Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК
Допустить к защите

Зав.каф. _____И.Н.Фионов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Разработка онлайн системы интеллектуального анализа данных расчетов течений разреженного газа на гибридных кластерах: 3D-визуализация

Магистерская диссертация

по направлению 09.04.01 «Информатика и вычислительная техника»

Студент Нужнов А.В. /……/

Руководитель Малков Е.А /……..……/

Новосибирск 2018 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разреженный газ представляет собой среду, состоящую из частиц - молекул воздуха, ионов и электронов плазмы, частиц взвеси, звезд в звездных скоплениях, длина свободного пробега которых сравнима или больше характерных размеров задачи [1-3]. Приведем примеры процессов, когда применима физическая модель разреженных газов:

1. Вхождение космического аппарата в атмосферу Земли. Длина космического корабля равна 10 метрам, а длина свободного пробега на высоте 200 км составляет сотни метров [4].
2. Течение воздуха в бронхах человека. Длина свободного пробега молекул воздуха порядка 0.1 микрона, а размер микроскопических отростков бронх, в которых происходит газообмен, составляет от 0.15 микрон до 100 микрон.
3. Напыление микросхем. Длина свободного пробега молекул газа в установках для напыления микросхем сравнима с характерными деталями микросхемы.
4. Звездные скопления могут быть рассмотрены как разреженный газ. Так для шаровых скоплений длина свободного пробега составляет порядка сотен радиусов звездной системы. А для галактик длина свободного пробега больше в миллион и миллиард раз размеров галактик. Поэтому типичная звезда в галактике делает больше миллиона оборотов не испытывая влияния парных столкновений, что соответствует времени в миллионы раз больше времени существования Вселенной [2,5].

Отношение длины свободного пробега к характерному размеру задачи, так называемое число Кнудсена, для разреженных газов находится в пределах от одной сотой, до бесконечности.

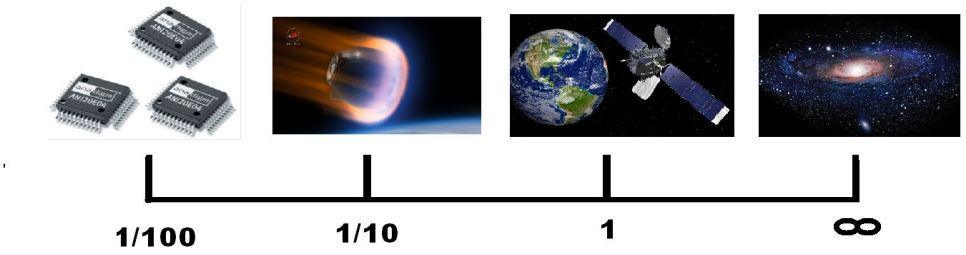


Рисунок 1. Число Кнудсена

Изучение течений разреженного газа основано на микроскопическом описании, использующем функцию распределения $f(t, \vec{r}, \vec{v})$, являющуюся плотностью в шестимерном фазовом пространстве - количество частиц в элементе фазового объема с координатами $\vec{r}$ и $\vec{v}$ в момент времени t, отнесенное к этому объему. Математическая модель разреженного газа представляется нелинейным интегро-дифференциальным уравнением - кинетическим уравнением Больцмана:

$$\frac{\partial f}{\partial t} + \vec{u}\frac{\partial f}{\partial \vec{r}} + \vec{F}(t,\vec{r},\vec{u})\frac{\partial f}{\partial \vec{u}} = \left(\frac{\partial f}{\partial t}\right)_{столк.} \qquad (1.1)$$

В случае полей самосогласованных объемных сил к кинетическому уравнению Больцмана добавляется уравнение поля. Для бесстолкновительной системы кинетическое уравнение и уравнение поля называются уравнениями Власова. Для самогравитрующих систем кинетическое уравнение дополняется уравнением Пуассона [2,5]:

$$\Delta \Phi = \frac{\partial}{\partial f} \frac{\partial \Phi}{\partial \vec{r}} = 4\pi G\rho(t, \vec{r}). \qquad (1.2)$$

Численное решение уравнений (1-2) требует значительных вычислительных ресурсов. Значительное продвижение в моделировании течений разреженного газа было достигнуто в последнее время благодаря появлению и доступности высокопроизводительных вычислительных систем. Особенно большую роль в развитии высокопроизводительных систем сыграло появление графических процессоров общего назначения (GPGPU). В связи с этим, важным направлением в исследовании течений разреженного газа является разработка параллельных алгоритмов для выполнения на графических процессорах [6-11].

При численном моделировании разреженного газа приходится иметь дело с большим объемом данных представляющих процессы на микроскопическом уровне требуется задание функции распределения в 6-мерном фазовом пространстве. При этом эти процессы развиваются во времени, что требуют на этапе постпроцессинга анализировать большое количество снапшотов, размер каждого из которых составляет от сотен мегабайт до нескольких гигабайт. Поэтому важное значение для контроля численного эксперимента может иметь предварительный анализ результатов расчета, основанный на визуализации данных в режиме онлайн.

Цель магистерской работы заключалось в численном моделировании динамической эволюции звездных систем и визуализации результатов моделирования в режиме онлайн. Для этого были решены следующие задачи:

1. Разработаны распараллеленные алгоритмы метода частиц-в-ячейках для выполнения на графических ускорителях, а именно:

1.1. Распараллелен метод свертки для нахождения потенциала гравитирующей системы на основе использования прикладной библиотеки CUDA cuFFT.

1.2. Распараллелено распределение масс частиц в узлы расчетной сетки.

1.3. Реализована интерполяция значений силового поля из узлов расчетной сетки в местоположения частиц на основе аппаратной линейной фильтрации с использование текстурной памяти (CUDA).

2. Разработан алгоритм инициализации начальной функции распределения посредством генерации выборок псевдо-случайных чисел на основе метода обращения и метода исключения Джона фон-Неймана.

3. Спроектировано и реализовано приложение для онлайн визуализации результатов моделирования течений разреженного газа на основе библиотеки OpenGL и программного интерфейса CUDA.

4. Проведены расчеты эволюции бесстолкновительных гравитирующих систем в области критических параметров устойчивости по отношению к возмущениям типа ядро-гало.

1. Математическая постановка задачи

Запишем соотношения связывающие декартовы и сферические координаты

$$\begin{cases} x = r\sin\Psi\cos\varphi, \\ y = r\sin\Psi\sin\varphi, \\ z = r\cos\Psi. \end{cases} \tag{2.1}$$

Запишем также соотношения для скоростей в декартовой и сферической системе координат

$$\begin{cases} \vartheta_x = \vartheta_r\sin\Psi\cos\varphi + \vartheta_\psi\cos\Psi\cos\varphi - \vartheta_\varphi\sin\varphi \\ \vartheta_y = \vartheta_r\sin\Psi\sin\varphi + \vartheta_\psi\cos\Psi\sin\varphi - \vartheta_\varphi\cos\varphi \\ \vartheta_z = \vartheta_r\cos\Psi - \vartheta_\psi\sin\varphi, \end{cases} \tag{2.2}$$

где

$$\begin{cases} \vartheta_r = \dfrac{dr}{dt}, \\ \vartheta_\varphi = r\sin\Psi\cdot\dfrac{d\varphi}{dt}, \\ \vartheta_\psi = r\cdot\dfrac{d\Psi}{dt}. \end{cases} \tag{2.3}$$

Перейдем в плоскости векторов ϑ_φ , ϑ_ψ к полярным координатам:

$$\begin{cases} \vartheta_\varphi = \vartheta_\perp\cos\beta, \\ \vartheta_\psi = \vartheta_\perp\sin\beta. \end{cases} \tag{2.4}$$

Функция распределения нестационарного шара Камма в сферических координатах имеет следующий вид:

$$f = \frac{\rho_0}{\pi^2 \alpha^2 r_0^2}\left\{\alpha^2\left(r_0^2 - \frac{r^2}{a^2}\right) - \right.$$

$$\left. - a^2\left[\left(\vartheta_r - \frac{1}{a}\frac{da}{dt}r\right)^2 + \vartheta_\perp^2\left(1 - \frac{r^2}{a^2 r_0^2}\right)\right]\right\}^{\frac{1}{2}} \qquad (2.5)$$

где зависимость от времени выражается соотношением

$$\begin{cases} a = 1 - e\cos\xi, \\ t = \varphi - e\sin\xi, \qquad \dfrac{da}{dt} = e\sin\xi, \end{cases} \qquad (2.6)$$

где e равно константе из диапазона от 0 до 1. $\alpha^2 = 1 - e^2$.

