Программное обеспечение гетерогенных компьютерных систем и структуры систем обработки данных с повышенной производительностью

А.А. Колпаков[a], Ю.А. Кропотов[a]

[a] *Муромский институт (филиал) ВлГУ, 602264, ул. Орловская, 23, Муром, Владимирская область, Россия*

Аннотация

Вопрос создания высокопроизводительных вычислительных комплексов на базе гетерогенных компьютерных систем является актуальным, так как объемы обрабатываемой информации, вычислений и исследований с большими массивами данных постоянно увеличиваются. Целью работы является разработка методов проектирования программного обеспечения гетерогенной компьютерной системы обработки данных. В результате разработана методика объединения шейдеров для повышения производительности гетерогенных вычислений. Представленное решение предполагается реализовать в виде программного модуля для компьютерной системы с использованием технологии CUDA.

Ключевые слова: параллельные вычисления; гетерогенные вычислительные системы; графические процессоры; CUDA; OpenCL.

1. Введение

GPGPU-программу можно условно представить при помощи следующих множеств:

- множество шейдеров, выполняющих вычисления;
- множество переменных, управляющих вычислениями;
- множество данных, над которыми проходят вычисления и в которые заносятся их результаты;
- множество инструкций, которые запускают тот или иной шейдер, предоставляют ему на вход те или иные данные и выводят результат в некоторую текстуру.

1.1 Программирование графического процессора на основе вершинных и пиксельных программ

Как известно, элементарным примитивом для визуализации, с которым работает графический процессор, является треугольник. При этом с каждой вершиной треугольника можно связать ограниченный набор произвольных данных, например ее цвет, нормаль и другие пользовательские данные. С самим примитивом может быть ассоциировано до 8 текстур – одно-, двух-, и трехмерные изображения. Структурная схема обработки данных на основе вершинных и пиксельных программ представлена на рис. 1.

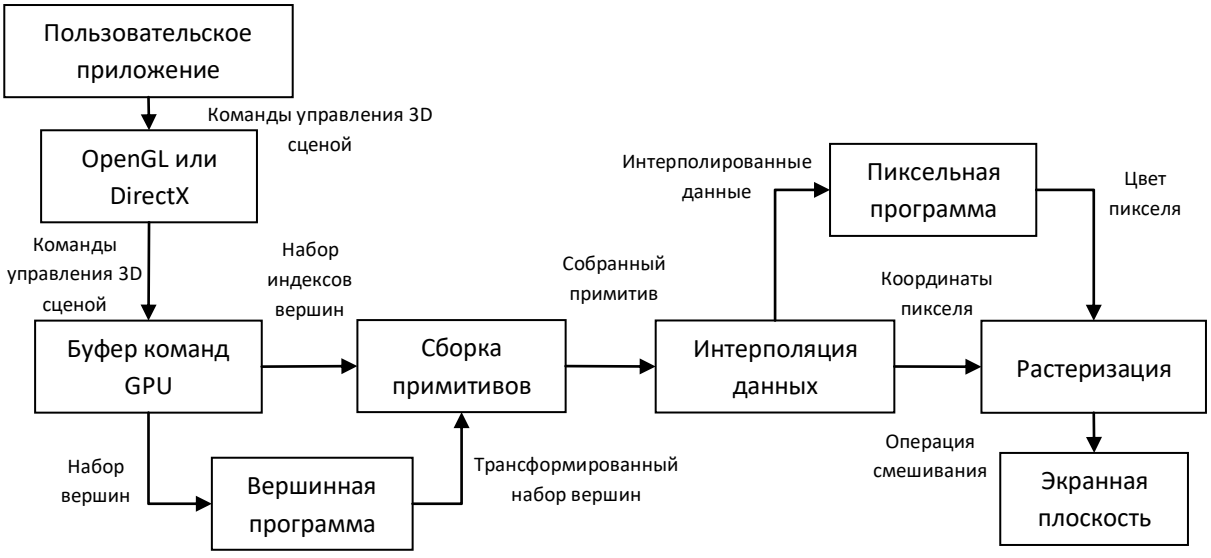


Рис. 1. Структурная схема обработки данных на основе вершинных и пиксельных программ.

Как представлено на рис. 1, приложение пользователя отправляет запросы на визуализацию трехмерной сцены программным библиотекам низкого уровня OpenGL или DirectX , являющимися частью операционной системы. Эти данные затем трансформируются с помощью драйвера видеокарты в прямые команды графического процессора [1,2,3].

