# Создание комплекса программ для численного моделирования магнетиков и устройств спинтроники.

Горячев И.А., Левченко В.Д.

8 декабря 2015 г.

## Содержание

# 1 Общие сведения

Комплекс программ имеет название LLGonGPU (модель Ландау-Лифшица-Ландау на GPU—ускорителях). Лежащая в основе комплекса программ математическая модель «атом-в-атом» требует привлечения высокопроизводительных вычислительных ресурсов. На сегодняшний день наиболее подходящим аппаратным решением для этих целей являются графические процессоры и графические ускорители, использование которых позволит проводить численное моделирование за обозримое время.

В качестве среды разработки программного комплекса выбирается технология CUDA, предлагаемая производителем графических процессоров NVIDIA. Для компиляции и выполнения программы требуется технология CUDA Toolkit версии не ниже 6.0. Помимо этого используется компиляторы C++ gcc, icc, llvm. Комплекс программ работает под операционной системой Linux. Для визуализации расчетов используются библиотека `openGL`, набор инструментов GLUT и набор программных модулей `im3D`.

Стандартная документация технологии CUDA содержит ряд указаний для разработчиков, которые связаны с такими факторами как векторизация и параллельность, выравненное хранение и доступ к памяти. Учет этих факторов дает возможность для создания высокопроизводительных кодов, однако не является достаточным условием.

Другой важный критерий — разработка структур данных для достижения эффективной работы с подсистемой памяти графических карт и высокопроизводительных алгоритмов для обработки этих структур, что позволит соблюсти баланс между темпом вычислений и латентностью памяти. Этот критерий опирается на два понятия: «локальность» и «асинхронность». Локализация данных для вычислений на разных уровнях иерархии памяти приводит к блочной оптимизации и вышеизложенному принципу «разделяй-и-властвуй». Это тот случай, когда действует параллелизм уровня данных (DLP). Асинхронность подразумевает независимое друг от друга выполнение инструкции и вычислений. То есть можно говорить об уровне параллелизма по инструкциям (ILP) и потокам (TLP), что подразумевает разбиение не только данных, но и выполнение команд по асинхронным вычислительным исполнителям.

Использование локально-рекурсивных нелокально-асинхронных алгоритмов позволит провести согласование между всеми уровнями параллелизма и рассматриваемой математической моделью с целью повешения производительности вычислений и сокращения полного времени моделирования.

# 2 Функциональное назначение

Проведение численного моделирования с помощью текущего кода покрывает класс задач, связанных с описанием динамики и распределения магнитных моментов в физических системах, включающие в себя магнитные материалы. Математическая модель принимает в рассмотрение каждый атом кристаллической решетки и состояние его собственного магнитного момента. Выбранная эволюционная модель позволяет учитывать магнитные моменты отдельного атома в различные моменты времени. Это соответствует так называемому атомистическому подходу или модели «атом-в-атом».

Подобного рода подход обеспечивает более точное описание физических процессов, происходящих в системе. Он позволяет учесть вкрапления специальных примесей и добавки нанометрового масштаба, неоднородности и дефекты материала, влияние шероховатостей и сложной формы границ, взаимодействие магнитных моментов разного сорта атомов в приграничном смежном слое — интерфейсе. Однако с вычислительной точки зрения атомистический подход является более ре-

сурсозатратным, что является причиной для разработки и внедрения новых алгоритмов для увеличения производительности расчетов и сокращения полного времени моделирования. Для этих целей привлекаются и разрабатываются локально-рекурсивные нелокально асинхронные (LRnLA) алгоритмы, которые позволяют учесть специфику архитектуры вычислительной системы и особенности математической модели.

В основе описания эволюционной модели лежат уравнения Ландау-Лифшица-Гильберт (ЛЛГ), где динамика магнитного момента каждого атома определяется эффективным магнитным полем в данной точке. В эффективное магнитное поле вносят свой вклад внешнее магнитное поле, наличие анизотропии в системе, обменное и магнитостатическое поля, а также случайные температурные флуктуации. Геометрия и свойства магнитного материала описываются пользователем в отдельном модуле. Задание свойств материалов включает в себя кубическую кристаллическую структуру со следующими видами подрешеток Браве: простая (SC), объёмноцентрированная (BCC), гранецентрированная (FCC).

Если рассматривать численное моделирование, описание которого близко к экспериментальным условиям, то возможны случаи неизотропной и неоднородной среды, наличия вкраплений и дефектов, а также гранулярной структуры магнетика. Внутренние свойства материала — анизотропия кристалла и межатомное обменное взаимодействие — носят локальный характер, и потому должны иметь собственные характеристические параметры (коэффициенты анизотропии, константы обменного взаимодействия и др.) для отдельной подобласти, гранулы или атома.

# 3 Описание логической структуры

Комплекс программ для моделирования магнитных материалов методом «атом-в-атом» состоит из нескольких функциональных подмодулей — подмодуль инициализации для задания кристаллической решетки в магнитной системе, подмодуль для диагностики и визуализации расчетов, вычислительный подмодуль с численными схемами. Их описание проводится в последующих разделах. На рисунке 1 приведена схематическая блок-схема процесса управления комплекса программ. В процессе управлении происходит запуск и выполнение задач на графическом и штатном (хост) устройствах. При запуске комплекса программ осуществляется разбор аргументов командной строки, где одним из параметров является выбор режима работы: тестовый, пакетный и интерактивный. Тестовый режим выполнятся в терминале, и предусмотрен для определения производительности вычислений и пропускной способности памяти. В терминале также выполняется пакетный режим, в котором после указанного числа итераций вычислительного процесса промежуточный результат сохраняется в постоянное запоминающее устройство (ПЗУ). Работа комплекса программ заканчивается после полного выполнения вычислительного цикла. Интерактивный режим предусматривает возможность просмотра результатов моделирования в реальном времени, проведения и сохранение в память ПЗУ результатов диагностики.

Процесс управления включает объявление основных алгоритмических структур, используемых в программе — grain-структур и grape-структур, — выделение необходимой области памяти с последующими вызовами инициализации и расчетного ядра. Взаимодействие алгоритмических структур с другими подмодулями происходит при помощи специальных методов загрузки и сохранения данных из памяти, получения индекса сорта и позиционных координат атома в магнитной системе. С точки зрения реализации экземпляр алгоритмических структур локализуется одновременно на двух устройствах — графическом и штатном. На графическом устройстве указатель на данные заносится в константную память в целях оптимизации темпа вычислений.


Рис. 1: Общая блок-схема процесса управления комплекса программ.

В интерактивном режиме происходит вызов средств визуализации с использованием сторонней библиотеки OpenGL [2]. Для этого находится подходящее графическое устройство, поддерживающее технологию CUDA, и связывается с устройством OpenGL после инициализации драйвера OpenGL. Затем создаётся пиксельный буфер в OpenGL и регистрируется в среде CUDA. Здесь же происходит настройка GLUT, запуск графического интерфейса и оконного менеджера. Интерактивный интерфейс организуется за счет стандартных функции `glutidlefunc(idle_func)`, `gglutkeyboardfunc(key_func)`, `gglutmousefunc(mouse_func)`, `gglutmotionfunc(motion_func)`, `gglutdisplayfunc(draw_func)` из библиотеки openGL, где `idle_func`, `key_func`, `mouse_func`, `motion_func` — статические функции, которые декларируются в глобальной памяти хост-устройства. В теле функции `key_func`, `mouse_func`, `motion_func` описываются действия, производимые при получении с внешних периферийных устройств — клавиатура или мышь — сигналов, называемых событиями. Между событиями циклически выполняется функция `idle_func`, в теле которой производится вызов вычислительного ядра.

