Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра компьютерных технологий

А.Г. Банных
С.С. Поромов

Анализ эффективности реализации генетических алгоритмов при использовании вычислений на видеокартах

Курсовая работа

Санкт-Петербург
2011

Оглавление

Куда, куда... вы удалились...

А.С. Пушкин

Где? Когда? Куда? Откуда? Почему? Зачем? и Как?

Введение

В последнее время большое распространение получают технологии, позволяющие использовать графические процессоры видеокарт (*Graphical Processing Unit, GPU*) для различных вычислений. Причиной этого является то, что стало сложнее повышать производительность центральных процессоров (*Central Processing Unit, CPU*) путем увеличения тактовой частоты. Поэтому основным способом повышения производительности стали параллельные вычисления.

Многие генетические алгоритмы требуют больших объемов вычислений, которые можно выполнять параллельно (например, вычисления значений функции приспособленности различных особей). В настоящей работе анализируются две задачи (о расстановке ферзей и о нахождении нулевого битого вектора) и две технологии (*CUDA* и *OpenCL*) параллельных вычислений с использованием видеокарт.

Глава 1. Обзор технологии

В связи с особенностями архитектуры, видеокарты имеют значительно больше вычислительных элементов (ядер), чем центральный процессор. Правда, сами ядра имеют сравнительно простую архитектуру и для программирования их необходимо использовать специализированные языки программирования и среды разработки (*Software Development Kit, SDK*).

1.1. Историческая справка

В 2007 г. вместе с выходом видеокарт *GeForce* восьмого поколения компания *Nvidia* представила технологию *CUDA* (*Compute Unified Device Architecture*), позволяющую реализовывать на упрощенном диалекте языка *C++* программы для выполнения их на графических процессорах новых видеокарт.

В конце 2008 г. международная организация *Khronos Group* представила первую спецификацию открытого стандарта *OpenCL* (*Open Computing Language*) — универсального языка для параллельных вычислений, как на видеокартах, так и на центральных процессорах и других вычислительных устройствах. Этот стандарт во многом похож на *CUDA*, а используемый им язык является некоторым подмножеством языка *C*.

В середине 2009 г. компания *AMD* опубликовала первую версию *ATI Stream SDK v2.0*, позволяющую использовать язык *OpenCL* для графических процессоров *AMD* и центральных процессоров с архитектурой *x86*. В конце 2009 г. *Nvidia* также включила поддержку *OpenCL* в технологию *CUDA*.

В конце 2009 г. компания *Microsoft* представила свой вариант технологии — *DirectCompute*, который вошел как часть в *DirectX 11*. В настоящее время эта технология еще не получила большого распространения, и поэтому не будет рассматриваться в данной работе.

Остановимся подробнее на рассматриваемых технологиях.

1.2. Технология *CUDA*

Данная технология используется для разработки продуктов для эффективного использования видеокарт компании *Nvidia*. Разработчику предлагается широкий выбор средств — от возможности написания программ с интуитивно понятным синтаксисом, подобным языку *C++*, до низкоуровневой работы с драйвером видеокарты.

Эта технология, несмотря на свою новизну, имеет широкое распространение. Ее возможности хорошо исследованы и она нашла применение во многих областях. В частности, существуют работы, посвященные реализации генетических алгоритмов на видеокартах с использованием этой технологии [1].

Достоинством рассматриваемой технологии является то, что при разработке приложения есть возможность учесть аппаратные особенности графических чипов *Nvidia*. Модель памяти в *CUDA* довольно сложна и эффективное использование ресурсов дает значительный прирост производительности.

1.3. *OpenCL* (*ATI Stream SDK v2.0*)

Разработчику предоставляется возможность из программы, написанной на языке *C++*, вызывать некоторый программный код, называемый «ядром» (*kernel*). Кроме этого языка возможна поддержка других языков. Ядро должно быть написано на подмножестве языка C с некоторыми расширениями. В основном, расширения связаны с особенностями архитектуры графических процессоров.

