САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И МНОГОПРОЦЕССОРНЫХ СИСТЕМ

Яшникова Анастасия Павловна

Магистерская диссертация

Оптимизация модели дыма Шредингера с учетом тягучести для вычислений на массивно–параллельных архитектура

Направление 02.04.02

Фундаментальная информатика и информационные технологии

Основная образовательная программа магистратуры

Вычислительные технологии

Научный руководитель:
профессор кафедры компьютерного
моделирования и многопроцессорных систем
к.т.н Дегтярев Александр Борисович

Санкт–Петербург
2019

Содержание

Введение 5
Глава 1. Дым Шредингера 10
1.1 Что такое дым Шредингера 10
1.2 Основной алгоритм дыма Шредингера 10
1.1.1. Базовый алгоритм несжимаемого потока Шредингера 11
1.1.2. Уравнение Шредингера интегрированное по времени 12
1.1.3. Расчет давления 13
1.3. Тест производительности 13
Глава 2. Вязкость 15
2.1. Уравнение неразрывности 15
2.2. Уравнение сохранения импульса 15
2.2.1. Уравнение Эйлера 15
2.2.1. Уравнение Навье–Стокса 16
2.3. Вязкая несжимаемой жидкости 16
2.4. Метод проекционного ограничения скорости 17
Глава 3. Реализация 19
3.1. Этапы реализации 19
3.2 Библиотека ArrayFire 19
3.3 CUDA 20
3.4. OpenCL 20
3.5. Тестирования 21
Выводы 29
Список литературы 30

Аннотация

Работа посвящена оптимизации модели дыма Шредингера для вычислений на массивно–параллельных архитектура с учетом тягучести.

В магистерской диссертации представлен способ ускорения алгоритма дыма Шредингера с помощью библиотеки ускоренных вычислений ArrayFire, которая поддерживает работу с OpenCL и CUDA. CUDA написана на объектно–ориентированно языке программирования C#. Приведены результаты тестов производительности. А также даны теоретические обоснования для внедрения параметра вязкости в алгоритм.

Работа содержит 31 страниц, 3 главы, 21 рисунок, 15 источников.

***Ключевые слова**: дым Шредингера, вязкость, ArrayFire, OpenCL, CUDA.*

Abstract

The work is devoted to optimization of the Schrödinger’s smoke model for computations on massively parallel architectures with account of viscosity.

The master's thesis presents a way to accelerate the Schrödinger’s smoke algorithm using the ArrayFire accelerated computing library, which supports OpenCL and CUDA. CUDA is written in an object-oriented C# programming language. The results of performance tests are presented. As well as the theoretical justification for the introduction of the viscosity parameter in the algorithm.

The work contains 31 pages, 3 chapters, 21 figure, 15 source.

Keywords: *schrödinger smoke, viscosity, ArrayFire, OpenCL, CUDA.*

Введение

В работе рассматривается новый подход моделирования несжимаемых жидкостей – дым Шредингера. С его помощью можно визуализировать созданную модель.

В компьютерной графике симуляция потока жидкости или газа является одной из наиболее любопытных и актуальных задач, ведь ее перед собой ставят специалисты в различных сферах деятельности – ученые, разработчики компьютерных игр, мультипликаторы.

Изначально в компьютерной графике разработка симуляций жидкостей была исключительно для научных целей. Например, моделирование атмосферных явлений (рис.1). Это стало возможно благодаря методам вычислительной гидродинамики. И несмотря на то, что сейчас более популярней становятся другие отрасли применения симуляции жидкости, использование средств вычислительной гидродинамики в научных целях не теряет своей актуальности в компьютерной графике. Интересно то, что порой модели, созданные в научных целях, не уступают в своей зрелищности и реалистичности симуляциям, использующиеся в коммерческих целях.

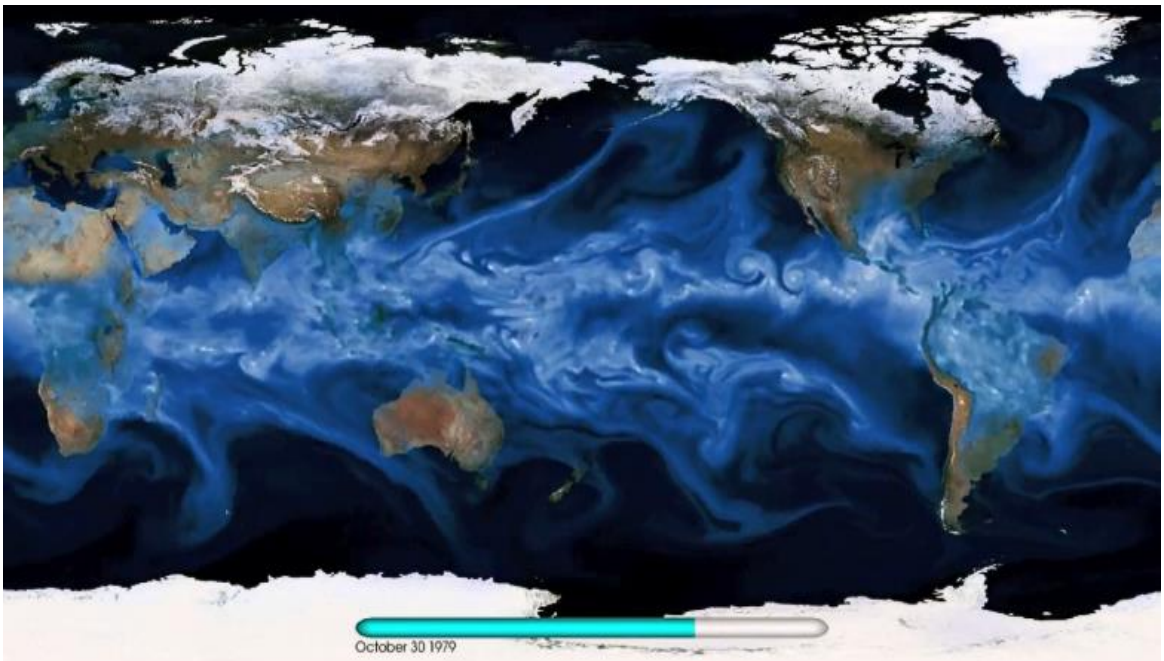


Рисунок 1. Моделирование атмосферных явлений по данным за октябрь 1979 г.

Наиболее популярным применением моделирования потока жидкостей являются спецэффекты в фильмах. Такие симуляции обычно требуют высокоточных вычислений и выглядят реалистично. На рис.2 представлена такая модель.

Рисунок 2. Компьютерное моделирование воды в фильме Оз: Великий и Могучий. Картинка сверху – как снималось на самом деле. Картинка снизу – что сделали с помощь. компьютерной графики.

Не меньшей популярностью симуляция жидкостей пользуется в мультфильмах и компьютерных играх. И если в мультфильмах художники нередко стремятся создать реалистичные модели (рис.3), вычисления которых зачастую занимают часы и даже месяцы, то в компьютерных играх используются менее точные вычисления, которые производятся в режиме реального времени (рис. 4).

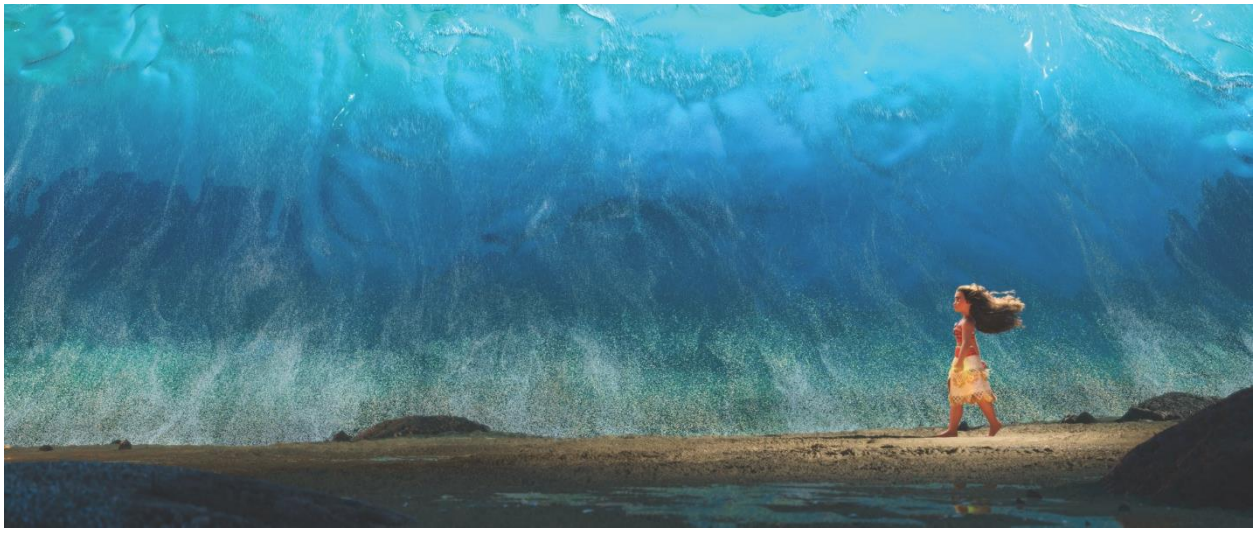


Рисунок 3. Реализация воды в мультфильме Моана.

Рисунок 4. Реализация воды в компьютерной игре The Elder Scrolls V: Skyrim

При моделировании потока жидкости стоит учитывать несколько важных факторов. Пожалуй, одним из самых значимых условий является возможность отобразить некоторые физические характеристики потока в компьютерной модели, например, плотность, скорость, сжимаемость/несжимаемость, давление, теплопроводность и температура. Также возможность добавлять в модель объекты, мешающие движению потоку (различные препятствия), и конечно же симуляция взаимодействия двух жидкостей.