## 3.1 Инициализация и задание кристаллической структуры

Инициализация математической модели происходит на штатном (CPU) устройстве. Результатом является анализ всей области моделирования, определение подобластей, не обладающих магнитными свойствами, а также индексация атомов. Индексация позволяет различить разные типы кристаллических решеток с разными сортами атомов вещества между собой. На основе индекса происходит обращение к физические параметрам того или иного материала. Физические параметры хранятся отдельно в константной или текстурной памяти графического устройства. При этом одному индексу не могут соответствовать два различных набора физических параметров.

Помимо этого определяются пары соседей, которые будут учитываться при вычислении обменного взаимодействия в расчетной части. На блок-схеме 2 приведена схематическая блок-схема инициализации модели в комплексе программ.

Первоначально пользователем указывается типы кристаллической решетки, из которых будет состоять образец. Тип подрешетки реализуется как наследуемая структура `crystalType` с виртуальными методами `set_id` и `set_mom`. Абстрактная структура `crystalType` содержит kern-ячейку с информацией об их магнитных моментах, координаты каждого атома, индексы атомов (листинг 1). Виртуальные методы `set_id` и `set_mom` определяются пользователем с указанием индекса и начальной конфигурации магнитных моментов. Атомы отдельно взятого типа подрешетки помечаются отдельным индексом, что соответствует отдельному сорту материала, из которого сделан образец.

Листинг 1: Декларация типа кристаллической структуры.

```
template <int Nat>
struct kern{ float3 m[Nat]; };

template <const int Nat>
struct crystalType{
  int id[Nat];
  float3 co[Nat];
  kern<Nat> kn;
  crystalType(){set_id(); set_mom()};
  virtual void set_id()=0;
  virtual void set_mom()=0;
```

```
};

typedef crystalType<2> kernBCC;
template <> void kernBCC::set_id(){
  id[0]=1; id[1]=2;
}
template <> void kernBCC::set_mom(){
  kn.m[0] = make_float3(1.f,0.f,0.f);
  kn.m[1] = make_float3(0.f,1.f,0.f);
}

typedef crystalType<4> kernFCC;
template <> void kernFCC::set_id(){
  d[0]=1; id[1]=2; id[2]=2, id[3]=2;
}
template <> void kernFCC::set_mom(){
  kn.m[0] = make_float3(1.f,0.f,0.f);
  kn.m[1] = make_float3(0.f,1.f,0.f);
  kn.m[2] = make_float3(0.f,1.f,0.f);
  kn.m[3] = make_float3(0.f,1.f,0.f);
}
```

Кристаллическая решетка реализуется как наследник абстрактного класса `crystal`, параметризованного по типу кристаллической решетки — кубическая, гранецентрированная, объемноцентрированная (листинг 2).

Листинг 2: Реализация задания кристаллической решетки с определенной пространственной конфигурацией.

```
struct Grid{
private:
  int xS, yS, zS;
  short int* id;
public:
  Grid();
  int Nmat;
  inline short int& get_id(int ix, int iy, int iz;)
  void make(int xs, int ys, int zs);
  void clean();
};

template<typename T>
class crystal {
private:
  int3 size;
  float3 dist;
  int x0, y0, z0;
  bool checkSize, checkDist, checkCenter;
public:
  T* lattice;
```

```
  crystal( );
  crystal(int xs, int ys, int zs);
  void set_dist(float x, float y, float z)
  void set_size(int xs, int ys, int zs);
  void set_size(float3 phys_size);
  void set_center(int x, int y, int z);
  inline T& get_kern(int ix, int iy, int iz);
  void create(Grid& grid);
  void rotate();
  virtual bool get_geom(int ix, int iy, int iz ) = 0;
  void clean();
};

class BCC: public crystal<kernBCC>{
  public: bool get_geom(int ix, int iy, int iz){return true;};
};

class FCC: public crystal<kernFCC>{
  public: bool get_geom(int ix, int iy, int iz){return true;}
};
```

В абстрактном классе `crystal` содержатся поля `dist` — межатомное расстояние внутри подрешетки в ангстремах, `size` размер (в количестве атомов вдоль каждого из трех направлений) подобласти с указанным типом кристалла. Таким образом, возможно задать размер системы либо в количестве атомов по каждой из осей, либо размер в нанометрах с указанием межатомного расстояния с помощью методов `set_size`. Подразумевается, что пользователь указывает размер параллелепипеда, из которого в дальнейшем будет получен нужной формы образец. Центр параллелепипеда указывается в полях `x0, y0, z0` абстракного класса `crystal` с помощью метода `set_center`.

Описание формы и геометрии образца проводится пользователем на основе виртуального метода `get_geom` на языке программирования C/C++. В листинге 2 приведен пример, когда кристалл занимает весь параллелепипед. Метод `get_geom` проверяет принадлежность атома подрешетки геометрии магнитного образца и индексирует атом нулевым значением в отрицательном случае. Входным параметром метода `get_geom` является пространственная координата атома, выходным — булевая переменная, которая указывает на принадлежность атома магнитному материалу. Сорта атомов, помеченные нулевыми индексами, полагаются отсутствующими, и не учитываются в вычислительной части. В программном комплексе реализованы стандартные геометрические примитивы — куб, шар, эллипсоид, которые могут быть использованы в методе `get_geom`. Метод `rotate` позволяет совершить поворот всей кристаллической решетки внутри геометрической области.

Листинг 3: Реализация инициализации двухслойной структуры ферромагнетик-антиферромагнетик с объемноцентрированной и гранецентрированной типами кристаллических решеток.

```
class grainMap: public Map{
public:
void create(){
  xSize=100; ySize=100; zSize=200;
```

```
  make_grid();
    FCC AF;
    AF.set_size(100, 100, 100);
    AF.set_center(50, 50, 50);
    AF.create(grid);
    BCC F;
    F.set_size(100, 100, 100);
    F.set_center(50, 50, 150);
    F.create(grid);
  make_interface();
  }
};
```

Построение модели опирается на предварительное составление карты, отражающей все подобласти с разными кристаллическими решетками, а также связи между атомами в обменном взаимодействии. Заполнение карты происходит в отнаследованном от `Map` классе. В листинге 3 приведена реализация двухслойного образца, состоящего из ферромагнитного и антиферромагнитного слоев. В виртуальном методе `create` сначала указывается размер всей моделируемой области либо в количестве атомов вдоль каждого из направления, либо в физических единицах с указанием межатомного расстояния. Затем вызывается метод `make_grid` для определения индексов ячеек и определения границы раздела между материалами. После этого задается слой антиферромагнетика с гранецентрированной решеткой размером $100 \times 100 \times 100$ атомов и с центром в точке (50, 50, 50). Метод `create` заполняет первую половину карты антиферромагнитным слоем и устанавливает межатомные связи для обменного взаимодействия. Нижний слой ферромагнетика определяется аналогично. В конце при вызове метода `make_interface` определяется граница размера материалов, и устанавливаются межатомные связи для обменного взаимодействия. Подсчет соседей на границе раздела сред для отдельного атома определяется исходя из радиуса взаимодействия `delta`, указываемого пользователем.

Как только карта будет получена, подобласти с проиндексированными ненулевым индексом атомами разбиваются на алгоритмические grain-кубы [3]. В одном Grain-кубе содержится восемь kern-ячеек. Отдельная kern-ячейка определяется на основании типа кристаллической решетки, из которого состоит образец [3].

На стадии инициализации проводится определение генератора случайных чисел для последующего учета термодинамических флуктуации в расчетном подмодуле. Инициализация генератора случайных чисел происходит в функции `init_rand`, выполняемой на графическом устройстве. При инициализации используется массив состояний (states), на основе которого последующее случайное число генерируется на базе предыдущих. Это помогает избежать повторяющихся последовательностей.