Модель шара Камма представляет гравитирующую систему с радиусом R равным a r_0 , плотность которой меняется как $\rho(t) = \frac{\rho_0}{a^3}$, где ρ_0 константа. Масса системы M постоянна и равна $\frac{4}{3}\pi\rho\, r_0^3$. В случае самосогласованного гравитационного поля функция распределения должна удовлетворять уравнению Пуассона (1.2), отсюда получаем выражение для потенциала системы

$$\Phi(t,r) = \begin{cases} \dfrac{2}{3}\pi G\rho_0 \dfrac{r^3}{a^3} - 2\pi G\rho_0 r_0^2 \dfrac{1}{a}, & r < R \equiv a r_0, \\ -\dfrac{4}{3}\pi G\rho_0 \dfrac{r_0^3}{r}, & r > R \equiv a r_0 . \end{cases} \qquad (2.8)$$

В дальнейшем полагаем $\frac{4}{3}\pi G\rho_0 = 1$

Отношение гравитационной энергии к удвоенной кинетической энергии, называемое вириальным соотношением, для этой системы равно

$$\frac{|W|}{E_{min}} = \frac{a^2}{a} = \frac{1-e^2}{(1+e\cos\xi)^2}, \qquad (2.8)$$

и меняется во время пульсации системы.

При e = 0 это отношение равно 1, в этом случае система является стационарной. Вириальное соотношение в момент времени, соответствующем максимальному расширению системы можно принять за параметр, характеризующей степень нестационарности системы и задающем амплитуду пульсации A = (1 + e)/(1 - e).

Точные нестационарные модели были построены в работах [ссылка], где также была исследована устойчивость этих моделей по отношению к возмущениям “шар-эллипсоид” и “ядро-гало” в линейном приближении. Были найдены критические значения амплитуды пульсации при которых наступает неустойчивость вышеназванных возмущений.

Развитие этих неустойчивостей на нелинейной стадии можно исследовать только численно. В следующем разделе описывается программная реализация метода частиц-в-ячейках [12-14], позволяющая проводить эти исследования.

Функция распределения нестационарного шара Камма [15-17] является достаточно искусственной, чтобы детально представлять реальные звездные системы. Однако развитие возмущений “шар-эллипсоид” и “ядро-гало” зависит только от вторых и третьих моментов функции распределения. Поэтому следует ожидать, что критические значения амплитуды пульсации реальных систем будут близки к значениям теоретической модели нестационарного шара Камма.

Для сравнения, мы рассчитываем эволюцию звездной системы с “естественной” функцией распределения начальный момент времени равный

$$f(0,\vec{r},\vec{\vartheta}) = \frac{\rho_0}{(2\pi T)^{\frac{3}{2}}} e^{-\frac{\vartheta_x^2+\vartheta_y^2+\vartheta_z^2}{2T}}, \tag{2.9}$$

где

$$T = T_0\sqrt{1-\frac{r}{R_0}}\ . \tag{2.10}$$

2. Программная реализация метода частиц-в-ячейках

При численном моделировании разреженного газа приходится иметь дело с большим объемом данных, поэтому была проведена параллелизация основных алгоритмов в рамках модели SIMT(Single Instruction Multiple Threads) с использованием интерфейса CUDA [18-21].

2.1 Инициализация начальных данных

Вначале необходимо задать начальное положение и скорости частиц, представляющих дискретную аппроксимацию функции распределения в 6-мерном фазовом пространстве.

Согласно идеологии метода частиц-в-ячейках положение частиц в фазовом пространстве выбираются случайным образом, в соответствии с функцией распределения, которая может интерпретироваться как распределение вероятностей.

При компьютерной реализации выборки случайных величин используются библиотечные функции - генераторы псевдослучайных чисел.

Для генерации последовательностей значений случайной величины с соответствующей функцией распределения сравнивались интерфейсы <*random*> стандартной библиотеки шаблонов C++11, генератор *rand()* стандартной библиотеки Си *stdlib.h* и интерфейс curand.h библиотеки *CUDA*.

По результатам тестирования был отобран интерфейс <random> и генератор *rand()*.

Процедура генерации положений и скоростей частиц разбивалась на два этапа.

На первом этапе генерировались положения частиц, при этом в качестве базового использовался датчик случайных чисел, соответствующих однородному распределению на отрезке [0,1]. Розыгрыши сферической координаты *r* и косинуса полярного угла производились модифицированными

генераторами случайных чисел, полученных из однородного генератора на основе метода обращения[22]

Ниже представлен фрагмент кода реализующего инициализацию положений частиц для обеих функций распределений:

Листинг 1.

```
#include <stdlib.h>
#define REAL float
struct position{
  REAL x,y,z;
};
inline REAL Rand(REAL range){
  return (((REAL)rand())/RAND_MAX)*range;
}
…….
   srand( (unsigned)time( NULL ) );
   for(int counter=0;counter<NP;counter++){
    REAL r=Rand(R0);
    REAL cpsi=-1.0+Rand(2.0);
    REAL phi=Rand(2.0*M_PI);

    r=pow(r, 1.0/3.0);

    P[counter].x=r*sqrt(1.0-cpsi*cpsi)*cos(phi);
    P[counter].y=r*sqrt(1.0-cpsi*cpsi)*sin(phi);
    P[counter].z=r*cpsi;
……...
```

Таким образом однородно заполняется сферический объем радиуса R_0 .

Для генерации скоростей в случае функции распределения (2.8) использовался генератор нормального распределения *std::normal_distribution* интерфейса <*random*> [23]:

Листинг 2.

```
struct velocity{
  REAL vx,vy,vz;
};
std::random_device   dev_rand;
```

```
std::default_random_engine gen(dev_rand());

for(int i=0;i<NP;i++){
   REAL r=sqrt(P[i].x*P[i].x + P[i].y*P[i].y + P[i].z*P[i].z);
   REAL T=T0*sqrt(1.0-r/R0);

   std::normal_distribution<REAL> VX(0.0, sqrt(T));
   std::normal_distribution<REAL> VY(0.0, sqrt(T));
   std::normal_distribution<REAL> VZ(0.0, sqrt(T));

   V[i].vx=VX(gen);
   V[i].vy=VY(gen);
   V[i].vz=VZ(gen);
 }
```

Для функции распределения шара Камма не существует библиотечного генератора псевдо-случайных чисел. Поэтому для инициализации скоростей частиц использовался метод исключения фон Неймана[22]:

Листинг 3.

```
 for(int counter=0;counter<NP;counter++){
   REAL vtmax=sqrt( (1.0-e)/(1.0+e) );
   REAL vrmax=vtmax;
    ……..
   REAL vt, vr,Z;
   do{
     vt=Rand(vtmax);
     vr=-vrmax+Rand(2.0*vrmax);
     Z=Rand(100);
     D=a*a*( (1.0-r*r/a/a)*((1.0-e)/(1.0+e)-vt*vt)-vr*vr);
   }while(D<0 || (Z > r*r*vt/sqrt(D)))

   REAL beta=Rand(2.0*M_PI);
   V[counter].vx=vr*sqrt(1.0-cpsi*cpsi)*cos(phi)-
         vt*cos(beta)*cpsi*cos(phi)-
         vt*sin(beta)*sin(phi);
   V[counter].vy=vr*sqrt(1.0-cpsi*cpsi)*sin(phi)-
         vt*cos(beta)*cpsi*sin(phi)+
```

```
            vt*sin(beta)*cos(phi);
    V[counter].vz=vr*cpsi+
            vt*cos(beta)*sqrt(1.0-cpsi*cpsi);


……...
```

