = 0; i < NUM_ITERATIONS; i++ )
    {
        start_board = generate_valid_board();
        generate_boards_test( args );

        check_boards( &start_board, boards, 0 );
        cout << "* " << flush;
    }

    cout << endl << "All tests passed." << endl;
}

/**
 * Test to determine whether the OpenCL player correctly
 *  assigns a score to boards.
 *
 * @param src
 *  The source code for the OpenCL player.
 */
void test_evaluate_boards( string src )
{
    cout << "Testing evaluate_boards..." << endl;

    Board boards[259];

    CLKernel evaluate_boards_test( "evaluate_boards_test", src );
    CLArgument host_boards( "Board", boards, 259 );
    vector<CLArgument> args;
    args.push_back( host_boards );
    evaluate_boards_test.setGlobalDimensions( 1, 216 );
    evaluate_boards_test.setLocalDimensions( 1, 216 );

    for ( int j = 0; j < NUM_ITERATIONS; j++ )
    {
        for ( int i = 0; i < 259; i++ )
        {
            boards[i] = generate_valid_board();
        }
```

```
        evaluate_boards_test( args );

        for ( int i = 0; i < 259; i++ )
        {
            assert( boards[i].score == minimax_eval( &boards[i] ) );
        }
        cout << "* " << flush;
    }

    cout << endl << "All tests passed." << endl;
}

/**
 * Helper method for test_minimax to determine whether the minimax
 *  implementation in the OpenCL player worked properly.
 *
 * @param opencl_boards
 *  The array of boards returned from the OpenCL player.
 * @param idx
 *  The root of the current minimax subtree in the array.
 */
int check_minimax( Board *opencl_boards, int idx )
{
    // We are a leaf node
    if ( idx >= 43 )
        return opencl_boards[idx].score;

    int best_score;
    if ( opencl_boards[idx].player_to_move == TOP )
        best_score = INT_MIN;
    else
        best_score = INT_MAX;

    int child_offset = tree_array_first_child( 6, idx );
    bool has_legal_move = false;
    for ( int i = 0; i < 6; i++ )
    {
        if ( !opencl_boards[child_offset + i].legal_move )
            continue;

        int score = check_minimax( opencl_boards, child_offset + i );

        if ( ( opencl_boards[idx].player_to_move == TOP && score > best_score )
             || ( opencl_boards[idx].player_to_move == BOTTOM && score < best_score ) )
        {
            has_legal_move = true;
            best_score = score;
        }
    }

    if ( !has_legal_move )
        return opencl_boards[idx].score;

    assert( opencl_boards[idx].score == best_score );
    return best_score;
}

/**
 * Test to determine whether the minimax implementation in the
 *  OpenCL player works correctly.
 *
 * @param src
 *  The source code for the OpenCL player.
 */
```

```
void test_minimax( string src )
{
    cout << "Testing minimax..." << endl;

    Board boards[259];

    CLKernel minimax_test( "minimax_test", src );
    CLArgument host_boards( "Board", boards, 259 );
    vector<CLArgument> args;
    args.push_back( host_boards );
    minimax_test.setGlobalDimensions( 1, 216 );
    minimax_test.setLocalDimensions( 1, 216 );

    for ( int j = 0; j < NUM_ITERATIONS; j++ )
    {
        // Generate a random array, put scores in the bottom,
        //  and randomly mark some nodes as illegal moves.
        for ( int i = 0; i < 259; i++ )
        {
            boards[i] = generate_valid_board();
            if ( ( rand() % 5 ) == 0 )
                boards[i].legal_move = FALSE;
        }

        if ( !boards[0].legal_move )
            boards[0].legal_move = TRUE;

        // The OpenCL code assumes that all the boards are scored, for game-overs.
        for ( int i = 0; i < 259; i++ )
            boards[i].score = minimax_eval( &boards[i] );

        minimax_test( args );
        check_minimax( boards, 0 );

        cout << "* " << flush;
    }

    cout << endl << "All tests passed." << endl;
}

/**
 * Test to determine whether the parallel component of the OpenCL
 *  player returns the same results as the sequential minimax
 *  algorithm.
 *
 * @param src
 *  The source code for the OpenCL player.
 */
void test_combination( string src )
{
    cout << "Testing OpenCL kernel..." << endl;

    Board start_boards[256];

    CLKernel opencl_player( "opencl_player", src );
    CLArgument host_start_boards( "Board", start_boards, 256 );
    vector<CLArgument> args;
    args.push_back( host_start_boards );
    opencl_player.setGlobalDimensions( 256, 216 );
    opencl_player.setLocalDimensions( 1, 216 );

    for ( int i = 0; i < NUM_ITERATIONS; i++ )
    {
        // Generate start boards.
```

```
        for ( int j = 0; j < 256; j++ )
            start_boards[j] = generate_valid_board();

        // Run kernel
        opencl_player( args );

        // Test result
        for ( int j = 0; j < 256; j++ )
        {
            MinimaxResult mr = minimax_move( &start_boards[j], PARALLEL_DEPTH - 1 );

            if ( start_boards[j].score != mr.score )
            {
                cout << "Parallel: " << start_boards[j].score << " Sequential: " << mr.
                    score << endl;
                board_print( &start_boards[j] );
            }

            assert( start_boards[j].score == mr.score );
        }

        cout << "* " << flush;
    }

    cout << endl << "All tests passed." << endl;
}

/**
 * Test to determine whether the parallel component and the
 *  bottom-most sequential minimax component of the OpenCL player
 *  give output identical with the sequential minimax algorithm.
 *
 * @param src
 *  The source code for the OpenCL player.
 */
void test_opencl_object( string src )
{
    cout << "Testing entire OpenCL component..." << endl;

    OpenCLPlayer opencl_player( SEQUENTIAL_DEPTH, src );

    for ( int i = 0; i < NUM_ITERATIONS; i++ )
    {
        Board start = generate_valid_board();

        opencl_player.set_board( start );
        MinimaxResult ocl_result = opencl_player.makeMove();
        MinimaxResult seq_result = minimax_move( &start, SEQUENTIAL_DEPTH +
            PARALLEL_DEPTH - 1 );

        assert( ocl_result.move == seq_result.move );
        assert( ocl_result.score == seq_result.score );

        cout << "* " << flush;
    }

    cout << endl << "All tests passed." << endl;
}

/**
 * Test to determine whether all 3 levels of the OpenCL player
 *  together give the same result as the sequential minimax
 *  algorithm.
 *
```

```
 * @param src
 *  The source code for the OpenCL player.
 */
void test_pre_minimax( string src )
{
    cout << "Testing pre-OpenCL component minimax..." << endl;
    OpenCLPlayer opencl_player( SEQUENTIAL_DEPTH, src );

    for ( int i = 0; i < NUM_ITERATIONS; i++ )
    {
        Board start = generate_valid_board();

        MinimaxResult p_ocl_result = opencl_player_pre_minimax( opencl_player, start, 0
            );
        MinimaxResult seq_result = minimax_move( &start, PRE_DEPTH + SEQUENTIAL_DEPTH +
            PARALLEL_DEPTH - 1 );

        assert( p_ocl_result.move == seq_result.move );
        assert( p_ocl_result.score == seq_result.score );

        cout << "* " << flush;
    }

    cout << endl << "All tests passed." << endl;
}

/**
 * Test to determine whether being TOP or BOTTOM makes a difference
 *  for the minimax player's choices.
 *
 * @param src
 *  The source code for the OpenCL player.
 */
void test_top_bottom( string src )
{
    cout << "Testing whether being TOP or BOTTOM makes a difference..." << endl;
    OpenCLPlayer opencl_player( 4, src );

    for ( int i = 0; i < NUM_ITERATIONS; i++ )
    {
        Board start = generate_valid_board();

        MinimaxResult ocl_result = opencl_player_pre_minimax( opencl_player, start, 0 )
            ;

        // Flip the board around.
        Board flipped;
        board_initialize( &flipped, TOP );
        if ( start.player_to_move == TOP )
            flipped.player_to_move = BOTTOM;
        else
            flipped.player_to_move = TOP;

        for ( int i = 0; i < 7; i++ )
        {
            flipped.board[i] = start.board[i+7];
            flipped.board[i+7] = start.board[i];
        }

        MinimaxResult flipped_result = opencl_player_pre_minimax( opencl_player,
            flipped, 0 );

        if ( start.player_to_move == TOP )
            assert( ocl_result.move == ( flipped_result.move + 7 ) );
```

```
            else
                assert( ( ocl_result.move + 7 ) == flipped_result.move );

            assert( ocl_result.score == -flipped_result.score );
            cout << "* " << flush;
        }

        cout << endl << "All tests passed." << endl;
    }

    void speed_test( string src )
    {
        KalahPlayer cl_player = minimax_player();

        Board start;
        board_initialize( &start, TOP );
        struct timeval tv_start;
        struct timeval tv_end;

        for ( int i = 0; i < 5; i++ )
        {
            gettimeofday( &tv_start, NULL );
            cl_player.make_move( &start );
            gettimeofday( &tv_end, NULL );

            int diff_sec = tv_end.tv_sec - tv_start.tv_sec;
            int diff_usec = tv_end.tv_usec - tv_start.tv_usec;

            double diff = diff_sec + ( 0.000001 * diff_usec );

            cout << i << ": " << diff << endl;
        }
    }

    int main ( void )
    {
        // Seed the random number generator
        srand( (unsigned) time( NULL ) );

        // Read the source code for the OpenCL player from the file.
        ifstream t("opencl_player.cl");
        string src((std::istreambuf_iterator<char>(t)),
                   std::istreambuf_iterator<char>());

        test_generate_boards( src );
        test_evaluate_boards( src );
        test_minimax( src );
        test_combination( src );
        test_opencl_object( src );
        test_pre_minimax( src );
        test_top_bottom( src );
    //  speed_test( src );

        return 0;
    }
```

kalah/README

```
    OpenCL Mankalah Player
    ======================
    The Makefile will build two executables:
        kalah: play a game of Mankalah between 2 players.
        opencl_player_tests: run the unit and fuzz tests of the OpenCL player.
```

```
To change the search depth, change the values in depths.h.

The board evaluation function is fairly simple, and can be improved upon.
    This function is found in evaluate.c.
```

kalah/simple_players.c

```
/**
 * Definitions of some simple Kalah players:
 *  -Human
 *  -Bonzo (go-again if possible, then first possible move)
 *  -Random
 *  -Sequential Minimax
 *
 * @author John Kloosterman
 * @date Dec. 24, 2012
 */

#include "simple_players.h"
#include "depths.h"
#include "evaluate.h"

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>

/**
 * Human player
 */
const char *human_name( void )
{
    return "Human";
}

int human_move( Board *b )
{
    int move = -1;

    while ( !board_legal_move( b, move ) )
    {
        printf( "Move: " );
        if ( scanf( "%d", &move ) != 1 )
            continue;

        if ( !board_legal_move( b, move ) )
            printf( "Illegal move. Try again.\n" );
    }

    return move;
}

KalahPlayer human_player( void )
{
    KalahPlayer human;

    human.get_name = human_name;
    human.make_move = human_move;

    return human;
}

/**
 * Bonzo player.
```

```
 */
int bonzo_move( Board *b )
{
    int i;

    if (b->player_to_move == TOP)
    {
        for (i=12; i>=7; i--)               // try first go-again
            if (b->board[i] == 13-i) return i;
        for (i=12; i>=7; i--)               // otherwise, first
            if (b->board[i] > 0) return i;    // available move
    } else {
        for (i=5; i>=0; i--)
            if (b->board[i] == 6-i) return i;
        for (i=5; i>=0; i--)
            if (b->board[i] > 0) return i;
    }
    return -1;              // an illegal move if there aren't any legal ones.
}                           // this can't happen unless game is over.


const char *bonzo_name( void )
{
    return "Bonzo";
}

KalahPlayer bonzo_player( void )
{
    KalahPlayer bonzo;

    bonzo.get_name = bonzo_name;
    bonzo.make_move = bonzo_move;

    return bonzo;
}

/**
 * Sequential minimax player
 */
const char *minimax_name( void )
{
    return "Minimax";
}

MinimaxResult minimax_move( Board *b, int depth )
{
    int i;
    MinimaxResult ret;
    MinimaxResult rec_result;
    MinimaxResult best_result;
    Board moved_board;
    best_result.move = -1;

    if ( depth == 0 || board_game_over( b ) )
    {
        ret.move = -1;
        ret.score = minimax_eval( b );

        return ret;
    }

    if ( b->player_to_move == TOP )
    {
        // MAX
```

```
        best_result.score = INT_MIN;

        for ( i = 7; i < 13; i++ )
        {
            if ( board_legal_move( b, i ) )
            {
                moved_board = *b;
                board_make_move( &moved_board, i );

                rec_result = minimax_move( &moved_board, depth - 1 );

                if ( rec_result.score > best_result.score )
                {
                    best_result.move = i;
                    best_result.score = rec_result.score;
                }
            }
        }
    }
    else
    {
        // MIN
        best_result.score = INT_MAX;

        for ( i = 0; i < 6; i++ )
        {
            if ( board_legal_move( b, i ) )
            {
                moved_board = *b;
                board_make_move( &moved_board, i );

                rec_result = minimax_move( &moved_board, depth - 1 );

                if ( rec_result.score < best_result.score )
                {
                    best_result.move = i;
                    best_result.score = rec_result.score;
                }
            }
        }
    }

    return best_result;
}

int minimax_make_move( Board *b )
{
    MinimaxResult res = minimax_move( b, MINIMAX_DEPTH );

    return res.move;
}

KalahPlayer minimax_player( void )
{
    KalahPlayer minimax;

    minimax.get_name = minimax_name;
    minimax.make_move = minimax_make_move;

    return minimax;
}

/**
 * Random-move player.
```

```
 */
const char *random_name( void )
{
    srand( (unsigned) time( NULL ) );
    return "Random";
}

int random_move( Board *b )
{
    int move;

    do {
        move = rand() % 14;
    } while ( !board_legal_move( b, move ) );

    return move;
}

KalahPlayer random_player( void )
{
    KalahPlayer random;

    random.get_name = random_name;
    random.make_move = random_move;

    return random;
}
```

kalah/simple_players.h

```
#ifndef _SIMPLE_PLAYERS_H
#define _SIMPLE_PLAYERS_H

#include "kalah_player.h"

KalahPlayer human_player( void );
KalahPlayer bonzo_player( void );
KalahPlayer random_player( void );

MinimaxResult minimax_move( Board *b, int depth );

KalahPlayer minimax_player( void );
KalahPlayer stackless_minimax_player( void );

#endif
```

kalah/tree_array.c

```
#include "tree_array.h"

#ifndef _OPENCL_
#include <math.h>
#endif

/**
 * Returns the index of the first child node of
 *  a given node.
 *
 * @param ary
 *  How many children each node in the tree has.
 * @param depth
 *  How many levels the tree has.
 */
int tree_array_first_child( int ary, int node )
```

```
{
    return ( node * ary ) + 1;
}

/**
 * Returns the index of the parent node of a
 *  given node.
 *
 * @param ary
 *  How many children each node in the tree has.
 * @param depth
 *  How many levels the tree has.
 */
int tree_array_parent( int ary, int node )
{
    // Integer division rounds down.
    return ( node - 1 ) / ary;
}

/**
 * Finds the total number of nodes in a complete n-ary
 *  tree.
 *  Due to a theorem (https://ece.uwaterloo.ca/~dwharder/aads/LectureMaterials/4.07.
     NaryTrees.pdf),
 *    there are ( N^(k+1) - 1 ) / ( N - 1 ) nodes in a complete N-ary tree.
 *
 * @param ary
 *  How many children each node in the tree has.
 * @param depth
 *  How many levels the tree has.
*/
int tree_array_size( int ary, int depth )
{
    return
        ( (int) pow( (float) ary, depth + 1 ) - 1 )
        / ( ary - 1 );
}
```

kalah/tree_array.h

```
#ifndef _TREE_ARRAY_H
#define _TREE_ARRAY_H

/*
 * Helper functions for storing n-ary trees in arrays.
 *
 * In general, for an n-ary tree, the children of the node
 *  with index k are stored in indices n*k through n*k + (n - 1).
 *
 * @author John Kloosterman
 * @date December 2012
 */

int tree_array_first_child( int ary, int node );
int tree_array_parent( int ary, int node );
int tree_array_size( int ary, int depth );

#endif
```

kalah/types.h

```
#ifndef _TYPES_H
#define _TYPES_H
```

```
/*
 * C/C++ to OpenCL C interoperability types.
 *
 * All structures that can be passed between the host
 *  and OpenCL code *must* use these types.
 */

#ifndef FALSE
#define FALSE 0
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifdef HOST
#define CL_INT cl_int
#define CL_CHAR cl_int
#else /* OpenCL */
#define CL_INT int
#define CL_CHAR char
#endif

// This is only used on the host side, so it's OK to
//  use regular types.
typedef struct
{
    int move;
    int score;
} MinimaxResult;

#endif
```

A.3 local_malloc

local_malloc/AllocationAST.cpp

```
#include "AllocationAST.h"

#include <iostream>
#include <string>

using namespace clang;
using namespace std;

bool AllocationASTVisitor::VisitStmt( Stmt *s )
{
    if ( isa<CallExpr>(s) )
    {
        CallExpr *call = cast<CallExpr>(s);

        FunctionDecl *callee = call->getDirectCallee();
        DeclarationName DeclName = callee->getNameInfo().getName();
        string FuncName = DeclName.getAsString();

        if ( FuncName == "local_malloc" || FuncName == "local_free" )
        {
            if ( call->getNumArgs() == 1 )
            {
                Expr *size_arg = call->getArg( 0 );
                llvm::APSInt value;

                if ( !size_arg->EvaluateAsInt( value, myASTContext ) )
                {
```

```
                        cout << "Malloc argument not something foldable to an int!" <<
                            endl;
                        exit( 1 );
                    }

                if ( FuncName == "local_malloc" )
                    {
                        cout << "*** Malloc of size " << value.toString(10) << endl;
                        myCallGraph.malloc( (int)value.getLimitedValue() );
                    }
                    else
                    {
                        cout << "*** Free of size " << value.toString(10) << endl;
                        myCallGraph.free( (int)value.getLimitedValue() );
                    }
                }
                else
                {
                    cout << "Malloc/free: wrong # of arguments." << endl;
                }
            }
            else
            {
                cout << "*** Calling function " << FuncName << endl;
                myCallGraph.call( FuncName );
            }
        }

        return true;
    }

bool AllocationASTVisitor::VisitFunctionDecl( FunctionDecl *f )
{
    // Only function definitions (with bodies), not declarations.
    if ( f->hasBody() )
    {
        Stmt *FuncBody = f->getBody();

        // Type name as string
        QualType QT = f->getResultType();
            string TypeStr = QT.getAsString();

            // Function name
            DeclarationName DeclName = f->getNameInfo().getName();
            string FuncName = DeclName.getAsString();
            myCallGraph.enter_function( FuncName );

//          cout << "Begin function " << FuncName << endl;
        }

    return true;
}

bool AllocationASTConsumer::HandleTopLevelDecl( DeclGroupRef DR )
{
    for (DeclGroupRef::iterator b = DR.begin(), e = DR.end();
         b != e; ++b)
        // Traverse the declaration using our AST visitor.
        Visitor.TraverseDecl(*b);
    return true;
}
```

local_malloc/AllocationAST.h

```
/**
 * Classes to traverse a Clang AST that track how
 *  much memory is local_malloc()ed and local_free()d
 *  at one time.
 *
 * Based on code from Eli Bendersky (eliben@gmail.com)
 *  at http://eli.thegreenplace.net/2012/06/08/basic-source-to-source-transformation-
     with-clang/
 *
 * @author John Kloosterman
 * @date March 23, 2013
 */

#ifndef _ALLOCATION_AST_
#define _ALLOCATION_AST_

#include "clang/AST/ASTConsumer.h"
#include "clang/AST/RecursiveASTVisitor.h"
#include "clang/Rewrite/Core/Rewriter.h"

#include "CallGraph.h"

class AllocationASTVisitor : public clang::RecursiveASTVisitor<AllocationASTVisitor>
{
public:
    AllocationASTVisitor(
        clang::Rewriter &R,
        CallGraph &G,
        clang::ASTContext &context )
        : TheRewriter(R),
        myCallGraph( G ),
        myASTContext( context )
        {}

    bool VisitStmt( clang::Stmt *s);
    bool VisitFunctionDecl( clang::FunctionDecl *f);

private:
    clang::Rewriter &TheRewriter;
    CallGraph &myCallGraph;
    clang::ASTContext &myASTContext;
};


// Implementation of the ASTConsumer interface for reading an AST produced
// by the Clang parser.
class AllocationASTConsumer : public clang::ASTConsumer
{
public:
    AllocationASTConsumer( clang::Rewriter &R, CallGraph &G, clang::ASTContext &context
         )
        : Visitor( R, G, context )
        {}

    // Override the method that gets called for each parsed top-level
    // declaration.
    virtual bool HandleTopLevelDecl( clang::DeclGroupRef DR );

private:
    AllocationASTVisitor Visitor;
};

#endif
```

local_malloc/CallGraph.cpp

```
// Mar. 22, 2013

#include "CallGraph.h"
#include <iostream>
using namespace std;

void CallGraph::enter_function( string name )
{
    CallGraphFunction *function = new CallGraphFunction;
    myFunctions.push_back( function );
    myFunctionMap.insert( make_pair<string,CallGraphFunction*>( name, function ) );
    myCurrentFunction = function;
}

void CallGraph::malloc( int size )
{
    myCurrentFunction->actions.push_back(
        new CallGraphActionMalloc( size )
        );
}

void CallGraph::free( int size )
{
    myCurrentFunction->actions.push_back(
        new CallGraphActionFree( size )
        );
}

void CallGraph::call( string name )
{
    myCurrentFunction->actions.push_back(
        new CallGraphActionCall( name, *this )
        );
}

int CallGraph::maximum_alloc( string start_function )
{
    CallGraphFunction *function = myFunctionMap[start_function];

    if ( function == NULL )
    {
        cout << "Ignoring unknown function " << start_function << endl;
        return 0;
    }

    int current_allocation = 0;
    int peak = 0;

    for ( int i = 0; i < function->actions.size(); i++ )
    {
        CallGraphAction *action = function->actions[i];
        int action_usage = action->call();

        current_allocation += action_usage;
        if ( current_allocation > peak )
            peak = current_allocation;
    }

    return peak;
}

bool CallGraph::isDefined( string name )
```

```
{
    bool def = myFunctionMap[name] != NULL;
//  cout << "* " << name << " defined: " << def << endl;

    return def;
}
```

local_malloc/CallGraph.h

```
#ifndef _CALL_GRAPH_H
#define _CALL_GRAPH_H

#include <string>
#include <vector>
#include <map>

class CallGraph;

