Санкт-Петербургский Государственный Университет

Кафедра Теории Систем Управления Электрофизической Аппаратурой

Герасименко Вячеслав Викторович

Магистерская диссертация

“Минимизация использования системных ресурсов при численном моделировании динамики пучков заряженных частиц”

010900

Прикладная математика и физика

Математические и информационные технологии

Научный руководитель,

доктор физ-мат. наук,

профессор

Дривотин О.И.

Санкт-Петербург

2016 г.

Abstract

Master's thesis: 74 pages, 12 figures, 11 tables, 60 source, 1 appendix.
Keywords: Math modeling, Beam of charged particles, Minimization of use computing resources, parallel technologies.

The work deals with minimizing of use of system computing resources in the numerical modeling of the dynamics of charged particle beams by using the following optimization tools:

- specialized math libraries;
- CPU optimization tools: OpenMP and OpenMPI;
- GPU optimization tools: CUDA and OpenACC.

The paper provides an analysis and assessment of the possibilities of optimization modeling of the beam of charged particles; it provides an overview of actual material on modeling the dynamics of the particle beam and using optimization tools to optimize the use of computing resources.

According to the equations describing the dynamics of a beam of charged particles we made calculation algorithms, by which developed software that preforms the simulation of the dynamics of the particle beam, after that we use CPU and GPU optimization tools to optimize it. As a result of optimization could improve the performance of computing in 1–5 times.

In the conclusions about the most effective ways possible to optimize applicable to the numerical simulation of a charged particle beam.

Аннотация

Магистерская диссертация: 74 страницы, 12 рисунков, 11 таблиц, 60 источников, 1 приложение

Математическое моделирование, динамика пучка заряженных частиц, минимизация использования вычислительных ресурсов, параллельные технологии.

Рассматриваются вопросы минимизации использования системных ресурсов при численном моделировании динамики пучков заряженных частиц с помощью следующих средств оптимизации:

- специализированных математических библиотек;
- технологий оптимизации CPU: OpenMP и OpenMPI;
- технологий оптимизации GPU: CUDA и OpenACC.

Приводится анализ и оценка возможностей оптимизации моделирования динамики пучка заряженных частиц; приводится обзор актуального материала по моделированию динамики пучка частиц и применению средств оптимизации вычислений.

По уравнениям описывающим динамику пучка заряженных частиц составлены алгоритмы вычисления, по которым разработано программное обеспечение, выполняющее моделирование динамики пучка частиц, произведена его оптимизация с помощью средств GPU и CPU. В результате оптимизации удалось улучшить производительность вычислений в 1–5 раз.

Сделаны выводы о наиболее эффективных возможных способах оптимизации применимых к численному моделированию динамики пучка заряженных частиц.

СОДЕРЖАНИЕ

Введение

Задача моделирования динамики пучка заряженных частиц относится к граничной области знаний физики и математики: без понимания физических основ процесса невозможно построить верную математическую модель и провести численный эксперимент, а без знания математического аппарата невозможно применять полученные математические модели. Современная физика не стоит на месте: где вчера можно было производить только теоретические исследования сегодня с помощью математических моделей производятся численные эксперименты, достоверно отражающие физику процессов моделируемых задач. Такие возможности появились не только благодаря растущему с каждым днём технологическому прогрессу, но и благодаря разработке новых, более точных научных теорий и появлению новых подходов к решению задач численного моделирования. Решение этих задач приводит к необходимости использования нестандартных методов вычисления для получения точных результатов, допускающих их проверку, что в свою очередь приводит к необходимости применять различные методы оптимизации для улучшения результатов вычисления с точки зрения эффективности.