Для того, чтобы система в целом не меняло своего положения в пространстве, после инициализации скоростей частиц производится их корректировка с тем, чтобы средняя скорость равнялась нулю:

Листинг 4.

```
 for(int counter=0;counter<NP;counter++){
……
   Vx+=V[counter].vx;
   Vy+=V[counter].vy;
   Vz+=V[counter].vz;
  }

  Vx/=num_of_verticies;
  Vy/=num_of_verticies;
  Vz/=num_of_verticies;

  for(int counter=0;counter<NP;counter++){
   V[counter].vx-=Vx;
   V[counter].vy-=Vy;
   V[counter].vz-=Vz;
  }
```

2.2 Распределение массы частиц в узлы расчетной сетки

Потенциал самосогласованного гравитационного поля задается в узлах однородной расчетной сетки. Для его вычисления необходимо распределить массы частиц в эти узлы. При этом могут быть использованы различные методы интерполяции. В работе был выбран метод ближайшего соседа, проиллюстрированный на рисунке 1 и в листинге 5 .

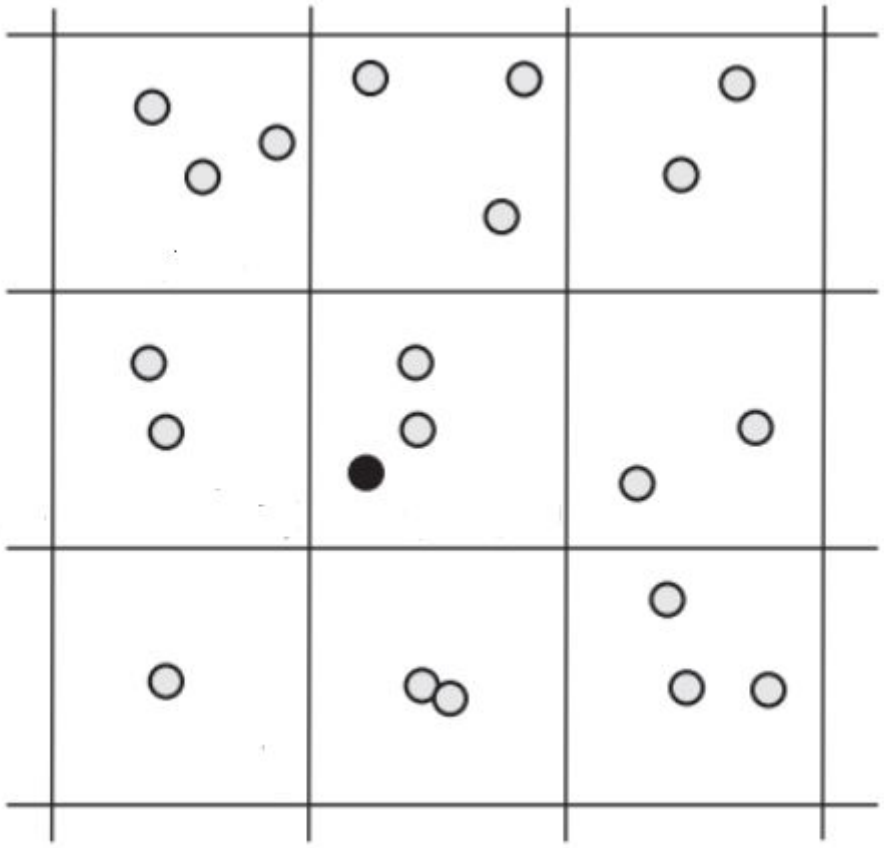


Рисунок 2. Метод ближайшего соседа

Листинг 5.

```
__global__ void gRho(position* P, int* rho){
  int i=threadIdx.x+blockIdx.x*blockDim.x;

  int nx=(int)((P[i].x-xmin)/h);
  int ny=(int)((P[i].y-xmin)/h);
  int nz=(int)((P[i].z-xmin)/h);

  if( (nx>=0 && nx<NDIM)&&(ny>=0 && ny<NDIM)&&(nz>=0 &&
nz<NDIM) )
  atomicAdd(&rho[nz+ny*NDIM+nx*NDIM*NDIM],1);
 }
```

Для каждой частицы вычисляется положение ближайшего узла расчетной сетки с координатами nx, ny, nz и значение элемента целочисленного массива соответствующего этому узлу увеличивается на единицу. Поскольку эта операция производится на каждом шаге вычислений, алгоритм распараллелен для выполнения на графическом процессоре с использованием интерфейса прикладного программирования CUDA. Чтобы разрешить конфликт чтение/запись при выполнении различных нитей вместо операции инкремента используется атомарная функция atomicAdd.

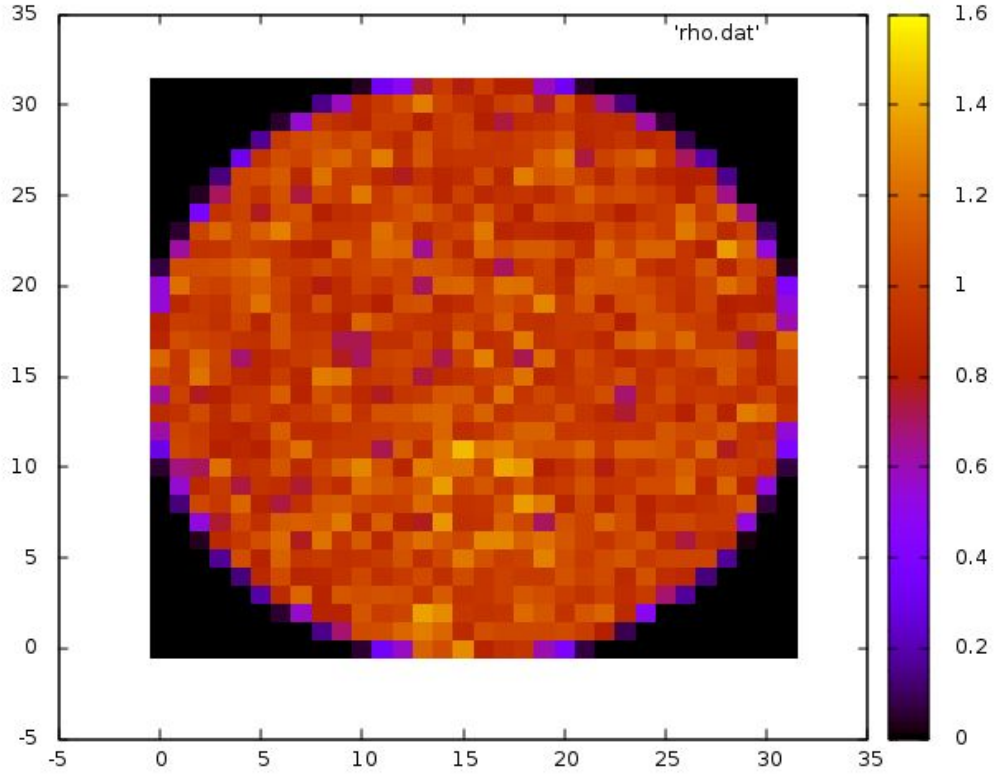


Рисунок 3.

Далее значения в узлах нормируются, чтобы привестись к пространственной плотности модели. На рисунке 3 изображено центральное сечение гравитирующей системы с однородной плотностью равной единице в начальный момент времени.