Код ядра запускается одновременно в несколько потоков, различающихся только своим номером, который можно узнать при исполнении. Таким образом, получается, что параллельно можно запускать только одинаковые ядра, при этом отличия в их поведении могут базироваться лишь на их номерах. Такая организация вычислений хорошо подходит для расчета функций приспособленности в генетических алгоритмах.

Обмен данными между основной программой и ядрами происходит с помощью специальных функций загрузки-выгрузки, которые являются аналогами обмена данными между *CPU* и *GPU*.

Особенностью *OpenCL* является его универсальность — код можно исполнять как на видеокартах *AMD*, так и на любом современном x86-процессоре с использованием всех ядер этого процессора. При этом не потребуется почти ничего изменять в коде программ. В дальнейшем в настоящей работе будет применяться и исследоваться такая возможность.

Однако, в настоящее время драйвера для этой достаточно молодой технологии еще не до конца оптимизированы, и поэтому она проигрывает более распространенной и изученной технологии *CUDA*. Компания *AMD* обещает каждые полгода или чаще выпускать новые версии своих средств разработки, которые должны будет значительно увеличивать производительность.

Глава 2. Постановка задачи

Технологии вычисления на видеокартах хорошо зарекомендовали себя во многих областях [2]. Поэтому возникает вопрос о применимости этих технологий для реализации генетических алгоритмов.

Существуют работы, посвященные исследованию применимости технологии *CUDA* для генетических алгоритмов, например, [1, 3]. Кроме того, некоторые исследователи пытались совместить использование распределенных систем и технологию *CUDA* [4]. Во многих случаях был получен значительный прирост производительности по сравнению с традиционной реализацией.

В то же время, некоторые исследователи скептически относятся к столь большому приросту производительности. В частности, в статье [5] показано, что во многих случаях эффективное использование ресурсов современных процессоров позволяет сократить разрыв в производительности до 2,5 раз.

Задача настоящей работы состоит в экспериментальном исследовании реализации генетических алгоритмов с использованием технологий *CUDA* и *OpenCL*. Постановка задачи подразумевает:

1. Отказ от рассмотрения распределенных систем.
2. Небольшие расходы на модификацию кода для последующего запуска на видеокарте.
3. Предсказание рациональности приложенных усилий по переносу кода на видеокарту.
4. Обозначение возможных проблем.
5. Сравнение технологий *CUDA* и *OpenCL*.

Глава 3. Задачи

В качестве задач генетического программирования рассмотрены две простые задачи, в которых можно легко добиться того, чтобы основное время работы занимали параллельное вычисление значений функций приспособленности различных особей, операций скрещивания и мутации.

3.1. Задача про расстановку ферзей

На шахматном поле $N \times N$ необходимо расставить N ферзей таким образом, чтобы они не били друг друга.

Каждая отдельная особь (кандидат на решение) представляет собой перестановку, в которой i-ый элемент обозначает в какой строке находится ферзь, расположенный в i-ом столбце шахматной доски. Таким образом, достигается то, что никакие два ферзя не бьют друг друга по горизонтали или вертикали и достаточно добиться того, чтобы они не били друг друга по диагонали.

Функция приспособленности особи равна числу пар бьющих друг друга ферзей. Тогда задача сводиться к поиску особи со значением функции приспособленности, равным нулю.

В качестве мутации используется обмен двух элементов перестановки. Скрещивание же происходит по специальному алгоритму скрещивания двух перестановок, основанному на том, что по очереди для каждого элемента берется один из двух циклов, в которых он представлен в перестановках, соответствующих скрещиваемым особям.

Из каждого поколения несколько лучших особей автоматически переходят в следующее поколение — используется элитизм.

При решении данной задачи будет использоваться способ вычисления функции приспособленности со временем работы $O(N^2)$ — перебираются все

пары элементов перестановки. Если в поколении M особей, то на вычисление всех функций приспособленности тратится $O(MN^2)$ единиц времени. На остальные операции (скрещивание и мутация) требуется $O(MN)$ единиц времени.

Поэтому при увеличении размера поля большую часть времени алгоритм может выполняться параллельно, что и будет использоваться в дальнейшем. Существуют способы вычисления каждой функции приспособленности за $O(N)$, но они не используются в данной работе, так как тогда эта часть вычислений будет сравнима по времени с остальной, непараллельной частью.