Генерация последовательностей осуществляется исходя из начального условия, называемого сидом (seed). Сид представляет собой целое число, которое определяется из индекса CUDA-блока, и является первым аргументом при вызове функции `curand_init`. Поскольку генерируется сразу несколько последовательностей случайных чисел, то каждый CUDA-тред выбирает собственную последовательность, опираясь на собственный номер в CUDA-блоке. Этот индекс указывается вторым аргументом при вызове функции `curand_init` Для упрощения последующей разработки создана структура normRand (листинг 4).


Рис. 2: Общая блок-схема инициализации модели.

Листинг 4: Реализация генератора случайных чисел в комплексе программ.

```
#define CUDA_CALL(x) do {                                          \
        if((x)!=cudaSuccess) {                                     \
                printf("Error at %s:%d\n",__FILE__,__LINE__);      \
                throw -1;                                          \
        } } while(0)
#define CURAND_CALL(x) do {                                        \
        if((x)!=CURAND_STATUS_SUCCESS) {                           \
                printf("Error at %s:%d\n",__FILE__,__LINE__);      \
                throw -1;                                          \
        } } while(0)
#include <curand.h>
#include <curand_kernel.h>

__global__ void init_rand(curandState *states){
  unsigned int tid = threadIdx.x + blockDim.x * blockIdx.x;
  curand_init(blockIdx.x, threadIdx.x, 0, &states[tid]);
};

struct normRand{
  curandState *states;
```

```
  void initRand(){
    CUDA_CALL(cudaMalloc((void **)&states, NA*NV*sizeof(curandState)));
    CUDA_CALL(cudaThreadSynchronize());
    init_rand<<< NA , NV >>>(states);
    CUDA_CALL(cudaThreadSynchronize());
  }
  void clear();
};
```

Макросы `CUDA_CALL` и `CURAND_CALL` позволяют обрабатывать сообщения об ошибках выполнения, возвращаемые средой разработки CUDA. Метод `clear` удаляет массив состояний из памяти графического устройства. Экземпляр структуры `normRand` хранится в константной памяти графического устройства.

## 3.2 Хранение данных

Магнитный образец может содержать в себе как ферромагнитный, так и антиферромагнитный материалы. Каждый материал имеет свою кристаллическую решетку — простую (BC), объемноцентрированную (BCC) или гранецентрированную (FCC). Согласно базовым алгоритмическим принципам [3] кристаллическая структура образца делится на grape-кластеры. В свою очередь, в целях оптимального хранения данных grape-кластер делится на grain-кубы.

Хранение информации об отдельной элементарной ячейке внутри grain-куба организуется в алгоритмических kern-ячейках. Kern-ячейка содержит в себе различные комбинации атомов в зависимости от вида кристаллической решетки [3]. Алгоритмическая kern-ячейка простой решетки охватывает 1 атом; объемноцентрированной — 2 атома разных подрешеток, сдвинутых по пространству друг относительно друга на половину атомного расстояния по каждой из осей; гранецентрированной решетки — 1 атом одной подрешетки, лежащий в вершине куба, и 3 атома остальных подрешеток, лежащих в центре смежных граней. Алгоритмическая kern-ячейка содержит 3 компоненты вектора магнитного момента $\vec{m} = (M_x, M_y, M_z)$ каждого входящего атома.

Стандартный для CUDA тип данных `float3` не подходит для хранения трех компонент вектора $\vec{m}$ по причине возникновения конфликта при обращении CUDA-варпа (CUDA-warp) в невыравненную (aligned) память. Бесконфликтный доступ в выравненную память с учетом коалесинга (coalescing) обеспечивает хранение 4 чисел с одинарной точностью, что отвечает стандартному CUDA типу `float4`. Для оптимального хранения данных в трех переменных типа `float4` помещаются компоненты четырех векторов магнитного момента. Зафиксировав 16 байт (`float4` ) на атом, определим размер grain-подкуба, состоящего из одного сорта элементарных ячеек, равным $2 \times 2 \times 2$, что в сумме дает 128 байт — размер одной строки в организации памяти. Целый grain-куб включает подкубы всех сортов атомов в кристаллической решетке. Размер целого grain-куба полагается равным количеству элементарных ячеек в одной алгоритмической kern-ячейке, помноженному на 128 байт, и определяется согласно таблице 7. В листинге 5 представлена реализация grain-куба на языке программирования C/C++.

Листинг 5: Реализация grain-куба в комплексе программ.

```
struct Grain { floatT4 m[K4gn*3/2]; };
```

Размер одного grape-кластера регламентируется количеством kern-ячеек (табл. 7). Количество kern-ячеек в grape-кластере ограничивается сверху количеством CUDA-нитей (CUDA-thread), которые возможно запустить в одном CUDA-блоке. Это ограничение вызвано тем, что отдельная

CUDA-нить вычисляет эволюцию магнитного момента атомов кристаллической решетки, принадлежащих отдельной kern-ячейке. Таким образом, имеется взаимнооднозначное соответствие между элементами программной модели CUDA и структурного элемента алгоритма. Все kern-ячейки хранятся в виде массива [3] и обращение к каждому элементу этого массива происходит через индексы CUDA-нитей и CUDA-блоков(CUDA-block).

Листинг 6: Реализация grape-кластера в комплексе программ.

```
template <int m4k>
struct Grape {
  struct Grain { floatT4 m[k4gn*3/2]; } grains[gn4gp];
__device__ __forceinline__ void set_mom(float3 m);
__device__ __forceinline__ void set_mom(floatT mx,floatT my,floatT mz,char mid=0)
__device__ __forceinline__ void set_mom(float3 m1, float3 m2);
__device__ __forceinline__ void set_mom(float3 m1,float3 m2,char mid1,char mid2);
__device__ __forceinline__ float3 get_mom();
__device__ __forceinline__ void get_mom(float3& rm1, float3& rm2);
}
```

В целях упрощения разработки комплекса программ внутри grape-структуры внесены методы сохранения и загрузки информации о магнитных моментах и kern-ячейках. С помощью методов `set_mom` можно сохранять целиком трехкомпонентные вектора, либо вектор с указанием каждой компоненты по отдельности и сорта атома, к которому принадлежит вектор магнитного момента. Методы `set_mom` также позволяет сохранить kern-ячейку целиком с указанием тех сортов атомов подрешеток, которые затрагивает kern-ячейка. Методы `get_mom`, наоборот, предназначены для загрузки информации магнитного момента или kern-ячейки из grape-структуры.

Все grape-структуры, которые исполняются на одном выделенном мультипроцессоре, объединяются в **bunch**-потоки. Организация bunch-потоков осуществляется на уровне планировщика задач на одном мультипроцессоре, и их запускаемое количество на одной графической плате (GPU) определяется количеством SM-процессоров. Bunch-потоки объединяются в vine-структуры.

## 3.3 Расчетный подмодуль

Основной расчет эволюции намагниченности образца проводится по модифицированной схеме Рунге-Кутты четвертого порядка. Это подразумевает продвижение эволюции системы на один шаг по времени. Подсчет эффективного магнитного поля происходит на каждой стадии численной схемы, где учитываются внешнее поле, анизотропия, обменное взаимодействие, влияние полей демагнитизации, вычисленных на предыдущем шаге по времени. Вклад температурных флуктуаций определяется после проведения четырех стадий численной схемы.

Выполнение численной схемы происходит в CUDA-ядре `stencil`. Соответствующие одному grape-кластеру данные хранятся в глобальной памяти графического устройства (GPU). CUDA-нить загружает по одной kern-ячейке в регистровый файл на основании своего собственного индекса. Загрузка происходит через систему кэшей графического процессора по запросам. На один запрос происходит загрузка 128 байтового слова или одного алгоритмического grain-подкуба.

Один grape-кластер обрабатывается одним CUDA-блоком на скалярном мультипроцессоре (SM) графического процессора. В Grape-кластере происходит вычисление до 1024 kern-ячеек на одном SM-процессоре современных графических систем (кроме nVidia Tesla K80, где это число удваивается). Описание параметризованной grape-структуры включено в листинг 6. Количество атомов,


Рис. 3: Блок-схема расчетного модуля.