В зависимости от поставленной задачи, может отдавать предпочтения различным типам моделей. Например, сеточные методы подходят для симуляции гладкой водной поверхности как нельзя лучше. А в методах с использованием частиц не возникнет трудностей с соблюдением закона сохранения массы (или эти проблемы можно решить путем небольших модификаций), и обычно работают быстрее сеточных симуляций.

В последние годы популярность стали набирать так называемые смешанные методы, которые использует подходящие для решения задачи преимущества одной из групп методов. Однако гибридные методы требуют больших вычислительных затрат, которые не всегда себя оправдывают.

Разработчики модели сталкиваются с такой сложной проблемой как поиск компромисса между реалистичностью симуляции и разумным использованием имеющихся вычислительных ресурсов. Для построения точной и реалистичной модели, как правило, используются суперкомпьютеры с большим объемом оперативной памяти и видеопамяти.

Рассматриваемый в данной работе алгоритм, называющийся дымом Шредингера, был разработан и представлен на конференции SIGGRAPH в 2016 году [1]. Он позволяет преодолевать традиционно возникающие проблемы в частях алгоритма, связанных с завихренностью и адвекцией потока. В то же время, по словам авторов этого метода, он дает реалистичные результаты, является стабильным и относительно несложным в реализации, при этом не требуя большого количества вычислительных ресурсов. Но у него есть некоторые недостатки.

Хоть алгоритм и не требует больших вычислительных ресурсов, некоторые из его частей являются ресурсозатратными (например, быстрое преобразование Фурье). Из–за этого увеличивается время на вычисления модели.

В рамках данной магистратской диссертации была поставлена цель адаптировать модель дыма Шредингера для вычислений на массивно–параллельных архитектурах, в которой также будет учитываться вязкость используемых веществ.

Для достижения данной цели нужно было решить следующие задачи:

1. Изучить алгоритм несжимаемого потока Шредингера и оптимизировать его с помощью объекто – ориентированного языка программирования C#.
2. Найти теоретическое обоснование добавления параметра вязкости вещества в существующую модель.
3. Перенести полученный код на GPU с использованием библиотеки ускоренных вычислений ArrayFire, которая позволяет запускать приложения на CUDA или OpenCL для параллельных вычислений.
4. Провести ряд тестов на производительность полученного кода.

Работа структурирована следующим образом:

- В первой главе рассказывается о подходите моделирования несжимаемый жидкостей, описывается его исходный алгоритм.
- Вторая глава посвящена обоснованию параметра вязкости, который можно добавить в реализованную модель.
- В третьей главе более подробно рассказывается о переносе алгоритма дыма Шредингера. Также представлены результаты ряда тестирований производительности к реализованного кода.
- В заключении представлены выводы про проделанной работе.

Глава 1. Дым Шредингера

В этой главе рассказывается о том, что такое дым Шредингера, описывается его исходный алгоритм.

1.1 Что такое дым Шредингера

Дым Щредингера – этот тезис вводит новый взгляд на динамику несжимаемой жидкости. В частности, формулируются и моделируются классические жидкости, с использованием $\mathbb{C}^2$–значного уравнения Шредингера с учетом ограничения несжимаемость. Такой поток жидкости называется несжимаемым потоком Шредингера (ISF). Базовая динамическая система основана на гамильтоновой системе, и управляется за счет кинетической энергией жидкости вместе с энергией типа Ландау–Лифшица. Последнее гарантирует, что динамика из–за тонкие вихревые структуры, которые важны для моделирования, точно будут воспроизведены. Это позволяет надежно моделировать сложные явления, такие как вихревые следы и взаимодействующие вихревые нити, даже на сетках небольшого размера [3] (рис.1).

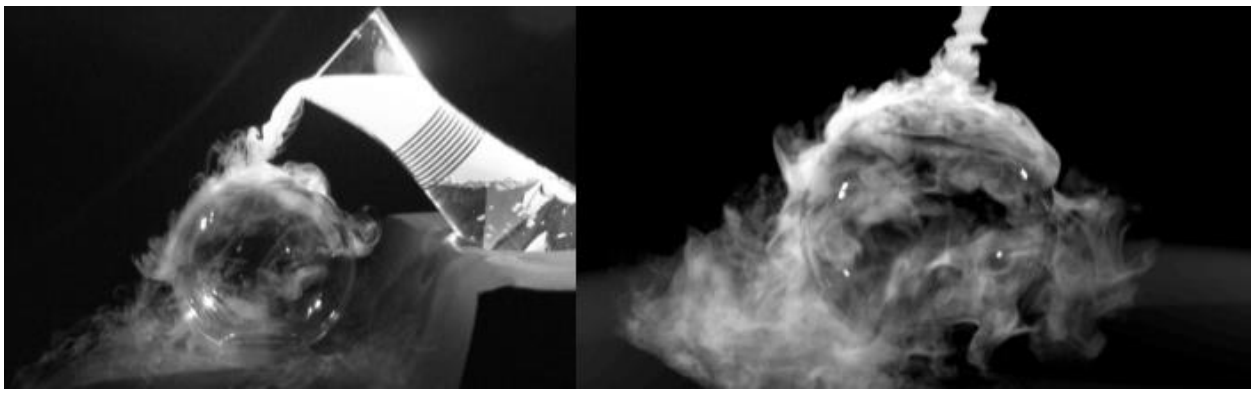


Рисунок 1: Сравнение эксперимента (пары сухого льда, слева) с имитацией ISF (справа)

1.2 Основной алгоритм дыма Шредингера

Вычисления выполняются на трехмерной сетке (рис. 2):

$$\mathcal{V} = \{0, \dots, \mathcal{N}_x - 1\} \times \{0, \dots, \mathcal{N}_y - 1\} \times \{0, \dots, \mathcal{N}_z - 1\}$$

$\mathcal{N}_x, \mathcal{N}_y, \mathcal{N}_z$ – размеры сетки по соответствующим осям.

Вершинах $v \in \mathcal{V}$ хранятся значения волновой функции $\psi_v \in \mathbb{C}^2$, давления $q_v \in \mathbb{R}$, дивергенции $\xi_v \in \mathbb{R}$.

Дискретная скорость в алгоритме определяется как вес направленных ребер сетки $vw \in \mathcal{E}$:

$$\eta_{vw} = \hbar \, arg\langle \psi_v, \psi_w \rangle_{\mathbb{C}},$$

$$\eta_{vw} = -\eta_{wv}$$

значения которой хранятся в шахматном порядке в вершинах сетки.

Дискретная дивергенция имеет следующий вид:

$$\xi_v = \frac{1}{V_v} \sum_{vw \in \mathcal{E}} \frac{A_{vw}}{l_{vw}} \eta_{vw},$$

где A_{vw} – удвоенная площадь грани, l_{vw} – длина ребра сетки, V_v – удвоенный объем ячейки.

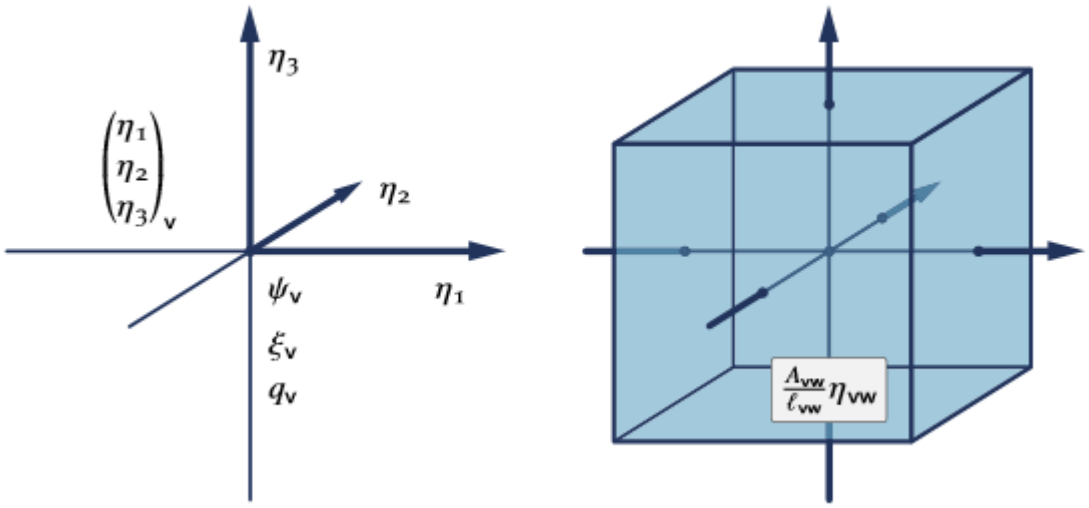


Рисунок 2. Трехмерная сетка с набором вершин для вычислений. Вершинах $v \in \mathcal{V}$ хранятся значения волновой функции $\psi_v \in \mathbb{C}^2$, давления $q_v \in \mathbb{R}$, дивергенции $\xi_v \in \mathbb{R}$.

1.1.1. Базовый алгоритм несжимаемого потока Шредингера

В этом алгоритме применяется оператор расщепления, выполняется нормализация, а также рассчитывается давление. Так же здесь можно проводить учет препятствий и плавучесть (с учетом гравитационных сил).

Начальные значения авторы метода нашли с помощью подбора.

На вход подается начальное значение функции $\psi^{(0)}$, временной шаг $dt > 0$, а также сила квантизации $\hbar > 0$.

Базовый алгоритм несжимаемого потока Шредингера представлен ниже.

Algorithm: Basic ISF

Input:$\psi^{(0)}, dt, \hbar$ //начальное состояние и параметры

for j← 0,1,2,… **do**

$\psi^{tmp} \leftarrow Schrödinger(\psi^{(j)}, dt, \hbar)$

$\psi^{tmp} \leftarrow \frac{\psi^{tmp}}{|\psi^{tmp}|}$ //нормализация

$\psi^{(j+1)} \leftarrow PressureProject(\psi^{tmp})$

end for

1.1.2. Уравнение Шредингера интегрированное по времени

Данная функция приводить матрицу (сетку) к диагональному виду. Для этого используется алгоритм быстрого преобразования Фурье (БПФ). Для значений на гранях сетки – дискретное косинусное преобразование (ДКТ).