1.2 Программирование графического процессора на основе библиотеки CUDA

Несмотря на то. что программирование вершинных и пиксельных программ оказалось эффективным для моделирования различных физических процессов, использование этого метода не является удобным для программиста. Для выполнения вычислительных задач на графическом процессоре программисту требуется иметь достаточно высокую квалификацию, т.е. использовать принципы работы графического процессора в деталях, так как небольшая неточность в коде управления графическим процессором может привести к значительному искажению результата вычислений. Структурная схема работы программного обеспечения на основе библиотеки CUDA представлена на рис. 2.

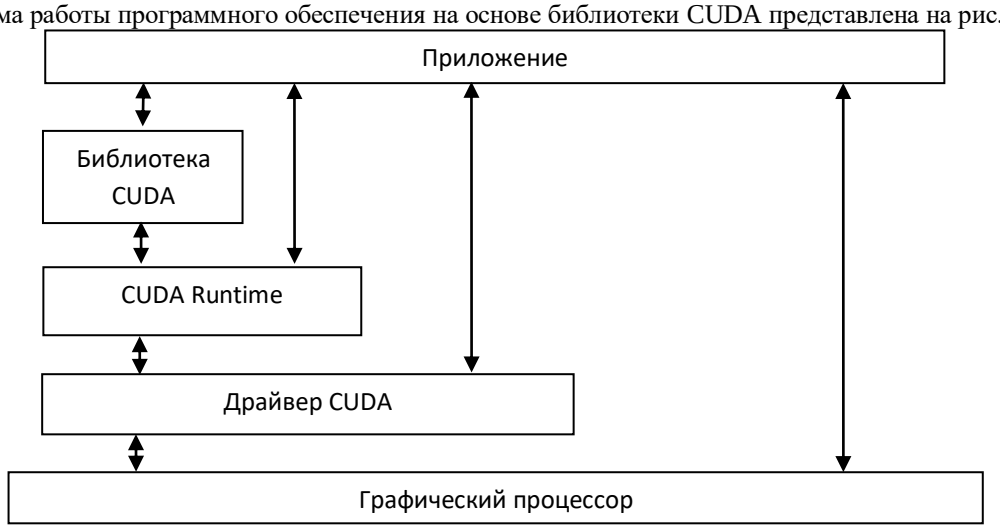


Рис.2. Структурная схема работы программного обеспечения на основе библиотеки CUDA.

Исполняемой единицей CUDA-программы является warp. Размер warp составляет 32 потока. Связано это с тем, что задержка (latency) при выполнении одной инструкции на мультипроцессоре составляет 4 такта. Только в отношении warp можно говорить о параллельном выполнении потоков, никаких других допущений делать нельзя. Однако это не означает, что warp-ы исполняются на мультипроцессоре последовательно. Выполнение warp-ов может быть параллельным, например, в том случае, когда один warp ожидает данные из глобальной памяти, другие warp-ы могут в это время выполняться.

Взаимодействие между потоками может осуществляться только внутри блока. Обмен данными осуществляется посредством разделяемой (shared) памяти, общей для всех потоков в блоке. Синхронизацию выполнения потоков можно осуществить путем вызова специальных функций синхронизации.

1.3 Программирование графического процессора на основе библиотеки OpenCL

Результатом развития GPU и его использования в задачах, несвязанных с компьютерной графикой, стала разработка единого стандарта описания вычислений на высокопараллельных системах – OpenCL (Open Computational Library). Сформировавшаяся библиотека OpenCL появилась на основе ранее разработанной технологии Nvidia CUDA. описывающей интерфейс взаимодействия приложения с вычислительными ресурсами графического процессора. В отличие от технологии CUDA, технология OpenCL описывает модель вычислений без связи с конкретным типом устройства, на котором эти вычисления будут исполняться. В силу того, что OpenCL разработан именно как стандарт вычислений на высокопараллельных системах, многие специфические возможности технологии CUDA были исключены из стандарта. В связи с этим, в целом, технология CUDA имеет больше возможностей, в сравнении с OpenCL. для описания параллельных вычислений, если в качестве вычислительного устройства выступает графический процессор Nvidia.

OpenCL позволяет описывать вычисления, абстрагируясь от конкретного устройства, на котором эти вычисления будут реализованы. В общем случае, алгоритмы, написанные с использованием OpenCL. могут исполняться на нескольких ядрах центрального процессора, на графическом процессоре. или на процессорах IBM Cell/B.E. В своей реализации OpenCL использует расширения языка С для описания алгоритма [3,4].