В то же время, одной из проблем вычисления такой функции приспособленности на видеокарте может стать то, что в процессе вычисления необходимо выполнить большое число операций ветвления, которые выполняются на графическом процессоре значительно дольше, чем арифметические. Данная проблема хорошо известна, и для борьбы с ней были разработаны различные подходы, такие как выполнение всех возможных случаев с последующим выбором необходимого. К сожалению, данный метод применим не всегда, и его рассмотрение выходит за рамки данной работы. Следует также отметить, что современные процессоры оптимизированы для эффективной работы с операциями ветвления. Поэтому различия в скорости работы на коде с большим числом условных переходов весьма велики [6].

3.2. Задача про нулевой битовый вектор

В качестве второй задачи была выбрана проблема поиска нулевого битового вектора. Генетические алгоритмы решения этой задачи хорошо изучены и для некоторых из них получены оценки времени работы [7]. Также она проста в реализации и используемый алгоритм ее решения имеет свойства, характерные для многих генетических алгоритмов:

1. Представление особи в виде битовой строки.
2. Малое число логических операций при вычислении функции приспособленности, выполнении мутации и скрещивания.
3. Последовательный доступ к памяти.

Операция мутации стандартна для таких особей — изменение значения одного случайного бита. В качестве операции скрещивания используется одноточечный кроссовер.

Функцией приспособленности особи является число единиц в ней. Соответственно, необходимо вывести идеальную особь с нулевой функцией приспособленности. Существует алгоритм, позволяющий вычислять число единиц в 32-битном числе, используя только арифметические операции.

Таким образом, сопоставляя результаты по данным двум задачам, выполним сравнение эффективности выполнения различных видов операций на графическом процессоре.

Также при решении этой задачи выполнение операций мутации и скрещивания особей происходило параллельно.

Глава 4. Используемые вычислительные устройства

4.1. Конфигурация тестовой машины для *OpenCL*

Intel Core 2 Quad Q9400 (2.66GHz), 4GB RAM — четыре ядра, пиковая производительность около 40 GFLOP/s, при использовании одного ядра 10 GLOP/s, *ATI Radeon HD 4650* — 320 унифицированных ядер, около 400 GFLOP/s, *Windows 7 x64*, компилятор *MS Visual Studio 2008* в release режиме.

4.2. Тестовая машина для *CUDA*

Intel Core 2 Duo P8400 (2.26Ghz), 4GB RAM — два ядра, около 15 GFLOP/s, *Nvidia GeForce 9300m GS* — восемь потоков, около 30 GFLOP/s, *Gentoo Linux x64*, компилятор *gcc 4.3.5* с ключом оптимизации -O3.

4.3. Вторая тестовая машина для *CUDA*

Intel Core 2 Duo E7400 (2.8 Ghz), 4GB RAM — два ядра, около 20 GFLOP/s, *Nvidia GeForce GTX460 1Gb* — 336 потоков, 907.2 GFLOP/s, *Debian Linux x64*, компилятор *gcc 4.3*

4.4. Некоторые замечания по используемым устройствам

- В 4xxx поколении видеокарт *ATI* вследствие особенностей архитектуры памяти отсутствует локальная память у вычислительных ядер, потому в качестве нее используется глобальная память видеокарты и с этим связаны значительные задержки при вычислениях. Эта проблема отсутствует в более поздних поколениях видеокарт *ATI*.
- Используемые видеокарты сильно различаются по заявленной пиковой производительности.

Глава 5. Результаты

Для каждой задачи исследовалось среднее время, потраченное на получение нового поколения для различных размеров задачи и числа особей в поколении.

Для этого запускалось несколько итераций получения очередного поколения (около 100 – 1000 запусков) и общее время, потраченное на всю работу алгоритма, делилось на число полученных поколений.