Здесь λ_v – собственные значения трехмерного непрерывного оператора Лапласа.

Функция интегрирования по времени уравнения Шредингера представлена ниже.

Algorithm: Time integration of Schrödinger equation

function $Schrödinger(\psi, dt, \hbar)$

$\hat{\psi} \leftarrow FFT3D(\psi)$

$\hat{\psi} \leftarrow e^{i\lambda dt\frac{\hbar}{2}}\hat{\psi}$

return $InvFFT3D(\hat{\psi})$

end function

1.1.3. Расчет давления

Данная функция рассчитывает давление несжимаемой жидкости как внутри, так и на его стенки ограничивающего объема.

Функция расчета давления представлена ниже.

Algorithm: Divergence free constrain

function $PressureProject(\psi)$

for each $vw \in \mathcal{E}$ **do**

$\tilde{\eta}_{vw} = arg\langle\psi_v, \psi_w\rangle_{\mathbb{C}}$

end for

for each $v \in \mathcal{V}$ **do**

$\xi_v = \frac{1}{V_v}\sum_{vw\in\mathcal{E}} \frac{A_{vw}}{l_{vw}}\tilde{\eta}_{vw}$

end for

$\hat{\xi} \leftarrow FFT3D(\xi)$

$\hat{\xi} \leftarrow \hat{\xi}\begin{cases}\tilde{\lambda}^{-1} \\ 0\end{cases}, if\tilde{\lambda} \neq 0$

$q \leftarrow InvFFT3D(\hat{\xi})$

return $e^{-iq}\psi$ //калибровочное преобразование

end function

1.3. Тест производительности

Реализованный алгоритм дыма Шредингера был протестировали на CPU и GPU.

Бенчмарк (усредненние по набору из трех запусков) реализации заданного количества частиц (ось x), выполненная за время, измеряющиеся в миллисекундах (ось y). Синяя (нижняя) линия показывает время, затраченное на реализацию GPU. Красная (вверхняя) линия показывает время процессора.

Результаты теста были получены на процессоре Intel i5 7–го поколения с GPU, интегрированным на том же чипе[2].

Полученные после проведения испытаний, результаты представленные ниже (рис.3), демонстрируют, что модель можно ускорить даже для использования на встроенных графических процессорах без каких–либо кардинальных изменений в ее логике.

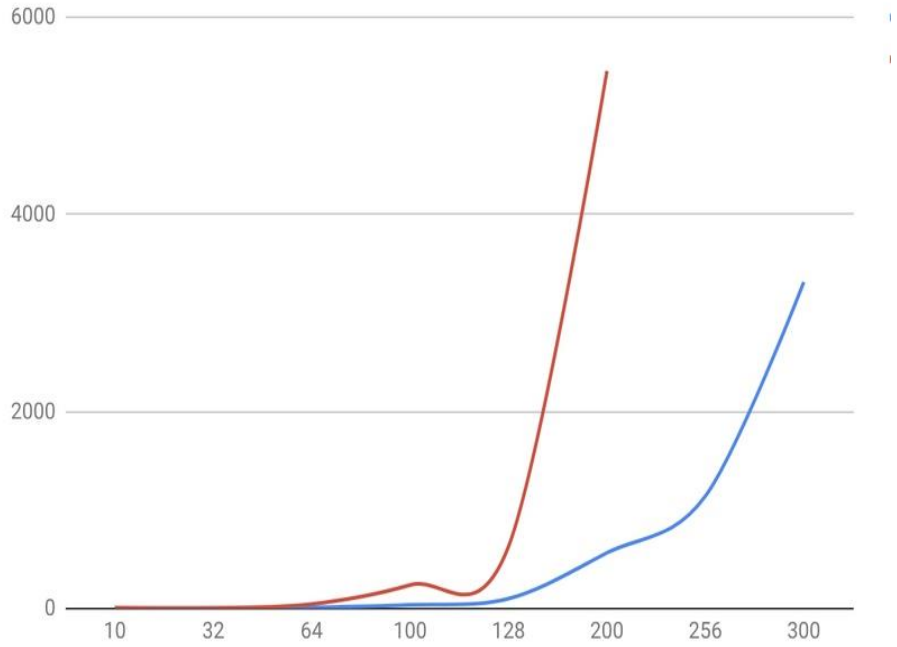


Рисунок 3. Бенчмарк реализации заданного количества частиц, выполненная за время, измеряющиеся в миллисекундах.

Глава 2. Вязкость

В этой главе рассказывается о теоретической возможности добавления параметра вязкости в алгоритм моделирования несжимаемых жидкостей, которая позволит моделировать вязкие несжимаемые жидкости.

2.1. Уравнение неразрывности

Данное уравнение является выражением закона сохранения массы в элементарном объеме, или по–другому – связи изменения потока массы газа или жидкости и скорости изменения плотности вещества со временем в пространстве [7].

Уравнение неразрывности имеет следующую дифференциальную форму:

$$\frac{\partial \rho}{\partial t} + \operatorname{div} \rho v = \frac{\partial \rho}{\partial t} + \rho \operatorname{div} v + v \cdot \ grad\, \rho = 0$$

где $\rho = \rho(x, y, z, t)$ – плотность газа или жидкости, $v = v(x, y, z, t)$ – вектор скорости газа или жидкости в точке, имеющей координаты (x, y, z) в момент времени t, вектор ρv – плотность потока жидкости.

Если уравнение составляется для однородной несжимаемой жидкости, то $\rho = const$ и уравнение имеет вид:

$$\operatorname{div} v = 0$$

Из этого предполагается, что поле скорости можно считать соленоидальным (или вихревым).

2.2. Уравнение сохранения импульса

Вид этого уравнения зависит от наличия или отсутствия трения. Уравнение Навье–Стокса применяется для потоков с трением, уравнение Эйлера используется для потоков без трения.

2.2.1. Уравнение Эйлера

Данное уравнение является уравнением движения идеальной жидкости. Уравнение Эйлера для движения идеальной жидкости в поле тяжести имеет

следующий вид [7].

$$\frac{\partial v}{\partial t} + (v \cdot \nabla)v = g - \frac{1}{p}\nabla p$$

где $\rho(x, y, z, t)$ – плотность жидкости, $p(x, y, z, t)$ – давление в жидкости, $v(x, y, z, t)$ – вектор скорости жидкости, $g(x, y, z, t)$ – вектор напряженности силового поля, ∇ – оператор набла для трехмерного пространства.

2.2.1. Уравнение Навье–Стокса

Уравнение Навье–Стокса – система дифференциальных уравнений в частных производных, которой описывается движение вязкой ньютоновской жидкости. В случае несжимаемой жидкости данная система состоит из двух уравнений – движения и неразрывности, однако в гидродинамике под уравнением Навье – Стокса понимается только одно уравнение движения. [6]

Уравнение для жидкости в векторном виде записывается так:

$$\frac{d\vec{v}}{dt} = -(\vec{v} \cdot \nabla)\vec{v} + \nu\Delta\vec{v} - \frac{1}{\rho}\nabla p + \vec{f}$$

где $\vec{v} = (v^1, v^2, \dots, v^n)$ – векторное поле скоростей, t – время, ∇ – оператор набла, Δ – векторный оператор Лапласа, ν – коэффициент кинематической вязкости, ρ – плотность, p – давление, $\vec{f}$ – векторное поле массовых сил.

При учете сжимаемости жидкости, а также постоянства динамической и объемной вязкостей уравнения Навье–Стокса имеют вид [6]:

$$\rho\left(\frac{d\vec{v}}{dt} + (\vec{v} \cdot \nabla)\vec{v}\right) = -\nabla p + \eta\Delta\vec{v} + \left(\zeta + \frac{\eta}{3}\right)\nabla \operatorname{div} \vec{v}$$

где η – коэффициент динамической (сдвиговой) вязкости, ζ – коэффициент объемной вязкости.

2.3. Вязкая несжимаемой жидкости

В случае несжимаемой жидкости плотность $\rho = const$, а уравнение неразрывности имеет вид

$$\operatorname{div} v = 0$$

или в проекции на оси координат

$$\frac{dv_x}{dx} + \frac{dv_y}{dy} + \frac{dv_z}{dz} = 0$$

Если предположить, что вязкость будет постоянной по всему объему, то получим следующее уравнение движения.

$$\frac{dv_x}{dt} + v_x\frac{dv_x}{dx} + v_y\frac{dv_y}{dy} + v_z\frac{dv_z}{dz} = X - \frac{1}{\rho}\frac{dp}{dx} + v\left(\frac{d^2v_x}{dx^2} + \frac{d^2v_x}{dy^2} + \frac{d^2v_x}{dz^2}\right)$$

$$\frac{dv_y}{dt} + v_x\frac{dv_y}{dx} + v_y\frac{dv_y}{dy} + v_z\frac{dv_y}{dz} = Y - \frac{1}{\rho}\frac{dp}{dy} + v\left(\frac{d^2v_y}{dx^2} + \frac{d^2v_y}{dy^2} + \frac{d^2v_y}{dz^2}\right)$$

$$\frac{dv_z}{dt} + v_x\frac{dv_z}{dx} + v_y\frac{dv_z}{dy} + v_z\frac{dv_z}{dz} = Z - \frac{1}{\rho}\frac{dp}{dz} + v\left(\frac{d^2v_z}{dx^2} + \frac{d^2v_z}{dy^2} + \frac{d^2v_z}{dz^2}\right)$$

2.4. Метод проекционного ограничения скорости

Мы строим первую форму скорости через проекцию ограничения, которая обеспечивает ограничение скорости и впоследствии обеспечивает исчезающую дивергенцию через проекцию давления.

Метод проекционного ограничения скорости представлен ниже.