лежащих в kern-ячейке, указывается в параметре шаблона.

Для пересчета обменного взаимодействия отдельные grain-кубы, принадлежащие одному grape-кластеру, выполняются на отдельном мультипроцессоре, и данные о значениях магнитных моментов сохраняются в разделяемую (shared) память. Это позволяет отдельным CUDA-тредам из разных CUDA-варпов обмениваться информацией, которая локализована в регистрах. CUDA-треды, принадлежащие одному CUDA-варпу, обмениваются данными посредством шафл-функции (shuffle).

В гамильтониане атомистической модели учитывается влияние термодинамических флуктуации, что подразумевает в математическом описании дополнительный вклад от случайного источника в эволюцию магнитного момента после проведения всех четырех стадий численной схемы. Случайный источник представляет собой трехмерный вектор $\xi = (\xi_x, \xi_y, \xi_z)$, каждая компонента которого выбирается согласно нормальному распределению случайных чисел. Общая формула для вклада термодинамических флуктуации записывается как $\mathbf{H_{therm}} = 2\sqrt{\alpha T_h h}\cdot[\mathbf{m_0}\times\xi]$, где параметры $\alpha$ — коэффициент затухания, $T_h$ — значение температуры, $h$ — шаг численной схемы, являются входными параметрами. Трехмерный вектор $\mathbf{m_0}$ есть начальный вектор намагниченности перед выполнением четырехстадийной численной схемы.

В реализации источника термодиначеских флуктуаций использовалась сторонняя библиотека для генерации случайный чисел cuRand, предоставляемая корпорацией Nvidia.

При каждом выполнении численной схемы отдельный CUDA-тред копирует в регистровый файл информацию о текущем состоянии генератора случайных чисел из массива состояний. Затем выполняется метод `get_rand`, на выходе которого получается трехмерный вектор $\xi$, используемый в дальнейшем при расчете термодинамических флуктуаций.

Листинг 7: Реализация grape-кластера в комплексе программ.

```
float3 __device__ __forceinline__ get_rand(curandState *state) {
  float3 res=make_float3(curand_normal(state), curand_normal(state),
     curand_normal(state));
  return res;
}
```

По окончании выполнения всех стадий численной схемы происходит окончание выполнения вычислений на текущем временном шаге и передача управления подмодулю для диагностики и визуализации результатов в случае выбора интерактивного режима.

## 3.4 Вычисление преобразования поворота

Используемая в комплексе программ численная схема оперирует преобразованиями поворота вектора вокруг выделенной оси на заданный угол [3]. Согласно общепринятой формуле линейной алгебры поворот трехкомпонетного вектора вокруг оси, заданной единичной направляющей $\mathbf{u} = (u_x, u_y, u_z)$, на угол $\theta$ осуществляется за счет умножения на невырожденную матрицу поворота третьего ранга:

$$\begin{pmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{pmatrix}$$

Перемножение матрицы на вектор требует 9 арифметических операций «сложение-умножение» (FMA). Помимо этого требуется дополнительные накладные расходы на подготовку самой матрицы поворота. Темп вычисления тригонометрических функции согласно стандартной документации

[1] равносилен 6 FMA-арифметическим операциям на мультипроцессор. В общей сложности необходимо порядка 60 FMA операции для вычисления преобразования поворота методами линейной алгебры.

С математической точки зрения оператор пороворота может быть представлен помимо как элемент из категории невырожденных матриц, но и как элемент гипергеометрических чисел — кватернионов. Поворот трехмерного вектора $\mathbf{v}$ вокруг фиксированного вектора $\mathbf{u}$ единичной длины в этом случае записывается через мнимую часть гипергеометрического числа в виде формулы:

$$\mathbf{v}' = 2(\mathbf{u}' \cdot \mathbf{v}) \cdot \mathbf{u}' + \left(s^2 - (\mathbf{u}')^2\right) \cdot \mathbf{v} + 2s \cdot [\mathbf{u}' \times \mathbf{v}], \tag{1}$$

где $\mathbf{v}'$ — результирующий вектор, $s = \cos(\alpha)$, $\mathbf{u}' = \mathbf{u} \cdot \sin(\alpha)$, $\alpha$ — угол поворота.

На основе формулы (8) реализована универсальная функция, запуск которой может происходить как с графических устройств (GPGPU), так и под управлением штатного процессора(CPU).

Листинг 8: Реализация преобразования поворота в комплексе программ.

```
__forceinline__ __host__ __device__ float3 rotate(float3 v, float3 H){
  register float a = length(H);
  if(a < 1e-15) return v;
  #if defined(__CUDA_ARCH__)
  register float3 u = __sinf(a)*normalize(H);
  register float  s = __cosf(a);
  #else
  register float3 u = sinf(a)*normalize(H);
  register float  s = cosf(a);
  #endif
  return (2.0f*dot(u, v))*u + (s*s-dot(u, u))*v + (2.0f*s)*cross(u, v);
};/* rotate */
```

На вход функции `rotate` подается два трехкомпонентных вектора $\mathbf{v}$ и $\mathbf{H}$. Результатом её выполнения является вектор, полученный путем поворота вектора $\mathbf{v}$ вокруг вектора $\mathbf{H}$ на угол $a$. Угол $a$ вычисляется на основании длины вектора $\mathbf{H}$. В реализации использованы оптимизированные тригонометрические функции `__cosf(a), __sin(a)`, входящие в пакет CUDA и предоставляемые компанией NVIDIA. Вычисления базовых операции линейной алгебры происходят с помощью оптимизированных функции скалярного произведения (dot), векторного произведения (cross), нормировки вектора на единицу длины (normalize) и вычисления длины вектора (length), которые также входят в пакет CUDA. Накладные расходы в этом случае уменьшаются до 35 FMA-операциям.

Согласно теоретическим оценкам прирост производительности при использовании формулы для кватернионов увеличивается почти в 2 раза, что позволяет сократить общее время моделирования.

# 4 Используемые технические средства

В качестве вычислительной архитектуры выбираются графические устройства общего назначения (GPGPU), использование которых подразумевает ускорение по сравнению со штатными процессорами (CPU) в 10 — 100 раз. Комплекс программ может быть развернут на персональном компьютере, а также на узлах выислительного кластера при условии наличия в них видеокарт компании NVIDIA с архитектурой Kepler (cc 3.5) или Maxwell. Для компиляции и выполнения программы

требуется инструмент CUDA Toolkit версии не ниже 6.0. Помимо этого используется компилятор C++ gcc, icc, llvm. Комплекс программ работает под операционной системой Linux.

# 5 Вызов и загрузка

Вызов комплекса программ производится в терминальном режиме. При запуске комплекса программ могут быть указаны дополнительные опции (листинг 9).

Листинг 9: Список опции командной строки и их значения по умолчанию.

```
[--devQ] [--zoom "1. 1. 1."] [--step "1. 1. 1."] [--base "1. 1. 1."]
[--box "1. 1. 1."] [--mesh "200. 200. 200."] [--Dmesh 5.] [--drop_dir "."]
[--bkgr_col "0.1 0.1 0.1"] [--mesh_col "0.8 0.8 0.2"][--box_col "1. 1. 1."]
[--box_shrink "1. 1. 1."] [--sensor "1 1 1"] [--help|--test|--batch]
```

Описание опций приведено в таблице 1.

| Опция | Описание опции |
|---|---|
| devQ | Выдаёт информацию о видеокартах на компьютере |
| zoom | Масштабный фактор размера окна |
| box | Коррекция пропорций размера бокса в 3D режиме |
| step | Шаги между точками, действует только на тики |
| base | Шаги между точками, действует только на тики |
| mesh | Расстояние между линиями сетки в боксе по координатам в ячейках (до коррекции) |
| Dmesh | Ширина линии сетки в пикселях |
| drop_dir | Имя директории для сохранения результатов диагностики |
| bkgr_col | Цвет фона |
| mesh_col | Цвет линий сетки |
| box_col | Цвет линий бокса |
| box_shrink | Коэффициент растяжения размеров бокса |
| sensor | Координаты сенсоров |
| test | Запуск тестового режима |
| batch | Запуск пакетного режима |
| help | Вызов справки |

Таблица 1: Таблица описания опции командной строки.

