

Санкт-Петербургский национальный исследовательский университет  
информационных технологий, механики и оптики

Кафедра компьютерных технологий

А. И. Бочкарев

Портирование генетических алгоритмов на  
платформу *OpenCL* на примере генерации  
автомата в задаче „Умный муравей“

Курсовая работа

Санкт-Петербург  
2011

## Оглавление

Оглавление.....	2
Введение.....	3
Глава 1. Постановка задачи.....	4
Описание задачи.....	4
Описание генетического алгоритма.....	4
Глава 2. Особенности реализации.....	5
Реализация на центральном процессоре.....	5
Реализация для графического устройства.....	5
Основная программа.....	5
Глава 3. Результаты.....	6
Глава 4. Выводы.....	8
Источники.....	9

## Введение

*OpenCL* — программное обеспечение для написания компьютерных программ, связанных с параллельными вычислениями на различных графических (англ. GPU) и центральных (англ. CPU) процессорах. Разработчику предоставляется возможность из своего приложения вызвать некоторый программный код, называемый «ядром» (англ. *Kernel*). Код ядра базируется на подмножестве языка C, с некоторыми расширениями.

Код ядра запускается одновременно в несколько потоков, различающихся только своим номером, который можно узнать при выполнении. Таким образом, параллельно можно запускать только одинаковые ядра, при этом отличия в их поведении могут базироваться лишь на номерах. Такая модель вычислений хорошо подходит для выполнения расчетов над множеством однотипных объектов, например, расчетом нового поколения или вычислением функции приспособленности в генетическом алгоритме. Данная работа посвящена реализации такого алгоритма и сравнению скорости его выполнения со стандартной реализацией на центральном процессоре.

# Глава 1. Постановка задачи

## Описание задачи

В последнее время стали популярными технологии вычислений на графических процессорах. Поэтому возникает вопрос о применимости этих технологий для генетических алгоритмов.

Задача настоящей работы состоит в экспериментальном сравнении реализаций генетических алгоритмов с использованием технологий *OpenCL* и стандартной многопоточности. В качестве примера для исследования взята генерация автомата в известной задаче об «Умном муравье» [1].

Постановка задачи подразумевает:

- небольшие расходы на модификацию кода для запуска на видеокарте;
- изучение рациональности приложенных усилий по переносу кода на видеокарту;
- выполнение абсолютно всех вычислений на графическом процессоре;
- изучение полученных результатов;
- рекомендации для последующих реализаций.

## Описание генетического алгоритма

Автомат представлен полной таблицей переходов, этот вариант наиболее прост для реализации на графическом процессоре, кроме того, код для видеокарты практически не будет отличаться от кода для центрального процессора.

Для генерации нового поколения особей используется клеточный генетический алгоритм. Все особи расположены в таблице заданного размера, противоположные стороны замыкаются между собой. В каждой клетке таблицы расположена только одна особь. Следует отметить, что клеточный алгоритм идеально вписывается в логику вычислений на видеокартах.

Новое поколение получается путем скрещивания каждой особи с четырьмя своими соседями (справа, слева, сверху, снизу), а также случайными мутациями. Из полученных в результате потомков выбирается лучший и помещается в таблицу вместо своего родителя.

В качестве функции приспособленности взята стандартная для задачи об «Умном муравье» — число съеденных продуктов питания на заданной карте.

## Глава 2. Особенности реализации

### Реализация на центральном процессоре

Реализация для центрального процессора совершенно стандартная, но имеет некоторые особенности, которые были внесены для применения многопоточности и для упрощения переноса на видеокарту. Вот, некоторые из них:

- код написан на смеси языков *C*, *C++* для упрощения переноса на видеокарту;
- реализована своя функция для генерации случайных чисел, так как стандартная не применима в многопоточных приложениях.

Многопоточность реализована с помощью библиотеки *Boost*. Таблица с особями разделяется на заданное число полос и каждый поток обрабатывает только свою полосу. В итоге имеется возможность полностью загрузить процессор, состоящий из нескольких ядер, при этом мы получаем ускорение пропорциональное числу ядер.

### Реализация для графического устройства

Реализация для видеокарты скопирована с реализации для центрального процессора, но имеет множество своих особенностей, явившихся результатом непереносимости некоторых фрагментов, а также различными ускорениями программы. К таковым можно отнести:

- код является подмножеством стандарта *C99*, поэтому убраны все объектно-ориентированные возможности [2];
- реализована функция для генерации случайных чисел, так как в *OpenCL* она не предусмотрена;
- все динамические выделения памяти заменены на работу с буфером;
- для уменьшения объемов передаваемой памяти между основной программой и устройством реализован подсчет статистики прямо на устройстве;
- реализована возможность вычисления заданного числа поколений без пересылки данных к основной программе, статистика передается по завершению вычислений;
- применены специальные функции и изменены некоторые участки кода, для ускорения программы.

### Основная программа

Кроме различных реализаций генетического алгоритма создана лаборатория для управления и визуального контроля за ростом поколений. Так же имеется возможность просмотра статистики поколений и лучших особей. Просмотр можно осуществлять не останавливая процесс выращивания особей.

## Глава 3. Результаты

Для исследования задачи использовалось число поколений, полученных за определенное время, а так же качество полученного результата. Входными параметрами являлись размеры таблицы для клеточного алгоритма, число потоков (число вычисляемых поколений за один запуск для GPU), число состояний в автомате.