Algorithm: Velocity constraint projection

function $Constraint\ Projection(\psi, \Omega, k, \hbar, t)$

$\psi^{tmp} \leftarrow \psi$

$\psi^{tmp}|_\Omega \leftarrow \xi^{k,t,\hbar}(|\psi_1|, |\psi_2|)^T$

return PressureProject(ψ^{tmp})

end function

Чтобы смоделировать струю (рис.4), потребуется применить фиксированную скорость в некоторой области Ω, сопле и на протяжении всего моделирования.

Рисунок 4. Смоделированная струя с применением фиксированной скорости в некоторой области **Ω**, сопле и на протяжения. Слева на право уменьшается параметр квантовой завихренности, иллюстрирующий более мелкие детали.

Для этого мы используем метод принудительного воздействия на объем. [12]. Этот метод был разработан для стандартного метода моделирования жидкости и включает в себя ограничение через параметр η в уравнении Навье–Стокса.

$$\frac{d}{dt}v(v\cdot\nabla)v = \nu\Delta v - \nabla p - \frac{1}{\eta}X_{\Omega}(v - v_{\Omega})$$

$$\operatorname{div} v = 0,$$

Глава 3. Реализация

3.1. Этапы реализации

На рисунке 5 показаны какие этапы надо пройти, чтобы получить конечную реализацию кода.

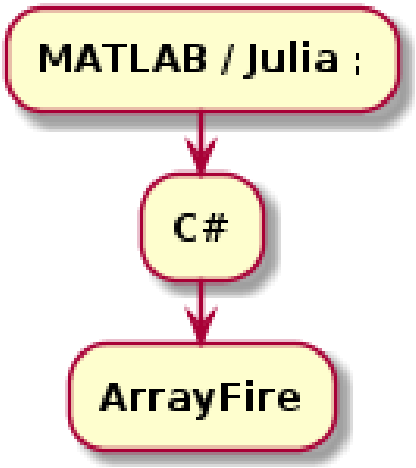


Рисунок 5. Этапы переноса алгоритма

3.2 Библиотека ArrayFire

Библиотека ускоренных вычислений ArrayFire представляет собой упрощенный способ для распараллеливания вычислительных операций с помощью включения в него GPU.

ArrayFire – это универсальный вычислительный инструмент со свободным доступом к своему открытому исходному коду, который облегчает процесс разработки программного обеспечения для параллельных и массивно – параллельных архитектур, включая процессоры, графические процессоры и другие аппаратные ускорители [9].

Библиотека имеет ряд своих особенностей таких как [13]:

- Доступно на C/C ++, Python
- Объединяет и расширяет все доступные библиотеки CUDA и OpenCL, включая такие реализации FFT, BLAS и LAPACK

- Позволяет заменить 10 – 100X строк в исходном коде CUDA или OpenCL на несколько строк кода ArrayFire
- Простая интеграция с приложениями CUDA и OpenCL
- Содержит множество функций, необходимых для ускорения кода, включая арифметику, линейную алгебру, статистику, обработку сигналов изображений и связанных с ними алгоритмов
- Поддерживает манипуляции с векторами, матрицами и N – мерными массивами

3.3 CUDA

CUDA – это параллельная вычислительная платформа, разработанная компанией NVIDIA для общих вычислений на графических процессорах (GPU) [14]. С помощью CUDA разработчики могут значительно ускорить вычислительные приложения, используя мощность графических процессоров.

В GPU–ускоренных приложениях последовательная часть рабочей нагрузки выполняется на CPU, который оптимизирован для однопоточной производительности, в то время как вычислительная часть приложения работает на тысячах ядер GPU параллельно.

Инструментарий CUDA от NVIDIA предоставляет все необходимое для разработки GPU–ускоренных приложений. Инструментарий CUDA включает библиотеки с ускорением GPU, компилятор, средства разработки и среду выполнения CUDA.

3.4. OpenCL

OpenCL (Open Computing Language) –это низкоуровневый API для вычислений, работающий на графических процессорах с поддержкой CUDA [15]. Используя API OpenCL, разработчики могут запускать вычислительные ядра.

В дополнение к OpenCL, NVIDIA поддерживает различные библиотеки с ускорением GPU и высокоуровневые программные решения, которые позволяют разработчикам быстро начать работу с GPU-вычислениями.

3.5. Тестирования

На рисунках 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 представлены результаты тестов скорости, зависящей от объемов.

На рисунках 17, 18, 19 результаты загрузки ЦП.

На рисунках 20, 21, 22 результаты «горячих» кодов.

```
WGL: The driver does not appear to support OpenGLError: Could not Create GLFW Window!
ArrayFire v3.6.2 (CPU, 64-bit Windows, build dc38ef13)
[0] Intel: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, 57343 MB, Max threads(8)
w       h       d       ms (avg on 10 runs)     ms (min on 10 runs)     ms (max on 10 runs)
10,     10,     10,     168,    0,      1685
32,     32,     32,     5,      4,      6
64,     64,     64,     46,     43,     67
100,    100,    100,    176,    170,    183
128,    128,    128,    388,    379,    392
200,    200,    200,    1505,   1491,   1524
```

Рисунок 6

```
WGL: The driver does not appear to support OpenGLError: Could not Create GLFW Window!
ArrayFire v3.6.2 (CPU, 64-bit Windows, build dc38ef13)
[0] Intel: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, 57343 MB, Max threads(8)
w       h       d       ms (avg on 10 runs)     ms (min on 10 runs)     ms (max on 10 runs)
32,     32,     32,     6,      4,      9
32,     32,     32,     5,      4,      6
64,     64,     64,     46,     43,     50
100,    100,    100,    178,    173,    182
128,    128,    128,    388,    379,    391
200,    200,    200,    1523,   1514,   1549
```

Рисунок 7

```
WGL: The driver does not appear to support OpenGLError: Could not Create GLFW Window!
ArrayFire v3.6.2 (CPU, 64-bit Windows, build dc38ef13)
[0] Intel: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, 57343 MB, Max threads(8)
w       h       d       ms (avg on 10 runs)     ms (min on 10 runs)     ms (max on 10 runs)
70,     70,     70,     63,     53,     68
32,     32,     32,     5,      4,      7
64,     64,     64,     47,     44,     54
100,    100,    100,    184,    174,    204
128,    128,    128,    407,    399,    421
200,    200,    200,    1551,   1524,   1600
```

Рисунок 8

```
WGL: The driver does not appear to support OpenGLError: Could not Create GLFW Window!
ArrayFire v3.6.2 (CPU, 64-bit Windows, build dc38ef13)
[0] Intel: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, 57343 MB, Max threads(8)
w       h       d       ms (avg on 10 runs)     ms (min on 10 runs)     ms (max on 10 runs)
100,    100,    100,    188,    178,    196
32,     32,     32,     5,      4,      7
64,     64,     64,     44,     42,     47
100,    100,    100,    179,    173,    193
128,    128,    128,    413,    400,    449
200,    200,    200,    1617,   1526,   1886
```

Рисунок 9.

```
WGL: The driver does not appear to support OpenGLError: Could not Create GLFW Window!
ArrayFire v3.6.2 (CPU, 64-bit Windows, build dc38ef13)
[0] Intel: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, 57343 MB, Max threads(8)
w       h       d       ms (avg on 10 runs)     ms (min on 10 runs)     ms (max on 10 runs)
128,    128,    128,    399,    382,    440
32,     32,     32,     5,      4,      6
64,     64,     64,     44,     44,     45
100,    100,    100,    173,    136,    181
128,    128,    128,    391,    383,    397
200,    200,    200,    1525,   1519,   1544
```

Рисунок 10.

```
WGL: The driver does not appear to support OpenGLError: Could not Create GLFW Window!
ArrayFire v3.6.2 (OpenCL, 64-bit Windows, build dc38ef13)
[0] NVIDIA: Tesla K80, 11448 MB
w       h       d       ms (avg on 10 runs)     ms (min on 10 runs)     ms (max on 10 runs)
10,     10,     10,     29,     0,      282
32,     32,     32,     13,     0,      122
64,     64,     64,     14,     5,      85
100,    100,    100,    26,     17,     82
128,    128,    128,    51,     39,     124
200,    200,    200,    173,    154,    235
```

Рисунок 11.

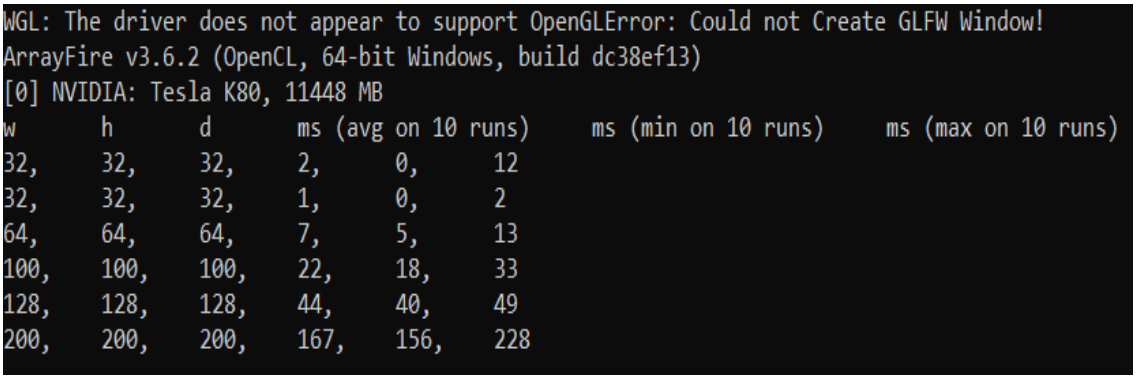


Рисунок 12.