/*********/
class CallGraphAction {
public:
    virtual int call() = 0;
};

/**********/
class CallGraphFunction {
public:
    std::vector<CallGraphAction *> actions;
};

/*************/
class CallGraph {
public:
    void enter_function( std::string name );
    void malloc( int size );
    void free( int size );
    void call( std::string name );

    bool isDefined( std::string name );
    int maximum_alloc( std::string start_function );

private:
    CallGraphFunction *myCurrentFunction;
    std::vector<CallGraphFunction *> myFunctions;
    std::map<std::string, CallGraphFunction *> myFunctionMap;
};

/**************/
class CallGraphActionMalloc : public CallGraphAction {
public:
    CallGraphActionMalloc( int size )
    : mySize( size )
        {
        };

    virtual int call()
        {
            return mySize;
        }
private:
    int mySize;
};

class CallGraphActionFree : public CallGraphAction {
```

```
public:
    CallGraphActionFree( int size )
    : mySize( size )
        {
        };

    virtual int call()
        {
            return -mySize;
        }
private:
    int mySize;
};

/**********/
class CallGraphActionCall : public CallGraphAction {
public:
    CallGraphActionCall( std::string name, CallGraph &callGraph )
        : myName( name ),
          myCallGraph( callGraph )
        {
        };

    virtual int call()
        {
            return myCallGraph.maximum_alloc( myName );
        }

private:
    std::string myName;
    CallGraph &myCallGraph;
};


#endif
```

local_malloc/ClangInterface.cpp

```
#include "ClangInterface.h"

//------------------------------------------------------------------------------
//
// rewritersample.cpp: Source-to-source transformation sample with Clang,
// using Rewriter - the code rewriting interface.
//
// Eli Bendersky (eliben@gmail.com)
// This code is in the public domain
//
#include <cstdio>
#include <iostream>
#include <string>
#include <sstream>
#include <cassert>

#define LIBCLC_INCLUDE_PATH "/home/john/senior-project/local_malloc/libclc/generic/
    include"

#include "clang/AST/ASTConsumer.h"
#include "clang/AST/RecursiveASTVisitor.h"
#include "clang/Basic/Diagnostic.h"
#include "clang/Basic/FileManager.h"
#include "clang/Basic/SourceManager.h"
#include "clang/Basic/TargetOptions.h"
#include "clang/Basic/TargetInfo.h"
```

```
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Lex/Preprocessor.h"
#include "clang/Parse/ParseAST.h"
#include "clang/Rewrite/Core/Rewriter.h"
#include "clang/Rewrite/Frontend/Rewriters.h"
#include "llvm/Support/Host.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/ADT/APSInt.h"

#include "CallGraph.h"
#include "AllocationAST.h"

using namespace clang;
using namespace std;

ClangInterface::ClangInterface( string fileName )
{
    TO = new TargetOptions();
    invocation = new CompilerInvocation();

    cout << "File: " << fileName << endl;

    myCompilerInstance.createDiagnostics(0, 0);

    // Target info
    TO->Triple = llvm::sys::getDefaultTargetTriple();
    TargetInfo *TI = TargetInfo::CreateTargetInfo(
        myCompilerInstance.getDiagnostics(), *TO );
    myCompilerInstance.setTarget( TI );

    // Invocation, including all the headers necessary for
    //  OpenCL support.
    const char * const options[] = {
        "-x", "cl",
        "-I", "/usr/include/clang/3.0/include",
        "-D", "__LOCAL_MALLOC_ANALYSIS__",
        "-I", LIBCLC_INCLUDE_PATH,
        "-include", "clc/clc.h",
        };
    CompilerInvocation::CreateFromArgs(
        *invocation,
        options,
        options + 10,
        myCompilerInstance.getDiagnostics()
        );
    myCompilerInstance.setInvocation( invocation );

    // Files
    myCompilerInstance.createFileManager();
    FileManager &FileMgr = myCompilerInstance.getFileManager();
    myCompilerInstance.createSourceManager(FileMgr);
    SourceManager &SourceMgr = myCompilerInstance.getSourceManager();
    myCompilerInstance.createPreprocessor();
    myCompilerInstance.createASTContext();

    // A Rewriter helps us manage the code rewriting task.
    myRewriter.setSourceMgr(SourceMgr, myCompilerInstance.getLangOpts());

    // Set the main file handled by the source manager to the input file.
    const FileEntry *FileIn = FileMgr.getFile( fileName );
    SourceMgr.createMainFileID(FileIn);
    myCompilerInstance.getDiagnosticClient().BeginSourceFile(
        myCompilerInstance.getLangOpts(),
        &myCompilerInstance.getPreprocessor());
```

```
}

ClangInterface::~ClangInterface()
{
    // This will crash. But any order of the objects in
    //  the class also crashes. Does Clang have a way to
    //  deallocate the objects in the proper order?

//  delete invocation;
//  delete TO;
}

void ClangInterface::processAST( ASTConsumer *astConsumer )
{
   ParseAST(
       myCompilerInstance.getPreprocessor(),
       astConsumer,
       myCompilerInstance.getASTContext()
       );
}

string ClangInterface::getRewrittenCode()
{
    SourceManager &SourceMgr = myRewriter.getSourceMgr();

    const RewriteBuffer *rewriteBuf =
        myRewriter.getRewriteBufferFor( SourceMgr.getMainFileID() );

    if ( !rewriteBuf )
    {
        const RewriteBuffer &cleanBuf = myRewriter.getEditBuffer( SourceMgr.
            getMainFileID() );
        return string(cleanBuf.begin(), cleanBuf.end());
    }

    return string( rewriteBuf->begin(), rewriteBuf->end() );
}
```

local_malloc/ClangInterface.h

```
#ifndef _CLANG_INTERFACE_H
#define _CLANG_INTERFACE_H

#include <string>

#include "clang/Frontend/CompilerInstance.h"
#include "clang/Rewrite/Core/Rewriter.h"

class ClangInterface {
public:
    ClangInterface( std::string fileName );
    ~ClangInterface();

    void processAST( clang::ASTConsumer *astConsumer );
    std::string getRewrittenCode();

    clang::CompilerInstance &getCompilerInstance() {
        return myCompilerInstance;
    }
    clang::Rewriter &getRewriter() {
        return myRewriter;
    }
    clang::ASTContext &getASTContext() {
        return myCompilerInstance.getASTContext();
```

```
    }

private:
    clang::Rewriter myRewriter;
    clang::CompilerInvocation *invocation;
    clang::TargetOptions *TO;
    clang::CompilerInstance myCompilerInstance;
};

#endif
```

local_malloc/embed_src.sh

```
# Quick and dirty tool to embed a file as C++ source code.
#
# @author John Kloosterman
# @date April 13, 2013
#
# Usage:
#  embed_src.sh <variable name> <file to embed>
#
# C++ source is outputted to standard output.
#

echo "#include <string>"
echo "std::string $2 = "
sed "s/\\\/\\\\\\\\/g" $1 | sed 's/"/\\"/g' | sed -e 's/.*/\"&\\n\"/'
echo ";"
```

local_malloc/example/example.cl

```
/**
 * A simple OpenCL kernel that uses local_malloc()
 *  to allocate workgroup scratch memory.
 *
 * @author John Kloosterman
 * @date April 10, 2013
 */

#define SIZEOF_INT 4
#define NUM_THREADS 256

int function( int i )
{
    int ret;

    __local int *useless_ptr = local_malloc( SIZEOF_INT );
    *useless_ptr = i * 2;
    ret = *useless_ptr;
    local_free( SIZEOF_INT );

    return ret;
}

__kernel
void
function_sum(
    __global int *range_start,
    __global int *sum
    )
{
    size_t num_threads = get_local_size( 0 );
    size_t local_id = get_local_id( 0 );
    __local int local_sum;
```

```
    // Initialize the sum.
    if ( local_id == 0 )
        local_sum = 0;
    barrier( CLK_LOCAL_MEM_FENCE );

    // Allocate scratch __local memory.
    __local int *values = local_malloc( NUM_THREADS * SIZEOF_INT );

    // Run the function at our thread's location.
    values[local_id] = function( *range_start + local_id );

    // Find the sum using atomics.
    atomic_add( &local_sum, values[local_id] );

    // Free scratch memory.
    local_free( NUM_THREADS * SIZEOF_INT );

    // Copy the result to global memory.
    if ( local_id == 0 )
        *sum = local_sum;
}
```

local_malloc/example/example.cpp

```
#include <LocalMallocRewriter.h>
#include <CLKernel.h>
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main ( int argc, char *argv[] )
{
    // Read kernel source from file
    ifstream t( "example.cl" );
    string src( (std::istreambuf_iterator<char>(t) ),
                std::istreambuf_iterator<char>() );

    // Rewrite the code t make local_malloc work.
    LocalMallocRewriter rewriter( src );
    string rewritten_src = rewriter.rewrite( "function_sum" );

    cout << "Rewritten source code: " << endl;
    cout << rewritten_src << endl;

    // Run the kernel.
    CLKernel function_sum( "function_sum", rewritten_src );
    cl_int range_start = 20;
    cl_int host_sum;
    CLArgument sum( &host_sum, 1 );

    function_sum.setGlobalDimensions( 256 );
    function_sum.setLocalDimensions( 256 );

    vector<CLArgument> args;
    args.push_back( range_start );
    args.push_back( sum );
    function_sum( args );

    cout << "Sum was " << host_sum << endl;

    return 0;
}
```

local_malloc/example/Makefile

```
OPENCL_INCLUDE = /opt/AMDAPP/include
OPENCL_LIBS = /opt/AMDAPP/lib/x86_64
CL_TEST = ../../cl_test

example: example.cpp example.cl
    clang++ example.cpp -o example \
        -I .. -I $(CL_TEST) -I $(OPENCL_INCLUDE) \
        -L .. -L $(CL_TEST) -L $(OPENCL_LIBS) \
        -lLocalMallocRewriter -lCLTest -lOpenCL

# example.cl: example.cl.in
#   cpp -I .. example.cl.in > example.cl

clean:
    rm -f example
```

local_malloc/example/README

```
An example program that uses a LocalMallocRewriter on a kernel.

You will need to set LD_LIBRARY_PATH to the folder containing libLocalMallocRewriter.so
    .
```

local_malloc/functions.cpp

```
//------------------------------------------------------------------------------
//
// rewritersample.cpp: Source-to-source transformation sample with Clang,
// using Rewriter - the code rewriting interface.
//
// Eli Bendersky (eliben@gmail.com)
// This code is in the public domain
//
#include <cstdio>
#include <iostream>
#include <string>
#include <sstream>

#include "clang/AST/ASTConsumer.h"
#include "clang/AST/RecursiveASTVisitor.h"
#include "clang/Basic/Diagnostic.h"
#include "clang/Basic/FileManager.h"
#include "clang/Basic/SourceManager.h"
#include "clang/Basic/TargetOptions.h"
#include "clang/Basic/TargetInfo.h"
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Lex/Preprocessor.h"
#include "clang/Parse/ParseAST.h"
#include "clang/Rewrite/Core/Rewriter.h"
#include "clang/Rewrite/Frontend/Rewriters.h"
#include "llvm/Support/Host.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/ADT/APSInt.h"

#include "CallGraph.h"
#include "AllocationAST.h"

using namespace clang;
using namespace std;


    // CompilerInstance will hold the instance of the Clang compiler for us,
```

```
// managing the various objects needed to run the compiler.
CompilerInstance TheCompInst;
TheCompInst.createDiagnostics(0, 0);

CallGraph TheCallGraph;

// Initialize target info with the default triple for our platform.
TargetOptions TO;
TO.Triple = llvm::sys::getDefaultTargetTriple();
TargetInfo *TI = TargetInfo::CreateTargetInfo(
    TheCompInst.getDiagnostics(), TO);
TheCompInst.setTarget(TI);

CompilerInvocation invocation;
const char * const options[] = {
    "-x", "cl",
    "-I", "libclc/generic/include",
    "-include", "clc/clc.h",
    "-I", "/usr/include/clang/3.0/include"
};
CompilerInvocation::CreateFromArgs(
    invocation,
    options,
    options + 8,
    TheCompInst.getDiagnostics()
    );
TheCompInst.setInvocation( &invocation );

TheCompInst.createFileManager();
FileManager &FileMgr = TheCompInst.getFileManager();
TheCompInst.createSourceManager(FileMgr);
SourceManager &SourceMgr = TheCompInst.getSourceManager();
TheCompInst.createPreprocessor();
TheCompInst.createASTContext();

// A Rewriter helps us manage the code rewriting task.
Rewriter TheRewriter;
TheRewriter.setSourceMgr(SourceMgr, TheCompInst.getLangOpts());

// Set the main file handled by the source manager to the input file.
const FileEntry *FileIn = FileMgr.getFile(argv[1]);
SourceMgr.createMainFileID(FileIn);
TheCompInst.getDiagnosticClient().BeginSourceFile(
    TheCompInst.getLangOpts(),
    &TheCompInst.getPreprocessor());

// Create an AST consumer instance which is going to get called by
// ParseAST.
AllocationASTConsumer TheConsumer(
    TheRewriter,
    TheCallGraph,
    TheCompInst.getASTContext()
    );

// Parse the file to AST, registering our consumer as the AST consumer.
ParseAST(
    TheCompInst.getPreprocessor(),
    &TheConsumer,
    TheCompInst.getASTContext()
    );

// At this point the rewriter's buffer should be full with the rewritten
// file contents.
const RewriteBuffer *RewriteBuf =
```

```
        TheRewriter.getRewriteBufferFor(SourceMgr.getMainFileID());
//  llvm::outs() << string(RewriteBuf->begin(), RewriteBuf->end());

    int max_alloc = TheCallGraph.maximum_alloc( "entry" );
    cout << "Maximum allocation: " << max_alloc << endl;

    exit( 0 );
    return 0;
}
```

local_malloc/local_malloc.h

```
/**
 * local_malloc: Allow for allocation of workgroup-local
 *  scratch memory.
 *
 * @author John Kloosterman
 * @date March 23, 2012
 */

#ifndef _LOCAL_MALLOC_H
#define _LOCAL_MALLOC_H

#ifdef __LOCAL_MALLOC_ANALYSIS__
/* Define different prototypes when not actually compiling. */

__local void *local_malloc( size_t size );
__local void *local_free( size_t size );

#else

typedef struct {
    size_t offset;
    size_t max_size;
    __local char *buffer;
} LocalMallocState;

void local_malloc_init(
    __local char *buffer,
    size_t max_size,
    LocalMallocState *state )
{
    state->offset = 0;
    state->max_size = max_size;
    state->buffer = buffer;
}

/**
 * This needs to be callable by all threads, because
 *  a pointer of the form
 *    __local int *ptr;
 *  is stored in private memory, not local memory.
 */
__local void *local_malloc( size_t size, LocalMallocState *state )
{
    state->offset += size;

    if ( state->offset > state->max_size )
    {
        // Is there a way to give better diagnostic information?
//      printf( "local_malloc: buffer overflow.\n" );
        return (__local void *) 0;
    }
```

```
    return state->buffer + state->offset;
}

void local_free( size_t size, LocalMallocState *state )
{
    state->offset -= size;
}

#endif

#endif
```

local_malloc/LocalMallocRewriter.cpp

```
#include "LocalMallocRewriter.h"
#include "ClangInterface.h"
#include "CallGraph.h"
#include "AllocationAST.h"
#include "RewriterAST.h"
#include <cstdio>
#include <unistd.h>
#include <iostream>
#include <fstream>
using namespace std;

// The source code for the local_malloc header.
extern string local_malloc_header;

LocalMallocRewriter::LocalMallocRewriter( string src )
    : mySrc( src )
{
    // Put the source code in a temp file.
    myTempFileName = strdup( "/tmp/tmpfileXXXXXX" );
    mkstemp( myTempFileName );
    ofstream temp_file( myTempFileName );
    temp_file << local_malloc_header << mySrc;
    temp_file.close();
}

LocalMallocRewriter::~LocalMallocRewriter()
{
    if ( unlink( myTempFileName ) != 0 )
    {
        perror( "Could not delete temporary file." );
    }
    free( myTempFileName );
}

string LocalMallocRewriter::rewrite( string entry )
{
    ClangInterface clangInterface( myTempFileName );

    // Find the maximum allocation
    CallGraph callGraph;
    AllocationASTConsumer allocationConsumer(
        clangInterface.getRewriter(),
        callGraph,
        clangInterface.getASTContext()
        );
    clangInterface.processAST( &allocationConsumer );

    int max_alloc = callGraph.maximum_alloc( entry );
    cout << "Maximum allocation: " << max_alloc << endl;
```

```
    ClangInterface clangInterface2( myTempFileName );
    RewriterASTConsumer rewriterConsumer(
        clangInterface2.getRewriter(),
        callGraph,
        clangInterface2.getASTContext(),
        entry,
        max_alloc
        );
    clangInterface2.processAST( &rewriterConsumer );

    // At this point the rewriter's buffer should be full with the rewritten
    // file contents.
    return clangInterface2.getRewrittenCode();

    // Inject the local_malloc code!
}
```

local_malloc/LocalMallocRewriter.h

```
/**
 * LocalMallocAnalyzer takes OpenCL source code
 *  in memory and makes the local_malloc code
 *  word properly.
 *
 * @author John Kloosterman
 * @date March 23, 2013
 */

#include <string>

class LocalMallocRewriter {
public:
    LocalMallocRewriter( std::string src );
    ~LocalMallocRewriter();

    /**
     * Get rewritten source code.
     *
     * @param entry
     *  The name of the kernel to process.
     * @return
     *  Rewritten source code.
     */
    std::string rewrite( std::string entry);

private:
    std::string mySrc;
    char *myTempFileName;
};
```

local_malloc/main.cpp

```
#include "LocalMallocRewriter.h"
#include <iostream>
#include <fstream>
#include <cassert>
#include <cstdlib>
using namespace std;

int main ( int argc, char *argv[] )
{
    assert( argc == 2 );

    ifstream t( argv[1] );
```

```
    string src( (std::istreambuf_iterator<char>(t) ),
                std::istreambuf_iterator<char>() );

    LocalMallocRewriter rewriter( src );
    cout << rewriter.rewrite( "entry" );

    exit( 0 );
    // There is a bug with the destructor. Solve it later.
    return 0;
}
```

local_malloc/Makefile

```
CXX = clang++ -g
CFLAGS = -fno-rtti

LLVM_SRC_PATH = /home/john/clang-dev/llvm-3.2.src
LLVM_BUILD_PATH = /home/john/clang-dev/llvm-build

LLVM_BIN_PATH = $(LLVM_BUILD_PATH)/Debug+Asserts/bin
LLVM_LIBS=all
LLVM_CONFIG_INCLUDES_COMMAND = $(LLVM_BIN_PATH)/llvm-config --cxxflags
LLVM_CONFIG_LIBS_COMMAND = $(LLVM_BIN_PATH)/llvm-config --ldflags --libs $(LLVM_LIBS)
CLANG_BUILD_FLAGS = -I$(LLVM_SRC_PATH)/tools/clang/include -I$(LLVM_BUILD_PATH)/tools/
    clang/include

CLANGLIBS = \
  -lclangFrontendTool -lclangFrontend -lclangDriver \
  -lclangSerialization -lclangCodeGen -lclangParse \
  -lclangSema -lclangStaticAnalyzerFrontend \
  -lclangStaticAnalyzerCheckers -lclangStaticAnalyzerCore \
  -lclangAnalysis -lclangARCMigrate -lclangRewriteCore -lclangRewriteFrontend \
  -lclangEdit -lclangAST -lclangLex -lclangBasic -lclangTooling

OBJS = CallGraph.o AllocationAST.o RewriterAST.o ClangInterface.o LocalMallocRewriter.o
     MallocHeader.o

all: rewriter libLocalMallocRewriter.so

rewriter: $(OBJS) main.o
    $(CXX) $(OBJS) main.o \
        $(CFLAGS) -o rewriter \
        $(CLANG_BUILD_FLAGS) $(CLANGLIBS) `$(LLVM_CONFIG_LIBS_COMMAND)` -ldl

libLocalMallocRewriter.so: $(OBJS)
    g++ -Wall -shared $(OBJS) $(CFLAGS) -o libLocalMallocRewriter.so \
        $(CLANG_BUILD_FLAGS) $(CLANGLIBS) `$(LLVM_CONFIG_LIBS_COMMAND)` -ldl

MallocHeader.cpp: local_malloc.h
    ./embed_src.sh local_malloc.h local_malloc_header > MallocHeader.cpp

%.o: %.cpp *.h
    clang++ $(CFLAGS) $(CLANG_BUILD_FLAGS) `$(LLVM_CONFIG_INCLUDES_COMMAND)` -g -c $< -
        o $@

clean:
    rm -rf *.o rewriter libLocalMallocRewriter.so MallocHeader.cpp
```

local_malloc/README

```
Building this library:
======================
You will need a copy of LLVM and Clang. I compiled against LLVM and Clang 3.2; other
    versions
```

```
might have a slightly different API and require modifications to this library.

In the Makefile, set LLVM_SRC_PATH and LLVM_BUILD_PATH to the correct locations for
    your
LLVM build. Depending on the flags you gave when configuring LLVM, you may need to
adjust LLVM_BIN_PATH to Release+Asserts instead of Debug+Asserts.

In ClangInterface.cpp, there is a constant, LIBCLC_INCLUDE_PATH, that needs to point
to the location of libclc if you want fewer compiler warnings/errors.