Тема моделирования и оптимизации пучка частиц в ускорителях частиц является ведущим научным направлением кафедры ТСУЭФА Санкт-Петербургского Государственного Университета, по которому и в настоящее время продолжается активная работа под руководством Овсянникова Дмитрия Александровича [1–13]. Большой вклад в данном направлении внёс также и научный руководитель автора – Дривотин Олег Игоревич, в его работах [14–21], применяется доступный и развёрнутый подход к описания динамики пучка частиц. Актуальные результаты исследований по теме ускорителей частиц и особенностей построения математических моделей пучка частиц можно отследить в монографиях [22–32]. Существенным недостатком этих работ по мнению автора является недостаточное освящение вопроса связанного с эффективным использованием системных вычислительных ресурсов в процессе моделирования, что связано с другой направленностью этих работ. Этот факт является позитивным знаком и свидетельствует о том, что задачи минимизации использования

системных ресурсов при моделировании динамики частиц – востребованная и актуальная тема исследований, способная найти прикладное практическое применение.

Помимо публикаций, связанных непосредственно с моделированием, в процессе работы использованы материалы по архитектуре программного обеспечения [33–35], параллельным технологиям [36–40], оптимизации вычислений [41,42], а также общие материалы, описывающие применяемый математический аппарат [43–47].

В работе рассматривается модель динамики пучка частиц в ускоряющем тракте с пространственно однородной квадрупольной фокусировкой (далее по тексту – ПОКФ). ПОКФ открыли Российские учёные: В.В. Владимирский, И.М. Капчинский и В.А.Тепляков 25 октября 1968 г., а 25 марта 1969 г. экспериментально подтвердили разработанную теорию. Им удалось установить, что в переменном электрическом поле квадрупольный фокусирующий эффект возникает в случае, когда поле обладает пространственно однородной структурой вдоль продольной оси движения пучка. Суть явления заключается в том, что в поле (переменном во времени), заряженные частицы при движении вдоль продольной оси симметрии структуры периодически попадают в пространственные области сил с переменным направлением, за счёт чего возникает переменный фокусирующий/дефокусирующий эффект.

Главное значение открытия заключается в том, что использование ПОКФ привело к созданию нового поколения ускорителей частиц (в ускорителях до открытия ПОКФ использовалась преимущественно пространственно-неоднородная фокусировка). За счёт ПОКФ достигается снижение энергии инжекции на порядок и более, что позволяет производить практически полный захват частиц в режим ускорения (порядка $95-97\%$) и обеспечивать высокое значение предельного тока пучка частиц. К преимуществам таких ускорителей можно отнести и то, что они могут работать при весьма низкой начальной энергии частиц (порядка 60 – 100 кэВ).

До 70-х годов в линейных ускорителях и каналах транспортировки применялась фокусировка частиц со знакопеременной пространственно периодической структурой, состоящей из статических квадрупольных линз (пространственно-неоднородная фокусировка), но она является менее

эффективной по сравнению с ПОКФ. Практическая разработка ускорителей частиц с ПОКФ началась в СССР в 1970 году, практически сразу после экспериментального подтверждения, за рубежом же разработка такого рода ускорителей широко развернулась лишь с 1979 года. В настоящее время эффект ПОКФ применяется в инжекторах протонных и тяжёло-ионных синхротронных ускорителей. Использование ПОКФ в ускорителях позволяет получать сильноточные пучки ионов, используемых в новых технологиях, таких как:

— производство высокопоточных нейтронных генераторов использующихся в радиационном материаловедении (проблемы термоядерных реакторов);

— формирование сильноточных пучков протонов для наработки ядерного топлива (переработка ядерных отходов АЭС);

— создание линейных ускорителей сверхтяжёлых малозарядных ионов (ионный термоядерный синтез);

— глубокое исследование человека без хирургического вмешательства (ПЭТ и другие виды томографии и рентгенографии);

— разработка малогабаритных генераторов мощных атомных пучков.