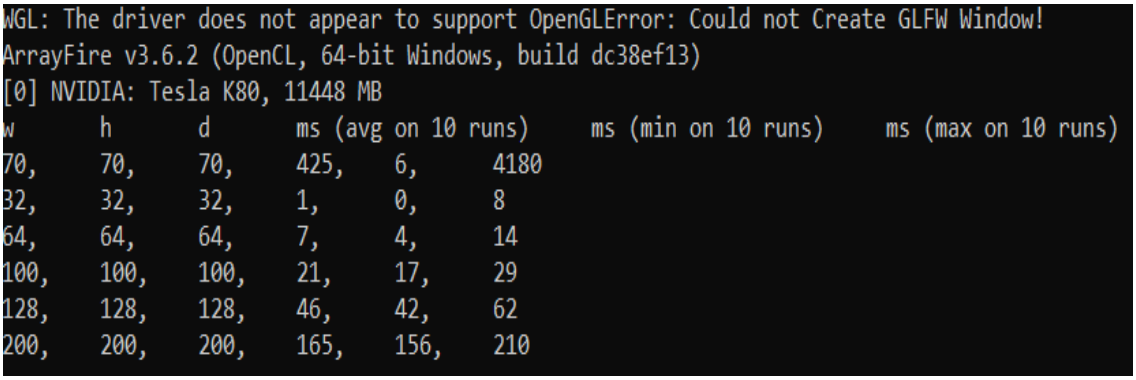


Рисунок 13.

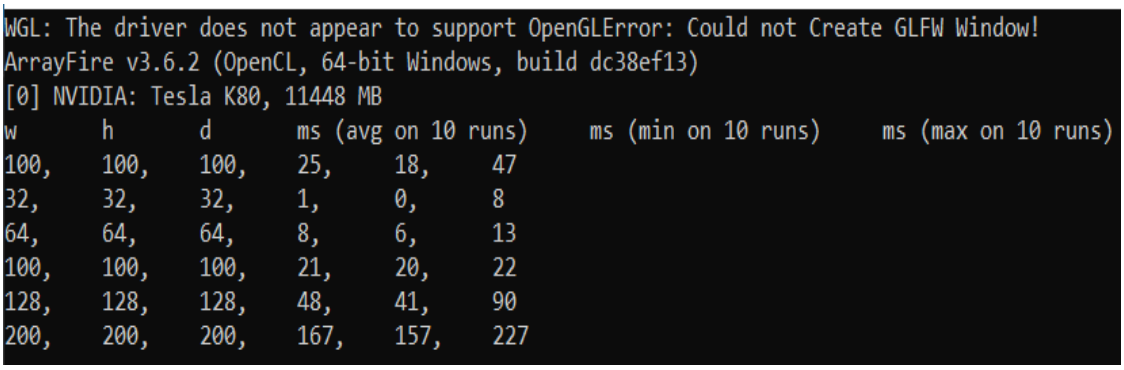


Рисунок 14.

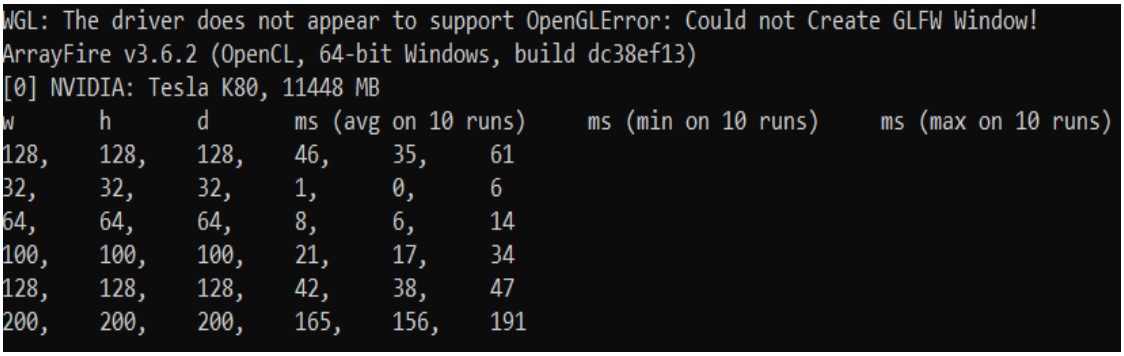
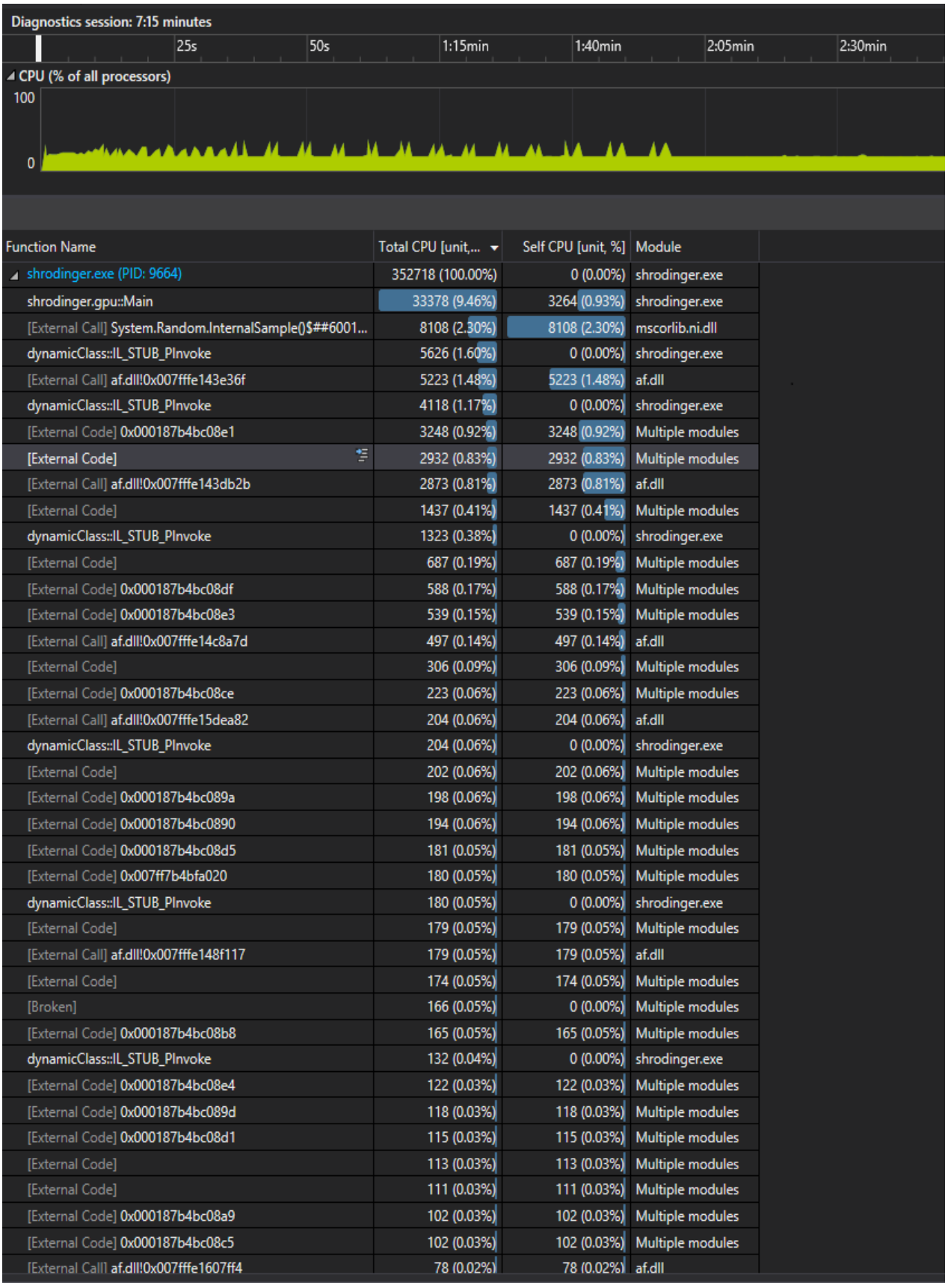
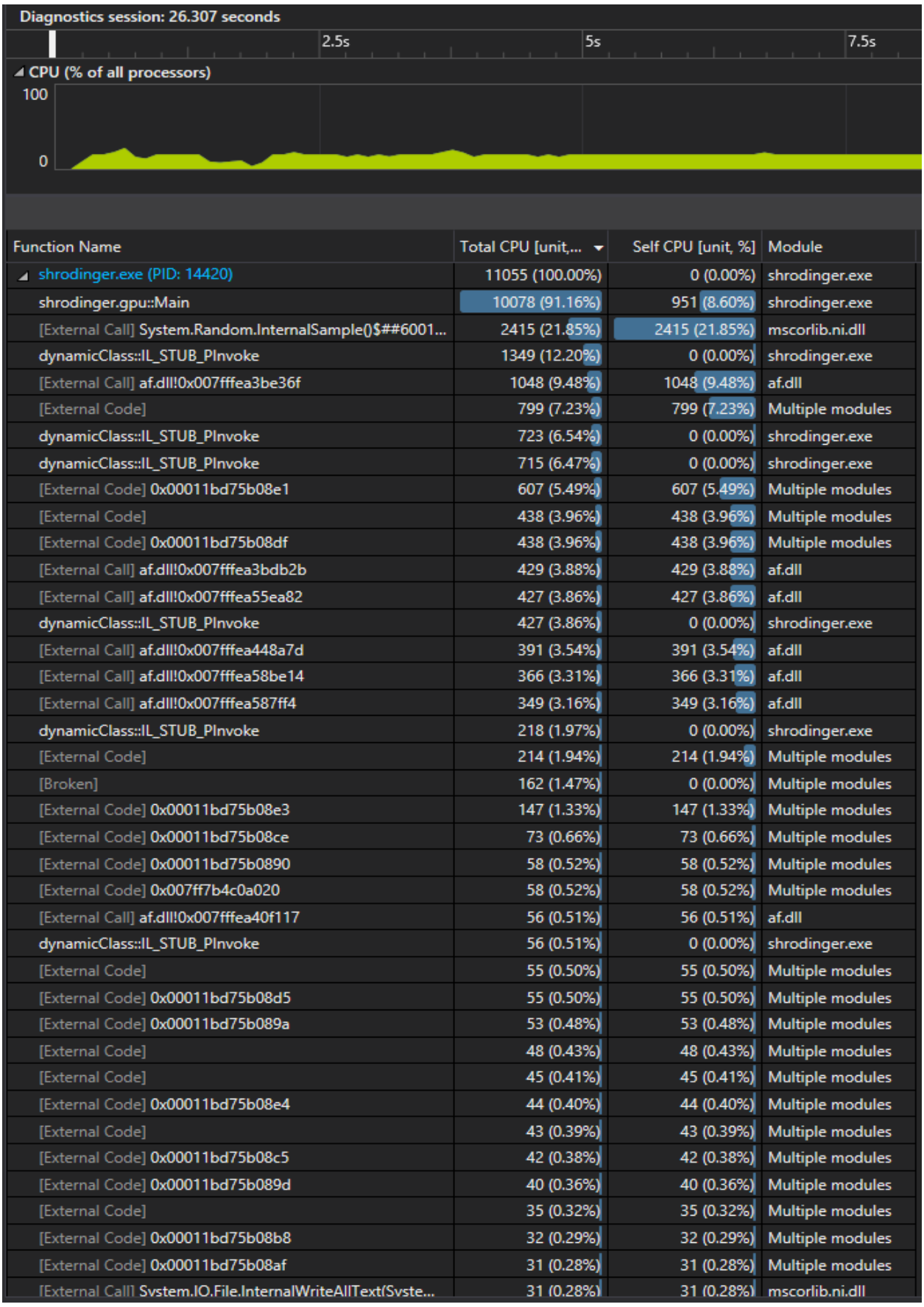