2.3 Расчет силового поля

Сила действующая на частицу в гравитационном поле равно градиенту потенциала $\Phi(t,\vec{r})$ с обратным знаком.

$$F(t,\vec{r}) = \frac{\partial \Phi}{\partial \vec{r}} \ . \tag{3.1}$$

Функция $\Phi(t,\vec{r})$ подчиняется уравнению Пуассона:

$$\Delta \Phi - \frac{\partial}{\partial \vec{r}} \frac{\partial \Phi}{\partial \vec{r}} = 4\pi G\rho(t,\vec{r}) \ . \tag{3.2}$$

Для изолированных гравитирующих систем, когда выполняется граничные условия

$$\Phi(\vec{r}) \rightarrow - \frac{1}{r} \text{ , при } r \rightarrow \infty$$

Решение Пуассона можно записать в виде

$$\Phi(\vec{r}) = G \int \frac{\rho(\vec{r}\,')d^3(\vec{r}\,')}{|\vec{r} - \vec{r}\,'|} \ . \tag{3.3}$$

Полагаем в дальнейшем гравитационную постоянную G=1.

Дискретная аппроксимация этого выражения имеет вид свертки.

$$\Phi_{i,j,k} = \sum_{i=1}^{L} \sum_{j=1}^{M} \sum_{k=1}^{N} \frac{\rho_{i',j',k'}}{\sqrt{(x_i - x_i')^2 + (y_j - y_j')^2 + (z_k - z_k')^2}} \ , \tag{3.4}$$

где $\Phi_{i,j,k}$ - значение потенциала в узлах расчетной сетке с размерностью (L,M,N), $\rho_{i',j',k'}$ - значение плотности в узлах расчетной сетки.

Для вычисления потенциала $\Phi_{i,j,k}$ применяется теорема о свертке [24], тогда вычисление последовательности (3.4) сводится к вычислению последовательности,

$$\widehat{\Phi}_{\alpha,\beta,\gamma} = \sqrt{LMN}\,\widehat{\rho}_{\alpha,\beta,\gamma}\widehat{g}_{\alpha,\beta,\gamma}\,, \tag{3.5}$$

получаемой произведением фурье-образа $\widehat{\rho}_{\alpha,\beta,\gamma}$ плотности и фурье-образа функции Грина -

$$g_{i,j,k} = \frac{1}{\sqrt{x_i^2 + y_j^2 + z_k^2}}\,, \tag{3.6}$$

, $\widehat{g}_{\alpha,\beta,\gamma}$ с последующим обращением фурье-образа $\widehat{\Phi}_{\alpha,\beta,\gamma}$.

Описанная операция эффективно выполняется с помощью быстрого преобразования Фурье (FFT) [25]. В программной реализации быстрого преобразования Фурье использовалась прикладная библиотека *CUDA cuFFT* [26]. Функции интерфейса *cuFFT* реализуют параллельные алгоритмы, выполняемые на графических процессорах. Фурье-преобразование функции Грина выполняется один раз и результат сохраняется для последующего умножения на фурье-образ плотности на каждом шаге вычисления эволюции гравитирующей системы. В приведенном ниже фрагменте кода *Nx,Ny,Nz* - размерности расчетной сетки в физическом пространстве. Для повышения точности вычисления фурье-преобразования, функция Грина продолжается таким образом, что значения на границе области определения совпадают. Рисунок 1 иллюстрирует это продолжение для одномерного случая. Расчетная сетка для вычисления фурье-образа при этом увеличивается в восемь раз.

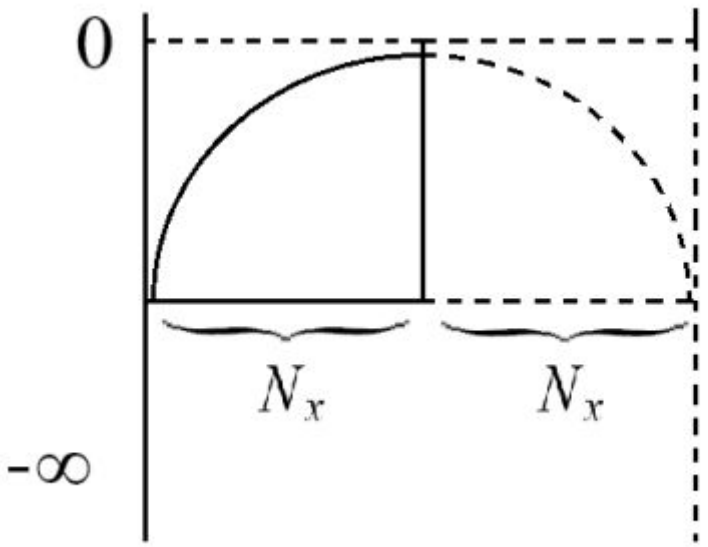


Рисунок 4. Функция Грина

Листинг 5.

```
cufftComplex *g_exp_dev_t;
cufftHandle planC2C;

void hGreen(int Nx, int Ny, int Nz){
  float* g_exp_dev;
  gGreenExpand<<<blocks2, threads>>>(g_exp_dev_t, g_exp_dev, Nx,Ny,Nz);
  cudaDeviceSynchronize();

  cufftPlan3d(&planC2C, 2*Nx, 2*Ny, 2*Nz, CUFFT_C2C);
  cufftExecC2C(planC2C, g_exp_dev_t, g_exp_dev_t, CUFFT_FORWARD);

  cudaFree(g_exp_dev);
}
```

Для выполнения фурье-преобразования вызывается функция *cufftExecC2C*, первым параметром которой является дескриптор структуры *cufftHandle*, получаемое после вызова функции *cufftPlan3d*. Этот же дескриптор используется для фурье-преобразования функции плотности. Основные шаги вычисления

потенциала на основе теоремы о свертки представленны ниже следующем фрагменте кода (листинг 6).

Листинг 6.

```
void Phi(float* rho_dev, float* phi_dev){
      ……...
        cufftExecC2C(planC2C,rho_dev_exp_t, rho_dev_exp_t,
CUFFT_FORWARD);
        gMult<<<blocks2,threads>>>(rho_dev_exp_t, g_exp_dev_t);
        cudaDeviceSynchronize();

        cufftExecC2C(planC2C,rho_dev_exp_t, rho_dev_exp_t, CUFFT_INVERSE);

        gGetPhi<<<blocks,threads>>>(phi_dev, rho_dev_exp_t);
        cudaDeviceSynchronize();
      }
```

На рисунке 5 представлено центральное сечение потенциала однородной сферы с радиусом равным единице, рассчитанного по вышеописанной схеме. Размерность расчетной сетки - $64 \times 64 \times 64$. Для нормирования потенциала полагалось $\frac{4}{3}\ \pi G\rho\ =\ 1$.

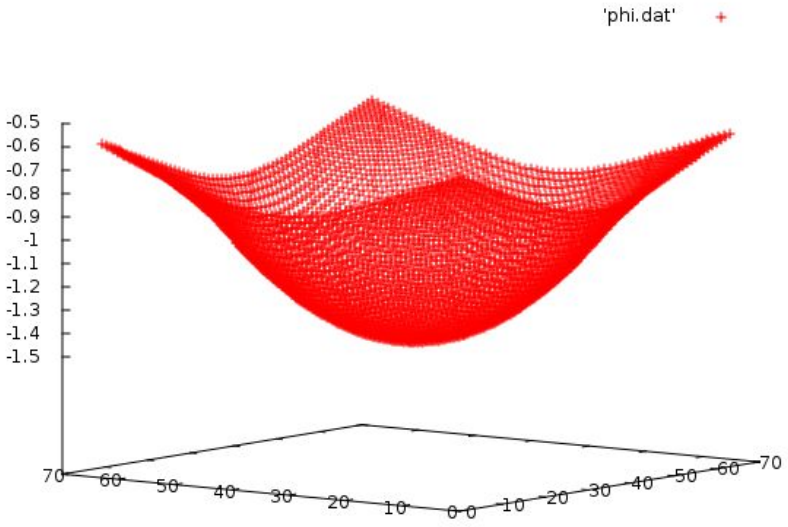


Рисунок 5.