Библиотека OpenCL является перспективной библиотекой для применения в различных научных исследованиях. Достоинством библиотеки OpenCL является ее поддержка со стороны высокопроизводительных кластеров. Любое приложение, использующее OpenCL, может быть запущено без модификаций на кластере, содержащем, в том числе, и графические процессоры. Такому приложению будут доступны все имеющиеся вычислительные ресурсы в системе. Структурная схема работы программного обеспечения на основе библиотеки OpenCL представлен на рис. 3.

Центральным элементом модели платформы OpenCL выступает понятие хоста (host) - первичного устройства, которое управляет OpenCL-вычислениями и осуществляет все взаимодействия с пользователем. Хост всегда представлен в единственном экземпляре, в то время как OpenCL-устройства (devices), на которых выполняются OpenCL-инструкции могут быть представлены во множественном числе. OpenCL-устройством может быть CPU, GPU, DSP или любой другой процессор в системе, поддерживающийся установленными в системе OpenCL-драйверами.

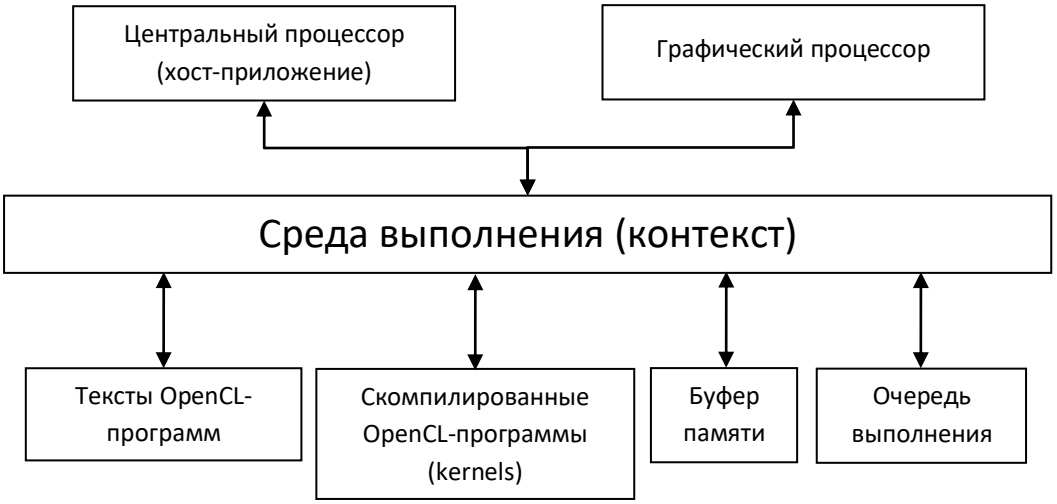


Рис. 3. Структурная схема работы программного обеспечения на основе библиотеки OpenCL.

2. Программное обеспечение гетерогенной компьютерной системы обработки данных

Структурная схема программного обеспечения гетерогенной компьютерной системы обработки данных изображена на рис. 4.

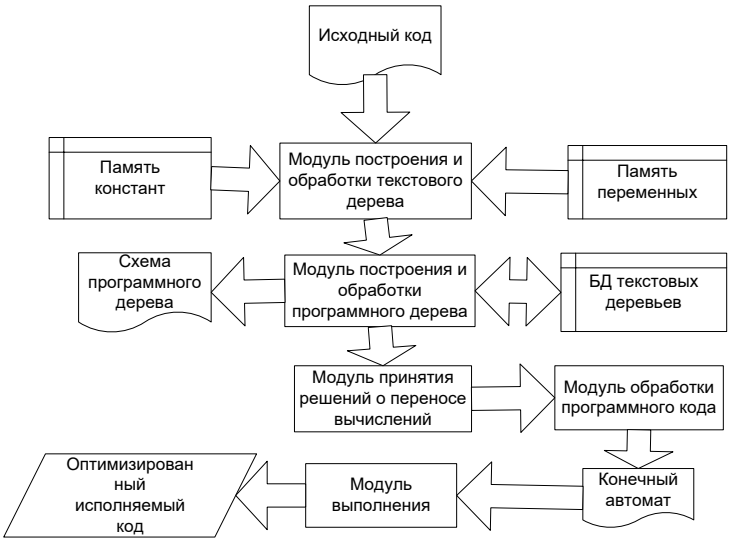


Рис. 4. Структурная схема разработанного программного обеспечения гетерогенной компьютерной системы обработки данных.

Согласно рис. 4 входным документом программного обеспечения является исходный код обрабатываемой программы. Он поступает в модуль построения и обработки текстового дерева, в котором происходит первоначальная обработка исходного кода и построение на его основе текстового дерева. Для работы с текстовым деревом используется память констант и память переменных.