Задача работы заключается в анализе и разработке возможных методов минимизации использования системных ресурсов при численном моделировании динамики пучка заряженных частиц. Оптимизации вычислений можно достичь с помощью:

- аналитических методов, которые заключаются в сокращении входных данных и упрощении уравнений математических моделей. Упрощение математической модели даже на несколько элементарных математических операций может дать ощутимый прирост в производительности вычислений;

- программных методов, которые заключаются в использования специфических конструкций языка, на котором реализуется математическая модель:

  - использование специальных алгоритмов, характерных для языка исполнения может привести к ускорению процесса вычисления (например, транспонирование матриц в языке *C*), использование inline-функций, задание констант вместо переменных, объединение объёмов обрабатываемых данных в кратные размеру кластера файловой системы или оперативной памяти массивы;
  - использование машинных инструкций: в работе были использованы sse-инструкции для оптимизации “однотипного произведения переменных” [41] в процессе интегрирования правых частей дифференциального уравнения.

- с помощью средств, предназначенных для оптимизации вычислений:
  - с помощью оптимизированных математических библиотек. В работе рассматриваются способы оптимизации с помощью библиотеки BLAS [48];
  - с помощью центрального процессора (далее по тексту – CPU). В работе рассматриваются способы оптимизации с помощью OpenMP [38] и Open MPI [40];
  - с помощью графического видео-ускорителя (далее по тексту – GPU). В работе рассматриваются способы оптимизации с помощью CUDA [39] и OpenACC [49].

Эффективность оптимизации зависит от целевой системы, на которой применяется, а также способа, к которому применяется. С учётом того, что тестовый стенд, развёрнутый для проведения испытаний не обладает большим количеством CPU, очевидно, что многопроцессорная оптимизация будет иметь худшие показатели результата (но может показать себя лучше на специализированных кластерных стендах). Также стоит сказать про популярные в научной и профессиональной среде средства с помощью которых возможно провести моделирование динамики пучка частиц – Matlab [50] и CST PS [51]. Данные программные комплексы при всех своих преимуществах не могут быть оптимизированы с точки зрения снижения потребления системных ресурсов и в работе не рассматриваются.

Для разработки вычислительных модулей используется язык программирования C в силу простоты, гибкости и широких возможностей применения (возможность управления памятью и файловой системой на низком уровне, возможность использования процессорных инструкций и т.п.), что позволяет более эффективно распределить ресурсы системы в процессе вычислений.

Помимо программных существуют аппаратные и аналитические решения минимизации используемых системных ресурсов, в работе они рассматриваются в меньшей степени, т.к. тематика минимизации с аппаратной или аналитической точки зрения являются темами, заслуживающими отдельного, более широкого исследования.

1 Численное описание динамики пучка заряженных частиц

Глава посвящена анализу моделей динамики пучка заряженных частиц в канале ускорения. В главе приводятся основные математические выкладки используемые в процессе моделирования динамики пучка заряженных частиц в канале ускорения; приведены уравнения продольной и поперечной динамики частиц; приведён способ задания начально распределения частиц.

1.1 Общие сведения об ускорителях частиц

Ускорители частиц (рисунок 1.1) – специальные установки, производящие ускорение элементарных частиц до скоростей высоких энергий [26], конструктивно состоящие из трёх частей:

— инжектора – системы, генерирующей частицы для ускорения;

— ускоряющего тракта – системы, производящей ускорение частиц (в особой среде, обычно ускоряются пучки частиц) полученных от инжектора;

— выводящего тракта – системы, производящей вывод рабочих частиц на мишень.

С точки зрения траектории по которой ускоряются частицы ускорители можно разделить на два класса – линейные и циклические. В линейных ускорителях частицы в процессе ускорения двигаются прямолинейно, а в циклических – либо по одной и той же замкнутой траектории, многократно проходя одни и те же ускоряющие промежутки (синхротроны), либо по спиралевидной траектории (циклотроны, микротроны, фазотроны).