2.4 Схема решения уравнения движения

После вычисления потенциала вычисляем силу в узлах расчетной сетки. В качестве разностной аппроксимации градиента выбираем центральную разность, обеспечивающую второй порядок аппроксимации [27]. Так, что

$$F_{x\,i,j,k} = -\frac{\Phi_{i+1,j,k} - \Phi_{i-1,j,k}}{2\Delta x},$$
$$F_{y\,i,j,k} = -\frac{\Phi_{i,j+1,k} - \Phi_{i,j-1,k}}{2\Delta y}, \qquad (3.7)$$
$$F_{z\,i,j,k} = -\frac{\Phi_{i,j,k+1} - \Phi_{i,j,k-1}}{2\Delta z}.$$

В листинге 7 представлена параллельная реализация алгоритма вычисления силы, когда каждая нить выполняет расчет для отдельного узла с индексами *i,j,k*.

Листинг 7.

```
__global__ void gForce(REAL* Fx, REAL* Fy, REAL* Fz, REAL* phi, int  N){
      int i=threadIdx.x+blockIdx.x*blockDim.x;
      int j=threadIdx.y+blockIdx.y*blockDim.y;
      int k=threadIdx.z+blockIdx.z*blockDim.z;

      if(i>0 && i<N-1)
      Fx[k+j*N+i*N*N]=-(phi[k+j*N+(i+1)*N*N]-phi[k+j*N+(i-1)*N*N])/2.0/h;
      if(j>0 && j<N-1)
      Fy[k+j*N+i*N*N]=-(phi[k+(j+1)*N+i*N*N]-phi[k+(j-1)*N+i*N*N])/2.0/h;
      if(k>0 && k<N-1)
      Fz[k+j*N+i*N*N]=-(phi[(k+1)+j*N+i*N*N]-phi[(k-1)+j*N+i*N*N])/2.0/h;
    }
```

Для интегрирования уравнений движения используем схему “с перешагиванием”:

$$\begin{cases} \dfrac{u^{(n+1)}-u^{(n)}}{\tau} = \dfrac{1}{m}F\left(r^{(n)}\right), \\ \dfrac{r^{(n+1)}-r^{(n)}}{\tau} = \dfrac{u^{(n+1)}+u^{(n)}}{\tau}, \end{cases} \tag{3.8}$$

которая позволяет вычислить значения скоростей ν и положений r частиц на $n+1$ временном слое, зная их значения на n-ом временном слое, здесь τ - временной шаг. Проблема состоит в том, чтобы интерполировать силу F в точках расчетной сетки, соответствующих положениям частиц и, в общем случае, не совпадающими с узлами расчетной сетки. В программе используется линейная интерполяция, проиллюстрированная на рисунке 6. На рисунке значение силы в точке обозначенной символом “ * ” , в месте расположения частицы, находятся по формуле

$$F_* = d_1 F_1 + d_2 F_2 + d_3 F_3 + d_4 F_4 \,, \tag{3.9}$$

где d_i отношениям площади пересечения прямоугольника с центром в точке “ $*$ " с ячейками ближайших узлов к площади ячеек.

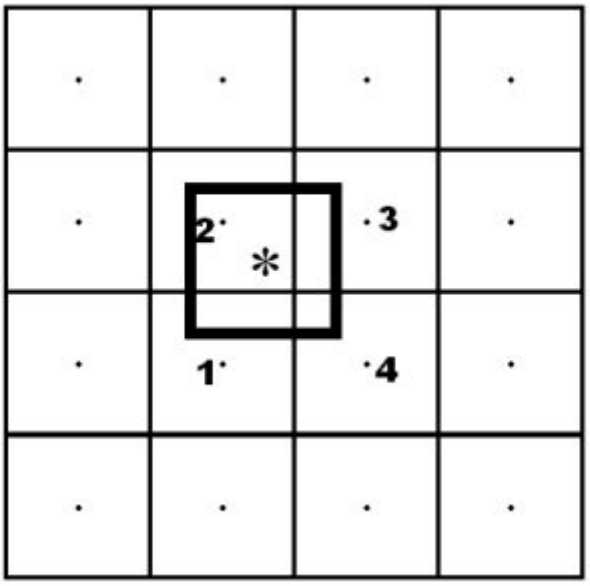


Рисунок 6.

Архитектура *CUDA* предоставляет интерфейс текстурной памяти, реализующий аппаратную линейную интерполяцию, что существенно ускоряет эту процедуру.

В листинге 8 приводится код ядра, параллельно исполняемый нитями графического процессора, реализующий схему решения уравнения движения с использованием текстурной памяти с размерностью *NDIM* × *NDIM* × *NDIM.*

Листинг 8.

```
texture<REAL, 3, cudaReadModeElementType> Fx_tex;
texture<REAL, 3, cudaReadModeElementType> Fy_tex;
texture<REAL, 3, cudaReadModeElementType> Fz_tex;

__global__ void gMove(position* P, velocity* V){
  int n=threadIdx.x+blockIdx.x*blockDim.x;
```

```
    REAL X=(P[n].x-XMIN)*NDIM/2+0.5;
    REAL Y=(P[n].y-YMIN)*NDIM/2+0.5;
    REAL Z=(P[n].z-ZMIN)*NDIM/2+0.5;

    V[n].vx=V[n].vx + tau*tex3D(Fx_tex,Z, Y, X);
    V[n].vy=V[n].vy + tau*tex3D(Fy_tex,Z, Y, X);
    V[n].vz=V[n].vz + tau*tex3D(Fz_tex,Z, Y, X);

    P[n].x=P[n].x+tau*V[n].vx;
    P[n].y=P[n].y+tau*V[n].vy;
    P[n].z=P[n].z+tau*V[n].vz;
  }
  ……...
   gMove<<<num_of_verticies/256,256>>>(P,V);
   cudaDeviceSynchronize();
```

После изменений положений частиц повторяется цикл, включающий распределение массы частиц в узлы расчетной сетки, вычисление потенциала и силы в узлах расчетной сетки.

В заключение описанного алгоритма решения уравнения Власова-Пуассона методом частиц-в-ячейках приведем список основных этапов этой процедуры:

1. Генерация начального распределения.
2. БПФ функции Грина.
3. Распределение массы частиц в узлы сетки.
4. БПФ пространственной плотности.
5. Умножение фурье-образов функции Грина и пространственной плотности.
6. Обратное БПФ - восстановление потенциала.
7. Вычисление силового поля.
8. Решение уравнения движения.

3. Архитектура приложения OpenGL для онлайн визуализации расчетов течений разреженных газов

Архитектура приложения для онлайн визуализации основана на взаимодействии модулей программы, написанных с использованием интерфейсов программирования OpenGL [28-30] и CUDA. Эти модули имеют общие (разделяемые) области памяти, что обеспечивает высокую производительность вычислений, поскольку отсутствует необходимость копирования данных между различными областями видеопамяти и копирования данных между видеопамятью и памятью центрального процессора. Взаимодействие обеспечивается интерфейсом <cuda_gl_interop.h>, заметим, что при этом сочетаются два стиля программирования - императивное и программирование состояний. Ниже приводятся фрагменты кода, поясняющие механизм этого взаимодействия (листинги 9-10). Вначале создается таблица, связывающая идентификаторы буферов, переменных-объектов OpenGL, с указателями на соответствующие области памяти (вызов glGenBuffers). Затем под буферы выделяется память (вызовы пары glBindBuffer, glBufferData).