Рисунок 15.



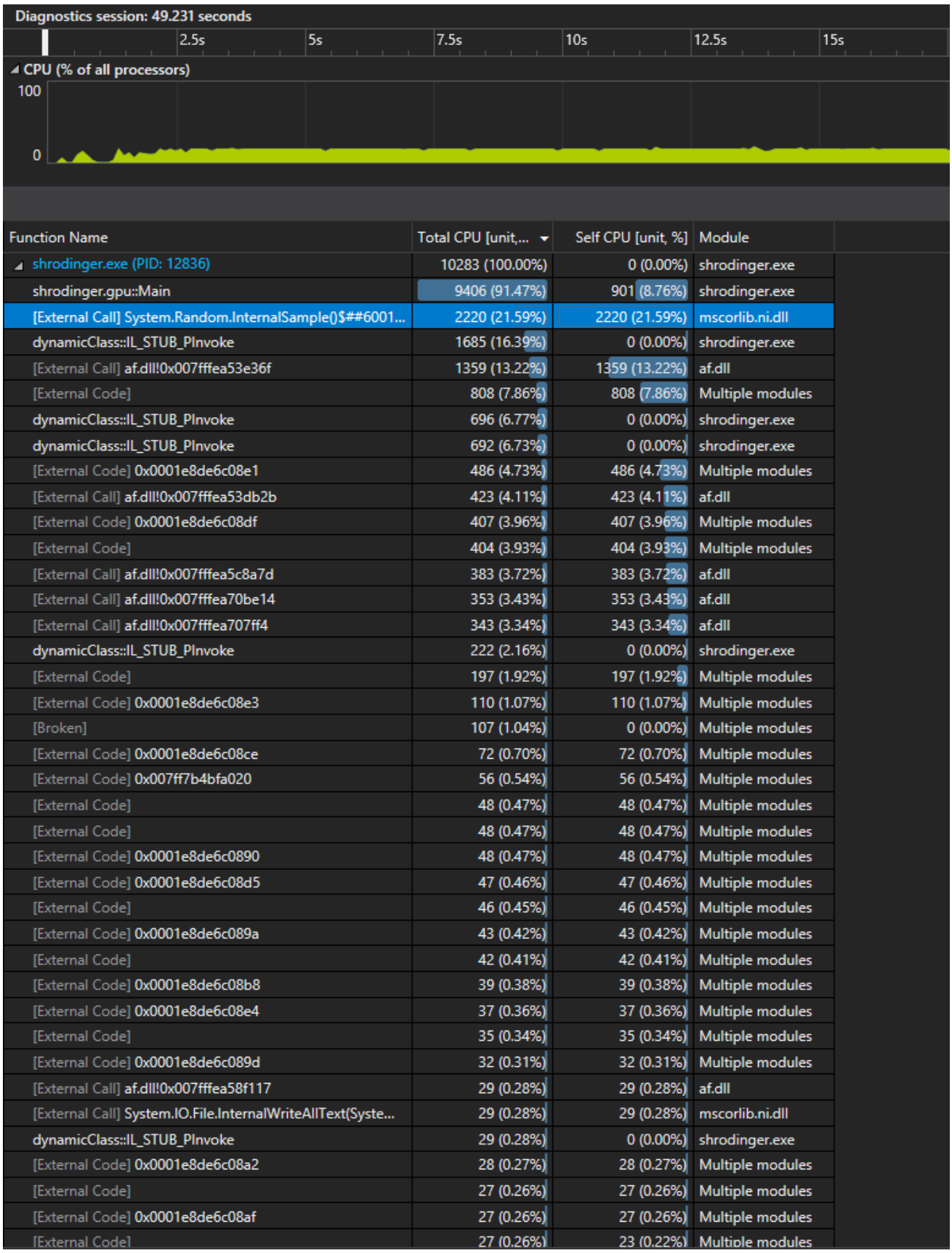
Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
shrodinger.exe (PID: 9664)	352718 (100.00%)	0 (0.00%)	shrodinger.exe
shrodinger.gpu::Main	33378 (9.46%)	3264 (0.93%)	shrodinger.exe
[External Call] System.Random.InternalSample()$##6001...	8108 (2.30%)	8108 (2.30%)	mscorlib.ni.dll
dynamicClass::IL_STUB_PInvoke	5626 (1.60%)	0 (0.00%)	shrodinger.exe
[External Call] af.dll!0x007fffe143e36f	5223 (1.48%)	5223 (1.48%)	af.dll
dynamicClass::IL_STUB_PInvoke	4118 (1.17%)	0 (0.00%)	shrodinger.exe
[External Code] 0x000187b4bc08e1	3248 (0.92%)	3248 (0.92%)	Multiple modules
[External Code]	2932 (0.83%)	2932 (0.83%)	Multiple modules
[External Call] af.dll!0x007fffe143db2b	2873 (0.81%)	2873 (0.81%)	af.dll
[External Code]	1437 (0.41%)	1437 (0.41%)	Multiple modules
dynamicClass::IL_STUB_PInvoke	1323 (0.38%)	0 (0.00%)	shrodinger.exe
[External Code]	687 (0.19%)	687 (0.19%)	Multiple modules
[External Code] 0x000187b4bc08df	588 (0.17%)	588 (0.17%)	Multiple modules
[External Code] 0x000187b4bc08e3	539 (0.15%)	539 (0.15%)	Multiple modules
[External Call] af.dll!0x007fffe14c8a7d	497 (0.14%)	497 (0.14%)	af.dll
[External Code]	306 (0.09%)	306 (0.09%)	Multiple modules
[External Code] 0x000187b4bc08ce	223 (0.06%)	223 (0.06%)	Multiple modules
[External Call] af.dll!0x007fffe15dea82	204 (0.06%)	204 (0.06%)	af.dll
dynamicClass::IL_STUB_PInvoke	204 (0.06%)	0 (0.00%)	shrodinger.exe
[External Code]	202 (0.06%)	202 (0.06%)	Multiple modules
[External Code] 0x000187b4bc089a	198 (0.06%)	198 (0.06%)	Multiple modules
[External Code] 0x000187b4bc0890	194 (0.06%)	194 (0.06%)	Multiple modules
[External Code] 0x000187b4bc08d5	181 (0.05%)	181 (0.05%)	Multiple modules
[External Code] 0x007ff7b4bfa020	180 (0.05%)	180 (0.05%)	Multiple modules
dynamicClass::IL_STUB_PInvoke	180 (0.05%)	0 (0.00%)	shrodinger.exe
[External Code]	179 (0.05%)	179 (0.05%)	Multiple modules
[External Call] af.dll!0x007fffe148f117	179 (0.05%)	179 (0.05%)	af.dll
[External Code]	174 (0.05%)	174 (0.05%)	Multiple modules
[Broken]	166 (0.05%)	0 (0.00%)	Multiple modules
[External Code] 0x000187b4bc08b8	165 (0.05%)	165 (0.05%)	Multiple modules
dynamicClass::IL_STUB_PInvoke	132 (0.04%)	0 (0.00%)	shrodinger.exe
[External Code] 0x000187b4bc08e4	122 (0.03%)	122 (0.03%)	Multiple modules
[External Code] 0x000187b4bc089d	118 (0.03%)	118 (0.03%)	Multiple modules
[External Code] 0x000187b4bc08d1	115 (0.03%)	115 (0.03%)	Multiple modules
[External Code]	113 (0.03%)	113 (0.03%)	Multiple modules
[External Code]	111 (0.03%)	111 (0.03%)	Multiple modules
[External Code] 0x000187b4bc08a9	102 (0.03%)	102 (0.03%)	Multiple modules
[External Code] 0x000187b4bc08c5	102 (0.03%)	102 (0.03%)	Multiple modules
[External Call] af.dll!0x007fffe1607ff4	78 (0.02%)	78 (0.02%)	af.dll

