Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э.Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:
«Моделирование развития клеточного организма при помощи технологии CUDA»

Студент ИУ9-52 _____ _____
(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____
(Подпись, дата) (И.О.Фамилия)

2016 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Трудно представить себе сегодняшний мир без процесса обучения. Среди множества быстро развивающихся технологий современный человек должен еще быстрее ориентироваться, собирать, классифицировать и анализировать новую информацию. Помогают ему в этом различные приложения для моделирования естественных процессов. Эта работа посвящена разработке программы для моделирования развития клеточного организма. *Объектом исследования* являются эвристические алгоритмы, используемые для решения задач оптимизации и моделирования. *Предметом исследования* был выбран генетический алгоритм моделирования развития организма и его реализация с помощью технологии CUDA. Их *практическое значение* определяется их относительной простотой и наглядностью.

Целью работы является разработка приложения с помощью технологии CUDA, позволяющего моделировать процесс дифференциации клеток многоклеточного организма в ходе онтогенеза. В ходе работы выполняются следующие *задачи*: изучение литературы по соответствующей тематике, с целью ознакомления с предметной областью, освоение технологии CUDA, рассмотрение и анализ генетических алгоритмов, их реализация.

1.ОБЗОР ТЕХНОЛОГИИ CUDA

Текущие актуальные тенденции таковы, что акцент в вычислениях смещается от «централизованной обработки» на CPU к «распределенной обработке» на GPU. Для реализации новой вычислительной парадигмы компания NVIDIA изобрела архитектуру параллельных вычислений CUDA (англ. *Compute Unified Device Architecture*). Относительно низкая стоимость и удельное энергопотребление видеокарт позволяют применять эту технологию почти повсеместно.

CUDA позволяет использовать GPU для вычислений и, таким образом, существенно увеличивать производительность. Вычислительная архитектура основана на концепции SIMD.

Также NVIDIA предоставляет CUDA SDK, что позволяет программистам реализовывать на специальном диалекте Си алгоритмы, выполнимые на GPU. CUDA также дает разработчику возможность доступа к памяти видеокарты. Это предоставляет в распоряжение программиста низкоуровневый, распределяемый и высокоскоростной доступ к оборудованию.

Выполнение расчётов на GPU показывает отличные результаты в алгоритмах, использующих параллельную обработку данных. Согласно данным, взятым с официального сайта NVIDIA, количество операций с плавающей точкой в секунду на GPU, примерно в 10 раз превышает количество операций на CPU фирмы Intel.

Чтобы понять, чем технология CUDA отличается от других технологий распараллеливания, надо понять, в чем состоит отличие вычислений на CPU, от вычислений, проводимых на GPU.

1.1 Принципиальная разница между CPU и GPU

Рассмотрим вкратце некоторые существенные отличия между обработкой данных на центральном и графическом процессорах.

Основная причина медленной работы большинства вычислительных систем заключается в том, что доступ к памяти занимает гораздо больше времени, чем работа процессора. Производители CPU добиваются существенного прироста производительности путем введения кэшей.

На GPU же медленные обращения к памяти никуда не исчезают, но маскируются с помощью использования параллельных вычислений. Таким образом, пока подготавливаются данные, необходимые одной задаче, работают другие, уже готовые к вычислениям. Это один из основных принципов CUDA, позволяющих сильно поднять производительность системы в целом.

Также имеются и существенные различия в поддержке многопоточности. CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеокарта может поддерживать до 1024 потоков на каждый мультипроцессор, которых, к тому же, несколько штук. Заметно отличается и время переключения с одного потока на другой, для CPU это занимает сотни тактов, для GPU же — 1-2 такта.

Итак, подведем итоги, касающиеся основных отличий, между архитектурами CPU и GPU. Центральные процессоры созданы для исполнения малого количества потоков последовательных инструкций, обрабатывающих различные данные с максимальной производительностью, а процессоры видеокарт проектируются для быстрого исполнения большого числа параллельно выполняемых потоков инструкций, работающих с однотипными данными.

Итак, основой для эффективного использования GPU является распараллеливание алгоритмов. Это превосходно подходит для нашей задачи создания приложения моделирования онтогенеза, требующего обработки большого количества однотипных данных.

1.2 Осуществление параллельных вычислений на GPU

Для начала введем основные термины и отношения между ними:

• Хост (Host) — центральный процессор.

• Устройство (Device) — видеоадаптер.

• Тред (Thread, поток) — единица выполнения программы.

• Блок (Block) — объединение тредов.

• Грид (Grid) — объединение блоков.

• Ядро (Kernel) — параллельная часть алгоритма, выполняется на треде.