Исследования проводились на тестовой машине:

- Процессор: *AMD Phenom II X4 955 — 3.20 GHz*;
- ОЗУ: *8.00 Gb*;
- Видеокарта: *AMD Radeon 6850*.

Далее скорости представлены в виде числа выращенных поколений за 30 секунд выполнения. В таблице приведены полученные результаты измерений.

Таблица 1. Первые результаты измерений

Размер поколения	<i>OpenCL</i> на GPU	<i>OpenCL</i> на CPU	5 потоков CPU
25	5700	21500	6000
50	5200	10800	3000
80	4900	6750	1900
100	4800	5400	1530
140	4700	3850	1100
180	4300	3000	850

При вычислениях на GPU исследования проводились с константами 10 и 20 для числа поколений, вычисленных перед пересылкой данных от видеокарты к основной программе. Существенной разницы в скорости замечено не было. Число один не использовалось в виду неэффективности, выявленной на ранней стадии профилирования программы.

Стоит заметить, что все устройства за одинаковое число смен поколений получали схожие по качеству результаты. Незначительная разница могла появляться только из-за разницы в генераторе случайных чисел, в остальном же реализации были идентичны. Поэтому не будет ошибкой связывать эффективность вычислений с их скоростью.

Далее, для большей наглядности, в диаграмме приведены результаты в графическом виде.

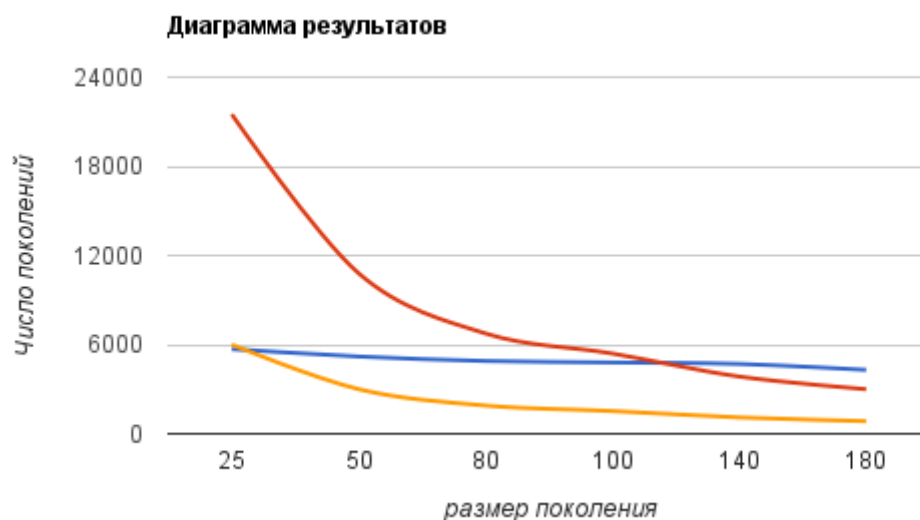


Рис 1. Первые результаты измерений  
Красный — *OpenCL* (CPU), синий — *OpenCL* (GPU), желтый — CPU (5 потоков)

Как можно заметить, кривая результатов для видеокарты близка к горизонтальной линии. Это связано с тем, что графический процессор обладает большим числом вычислительных устройств, и реализация оказалась хорошо масштабируемой.

При дальнейших ускорениях программы было внесено множество изменений в исходный код программы, также были обновлены драйвера *OpenCL*. Были получены следующие результаты:

Таблица 2. Сравнение исполнения *OpenCL* на разных устройствах.

Размер поколения	<i>OpenCL</i> на GPU	<i>OpenCL</i> на CPU
25	8740	105000
50	7800	52700
80	7620	33000
100	6940	26400
140	6500	18800
180	5760	14600
256	5360	11200

Из таблицы видно, что после произведенных изменений было получено некоторое ускорение при запуске на видеокарте, а вот при запуске на центральном процессоре получены результаты в несколько раз лучше предыдущих. Скорее всего, низкие результаты первых запусков на CPU являются следствием ошибок в компиляторе, возможно, раньше не использовались все ядра процессора. К сожалению, сейчас проверить этот факт проблематично.

## Глава 4. Выводы

Из полученных результатов можно сделать вывод, что технология *OpenCL* еще недостаточно хорошо реализована на конкретных устройствах, но стоит отметить, что идут постоянные усовершенствования. Таким образом, на данный момент нельзя сделать точный вывод об эффективности вычислений на GPU, этот вопрос требует дальнейшего изучения и детального анализа всех усовершенствований, вносимых в реализацию компиляторов.

Вычисления на графических устройствах являются перспективной областью и хорошей возможностью ускорить вычисление многих алгоритмов, но подходить к таким задачам нужно аккуратно, так как эта область еще недостаточно хорошо изучена и постоянно совершенствуется. В ближайшее время будет мы увидим не только множество работ на эту тему, но и множество новых устройств с поддержкой технологии *OpenCL*.



## ИСТОЧНИКИ

1. Бедный Ю. Д., Шалыто А. А. Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей»  
[http://is.ifmo.ru/works/\\_ant.pdf](http://is.ifmo.ru/works/_ant.pdf)
2. Khronos OpenCL Working Group: [The OpenCL Specification, Version 1.1, 2011](#)
3. AMD: [AMD Accelerated Parallel Processing OpenCL™ Programming Guide \(v1.3f\)](#), 2011