Полученное текстовое дерево передается на обработку в модуль построения и обработки программного дерева. Вышеуказанный модуль для построения программного дерева использует базу данных ранее обработанных текстовых деревьев, что позволяет существенно ускорить построение программного дерева [5,6].

Для проведения обработки программы необходимо построить дерево, представляющее текстовое описание в удобном формате. В программном дереве может быть несколько типов узлов:

- корневой узел - в качестве потомков содержит все данные, которые необходимо вычислить в ходе выполнения программы;
- узел с данными - представляет массив данных, который требуется передать на вход предку или в который необходимо записать результат потомка;
- узел с переменной - представляет переменную, которую надо передать на вход шейдеру-предку или которая описывает условие условного оператора-предка;
- узел с отсутствием операций - передает данные потомка предку без их изменения;
- узел с шейдером - выполняет шейдер с входными данными, полученными от потомков, и передает результат предку;

- узел с арифметической операцией - выполняет арифметическое преобразование входных данных, полученных от потомков, и передает результат предку;
- узел с началом ветвления - описывает условный оператор; первым потомком является переменная, описывающая, какую ветвь следует выбирать, далее следуют различные ветви выполнения;
- узел с завершением ветвления - описывает завершение условного оператора, все его ветви сводятся к этому узлу.

Деревья поочередно строятся для каждой инструкции. Парсер разбирает строковое представление, определяет выходной массив с данными, при необходимости раскрывает абстракции и определяет набор функций, которые необходимо выполнить для получения результата.

В каждом последующем дереве узел с данными, в которые была произведена запись в одной их предыдущих инструкций, заменяется деревом последней такой инструкции.

Программное дерево получается после обработки последней инструкции. Стоит отметить, что программное дерево может не являться деревом по определению - некоторые узлы могут иметь более одного предка, но чаще всего оно представляет древовидную или близкую к ней структуру.

Программное дерево передается в модуль принятия решений о переносе вычислений, где происходит разбиение исходного алгоритма на этапы и принятие решения о переносе вычислений на GPU для каждого этапа с дальнейшей передачей в модуль обработки программного кода.

3. Разработка модуля обработки программного кода

Модуль обработки программного кода производит необходимые изменения в исходном программном коде для получения конечного автомата выполнения программы, передаваемого в модуль выполнения, генерирующего обработанный исполняемый код.

Уровень целевого исполнителя для семейства ускорителей принимает на вход граф функциональных операций, а на выходе получает программу для ГПУ [7,8]. Эта программа имеет вид сценария исполнения, в котором доступны операции запуска шейдера на ГПУ с определенным набором параметров, выделение и освобождение памяти, а также передача данных из ОЗУ в видеоОЗУ и обратно. Основные этапы трансляции программы на уровне целевого исполнителя:

1. Разбиение исходного графа на подграфы, соответствующие различным проходам. В дальнейшем каждый подграф транслируется отдельно.

2. Выбор отображения для данных и для кода. Для кода выбирается и для данных выбирается их отображение на архитектуру ГПУ. При этом, например, один логический массив данных может отображаться на несколько физических буферов ГПУ, а операция редукции в исходном коде – на цикл, каждая итерация которого обрабатывает блок из 4-х элементов.

3. Генерация кода для ГПУ. После того, как отображения выбраны, по ним осуществляется генерация кода на ГПУ. В частности, осуществляется вычисление индексов для буферов ГПУ и ликвидация повторяющихся чтений.

4. Преобразования сгенерированного кода. На этом этапе производится слияние арифметических выражений в векторные команды и свертка констант.

Поскольку отдельные выражения, вычисляемые на ГПУ, относительно невелики, а проход требует большой вычислительной мощности шейдера, первый этап относительно прост. Чаще всего результатом его работы является единственный подграф, совпадающий с исходным графом. Основным критерием разбиения на проходы является повторное использование данных. Разбиение осуществляется только в том случае, если без него не удастся избежать многократного вычисления одних и тех же данных.

Самыми сложными с точки зрения реализации являются этапы 2 и 3, их описанию посвящены два следующих раздела. Этап 4 относительно прост и выполняется при помощи арифметических преобразований. Возможность слияния арифметических выражений в векторные операции обеспечивается эффективным выбором отображений на этапах 2 и 3.

Функциональный граф C$ определяет следующие типы вершин [9,10]:

Листовые вершины:

1. Константы.