Всего в CUDA есть 6 типов памяти.

Регистры используются для хранения локальных переменных. Расположены на кристалле GPU. Скорость доступа самая быстрая.

Локальная — используется для хранения локальных переменных, когда регистров не хватает. Скорость доступа низкая. Выделяется отдельно для каждого треда.

Разделяемая — используется для хранения массивов данных, используемых совместно всеми тредами в блоке. Расположена на кристалле GPU. Имеет чуть меньшую скорость доступа, чем регистры. Выделяется на блок.

Глобальная — основная память видеокарты. Используется для хранения больших массивов данных. Имеет медленную скорость доступа. Выделяется целиком на грид.

Константная — используется для хранения констант. Выделяется целиком на грид.

Текстурная — используется для хранения больших массивов данных. Выделяется целиком на грид.

Рассмотрим вычислительную модель GPU более подробно.

CUDA C разрешает программисту определять C-функции, (ядра), которые выполняются N раз параллельно N разными тредами. Каждый тред, который исполняет ядро, имеет уникальный *thread ID* , который доступен извне ядра.

Треды объединяются в блок, сформированный в виде трехмерного массива.

Количество тредов в блоке ограничено, так как треды выполняются на одном и том же процессоре и имеют разделяемые и ограниченные ресурсы. На современных GPU количество тредов в блоке может достигать 1024.

Ядро может выполняться несколькими блоками, так что общее количество потоков будет равно количеству тредов в блоке, умноженному на количество блоков.

Блоки организовываются в одно-, двух- или трехмерный грид. Количество блоков в гриде обычно выбирается исходя из размера данных, которые необходимо обработать.

К каждому блоку грида можно обратиться, используя встроенную переменную *blockIdx*. За размерность блока отвечает встроенная переменная *blockDim*.

1.3 Процесс программирования на GPU

Обычно программа на CUDA описывается на подмножестве языка C с некоторыми расширениями, хотя существует поддержка и других языков. Для некоторых версий CUDA Toolkit также наложены ограничения, например, нельзя брать адрес от функции или рекурсивно вызывать функции. Сейчас остановимся подробнее на расширения стандарта C, которые позволяют программировать на CUDA.

Спецификаторы функций:

__device__ — этот спецификатор объявляет функцию, исполняемую на GPU.

__global__ — этот спецификатор объявляет функцию ядра. Функции такого типа исполняются на GPU, но вызываются с CPU.

__host__ — этот спецификатор объявляет функцию, исполняемую на CPU.

Спецификаторы переменных:

__device__ — этот спецификатор указывает, что переменная хранится в памяти GPU.

__constant__ — этот спецификатор указывает, что переменная хранится в константной памяти.

__shared__ — этот спецификатор указывает, что переменная хранится в разделяемой памяти блока.

Новые типы данных:

Это векторные типы данных, состоящие из базовых целочисленных типов и типов с плавающей точкой. Они являются одно-, двух-, трех- и четырехкомпонентными структурами, с полями x, y, z и w. Все они обладают конструктором make_<type name>.

Функции синхронизации:

void __syncthreads();

Ожидает пока все треды в блоке достигнут этой точки.

__syncthreads() используется для сообщения между тредами одного блока. Проблема состоит в том, что обращения к глобальной или разделяемой памяти могут приводить к «состоянию гонки». Эта угроза нарушения целостности данных избегается с помощью функции __syncthreads().

Также имеются расширения для работы с текстурами, математическими функциями, функциями измерения времени, атомарными функциями, функциями форматированного вывода, инструкции для обработки видео, которые не имеют отношения к нашей задаче, поэтому здесь не рассматриваются.

2.РАЗРАБОТКА МОДЕЛИ ГЕНЕТИЧЕСКОГО КОДА

2.1 Основные принципы развития многоклеточных организмов

Прежде чем разработать алгоритм для моделирования процесса дифференциации клеток многоклеточного организма, необходимо определиться с основными понятиями предметной области.

Рассмотрим понятие морфогенеза.

Морфогенез (от греч. Morphe) — формообразование, возникновение новых форм и структур, как в онтогенезе, так и в филогенезе организмов. [5] Естественно, в рамках этого проекта уместно рассматривать только процесс онтогенеза.

Онтогенез (от греч. Ontos) — индивидуальное развитие особи, вся совокупность ее преобразований от зарождения до конца жизни.

Некоторые идеи касательно влияния физических и химических процессов на протекание морфогенеза высказывались еще в середине XX века Д`Арси Вентвортом Томпсоном и Аланом Тьюрингом. Например, в работе Тьюринга «Химические основы морфогенеза» впервые описывается процесс самоорганизации материи с математической стороны. Работы этих авторов помогли понять, что процесс развития организма тесно связан с такими понятиями и явлениями, как диффузия, активация и деактивация.