При каждых параметрах задач сравнивалась скорость для однопоточного вычисления на центральных процессорах, многопоточного вычисления на четырехъядерном процессоре *Intel Core 2 Quad Q9400* с использованием *OpenCL*, многопоточного вычисления на графическом процессоре видеокарты *ATI Radeon HD 4650* с использованием *OpenCL*, на графических процессорах видеокарт *Nvidia GeForce 9300m GS* и *Nvidia GeForce GTX460* с использованием *CUDA*.

5.1. Задача про расстановку ферзей

Для этой задачи варьировались размеры поля (число ферзей) и число особей в поколении.

По полученным результатам можно заметить, что при малом размере поля использование мощной видеокарты может принести значительный прирост производительности (рис. 5.3).

С увеличение размера поля вычисления на видеокарте начинают сдавать позиции. На первое место выходит полноценное использование ресурсов центрального процессора посредством технологии *OpenCL* (рис. 5.2).

При этом в одинаковых условиях видеокарта *ATI HD 4650* проигрывает центральному процессору больше, чем видеокарта *Nvidia 9300m GS*, а процессор *P8400*, несмотря на более низкую тактовую частоту выигрывает у

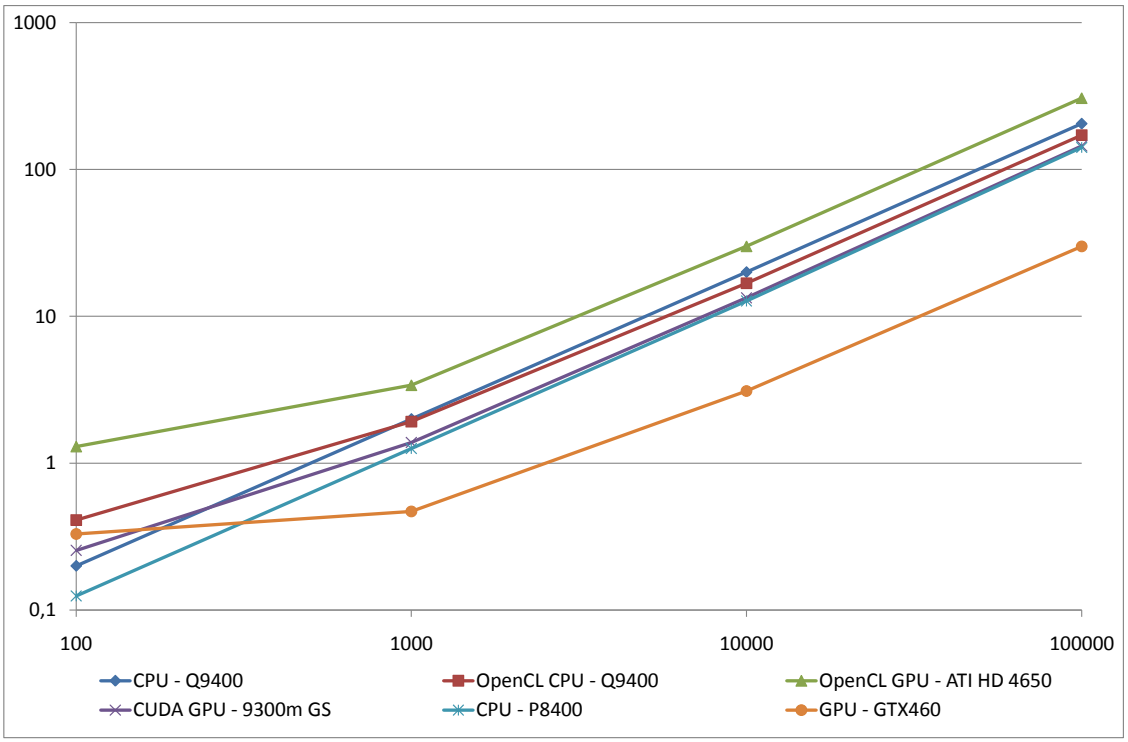

Рис. 5.1. Время генерации одного поколения в зависимости от размера поколения, задача о расстановки ферзей, N=20, значения приведены в миллисекундах