The public interface to the library is the LocalMallocRewriter class. See the example
 in the examples/ directory for usage.
```

local_malloc/RewriterAST.cpp

```
#include "RewriterAST.h"

#include <iostream>
#include <string>
#include <sstream>

#include "clang/Lex/Preprocessor.h"

using namespace clang;
using namespace std;

bool RewriterASTVisitor::VisitStmt( Stmt *s )
{
    if ( isa<CallExpr>(s) )
    {
        CallExpr *call = cast<CallExpr>(s);

        FunctionDecl *callee = call->getDirectCallee();
        DeclarationName DeclName = callee->getNameInfo().getName();
        string funcName = DeclName.getAsString();

        if ( funcName == "local_malloc"
             || funcName == "local_free"
             || myCallGraph.isDefined( funcName ) )

        {
            // For all calls to local_malloc and local_free, add the
            //  local_malloc parameter.

            // For all functions that had bodies in last AST build that
            //  were not kernels, add a parameter for the local_malloc
            //  object.

            unsigned numArgs = call->getNumArgs();
            string add_param_string( "__local_malloc_state" );

            SourceLocation arg_insert_location;
            if ( numArgs == 0 )
            {
                arg_insert_location = call->getRParenLoc();
            }
            else
            {
                Expr **args = call->getArgs();
                SourceLocation last_arg = args[numArgs - 1]->getSourceRange().getEnd();

                SourceManager &sourceManager = TheRewriter.getSourceMgr();
                const LangOptions &langOpts = TheRewriter.getLangOpts();
                arg_insert_location = clang::Lexer::getLocForEndOfToken( last_arg, 0,
```

```
                    sourceManager, langOpts );

                add_param_string = ", " + add_param_string;
            }

            TheRewriter.InsertText( arg_insert_location, add_param_string, true, true )
                ;
        }
        else
        {
            // For OpenCL built-in functions, don't modify anything
            cout << "call to builtin function: " << funcName << endl;
        }
    }

    return true;
}

bool RewriterASTVisitor::isOpenCLKernel( FunctionDecl *f )
{
    if ( !f->hasAttrs() )
        return false;

    AttrVec &attributes = f->getAttrs();
    PrintingPolicy policy( TheRewriter.getLangOpts() );
    policy.Indentation = 0;

    for (
        AttrVec::iterator it = attributes.begin();
        it != attributes.end();
        it++
        )
    {
        string attribute_name;
        llvm::raw_string_ostream attribute_name_stream( attribute_name );
        (*it)->printPretty( attribute_name_stream, policy );

        // Flushes the stream to the string.
        attribute_name_stream.str();

        // This is how clang mungs up __kernel.
        if ( attribute_name == " __attribute__((opencl_kernel_function))" )
        {
            return true;
        }
    }

    return false;
}

bool RewriterASTVisitor::VisitFunctionDecl( FunctionDecl *f )
{
    DeclarationName DeclName = f->getNameInfo().getName();
    string funcName = DeclName.getAsString();

    if ( funcName == "local_malloc"
         || funcName == "local_free" )
    {
        // For all calls to local_malloc and local_free, add the
        //  local_malloc parameter.

        cout << "malloc function!" << endl;
    }
    else if ( funcName == myEntryFunction )
```

```
    {
        if ( f->hasBody() )
        {
            // Put in the prelude.
            stringstream prelude;
            prelude << "__local char __local_malloc_buffer[" << myBufferSize << "];" <<
                 endl;
            prelude << "LocalMallocState __local_malloc_state_backing;" << endl;
            prelude << "LocalMallocState *__local_malloc_state = &
                __local_malloc_state_backing;" << endl;
            prelude << "local_malloc_init( __local_malloc_buffer, " << myBufferSize <<
                ", __local_malloc_state );" << endl;

            Stmt *FuncBody = f->getBody();
            SourceLocation ST = FuncBody->getSourceRange().getBegin();
            TheRewriter.InsertText(ST.getLocWithOffset(1), prelude.str(), true, true);
        }
    }
    else if ( myCallGraph.isDefined( funcName ) )
    {
        // For all functions that had bodies in last AST build that
        //  were not kernels, add a parameter for the local_malloc
        //  object.

        cout << "****** Real function! ******" << endl;

        if ( isOpenCLKernel( f ) )
        {
            cout << "Is an openCL kernel." << endl;
            return true;
        }
        else
        {
            if ( f->getNumParams() == 0 )
            {
                cout << "Has void as parameter. " << endl;

                SourceLocation currentLocation = f->getSourceRange().getBegin();
                SourceManager &sourceManager = TheRewriter.getSourceMgr();

                SourceLocation openParen, closeParen;

                // Eat characters until we get to an open paren.
                while ( true )
                {
                    SourceRange currentRange( currentLocation, currentLocation.
                        getLocWithOffset( 1 ) );
                    string character = TheRewriter.getRewrittenText( currentRange );

                    if ( character == "(" )
                        break;

//                  cout << "Eating character " << character << endl;
                    currentLocation = currentLocation.getLocWithOffset( 1 );
                }
                openParen = currentLocation.getLocWithOffset( 1 );

                while ( true )
                {
                    SourceRange currentRange( currentLocation, currentLocation.
                        getLocWithOffset( 1 ) );
                    string character = TheRewriter.getRewrittenText( currentRange );

                    if ( character == ")" )
```

```
                        break;

//                  cout << "Eating character " << character << endl;
                    currentLocation = currentLocation.getLocWithOffset( 1 );
                }
                closeParen = currentLocation.getLocWithOffset( -1 );

                SourceRange betweenParensRange( openParen, closeParen );
                TheRewriter.ReplaceText( betweenParensRange, "LocalMallocState *
                    __local_malloc_state" );

                return true;
            }

            ParmVarDecl *last_param = f->getParamDecl( f->getNumParams() - 1 );
            // The locWithOffset brings us after the last param.
            SourceManager &sourceManager = TheRewriter.getSourceMgr();
            const LangOptions &langOpts = TheRewriter.getLangOpts();
            SourceLocation lastParam = last_param->getSourceRange().getEnd();
            SourceLocation real_end = clang::Lexer::getLocForEndOfToken( lastParam, 0,
                sourceManager, langOpts );
            TheRewriter.InsertText( real_end, ", LocalMallocState *__local_malloc_state
                ", true, true );

            cout << "defined function: " << funcName << endl;
        }
    }
    else
    {
//      cout << "builtin function: " << funcName << endl;
    }

    return true;
}

bool RewriterASTConsumer::HandleTopLevelDecl( DeclGroupRef DR )
{
    for (DeclGroupRef::iterator b = DR.begin(), e = DR.end();
         b != e; ++b)
        // Traverse the declaration using our AST visitor.
        Visitor.TraverseDecl(*b);
    return true;
}
```

local_malloc/RewriterAST.h

```
/**
 *
 * @author John Kloosterman
 * @date March 28, 2013
 */

#ifndef _REWRITER_AST_
#define _REWRITER_AST_

#include "clang/AST/ASTConsumer.h"
#include "clang/AST/RecursiveASTVisitor.h"
#include "clang/Rewrite/Core/Rewriter.h"

#include "CallGraph.h"

class RewriterASTVisitor : public clang::RecursiveASTVisitor<RewriterASTVisitor>
{
public:
```

```
    RewriterASTVisitor(
        clang::Rewriter &R,
        CallGraph &G,
        clang::ASTContext &context,
        std::string entryFunction,
        unsigned bufferSize )
        : TheRewriter(R),
        myCallGraph( G ),
        myASTContext( context ),
        myEntryFunction( entryFunction ),
        myBufferSize( bufferSize )
        {}

    bool VisitStmt( clang::Stmt *s);
    bool VisitFunctionDecl( clang::FunctionDecl *f);

private:
    bool isOpenCLKernel( clang::FunctionDecl *f );

    clang::Rewriter &TheRewriter;
    CallGraph &myCallGraph;
    clang::ASTContext &myASTContext;
    std::string myEntryFunction;
    unsigned myBufferSize;
};


// Implementation of the ASTConsumer interface for reading an AST produced
// by the Clang parser.
class RewriterASTConsumer : public clang::ASTConsumer
{
public:
    RewriterASTConsumer(
        clang::Rewriter &R,
        CallGraph &G,
        clang::ASTContext &context,
        std::string entryFunction,
        unsigned bufferSize )
        : Visitor( R, G, context, entryFunction, bufferSize )
        {}

    // Override the method that gets called for each parsed top-level
    // declaration.
    virtual bool HandleTopLevelDecl( clang::DeclGroupRef DR );

private:
    RewriterASTVisitor Visitor;
};

#endif
```

local_malloc/test0.cl

```
/*
 * test0: simple allocation and deallocation.
 */

#include "local_malloc.h"

#define LOCAL_MALLOC_SIZE 200
#define ALLOC_SIZE 40

__kernel
void entry( void )
```

```
{
    __local void *ptr;

    ptr = local_malloc( 40 );
    ptr = local_malloc( ALLOC_SIZE + 20 );
    local_free( 60 );
    local_free( ALLOC_SIZE );

    ptr = local_malloc( 80 );
    local_free( ALLOC_SIZE + 40 );
}
```

local_malloc/test1.cl

```
/*
 * test0: simple allocation and deallocation.
 */

#include "local_malloc.h"

#define LOCAL_MALLOC_SIZE 200
#define ALLOC_SIZE 40

__kernel
void entry( void )
{
    __local char buffer[LOCAL_MALLOC_SIZE];
    LocalMallocState state;
    __local void *ptr;

    local_malloc_init( buffer, LOCAL_MALLOC_SIZE, &state );

    for ( int i = 0; i < 20; i++ )
    {
        if ( i % 2 )
        {
            ptr = local_malloc( 20, &state );
            local_free( 20, &state );
        }
        else
        {
            ptr = local_malloc( 40, &state );
            local_free( 40, &state );
        }
    }
}
```

local_malloc/test2.cl

```
/*
 * test0: simple allocation and deallocation.
 */

#define LOCAL_MALLOC_SIZE 200
#define ALLOC_SIZE 40

void some_function( int i )
{
    __local void *ptr = local_malloc( 20 );
}

void another_function( void )
{
    __local void *ptr = local_malloc( 20 );
```

```
}

__kernel
void entry( void )
{
    int i = get_local_id( 0 );

    some_function( 10 );
    another_function();
}
```

A.4 raytracer

raytracer/CLRenderer.cpp

```
#include "CLRenderer.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

CLRenderer::CLRenderer(
    string src,
    unsigned char *pixel_buffer,
    size_t width,
    size_t height,
    bool use_cpu )
    : myPixelBuffer( pixel_buffer ),
      myWidth( width ),
      myHeight( height ),
      rayTrace( "raytrace", src ),
      pixbuf_arg( "uchar", pixel_buffer, width * height * 4, false, true )
{
    rayTrace.setGlobalDimensions( myWidth, myHeight );
}

void CLRenderer::render(
    Object *world,
    int num_objects,
    Light *lights,
    int num_lights,
    cl_float3 camera_position )
{
    size_t pixel_buffer_size = myWidth * myHeight * 4;

    CLArgument world_arg( "Object", world, num_objects, true, false );
    CLArgument lights_arg( "Light", lights, num_lights, true, false );

    vector<CLArgument> args;
    args.push_back( pixbuf_arg );
    args.push_back( world_arg );
    args.push_back( lights_arg );
    args.push_back( camera_position );
    args.push_back( num_objects );
    args.push_back( num_lights );

    rayTrace( args );
}
```

raytracer/CLRenderer.h

```
/**
 * Host interface for OpenCL raytracing renderer.
```

```
 *
 * @author John Kloosterman for CS352 at Calvin College
 * @date Dec. 12, 2012
 */

#ifndef _CL_RENDERER_H
#define _CL_RENDERER_H

#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>

#define _HOST_
#include "objects.h"

#include <vector>
#include <CLKernel.h>

class CLRenderer
{
public:
    CLRenderer( std::string src, unsigned char *pixel_buffer, size_t width, size_t
        height, bool use_cpu );
    void render( Object *world, int num_objects, Light *lights, int num_lights,
        cl_float3 camera_position );

private:
    CLKernel rayTrace;
    CLArgument pixbuf_arg;

    size_t myWidth, myHeight;
    cl_uchar *myPixelBuffer;
    std::string src;
};

#endif
```

raytracer/Makefile

```
CL_TEST = ../cl_test
OPENCL_INCLUDE = /opt/AMDAPP/include
OPENCL_LIB = /opt/AMDAPP/lib/x86_64

raytracer: ui.cpp raytracer.cl CLRenderer.h CLRenderer.cpp objects.h
    clang++ -std=c++11 ui.cpp CLRenderer.cpp -g -L $(CL_TEST) -I $(CL_TEST) -I $(
        OPENCL_INCLUDE) -L $(OPENCL_LIB) `pkg-config --cflags gtk+-2.0` `pkg-config --
        libs gtk+-2.0` -o raytracer -lCLTest -lOpenCL

raytracer.cl: raytracer.cl.in objects.h
    cpp raytracer.cl.in > raytracer.cl

clean:
    rm -f raytracer.cl raytracer
```

raytracer/objects.h

```
/**
 * Define objects for raytracer.
 *
 * @author John Kloosterman for CS352 at Calvin College
 * @date Dec. 12, 2012
 */

#ifdef _HOST_
#define FLOAT3_T cl_float3
```

```
#define UCHAR3_T cl_uchar3
#define UCHAR4_T cl_uchar4
#define INT3_T cl_int3
#else
#define FLOAT3_T float3
#define UCHAR3_T uchar3
#define UCHAR4_T uchar4
#define INT3_T int3
#endif

typedef struct {
    FLOAT3_T position;
} Light;

typedef struct {
    float radius;
} Sphere;

typedef struct {
    FLOAT3_T normal;
} Plane;

/* Object types */
#define SPHERE_TYPE 0
#define PLANE_TYPE 1
#define CUBE_TYPE 2

typedef struct {
    // if 4th component is 1, then the object is mirrored.
    UCHAR4_T colour;

    int type;
    FLOAT3_T position;
    union {
        Sphere sphere;
        Plane plane;
    } objects;
} Object;
```

raytracer/raytracer.cl.in

```
/**
 * OpenCL kernel for raytracer.
 *
 * @author John Kloosterman for CS352 at Calvin College
 * @date Dec. 12, 2012
 */

#pragma OPENCL EXTENSION cl_amd_printf : enable
#define NUM_CHANNELS 4
#include "objects.h"

/**
 * For a given pixel in the image, compute the direction of
 *  the ray shot from our eye.
 *
 * Based on code from
 *   http://homepages.paradise.net.nz/nickamy/simpleraytracer/simpleraytracer.htm
 *
 * @param x, y
 *  A pixel in the image
 * @param num_x, num_y
 *  The number of x and y pixels in the image
 *
```

```
 * @return
 *  A normalized direction vector for the ray to shoot for
 *   this pixel.
 */
float3 get_ray_direction( size_t x, size_t num_x, size_t y, size_t num_y )
{
    float xFrac = (float) x / (float) num_x;
    float yFrac = (float) y / (float) num_y;

    float3 direction = (float3)( 1.0, -(xFrac - 0.5), -(yFrac - 0.5) );
    return normalize( direction );
}

/**
 * Set the pixel in the pixel buffer corresponding to this kernel's
 *  position to a given value.
 *
 * @param pixbuf
 *  The pixel buffer from the host
 * @param r, g, b
 *  8-bit values to set as the r,g,b values of the pixel.
 */
void set_pixel( __global uchar4 *pixbuf, uchar r, uchar g, uchar b )
{
    size_t x = get_global_id( 0 );
    size_t y = get_global_id( 1 );
    size_t num_x = get_global_size( 0 );

    __global uchar4 *pixel = pixbuf + ( y * num_x ) + x;
    (*pixel).x = r;
    (*pixel).y = g;
    (*pixel).z = b;
    (*pixel).w = 255;
}

/**
 * Determine whether a ray intersects with a sphere, and if so,
 *  compute properties of that intersection.
 *
 * Based on the equations from
 *   http://en.wikipedia.org/wiki/Raytracing#Example.
 *
 * @param camera_position, camera_direction
 *  The ray shot from the camera.
 * @param normal
 *  A location to store the sphere's surface normal
 *   at the intersection point
 * @param intersect
 *  A location to store the intersection point of
 *   the ray with the sphere.
 * @param object
 *  The sphere object in question.
 *
 * @return
 *  The distance from the camera to the intersection point
 *   on the sphere, or -1 if there is no intersection.
 */
float sphere_get_distance(
    float3 camera_position,
    float3 camera_direction,
    float3 *normal,
    float3 *intersect,
    Object object )
{
```

```
    float3 v = camera_position - object.position;
    float v_dot_d = dot( v, camera_direction );
    float r = object.objects.sphere.radius;
    float v_len = length( v );
    float discriminant = ( v_dot_d * v_dot_d ) - ( ( v_len * v_len ) - ( r * r ) );

    // No intersection.
    if ( discriminant < 0 )
        return -1;

    float intersect_1 = -v_dot_d + sqrt( discriminant );
    float intersect_2 = -v_dot_d - sqrt( discriminant );
    float min_intersect = min( intersect_1, intersect_2 );

    // It is behind the camera.
    if ( min_intersect < 0 )
        return -1;

    float3 intersection_point = camera_position + ( min_intersect * camera_direction );

    *normal = normalize( intersection_point - object.position );
    *intersect = intersection_point;

    return distance( camera_position, intersection_point );
}

/**
 * Determine whether a ray intersects with a plane, and if so,
 *  compute properties of that intersection.
 *
 * @param camera_position, camera_direction
 *  The ray shot from the camera.
 * @param normal
 *  A location to store the plane's surface normal
 *   at the intersection point; note this is constant
 *   everywhere on the plane.
 * @param intersect
 *  A location to store the intersection point of
 *   the ray with the plane.
 * @param object
 *  The plane object in question.
 *
 * @return
 *  The distance from the camera to the intersection point
 *   on the plane, or -1 if there is no intersection.
 */
float plane_get_distance(
    float3 camera_position,
    float3 camera_direction,
    float3 *normal,
    float3 *intersect,
    Object object )
{
    // point on the plane:
    float3 point = object.position;
    float3 p_normal = object.objects.plane.normal;

    float numerator = dot( point - camera_position, p_normal );
    float denominator = dot( camera_direction, p_normal );

    // If either numerator or denominator 0, then we either
    //  never intersect or always intersect the plane, so
    //  don't show it.
    if ( numerator == 0 || denominator == 0 )
```

```
        return -1;

    if ( ( numerator / denominator ) < 0 )
        return -1;

    float3 position =
        camera_position
        + ( ( numerator / denominator) * camera_direction );

    *intersect = position;
    *normal = normalize( p_normal );
    return distance( camera_position, position );
}

/**
 * Determine whether a ray intersects with a given object,
 *   and if so, compute properties of that intersection.
 *
 * @param camera_position, camera_direction
 *  The ray shot from the camera.
 * @param normal
 *  A location to store the objects's surface normal
 *   at the intersection point
 * @param intersect
 *  A location to store the intersection point of
 *   the ray with the object.
 * @param object
 *  The object in question.
 *
 * @return
 *  The distance from the camera to the intersection point
 *   on the object, or -1 if there is no intersection.
 */
float get_distance(
    float3 camera_position,
    float3 camera_direction,
    float3 *normal,
    float3 *intersect,
    Object object )
{
    if ( object.type == SPHERE_TYPE )
    {
        return sphere_get_distance(
            camera_position,
            camera_direction,
            normal,
            intersect,
            object
        );
    }
    else if ( object.type == PLANE_TYPE )
    {
        return plane_get_distance(
            camera_position,
            camera_direction,
            normal,
            intersect,
            object
        );
    }

    return -1;
}
```

```
/**
 * Find the first object in the world that a ray will
 *  hit.
 *
 * @param camera_position, camera_direction
 *  The start point and direction of the ray.
 * @param num_objects
 *  The number of objects in the world.
 * @param distance
 *  A location to store the distance from the start
 *   point of the ray and the intersection point with
 *   the first object.
 * @param normal
 *  A location to store the object's surface normal
 *   at the intersection point.
 * @param intersect
 *  A location to store the intersection point with the
 *   first object hit.
 * @param world
 *  An array of objects in the world.
 *
 * @return
 *  The index in @c world of the first object hit by
 *   the ray, or -1 if none.
 */
int closest_object(
    float3 camera_position,
    float3 camera_direction,
    int num_objects,
    float *distance,
    float3 *normal,
    float3 *intersect,
    int ignore_index,
    __global Object *world )
{
    float min_distance = 1000000000;
    int min_object = -1;
    float3 min_normal;
    float3 min_intersect;

    for ( int i = 0; i < num_objects; i++ )
    {
        if ( i == ignore_index )
            continue;

        float3 normal_t;
        float3 intersect_t;
        float distance = get_distance(
            camera_position,
            camera_direction,
            &normal_t,
            &intersect_t,
            world[i]
        );

        if ( distance > 0
             && distance < min_distance )
        {
            min_distance = distance;
            min_object = i;
            min_normal = normal_t;
            min_intersect = intersect_t;
        }
    }
```

```
    *normal = min_normal;
    *intersect = min_intersect;
    *distance = min_distance;
    return min_object;
}

// Shadows, specular, ambient, diffuse, etc.
#define FOGGINESS 0.03f
#define L_AMBIENT 0.5f
#define DIFFUSE_ATTENUATION 1.2f

/**
 * Compute lighting for a given point on an object
 *  in the world.
 *
 * @param lights
 *  An array of the light sources in the scene.
 * @param num_lights
 *  The number of lights in the scene.
 * @param camera_position, camera_direction
 *  The start and direction of the ray from the camera.
 * @param position
 *  The coordinate in the world we are computing lighting for.
 * @param normal
 *  The surface normal of the object at the point we are
 *   computing lighting for.
 * @param colour
 *  The colour of the object the point is on.
 * @param object_distance
 *  How far away this position is from the camera.
 * @param num_objects
 *  The number of objects in the world.
 * @param world
 *  An array of the objects in the world.
 *
 * @return
 *  r,g,b values of the lighting at the given point.
 */
uchar3 compute_lighting(
    __global Light *lights,
    int num_lights,
    float3 camera_position,
    float3 camera_direction,
    float3 position,
    float3 normal,
    uchar4 colour,
    float object_distance,
    int num_objects,
    int index,
    __global Object *world)
{
    float3 f_colour;

    /*
     * If the object is mirrored, find the colour by
     *  following the reflection ray to the first object.
     *
     * OpenCL cannot do recursion, so this will only reflect
     *  the colour of the mirrored object and will not reflect
     *  lighting on that object.
     */
    if ( colour.w == 1 )
    {
```