1.2 Квадрупольная фокусировка пучков частиц

Существуют два типа ускоряющих структур, с помощью которых может быть выполнена квадрупольная фокусировка частиц:

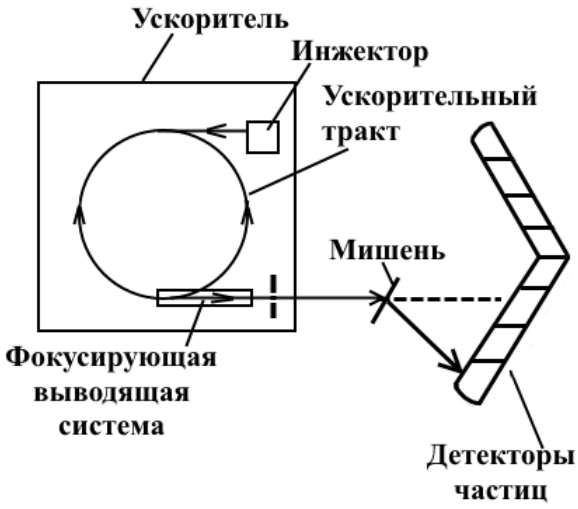

Рисунок 1.1 – Схема ускорителя частиц.

— структура квадрупольных линз с выраженной пространственной периодичностью (рисунок 1.2). Поля в таких структурах постоянны во времени, но пространственно неоднородны. Такой способ ускорения применяется в ускорителях на трубках дрейфа [23];

— структура в виде четырёхпроводной линии с квадрупольной симметрией (рисунок 1.3). Исполняется в виде системы электродов расположенной вдоль оси ускорителя с явно выраженной квадрупольной симметрией. Поле в такой структуре пространственно однородно и периодически изменяется во времени за счёт периодического изменения напряжения на электродах. В работе основное внимание уделено именно этому способу фокусировки пучка частиц.

Фокусирующие элементы четырёхпроводной квадрупольной структуры расположены непрерывно вдоль линии ускорителя, соответственно частицы взаимодействуют с создаваемым фокусирующим полем непрерывно. Электрическое поле такой структуры обладает квадрупольной симметрией, которая достигается за счёт приложения к электродам переменного высокочастотного напряжения $(\pm U_0 \cos(\omega t))$, таким образом, чтобы на смежных электродах было приложено напряжение противоположных знаков. Частицы пучка, при прохождении вдоль оси ускоряющей структуры (внутри линии) испытывают на себе действие электрических полей с чередующимися знаками поля, из-за чего происходит фокусировка частиц.

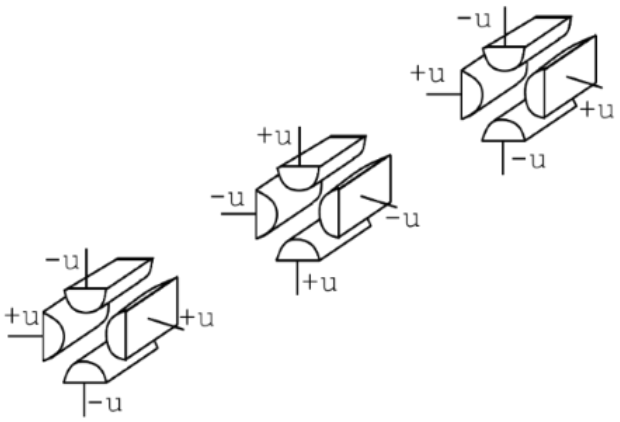

Рисунок 1.2 – Пространственно-неоднородная квадрупольно-фокусирующая структура.

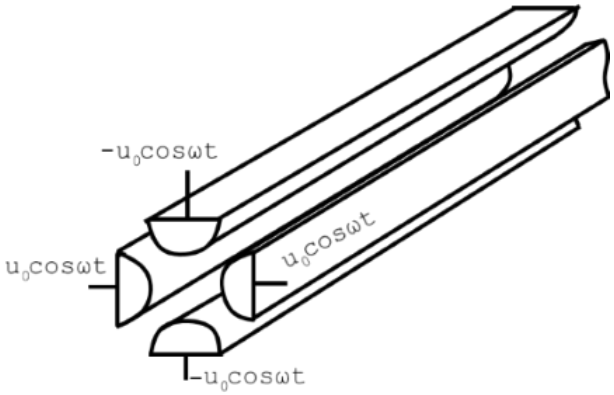

Рисунок 1.3 – Пространственно-однородная квадрупольно-фокусирующая структура.