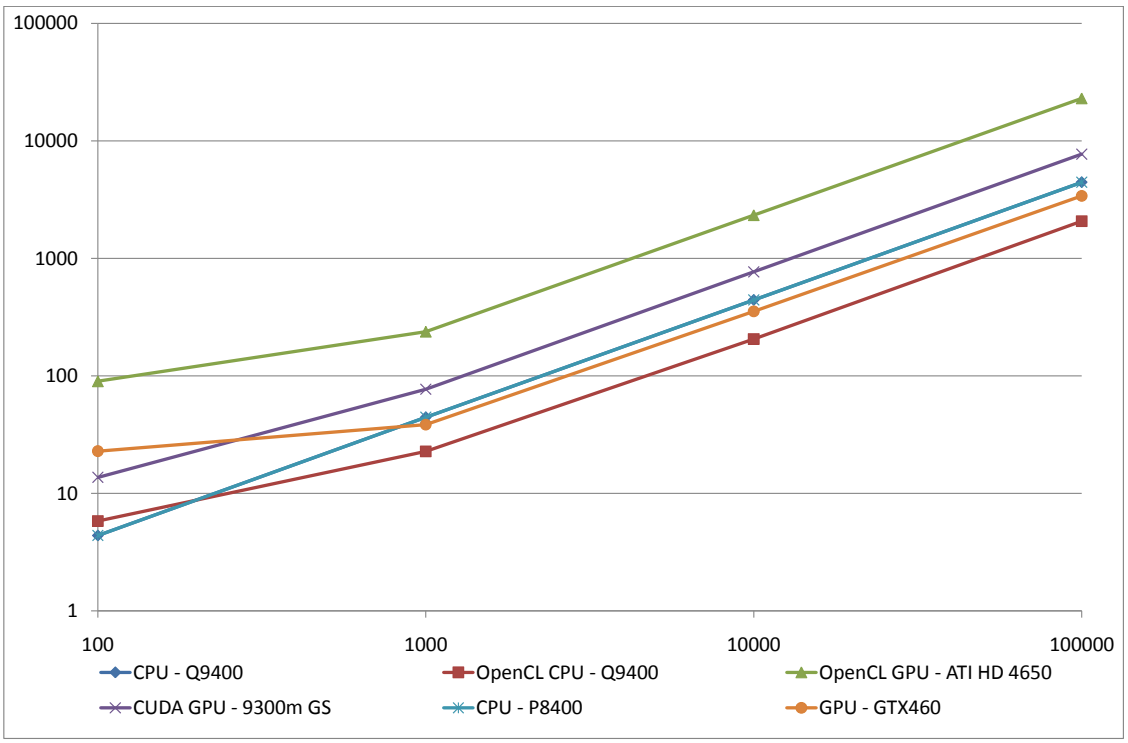

Рис. 5.2. Время генерации одного поколения в зависимости от размера поколения, задача о расстановки ферзей, N=200, значения приведены в миллисекундах

Q9400, что можно обусловить различиями в используемых компиляторах.

В то же время уже сравнительно мощная видеокарта *Nvidia GTX460* выигрывает у центрального процессора при малых размерах поля примерно в пять раз. При этом выигрыш по сравнению со значительно (в 30 раз) менее мощной видеокартой *Nvidia 9300m GS* варьируется в диапазоне от трех до шести раз.

5.2. Задача про нулевой битовый вектор

Для этой задачи варьировалось N — число 32-битных целых в массиве (число бит в векторе в 32 раза превосходит размер массива) и число особей в поколении.