2. Связанные переменные. По сути, аналогичны параметрам цикла. Пробегают определенный диапазон и могут использоваться для вычисления индексных выражений.

3. Функции. Могут быть функциями-членами (в т.ч. стандартными операциями) или массивами. Если встречается функция-выражение, то она применяется к своим аргументам, таким образом, в конечном представлении ее не будет.

Внутренние вершины:

5. Применение функции к аргументам (@). Это может быть применение обычной функции или же обращение к элементу массива.

6. Операция редукции (R). Имеет 2 аргумента, первый из которых задает редуцирующую функцию, а второй – редуцируемую. Редукция применяется по всем измерениям массива, не содержащим связанных переменных, а также по части связанных переменных, что позволяет реализовать частичную редукцию.

Связанные переменные «распространяются» снизу вверх по графу. Оканчивать свое распространение они могут только на вершинах редукции или на корневой вершине. Также считаем, что ациклический граф устроен таким образом, что все пути распространения связанной переменной заканчиваются на одной и той же вершине.

Ниже представлен пример объединения двух шейдеров, в котором можно увидеть неоптимальные участки кода. Блок-схема первого шейдера представлена на рис. 5.

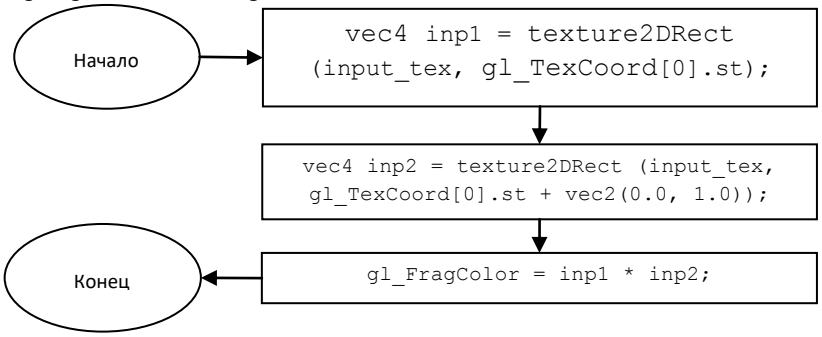


Рис. 5. Блок-схема первого шейдера.

Блок-схема второго шейдера представлена на рис. 6.

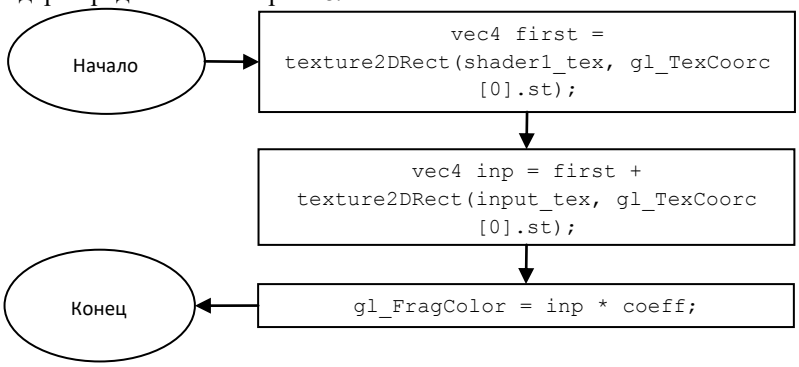


Рис. 6. Блок-схема второго шейдера.

Блок-схема объединенного шейдера, полученная после проведения слияния последовательности этих шейдеров, представлена на рис. 7.

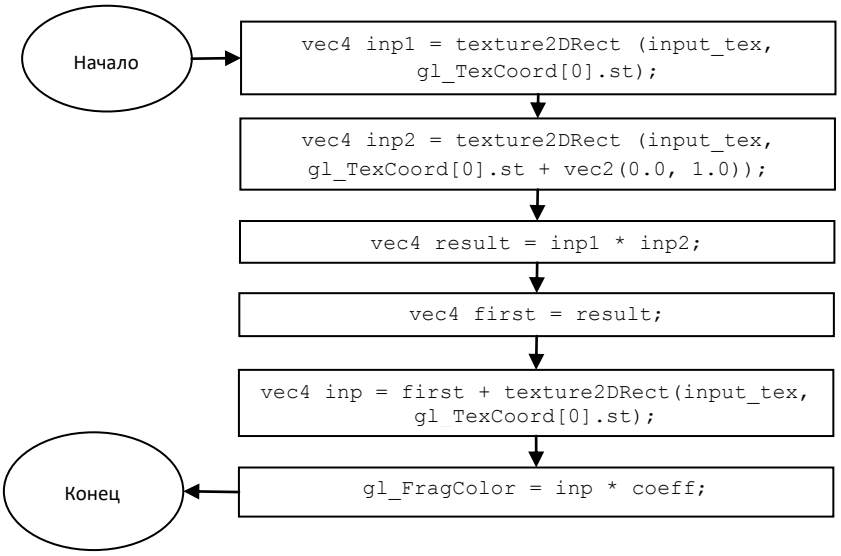


Рис. 7. Блок-схема объединенного шейдера.