Однако чтобы понять, как смоделировать развитие организма необходимо иметь более четкое представление о том, как именно происходит передача «инструкций», по которым «строится» организм, между клетками.

Определим для этого следующие понятия.

Транскрипция — биосинтез молекул РНК на соответствующих участках ДНК; первый этап реализации генетической информации в клетке, в процессе которого последовательность нуклеотидов ДНК «переписывается» в нуклеотидную последовательность РНК.[6]

Экспрессия генов — это процесс, в котором вся наследственная информация от гена преобразуется в функциональный продукт — РНК или белок. Экспрессия генов может регулироваться на всех стадиях процесса. [7]

Факторы транскрипции — белки, контролирующие процесс синтеза мРНК на матрице ДНК путём связывания со специфичными участками ДНК. [6]

Факторы транскрипции необходимы для регуляции экспрессии генов. Однако, помимо факторов транскрипции, необходимых для включения экспрессии всех генов, существуют и факторы транскрипции для включения или выключения каждого конкретного гена в определенный момент.

МикроРНК — малые некодирующие молекулы РНК длиной 18-25 нуклеотидов (в среднем 22), обнаруженные у растений, животных и некоторых вирусов, принимающие участие в транскрипционной и посттранскипционной регуляции экспрессии генов путём РНК-интерференции[8].

РНК-интерференция — процесс подавления экспрессии гена на стадии транскрипции, трансляции, или деградации мРНК при помощи малых молекул РНК.[9]

Оперон — группа генов в хромосоме, включающая структурные гены и ген-оператор. Структурные гены управляют синтезом ферментов, задействованных в образовании клеточного составляющего или в потреблении питательных веществ. Ген-оператор связан с молекулой репрессора и может существовать в открытом или закрытом состоянии. Когда ген-оператор открыт, гены, которые он контролирует, являются функциональными, т.е., производят белки. Взаимодействуя с репрессором, ген-оператор закрывается. [7]

Таким образом, можно заметить, что регуляция экспрессии генов, например, путем воздействия на факторы транскрипции или микроРНК,

регулирует и весь процесс дифференцирования, морфогенеза и адаптации клеток. Управление временем, местом и количественными характеристиками экспрессии генов позволяет контролировать эволюция, влияя не только на конкретный ген, но и на весь организм.

Однако до сих пор не до конца ясно, как из клеток одного типа образуются принципиально разные ткани? Подробнее этот вопросы мы рассмотрим на примере эмбриогенеза дрозофил.

2.2 Формирование градиентов сигнальных веществ

Мушка *Drosophila melanogaster* была введена в качестве организма для наблюдения эмбриогенеза еще в 1909 году Томасом Морганом, и поныне она остается одним из самых любимых модельных организмов для исследователей эмбриогенеза. Рассмотрим ее развитие и мы.

Как и других многоклеточных, развитие дрозофил начинается с дробления яйца и гаструляции. В ходе последовательных митотических делений, выделяется три типа зародышевых клеток — энтодерма, мезодерма и эктодерма. К этому моменту в зарождающемся организме уже распределены (неравномерно) сигнальные вещества, которые после будут влиять на экспрессию генов.

Как же они распределяются? Основа закладывается во время оогенеза, еще задолго до оплодотворения и откладки яиц. Во время созревания ооцита (яйцеклетки) синтезируется большое количество РНК и белков. Развивающийся ооцит имеет *градиенты концентраций* мРНК. Гены, которые кодируют такие мРНК, оказывают большое влияние на развитие организма. В частности, у мушек-дрозофил эти гены называются bicoid, hunchback, nanos и caudal. Они отвечают за формирование осей тела. Когда мРНК генов транслируется в белки, образуются градиенты генов bicoid и nanos, на переднем и заднем полюсе яйца соответственно. Белок bicoid блокирует трансляцию мРНК белка cadual,

поэтому белки типа cadual образуются только на переднем полюсе. Аналогично на переднем полюсе белок nanos блокирует белок hunchback.

Таким образом, белки nanos, bicoid и hunchback являются факторами транскрипции. Воздействуя на их градиент, ученым удавалось получить мутировавших мушек-дрозофил, не имеющих, например, средней части тела.