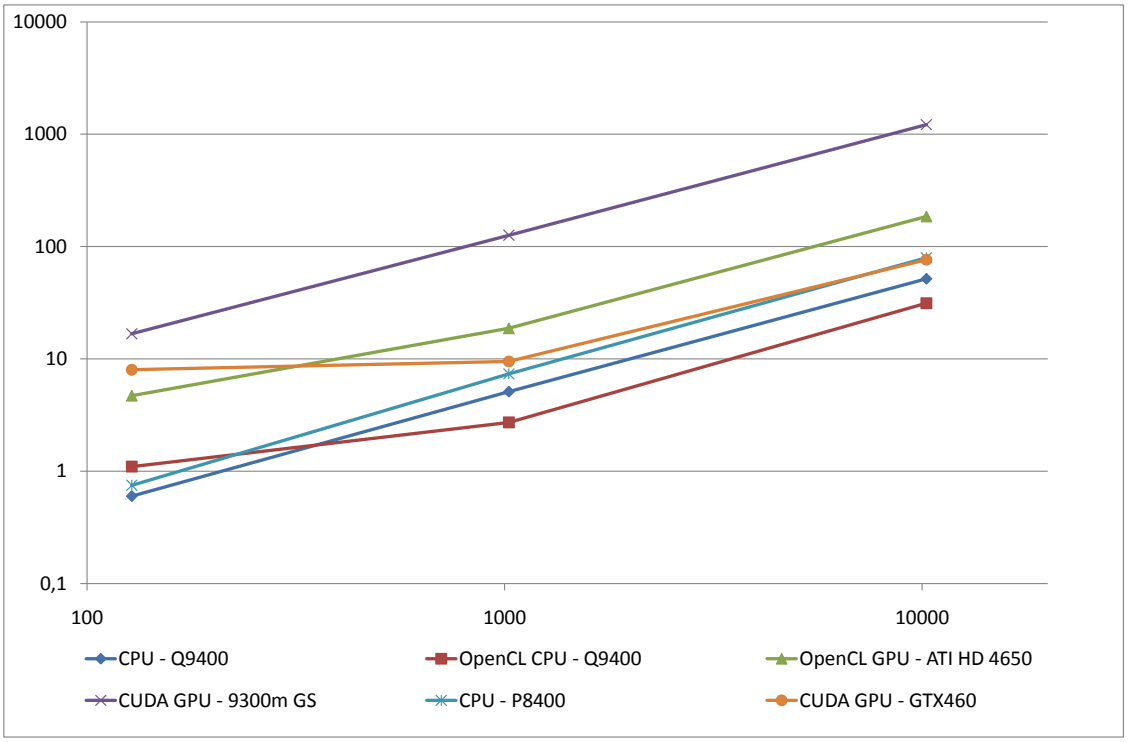

Рис. 5.3. Время генерации одного поколения в зависимости от размера поколения, задача про нулевой битовый вектор, N=1000, значения приведены в миллисекундах

В этой задаче также наблюдается рост производительности при использовании всех ядер центрального процессора с помощью *OpenCL*, вплоть до двух раз при увеличении размеров задачи. При фиксированной длине вектора и увеличении размеров поколения, проигрыш видеокарт центральному процессору уменьшается.

В этой задаче, видеокарта *ATI HD 4650* уже значительно выигрывает у видеокарты *Nvidia 9300m GS*, как и должно было быть, учитывая значительно большую теоретическую вычислительную мощность. Кроме того при $N = 10$ и 1024000 особей в одном поколении *ATI HD 4650* выигрывает у однопоточного вычисления на центральном процессоре.

Имея в виду специфику двух рассмотренных задач, можно сделать вывод, что видеокарта от *ATI* существенно больше оптимизирована для арифметических операций, как во второй задаче, чем для логических, как в первой. В то же время, видеокарта от *Nvidia* меньше проигрывает центральному процессору именно при выполнении логических операций.

Выводы

Как следует из результатов, при решении рассмотренных задач генетического программирования значимого выигрыша в производительности на имеющемся оборудовании удалось добиться не во всех случаях. Это связано с несколькими факторами.

В настоящей работе по большей части использовались одни из самых бюджетных вариантов устройств. Например, пиковое значение производительности видеокарты *Nvidia 9300M GS* всего в два раза превышает аналогичный показатель для *CPU*. Если учесть многолетнюю оптимизацию работы процессоров для вычислений общего назначения и молодой возраст технологий вычисления на видеокартах, становится ясно, что на указанном оборудовании прироста в производительности добиться практически невозможно. Лишь достаточно мощная видеокарта *GTX460* в полной мере раскрыла потенциал технологии *CUDA*.