```
    float d; float3 norm, inter;

    float3 reflection_direction =
        camera_direction - ( 2 * dot( camera_direction, normal ) * normal );

    int closest = closest_object(
        position,
        reflection_direction,
        num_objects,
        &d,
        &norm,
        &inter,
        index,
        world
    );

    if ( closest == -1 )
        return (uchar3)( 0, 0, 0 );
    else
    {
        uchar4 oc = world[closest].colour;
        f_colour.x = (float) oc.x;
        f_colour.y = (float) oc.y;
        f_colour.z = (float) oc.z;
    }
}
else
{
    f_colour.x = (float) colour.x;
    f_colour.y = (float) colour.y;
    f_colour.z = (float) colour.z;
}

// Fog
float distance_factor = FOGGINESS * object_distance;
if ( distance_factor > 1 )
    distance_factor = 1;
float3 fog = (float3)( 128, 128, 128 ) * distance_factor;

// Ambient
float3 f_ambient = L_AMBIENT * f_colour;

// Diffuse
float3 f_diffuse = (float3)(0,0,0);
for ( int i = 0; i < num_lights; i++ )
{
    float light_distance = distance( position, lights[i].position );
    float3 light_vector = normalize( lights[i].position - position );

    // See if we are in shadow.
    float d; float3 norm, inter;
    int closest = closest_object(
        lights[i].position,
        -light_vector,
        num_objects,
        &d,
        &norm,
        &inter,
        index,
        world
    );

    if ( closest != -1
         && d < light_distance )
```

```
            continue;

        float angle = dot( light_vector, normal );

//      if ( angle > 1 )
//          angle = 1;

        if ( angle < 0 )
            continue;

        f_diffuse += f_colour * angle
            * pow( 1 / light_distance, DIFFUSE_ATTENUATION )
            * ( 1 / (float) num_lights );
    }

    float3 light = f_ambient + f_diffuse - fog;

    // Make sure colour values can fit in 8 bits.
    float3 clamped_light = fmin( light, (float3)( 255, 255, 255 ) );
    clamped_light = fmax( clamped_light, (float3)( 0, 0, 0 ) );

    return (uchar3)( clamped_light.x, clamped_light.y, clamped_light.z );
}

/**
 * Raytracing kernel.
 *
 * @param pixbuf
 *  Pixel buffer as a 2-d array in row-major order.
 * @param objects
 *  Array of objects in the world.
 * @param lights
 *  Array of lights in the scene.
 * @param camera_position
 *  The location of the camera.
 * @param num_objects
 *  The number of objects in the world.
 */
__kernel void raytrace(
    __global uchar4 *pixbuf,
    __global Object *objects,
    __global Light *lights,
    __global float3 *host_camera_position,
    __global int *host_num_objects,
    __global int *host_num_lights )
{
    // Copy things to local memory.
    float3 camera_position = *host_camera_position;
    int num_objects = *host_num_objects;
    int num_lights = *host_num_lights;

    // Get information about our relative position.
    size_t x = get_global_id( 0 );
    size_t y = get_global_id( 1 );
    size_t num_x = get_global_size( 0 );
    size_t num_y = get_global_size( 1 );

    // Compute the first object the camera ray hits.
    float3 camera_direction = get_ray_direction( x, num_x, y, num_y );
    float distance;
    float3 normal;
    float3 intersect;
    int closest = closest_object(
        camera_position,
```

```
            camera_direction,
            num_objects,
            &distance,
            &normal,
            &intersect,
            -1,
            objects
        );

        // If it hit nothing, draw black.
        if ( closest == -1 )
        {
            set_pixel( pixbuf, 0, 0, 0 );
        }
        else
        {
            // If it hit something, compute the lighting
            //  at that point and set that as the value of our pixel.

            uchar3 colour = compute_lighting(
                lights,
                num_lights,
                camera_position,
                camera_direction,
                intersect,
                normal,
                objects[closest].colour,
                distance,
                num_objects,
                closest,
                objects
            );

            set_pixel( pixbuf, colour.x, colour.y, colour.z );
        }
    }
```

raytracer/README

```
OpenCL Raytracer : Honours Project for CS352, Fall 2012
=======================================================
Drag using the mouse to pan the image, and the mouse wheel to
  move in and out from the image.

The included scene can be manipulated smoothly on my Radeon 7970
  graphics card.

Compiling:
==========
  -prerequisites: GTK+ 2.14, OpenCL 1.1
  -Modify the Makefile to point to where the OpenCL libraries
    and headers are on your system.
```

raytracer/ui.cpp

```
/**
 * OpenCL raytracer GTK+ UI code.
 *
 * @author John Kloosterman for CS352 at Calvin College
 * @date Dec. 12, 2012
 */

#include "CLRenderer.h"
```

```
#include <gtk/gtk.h>
#include <png.h>

#include <string>
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

// The size of the rendered scene. Right now, this has
//  to be square or there will be distortion.
#define SIZEX 700
#define SIZEY 700
#define PIXEL_BUFFER_SIZE SIZEX * SIZEY * 4

// The pixel buffer that is drawn onto the screen
//  and rendered into.
GdkPixbuf *gdk_pixel_buffer;

// The current camera position
cl_float3 camera_position;

GtkWidget *light_x;
GtkWidget *light_y;
GtkWidget *light_z;
GtkWidget *image;

// The renderer object that will render the scene
CLRenderer *renderer;

// The objects in the scene
#define NUM_OBJECTS 1024
Object objects[NUM_OBJECTS];

#define NUM_SPHERES 2

/**
 * Initially prepare the scene.
 */
void setup_scene()
{
    // objects[0] is the sphere that follows
    //  the light around, that will be filled
    //  in when the scene is rendered.

    // Plane
    objects[1].colour.s[0] = 235;
    objects[1].colour.s[1] = 206;
    objects[1].colour.s[2] = 198;
    objects[1].colour.s[3] = 0;
    objects[1].type = PLANE_TYPE;
    objects[1].position.s[0] = 0;
    objects[1].position.s[1] = 0;
    objects[1].position.s[2] = -1;
    objects[1].objects.plane.normal.s[0] = 0;
    objects[1].objects.plane.normal.s[1] = 0;
    objects[1].objects.plane.normal.s[2] = 1;

    int num = 2;
    for ( int i = 0; i < NUM_SPHERES; i++ )
    {
        for ( int j = 0; j < NUM_SPHERES; j++ )
        {
            for ( int k = 0; k < NUM_SPHERES; k++ )
```

```
            {
                objects[num].colour.s[0] = 100 + i * 10;
                objects[num].colour.s[1] = 100 + k * 10;
                objects[num].colour.s[2] = 100 + j * 10;

                if ( k == 3 )
                    objects[num].colour.s[3] = 1;
                else
                    objects[num].colour.s[3] = 0;

                objects[num].type = SPHERE_TYPE;
                objects[num].position.s[0] = i;
                objects[num].position.s[1] = j;
                objects[num].position.s[2] = k;
                objects[num].objects.sphere.radius = 0.2;

                num++;
            }
        }
    }
}

/**
 * Run the OpenCL renderer to redraw the scene.
 */
void run_kernel()
{
    Light light;
    const int num_objects = ( NUM_SPHERES * NUM_SPHERES * NUM_SPHERES ) + 2;

    // Sphere that follows light
    objects[0].colour.s[0] = 253;
    objects[0].colour.s[1] = 204;
    objects[0].colour.s[2] = 135;
    objects[0].colour.s[3] = 0;
    objects[0].type = SPHERE_TYPE;
    objects[0].position.s[0] = gtk_range_get_value( GTK_RANGE( light_x ) );
    objects[0].position.s[1] = gtk_range_get_value( GTK_RANGE( light_y ) );
    objects[0].position.s[2] = gtk_range_get_value( GTK_RANGE( light_z ) );
    objects[0].objects.sphere.radius = 0.1;

    // The diffuse light
    light.position.s[0] = gtk_range_get_value( GTK_RANGE( light_x ) );
    light.position.s[1] = gtk_range_get_value( GTK_RANGE( light_y ) );
    light.position.s[2] = gtk_range_get_value( GTK_RANGE( light_z ) );

    // Render the scene.
    renderer->render( objects, num_objects, &light, 1, camera_position );
    gtk_image_set_from_pixbuf( GTK_IMAGE( image ), gdk_pixel_buffer );
}

/**
 * Save the current image as a PNG.
 *
 * Code based off of example at
 *   http://zarb.org/~gc/html/libpng.html
 */
static void png_button_clicked( GtkWidget *widget, gpointer data )
{
    FILE *fp = fopen( "output.png", "wb" );
    png_structp png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
        NULL);
    png_infop info_ptr = png_create_info_struct( png_ptr );
```

```
    png_init_io(png_ptr, fp);

    png_set_IHDR(png_ptr, info_ptr, SIZEX, SIZEY,
                 8, PNG_COLOR_TYPE_RGBA, PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(png_ptr, info_ptr);

    // libpng likes to be passed an array of pointers to rows of pixels.
    // Happily oblige, however silly because we have all our
    //  pixels in a continuous buffer.
    unsigned char *pixels = gdk_pixbuf_get_pixels( gdk_pixel_buffer );
    png_byte *row_pointers[SIZEY];
    for ( int i = 0; i < SIZEY; i++ )
    {
        row_pointers[i] = pixels + ( SIZEX * 4 * i );
    }

    png_write_image(png_ptr, row_pointers);
    png_write_end(png_ptr, NULL);
    fclose(fp);
}

/**
 * Exit the program when the window is closed.
 */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}

bool in_handler = false;
/**
 * Redraw the scene when the sliders are moved.
 */
static void redraw_callback( GtkWidget *widget,
                             gpointer   data )
{
    if ( in_handler )
        return;
    in_handler = true;

    run_kernel();

    while ( gtk_events_pending() )
        gtk_main_iteration();

    in_handler = false;
}

/**
 * Move the x and y position of the camera
 *  based on mouse dragging.
 */
bool dragging = false;
float drag_x, drag_y;
float prev_x, prev_y;
int drag_frame = -1;
static void motion_notify( GtkWidget *widget, GdkEvent *event, gpointer user_data )
{
    if ( in_handler )
        return;
    in_handler = true;
```

```
    GdkEventMotion *motion = (GdkEventMotion *) event;

    if ( motion->state & GDK_BUTTON1_MASK )
    {
        if ( !dragging )
        {
            prev_x = camera_position.s[1];
            prev_y = camera_position.s[2];
            drag_x = motion->x;
            drag_y = motion->y;
            dragging = true;
        }
        else
        {
            camera_position.s[1] = prev_x + 0.01 * ( motion->x - drag_x );
            camera_position.s[2] = prev_y + 0.01 * ( motion->y - drag_y );

            run_kernel();

            while ( gtk_events_pending() )
                gtk_main_iteration();
        }
    }
    else
    {
        dragging = false;
    }

    in_handler = false;
}

/**
 * Move the z coordinate of the camera based
 *  on mouse wheel scrolling.
 */
static void scroll( GtkWidget *widget, GdkEvent *event, gpointer user_data )
{
    if ( in_handler )
        return;
    in_handler = true;

    GdkEventScroll *scroll = (GdkEventScroll *) event;

    if ( scroll->direction == GDK_SCROLL_UP )
    {
        camera_position.s[0] += 0.3;
    }
    else if ( scroll->direction == GDK_SCROLL_DOWN )
    {
        camera_position.s[0] -= 0.3;
    }

    run_kernel();

    // This, along with in_handler is a trick to get rid of
    //  any mouse move or scroll events that are on GTK+s
    //  event queue. If we don't purge them, then we will
    //  render frames that we have no intent of displaying,
    //  and the GPU gets behind.
    while ( gtk_events_pending() )
        gtk_main_iteration();

    in_handler = false;
```

```
}

int main( int argc, char *argv[] )
{
    // Set up scene.
    setup_scene();

    // Initialize camera position
    camera_position.s[0] = 0;
    camera_position.s[1] = 0;
    camera_position.s[2] = 0;

    // Initialize GTK+ and the pixel buffer
    gtk_init( &argc, &argv );
    gdk_pixel_buffer = gdk_pixbuf_new( GDK_COLORSPACE_RGB, TRUE, 8, SIZEX, SIZEY );

    // Initialize OpenCL renderer
    bool use_cpu = false;
    if( argc == 2 && strcmp( argv[1], "-cpu" ) == 0 )
        use_cpu = true;

    ifstream t("raytracer.cl");
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());

    renderer = new CLRenderer( src, gdk_pixbuf_get_pixels( gdk_pixel_buffer ), SIZEX,
        SIZEY, use_cpu );

    // Initialize UI
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *vbox;
    GtkWidget *hbox;
    GtkWidget *eventBox;

    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    button = gtk_button_new_with_label( "Save as PNG" );

    // Sliders for light position
    light_x = gtk_hscale_new_with_range(
        -30,
        30,
        0.001 );
    gtk_range_set_value( GTK_RANGE( light_x ), 0 );
    g_signal_connect( light_x, "value-changed", G_CALLBACK( redraw_callback ), NULL );

    light_y = gtk_hscale_new_with_range(
        -30,
        30,
        0.001 );
    gtk_range_set_value( GTK_RANGE( light_y ), 0 );
    g_signal_connect( light_y, "value-changed", G_CALLBACK( redraw_callback ), NULL );

    light_z = gtk_hscale_new_with_range(
        -30,
        30,
        0.001 );
    gtk_range_set_value( GTK_RANGE( light_z ), 0 );
    g_signal_connect( light_z, "value-changed", G_CALLBACK( redraw_callback ), NULL );

    // Horizontal layout
    vbox = gtk_vbox_new( FALSE, 10 );
    gtk_widget_set_size_request( vbox, 400, 200 );
    hbox = gtk_hbox_new( FALSE, 10 );
```

```
    // Draw the initial image.
    image = gtk_image_new_from_pixbuf( gdk_pixel_buffer );
    eventBox = gtk_event_box_new();
    gtk_container_add( GTK_CONTAINER( eventBox ), image );
    gtk_container_add( GTK_CONTAINER( hbox ), eventBox );

    run_kernel();

    // Vertical layout inside right side.
    gtk_container_add( GTK_CONTAINER( vbox ), button );
    gtk_container_add( GTK_CONTAINER( vbox ),
                       gtk_label_new( "Coordinates of light source:" ) );
    gtk_container_add( GTK_CONTAINER( vbox ), light_x );
    gtk_container_add( GTK_CONTAINER( vbox ), light_y );
    gtk_container_add( GTK_CONTAINER( vbox ), light_z );
    gtk_container_add( GTK_CONTAINER( hbox ), vbox );

    // Save a PNG when the button clicked
    g_signal_connect( button, "clicked", G_CALLBACK( png_button_clicked ), NULL );

    // Exit when the window is destroyed
    g_signal_connect( window, "destroy", G_CALLBACK( destroy ), NULL);

    // Connect signals for mouse move and scroll events
    gtk_widget_set_events( eventBox, GDK_POINTER_MOTION_MASK | GDK_BUTTON_PRESS_MASK );
    g_signal_connect (eventBox, "motion-notify-event", G_CALLBACK (motion_notify), NULL
        );
    g_signal_connect (eventBox, "scroll-event", G_CALLBACK (scroll), NULL);

    // Add everything to the window, and show it all.
    gtk_container_add( GTK_CONTAINER( window ), hbox );
    gtk_widget_show_all( window );
    gtk_main();

    return 0;
}
```

A.5 societies

societies/1-resource-extraction/effort.cl

```
/**
 * Functions to compute the effort needed to
 *  extract a resource.
 */

/**
 * Let @c max be the maximum effort to extract a resource.
 * Let @c min be the minimum effort to extract a resource.
 *
 * Then the effort to extract a resource given @c exp experience
 *  with that resource is:
 *
 * max - ((max - min)*e^(-2*sqrt(max))*e^(-(max - min)/2)*(exp/max))
 *
 * This is a Gompertz curve y(t) with:
 *   a = ( max - min ) [upper asymptote]
 *   b = -2 * sqrt( max ) [y displacement]
 *   c = -( max - min ) / ( 2 * max ) [growth rate]
 * See the formula on
 *   http://en.wikipedia.org/w/index.php?title=Gompertz_function&oldid=534656748
 *
 * Prof. Haarsma wrote:
```

```
 *   I realized that I prefer the Gompertz curve
 *   to the generalized logistic curve for the
 *   purpose of representing agents learning
 *   and gaining efficiency through experience.
 *   With the Gompertz curve, as agents gain efficiency,
 *   the agents initially make small gains in efficiency
 *   (the initial flat part of the curve), but pretty
 *   soon start making rapid gains in efficiency.
 *   But once the curve starts to go flat again when
 *   the agents have lots of experience, agents continue
 *   to gain efficiency slowly, and approach the asymptote
 *   of maximum efficiency in a nice, gradual way.
 *
 * @param experience
 *  The amount of experience the agent has with a given
 *   resource.
 * @param max_experience
 *  The maximum amount of experience with a resource agents
 *   can have.
 * @param min_effort
 *  The minimum effort to extract a resource.
 * @param max_effort
 *  The maximum effort to extract a resource.
 *
 * @return
 *  The number of minutes the agent needs to extract that
 *   resource.
 */
float resource_effort( int experience, int max_experience, int min_effort, int
    max_effort )
{
    float f_experience = experience;
    float f_max_experience = max_experience;
    float f_min_effort = min_effort;
    float f_max_effort = max_effort;

    float effort =
        f_max_effort -
        ( ( f_max_effort - f_min_effort )
          * exp( -sqrt(f_max_effort) * 2
                * exp( -(f_max_effort - f_min_effort)
                       * f_experience / f_max_experience ) ) );

        return effort;
}
```

societies/1-resource-extraction/Makefile

```
CC = clang++
CFLAGS = -std=c++11 -g -Wall -I ../config
CL_TEST_INCLUDE = -I ../../cl_test -I /opt/AMDAPP/include
OCL_LIBS = ../../cl_test/libCLTest.a -L /opt/AMDAPP/lib/x86_64 -lOpenCL
OCL_CC = $(CC) $(CFLAGS) $(CL_TEST_INCLUDE)

resource-extraction: resource-extraction.cpp resource-extraction.h ../config/config.cpp
     resource-extraction.cl
    $(OCL_CC) resource-extraction.cpp ../config/config.cpp -o resource-extraction $(
        OCL_LIBS)

resource-extraction.cl: resource-extraction.cl.in effort.cl utility.cl ../util/
    choose_thread.cl ../util/max_min.cl
    cpp -I ../util -I ../mwc64x/cl -I ../config resource-extraction.cl.in > resource-
        extraction.cl
```

```
clean:
    rm -f resource-extraction.cl resource-extraction
```

societies/1-resource-extraction/README

```
In this phase, agents extract resources.

Inputs
======
(1) Configuration object
(2) The amount of each resource each agent has
(3) The amount of experience with each resource each agent has

Kernel Design
=============
Each agent corresponds to one resource. This will put a 256-resource
  limit on the simulation, because one workgroup can be up to 256
  threads.

1. Copy
  a) The amount of each resource the agent has
  b) The amount of experience with each resource the agent has
  to local memory.
2. While there is time remaining in the day:
  -Compute GPM for each resource
  -Compute the time it will take to extract each resource
  -Extract the one with the highest GPM that there is enough time to
    extract. (Local thread 0 does this)
  -Decrement the amount of time left in the day by that amount.
  -Update experience for that resource.
3. Copy amounts and experience back to global memory.