Листинг 9.

```
const int num_of_verticies=POINT_PER_CELL*NDIM*NDIM*NDIM;
enum bufferNames{POSITIONS, VELOCITIES, GRID,NUM_OF_BUFFERS};

void genBuffers(){
  glGenBuffers(NUM_OF_BUFFERS, bufferID);

  glBindBuffer( GL_ARRAY_BUFFER, bufferID[POSITIONS]);
  glBufferData( GL_ARRAY_BUFFER, num_of_verticies*sizeof(position),
                                0, GL_DYNAMIC_DRAW );

  glBindBuffer( GL_ARRAY_BUFFER, bufferID[VELOCITIES]);
  glBufferData( GL_ARRAY_BUFFER, num_of_verticies*sizeof(velocity),
                                0, GL_DYNAMIC_DRAW );
```

```
  glBindBuffer(GL_ARRAY_BUFFER, bufferID[GRID]);
  glBufferData(GL_ARRAY_BUFFER,
               NDIM*NDIM*NDIM*sizeof(float),
               0, GL_DYNAMIC_DRAW);
}
```

После этого идентификаторы буферов отображаются на указатели (вызовы cudaGLRegisterBufferObject и cudaGLMapBufferObject). Далее функции-ядра CUDA могут использовать указатели в качестве параметров и выполнять любые действия над данными в областях видеопамяти, выделенных под буферы. По окончании вызовов CUDA API буферы “отвязываются” от указателей (вызовы cudaGLUnmapBufferObject).

Листинг 10.

```
  cudaGLRegisterBufferObject(bufferID[POSITIONS]);
  cudaGLRegisterBufferObject(bufferID[VELOCITIES]);
  cudaGLRegisterBufferObject(bufferID[GRID]);

  position* P;
  velocity *V;
  REAL* rho;

  cudaGLMapBufferObject((void**)&P, bufferID[POSITIONS]);
  cudaGLMapBufferObject((void**)&V, bufferID[VELOCITIES]);
  cudaGLMapBufferObject((void**)&rho, bufferID[GRID]);
……….……
                код, выполняемый с использованием CUDA API
……….……
  cudaGLUnmapBufferObject(bufferID[POSITIONS]);
  cudaGLUnmapBufferObject(bufferID[VELOCITIES]);
  cudaGLUnmapBufferObject(bufferID[GRID]);
```

На следующем этапе, подготовленные с помощью CUDA данные визуализируются (листинг 11). Данные в буфере с идентификатором bufferID[POSITIONS] связываются с переменной вершинного шейдера “pos”, после чего точки отрисовываются на экране. Количество частиц в методе частиц-в-ячейках должно быть достаточно большим. Чем больше частиц, тем меньше уровень статистического шума. При визуализации нет необходимости отображать все частицы, число которых при расчетах варьировалось от миллиона до десятков миллионов. Число отображаемых частиц контролируется переменной OFFSET.

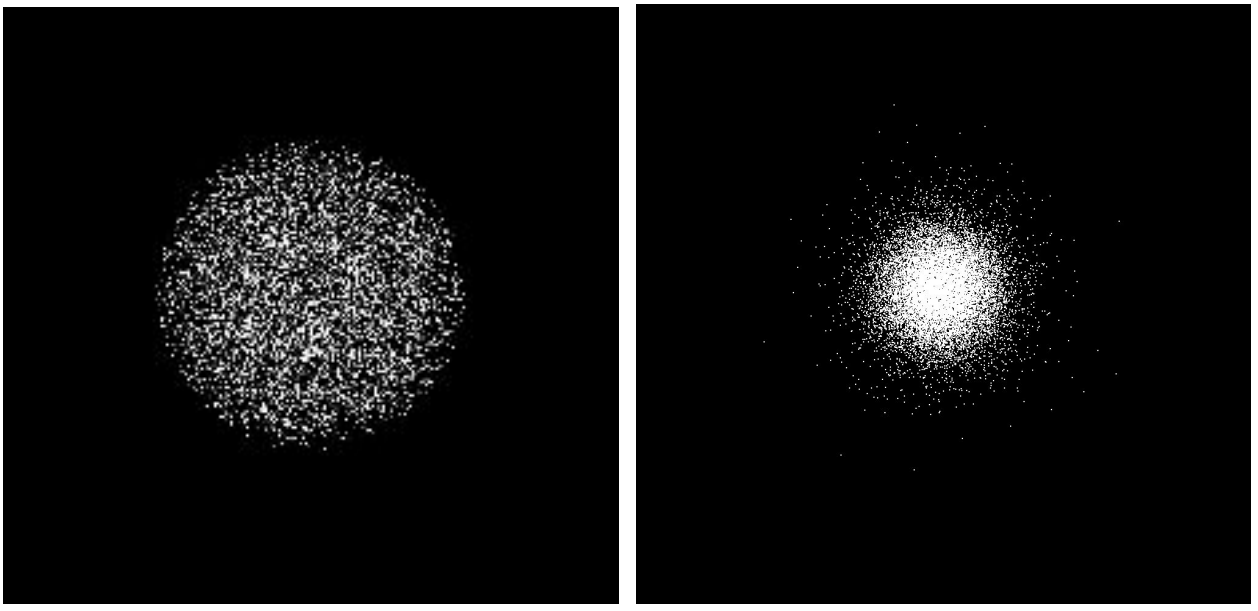
Листинг 11.

```
GLuint genRenderProg() {
   GLuint progHandle = glCreateProgram();
   GLuint vp = glCreateShader(GL_VERTEX_SHADER);
……..
  const char *vpSrc[] = {
        "#version 430\n",
        "layout(location = 0) in vec3 pos;\
         out vec4 vs_color;\
         void main() {\
           gl_Position = vec4(pos,1.0);\
           vs_color=vec4(1.0,1.0,1.0,1.0);\
         }"
        };
…….
}

GLuint genRenderProg();
GLuint progHandle;
void display(){
 progHandle=genRenderProg();
 glUseProgram(progHandle);
 glBindBuffer( GL_ARRAY_BUFFER, bufferID[POSITIONS]);
 GLint posPtr = glGetAttribLocation(progHandle, "pos");
 glVertexAttribPointer(posPtr, 3, GL_FLOAT, GL_FALSE, 12*OFFSET, 0);
 glEnableVertexAttribArray(posPtr);
```

```
glDrawArrays(GL_POINTS, 0, num_of_verticies);

glDisableVertexAttribArray(posPtr);   }
```