Аттрибуты «бокс», «фон», «сетка» используются подмодулем визуализатора и могут быть заданы по умолчанию. Сенсоры и их координаты указываются пользователем в комплексе программ для последующей диагностики.

При запуске комплекса программ осуществляется разбор аргументов командной строки, где одним из параметров является выбор режима работы: тестовый, пакетный и интерактивный. Тестовый режим выполнятся в терминале, и предусмотрен для определения производительности вычислений и пропускной способности памяти. В терминале также выполняется пакетный режим, в котором после указанного числа итераций вычислительного процесса расчет останавливается, результат сохраняется в постоянное запоминающее устройство (ПЗУ), работа комплекса программ заканчивается. Интерактивный режим запускается по умолчанию и предусматривает возможность просмотра результатов моделирования в реальном времени, проведения и сохранение в память ПЗУ результатов диагностики. В интерактивном режиме предусмотрена обработка событий поступаю-

щих после нажатия клавиш с периферийного устройства — клавиатуры. Список запрограммированных клавиш представлен в таблице 2.

| Комбинация клавиш | Результат обработки события |
|---|---|
| Enter | Пересчёт одного временного шага |
| \| | Изменение функции для визуализации: F\|G\|c |
| ESC | Выход из программы |
| 3 , 2 | Переключает рендеринг |
| w , W | Сохранение текущего набора опций визуализации в файл «im3DI.opt» или в произвольное число наборов файлов последовательно |
| r , R | Загрузка ранее сохранённых наборов опций последовательно или загрузка без перехода к следующему набору |
| Ctr-r | Сброс параметров в значения по умолчанию |
| f , F | Переход к началу или концу файла сохранённых наборов опций |
| v , V | Увеличение или уменьшение уровня вывода диагностики |
| Ctr-v | Печатает диагностику, особено актуально, если заголовок окна не виден |
| Ctr-w | Переключает режим вывода в заголовок окна диагностики по умолчанию |
| s , S | Сохранение картинки в формате png или вместе с зарамочным оформлением в gnuplot |
| # , @, $ | В режиме 3D переключение режима фона: сетка, сохранённая картинка, рёбра бокса |
| ! | Сохранить картинку для фона |
| m , M | Уменьшение или увеличение шага вдоль луча для соответствующего изменения точности (2) |
| e , E | Размазывание луча по горизонтали для соответствующего изменения муара (0.25) |
| d , D | Увеличение или уменьшение плотности цвета при суммировании вдоль луча (0.5) |
| a , A | Установка пределов палитры из пределов текущего массива или из значений fMin..fMax |
| Ctr-a | Установка значений fMin..fMax из текущих пределов палитры |
| Ctr-s | Установка пределов палитры, используя пределы массива в сечении поперёк выбранной оси |
| 1 | Переключает (циклически, по xyz) ось, вдоль которой строится одномерный график в gnuplot (x) |
| o , Ctr-o | Выводит в окно gnuplot сечение вдоль выбранной оси без перерисовки или с перерисовкой |
| 0 , Q | Для точки (x0,y0,z0): Печатает в терминале значение текущего поля и выводит в файл сечения вдоль лучей, проходящих через неё.Добавляет сенсор |
| q , Ctr-q | Сохраняет значения сенсоров в файле sensors.dat или выводит в окно gnuplot запись сенсоров |

Таблица 2: Общее управление программой.

| Комбинация клавиш | Результат обработки события |
|---|---|
| b | Включает пересчёт в динамике |
| g , G | Отключение\|включение постоянной перерисовки в цикле GLUT (0) |
| yz , XY | Вращение вокруг осей x,y вперёд\|назад |
| z , Z | Приближение\|удаление объекта |

Таблица 3: Общее управление динамикой расчета.

| Комбинация клавиш | Результат обработки события |
|---|---|
| L , R , M | вращение\|изменение масштаба\|сдвиг рисунка |
| L , R , M | устанавливает нижний\|верхний пределы\|центр палитры, исходя из x-координаты выбранной точки |
| L , R | В режиме «Ctl-t» (бинарной прозрачности) делает цвет прозрачным\|видимым |

Таблица 4: Общее управление периферийным устройством типа «мышь».

| Комбинация клавиш | Результат обработки события |
|---|---|
| p , P | Выбор палитры, в порядке уменьшения или увеличения числа цветов |
| Ctl-p | Переключение вывода палитры на верх экрана |
| = , - | Точное (в 10 раз за 10 кликов) уменьшение или увеличение пределов по цветовой оси |
| + , _ | Грубое (в 10 раз) уменьшение или увеличение пределов по цветовой оси (0<f<1) |
| 0 , ) , ( | Центрирование пределов палитры или установка в 0 значения левого и правого пределов |
| 9 , 8 | Установка пределов палитры в значения [1..9] или [-1..1] |
| Ctl-c | Переключение палитры из линейной в дискретную и обратно (0). В частности - для уменьшения размера png |
| c , C | Переключение круговой или центрированной палитры на ограниченную пределами и обратно |
| t , T | Увеличение или уменьшение числа прозрачных цветов в палитре, (1) д.б. делителем числа цветов в палитре (20) |
| Ctl-t | Переключение на режим, в котором прозрачность каждого цвета в палитре кодируется битом числа 1 (0) |
| [ , ] | Уменьшение или увеличение показателя степени при нелинейном шкалировании по цветовой оси (1, шаг 0) |
| / , \ | Уменьшение или увеличение яркости цветов в палитре, важно для круговой (до 1, шаг 0) |
| ? , | Уменьшение или увеличение относительной яркости разных циклов в круговой палитре (1.09545 раз, шаг 1) |
| Ctr-r | Сброс параметров в значения по умолчанию |
| TAB | Инверсия цветов в палитре (кроме чёрного), для im3D решает проблему «грязи» на светлом фоне |

Таблица 5: Общее управление цветовой палитрой.

# 6 Входные данные

Согласно базовой вычислительной модели локально-рекурсивных нелокально-асинхронных алгоритмов (LRnLA) [3] расчетная задача связана с аппаратным вычислителем, на котором данная задача выполняется. Подобного рода связь содержит в себе аппаратные, алгоритмические, численные и физические параметры. В последующих подразделах представлено описание и табулирование этих параметров.

## 6.1 Аппаратные параметры

Производители графических устройств ежегодно представляют большое количество аппаратных решений, каждое из которых имеет свою спецификацию. В целях унификации аппаратных решений для дальнейшего использования в реализации локально-рекурсивных нелокально-асинхронных алгоритмов выделяется ряд аппаратные параметров, которые напрямую связаны как с характеристиками графических устройств, так и с программной средой CUDA. Аппаратными параметрами являются число мультипроцессоров графического процессора, размер CUDA-варпа, число элементов в CUDA-векторе, число синхронно и асинхронно выполняемых задач. Под выполняемой задачей согласно [3] подразумевается обработка одного grape-кластера мультипроцессором. Поскольку размер и количество асинхронно выполняющихся grape-кластеров диктуются уровнем параллелизма графического вычислительного узла, то можно выделить базовые параметры: CUDA-вектор и число мульпроцессоров. Все остальные аппаратные параметры выражаются через базовые домножением на целое положительное число.

| Описание параметра | тип | обозначение | значение по умолчанию |
|---|---|---|---|
| Размер CUDA-варпа | `const int` | NW | 32 |
| Число мультипроцессоров в графическом процессоре | `int` | NSM | 8 |
| Число элементов в CUDA-векторе (CUDA-блоке) | `const int` | NV | $NW \cdot 32$ |
| Число асинхронных задач кратное числу мультипроцессоров | `int` | NA | $NSM \cdot 16$ |
| Число синхронных задач | `int` | NS | NA |

Таблица 6: Таблица общих входных аппаратных параметров.