При кажущейся простоте получения прироста производительность, для получения существенных результатов необходимо хорошее знание архитектуры видеокарты. Существует большое число возможных проблем, которые необходимо учитывать:

1. Индивидуальные особенности видеокарты.
2. Накладные расходы на пересылку данных между оперативной памятью и памятью видеокарты.
3. Сложную архитектуру памяти.

Кроме того, производители видеокарт оптимизируют свою продукцию для нужд игровой индустрии, а внутренняя архитектура практически не стандартизирована. Поэтому гарантировать прирост производительности

можно лишь в случае мощного устройства и использовании сложных арифметических операций для получения значения функции приспособленности.

Кроме всего прочего, следует избегать большого размера особей. Последнее связано с тем, что при повышении размера особи, возрастают расходы на пересылку данных между оперативной памятью и памятью видеокарты. По закону Амдала [8] проблемы такого рода могут практически свести на нет использование большого числа вычислительных устройств. Именно это явилось причиной поражения видеокарты во второй задаче.

Справедливости ради, отметим, что одним из путей решения этой проблемы является выполнение всех действий на видеокарте. Однако, эта задача не всегда тривиальна. Кроме того, память видеокарты обычно существенно уступает в размере оперативной памяти компьютера.

Как и ожидалось, в некоторых случаях использование видеокарты позволяет получить существенный прирост производительности без необходимости существенно модифицировать программный код.

Интересным результатом является возможность применения технологии *OpenCL* для более полного использования ресурсов *CPU*. В то же время данная технология позволяет переносимым образом использовать различные многоядерные устройства, в том числе графические процессоры.

Источники

[1] *Langdon, W.B.* A Many Threaded CUDA Interpreter Genetic Programming // EuroGP. — 2010.

[2] GPGPU, http://www.gpgpu.org

[3] *Harding S.L., Banzhaf W.* Fast genetic programming on GPUs // LNCS: Proceedings of the 10th European Conference on Genetic Programming. — 2007. — Vol. 4445.

[4] *Harding S.L., Banzhaf W.* Distributed genetic programming on GPUs using CUDA // WPABA: Proceedings of the Second International Workshop on Parallel Architectures and Bioinspired Algorithms. — 2009.

[5] *Lee V.W., Kim C. Chhugani J.* Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU // SIGARCH Computer Architecture News. — 2010. — Vol. 38, no. 3.

[6] NVIDIA GPU Computing Documentation, http://developer.nvidia.com/nvidia-gpu-computing-documentation

[7] *Dietzfelbinger M., Naudts B., Hoyweghen C., Wegener I.* The Analysis of a Recombinative Hill-Climber on HIFF. 2002.

[8] *Amdahl G.* Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities // AFIPS Conference Proceedings. — 1967. — no. 30.

Приложение А. Расстановка ферзей. Тестовые запуски

Для этой задачи варьировалось значение N — размеры поля (число ферзей) и число особей в поколении.

Таблица А.1. Время генерации одного поколения, задача о расстановки ферзей, $N = 10$, значения приведены в миллисекундах

Кол-во особей в поколении	100	1000	10000	100000
CPU - Q9400	0,11	1,15	11,4	118
OpenCL CPU - Q9400	0,31	1,26	10,5	107
OpenCL GPU - ATI HD 4650	0,61	1,8	13,1	122
CUDA GPU - 9300m GS	0,1	0,65	5,6	64,2
CPU - P8400	0,055	0,575	5,92	68,75
CUDA GPU - GTX460	0,12	1,4	2,5	27

Таблица А.2. Время генерации одного поколения, задача о расстановки ферзей, $N = 20$, значения приведены в миллисекундах

Кол-во особей в поколении	100	1000	10000	100000
CPU - Q9400	0,2	2	20	205
OpenCL CPU - Q9400	0,41	1,92	16,8	171
OpenCL GPU - ATI HD 4650	1,3	3,4	30	306
CUDA GPU - 9300m GS	0,255	1,385	13,37	144,6
CPU - P8400	0,125	1,26	12,725	141,25
CUDA GPU - GTX460	0,33	0,47	3,1	29,9

Таблица А.3. Время генерации одного поколения, задача о расстановки ферзей, $N = 50$, значения приведены в миллисекундах