Рисунок.16



Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
shrodinger.exe (PID: 14420)	11055 (100.00%)	0 (0.00%)	shrodinger.exe
shrodinger.gpu::Main	10078 (91.16%)	951 (8.60%)	shrodinger.exe
[External Call] System.Random.InternalSample()$##6001...	2415 (21.85%)	2415 (21.85%)	mscorlib.ni.dll
dynamicClass::IL_STUB_PInvoke	1349 (12.20%)	0 (0.00%)	shrodinger.exe
[External Call] af.dll!0x007fffea3be36f	1048 (9.48%)	1048 (9.48%)	af.dll
[External Code]	799 (7.23%)	799 (7.23%)	Multiple modules
dynamicClass::IL_STUB_PInvoke	723 (6.54%)	0 (0.00%)	shrodinger.exe
dynamicClass::IL_STUB_PInvoke	715 (6.47%)	0 (0.00%)	shrodinger.exe
[External Code] 0x00011bd75b08e1	607 (5.49%)	607 (5.49%)	Multiple modules
[External Code]	438 (3.96%)	438 (3.96%)	Multiple modules
[External Code] 0x00011bd75b08df	438 (3.96%)	438 (3.96%)	Multiple modules
[External Call] af.dll!0x007fffea3bdb2b	429 (3.88%)	429 (3.88%)	af.dll
[External Call] af.dll!0x007fffea55ea82	427 (3.86%)	427 (3.86%)	af.dll
dynamicClass::IL_STUB_PInvoke	427 (3.86%)	0 (0.00%)	shrodinger.exe
[External Call] af.dll!0x007fffea448a7d	391 (3.54%)	391 (3.54%)	af.dll
[External Call] af.dll!0x007fffea58be14	366 (3.31%)	366 (3.31%)	af.dll
[External Call] af.dll!0x007fffea587ff4	349 (3.16%)	349 (3.16%)	af.dll
dynamicClass::IL_STUB_PInvoke	218 (1.97%)	0 (0.00%)	shrodinger.exe
[External Code]	214 (1.94%)	214 (1.94%)	Multiple modules
[Broken]	162 (1.47%)	0 (0.00%)	Multiple modules
[External Code] 0x00011bd75b08e3	147 (1.33%)	147 (1.33%)	Multiple modules
[External Code] 0x00011bd75b08ce	73 (0.66%)	73 (0.66%)	Multiple modules
[External Code] 0x00011bd75b0890	58 (0.52%)	58 (0.52%)	Multiple modules
[External Code] 0x007ff7b4c0a020	58 (0.52%)	58 (0.52%)	Multiple modules
[External Call] af.dll!0x007fffea40f117	56 (0.51%)	56 (0.51%)	af.dll
dynamicClass::IL_STUB_PInvoke	56 (0.51%)	0 (0.00%)	shrodinger.exe
[External Code]	55 (0.50%)	55 (0.50%)	Multiple modules
[External Code] 0x00011bd75b08d5	55 (0.50%)	55 (0.50%)	Multiple modules
[External Code] 0x00011bd75b089a	53 (0.48%)	53 (0.48%)	Multiple modules
[External Code]	48 (0.43%)	48 (0.43%)	Multiple modules
[External Code]	45 (0.41%)	45 (0.41%)	Multiple modules
[External Code] 0x00011bd75b08e4	44 (0.40%)	44 (0.40%)	Multiple modules
[External Code]	43 (0.39%)	43 (0.39%)	Multiple modules
[External Code] 0x00011bd75b08c5	42 (0.38%)	42 (0.38%)	Multiple modules
[External Code] 0x00011bd75b089d	40 (0.36%)	40 (0.36%)	Multiple modules
[External Code]	35 (0.32%)	35 (0.32%)	Multiple modules
[External Code] 0x00011bd75b08b8	32 (0.29%)	32 (0.29%)	Multiple modules
[External Code] 0x00011bd75b08af	31 (0.28%)	31 (0.28%)	Multiple modules
[External Call] System.IO.File.InternalWriteAllText(Syste...	31 (0.28%)	31 (0.28%)	mscorlib.ni.dll

Рисунок.17



Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
shrodinger.exe (PID: 12836)	10283 (100.00%)	0 (0.00%)	shrodinger.exe
shrodinger.gpu::Main	9406 (91.47%)	901 (8.76%)	shrodinger.exe
[External Call] System.Random.InternalSample()$##6001...	2220 (21.59%)	2220 (21.59%)	mscorlib.ni.dll
dynamicClass::IL_STUB_PInvoke	1685 (16.39%)	0 (0.00%)	shrodinger.exe
[External Call] af.dll!0x007fffea53e36f	1359 (13.22%)	1359 (13.22%)	af.dll
[External Code]	808 (7.86%)	808 (7.86%)	Multiple modules
dynamicClass::IL_STUB_PInvoke	696 (6.77%)	0 (0.00%)	shrodinger.exe
dynamicClass::IL_STUB_PInvoke	692 (6.73%)	0 (0.00%)	shrodinger.exe
[External Code] 0x0001e8de6c08e1	486 (4.73%)	486 (4.73%)	Multiple modules
[External Call] af.dll!0x007fffea53db2b	423 (4.11%)	423 (4.11%)	af.dll
[External Code] 0x0001e8de6c08df	407 (3.96%)	407 (3.96%)	Multiple modules
[External Code]	404 (3.93%)	404 (3.93%)	Multiple modules
[External Call] af.dll!0x007fffea5c8a7d	383 (3.72%)	383 (3.72%)	af.dll
[External Call] af.dll!0x007fffea70be14	353 (3.43%)	353 (3.43%)	af.dll
[External Call] af.dll!0x007fffea707ff4	343 (3.34%)	343 (3.34%)	af.dll
dynamicClass::IL_STUB_PInvoke	222 (2.16%)	0 (0.00%)	shrodinger.exe
[External Code]	197 (1.92%)	197 (1.92%)	Multiple modules
[External Code] 0x0001e8de6c08e3	110 (1.07%)	110 (1.07%)	Multiple modules
[Broken]	107 (1.04%)	0 (0.00%)	Multiple modules
[External Code] 0x0001e8de6c08ce	72 (0.70%)	72 (0.70%)	Multiple modules
[External Code] 0x007ff7b4bfa020	56 (0.54%)	56 (0.54%)	Multiple modules
[External Code]	48 (0.47%)	48 (0.47%)	Multiple modules
[External Code]	48 (0.47%)	48 (0.47%)	Multiple modules
[External Code] 0x0001e8de6c0890	48 (0.47%)	48 (0.47%)	Multiple modules
[External Code] 0x0001e8de6c08d5	47 (0.46%)	47 (0.46%)	Multiple modules
[External Code]	46 (0.45%)	46 (0.45%)	Multiple modules
[External Code] 0x0001e8de6c089a	43 (0.42%)	43 (0.42%)	Multiple modules
[External Code]	42 (0.41%)	42 (0.41%)	Multiple modules
[External Code] 0x0001e8de6c08b8	39 (0.38%)	39 (0.38%)	Multiple modules
[External Code] 0x0001e8de6c08e4	37 (0.36%)	37 (0.36%)	Multiple modules
[External Code]	35 (0.34%)	35 (0.34%)	Multiple modules
[External Code] 0x0001e8de6c089d	32 (0.31%)	32 (0.31%)	Multiple modules
[External Call] af.dll!0x007fffea58f117	29 (0.28%)	29 (0.28%)	af.dll
[External Call] System.IO.File.InternalWriteAllText(Syste...	29 (0.28%)	29 (0.28%)	mscorlib.ni.dll
dynamicClass::IL_STUB_PInvoke	29 (0.28%)	0 (0.00%)	shrodinger.exe
[External Code] 0x0001e8de6c08a2	28 (0.27%)	28 (0.27%)	Multiple modules
[External Code]	27 (0.26%)	27 (0.26%)	Multiple modules
[External Code] 0x0001e8de6c08af	27 (0.26%)	27 (0.26%)	Multiple modules
[External Code]	27 (0.26%)	23 (0.22%)	Multiple modules

Рисунок.18

```
                19        static void Main(string[] args)
                20        {
530 (0.61%)     21            Device.SetBackend(Backend.CPU);
  2 (0.00%)     22            Device.PrintInfo();
                23            Console.WriteLine("w\th\td\tms (avg on 10 runs)\tms (min on 10 runs)\tms (max on 10 runs)");
                24            var staps = new[] {10, 32, 64, 100, 128, 200, 256, 300};
                25            for (int i = 0; i < staps.Length; i++) {
                26                var stats = new List<double>();
                27                var width  = staps[i];
                28                var height = staps[i];
                29                var depth  = staps[i];
                30                Random r = new Random();
                31                for (int j = 0; j < 10; j++) {
  8 (0.01%)     32                    var src = new Complex[width,height,depth];
  1 (0.00%)     33                    for (int i1 = 0; i1 < src.GetLength(0); i1++) {
 13 (0.01%)     34                        for (int i2 = 0; i2 < src.GetLength(1); i2++) {
6458 (7.39%)    35                            for (int i3 = 0; i3 < src.GetLength(2); i3++) {
4034 (4.62%)    36                                src[i1,i2,i3] = new Complex(r.NextDouble(), r.NextDouble());
                37                                //src[i1,i2,i3] = (float)r.NextDouble();
                38                            }
                39                        }
                40                    }
  6 (0.01%)     41                    var arr1     = Data.RandUniform<float>(width, height, depth);
  2 (0.00%)     42                    var arr2     = Data.RandUniform<float>(width, height, depth);
1859 (2.13%)    43                    var carr = Data.CreateArray(src);// Data.CreateComplexArray(arr1, arr2);
  3 (0.00%)     44                    var carr1 = Data.CreateComplexArray(arr1, arr2);
                45                    var carr2 = Data.CreateComplexArray(arr1, arr2);
                46                    var carr3 = Data.CreateComplexArray(arr1, arr2);
                47                    var carr4 = Data.CreateComplexArray(arr1, arr2);
                48                    var carr5 = Data.CreateComplexArray(arr1, arr2);
                49
```