## 6.2 Алгоритмические параметры

Подробное описание алгоритмических структур представлено в [3]. Размер этих структур незафиксирован явным образом, и задается алгоритмическими параметрами. Часть алгоритмических параметров определяется на основе спецификации графических процессоров — число Grape-структур в wine-образце, число kern-ячеек и число grain-кубов в grape-кластере, число атомов в каждой алгоритмической структуре. Другая часть алгоритмических параметров зависит от строения кристаллической решетки — kern-ячейка — и от методов оптимизации доступа к памяти — grain-куб [3]. Базовыми параметрами в этом случае являются размер grain-куба и количество атомов образца в элементарной kern-ячейке. Лишний раз стоит напомнить, что каждому отдельному атому соответствует собственный трехмерный вектор магнитного момента. В таблице 7 представлено описание и табулирование алгоритмических параметров.

| Описание параметра | тип | обозначение | значение по умолчанию |
|---|---|---|---|
| Число Grape-структур в Wine | `int` | Ngrapes | NA · NS |
| Число моментов атомов в kern-ячейке | `int` | M4kn | 2 |
| Число kern-ячеек в Grain-структуре | `int` | K4gn | 8 |
| Число kern-ячеек в Grape-структуре | `int` | K4gp | NV |
| Число Grain-структур в Grape-структуре | `int` | gn4gp | K4gp / K4gn |
| Число моментов атомов в Grain-структуре | `int` | M4gn | K4gn · M4kn |
| Число моментов атомов в Grape-структуре | `int` | M4gp | M4gn · gn4gp |
| Число моментов атомов в Wine-структуре | `int` | M4wine | Ngrapes · M4gp |

Таблица 7: Таблица общих входных алгоритмических параметров.

## 6.3 Численные и физические параметры

Физические величины соотносятся с их обезразмеренными значениями в комплексе программ согласно таблице 8.

| Физическая величина | Обозначение | Единица измерения обезразмеривающего коэффициента | Значение обезразмеривающего коэффициента |
|---|---|---|---|
| Атомный магнитный момент | $\mu_s$ | Джоуль/Тесла $[JT^{-1}]$ | 1 |
| Энергия обменного взаимодействия | $J_{i,j}$ | Джоуль/связь $[J/link]$ | 1 |
| Энергия анизотропии | $k_u$ | Джоуль/атом $[J/atom]$ | 1 |
| Магнитное поле | $\mathbf{H}$ | Тесла $[T]$ | 1 |
| Значение температуры | $k_BT$ | Джоуль $[J]$ | 1 |
| Время физического процесса | $t$ | Пикосекунда $[ps]$ | $\frac{1}{\|\mathbf{H}\|}$ |
| Межатомное расстояние | $a$ | Ангстрем [Å] | $\frac{1}{4\|\mathbf{H}\|}$ |

Таблица 8: Обезразмеривание атомной системы единиц ($\frac{m_e}{e} = 1$ ).
$k_B = 1.3807 \times 10^{-23}$ Джоуль/К — константа Больцмана.
Последующее описание входных физических параметров в таблице 9 будет проводиться в соответствии с безразмерными параметрами, представленными в таблице 8.

| Описание параметра | тип | обозначение | значение по умолчанию |
|---|---|---|---|
| максимальное количество сортов атомов (для типов параметров обменного взаимодействия J) | `const int` | AtomSpecs | 2 |
| Физический размер области, исчисляемый в единицах межатомного расстояния | `int` | xSize, ySize, zSize | |
| Радиус цилиндрической формы образца | `float` | R | min (xSize/2, ySize/2) |
| Коэффициент затухания | `float` | alpha | 0.01 |
| Численый шаг схемы | `float` | h | 0.01 |
| Значение температуры | `float` | Th | 1.0 |
| Значения компонент магнитного поля | `float` | Hx, Hy, Hz | 0.0 |
| Параметры анизотропии | `float` | Ku | 1.0 |
| Параметры обменного взаимодействия | `float` | J | 1.0 |

Таблица 9: Таблица общих входных физических параметров

# 7 Выходные данные

Вне зависимости от выбора предоставленного режима на начальном этапе работы комплекса программ в окно терминала поступает стандартная информация, содержащая:

- размер системы в атомных расстояниях,
- размер каждой алгоритмической структуры,
- время инициализации,
- количество полной, свободной и занятой памяти на графическом устройстве,
- количество всех grape-структур и grape-структур с ненулевым индексом.

Указанные пользователем сенсоры выводят в ПЗУ информацию о конпонентах магнитного момента выделенного атома в файл формата `*.dat`. Последовательность вывода информации с сенсоров через табуляцию следующая: временной шаг численной схемы, компонета $M_x$, компонета $M_y$, компонета $M_z$, перевод каретки на новую строку. Данный формат вывода удобен для использования в сторонней программе `gnuplot` построения графиков и схем.
Также в файл формата `*.dat` происходит вывод через табуляцию информации об интегральных характеристиках системы — среднем значении магнитного момента, среднего квадрата магнитного момента, средней и полной энергии — для каждого временного шага численной схемы.
Вывод информация о векторных полях (в том числе пространственное распределение эффективного магнитного поля) на каждом шаге по времени численной схемы происходит в формате программного модуля im3D.
Помимо этого предусмотрен формат вывода для визуализатора Mview.

## 7.1 Выходные данные в тестовом режиме

Результатом выполнения комплекса программ в тестовом режиме информация о полном времени выполнения расчета, а также темп загрузки, сохранения данных и обсчета численной схемы, измеренный в количестве миллиардов атомов (моментов) в секунду (листинг 10).

Листинг 10: Поступающие на терминал выходные данные тестового режима.

```
Total amount of atoms (moments):: 12845056
Sizes: Grain: 192B  Grape: 14.00KB  Vine: 171.50MB
10624 not-void of 12544 grapes
Init Array 112x112x512: 106.7 msec
Total GPU Mem Size ::       2146762752  (    2047 MB)
Busy  GPU Mem Size ::        296992768  (     283 MB)
Free  GPU Mem Size ::       1849769984  (    1764 MB)
Test only
Load  LdSt  Calc  Stencil  Total time, sec
7.12  3.32  0.87  0.35     57.28 msec
....  ....  ....  ....     ....
```

## 7.2 Выходные данные в пакетном режиме

Пакетный режим предусматривает проведение расчета без привлечения функциональной части подмодуля визуализатора, что позволяет ускорить высоконагруженные вычисления или перенести комплекс программ на вычислительные узлы суперкомпьютеров. Пакетный режим предполагает сброс данных в процессе расчета в ПЗУ на каждом конкретном временном шаге, указанном пользователем. Сброс данных происходит согласно формату Mview, где сохраняются распределенные по пространству значения компонент магнитных моментов. Помимо этого, сохраняются значения сенсоров, указанных пользователем. Значения сенсоров включают в себя временную эволюцию магнитного момента отдельного атома. В отдельный файл формата `*.dat` складываются значения усредненных по подрешёткам функциям полей и магнитных моментов атомов.