2.3 Выбранная модель

При выборе модели реализации сначала необходимо определить, какие свойства объекта значимы и будут промоделированы, а какие — нет. Было решено абстрагироваться от ряда особенностей реальных организмов, таких как:

- изменения функций клеток, их неравномерного роста, запрограммированного апоптоза;
- трёхмерности организма, деления клеток на две половинки в трёх измерениях;
- функциональных (не сигнальных) веществ;
- взаимодействия веществ между собой, их распада, ингибирования;
- стохастическое взаимодействие оператора с репрессором;
- разделение ДНК на триплеты, гены, стоп-кодоны и пр.

Таким образом, выбранная модель отражает:

- начальную анизотропию веществ;
- изменение сигнальных веществ под действием генов;
- диффузию сигнальных веществ в соседние клетки;
- регуляцию генов.

Руководствуясь этими данными, было решено построить модель следующим образом.

Организм состоит из NxN клеток, каждая из клеток имеет вектор сигнальных веществ. Градиент концентрации первых двух сигнальных веществ (вертикального и горизонтального) отвечает за начальную анизотропию веществ. Также были выделены три не сигнальных вещества — цветовые компоненты.

Геном представлен в виде массива 16-битных слов. Каждое слово является либо оператором, либо частью оперона. Необходимость такого представления напрямую проистекает из выдвинутых требований — геном должен генерироваться эволюционным путём, причём механизм эволюции может менять случайные биты, вставлять случайное слово в случайную позицию, удалять слово из случайной позиции, копировать или переносить случайные фрагменты в рамках одного гена, создавать ген из двух других (конъюгация и кроссинговер) — кодирование должно быть таким, чтобы после этих преобразований оставался корректный геном. Подробнее о двоичном представлении генома рассказано в пункте 3.1.

Оперон включает в себя индекс вещества, на которое действует оперон, коэффициент, содержит коэффициент, определяющий интенсивность генерации или разрушения вещества. Также в опероне содержится знак, определяющий, увеличивается или уменьшается концентрация вещества, на которое влияет оперон.

Оператор включает в себя индекс вещества, которое отвечает за активацию гена, значение порога и также знак, определяющий взаимоотношение между значением вещества и порога.

Остановимся более подробно на алгоритме клеточной дифференциации. Он состоит из двух частей — подсчета изменений сигнальных веществ и перетекания сигнальных веществ из одной клетки в другую.

В первой части алгоритма для каждого гена, каждого оператора в нем, вычисляется разность между порогом и значением вещества или значением

вещества и порогом (на это влияет знак оператора). После, для каждого оперона в гене и каждого значения разности, вычисляется значение сигма-функции от разности, умножается на коэффициент взаимодействия из оперона и, в зависимости от знака оперона, прибавляется к текущему веществу или отнимается от него.

Решение вычислять значение сигма-функции от разности, а не работать с линейным приращением было принято вследствие стохастичности процесса экспрессии. На самом деле, либо ген полностью блокирован, либо полностью работает. Но репрессор может случайным образом прицепляться и отцепляться. Чем больше концентрация репрессора, тем чаще его молекула ингибирует экспрессию гена. Поэтому, вместо моделирования стохастического процесса, мы усредняем его, считая, что непрерывное изменение концентрации непрерывно меняет репрессию. Сигма-функция была выбрана из-за подходящей для этой модели асимптотики.

Вторая часть алгоритма включает в себя диффузию веществ между клетками. Процесс протекает независимо для каждой из компонент вектора веществ.

3.РЕАЛИЗАЦИЯ МОДЕЛИ ГЕНЕТИЧЕСКОГО КОДА С ПОМОЩЬЮ ТЕХНОЛОГИИ CUDA

3.1. Детали реализации

Для реализации модели был выбран язык C и технология CUDA. Таким образом, модель, представленная в пункте 2.3 реализуется в виде набора следующих структур и функций.

Структура организма:

```
struct creature{
    int n;
    struct cell* cells;
};
```

Структура состоит из двух полей — размера одной строки матрицы клеток и массива из структур клеток, представленных для удобства работы в виде одномерного массива.

Структура клетки:

```
struct cell{
    unsigned int v[SUBSTANCE_LENGTH];
    int dv[SUBSTANCE_LENGTH];
};
```

Структура клетки состоит из двух массивов. Массив v содержит текущие значения сигнальных веществ. Переменные v[0] и v[1] отвечают за положение клетки в организме (v[0] кодирует верхнюю часть организма, v[1] — левую). Переменные v[2] – v[4] отвечают за RGB-параметры модели. Массив dv содержит изменения веществ на текущей итерации алгоритма дифференцирования.

Структура генома:

```
struct genome{
    struct gene *genes;
    int length;
};
```

Структура генома состоит из массива генов и длины генома. Геном единственен для каждого организма. Разные организмы имеют различные геномы.