Кол-во особей в поколении	100	1000	10000	100000
CPU - Q9400	0,54	5,3	54	539
OpenCL CPU - Q9400	0,82	3,9	36	351
OpenCL GPU - ATI HD 4650	5,6	15,6	150	1480
CUDA GPU - 9300m GS	1,015	5,815	57,95	587,7
CPU - P8400	0,44	4,375	44,15	456,2
CUDA GPU - GTX460	1,7	2,3	19,6	183,5

Таблица А.4. Время генерации одного поколения, задача о расстановки ферзей, $N = 200$, значения приведены в миллисекундах

Кол-во особей в поколении	100	1000	10000	100000
CPU - Q9400	4,39	44,5	443	4444
OpenCL CPU - Q9400	5,82	22,8	206	2073
OpenCL GPU - ATI HD 4650	90	238	2333	23000
CUDA GPU - 9300m GS	13,7	77	768	7700
CPU - P8400	4,39	44,5	443	4444
CUDA GPU - GTX460	22,9	38,5	354	3402

Таблица А.5. Время генерации одного поколения, задача о расстановки ферзей, $N = 1000$, значения приведены в миллисекундах

Кол-во особей в поколении	100	1000	10000
CPU - Q9400	88,9	885	8848
OpenCL CPU - Q9400	118	385	3680
OpenCL GPU - ATI HD 4650	2400	-	-
CUDA GPU - 9300m GS	602	6029	60300
CPU - P8400	96,6	966	9674
CUDA GPU - GTX460	618	1099	10087

Приложение B. Нулевой битовый вектор. Тестовые запуски

Для этой задачи варьировалось значение N — число 32-битных целых в массиве и число особей в поколении.

Таблица B.1. Время генерации одного поколения, задача про нулевой битовый вектор, $N = 10$, значения приведены в миллисекундах

Кол-во особей в поколении	128	1024	10240	102400	1024000
CPU - Q9400	0,032	0,25	2,4	26,4	280
OpenCL CPU - Q9400	0,38	0,56	2,5	22,2	230
OpenCL GPU - ATI HD 4650	0,77	1	3,4	28	277
CUDA GPU - 9300m GS	0,15	0,545	7,5	80	941
CPU - P8400	0,05	0,2	2,5	27,5	469,5
CUDA GPU - GTX460	0,08	0,14	1,03	13	237,4

Таблица B.2. Время генерации одного поколения, задача про нулевой битовый вектор, $N = 100$, значения приведены в миллисекундах

Кол-во особей в поколении	128	1024	10240	102400
CPU - Q9400	0,08	0,69	8,1	83,1
OpenCL CPU - Q9400	0,45	0,74	4,7	48
OpenCL GPU - ATI HD 4650	1,08	2,5	19,8	189
CUDA GPU - 9300m GS	0,85	6,865	73	719
CPU - P8400	0,1	0,75	9,5	106
CUDA GPU - GTX460	0,74	1	7,61	84,2

Таблица B.3. Время генерации одного поколения, задача про нулевой битовый вектор, $N = 1000$, значения приведены в миллисекундах

Кол-во особей в поколении	128	1024	10240
CPU - Q9400	0,6	5,11	51,7
OpenCL CPU - Q9400	1,1	2,72	31,3
OpenCL GPU - ATI HD 4650	4,7	18,7	185
CUDA GPU - 9300m GS	16,75	126,1	1216
CPU - P8400	0,75	7,35	79,5
CUDA GPU - GTX460	8	9,5	76,4

Таблица B.4. Время генерации одного поколения, задача про нулевой битовый вектор, $N = 10000$, значения приведены в миллисекундах

Кол-во особей в поколении	128	1024	4096
CPU - Q9400	6,05	50,2	208
OpenCL CPU - Q9400	8,6	29,6	121
OpenCL GPU - ATI HD 4650	41,7	190,9	375
CUDA GPU - 9300m GS	174,05	1248	-
CPU - P8400	9,05	76,15	-
CUDA GPU - GTX460	73,5	94,1	297,1