Для более производительного выполнения полученного шейдера необходимо выполнить его коррекцию. Блок-схема итогового скорректированного шейдера представлен на рис. 8.

После проведения всех вышеописанных изменений код шейдера будет компилироваться и будет готов к использованию в вычислениях.

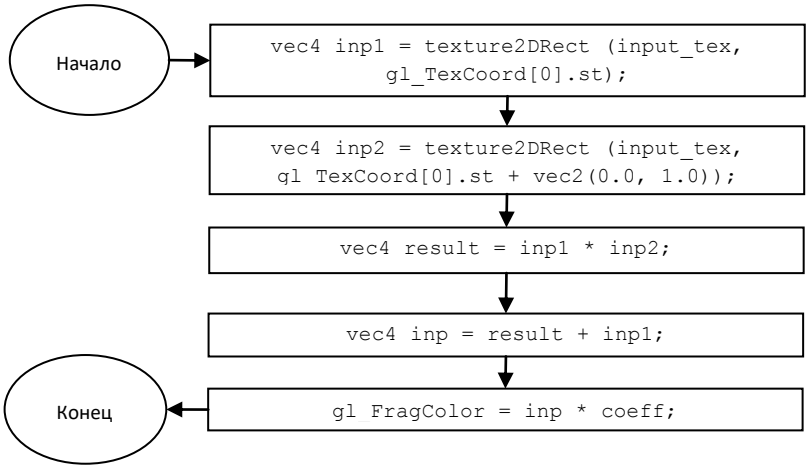


Рис. 8. Блок-схема итогового шейдера.

4. Выводы

В качестве выходных данных программа предоставляет схемы обработанного программного дерева и сгенерированного автомата, а также полученные шейдеры. Схема обработанного программного дерева представлена на рис. 9.

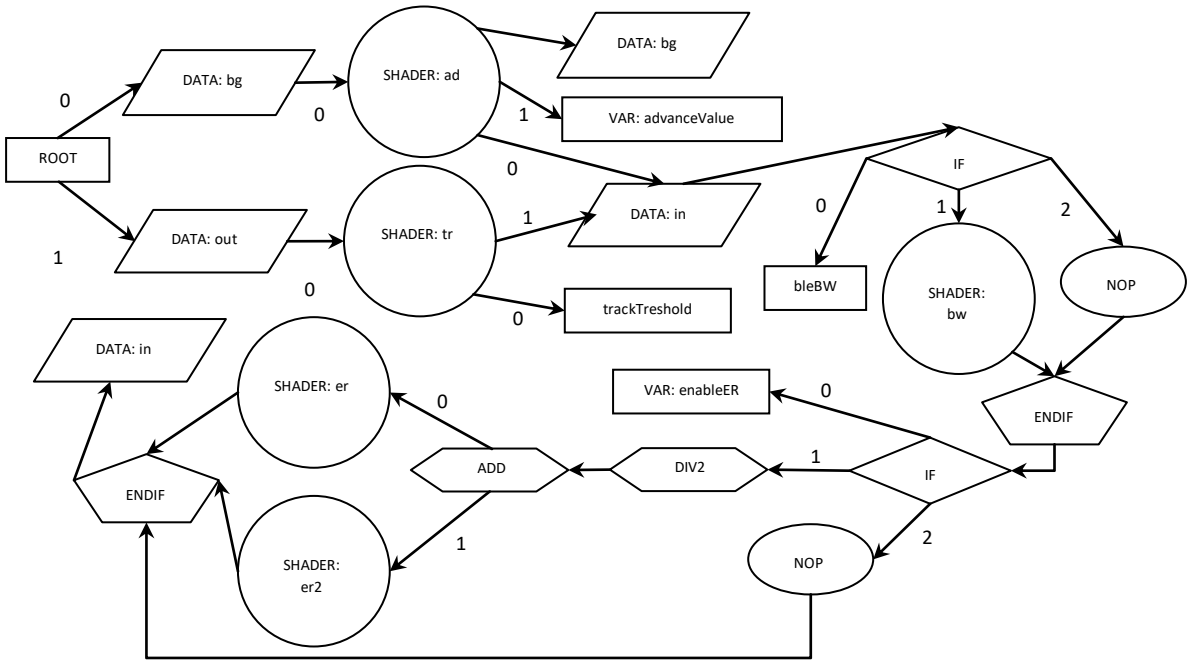


Рис. 9. Обработанное программное дерево.