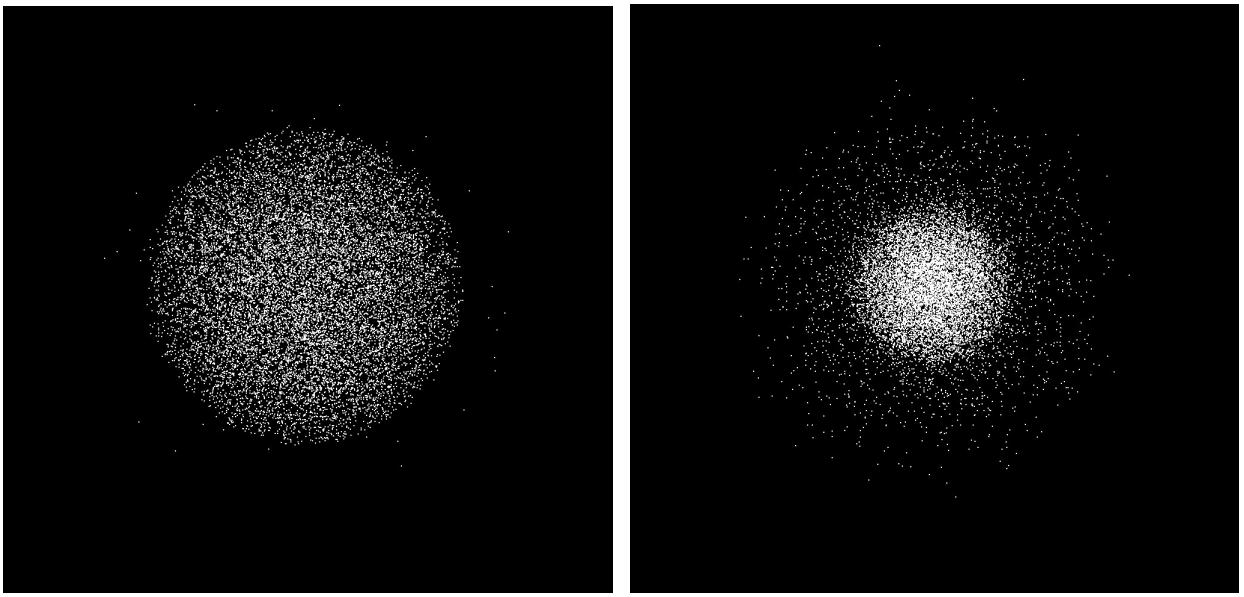
4. Тестовые расчеты

Теоретические исследования устойчивости нестационарной модели шара Камма (2.5) показали, что ведущие в механизме бесстолкновительной релаксации возмущения типа “шар-эллипсоид” и “ядро-гало” становятся неустойчивыми при амплитуде пульсации $A_{S-E} = 24.0$ и $A_{K-H} = 8.1$ соответственно. Критические значения амплитуды пульсации соответствуют вириальным отношениям в фазе максимального расширения $\frac{|W|}{2E}\Big|_{S-E} = 12.5$ $\frac{|W|}{2E}\Big|_{K-H} = 4.5$. С целью верификации разработанного программного кода были проведены тестовые расчеты эволюции звездной системы с начальными условиями, соответствующими “естественной” модели (2.9), и вириальным отношениям равным 5.0 - закритическим значением для возмущений типа “ядро-гало”. На рисунках 7-11 изображено распределение плотности на



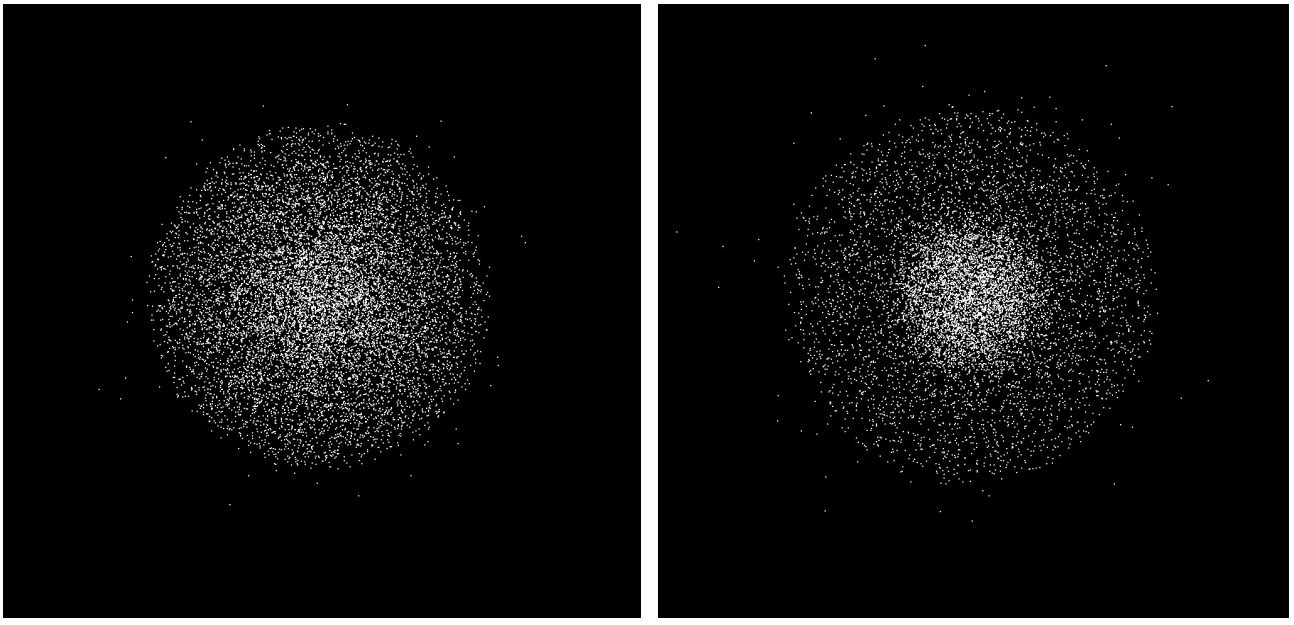
а) б)

Рисунок 7. Распределение плотности в начальный момент времени, а), и в момент времени t = P/2, где P характерный период пульсации системы



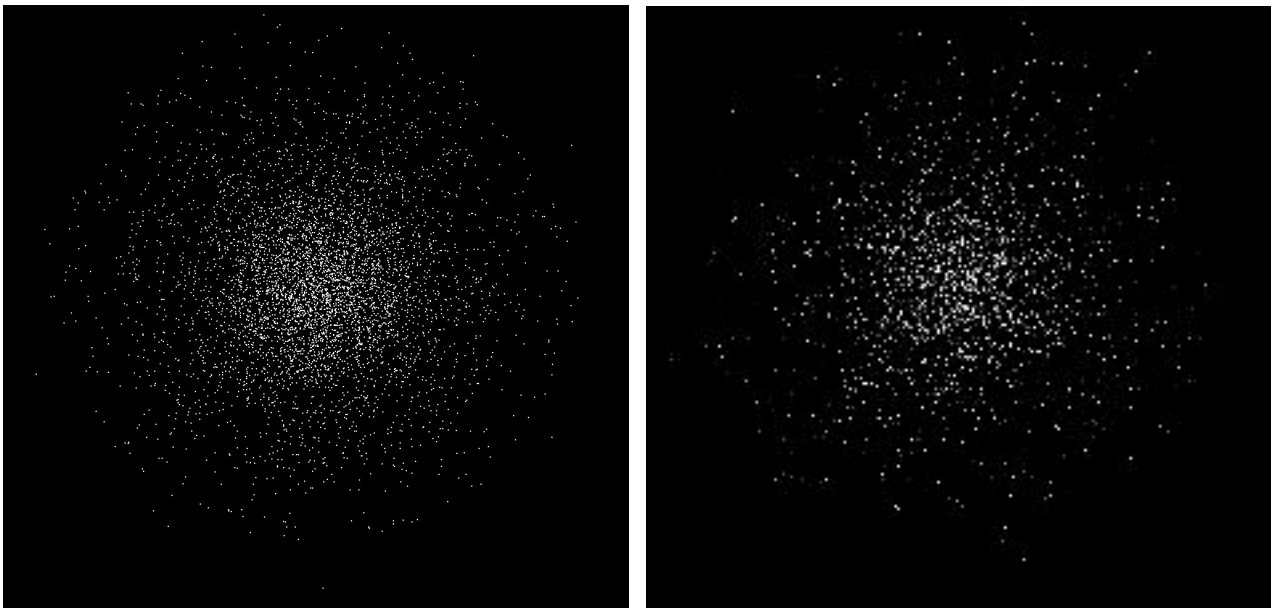
а) б)

Рисунок 8. Распределение плотности в момент времени t = 2P, а), и в момент времени t =5 P/2.



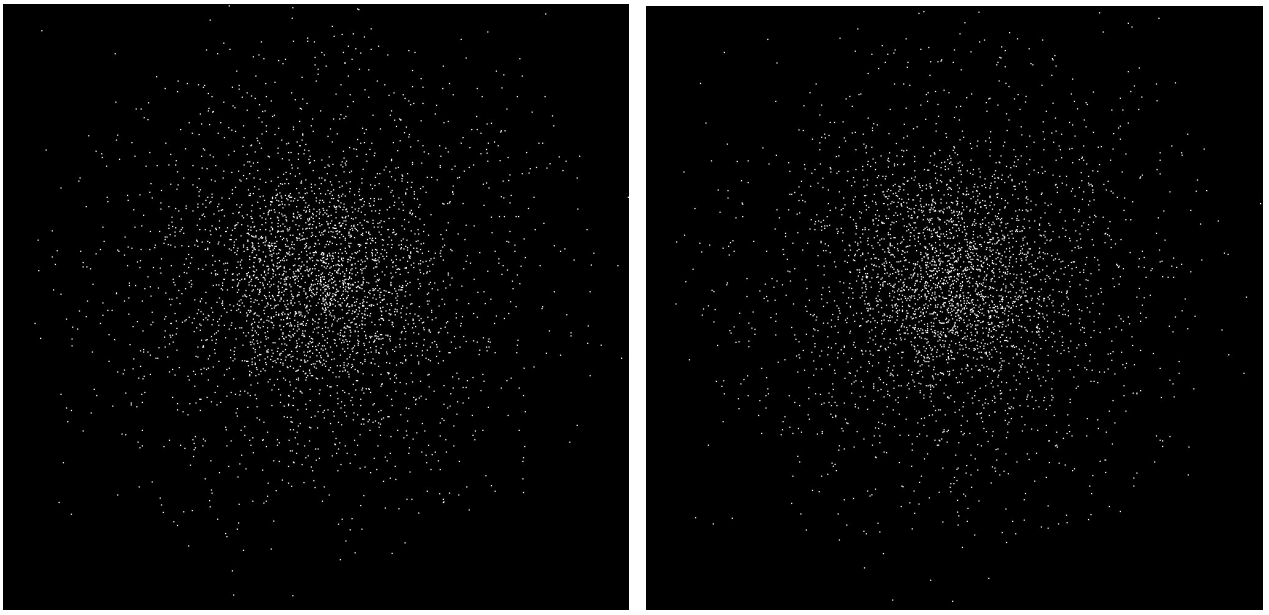
а) б)