Структура гена:

```
struct gene{
    struct operator *operator;
    struct operon *operons;
    int cond_length;
    int oper_length;
};
```

Структура гена состоит из четырех компонент — векторов операторов и оперонов, и их длин.

Структура оперона:

```
struct operon{
    unsigned char rate : 7;
    unsigned char sign : 1;
    unsigned char substance : 7;
};
```

Структура оперона включает в себя индекс вещества, с которым взаимодействует оперон, коэффициент, влияющий на значении пороговой функции и знак, отвечающий за приращение или уменьшение сигнального вещества.

Структура оператора:

```
struct operator{
    unsigned char threshold : 7;
    unsigned char sign : 1;
    unsigned char substance : 7;
};
```

Структура оператора включает в себя индекс вещества и значение порога, которые влияют на активность оперонов. В дополнение к этому, также есть бит знака, влияющий на вычисление разности значений между пороговым веществом и значением порога.

Все вместе гены образуют геном, влияющий на регулировку экспрессии и, тем самым, на развитие организма.

Опишем более подробно представление генома в виде массива 16-битных слов.

В таблицах можно увидеть расположение веществ в машинном слове в записи little-endian.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sb	threshold							fb	substance						

Таблица 1. Представление оператора в машинной памяти

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sb	rate							fb	substance						

Таблица 2. Представление одной единицы транскрипции (слова оперона) в машинной памяти

Поясним обозначения, использующиеся в таблицах.

sb — бит, определяющий знак (поле sign в структурах operon и operator)

fb — бит, позволяющий отличить слово оперона от оператора

Остальные биты определяют соответствующие поля в структурах operon и operator.

Основная часть алгоритма регулировки экспрессии приведена далее:

```
void calc_diff(struct creature *c, struct genome genome){
(цикл по всем клеткам модели)
  (цикл по всем генам генома)
    (цикл по всем операторам)
    int delta[i] = cond.sign
        ? c->cells[cell].v[cond.subst] - cond.threshold
        : cond.threshold - c->cells[cell].v[cond.subst];
    (цикл по всем оперонам)
      (цикл по всем операторам)
       op.sign
 ? c->cells[cell].v[cond.subst]-=op.rate*calc_sigma(delta[i])
 : c->cells[cell].v[cond.subst]+=op.rate*calc_sigma(delta[i]);
}
```

Эта функция отражает первую часть алгоритма дифференцирования клеток — подсчет изменений сигнальных веществ. После ее выполнения вектор dv содержит изменения всех сигнальных веществ. Затем необходимо для каждого вещества в каждой клетке посчитать сумму текущего значения сигнального вещества и изменения и записать эту сумму в качестве нового значения.

Далее приведем примерную реализацию алгоритма, моделирующего диффузию веществ:

```
void blurKernel(struct creature* c,  struct matrix* m){
   (цикл по всем клеткам)
     int sz = m.size / 2;
     int core_point = x * c_size + y;
    float accum[SUBSTANCE_LENGTH] = { 0 };
     for (int k = -sz; k <= sz; k++){
      for (int l = -sz; l <= sz; l++){
        cell_i = x + k;
        cell_j = y + l;
        if(cell_j < 0)  cell_j = -l;
        if(cell_j >= c_size)  cell_j = c_size - l - 1;
        if(cell_i < 0)  cell_i = -k;
        if(cell_i >= c_size)  cell_i = c_size - k – 1;
       (цикл по всем веществам в клетке)
         int cur_subst=(cell_i*c_size+cell_j)*(SUB_LEN- 1)+p;
         int cur_val = (sz+k)*m->size+(sz+l);
        accum[p]+=(v[cur_subst]*m->val[cur_val]);
     (цикл по всем веществам в клетке)
        int cur_subst=(cell_i*c_size+cell_j)*(SUB_LEN-1)+p;
        dv[cur_subst] = (int)(accum[p])/m->norm_rate;
}
```

Таким образом, используя теоретические сведения, представленные ранее, реализуется простейшая модель развития многоклеточного организма.

3.2. Проблемы реализации

Одна из особенностей технологии CUDA — невозможность разыменования указателя на память GPU с помощью CPU. Очевидно, это затрудняет аллокацию и копирование вложенных структур. У этой проблемы есть ряд различных решений. Перечислим основные.

Первый способ — аллоцировать память и инициализировать память на GPU с помощью CPU. Однако во избежание указанной выше проблемы, необходимо создать в памяти хоста структуру с указателями на память устройства, а после инициализировать каждый из указателей отдельно. Для трехуровневых структур такой подход кажется неоправданно сложным.

Второй способ — аллоцировать память с помощью CPU, а инициализацию произвести в отдельном ядре, передав туда вектора веществ с хоста. Этот подход видится более удобным, но он оказывается и существенно более затратным с точки зрения ресурсов GPU.