Как видно из рис. 9, программное дерево состоит из узлов различных типов, каждый из которых описывает поведение программы: корневой узел; узел с данными; узел с переменной; узел с отсутствием операций; узел с шейдером; узел с арифметической операцией; узел с началом ветвления; узел с завершением ветвления.

Генерация кода для графического процессора осуществляется в соответствии с выбранным отображением. Каждой ключевой вершине ставится в соответствие некоторый набор выходных переменных, за вычисление которых она отвечает. Корневая вершина, помимо их вычисления, отвечает и за запись их в выходные буферы. Для каждого подграфа операций генерируется шаблон кода. На основании этого шаблона генерируется код для каждого набора значений экземпляров блочных измерений [11,12].

Наиболее сложной частью этапа генерации кода является генерация команд доступа в ОЗУ. По отображению определяется множество номеров буферов, которые могут участвовать этой операции чтения данных. Далее, для каждого из номеров буферов определяется для различных значений блочных индексов набор двумерных индексов для чтения из этого буфера. Для каждого индекса, на основании набора переменных, от которых он реально зависит, находится самая близкая к корню вершина, в которой индекс может быть вычислен. Улучшается вычисление индексов, которые линейны по итерациям цикла. В каждой ключевой вершине отмечается набор индексов, которые в ней требуется вычислить, что позволяет избежать повторного вычисления индексов. Операция чтения данных выполняется в той же ключевой вершине, к которой принадлежит индекс. Это позволяет избежать повторного чтения одних и тех же данных из ОЗУ, даже если они производятся разными вершинами чтения.

Для работы с индексами используются средства автоматического преобразования и упрощения целочисленных выражений. Они поддерживают преобразование выражений, содержащих арифметические операции, минимум, максимум, а также операции сравнения.

Программа позволяет проводить гибкую манипуляцию опциями.

В ходе исследования эффективности разработанных алгоритмов обработки информации была использована задача нахождения нулевых битовых векторов, которая решается с применением генетических алгоритмов [42]. При решении указанной задачи основное время работы занимают параллельные вычисления значений функций приспособленности различных особей, операций скрещивания и мутации. 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.

Данные свойства позволяют эффективно использовать вычисления на графическом процессоре.

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.

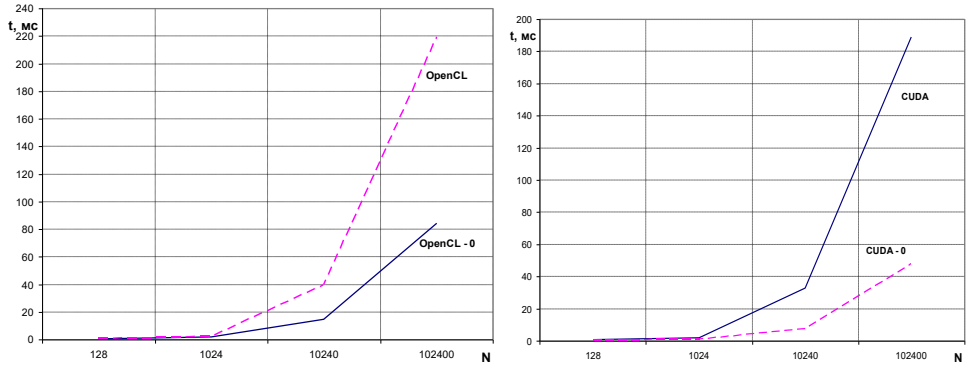
Для проведения экспериментальной оценки эффективности работы алгоритма повышения производительности обработки данных использовалась тестовая компьютерная система следующей конфигурации: центральный процессор Intel Core 2 Quad Q9400 (2.66GHz), ОЗУ 8GB, графическая карта Nvidia GeForce GTX560 2Gb 336 потоков, операционная система Windows 7 x64**,** компилятор MS Visual Studio 2008 в release режиме.

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 исследовании производительности первой тестовой задачей изменялось количество 32-битных целых чисел в массиве (M) и число параллельных потоков (N).