Source code map
===============
resource-extraction.c:
   driver for this stage.

marginal-utility.c:
   Implementations of economic equations.
```

societies/1-resource-extraction/resource-extraction.cl.in

```
/**
 * Simulate one day's worth of collecting resources.
 *
 * Approaches:
 *  -Loop that stores time left and what we got. (fewer iterations)
 */

#define _OPENCL_
#include <config.h>
#include <mwc64x.cl>
#include <choose_thread.cl>
#include <max_min.cl>
#include "effort.cl"
#include "utility.cl"

#define TRUE 1
#define FALSE 0

// Local ids: 1-dimensional: 1 per resource.
// Global ids: resources x agents

/**
```

```
 * Compute the gain per minute of collecting this resource.
 */
void compute_gain_per_minute(
    __local uint *resources,
    __local uint *experiences,
    __local float *gains_per_minute,
    __global SocietiesConfig *config
    )
{
    size_t local_id = get_local_id( 0 );

    float effort = resource_effort(
        experiences[local_id],
        config->max_experience,
        config->min_effort,
        config->max_effort
        );

    gains_per_minute[local_id] = gpm(
        resources[local_id],
        config->resource_D[local_id],
        config->resource_n[local_id],
        effort
        );
}

/**
 * Determine whether this resource is within an epsilon
 *  of the resource with the best gain per minute.
 */
int is_within_epsilon(
    uint max_gpm_index,
    __local float *gains_per_minute,
    __global SocietiesConfig *config
    )
{
    size_t local_id = get_local_id( 0 );

    float difference =
        gains_per_minute[max_gpm_index]
        - gains_per_minute[local_id];

    if ( fabs( difference ) <= config->resource_epsilon )
        return TRUE;
    else
        return FALSE;
}

/**
 * If the agent does not collect a given resource in a day,
 *  decrease its experience with that resource by
 *  config->idle_penalty.
 */
void take_experience_penalty(
    __global uint *all_resources,
    __local uint *resources,
    __local uint *experiences,
    __global SocietiesConfig *config
    )
{
    size_t local_id = get_local_id( 0 );

    // If we did not collect this resource today, take the
    //  penalty for being idle.
```

```
    if ( all_resources[local_id] == resources[local_id] )
    {
        if ( experiences[local_id] < config->idle_penalty )
            experiences[local_id] = 0;
        else
            experiences[local_id] -= config->idle_penalty;
    }
}

/**
 * Clamp the amount of experience the agent has
 *  with a resource to the maximum in config->max_experience.
 */
void clamp_experience(
    __local uint *experiences,
    __global SocietiesConfig *config
    )
{
    size_t local_id = get_local_id( 0 );

    if ( experiences[local_id] > config->max_experience )
        experiences[local_id] = config->max_experience;
}

// The __local arguments in the kernel allow us to dynamically
//  allocate their size.
__kernel void resource_extraction(
    __global uint *all_resources,
    __global uint *all_experiences,
    __global ulong *rng_base_offsets,
    __global SocietiesConfig *config,
    __local uint *resources,
    __local uint *experiences,
    __local float *gains_per_minute,
    __local uint *scratch  // num_threads * sizeof( uint )
    )
{
    volatile __local int counter;
    __local int minutes_used;
    __local uint chosen_resource;

    size_t local_id = get_local_id( 0 );
    size_t agent_offset = get_local_size( 0 ) * get_global_id( 1 );

    // Initialize the RNG.
    mwc64x_state_t rng_state;
    // The documentation ( http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.
        html )
    //  says that 2^40 is a good perStreamOffset if we don't know one.
    // 2^40 = 1099511627776
    //
    // Only thread 0 needs to do this.
    if ( local_id == 0 )
    {
        MWC64X_SeedStreams(
            &rng_state,
            rng_base_offsets[get_global_id( 1 )],
            1099511627776
            );

        minutes_used = 0;
    }

    // Copy our chunk to local memory.
```

```
    resources[local_id] = all_resources[agent_offset + local_id];
    experiences[local_id] = all_experiences[agent_offset + local_id];
    barrier( CLK_LOCAL_MEM_FENCE );

    // Compute gain per minute for all resources.
    compute_gain_per_minute( resources, experiences, gains_per_minute, config );

    while ( minutes_used < config->num_minutes )
    {
        // Find the resource with maximum gain per minute.
        uint max_gpm_index = max_index( gains_per_minute, scratch );
//      printf( "Max GPM: %d, %f\n", max_gpm_index, gains_per_minute[max_gpm_index] );

        // Set the random selection counter to 0.
        if ( local_id == 0 )
            counter = 0;
        barrier( CLK_LOCAL_MEM_FENCE );

        // If this thread's resources is within an epsilon of
        //  that maximum, add it to the list of threads that can
        //  be randomly chosen from.
        if ( is_within_epsilon( max_gpm_index, gains_per_minute, config ) )
        {
            choose_thread_add_to_options( &counter, scratch );
//          printf( "Adding thread %d to options!\n", get_local_id( 0 ) );
        }
        barrier( CLK_LOCAL_MEM_FENCE );

        if ( counter == 0 )
        {
            printf( "The maximum was %f, from thread %d.\n", gains_per_minute[
                max_gpm_index], max_gpm_index );
        }

        // Choose a resource.
        if ( local_id == 0 )
        {
            chosen_resource = choose_thread_make_choice( &counter, scratch, &rng_state
                );
        }
        barrier( CLK_LOCAL_MEM_FENCE );

        // Collect one of that resource and gain experience with it.
        if ( local_id == chosen_resource )
        {
            minutes_used += resource_effort(
                experiences[local_id],
                config->max_experience,
                config->min_effort,
                config->max_effort
                );

            resources[local_id]++;
            experiences[local_id]++;

            // Recompute gain per minute with the new amount and experience.
            compute_gain_per_minute( resources, experiences, gains_per_minute, config )
                ;
        }
        barrier( CLK_LOCAL_MEM_FENCE );
    }

    // Take an experience penalty if a resource was never collected.
    take_experience_penalty( all_resources, resources, experiences, config );
```

```
    // Ensure experience is below the maximum allowed.
    clamp_experience( experiences, config );

    // Copy data back to global memory.
    all_resources[agent_offset + local_id] = resources[local_id];
    all_experiences[agent_offset + local_id] = experiences[local_id];
}
```

societies/1-resource-extraction/resource-extraction.cpp

```
/**
 * Host code to drive the resource extraction
 *  stage of Societies.
 *
 * @date Feb. 7, 2013
 * @author John Kloosterman
 */

#define _HOST_
#define _CPP_11_
#include <CL/cl.hpp>
#include <CLKernel.h>
#include <config.h>
#include <fstream>
#include <random>
#include "resource-extraction.h"
using namespace std;

#define KERNEL_SOURCE "resource-extraction.cl"

/**
 * Perform one day of resource extraction.
 */
ResourceExtraction::ResourceExtraction(
    cl_uint *all_resources,
    cl_uint *all_experiences,
    SocietiesConfig &config,
    unsigned int random_seed
    )
    :
    myRandomOffsets( new cl_ulong[config.num_agents] ),
    resources( all_resources, config.num_agents * config.num_resources ),
    experiences( all_experiences, config.num_agents * config.num_resources ),
    random_offsets( myRandomOffsets, config.num_agents, false, false ),
    config( "SocietiesConfig", config ),
    myConfig( config ),
    mySeed( random_seed )
{
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());

    resource_extraction = new CLKernel( "resource_extraction", src );
    resource_extraction->setGlobalDimensions( config.num_resources, config.num_agents )
        ;
    resource_extraction->setLocalDimensions( config.num_resources, 1 );

    resource_extraction->setLocalArgument( 4, sizeof( cl_uint ) * config.num_resources
        );
    resource_extraction->setLocalArgument( 5, sizeof( cl_uint ) * config.num_resources
        );
    resource_extraction->setLocalArgument( 6, sizeof( cl_float ) * config.num_resources
         );
```

```
    resource_extraction->setLocalArgument( 7, sizeof( cl_uint ) * config.num_resources
        );
}

ResourceExtraction::~ResourceExtraction()
{
    delete myRandomOffsets;
    delete resource_extraction;
}

void ResourceExtraction::generateRandomOffsets()
{
    std::minstd_rand0 generator( mySeed );

    for ( int i = 0; i < myConfig.num_agents; i++ )
    {
        myRandomOffsets[i] = generator();
    }
}

void ResourceExtraction::extractResources()
{
    generateRandomOffsets();
    (*resource_extraction)(
        resources,
        experiences,
        random_offsets,
        config
        );
}

int main ( void )
{
    SocietiesConfig config;
    config = config_generate_default_configuration();

    const int total_resources = config.num_agents * config.num_resources;
    cl_uint all_resources[total_resources];
    cl_uint all_experiences[total_resources];

    for ( int i = 0; i < total_resources; i++ )
    {
        all_resources[i] = 0;
        all_experiences[i] = 0;
    }

    ResourceExtraction re(
        all_resources,
        all_experiences,
        config,
        42 );
    re.extractResources();

    for ( int i = 0; i < config.num_agents; i++ )
    {
        cout << "Agent " << i << ": " << endl;
        for ( int j = 0; j < config.num_resources; j++ )
        {
            cout << "(" << all_resources[i*config.num_resources + j] << "," <<
                all_experiences[i*config.num_resources + j] << ")" << " ";
        }
        cout << endl;
    }
}
```

societies/1-resource-extraction/resource-extraction.h

```
#ifndef _RESOURCE_EXTRACTION_H
#define _RESOURCE_EXTRACTION_H

#define _HOST_
#include <CLKernel.h>
#include <config.h>

class ResourceExtraction
{
public:
    ResourceExtraction(
        cl_uint *all_resources,
        cl_uint *all_experiences,
        SocietiesConfig &config,
        unsigned int random_seed );
    ~ResourceExtraction();

    void extractResources();

private:
    void generateRandomOffsets();

    cl_ulong *myRandomOffsets;
    CLArgument resources;
    CLArgument experiences;
    CLArgument random_offsets;
    CLArgument config;
    CLKernel *resource_extraction;
    SocietiesConfig &myConfig;
    unsigned int mySeed;
};

#endif
```

societies/1-resource-extraction/tests/effort-curve.cpp

```
/**
 * Test program for resource effort curve.
 *
 * Creates a table of effort vs. experience, which
 *  can be charted to see if it matches the expected curve.
 */

#define _CPP_11_
#include <CLFunction.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "../effort.cl"
#define OUTPUT_FILE "effort-curve.out"

int main ( void )
{
    const cl_int min_effort = 3;
    const cl_int max_effort = 9;
    const cl_int max_experience = 600;

    // Open output file.
    fstream out( OUTPUT_FILE, fstream::out );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
```

```
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLFunction<cl_float> resource_effort( "resource_effort", src );

    // Compute efforts.
    for ( cl_int experience = 0; experience <= max_experience; experience++ )
    {
        cl_double effort =
            resource_effort(
                experience,
                max_experience,
                min_effort,
                max_effort
                );

        out << experience << "\t" << effort << endl;
    }

    out.close();
}
```

societies/1-resource-extraction/tests/effort-curve.plot

```
#!/usr/bin/gnuplot

set title "Resource Effort Curve"
set xlabel "Experience with resource"
set ylabel "Minutes to extract one unit of resource"
plot "effort-curve.out" title "" with lines
pause -1 "Exit"
```

societies/1-resource-extraction/tests/gain-per-minute.cpp

```
/**
 * Test program for resource effort curve.
 *
 * Creates a table of effort vs. experience, which
 *  can be charted to see if it matches the expected curve.
 */

#define _CPP_11_
#include <CLFunction.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "../utility.cl"
#define OUTPUT_FILE "gain-per-minute.out"

int main ( void )
{
    const cl_int D = 3;
    const cl_int n = 4;
    const cl_int e = 3;

    // Open output file.
    fstream out( OUTPUT_FILE, fstream::out );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLFunction<cl_float> gpm( "gpm", src );

    // Compute efforts.
```

```
    for ( cl_int x = 0; x < 100; x++ )
    {
        cl_float gain_per_minute =
            gpm( x, D, n, e );

        out << x << "\t" << gain_per_minute << endl;
    }

    out.close();
}
```

societies/1-resource-extraction/tests/gain-per-minute.plot

```
#!/usr/bin/gnuplot

set title "Gain Per Minute"
set xlabel "Units of Resource"
set ylabel "Gain Per Minute"
plot "gain-per-minute.out" title "" with lines
pause -1 "Exit"
```

societies/1-resource-extraction/tests/Makefile

```
all: effort-curve utility marginal-utility gain-per-minute

clean:
    rm -f effort-curve utility marginal-utility gain-per-minute *.out

effort-curve: effort-curve.cpp ../effort.cl
    clang++ -std=c++11 -I ../../../cl_test -I /opt/AMDAPP/include -L /opt/AMDAPP/lib/
        x86_64 effort-curve.cpp ../../../cl_test/libCLTest.a -lOpenCL -o effort-curve

utility: utility.cpp ../utility.cl
    clang++ -std=c++11 -I ../../../cl_test -I /opt/AMDAPP/include -L /opt/AMDAPP/lib/
        x86_64 utility.cpp ../../../cl_test/libCLTest.a -lOpenCL -o utility

marginal-utility: marginal-utility.cpp ../utility.cl
    clang++ -std=c++11 -I ../../../cl_test -I /opt/AMDAPP/include -L /opt/AMDAPP/lib/
        x86_64 marginal-utility.cpp ../../../cl_test/libCLTest.a -lOpenCL -o marginal-
        utility

gain-per-minute: gain-per-minute.cpp ../utility.cl
    clang++ -std=c++11 -I ../../../cl_test -I /opt/AMDAPP/include -L /opt/AMDAPP/lib/
        x86_64 gain-per-minute.cpp ../../../cl_test/libCLTest.a -lOpenCL -o gain-per-
        minute
```

societies/1-resource-extraction/tests/marginal-utility.cpp

```
/**
 * Test program for resource effort curve.
 *
 * Creates a table of effort vs. experience, which
 *  can be charted to see if it matches the expected curve.
 */

#define _CPP_11_
#include <CLFunction.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "../utility.cl"
#define OUTPUT_FILE "marginal-utility.out"

int main ( void )
```

```
{
    const cl_int D = 3;
    const cl_int n = 4;

    // Open output file.
    fstream out( OUTPUT_FILE, fstream::out );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLFunction<cl_float> mu( "mu", src );

    // Compute efforts.
    for ( cl_int x = 0; x < 100; x++ )
    {
        cl_float marginal_utility =
            mu( x, D, n );

        out << x << "\t" << marginal_utility << endl;
    }

    out.close();
}
```

societies/1-resource-extraction/tests/marginal-utility.plot

```
#!/usr/bin/gnuplot

set title "Marginal Utility"
set xlabel "Units of Resource"
set ylabel "Marginal Utility"
plot "marginal-utility.out" title "" with lines
pause -1 "Exit"
```

societies/1-resource-extraction/tests/run_charts.sh

```
#!/bin/bash
#
# View all the charts that are produced in testing.
#

./effort-curve.plot
./gain-per-minute.plot
./marginal-utility.plot
./utility.plot
```

societies/1-resource-extraction/tests/run_tests.sh

```
#!/bin/bash
#
# Run all the unit tests for this stage in the
#  Societies simulation.

echo "Compiling tests..."
make
echo "effort-curve"
./effort-curve
echo "gain-per-minute"
./gain-per-minute
echo "marginal-utility"
./marginal-utility
echo "utility"
./utility
```

societies/1-resource-extraction/tests/utility.cpp

```
/**
 * Test program for resource effort curve.
 *
 * Creates a table of effort vs. experience, which
 *  can be charted to see if it matches the expected curve.
 */

#define _CPP_11_
#include <CLFunction.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "../utility.cl"
#define OUTPUT_FILE "utility.out"

int main ( void )
{
    const cl_int D = 3;
    const cl_int n = 4;

    // Open output file.
    fstream out( OUTPUT_FILE, fstream::out );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLFunction<cl_float> u( "u", src );

    // Compute efforts.
    for ( cl_int x = 0; x < 1000; x++ )
    {
        cl_float utility =
            u( x, D, n );

        out << x << "\t" << utility << endl;
    }

    out.close();
}
```

societies/1-resource-extraction/tests/utility.plot

```
#!/usr/bin/gnuplot

set title "Utility"
set xlabel "Units of Resource"
set ylabel "Utility"
plot "utility.out" title "" with lines
pause -1 "Exit"
```

societies/1-resource-extraction/utility.cl

```
/**
 * Functions to compute economic utility.
 */

/**
 * Computes the utility of a resource when posessing
 *  x units of that resource, using the formula
 *     U(x) = D * x^( 1/n ).
 *  See page 5 of the paper.
```

```
 *
 * @param x
 *  The number of units of the resource already owned.
 * @param D
 *  The level of utility for the resource.
 * @param n
 *  The diminishing marginal utility for the resource.
 */
float u( int x, int D, int n )
{
    float f_x = x;
    float f_D = D;
    float f_n = n;

    return f_D * pow( f_x, 1.0f / f_n );
}

// Take the area under the utility curve between x_old and x_new.
//
// This is not equivalent behaviour to the Python program, which
//  summed it discretely, but this is *much* faster.
// It's not worth doing some elaborate parallel discrete version.
//
// @XXX: This is for _marginal utility_ in Python, and utility is
//  the derivative for utility.
float utility_gain(
    int x_old,
    int x_new,
    int D,
    int n
    )
{
    float f_x_old = x_old;
    float f_x_new = x_new;
    float f_D = D;
    float f_n = n;

    // The antiderivative of U(x) = D * x^( 1/n )
    //  is U' = ( D * n * x^( 1/n + 1 ) ) / ( n + 1 ).

    float numerator_old = f_D * f_n * pow( f_x_old, ( 1 / f_n ) + 1 );
    float numerator_new = f_D * f_n * pow( f_x_new, ( 1 / f_n ) + 1 );
    float denominator = f_n + 1;

    return ( numerator_new - numerator_old ) / denominator;
}

/**
 * Computes the marginal utility of the next unit of
 *  a resource, given that x units are already owned,
 *  using the formula
 *     MU(x) = U(k) - U(k-1)
 *  See page 5 of the paper.
 *
 * @param x
 *  The number of units of the resource already owned.
 * @param D
 *  The level of utility for the resource.
 * @param n
 *  The diminishing marginal utility for the resource.
 */
float mu( int x, int D, int n )
{
    return u( x, D, n ) - u( x - 1, D, n );
```

```
}

/**
 * Compute the gain per minute for a given resource.
 *   See page 7 of the paper.
 *
 * @param x
 *  The number of units of the resource already owned.
 * @param D
 *  The level of utility for the resource.
 * @param n
 *  The diminishing marginal utility for the resource.
 * @param e
 *  The agent's current effort to extract the resource.
 */
float gpm( int x, int D, int n, float e )
{
    return mu( x + 1, D, n ) / e;
}
```

societies/2-trading/Makefile

```
CC = clang++
CFLAGS = -std=c++11 -g -Wall -I ../config
CL_TEST_INCLUDE = -I ../../cl_test -I /opt/AMDAPP/include
OCL_LIBS = ../../cl_test/libCLTest.a -L /opt/AMDAPP/lib/x86_64 -lOpenCL
OCL_CC = $(CC) $(CFLAGS) $(CL_TEST_INCLUDE)

trading: trading.cl trading.cpp trading.h ../config/config.h
    $(OCL_CC) trading.cpp ../config/config.cpp -o trading $(OCL_LIBS)

trading.cl: trading.cl.in menu.cl valuation.cl ../util/max_min.cl
    cpp trading.cl.in > trading.cl

clean:
    rm -f trading trading.cl
```

societies/2-trading/menu.cl

```
/**
 * Functions to compute the "menu" of the least
 *  valuable resources that each of the 2 agents
 *  wants to trade.
 *
 * In this file, a thread is a resource, and does
 *  computation for both trading agents.
 */

#include "../config/config.h"
#include "../1-resource-extraction/utility.cl"
#include "../util/max_min.cl"

/**
 * From the paper: "the value of a resource is simply
 *  the MU of the last unit of each resource they hold.
 *
 * I change this, because the MU of a resource we have
 *  nothing of is undefined. I use the marginal utility of
 *  the next unit of the resource.
 *
 * This has the side effect of making it really valuable to
 *  have at least 1 of something. But that might be OK.
 *
 * @param agent_resources
```

```
 *  The array to read the amount of resources the agent owns.
 * @param valuations
 *  An array to put the valuation of each resource.
 * @param config
 *  The societies config object.
 */
void menu_resource_value(
    __local uint *agent_resources,
    __local float *valuations,
    __global SocietiesConfig *config )
{
    size_t local_id = get_local_id( 0 );

    valuations[local_id] = mu(
        agent_resources[local_id] + 1,
        config->resource_D[local_id],
        config->resource_n[local_id]
    );
}

/**
 * Create the menus of the @c CONFIG_MENU_SIZE
 *  least valuable resources two agents own.
 *
 * @param agent_a, agent_b
 *  The agents to compute the menus for.
 * @param all_resources
 *  The array of all agents' owned resources.
 * @param agent_a_menu, agent_b_menu
 *  Arrays of size CONFIG_MENU_SIZE to put
 *  the menus for the agents.
 * @param resources_scratch
 *  Local memory of at least size
 *    sizeof( uint ) * CONFIG_NUM_RESOURCES
 * @param sort_tree_scratch
 *  Local memory of at least size
 *    sizeof( uint ) * ( CONFIG_NUM_RESOURCES / 2 )
 * @param mask_scratch
 *  Local memory of at least size
 *    sizeof( uchar ) * CONFIG_NUM_RESOURCES
 * @param valuations_scratch
 *  Local memory of at least size
 *    sizeof( float ) * CONFIG_NUM_RESOURCES
 * @param config
 *  The societies configuration object.
 */
void menu_create_menus(
    uint agent_a,
    uint agent_b,
    __global uint *all_resources,
    __local uint *agent_a_menu,
    __local uint *agent_b_menu,
    __local uint *resources_scratch, // num_resources
    __local uint *sort_tree_scratch, // num_resources / 2
    __local uchar *mask_scratch,     // num_resources
    __local float *valuations_scratch, // num_resources
    __global SocietiesConfig *config
    )
{
    size_t local_id = get_local_id( 0 );
    size_t resource_offset;

    // Copy agent A's resources to local memory.
    resource_offset = ( agent_a * CONFIG_NUM_RESOURCES ) + local_id;
```

```
    resources_scratch[local_id] = all_resources[resource_offset];
    barrier( CLK_LOCAL_MEM_FENCE );

//  printf( "Agent A has %d of resource %d.\n", resources_scratch[local_id], local_id )
    ;

    // Find the value of each resource.
    menu_resource_value( resources_scratch, valuations_scratch, config );
    barrier( CLK_LOCAL_MEM_FENCE );
//  printf( "Resource %d has value %f.\n", local_id, valuations_scratch[local_id] );

    // Find the minimum config->menu_size ones.
    n_min_indices(
        config->menu_size,
        valuations_scratch,
        sort_tree_scratch,
        agent_a_menu,
        mask_scratch
        );

    // Do the same for agent B.
    // Copy agent B's resources to local memory.
    resource_offset = ( agent_b * CONFIG_NUM_RESOURCES ) + local_id;
    resources_scratch[local_id] = all_resources[resource_offset];
    barrier( CLK_LOCAL_MEM_FENCE );

    // Find the value of each resource.
    menu_resource_value( resources_scratch, valuations_scratch, config );

    // Find the minimum config->menu_size ones.
    n_min_indices(
        config->menu_size,
        valuations_scratch,
        sort_tree_scratch,
        agent_b_menu,
        mask_scratch
        );
}
```

societies/2-trading/README

```
Strategy:
 -Randomly pair agents by generating a random permutation on the CPU and
   transferring that to the GPU.
 -The random permutaion allows each to randomly be agent A and agent B,
   so we don't need more randomness for that.
 -Dimensions: (resources)x(pair of agents)
 -configurable number of trading rounds

Functions I'll need:
 -n-minimum (for 5 least valuable resources)
 -maximum that might not use all threads (to find the pair with highest internal
     valuation)
   -for pairs, we could have 1 thread per possible pair. This would mean that the "menu
       "
    can be up to 16, because sqrt(256) = 16.
 -
```

societies/2-trading/tests/Makefile

```
CC = clang++
CFLAGS = -std=c++11 -g -Wall -I ../../config
CL_TEST_INCLUDE = -I ../../../cl_test -I /opt/AMDAPP/include
OCL_LIBS = ../../../cl_test/libCLTest.a -L /opt/AMDAPP/lib/x86_64 -lOpenCL
```