И, наконец, третий способ — уплотнить структуру. По факту, наше существо представляется всего лишь как два массива целых чисел. Структура генома, конечно, несколько сложнее. Геном было решено представить как два массива — условий и оперонов, каждые два байта которого содержат один оперон или одно условие. Этот

способ, очевидно, содержит меньше всего аллокаций и меньше всего выполняет обращений к памяти устройства. Однако возникают определенные сложности с индексацией.

Довольно просто обратиться к определенному веществу в клетке. Для этого необходимо просто высчитать смещение по формуле:

```
(i * size + j) * (SUBSTANCE_LENGTH) + k
```

Здесь i, j — индексы клетки, size — размер существа, k — индекс вещества, к которому мы обращаемся.

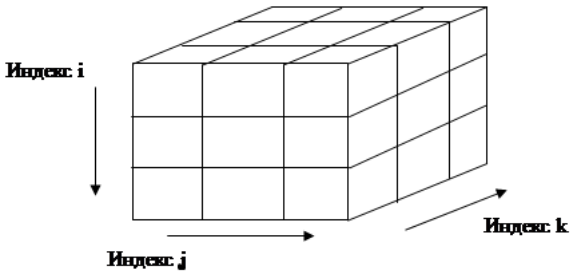

Рис.1 Модель существа

Чуть сложнее дела обстоят с индексацией веществ в геноме. Для этого были написаны специальные __device__ функции, которые осуществляли индексацию и выборку конкретных битов.

3.2. Параллельные вычисления

Основная польза технологии CUDA состоит в удобной параллельной обработке больших объемов примитивных данных. Для данной задачи это подходит как нельзя лучше.

Было принято решение запускать один поток CUDA для каждой клетки организма. Таким образом, необходимо было каким-то

образом динамически менять количество потоков во время выполнения в зависимости от размера существа.

Для модели видеокарты, на которой тестировалась программа, максимальное число потоков в блоке было 512. От этого числа и решено было отталкиваться. Тогда размер решетки вычисляется по простой формуле:

```
dim3 grSz;
grSz = (creature->n/threadNum+1, creature->n, 1);
```

Здесь creature->n — размер существа, threadNum = 512.

Тогда индексация клетки в ядре получается таким образом:

```
int y = blockDim.y*blockIdx.y + threadIdx.y;
int x = blockDim.x*blockIdx.x + threadIdx.x;
int cur_cell = y * creature_size + x;
```

Однако в таком случае может возникнуть проблема запуска лишних потоков для обработки изображения. Такие потоки необходимо завершать сразу же. Это делается простой проверкой индекса потока:

```
if (x >= creature_size || y >= creature_size)
    return;
```

Далее в ядре выполняется код функций, приведенных во втором этапе с некоторыми изменениями.

Общий алгоритм работы приложения реализован в функции main. Приведем псевдокод этой функции:

```
int main(int argc, char **argv){
    обработка аргументов командной строки;
    инициализация генома;
  инициализация существа, матрицы свертки;
  while(creature->n < MAX_CREATURE_SIZE){
    if(step != 0 && step % GROW_SIZE == 0){
       create_img(creature, path);
       grow(&creature);
    }
    calcWithCuda(creature, genome);
    apply_calc_changes(creature);
    blurWithCuda(creature, matrix);
    apply_blur_changes(creature);
         step++;
     }
    освобождение ресурсов;
```

-

4.ТЕСТИРОВАНИЕ

4.1. Ассемблирование генома

Одна из проблем тестирования приложения заключалась в сложной структуре задания генома. Это существенно затрудняло задание значений условий и оперонов.

Изначально была предложена идея, задавать все параметры простым текстовым файлом в виде пар ключ-значение. Однако количество таких пар могло доходить до тысячи, что, очевидно, не способствовало читаемости таких файлов. Тогда было решено представлять каждый ген генома строкой вида:

[<substance>] <sign> <threshold>, ... = <substance><sign><rate>, ...

Здесь слева от знака равенства стоят тройки, которые кодируют значения оператора, а справа — тройки, представляющие опероны. Каждая строка файла с описанием генома представляет собой один ген.

Однако такой геном, представленный в таком виде, неудобно преобразовывать непосредственно в структуру на языке С. Поэтому было решено сначала создавать по геному двоичный файл, который

содержал бы в себе только операторы и опероны. Специально для этой цели была написана программа на языке Python, осуществляющая разбор генома с помощью регулярных выражений.