Исследовалось среднее время t, потраченное на получение нового поколения для различного количества 32-битных целых чисел в массиве и числа параллельных потоков. Исследования проводились с использованием технологий OpenCL и NVIDIA CUDA [4]. Результаты исследований среднего времени, потраченного на получение нового поколения, для параметра $M = 10$ приведены на рис. 10.



а б

Рис. 10. Среднее время, потраченное распараллеленной системой на получение нового поколения, при M =10, а: OpenCL – базовый алгоритм, OpenCL-O – разработанный алгоритм, б: CUDA– базовый алгоритм, CUDA-O – разработанный алгоритм.

На рис. 10 графики OpenCL и CUDA показывают время выполнения базового алгоритма, OpenCL-O и CUDA-O – с применением разработанного алгоритма повышения производительности обработки данных. Как видно из графиков, при значении параметра $M = 10$ применение разработанного алгоритма оптимизации дает рост производительности: в случае применения OpenCL время обработки для 128 потоков сокращается с 1,08 мс до 0,75 мс и с 219 мс до 84,2 мс для 102400 потоков. В случае применения NVIDIA CUDA время обработки сокращается с 0,85 мс до 0,74 мс для 128 потоков и со 189 мс до 48,4 мс для 102400 потоков [4].

Литература

[1] Современные проблемы вычислительной математики и математического моделирования. Т. 1: Вычислительная математика. / Под ред. Бахвалов Н. С., Воеводин В. В. – М.: Наука, 2005. – 342 с.

[2] Колпаков, А. А. Аспекты оценки увеличения производительности вычислений при распараллеливании процессоров вычислительных систем / А.А. Колпаков // Методы и устройства передачи и обработки информации. – 2011. – №1(13). – С. 124-127.

[3] Колпаков, А. А. Теоретическая оценка роста производительности вычислительной системы при использовании нескольких вычислительных устройств / А.А. Колпаков // В мире научных открытий. – 2012. – №1. – С. 206-209.

[4] Колпаков, А. А. Оптимизация генетических алгоритмов при использовании вычислений на графических процессорах на примере задачи нулевых битовых векторов / А.А. Колпаков // Информационные системы и технологии, – 2013. – №2(76). – С. 22-28.
[5] Кропотов, Ю. А. Экспериментальные исследования закона распределения вероятности амплитуд сигналов систем передачи речевой информации / Ю.А. Кропотов // Проектирование и технология электронных средств. – 2006. – Т.4. – С. 37-42.
[6] Кропотов, Ю. А. Алгоритм подавления акустических шумов и сосредоточенных помех с формантным распределением полос режекции / Ю.А. Кропотов, А.А. Быков // Вопросы радиоэлектроники, – 2010. – Т.1 №1. – С. 60-65.
[7] Кропотов, Ю. А. Временной интервал определения закона распределения вероятности амплитуд речевого сигнала / Ю.А. Кропотов // Радиотехника, – 2006. – №6. – С. 97-98.
[8] Кропотов, Ю. А. О корреляционном оценивании параметров моделей акустических эхо-сигналов / В.А. Ермолаев, Ю.А. Кропотов // Вопросы радиоэлектроники, – 2010. – Т.1 №1. – С. 46-50.
[9] Кропотов, Ю. А., Проскуряков А. Ю., Белов А. А., Колпаков А. А. Модели, алгоритмы системы автоматизированного мониторинга и управления экологической безопасности промышленных производств / Ю.А. Кропотов, А. Ю. Проскуряков, А. А. Белов, А.А. Колпаков // Системы управления, связи и безопасности. – 2015. – №2. – С. 184-197.
[10] Кропотов, Ю. А., Белов А. А., Проскуряков А. Ю., Колпаков А. А. Методы проектирования телекоммуникационных информационно–управляющих систем аудиообмена в сложной помеховой обстановке / Ю.А. Кропотов, А. А. Белов, А. Ю. Проскуряков, А.А. Колпаков // Системы управления, связи и безопасности. – 2015. – №2. – С. 165-183.
[11] Kropotov, Ju.A. Identification of Models for Discrete Linear Systems with Variable, Slowly Varying Parameters / V.A. Ermolaev, V.T. Eremenko , O.E. Karasev , Ju.A. Kropotov //Journal of Communications Technology and Electronics. – 2010. – vol. 55, № 1. – P. 52-57.
[12] Kropotov, Ju.A. Algorithms for processing acoustic signals in telecommunication systems by local parametric methods of analysis [Electronic resource]/ V.A. Ermolaev, Ju.A. Kropotov // 2015 International Siberian Conference on Control and Communications (SIBCON) – Proceedings. – 2015. – Access mode: http://ieeexplore.ieee.org/document/7147109/