Рисунок.19

```
                24            var staps = new[] {10, 32, 64, 100, 128, 200, 256, 300};
                25            for (int i = 0; i < staps.Length; i++) {
                26                var stats = new List<double>();
                27                var width  = staps[i];
                28                var height = staps[i];
                29                var depth  = staps[i];
                30                Random r = new Random();
                31                for (int j = 0; j < 10; j++) {
    7 (0.06%)   32                    var src = new Complex[width,height,depth];
                33                    for (int i1 = 0; i1 < src.GetLength(0); i1++) {
   11 (0.10%)   34                        for (int i2 = 0; i2 < src.GetLength(1); i2++) {
 3759 (34.11%)  35                            for (int i3 = 0; i3 < src.GetLength(2); i3++) {
 2270 (20.60%)  36                                src[i1,i2,i3] = new Complex(r.NextDouble(), r.NextDouble());
                37                                //src[i1,i2,i3] = (float)r.NextDouble();
                38                            }
                39                        }
                40                    }
  415 (3.77%)   41                    var arr1     = Data.RandUniform<float>(width, height, depth);
                42
                43
    7 (0.06%)   44                    var arr2     = Data.RandUniform<float>(width, height, depth);
  727 (6.60%)   45                    var carr = Data.CreateArray(src);// Data.CreateComplexArray(arr1, arr2);
    1 (0.01%)   46                    var carr1 = Data.CreateComplexArray(arr1, arr2);
                47                    var carr2 = Data.CreateComplexArray(arr1, arr2);
    1 (0.01%)   48                    var carr3 = Data.CreateComplexArray(arr1, arr2);
    1 (0.01%)   49                    var carr4 = Data.CreateComplexArray(arr1, arr2);
                50                    var carr5 = Data.CreateComplexArray(arr1, arr2);
                51
  641 (5.82%)   52                    var outArr0 =Data.GetData3D<Complex>(carr);
  232 (2.11%)   53                    var outArr2 =Data.GetData3D<float>(arr1);
                54
```

Рисунок.20

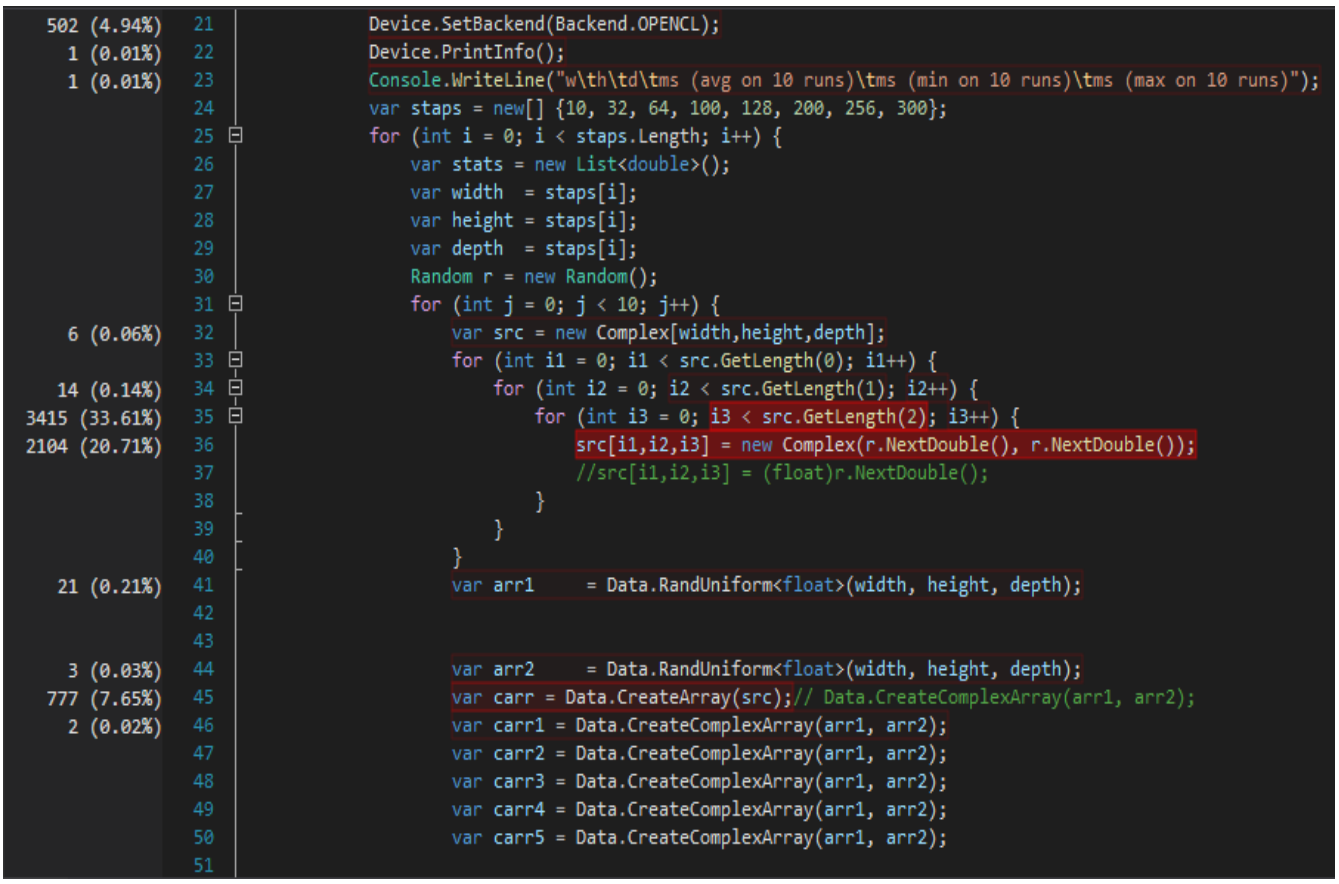

502 (4.94%) 21 Device.SetBackend(Backend.OPENCL);
1 (0.01%) 22 Device.PrintInfo();
1 (0.01%) 23 Console.WriteLine("w\th\td\tms (avg on 10 runs)\tms (min on 10 runs)\tms (max on 10 runs)");
24 var staps = new[] {10, 32, 64, 100, 128, 200, 256, 300};
25 for (int i = 0; i < staps.Length; i++) {
26 var stats = new List<double>();
27 var width = staps[i];
28 var height = staps[i];
29 var depth = staps[i];
30 Random r = new Random();
31 for (int j = 0; j < 10; j++) {
6 (0.06%) 32 var src = new Complex[width,height,depth];
33 for (int i1 = 0; i1 < src.GetLength(0); i1++) {
14 (0.14%) 34 for (int i2 = 0; i2 < src.GetLength(1); i2++) {
3415 (33.61%) 35 for (int i3 = 0; i3 < src.GetLength(2); i3++) {
2104 (20.71%) 36 src[i1,i2,i3] = new Complex(r.NextDouble(), r.NextDouble());
37 //src[i1,i2,i3] = (float)r.NextDouble();
38 }
39 }
40 }
21 (0.21%) 41 var arr1 = Data.RandUniform<float>(width, height, depth);
42
43
3 (0.03%) 44 var arr2 = Data.RandUniform<float>(width, height, depth);
777 (7.65%) 45 var carr = Data.CreateArray(src);// Data.CreateComplexArray(arr1, arr2);
2 (0.02%) 46 var carr1 = Data.CreateComplexArray(arr1, arr2);
47 var carr2 = Data.CreateComplexArray(arr1, arr2);
48 var carr3 = Data.CreateComplexArray(arr1, arr2);
49 var carr4 = Data.CreateComplexArray(arr1, arr2);
50 var carr5 = Data.CreateComplexArray(arr1, arr2);
51

Рисунок.21

Выводы

Все следующие задачи были реализованы:

1. Изучить алгоритм несжимаемого потока Шредингера и оптимизировать его с помощью объекто – ориентированного языка программирования C#.
2. Найти теоретическое обоснование добавления параметра вязкости вещества в существующую модель.
3. Перенести полученный код на GPU с использованием библиотеки ускоренных вычислений ArrayFire, которая позволяет запускать приложения на CUDA или OpenCL для параллельных вычислений.
4. Провести ряд тестов на производительность полученного кода.

Список литературы

1. Chern A., Knöppel F., Pinkall U., Schröder P., Weißmann S. Schrödinger’s Smoke // SIGGRAPH 2016, 2016.
2. Oleg Iakushkin a, Anastasia Iashnikova, Olga Sedova. GPGPU IMPLEMENTATION OF SCHRÖDINGER'S SMOKE FOR UNITY3D // Proceedings of the VIII International Conference "Distributed Computing and Grid–technologies in Science and Education" (GRID 2018), Dubna, Moscow region, Russia, September 10 – 14, 2018. С 467–469.
3. Albert Chern, Fluid Dynamics with Incompressible Schrödinger Flow, CALIFORNIA INSTITUTE OF TECHNOLOGY, Pasadena, California, 2017.
4. Проблемы гидродинамики и их математические модели. Лаврентьев М. А., Шабат Б. В., Главная редакция физико–математической литературы изд–ва «Наука», 1973 г.
5. Уравнения Навье–Стокса Существование и метод поиска глобального решения Mathematics in Computer Comp., Израиль, 2010, 106 с.
6. Темам Р. Уравнения Навье — Стокса. Теория и численный анализ. 2–е изд. М.: Мир, 1981. 408 с.
7. Ландау Л.Д., Лифшиц Е.М. Теоретическая физика. 3–е, переработанное–е изд. Т. VI. Москва: Главная редакция физико–математической литературы, 1986. 736 с.
8. Есьман Р. И., Шуб Л. И. Математическая модель движущихся теплоносителей // Энергетика. Известия высших учебных заведений и энергетических объединений СНГ. 2010, С.53–59
9. ArrayFire. https://arrayfire.com/

10. MATLAB GPU Computing Support for NVIDIA CUDA–Enabled GPUs. https://www.mathworks.com/solutions/gpu–computing.html
11. High–Performance GPU Computing in the Julia Programming Language. https://devblogs.nvidia.com/gpu–computing–julia–programming–language/
12. Numerical simulation of the transient flow behavior in tube bundles using a volume penalization method. http://wavelets.ens.fr/PUBLICATIONS/ARTICLES/PDF/200.pdf
13. ArrayFire. https://gpuopen.com/compute–product/arrayfire/
14. CUDA. https://developer.nvidia.com/cuda-zone
15. OpenCL. https://developer.nvidia.com/opencl