```
OCL_CC = $(CC) $(CFLAGS) $(CL_TEST_INCLUDE)

all: menu_test valuation_test trading_test

clean: menu_test_clean valuation_test_clean trading_test_clean

menu_test: menu_test.cl menu_test.cpp
    $(OCL_CC) menu_test.cpp ../../config/config.cpp -o menu_test $(OCL_LIBS)

menu_test.cl: menu_test.cl.in ../menu.cl
    cpp menu_test.cl.in > menu_test.cl

menu_test_clean:
    rm -f menu_test.cl menu_test

valuation_test: valuation_test.cl valuation_test.cpp
    $(OCL_CC) valuation_test.cpp ../../config/config.cpp -o valuation_test $(OCL_LIBS)

valuation_test.cl: valuation_test.cl.in ../menu.cl ../valuation.cl
    cpp valuation_test.cl.in > valuation_test.cl

valuation_test_clean:
    rm -f valuation_test.cl valuation_test

trading_test: trading_test.cpp ../trading.cl
    $(OCL_CC) trading_test.cpp ../../config/config.cpp -o trading_test $(OCL_LIBS)

trading_test_clean:
    rm -f trading_test
```

societies/2-trading/tests/menu_test.cl.in

```
/**
 * Test creating the menu of items to be traded.
 *
 * @author John Kloosterman
 * @date Feb. 15, 2013
 */

#define _OPENCL_
#include "../menu.cl"

__kernel void
menu_tester(
    __global uint *all_resources,
    __global uint *global_menu_1,
    __global uint *global_menu_2,
    __global SocietiesConfig *config
    )
{
    __local uint resources_scratch[CONFIG_NUM_RESOURCES];
    __local uint sort_tree_scratch[CONFIG_NUM_RESOURCES / 2];
    __local uchar mask_scratch[CONFIG_NUM_RESOURCES];
    __local float valuations_scratch[CONFIG_NUM_RESOURCES];

    __local uint menu_1[CONFIG_MENU_SIZE];
    __local uint menu_2[CONFIG_MENU_SIZE];

    menu_create_menus(
        0,
        1,
        all_resources,
        menu_1,
        menu_2,
```

```
        resources_scratch,
        sort_tree_scratch,
        mask_scratch,
        valuations_scratch,
        config
        );

    size_t local_id = get_local_id( 0 );
    if ( local_id < CONFIG_MENU_SIZE )
    {
        global_menu_1[local_id] = menu_1[local_id];
        global_menu_2[local_id] = menu_2[local_id];
    }
}
```

societies/2-trading/tests/menu_test.cpp

```
/**
 * Test program for resource effort curve.
 *
 * Creates a table of effort vs. experience, which
 *  can be charted to see if it matches the expected curve.
 */

#define _CPP_11_
#define _HOST_
#include <CLKernel.h>
#include <config.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "menu_test.cl"

int main ( void )
{
    cout << "Testing building the trading menus..." << endl << flush;

    SocietiesConfig config = config_generate_default_configuration();
    config.num_threads = 20;
    config.num_agents = 2;
    config.num_resources = 20;
    config.menu_size = 5;

    string compiler_flags = config_generate_compiler_flags( config );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLKernel menu_tester( "menu_tester", src, compiler_flags );
    menu_tester.setGlobalDimensions( config.num_threads, 1 );
    menu_tester.setLocalDimensions( config.num_threads, 1 );

    // For each agent,
    // Out of 20 resources, create 4 ones we have too much
    //  of and would like to trade.
    cl_uint host_all_resources[40];
    for ( int i = 0; i < 40; i++ )
    {
        host_all_resources[i] = 0;
    }
    // Agent 0: resources 3, 8, 13, and 17 will be the least valuable.
    // Agent 1: resources 2, 4, 15, and 16 will be the least valuable
    host_all_resources[3] = 20;
```

```
    host_all_resources[8] = 3;
    host_all_resources[13] = 10;
    host_all_resources[17] = 7;
    host_all_resources[20 + 2] = 14;
    host_all_resources[20 + 4] = 6;
    host_all_resources[20 + 15] = 3;
    host_all_resources[20 + 16] = 100;

    // The fifth thing in the menu can be something essentially random.
    cl_uint host_menu_1[5];
    cl_uint host_menu_2[5];

    CLArgument all_resources( host_all_resources, 40 );
    CLArgument menu_1( host_menu_1, 5 );
    CLArgument menu_2( host_menu_2, 5 );
    CLArgument config_buffer( "SocietiesConfig", config );

    menu_tester(
        all_resources,
        menu_1,
        menu_2,
        config_buffer
        );

/*
    cout << "Menu 1: ";
    for ( int i = 0; i < 5; i++ )
        cout << host_menu_1[i] << " ";
    cout << endl;

    cout << "Menu 2: ";
    for ( int i = 0; i < 5; i++ )
        cout << host_menu_2[i] << " ";
    cout << endl;
*/

    assert( host_menu_1[0] == 3 );
    assert( host_menu_1[1] == 13 );
    assert( host_menu_1[2] == 17 );
    assert( host_menu_1[3] == 8 );
    assert( host_menu_2[0] == 16 );
    assert( host_menu_2[1] == 2 );
    assert( host_menu_2[2] == 4 );
    assert( host_menu_2[3] == 15 );

    cout << "All tests passed!" << endl << flush;
}
```

societies/2-trading/tests/trading_test.cpp

```
/**
 */

#define _CPP_11_
#define _HOST_
#include <CLKernel.h>
#include <config.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "../trading.cl"

int main ( void )
{
```

```
    SocietiesConfig config = config_generate_default_configuration();
    config.num_threads = 20;
    config.num_agents = 2;
    config.num_resources = 20;
    config.menu_size = 5;
    config.num_trades = 3;

    string compiler_flags = config_generate_compiler_flags( config );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLKernel trading( "trading", src, compiler_flags );
    trading.setGlobalDimensions( 20, 1 );
    trading.setLocalDimensions( 20, 1 );

    // For each agent,
    // Out of 20 resources, create 4 ones we have too much
    //  of and would like to trade.
    cl_uint host_all_resources[40];
    for ( int i = 0; i < 40; i++ )
    {
        host_all_resources[i] = 8;
    }
    // Agent 0: resources 3, 8, 13, and 17 will be the least valuable.
    // Agent 1: resources 2, 4, 15, and 16 will be the least valuable
    host_all_resources[17] = 10;
    host_all_resources[20 + 15] = 10;

    cl_uint host_random_pairs[2] = { 1, 0 };

    CLArgument all_resources( host_all_resources, 40 );
    CLArgument random_pairs( host_random_pairs, 2 );
    CLArgument config_buffer( "SocietiesConfig", config );

    trading(
        all_resources,
        random_pairs,
        config_buffer
        );

    cout << "=== Results ===" << endl;
    cout << "Agent 1: ";
    for ( int i = 0; i < 20; i++ )
        cout << host_all_resources[i] << " ";
    cout << endl;

    cout << "Agent 2: ";
    for ( int i = 0; i < 20; i++ )
        cout << host_all_resources[20+i] << " ";
    cout << endl;

}
```

societies/2-trading/tests/valuation_test.cl.in

```
/**
 * Tester for the highest-valuation
 *  code.
 */

#define _OPENCL_
#include "../menu.cl"
```

```
#include "../valuation.cl"

__kernel void
valuation_test(
    __global uint *all_resources,
    __global uint2 *pairs,
    __global SocietiesConfig *config
    )
{
    __local uint menu_a[CONFIG_MENU_SIZE];
    __local uint menu_b[CONFIG_MENU_SIZE];
    __local uint resource_a, resource_b;
    __local float internal_valuations_scratch[CONFIG_NUM_RESOURCES];
    __local uint resource_scratch[CONFIG_NUM_RESOURCES];
    __local uint sort_tree[CONFIG_NUM_RESOURCES / 2];
    __local uchar mask_scratch[CONFIG_NUM_RESOURCES];

    __local uint2 local_pairs[CONFIG_NUM_TRADES];

    // Compute menus
    menu_create_menus(
        0, 1,
        all_resources,
        menu_a, menu_b,
        resource_scratch,
        sort_tree,
        mask_scratch,
        internal_valuations_scratch,
        config
        );
    barrier( CLK_LOCAL_MEM_FENCE );

    if ( get_local_id(0) == 0 )
    {
        printf( "A menu: " );
        for ( int i = 0; i < CONFIG_MENU_SIZE; i++ )
            printf( "%d ", menu_a[i] );
        printf( "\nB menu: " );
        for ( int i = 0; i < CONFIG_MENU_SIZE; i++ )
            printf( "%d ", menu_b[i] );
        printf( "\n" );
    }

    valuation_highest_trade_valuation_pairs(
        0,
        menu_a,
        menu_b,
        local_pairs,
        all_resources,
        internal_valuations_scratch,
        sort_tree,
        resource_scratch,
        mask_scratch,
        config
        );
    barrier( CLK_LOCAL_MEM_FENCE );

    if ( get_local_id( 0 ) < CONFIG_NUM_TRADES )
    {
        pairs[get_local_id(0)] = local_pairs[get_local_id(0)];
    }
}
```

societies/2-trading/tests/valuation_test.cpp

```
/**
 * Test program for valuation
 */

#define _CPP_11_
#define _HOST_
#include <CLKernel.h>
#include <config.h>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "valuation_test.cl"

int main ( void )
{
    cout << "Testing trade valuations..." << endl << flush;

    SocietiesConfig config = config_generate_default_configuration();
    config.num_threads = 20;
    config.num_agents = 2;
    config.num_resources = 20;
    config.menu_size = 5;
    config.num_trades = 5;

    string compiler_flags = config_generate_compiler_flags( config );

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
    CLKernel valuation_tester( "valuation_test", src, compiler_flags );
    valuation_tester.setGlobalDimensions( config.num_threads, 1 );
    valuation_tester.setLocalDimensions( config.num_threads, 1 );

    // For each agent,
    // Out of 20 resources, create 4 ones we have too much
    //  of and would like to trade.
    cl_uint host_all_resources[40];
    for ( int i = 0; i < 40; i++ )
    {
        host_all_resources[i] = 40 - i;
    }
    // Agent 0: resources 3, 8, 13, and 17 will be the least valuable.
    // Agent 1: resources 2, 4, 15, and 16 will be the least valuable
    host_all_resources[4] = 50;
    host_all_resources[5] = 60;
    host_all_resources[9] = 30;
    host_all_resources[20 + 9] = 40;

    cl_uint2 host_pairs[5];

    CLArgument all_resources( host_all_resources, 40 );
    CLArgument pairs( host_pairs, 5 );
    CLArgument config_buffer( "SocietiesConfig", config );

    valuation_tester(
        all_resources,
        pairs,
        config_buffer
        );

    for ( int i = 0; i < 5; i++ )
```

```
    {
        cout << "Pair " << i << " resources: " << host_pairs[i].s[0] << " " <<
            host_pairs[i].s[1] << endl;
    }
}
```

societies/2-trading/trading.cl.in

```
/**
 * Glue code to do trading.
 *
 * @author John Kloosterman
 * @date Feb. 12, 2013
 */

#pragma OPENCL EXTENSION cl_amd_printf : enable

#define _OPENCL_
#include "menu.cl"
#include "valuation.cl"

/**
 * Compute the gain or loss of utility made by
 *  trading @c resource_given_loss units of
 *  @c resource_given for @c resource_received_gain
 *  units of @c resource_recieved.
 *
 * @param resource_given, resource_received
 *  The resource numbers of the resources given and
 *  received.
 * @param resource_given_held, resource_received_held
 *  The amount of units of the given and received resources
 *  already held.
 * @param resource_given_loss, resource_given_gain
 *  The amounts of the resource that will be involved in
 *  the exchange.
 * @param config
 *  The societies configuration object.
 *
 * @return
 *  The amount of utility gained/lost by making the trade.
 */
float trading_utility_difference(
    uint resource_given,
    uint resource_received,
    uint resource_given_held,
    uint resource_received_held,
    uint resource_given_loss,
    uint resource_received_gain,
    __global SocietiesConfig *config
    )
{
    float resource_given_loss_utility =
        u( resource_given_held - resource_given_loss,
           config->resource_D[resource_given],
           config->resource_n[resource_given] )
        - u( resource_given_held,
             config->resource_D[resource_given],
             config->resource_n[resource_given] );

    float resource_received_gain_utility =
        u( resource_received_held + resource_received_gain,
           config->resource_D[resource_received],
           config->resource_n[resource_received] )
```

```
        - u( resource_received_held,
             config->resource_D[resource_received],
             config->resource_n[resource_received] );

    return resource_received_gain_utility - resource_given_loss_utility;
}

/**
 * Find the optimal multiple of num_given:num_received
 *  for agent number @c giver.
 * For instance, if @c num_given is 1 and @c num_received
 *  is 2, and the optimal trade at that ratio is given
 *  3 units or @c resource_given to receive 6 units of
 *  @c resource_received, this function will return 3.
 *
 * @param num_given, num_received
 *  The ratio of units of resource given to resource received.
 * @param giver
 *  The agent who will be giving @c resource_given in the trade.
 * @param recipient
 *  The agent who will be giving @c resource_received in the trade.
 * @param resource_given, resource_received
 *  The resources involved in the trade.
 * @param all_resources
 *  The global resource posession array.
 * @param sort_tree_scratch
 *  Scratch local memory, of size at least
 *   sizeof( uint ) * ( CONFIG_NUM_THREADS / 2 )
 * @param surpluses_scratch
 *  Scratch local memory, or size at least
 *   sizeof( float ) * ( CONFIG_NUM_THREADS )
 *
 * @return
 *  An integer to multiply the ratio num_received:num_given
 *   by to make the optimal trade for @c giver.
 */

int trading_find_maximum_surplus(
    uint num_given,
    uint num_received,
    uint giver,
    uint recipient,
    uint resource_given, // resource giver is giving up
    uint resource_received, // resource giver is receiving
    __global uint *all_resources,
    __local uint *sort_tree_scratch,  // num_threads / 2
    __local float *surpluses_scratch, // num_threads
    __global SocietiesConfig *config )
{
    // NUM_RESOURCES is a proxy for number of threads.
    int offset = 1;
    size_t local_id = get_local_id( 0 );

    // Is it faster to get only 1 thread to do this? Do we have to
    //  use global memory?
    int resource_offset = giver * CONFIG_NUM_RESOURCES;
    uint resource_given_amount = all_resources[resource_offset + resource_given];
    uint resource_received_amount = all_resources[resource_offset + resource_received];

    // Check to make sure the recipient can afford trades.
    int recipient_resource_offset = recipient * CONFIG_NUM_RESOURCES;
    uint recipient_received_amount = all_resources[recipient_resource_offset +
        resource_received];
```

```
    int maximum_found = FALSE;
    uint max_multiplier;
    while ( !maximum_found )
    {
        // The amount of change in amount of resource1 and resource2
        float resource_given_loss = num_given * ( local_id + offset );
        float resource_received_gain = num_received * ( local_id + offset );

        // If we can't afford this trade, don't consider it.
        float surplus = trading_utility_difference(
            resource_given,
            resource_received,
            resource_given_amount,
            resource_received_amount,
            resource_given_loss,
            resource_received_gain,
            config );

        if ( resource_given_loss > resource_given_amount )
            surplus = -1;
        else if ( resource_received_gain > recipient_received_amount )
            surplus = -2;

        printf( "Thread ID: %d\n", local_id );
        surpluses_scratch[local_id] = surplus;
        barrier( CLK_LOCAL_MEM_FENCE );
        printf( "Surplus for thread %d: %f.\n", local_id, surplus );

        // Find the maximum.
        max_multiplier = max_index( surpluses_scratch, sort_tree_scratch );

        // If the maximum is the last thread, we don't know if it was the true maximum.
        //  In that case, do another round, with the last thread being reconsidered.
        if ( max_multiplier == ( CONFIG_NUM_THREADS - 1 ) )
        {
            offset += CONFIG_NUM_THREADS - 2;
            barrier( CLK_LOCAL_MEM_FENCE );
        }
        else if ( surpluses_scratch[max_multiplier] < 0 )
        {
            // The maximum was a negative number, so we can't afford any trade.
            return -1;
        }
        else
        {
            maximum_found = TRUE;
        }
    }

    return max_multiplier + offset;
}

/**
 * Transfer the resources involved in the trade.
 *
 * @param agent_a, agent_b
 *  The agents involved in the trade.
 * @param resource_a
 *  The resource @c agent_a is giving up.
 * @param resource_b
 *  The resource @c agent_b is giving up.
 * @param amount_a
 *  The amount of @c resource_a to transfer from
 *  @c agent_a to @c agent_b.
```

```
 * @param amount_b
 *  The amount of @c resource_b to transfer from
 *  @c agent_b to @c agent_a.
 * @param all_resources
 *  The global array of all agents' resources.
 */
void trading_make_trade(
    uint agent_a,
    uint agent_b,
    uint resource_a,
    uint resource_b,
    uint amount_a,
    uint amount_b,
    __global uint *all_resources
    )
{
    int agent_a_offset = ( agent_a * CONFIG_NUM_RESOURCES );
    printf( "Giver: old #%d: %d, old #%d: %d, ",
            resource_a,
            all_resources[agent_a_offset + resource_a],
            resource_b,
            all_resources[agent_a_offset + resource_b] );
    all_resources[agent_a_offset + resource_a] -= amount_a;
    all_resources[agent_a_offset + resource_b] += amount_b;
    printf( "new #%d: %d, new #%d: %d\n",
            resource_a,
            all_resources[agent_a_offset + resource_a],
            resource_b,
            all_resources[agent_a_offset + resource_b] );

    int agent_b_offset = ( agent_b * CONFIG_NUM_RESOURCES );
    printf( "Receiver: old #%d: %d, old #%d: %d, ",
            resource_a,
            all_resources[agent_b_offset + resource_a],
            resource_b,
            all_resources[agent_b_offset + resource_b] );
    all_resources[agent_b_offset + resource_a] += amount_a;
    all_resources[agent_b_offset + resource_b] -= amount_b;
    printf( "new #%d: %d, new #%d: %d\n",
            resource_a,
            all_resources[agent_b_offset + resource_a],
            resource_b,
            all_resources[agent_b_offset + resource_b] );
}

/**
 * The kernel to perform a trade.
 *
 * Dimensions:
 *  (number of resources)x(pairs of agents)
 *
 * @param all_resources
 *  The array of all agents' resources.
 * @param random_pairs
 *  A random permutation of the set [0, CONFIG_NUM_AGENTS - 1]
 * @param config
 *  The Societies configuration object.
 */
__kernel void trading(
    __global uint *all_resources,
    __global uint *random_pairs,
    __global SocietiesConfig *config
    )
{
```

```
size_t local_id = get_local_id( 0 );
size_t pair_id = get_global_id( 1 );
int pair_offset = 2 * pair_id;
uint agent_a = random_pairs[pair_offset];
uint agent_b = random_pairs[pair_offset + 1];

__local uint menu_a[CONFIG_MENU_SIZE];
__local uint menu_b[CONFIG_MENU_SIZE];
__local uint sort_tree[CONFIG_NUM_THREADS / 2];
__local float float_scratch[CONFIG_NUM_THREADS];
__local uint resources_scratch[CONFIG_NUM_THREADS];
__local uchar mask_scratch[CONFIG_NUM_THREADS];

/* How this works in the Python code:
 (1) we generate menus
 (2) each agent makes a list of the top num_trades possible trades from
     the menus
 (3) In the first round, agent a proposes trading the things it wants
     to trade the most. Agent A proposes a number to trade, B can
     accept or reject. If B rejects, then B proposes a number to
     trade, and A can accept or reject.
 (4) In the second round, B proposes trading the things it wants the most.
 (5) In the third round, A proposes trading its second-favourite pair.
*/

// Generate the menus.
menu_create_menus(
    agent_a,
    agent_b,
    all_resources,
    menu_a,
    menu_b,
    resources_scratch,
    sort_tree,
    mask_scratch,
    float_scratch,
    config
    );
barrier( CLK_LOCAL_MEM_FENCE );

__local uint2 agent_a_pairs[CONFIG_NUM_TRADES];
__local uint2 agent_b_pairs[CONFIG_NUM_TRADES];

// Compute the num_trades favourite pairs for A.
valuation_highest_trade_valuation_pairs(
    agent_a,
    menu_a, menu_b,
    agent_a_pairs,
    all_resources,
    float_scratch,
    sort_tree,
    resources_scratch,
    mask_scratch,
    config
    );

valuation_highest_trade_valuation_pairs(
    agent_b,
    menu_b, menu_a,
    agent_b_pairs,
    all_resources,
    float_scratch,
    sort_tree,
    resources_scratch,
```

```
        mask_scratch,
        config
        );
    barrier( CLK_LOCAL_MEM_FENCE );

    // Do the trades.
    int a_is_first_mover = TRUE;
    uint given_resource;
    uint received_resource;
    for ( int i = 0; i < ( config->num_trades * 2); i++ )
    {
        if ( local_id == 0 )
            printf( "=== Trade number %d ===\n", i );

        uint first_mover;
        uint respondent;

        int pair_id = i / 2;
        if ( a_is_first_mover )
        {
            first_mover = agent_a;
            respondent = agent_b;
            given_resource = agent_a_pairs[pair_id].x;
            received_resource = agent_a_pairs[pair_id].y;
        }
        else
        {
            first_mover = agent_b;
            respondent = agent_a;
            given_resource = agent_b_pairs[pair_id].x;
            received_resource = agent_b_pairs[pair_id].y;
        }

        if ( local_id == 0 )
        {
            printf( "Agent %d is first mover, %d respondent.\n", first_mover,
                respondent );

            printf( "First mover resources: " );
            int fm_offset = first_mover * CONFIG_NUM_RESOURCES;
            for ( int j = 0; j < CONFIG_NUM_RESOURCES; j++ )
                printf( "%d ", all_resources[fm_offset + j] );
            printf( "\n" );

            printf( "Respondant resources: " );
            fm_offset = respondent * CONFIG_NUM_RESOURCES;
            for ( int j = 0; j < CONFIG_NUM_RESOURCES; j++ )
                printf( "%d ", all_resources[fm_offset + j] );
            printf( "\n" );


            printf( "Trade attempting: first_mover giving %d, receiving %d.\n",
                given_resource, received_resource );
        }

        // Calculate agent A and B's internal valuations
        //  to see if the trade is beneficial and what
        //  the bargaining price will be.
        float first_mover_valuation;
        float respondent_valuation;
//      if ( local_id == 0 )
//      {
            first_mover_valuation = valuation_internal_valuation(
                first_mover,
```

```
                given_resource, received_resource,
                all_resources,
                config );
//      }
//      else if ( local_id == 1 )
//      {
            respondent_valuation = valuation_internal_valuation(
                respondent,
                received_resource, given_resource,
                all_resources,
                config );
//      }
//      barrier( CLK_LOCAL_MEM_FENCE );