В полученном двоичном файле каждые два байта обозначают оператора или оперон. Оператор от оперона отличается старшим битом. Новый ген генома определяется наличием перехода от оперона к оператору.

Загрузка генома из файла происходит в функции load_genome, находящейся в файле genome.cpp.

4.2. Результаты тестирования

Тестирование программы можно разделить на два этапа — тестирование ядра вычисления и прилегающих к нему функций и ядра свертки с вспомогательными функциями.

Для каждого этапа были подготовлены свои тесты.

Для тестирования ядра свертки был искусственно создан организм размерами 16x16 клеток, левая часть которого была инициализирована нулями, а правая — числом 255. Итоговый результат зависит от матрицы свертки. Эталонные значения были рассчитаны вручную. Варьируя исходные значения и матрицу свертки, были созданы разные наборы тестов.

Еще один тест ядра свертки состоял в наблюдении за растеканием веществ при пустом геноме. Для этого теста инициализировалось обычное существо. Для этого теста изображение

строилось не на основе цветовых компонент (вещества 2-4), а на основе компонент, отвечающих за «низ» и «право» организма (вещества 0 и 1).

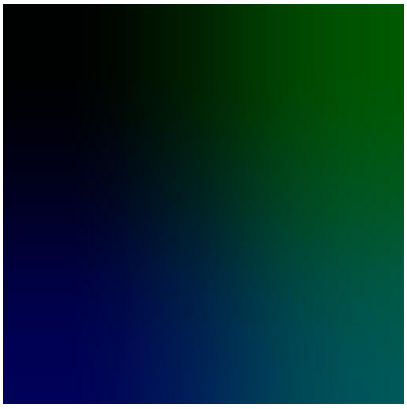

Рис.2. Результаты свертки при пустом геноме

Для тестирования работы ядра обсчета существа были разработаны несколько различных геномов, на которых и проверялась корректность результатов.

К примеру, геном (1):

```
[20] < 127 = [4] + 30
[0] > 0 = [5] + 15
[5] > 30 = [4] – 40
```

должен окрашивать верхнюю полуплоскость существа в синий цвет. Это происходит благодаря подавлению гена, отвечающего за окраску всего существа в синий цвет (первый ген) третьим геном.

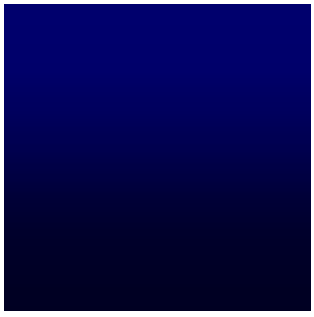

Рис.3. Результаты работы при геноме (1)

Приведем еще один пример генома (2):

```
[20] < 127 = [4] – 70
[1] > 0 = [5] + 15
[5] > 30 = [4] + 70

[30] < 127 = [4] + 70
[0] > 0 = [6] + 15
[6] > 30 = [4] – 70
```

Как можно увидеть на изображениях, изначально цветовая компонента подавляется первым геном. Затем, после наращивания компонент под номерами 5 и 6, которые подавляют проявления вещества вне необходимой зоны и усиливают его в верхнем правом углу, вещество начинает проявляться.

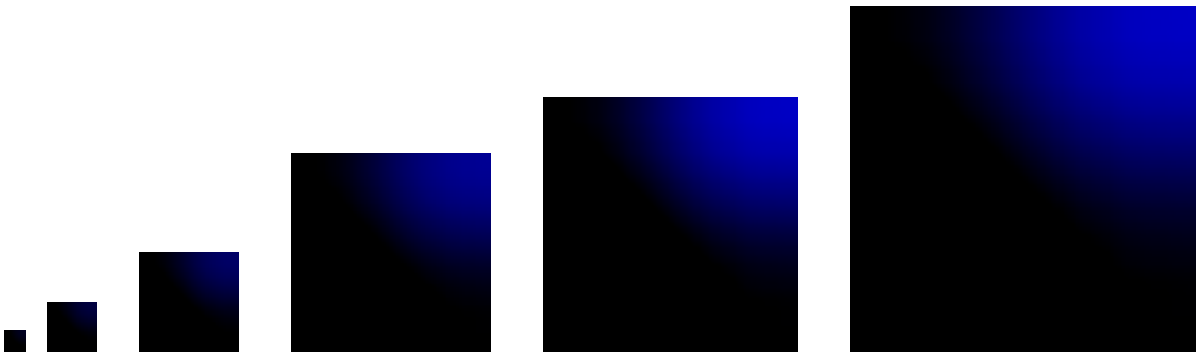

Рис. 3. Результаты работы генома (2)