## 7.3 Выходные данные в интерактивном режиме

Одной из главных проблем, встающих при проведении диагностики, является обработка и визуализация большого объема данных порядка гигабайтов. Такой размер данных соответствует системе в $10^7 \div 10^9$ атомов. Недостаточно проработанная визуализация векторных полей снижает наглядность полученных результатов и может привести к искажению восприятия физической картины. Для данной цели в программный комплекс встроен специальный программный модуль im3D для отображения магнитного момента на единичной сфере: каждый атом отдельной подрешетки окрашивается в цвет согласно цветовой палитре (рис. 5, 6) . Преимущество первого варианта окраски 5 заключается в выделении расположения и направления трех координатных осей. Однако столько обильное количество цветов снижает восприятие результатов моделирования и требует некоторого времени для осмысления структуры магнитной системы. Второй вариант окраски включает в себя трехцветовую гамму (рис. 6). Он более наглядным образом показывает эволюцию магнитного момента в плоском, но не в трехмерном случае.

Результат визуализации магнитной физической системы продемонстрирован на рисунке (4), где приведен простой пример стационарного состояния объемно-центрированной системы, состоящей из двух сортов атомов, при наличии внешнего магнитного поля и термодинамических флуктуации. Помимо наглядной визуализации в представленном модуле реализована возможность диагностики эволюции магнитного поля в трехмерном пространстве (рис.(4), справа).

Преобразование цветовой палитры задается в модуле визуализатора (листинг 11).

Листинг 11: Реализация цветовой палитры в комплексе программ.

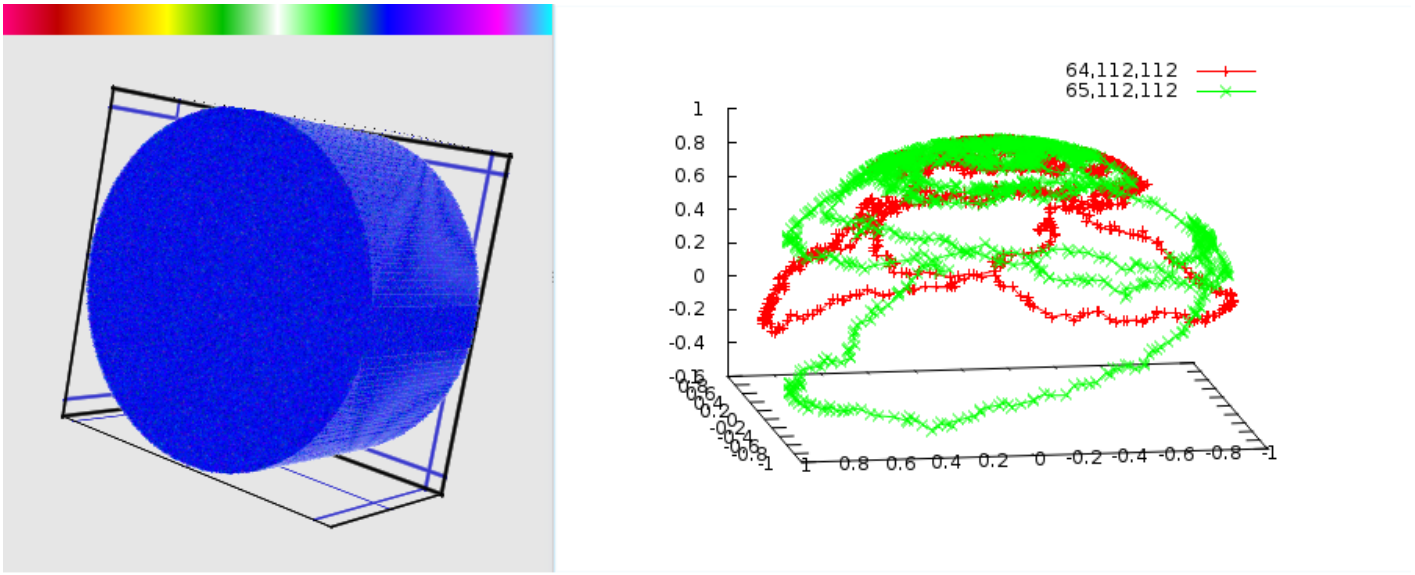


Рис. 4: Слева: визуализация векторного поля, относящегося к одной подрешетке, посредством представленных на рисунках 5 и 6 палитр. Вектор магнитного момента каждого атома отдельной кристаллической подрешетки отображается в собственный растровый пиксель. Справа: диагностика эволюции магнитного момента на единичной сфере двух атомов отдельных подрешеток (красный и зеленый цвета). Два атома относятся к объемно-центрированной кристаллической решетке и принадлежат одной алгоритмической kern-структуре с координатами (64, 112, 112). В представленном примере учитываются влияния внешнего постоянного магнитного поля и термодинамических флуктуации. Как видно из примера, стационарное состояние системы переориентируется вдоль направления постоянного внешнего магнитного поля.

```
__device__ float4 get_color_for3D(float4 f) {
  float A = length( make_float3(f.x,f.y,f.z) );
  if( A == 0 ) return make_float4( 0.0f );
  float3 fs = make_float3( fabs(f.x), fabs(f.y), fabs(f.z)), v=fs;
  fs *= 1.0f/A;
  if( palID%2 == 0 ) {
    if( fs.x<fs.y ) { float t=fs.x; fs.x=fs.y; fs.y=t; }
    if( fs.y<fs.z ) { float t=fs.y; fs.y=fs.z; fs.z=t; }
    if( fs.x<fs.y ) { float t=fs.x; fs.x=fs.y; fs.y=t; }
    float cs=fs.x, sn = 0.8 + 0.2*sqrt( fs.y*fs.y + fs.z*fs.z );
    for(int i=0; i<transparency_mode; i++) v *= cs;
    for(int i=0; i>transparency_mode; i--) v *= sn;
    const float3 r={ 0.f, 1.f, 1.f }, g={ 1.f, 0.f, 1.f }, b={ 1.f, 1.f, 0.f };
    const float3 c={ 1.f, 0.f, 0.f }, m={ 0.f, 1.f, 0.f }, y={ 0.f, 0.f, 1.f };
    float3 f3 = fscale*( v.x*(f.x<0?c:m) + v.y*(f.y<0?y:g) + v.z*(f.z<0?r:b) );
    return make_float4( f3.x, f3.y, f3.z, length(f3) );
  } else {
    float phi = atan2( f.y, f.x ), aphi = fabs(phi), L = f.z/A;
    float C = (1.0f-fabs(L))*fminf(1.0f,fscale*A);
    float X = C*( 1.0f-fabs( fmodf(phi+M_PI,2*M_PI/3) - M_PI/3 )/(M_PI/3) );
    float3 f3 = make_float3( 0 );
    f3.x = aphi < M_PI/3.0 ? 0.0f : ( aphi>2.0*M_PI/3.0?C:X );
```