        // Is the trade mutually beneficial? Is this the right way to compute that?
        if ( first_mover_valuation < 1
             || respondent_valuation < 1 )
        {
            // do something
        }

        if ( local_id == 0 )
            printf( "Valuations: First mover: %f, Respondant: %f\n",
                first_mover_valuation, respondent_valuation );

        // Bargaining price is the geometric mean of
        //  Agent A and B's internal valuations.
        float f_bargaining_price = sqrt( first_mover_valuation * respondent_valuation )
            ;

        // Calculate a nearby integer ratio for the bargaining price.
        int num_given, num_received;
        if ( f_bargaining_price > 1 )
        {
            num_received = 1;
            num_given = f_bargaining_price;
        }
        else
        {
            num_received = (int) ( 1.0f / f_bargaining_price );
            // Always trade at least one, so as to never trade for nothing.
            num_given = fmax( f_bargaining_price * (float) num_received, 1.0f );
        }

        if ( local_id == 0 )
            printf( "Bargaining price: giving %d to receive %d.\n", num_given,
                num_received );

        // First mover optimizes its maximum surplus. We assign one thread to each
        //  multiplier of more surplus.
        int maximum_surplus_multiplier = trading_find_maximum_surplus(
            num_given,
            num_received,
            first_mover,
            respondent,
            given_resource,
            received_resource,
            all_resources,
            sort_tree,
            float_scratch,
            config );
        barrier( CLK_LOCAL_MEM_FENCE );

        if ( local_id == 0 )
```

```
            printf( "Maximum surplus multiplier: %d\n", maximum_surplus_multiplier );

        // Respondent accepts only if it benefits from the trade. The respondent
            benefits when
        //  the utility of the resources received is greater than the loss of
        //  utility of the resources given up.
        if ( maximum_surplus_multiplier > 0
             && TRUE /* trading_respondent_benefits(...) */ )
        {
            if ( local_id == 0 )
            {
                trading_make_trade(
                    first_mover,
                    respondent,
                    given_resource,
                    received_resource,
                    num_given * maximum_surplus_multiplier,
                    num_received * maximum_surplus_multiplier,
                    all_resources
                    );
            }
        }

        // If the respondent didn't accept, the respondent gets to choose
        //  a ratio and propose it to A.

        a_is_first_mover = !a_is_first_mover;
        barrier( CLK_GLOBAL_MEM_FENCE );
    }
}
```

societies/2-trading/trading.cpp

```
#include "trading.h"

#include <CLKernel.h>

#include <random>
#include <iostream>
#include <fstream>
using namespace std;

#define KERNEL_SOURCE "trading.cl"

/// @XXX: make sture that random_seed isn't the same as some other class' random seed.
Trading::Trading(
    cl_uint *all_resources,
    SocietiesConfig &config,
    unsigned int random_seed
    )
    :
    myRandomPairs( new cl_uint[config.num_agents] ),
    myConfig( config ),
    myGenerator( random_seed ),
    allResources( all_resources, config.num_agents * config.num_resources ),
    randomPairs( myRandomPairs, config.num_agents ),
    configBuffer( "SocietiesConfig", config )
{
    generate_random_pairs();

    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());
```

```
    string compiler_flags = config_generate_compiler_flags( config );
    trading_kernel = new CLKernel( "trading", src, compiler_flags );

    trading_kernel->setGlobalDimensions( config.num_threads, config.num_agents );
    trading_kernel->setLocalDimensions( config.num_threads, 1 );

    for ( int i = 0; i < myConfig.num_agents; i++ )
    {
        cout << myRandomPairs[i] << " ";
    }
    cout << endl;
}

Trading::~Trading()
{
    delete[] myRandomPairs;
}

void Trading::trade()
{
    (*trading_kernel)(
        allResources,
        randomPairs,
        configBuffer
        );
}

void Trading::generate_random_pairs()
{
    // Do an "inside-out" Knuth shuffle
    //  http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle
    myRandomPairs[0] = 0;

    for ( int i = 1; i < myConfig.num_agents; i++ )
    {
        // 0 <= random_idx <= i
        int random_idx = myGenerator() % ( i + 1 );

        myRandomPairs[i] = myRandomPairs[random_idx];
        myRandomPairs[random_idx] = i;
    }
}

int main ( void )
{
    cl_uint resources;
    SocietiesConfig config = config_generate_default_configuration();

    Trading trading(
        &resources,
        config,
        42
        );

    cout << config_generate_compiler_flags( config ) << endl;
}
```

societies/2-trading/trading.h

```
#ifndef _TRADING_H
#define _TRADING_H

#define _HOST_
#define _CPP_11_
```

```
#include <CLKernel.h>
#include <config.h>
#include <random>

class Trading
{
public:
    Trading(
        cl_uint *all_resources,
        SocietiesConfig &config,
        unsigned int random_seed
        );

    ~Trading();

    void trade();

private:
    void generate_random_pairs();

    cl_uint *myRandomPairs;
    SocietiesConfig myConfig;
    std::minstd_rand0 myGenerator;

    CLKernel *trading_kernel;
    CLArgument allResources;
    CLArgument randomPairs;
    CLArgument configBuffer;
};

#endif
```

societies/2-trading/valuation.cl

```
/**
 * For every pair of resources, determine whether
 *  it would be a mutually beneficial trade.
 *
 * In this file, each thread is a possible pair of
 *  resources. Therefore, the menu size can't be larger
 *  than 16, since 16^2 = 256.
 *
 * @author John Kloosterman
 * @date Feb. 2013
 */

#include "../util/max_min.cl"

// returns TRUE if this thread will be involved, FALSE otherwise.
//  It depends on the size of the menu.

/**
 * Determine which two resources this thread corresponds to.
 *
 * @param resource_1, resource_2
 *  Where to put the two resource numbers.
 * @param config
 *  The Societies configuration object.
 *
 * @return
 *  TRUE if the current thread corresponds to a
 *   pair to be considered as a potential trade,
 *   FALSE otherwise.
 */
```

```
int valuation_resources(
    uint *resource_1,
    uint *resource_2,
    __global SocietiesConfig *config )
{
    size_t local_id = get_local_id( 0 );
    int max_thread_needed = ( config->menu_size * config->menu_size ) - 1;

    if ( local_id > max_thread_needed )
        return FALSE;

    (*resource_1) = local_id / config->menu_size;
    (*resource_2) = local_id % config->menu_size;

    return TRUE;
}

/**
 * Compute an agent's internal valuation of one
 *  resource against another.
 *
 * @param agent
 *  The index of the agent in question.
 * @param resource1, resource2
 *  The resources being considered.
 * @param all_resources
 *  The array of all agents' resources.
 * @param config
 *  The Societies configuration object.
 */
float valuation_internal_valuation(
    uint agent,
    uint resource1,
    uint resource2,
    __global uint *all_resources,
    __global SocietiesConfig *config
    )
{
    // Evaluate MU( outbound )
    int resource1_offset = ( agent * CONFIG_NUM_RESOURCES ) + resource1;
    uint resource1_amount = all_resources[resource1_offset];
    float resource1_valuation = mu(
        resource1_amount + 1,
        config->resource_D[resource1],
        config->resource_n[resource1]
        );

    // Evaluate MU( inboud )
    int resource2_offset = ( agent * CONFIG_NUM_RESOURCES ) + resource2;
    uint resource2_amount = all_resources[resource2_offset];
    float resource2_valuation = mu(
        resource2_amount + 1,
        config->resource_D[resource2],
        config->resource_n[resource2]
        );

    return resource2_valuation / resource1_valuation;
}

// I don't know how to check. And even if a trade is stupid for
//  both sides, it might not be any slower for us to discover
//  that later.
int valuation_trade_beneficial(
    float agent_a_valuation,
```

```
    float agent_b_valuation
    )
{
    return TRUE;
}

/**
 * Compute the @c CONFIG_NUM_TRADES best possible
 *  trades for @c agent_a made up of resources from
 *  @c menu_a and @c menu_b, and puts them in
 *  @c pairs.
 *
 * @param agent_a
 *  The index of the agent in question.
 * @param menu_a, menu_b
 *  Agent A and B's menus.
 * @param pairs
 *  Where to put the CONFIG_NUM_TRADES top pairs.
 * @param internal_valuations_scratch
 *  Local memory, of size
 *    sizeof( float ) * ( CONFIG_MENU_SIZE^2 )
 *  Note that CONFIG_MENU_SIZE^2 <= CONFIG_NUM_THREADS,
 *   because of restrictions checked by
 *   config_verify_configuration().
 * @param sort_tree
 *  Local memory, of size
 *    sizeof( uint ) * ( CONFIG_MENU_SIZE^2 / 2 )
 * @param indices_scratch
 *  Local memory, of size
 *    sizeof( uint ) * ( CONFIG_NUM_TRADES )
 * @param mask_scratch
 *  Local memory, of size
 *    sizeof( uchar ) * ( CONFIG_MENU_SIZE^2 )
 * @param config
 *  the Societies configuration object.
 */
void valuation_highest_trade_valuation_pairs(
    uint agent_a,
    __local uint *menu_a,
    __local uint *menu_b,
    __local uint2 *pairs,
    __global uint *all_resources,
    __local float *internal_valuations_scratch, // config_menu_size^2, which is <=
        num_threads
    __local uint *sort_tree,                    // config_menu_size^2 / 2
    __local uint *indices_scratch,              // config_num_trades
    __local uchar *mask_scratch,                // config_menu_size^2
    __global SocietiesConfig *config
    )
{
    size_t local_id = get_local_id( 0 );
    uint thread_index_1, thread_resource_1;
    uint thread_index_2, thread_resource_2;

    // Threads not in a pair still have to participate in
    //  synchronization and min/max.
    int is_pair = valuation_resources( &thread_index_1, &thread_index_2, config );
    thread_resource_1 = menu_a[thread_index_1];
    thread_resource_2 = menu_b[thread_index_2];

    printf( "Thread %d (is_pair %d): Resource %d and %d.\n", local_id, is_pair,
        thread_resource_1, thread_resource_2 );

    // This thread is involved in a pair.
```

```
    if ( is_pair )
    {
        internal_valuations_scratch[local_id] = valuation_internal_valuation(
            agent_a,
            thread_resource_1,
            thread_resource_2,
            all_resources,
            config
            );
    }
    barrier( CLK_LOCAL_MEM_FENCE );

    // Find the indices of the maximum num_trades pairs.
    n_max_indices(
        CONFIG_NUM_TRADES,
        internal_valuations_scratch,
        sort_tree,
        indices_scratch,
        mask_scratch
        );
    barrier( CLK_LOCAL_MEM_FENCE );

    // Turn those indices into pairs.
    for ( int i = 0; i < CONFIG_NUM_TRADES; i++ )
    {
        if ( local_id == indices_scratch[i] )
        {
            printf( "%d: Thread %d.\n", i, local_id );

            pairs[i].x = thread_resource_1;
            pairs[i].y = thread_resource_2;
        }
        barrier( CLK_LOCAL_MEM_FENCE );
    }
}
```

societies/config/config.cpp

```
/**
 * Functions to create and validate Societies
 *  config structs.
 *
 * @author John Kloosterman
 * @date Feb. 7, 2013
 */

#include "config.h"
#include <string>
#include <iostream>
#include <sstream>
using namespace std;

SocietiesConfig config_generate_default_configuration( void )
{
    SocietiesConfig config;

    config.num_threads = 256;

    config.num_agents = 100;
    config.num_resources = 256;
    config.num_days = 50;
    config.num_minutes = 600;

    for ( int i = 0; i < 256; i++ )
```

```
    {
        config.resource_D[i] = 10;
        config.resource_n[i] = 10;
    }

    config.resource_epsilon = 1;
    config.max_experience = 200;
    config.min_effort = 3;
    config.max_effort = 9;

    config.menu_size = 5;
    config.num_trades = 5;

    return config;
}

bool config_verify_configuration( SocietiesConfig config )
{
    /// @XXX: ask OpenCL for the real number.
    if ( config.num_resources > 256 )
    {
        cerr << "There are more resource than the OpenCL workgroup size." << endl;
        return false;
    }

    // We assume there is one thread per resources.
    if ( config.num_threads != config.num_resources )
    {
        cerr << "There have to be the same number of threads as resources." << endl;
        return false;
    }

    // The menu_size^2 needs to be >= the number of threads.
    if ( (config.menu_size * config.menu_size ) > config.num_threads )
    {
        cerr << "The trading menu size squared needs to be less than the number of
            threads." << endl;
        return false;
    }

    return true;
}

// Create -D flags that can be passed to the OpenCL compiler.
string config_generate_compiler_flags( SocietiesConfig config )
{
    stringstream flags;

    // num_resources
    flags << "-D CONFIG_NUM_RESOURCES=" << config.num_resources << " ";

    // menu_size
    flags << "-D CONFIG_MENU_SIZE=" << config.menu_size << " ";

    // num_threads
    flags << "-D CONFIG_NUM_THREADS=" << config.num_threads << " ";

    // num_trades
    flags << "-D CONFIG_NUM_TRADES=" << config.num_trades << " ";

    return flags.str();
}
```

societies/config/config.h

```
#ifndef _CONFIG_H
#define _CONFIG_H

/**
 * Structure that holds all the tunable configuration data.
 *
 * Anything that cannot change at runtime should be stored
 *  here so that it can be tuned in the configuration file.
 */

#include "../util/types.h"

typedef struct {
    /* The number of OpenCL threads to use per agent. */
    CL_INT num_threads;

    /* The number of agents */
    CL_INT num_agents;

    /* The number of resources */
    CL_INT num_resources;

    /* The number of days in the simulation */
    CL_INT num_days;

    /* The number of minutes in a day. */
    CL_INT num_minutes;

    /* The D and n values for each resource.
       Because of GPU limitations, there can be at
       most 256 resources, so these can be a known size.
    */
    CL_UCHAR resource_D[256];
    CL_UCHAR resource_n[256];

    /* How close to the maximum gain per minute a
       resource needs to be in order to be collected. */
    CL_FLOAT_T resource_epsilon;

    CL_INT max_experience;
    CL_INT min_effort;
    CL_INT max_effort;

    /* The number of resources that are offered during trades. */
    CL_INT menu_size;

    /* The number of trade rounds per pairing. */
    CL_INT num_trades;

    /* The experience penalty for not collecting a resources
       on a given day. */
    CL_INT idle_penalty;

} SocietiesConfig;

#ifdef _HOST_
#include <string>

SocietiesConfig config_generate_default_configuration( void );
bool config_verify_configuration( SocietiesConfig config );
std::string config_generate_compiler_flags( SocietiesConfig config );
```

```
#endif

#endif
```

societies/README

```
This project will be organized into directories per stage in the simulation.
Each of the steps is completely independent.

There will be a tests directory in each subfolder with the tests for the
  functions used therein.
There will be a test for every significant function.

This project should be compilable using Visual Studio, so we can optimize
  it. That means no C++11, sadly.

Everything needs to be so obvious that beginning coders can understand
  what's going on.
```

societies/util/choose_thread.cl

```
/**
 * Randomly choose one thread out of a set of
 *  valid ones in the current workgroup.
 *
 * @author John Kloosterman
 * @date Feb. 3, 2013
 */

// Random number generator
#include <mwc64x.cl>

/*
 * How this works:
 *  (1) In scratch memory, create a counter of the number of valid options
 *      and a contiguous array of those valid options.
 *  (2) Thread 0 chooses a random integer between (0, # of valid options)
 *  (3) All the threads check to see if it was them. If so,
 *      they can do something.
 */

/**
 * All threads that want to be an option
 *  call this function.
 *
 * Postconditions:
 *   There needs to be a local barrier before
 *   choose_thread_make_choice() can be called.
 *
 * @param counter
 *  A local variable, initialized to 0 before any threads call
 *   this function.
 * @param scratch
 *  An array of size at least
 *    sizeof( uint ) * CONFIG_NUM_THREADS
 */
void choose_thread_add_to_options(
    volatile __local int *counter,
    __local uint *scratch
    )
{
    int idx = atomic_inc( counter );

    scratch[idx] = get_local_id( 0 );
```

```
}

/**
 * This should only be called by one thread.
 *
 * Preconditions:
 *   counter > 0
 *   rng_state has been initialized
 *
 * @param counter
 *  The same counter variable passed to choose_thread_add_to_options().
 * @param scratch
 *  The same scratch array passed to choose_thread_add_to_options().
 * @param rng_state
 *  The state of the random number generator.
 *
 * @return
 *  The index of one of the threads added with
 *   choose_thread_add_to_options().
 */
uint choose_thread_make_choice(
    volatile __local int *counter,
    __local uint *scratch,
    mwc64x_state_t *rng_state
    )
{
    if ( *counter == 0 )
    {
        // Error. It makes no sense to pick out of
        //  0 options.
        printf( "choose_thread_make_choice: *counter == 0. Returning 0.\n" );
        return 0;
    }
    else if ( *counter == 1 )
    {
        return scratch[0];
    }
    else
    {
        // Choose a random integer between 0 and (counter - 1).
        uint random_integer = MWC64X_NextUint( rng_state );
        uint random_idx = random_integer % *counter;

        return scratch[random_idx];
    }
}
```

societies/util/max_min.cl

```
#ifndef _MAX_MIN_CL
#define _MAX_MIN_CL

/**
 * Efficient max and min functions for OpenCL, all running in lg(n)
 *  time with n threads.
 *
 * Conceptually, this works like making a heap. But since on the GPU
 *  it would take as long to reheapify as just build the heap again
 *  from scratch, we save over 1/2 the __local memory space by
 *  building a tree-like structure that doesn't survive intact afterwards.
 *
 * @author John Kloosterman
 * @date Feb. 9, 2013
 */
```

```
#ifndef NULL
#define NULL ((void *) 0)
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

typedef enum {
    MAX,
    MIN
} MaxMinType;

/**
 * Create the first level of the sort tree by
 *  comparing adjacent items and putting the
 *  index of the larger/smaller in the sort tree.
 *
 * @param type
 *  MAX or MIN
 * @param values
 *  The values the max/min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 * @param use_mask
 *  Whether or not to use @c value_mask to remove elements
 *   from consideration.
 * @param value_mask
 *  The ith element in @c values will not be considered as
 *   a candidate for max/min if value_mask[i] is TRUE.
 */
void max_min_first_pass(
    MaxMinType type,
    __local float *values,
    __local uint *sort_tree,
    int use_mask,
    __local uchar *value_mask
    )
{
    size_t local_id = get_local_id( 0 );
    size_t local_size = get_local_size( 0 );
    size_t sort_tree_size = ( local_size + 1 ) / 2; // The +1 means we round up.

    if ( local_id < sort_tree_size )
    {
        // The index in @gains_per_minute this thread is examining.
        int real_idx = local_id * 2;

        // If there is an odd number of threads, don't read past
        //  the end of the array.
        // It's OK if this is a masked value, we'll catch it in the
        //  next phase.
        if ( ( real_idx + 1 ) >= local_size )
            sort_tree[local_id] = real_idx;
        else
```

```
        {
            // Masking: all that's important is that a masked value can't
            //  win over an unmasked one. So if one is masked, choose the other.
            if ( use_mask
                 && ( value_mask[real_idx] || value_mask[real_idx + 1] ) )
            {
                if ( value_mask[real_idx] )
                    sort_tree[local_id] = real_idx + 1;
                else // value_mask[real_idx + 1]
                    sort_tree[local_id] = real_idx;
            }
            else if ( ( type == MAX
                        && values[real_idx] > values[real_idx + 1] )
                      ||
                      ( type == MIN
                        && values[real_idx] < values[real_idx + 1] ) )
            {
                // Put the index of the maximum of the two values
                //  in this thread's index in @sort_tree.
                sort_tree[local_id] = real_idx;
            }
            else
            {
                // If MAX, this meant that values[real_idx + 1] was larger
                // If MIN
                sort_tree[local_id] = real_idx + 1;
            }
        }
    }

    // Before proceeding, make sure memory is consistent.
    barrier( CLK_LOCAL_MEM_FENCE );
}

/**
 * Find the minimum value from the indices already placed
 *  in @c sort_tree.
 *
 * @param type
 *  MAX or MIN
 * @param values
 *  The values the max/min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 * @param use_mask
 *  Whether or not to use @c value_mask to remove elements
 *   from consideration.
 * @param value_mask
 *  The ith element in @c values will not be considered as
 *   a candidate for max/min if value_mask[i] is TRUE. It must
 *   have the same number of elements as @c values.
 */
void max_min_second_pass(
    MaxMinType type,
    __local float *values,
    __local uint *sort_tree,
    int use_mask,
    __local uchar *value_mask
    )
{
```

```
    size_t local_id = get_local_id( 0 );
    size_t local_size = get_local_size( 0 );
    size_t sort_tree_size = ( local_size + 1 ) / 2; // The + 1 means we round up when
        dividing.

    // How far apart the two values that are being compared are.
    int stride = 1;
    int power_of_two = 2;

    // ( sort_tree_size * 2 ) forces one extra iteration,
    //  which is necessary when the sort tree is not a
    //  power-of-two size.
    while ( power_of_two <= ( sort_tree_size * 2 ) )
    {
        if ( ( local_id < sort_tree_size )
             && ( ( local_id % power_of_two ) == 0 ) )
        {
            if ( ( local_id + stride ) >= sort_tree_size )
            {
                // If the comparison needs a value beyond the
                //  size of the sort tree, keep the current value.
            }
            else if ( use_mask
                      && value_mask[sort_tree[local_id + stride]] )
            {
                // If the other value is masked, keep
                //  the current value.
            }
            else if ( use_mask
                      && value_mask[sort_tree[local_id]] )
            {
                // If this value is masked, take the other value.
                sort_tree[local_id] = sort_tree[local_id + stride];
            }
            else if ( type == MAX
                      && values[sort_tree[local_id + stride]] > values[sort_tree[
                          local_id]] )
            {
                // Make the sort tree at this location equal
                //  to the larger of the two values we are comparing.
                sort_tree[local_id] = sort_tree[local_id + stride];
            }
            else if ( type == MIN
                      && values[sort_tree[local_id + stride]] < values[sort_tree[
                          local_id]] )
            {
                // Make the sort tree at this location equal
                //  to the smaller of the two values we are comparing.
                sort_tree[local_id] = sort_tree[local_id + stride];
            }
        }

        stride *= 2;
        power_of_two *= 2;

        barrier( CLK_LOCAL_MEM_FENCE );
    }
}

// Precondition: there is at least one value that is not masked.