Также покажем, как можно формировать более сложные изображения на примере генома (3):

```
[20] < 127 = [10] + 25
[1] > 0 = [5] + 15
```

```
[5] > 30 = [10] – 25

[30] < 127 = [10] - 25
[0] > 0 = [6] + 15
[6] > 30 = [10] + 25

[20] < 127 = [11] - 25
[1] > 0 = [7] + 15
[7] > 30 = [11] + 25

[30] < 127 = [11] + 25
[0] > 0 = [8] + 15
[8] > 30 = [11] - 25

[30] < 127 = [3] + 30
[10] > 10 = [3] - 30
[11] > 10 = [3] – 30
```

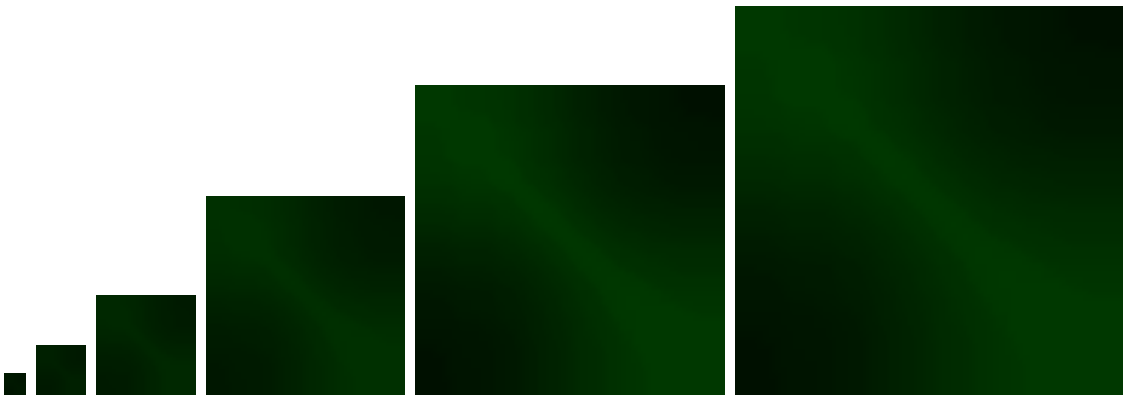

Рис.4. Результат работы генома (3)

Как можно заметить по текстовому описанию генома — изначально формируются сигнальные вещества 10 и 11 в верхнем правом и нижнем левом углу изображения. Они задаются сходным с веществом из генома (2) образом. Одновременно с этим, наращивается компонента, отвечающая зеленому цвету. После этого включаются гены, подавляющие компоненту зеленого цвета на участках, где сигнальные вещества 10 и 11 выражены наиболее сильно. Таким образом и формируется итоговое изображение.

ЗАКЛЮЧЕНИЕ

Результатом данной работы является приложение, разработанное с помощью технологии CUDA, для моделирования процесса дифференциации клеток многоклеточного организма в ходе онтогенеза. В процессе работы были изучены биологические основы развития организма, эвристические алгоритмы моделирования, выбрано и реализовано оптимальное решение.

Хочется надеяться, что данное приложение в сочетании с распределенной средой эволюции поможет кому-то чуть лучше понять, как именно происходит процесс дифференциации клеток в организме.

СПИСОК ЛИТЕРАТУРЫ

[1] CUDA C Best Practices Guide. — Version 3.2. — NVIDIA Corporation, 2010.

[2] КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ И МОДЕЛИРОВАНИЕ 2010 Т. 2 № 3 С. 295–308

[3] CUDA Toolkit Documentation — docs.nvidia.com/cuda/

[4] Шубин, Н. «Внутренняя рыба: история человеческого тела с древнейших времен до наших дней». — М.:Астрель : CORPUS, 2010 – 303с.

[5] «Биологический энциклопедический словарь.» Гл. ред. М. С. Гиляров; Редкол.: А. А. Бабаев, Г. Г. Винберг, Г. А. Заварзин и др. — 2-е изд., исправл. — М.: Сов. Энциклопедия, 1986.

[6] «Биология. Современная иллюстрированная энциклопедия.» Гл. ред. А. П. Горкин; М.:Росмэн, 2006.

[7] Патрушев Л. И. «Экспрессия генов» — М.: Наука, 2000. ISBN 5-02-001890-2

[8] Б. Аппель [и др.] «Нуклеиновые кислоты: от А до Я» — М.: Бином: Лаборатория знаний, 2013. — 413 с. — 700 экз. — ISBN 978-5-9963-0376-2

[9] Кудряшова Н.Ю., Кудряшов Ю.Б. «РНК-интерференция: к разгадке экспрессии генов // Биология в школе» — 2007. — № 2. — С. 7-11.