Рисунок 9. Распределение плотности в момент времени t = 6P, а), и в момент времени t =13 P/2.



а) б)

Рисунок 10. Распределение плотности в момент времени t =10P, а), и в момент времени t =21 P/2.



а) б)

Рисунок 11. Установившееся распределение плотности, t = 20P, а) и t = 41P/2.

разных этапах расчета. Установление пространственной плотности наступало примерно после 20 пульсации.

Количество частиц в численном эксперименте составляло 16M (более 16 миллионов). Размерность расчетной сетки $64 \times 64 \times 64$. Оконный интерфейс создавался с использованием кроссплатформенной библиотеки GLFW версии 3.2.

ЗАКЛЮЧЕНИЕ

В диссертации получены следующие результаты:

1. Программно реализован метод частиц-в-ячейках с распараллеливанием по потокам графического процессора:
   - 1.1. Распараллелен метод свертки для нахождения потенциала гравитирующей системы на основе использования прикладной библиотеки CUDA cuFFT.
   - 1.2. Распараллелено распределение масс частиц в узлы расчетной сетки.
   - 1.3. Реализована интерполяция значений силового поля из узлов расчетной сетки в местоположения частиц на основе аппаратной линейной фильтрации с использование текстурной памяти (CUDA).
2. Разработан алгоритм инициализации начальной функции распределения посредством генерации выборок псевдо-случайных чисел на основе метода обращения и метода исключения Джона фон-Неймана.
3. Спроектировано и реализовано приложение для онлайн визуализации результатов моделирования течений разреженного газа на основе библиотеки OpenGL и программного интерфейса CUDA.
4. Проведены расчеты эволюции бесстолкновительных гравитирующих систем в области критических параметров устойчивости по отношению к возмущениям типа ядро-гало.

В целом, выполненная работа является основой для постановки численных экспериментов в области эволюции бесстолкновительных самогравитирующих систем. Дальнейшее ее развитие связано с углубленной разработкой пользовательского интерфейса и оптимизацией математических и программных алгоритмов метода частиц-в-ячейках. Также предполагается в будущем программно реализовать конечно-разностные методы решения уравнения

Власова в рамках принятого в диссертации подхода к разработке приложений с визуализацией в режиме онлайн.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Коган М.Н. Динамика разреженного газа. Кинетическая теория. М: Наука, 1967.
2. Fridman, A. M, and V. L Poli͡achenko. Physics of Gravitating Systems Equilibrium and Stability. Berlin, Heidelberg: Springer Berlin Heidelberg , 1984.
3. Bird, G.A. Molecular gas dynamics and the direct simulation of gas flows, Clarendon Press, 1994.
4. Аронин Г. С. Практическая аэродинамика. — М.: Воениздат, 1962. 384 с
5. Binney J., Tremaine S. Galactic dynamics, Princeton University Press, 1987. 731p.
6. Malkov E.A., Ivanov M.S. Parallelization of algorithms for solving the Boltzmann equation for GPU-based computations // 27th International Symposium on Rarefied Gas Dynamics AIP Conference Proceedings. - 2011. - Vol.1333. - P.946-951.
7. A. Frezzotti, G.P. Ghiroldi, L. Gibelli Solving the Boltzmann equation on GPUs
8. Computer Physics Communications 182 (2011) P. 2445–2453
9. Malkov E. A., Ivanov M. S. Discrete velocity scheme for solving the Boltzmann equation with the GPGPU // 28th International Symposium on Rarefied Gas Dynamics, 2012. AIP Conference Proceedings. 2012. V.1501. P. 318-325.
10. 10. High-accuracy deterministic solution of the Boltzmann equation for the shock wave structure / E. A. Malkov, S. O. Poleshkin, A. A. Kokhanchik, Ye. A. Bondar, M. S. Ivanov // Shock waves. 2015. V.25. P. 387-397.
11. Numerical approach for solving kinetic equations in two-dimensional case on hybrid computational clusters / E. A. Malkov, S. O. Poleshkin, A. N. Kudryavtsev, A. A. Shershnev // 18th International Conference on the Methods

of Aerophysical Research (ICMAR2016) AIP Conference Proceedings. S.l., 2016. Vol.1770. P. 030077. DOI: 10.1063/1.4964019

12. Хокни Р., Иствуд Дж. Численное моделирование методом частиц М.: Мир, 1987. - 640 с.
13. Григорьев Ю.Н., Вшивков В.А., Федорук М.П. Численное моделирование частиц-в-ячейках, Новосибирск, 2004, 358с.
14. Месяц Е. А., Снытников А. В., Лотов К. В. О выборе числа частиц в методе частиц-в-ячейках для моделирования задач физики плазмы // Вычислительные технологии, т. 18, № 6, с. 83-96, 2013.
15. Антонов В.А., Нуритдинов С.Н. Неустойчивость нелинейно пульсирующей модели звездной системы. Часть I - сфера Эйнштейна.// Астрономический журнал, т. 58, с. 1158-1166, 1981.
16. Malkov E.A. Stability of spherical collapse of collisionless gravitating systems//Astrophysics 24(2) P. 221-226, 1986. DOI: 10.1007/BF01025789
17. Malkov E.A., Nuzhnova T.N. Collisionless collapse of stellar systems: Analytical vision // Astronomical and Astrophysical Transactions 22(2), P. 237-244, 2003. DOI: 10.1080/1055679031000118391
18. Farber R. CUDA Application Design and Development, Elsevier, 315 p. , 2011.
19. А.В. Боресков и др. Параллельные вычисления на GPU. Архитектура и программная модель CUDA, Изд-во МГУ, 333с., 2012.
20. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах, Изд. ДМК, 232с., 2013.
21. Cook Sh. CUDA Programming A Developer-s Guide to Parallel Computing with GPUs, Morgan Kaufmann, 565c., 2013.
22. Korn G.A., Korn T.M. Mathematical Handbook for scientists and engineers. NY: McGraw-Hill, 1968.
23. http://www.cplusplus.com/reference/cstdlib/rand/

24. Солонина А.И. Основы цифровой обработки сигналов, Спб: БХВ-Петербург, 768 с., 2005.
25. Марчук Г.И. Методы вычислительной математики, М: Наука, 454 с., 1977.
26. www.nvidia.com CUFFT Library User's Guide, 88 p., 2018.
27. Рябенький В.С. Введение в вычислительную математику, М.: ФИЗМАТЛИТ, 296 с., 2000.
28. Kessenich J., Sellers G., Shreiner D. OpenGL Programming Guid Ninth Edition, Addison-Wesley, 803 p., 2017.
29. Rodriguez J. GLSL Essentials, Pact Publishing, 97 p., 2013.
30. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов., М: ДМК, 368 с., 2015.