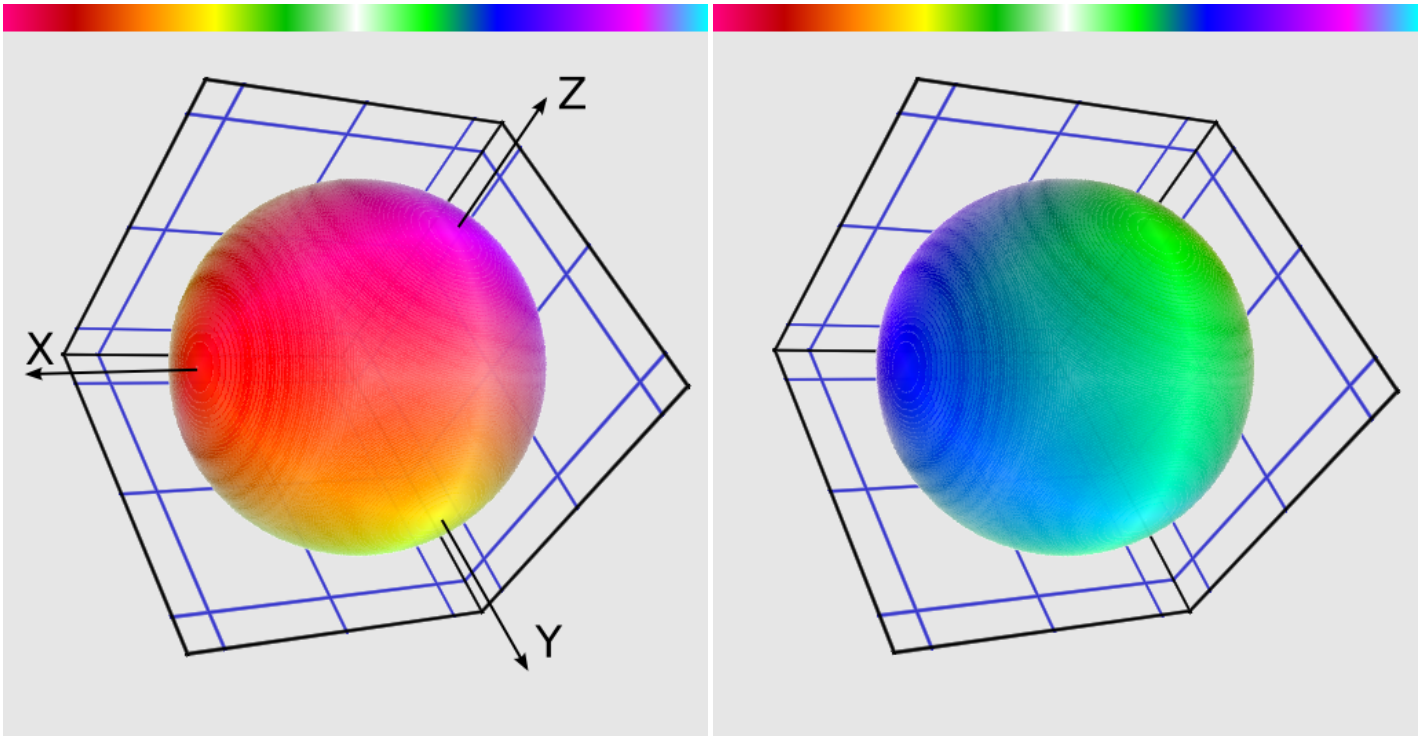


Рис. 5: Первый вариант окраски сферы для визуализации магнитного момента. Сфера поделена на 6 секторов, что соответствует 6 граням натянутого на сферу куба. Окраска граней происходит следующим образом: взаимнопротивоположные грани вдоль оси X — синяя и красная, взаимнопротивоположные грани вдоль оси Y — желтая и салатовая, взаимнопротивоположные грани вдоль оси Z — фиолетовая и зеленая.

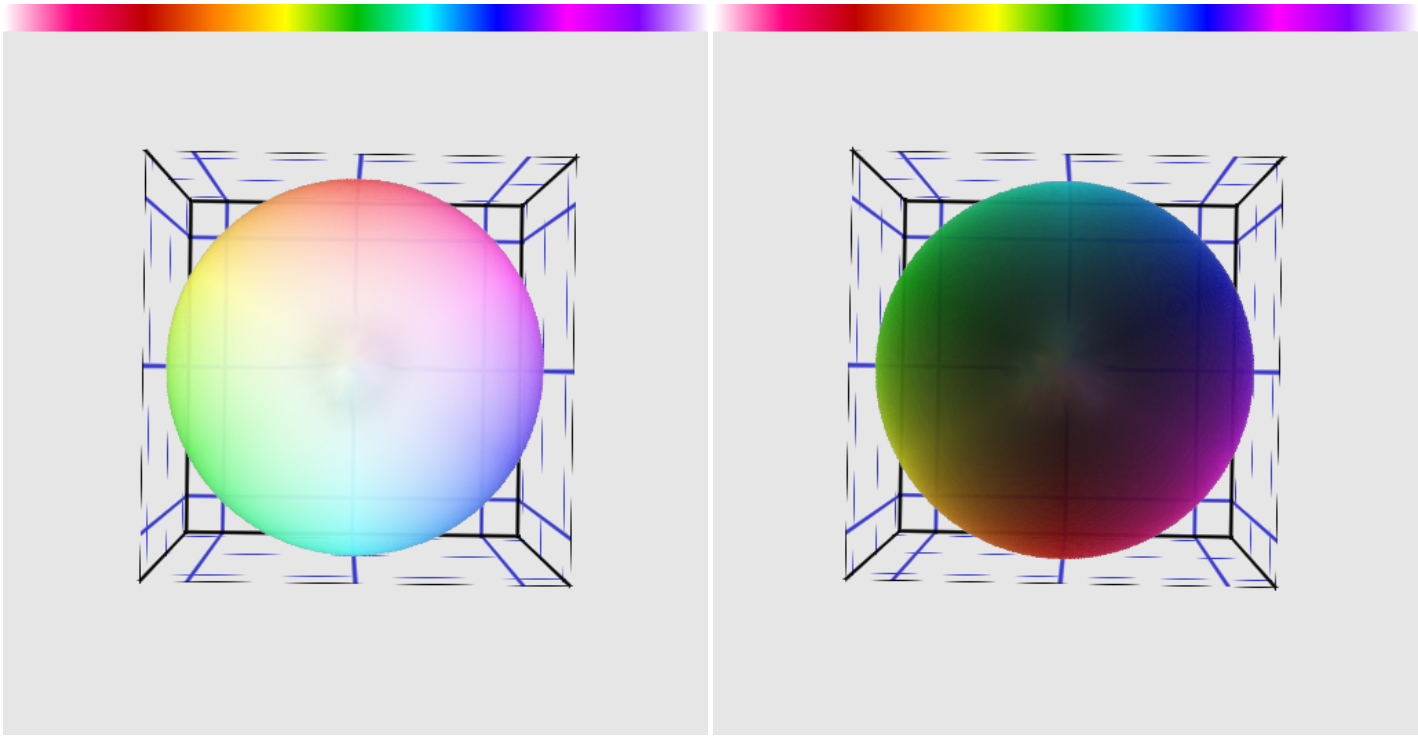

Рис. 6: Второй вариант окраски сферы для визуализации магнитного момента. Сфера поделена на 3 сектора под углом 120 градусов каждый: красный, зеленый, синий. В противоположных полюсах сферы градиент палитры переходит в белый и черный цвета соответственно.

```
        f3.y =  phi > M_PI/3.0 ? 0.0f : ( 0>phi&&phi>-2.0*M_PI/3.0?C:X );
        f3.z =  phi <-M_PI/3.0 ? 0.0f : ( 0<phi&&phi<2.0*M_PI/3.0?C:X );
        float m = 0.5*( L + 1.0 - C );
        return make_float4( m+f3.x, m+f3.y, m+f3.z, length(f3) );
    }
}
```

Выполнение функции `get_color_for3D` осуществляется на графическом устройстве. Входным параметром функции `get_color_for3D` является четырехкомпонентный вектор, содержащий в себе информацию о магнитном моменте отдельного атома. На основании значения переменной `palID` происходит переключение между палитрами (рис. (5), (6)). Выходным параметром функции `get_color_for3D` является элемент в формате стандартной цветовой модели (RGB).
Выходными данными работы подмодуля визуализатора являются растровые файлы формата `*.png` с результатами распределения намагниченности. Пример приведен на рис. 4.

# Список литературы


[1] NVIDIA Corporation, *CUDA Toolkit Documentation v7.5* , 2015

[2] Dave Shreiner *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1* Addison-Wesley Professional, 2009

[3] *Отчет о прикладных научных исследованиях.* Предсказательное моделирование спинтронных наноустройств, основанных на магнитных туннельных переходах по теме: Теоретические исследования поставленных перед ПНИ задач (промежуточный) Этап № 2, УДК 539.21:537.86 538.9–405:537.86.