/**
 * Find the index of the maximum/minimum value in
 *  @values.
```

```
 *
 * @param type
 *  MAX or MIN
 * @param values
 *  The values the max/min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 * @param use_mask
 *  Whether or not to use @c value_mask to remove elements
 *   from consideration.
 * @param value_mask
 *  The ith element in @c values will not be considered as
 *   a candidate for max/min if value_mask[i] is TRUE. It must
 *   have the same number of elements as @c values.
 */
uint max_min(
    MaxMinType type,
    __local float *values,
    __local uint *sort_tree,
    int use_mask,
    __local uchar *value_mask
    )
{
    max_min_first_pass( type, values, sort_tree, use_mask, value_mask );
    max_min_second_pass( type, values, sort_tree, use_mask, value_mask );

    return sort_tree[0];
}

/**
 * Find the index of the maximum value in @c values.
 *
 * @param values
 *  The values the max is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 *
 * @return
 *  The index of the maximum value in @c values.
 */
uint max_index(
    __local float *values,
    __local uint *sort_tree
    )
{
    return max_min( MAX, values, sort_tree, 0, NULL );
}

/**
 * Find the index of the minimum value in @c values.
 *
 * @param values
 *  The values the min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
```

```
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 *
 * @return
 *  The index of the maximum value in @c values.
 */
uint min_index(
    __local float *values,
    __local uint *sort_tree
    )
{
    return max_min( MIN, values, sort_tree, 0, NULL );
}

/**
 * Find the indices of the @c n maximum/minimum
 *  values in @c values.
 *
 * @param n
 *  How many maximum/minimum indices to find.
 * @param type
 *  MAX or MIN
 * @param values
 *  The values the min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 * @param results
 *  Where to put the @c n indices. Must have @c n elements.
 * @param mask
 *  Scratch space to create a mask in. It must
 *   have the same number of elements as @c values.
 */
void n_max_min_indices(
    uint n,
    MaxMinType type,
    __local float *values,
    __local uint *sort_tree, // 1/2 the size of values
    __local uint *results,   // size n
    __local uchar *mask       // the size of values
    )
{
    size_t local_id = get_local_id( 0 );

    mask[local_id] = FALSE;
    barrier( CLK_LOCAL_MEM_FENCE );

    for ( int i = 0; i < n; i++ )
    {
        uchar val = max_min(
            type,
            values,
            sort_tree,
            TRUE,
            mask
            );

        if ( local_id == 0 )
        {
            results[i] = val;
```

```
        }
        if ( local_id == val )
        {
            // Don't reuse this value.
            mask[local_id] = TRUE;
        }
        barrier( CLK_LOCAL_MEM_FENCE );
    }
}

/**
 * Find the indices of the @c n maximum
 *  values in @c values.
 *
 * @param n
 *  How many maximum indices to find.
 * @param values
 *  The values the min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 * @param results
 *  Where to put the @c n indices. Must have @c n elements.
 * @param mask
 *  Scratch space to create a mask in. It must
 *   have the same number of elements as @c values.
 */
void n_max_indices(
    uchar n,
    __local float *values,
    __local uint *sort_tree, // 1/2 the size of values
    __local uint *results,   // size n
    __local uchar *mask       // the size of values
    )
{
    return n_max_min_indices(
        n,
        MAX,
        values,
        sort_tree,
        results,
        mask
        );
}

/**
 * Find the indices of the @c n minimum
 *  values in @c values.
 *
 * @param n
 *  How many minimum indices to find.
 * @param values
 *  The values the min is being taken of. It
 *   must have as many elements as threads calling
 *   this function.
 * @param sort_tree
 *  The location of scratch memory for the sort heap.
 *   It must have n / 2 elements, if n is the number
 *   of threads calling this function.
 * @param results
 *  Where to put the @c n indices. Must have @c n elements.
```

```
 * @param mask
 *  Scratch space to create a mask in. It must
 *   have the same number of elements as @c values.
 */
void n_min_indices(
    uchar n,
    __local float *values,
    __local uint *sort_tree, // 1/2 the size of values
    __local uint *results,   // size n
    __local uchar *mask       // the size of values
    )
{
    return n_max_min_indices(
        n,
        MIN,
        values,
        sort_tree,
        results,
        mask
        );
}


#endif
```

societies/util/test/choose_thread_tester.cl.in

```
#include "../choose_thread.cl"

__kernel void choose_thread_tester(
    __global uchar *enableds,
    __global uchar *chosen,
    __global ulong *rng_base_offset
    )
{
    volatile __local int counter;
    __local uint scratch[255];
    size_t local_id = get_local_id( 0 );

    // Seed random number generator
    mwc64x_state_t rng;
    // The documentation ( http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.
        html )
    //  says that 2^40 is a good perStreamOffset if we don't know one.
    // 2^40 = 1099511627776
    MWC64X_SeedStreams( &rng, *rng_base_offset, 128 );

    // Set counter to 0
    if ( local_id == 0 )
    {
        counter = 0;
    }
    barrier( CLK_LOCAL_MEM_FENCE );

    // If our thread is enabled, add ourselves to the list.
    if ( enableds[get_local_id( 0 )] )
    {
        choose_thread_add_to_options( &counter, scratch );
    }
    barrier( CLK_LOCAL_MEM_FENCE );

    // Make a choice.
    __local uchar chosen_thread;
    if ( local_id == 0 )
```

```
    {
        chosen_thread = choose_thread_make_choice( &counter, scratch, &rng );
    }
    barrier( CLK_LOCAL_MEM_FENCE );

    // Return the choice.
    if ( local_id == chosen_thread )
    {
        *chosen = local_id;
    }
}
```

societies/util/test/choose_thread_tester.cpp

```
/**
 * Tester for the functions that randomly choose
 *  a thread.
 *
 * @author John Kloosterman
 * @date Feb. 3, 2013
 */

#define _CPP_11_
#include <CLKernel.h>

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

#define KERNEL_SOURCE "choose_thread_tester.cl"

int main()
{
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());

    CLKernel choose_thread_tester( "choose_thread_tester", src );
    choose_thread_tester.setGlobalDimensions( 256 );
    choose_thread_tester.setLocalDimensions( 256 );

    // Initialize some threads to enabled, others to disabled.
    cl_uchar host_enableds[256];
    for ( int i = 0; i < 256; i++ )
    {
        if ( i % 4 == 0 )
            host_enableds[i] = 1;
        else
            host_enableds[i] = 0;
    }
    CLArgument enableds( host_enableds, 256 );

    cl_uchar host_chosen;
    CLArgument chosen( &host_chosen, 1 );

    cl_ulong host_random_offset;
    srand( time( NULL ) );

    for ( int i = 0; i < 1000; i++ )
    {
        host_random_offset = rand();
```

```
            choose_thread_tester( enableds, chosen, host_random_offset );
            cout << (int) host_chosen << endl;
        }
    }
```

societies/util/test/choose_thread_tester.plot

```
#!/usr/bin/gnuplot

# set style data histogram
set yrange [0:]
plot "choose_thread_tester.out" with boxes notitle
pause -1 "Exit"
```

societies/util/test/choose_thread_tester.sh

```
#!/bin/bash

./choose_thread_tester > choose_thread_tester_raw.out
sort -n choose_thread_tester_raw.out | uniq -c | sed "s/\s*\([0-9]*\) \([0-9]*\)/\2
    \1/" > choose_thread_tester.out
./choose_thread_tester.plot
```

societies/util/test/Makefile

```
CC = clang++
CFLAGS = -std=c++11 -g -Wall
CL_TEST_INCLUDE = -I ../../../cl_test -I /opt/AMDAPP/include
OCL_LIBS = ../../../cl_test/libCLTest.a -L /opt/AMDAPP/lib/x86_64 -lOpenCL
OCL_CC = $(CC) $(CFLAGS) $(CL_TEST_INCLUDE)

all: choose_thread_tester max_min_tester

clean: choose_thread_tester_clean max_min_tester_clean
    rm -f *.out

choose_thread_tester: choose_thread_tester.cl choose_thread_tester.cpp
    $(OCL_CC) -o choose_thread_tester choose_thread_tester.cpp $(OCL_LIBS)

choose_thread_tester.cl: ../choose_thread.cl choose_thread_tester.cl.in
    cpp choose_thread_tester.cl.in -I ../../mwc64x/cl/ > choose_thread_tester.cl

choose_thread_tester_clean:
    rm -f choose_thread_tester.cl choose_thread_tester

max_min_tester: max_min_tester.cpp max_min_tester.cl
    $(OCL_CC) -o max_min_tester max_min_tester.cpp $(OCL_LIBS)

max_min_tester.cl: ../max_min.cl max_min_tester.cl.in
    cpp max_min_tester.cl.in > max_min_tester.cl

max_min_tester_clean:
    rm -f max_min_tester.cl max_min_tester
```

societies/util/test/max_min_tester.cl.in

```
#include "../max_min.cl"

__kernel void max_min_first_pass_tester(
    __global float *global_values,
    __global uint *global_sort_tree
    )
{
```

```
    // There are at most 256 threads in the workgroup, so allocate
    //  memory for the max. It doesn't hurt if the actual size is less.
    __local float values[256];
    __local uint sort_tree[128];

    size_t local_id = get_local_id( 0 );
    values[local_id] = global_values[local_id];
    barrier( CLK_LOCAL_MEM_FENCE );

    max_min_first_pass( MAX, values, sort_tree, 0, NULL );
    barrier( CLK_LOCAL_MEM_FENCE );

    size_t local_size = get_local_size( 0 );
    if ( local_id < ( ( local_size + 1 ) / 2 ) )
        global_sort_tree[local_id] = sort_tree[local_id];
    barrier( CLK_LOCAL_MEM_FENCE );
}

__kernel void max_tester(
    __global float *global_values,
    __global uint *global_sort_tree,
    __global uint *global_max
    )
{
    // There are at most 256 threads in the workgroup, so allocate
    //  memory for the max. It doesn't hurt if the actual size is less.
    __local float values[256];
    __local uint sort_tree[128];

    size_t local_id = get_local_id( 0 );
    values[local_id] = global_values[local_id];
    barrier( CLK_LOCAL_MEM_FENCE );

    uint max = max_index( values, sort_tree );
    if ( get_local_id( 0 ) == 0 )
        *global_max = max;

    size_t local_size = get_local_size( 0 );
    if ( local_id < ( ( local_size + 1 ) / 2 ) )
        global_sort_tree[local_id] = sort_tree[local_id];
    barrier( CLK_LOCAL_MEM_FENCE );
}

// Masks out every other value and takes a min.
__kernel void max_min_mask_tester(
    __global float *global_values,
    __global uint *global_result,
    __global uint *global_result_no_mask
    )
{
    __local uchar mask[256];
    __local float values[256];
    __local uint sort_tree[128];

    size_t local_id = get_local_id( 0 );
    values[local_id] = global_values[local_id];

    // Mask out every odd index.
    if ( local_id % 2 )
        mask[local_id] = TRUE;
    else
        mask[local_id] = FALSE;

    barrier( CLK_LOCAL_MEM_FENCE );
```

```
    uint mask_result = max_min(
        MIN,
        values,
        sort_tree,
        TRUE,
        mask
    );
    if ( get_local_id( 0 ) == 0 )
        *global_result = mask_result;

    uint result = max_min(
        MIN,
        values,
        sort_tree,
        FALSE,
        NULL
        );
    if ( get_local_id( 0 ) == 0 )
        *global_result_no_mask = result;
}

// Assume n=6.
__kernel void n_max_indices_tester(
    __global float *host_values,
    __global uint *host_results
    )
{
    __local float values[255];
    __local uint sort_tree[128];
    __local uint results[6];
    __local uchar mask[255];

    size_t local_id = get_local_id( 0 );
    values[local_id] = host_values[local_id];
    barrier( CLK_LOCAL_MEM_FENCE );

    n_max_indices(
        6,
        values,
        sort_tree,
        results,
        mask
        );

    if ( local_id < 10 )
    {
        printf( "Sort tree %2d: %d\n", local_id, sort_tree[local_id]);
    }

    printf( "Mask %2d: %d\n", local_id, (int)mask[local_id] );

    if ( local_id < 6 )
    {
        host_results[local_id] = results[local_id];
    }
}
```

societies/util/test/max_min_tester.cpp

```
/**
 * Tests the parallel max function used to find
 *  maximum gain per minute.
 */
```

```
#define _CPP_11_
#include <CLKernel.h>
#include <fstream>
#include <iostream>
using namespace std;

#define KERNEL_SOURCE "max_min_tester.cl"

void test_first_pass( string src )
{
    cout << "Testing max_index_first_pass()... ";

    CLKernel max_min_first_pass_tester( "max_min_first_pass_tester", src );
    max_min_first_pass_tester.setGlobalDimensions( 8, 1 );
    max_min_first_pass_tester.setLocalDimensions( 8, 1 );
    cl_uint *tree;

    // Power of two
    cl_float host_floats1[] =
        { 24, 6.333, 2.1, 9.76, 2.11, 44.2224, 11000, 23 };
    tree = new cl_uint[4];

    CLArgument floats1( host_floats1, 8 );
    CLArgument floats1_sort_tree( tree, 4 );
    max_min_first_pass_tester( floats1, floats1_sort_tree );

    cl_uint host_floats1_tree[4] =
        { 0, 3, 5, 6 };
    for ( int i = 0; i < 4; i++ )
        assert( tree[i] == host_floats1_tree[i] );
    delete[] tree;
    cout << "* " << flush;

    // Array size an odd number.
    cl_float host_floats2[] =
        { 54.2, 9000, 223.5, 222, 43 };
    tree  = new cl_uint[3];

    CLArgument floats2( host_floats2, 5 );
    CLArgument floats2_sort_tree( tree, 3 );
    max_min_first_pass_tester.setGlobalDimensions( 5, 1 );
    max_min_first_pass_tester.setLocalDimensions( 5, 1 );
    max_min_first_pass_tester( floats2, floats2_sort_tree );

    cl_uint host_floats2_tree[3] =
        { 1, 2, 4 };

    for ( int i = 0; i < 3; i++ )
    {
        assert( tree[i] == host_floats2_tree[i] );
    }
    delete[] tree;
    cout << "* " << flush;
    cout << "All tests passed." << endl << flush;
}

void test_function( string src )
{
    cout << "Testing max_index()... ";

    CLKernel max_tester( "max_tester", src );
    cl_uint host_max;
    CLArgument max( &host_max, 1 );
```

```
    // Nice power of two
    cl_float host_floats1[] =
        { 24, 6.333, 2.1, 9.76, 2.11, 44.2224, 11000, 23 };
    cl_uint host_floats1_tree[4];

    CLArgument floats1( host_floats1, 8 );
    CLArgument floats1_tree( host_floats1_tree, 4 );
    max_tester.setGlobalDimensions( 8, 1 );
    max_tester.setLocalDimensions( 8, 1 );
    max_tester( floats1, floats1_tree, max );

    cl_uint floats1_correct_tree[4] = { 6, 3, 6, 6 };
    for ( int i = 0; i < 4; i++ )
    {
        assert( host_floats1_tree[i] == floats1_correct_tree[i] );
    }
    assert( host_max == 6 );
    cout << "* ";

    // Odd number
    cl_float host_floats2[] =
        { 54.2, 43, 223.5, 222, 9000 };
    cl_uint host_floats2_tree[3];

    CLArgument floats2( host_floats2, 5 );
    CLArgument floats2_sort_tree( host_floats2_tree, 3 );
    max_tester.setGlobalDimensions( 5, 1 );
    max_tester.setLocalDimensions( 5, 1 );
    max_tester( floats2, floats2_sort_tree, max );

    cl_uint floats2_correct_tree[3] =
        { 4, 2, 4 };

    for ( int i = 0; i < 3; i++ )
    {
        assert( host_floats2_tree[i] == floats2_correct_tree[i] );
    }
    assert( host_max == 4 );
    cout << "* ";

    cout << "All tests passed." << endl << flush;
}

void test_mask( string src )
{
    cout << "Testing masking... ";

    CLKernel mask_tester( "max_min_mask_tester", src );
    cl_uint host_min;
    CLArgument min( &host_min, 1 );
    cl_uint host_min_no_mask;
    CLArgument min_no_mask( &host_min_no_mask, 1 );

    cl_float host_values[20] =
        {
            -27.342,
            73.887,  // masked
            -61.280,
            88.231, // masked
            -57.100,
            -564.927, // masked, attractive min
            -42.691,
            -97.121, // masked
```

```
            -2.643,
            8.741, // masked
            18.538,
            26.442, // masked
            36.100,
            42.445, // masked
            -104.102,  // actual min
            48.808, // masked
            63.522,
            -58.05, // masked
            69.232,
            100.676 // masked
        };

    CLArgument values( host_values, 20 );
    mask_tester.setGlobalDimensions( 20, 1 );
    mask_tester.setLocalDimensions( 20, 1 );
    mask_tester( values, min, min_no_mask );


    assert( host_min == 14 );
    cout << "* " << flush;
    assert( host_min_no_mask == 5 );
    cout << "* " << flush;

    cout << "All tests passed!" << endl << flush;
}

void test_n_max_indices( string src )
{
    cout << "Testing n_max_indices()... ";

    CLKernel n_max_indices_tester( "n_max_indices_tester", src );
    cl_uint host_results[6];
    CLArgument results( host_results, 6 );

    cl_float host_values[20] =
        {
            -27.342,
            73.887,
            -61.280,
            88.231,
            -57.100,
            -564.927,
            -42.691,
            -97.121,
            -2.643,
            8.741,
            18.538,
            26.442,
            36.100,
            42.445,
            -104.102,
            48.808,
            63.522,
            -58.05,
            69.232,
            100.676
        };

    CLArgument values( host_values, 20 );
    n_max_indices_tester.setGlobalDimensions( 20, 1 );
    n_max_indices_tester.setLocalDimensions( 20, 1 );
    n_max_indices_tester( values, results );
```

```
    assert( host_results[0] == 19 );
    assert( host_results[1] == 3 );
    assert( host_results[2] == 1 );
    assert( host_results[3] == 18 );
    assert( host_results[4] == 16 );
    assert( host_results[5] == 15 );
    cout << "* " << flush;

    // Duplicate values; make sure they're not reused.
    for ( int i = 0; i < 20; i++ )
        host_values[i] = 5;
    n_max_indices_tester( values, results );

    cout << endl;
    for ( int i = 0; i < 6; i++ )
    {
        cout << host_results[i] << " ";
    }
    cout << endl;

    cout << "All tests passed!" << endl << flush;
}

int main ( void )
{
    cout << "max_min_tester.cpp: Test the maximum and minimum functions and
        dependencies." << endl;

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());

    test_first_pass( src );
    test_function( src );
    test_mask( src );
    test_n_max_indices( src );
}
```

societies/util/test/n_minimum_tester.cl.in

```
/**
 * Tester for the n_minimum function.
 */

#include "../n_minimum.cl"

__kernel void
n_minimum_tester(
    __global uchar *n,
    __global float *values,
    __global float *mins,
    __global uchar *indices,
    __local float *local_mins,
    __local uchar *local_indices,
    __local int *locks
    )
{
    size_t local_id = get_local_id( 0 );

    n_minimum( *n, values[local_id], local_mins, local_indices, locks );
    barrier( CLK_LOCAL_MEM_FENCE );
```

```
    // Copy local to global values.
    if ( local_id < *n )
    {
        mins[local_id] = local_mins[local_id];
        indices[local_id] = local_indices[local_id];
    }
}
```

societies/util/test/n_minimum_tester.cpp

```
/**
 * Tests the parallel n_minimum function
 */

#define _CPP_11_
#include <CLKernel.h>
#include <fstream>
#include <iostream>
using namespace std;

#define KERNEL_SOURCE "n_minimum_tester.cl"

int main ( void )
{
    cout << "max_index_tester.cpp: Test the max_index() function and dependencies." <<
        endl;

    // Open OpenCL kernel
    ifstream t( KERNEL_SOURCE );
    string src((std::istreambuf_iterator<char>(t)),
               std::istreambuf_iterator<char>());

    CLKernel n_minimum_tester( "n_minimum_tester", src );

    cl_uint host_n = 6;
    cl_float host_values1[20] = {
        88.231,
        -84.102,
        -58.05,
        69.232,
        48.808,
        8.741,
        64.927,
        -2.643,
        -57.100,
        -42.691,
        73.887,
        100.676,
        63.522,
        18.538,
        42.445,
        36.100,
        -97.121,
        26.442,
        -61.280,
        -27.342
    };
    cl_float host_mins1[6];
    cl_uchar host_indices1[6];

    n_minimum_tester.setLocalArgument( 4, sizeof( cl_float ) * 6 );
    n_minimum_tester.setLocalArgument( 5, sizeof( cl_uchar ) * 6 );
    n_minimum_tester.setLocalArgument( 6, sizeof( cl_int ) * 6 );
```

```
    CLArgument values1( host_values1, 20 );
    CLArgument n( &host_n , 1 );
    CLArgument mins1( host_mins1, 6 );
    CLArgument indices1( host_indices1, 6 );

    n_minimum_tester.setGlobalDimensions( 20, 1 );
    n_minimum_tester.setLocalDimensions( 20, 1 );
    n_minimum_tester( n, values1, mins1, indices1 );

    for ( int i = 0; i < 6; i++ )
    {
        cout << host_mins1[i] << ", index " << (int) host_indices1[i] << endl;
    }
}
```

societies/util/types.h

```
#ifndef _TYPES_H
#define _TYPES_H

/**
 * Defines the types to be used in any structure
 *  passed between host and OpenCL code. These
 *  guarantee the correct alignment and naming for
 *  both worlds.
 *
 * @author John Kloosterman
 * @date Jan. 17, 2013
 */

#ifdef _OPENCL_
/* Types used in OpenCL code */

#define CL_INT int
#define CL_FLOAT_T float
#define CL_UCHAR uchar

#else
/* Types used in host code */
#include <CL/cl.hpp>

#define CL_INT cl_int
#define CL_FLOAT_T cl_float
#define CL_UCHAR cl_uchar

#endif

#endif
```