Преимущество такой фокусировки заключается в том, что она более эффективна по сравнению с пространственно неоднородной, т.к. чем большую часть периода фокусировки занимают электроды с квадрупольной симметрией, тем эффективнее производится фокусировка [52]. Чтобы понять каким образом происходит фокусировка заряженных частиц рассмотрим электрическое поле в поперечном сечении четырёхпроводной линии в виде силовых линий создаваемых приложенным напряжением (рисунок 1.4).

Пусть положительно заряженные частицы движутся рядом с осью Oz, направленной вдоль линии ускорителя. Допустим что на рассматриваемом полупериоде поля ($\cos(\omega t) > 0$) на частицы в плоскости xOz действуют фокусирующие силы, направленные к оси, а в плоскости yOz соответственно действуют дефокусирующие силы, тогда на следующем полупериоде поля ($\cos(\omega t) < 0$) силы будут действовать наоборот – в

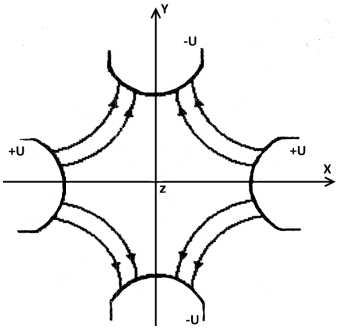

Рисунок 1.4 – Силовые линии в поперечном сечении четырёхпроводной линии.

плоскости xOz дефокусирующие силы, а в плоскости yOz – фокусирующие. Таким образом, при движении частиц вдоль линии ускорителя (оси Oz) на частицы действуют фокусирующие и дефокусирующие силы в каждой из плоскостей. Как правило линии с квадрупольной фокусировкой состоят из большого числа последовательно чередующихся фокусирующих и дефокусирующих электродов, в соответствии с которыми модулируется траектория частицы – на фокусирующих участках частица в среднем удалена от оси распространения больше, чем на смежных, дефокусирующих участках.

Потенциал электрического поля на оси Oz модулирован с периодом, который медленно меняется вдоль оси, за счёт чего возникает резонансный ускоряющий эффект, т.е. длина периода модуляции электродов определяется как:

$$L_i = \beta\lambda = v_s T = \frac{2\pi}{\omega} v_s, \tag{1.1}$$

где $\beta = \frac{v_S}{c}$ – приведённая продольная скорость заряженной частицы.

Скорость синхронной частицы v_S при прохождении ячеек структуры возрастает незначительно, поэтому в упрощённых моделях можно полагать что скорость постоянна на всём пути своего следования (1.1).

Пространственный период модуляции электродов является очень важным параметром, потому что он явно характеризует структуру ускоряющей линии (характер модуляции её электродов). Сегмент четырёхпроводной ускоряющей линии соответствующий периоду модуляции называют *пространственным периодом* структуры или *ячейкой*.

Для возникновения продольной ускоряющей компоненты электрического поля необходимо чтобы вдоль оси ускоряющей структуры расстояние между противоположными электродами периодически изменялось, т.е. необходимо чтобы пространственный период изменения расстояния между электродами был равен пути, который проходит равновесная частица за период, а фазы изменения расстояний в перпендикулярных плоскостях должны быть сдвинутыми на половину периода. В сечениях плоскостями вдоль оси структуры oZ:

— в плоскостях $z = nL_i, \quad n = 0, \pm1, \pm2, \ldots$ – горизонтальные электроды минимально удалены от оси структуры, в то время как вертикальные расположены на максимальном расстоянии от оси структуры;

— в плоскостях $z = L_i/4 + nL_i/2, \quad n = 0, \pm1, \pm2, \ldots$ – горизонтальные и вертикальные электроды располагаются на одинаковом расстоянии от оси структуры (точная квадрупольная симметрия);

— в плоскостях $z = L_i/2 + nL_i, \quad n = 0, \pm1, \pm2, \ldots$ – горизонтальные электроды максимально удалены от оси структуры, вертикальные расположены на минимальном расстоянии от оси структуры.

В результате можно сделать вывод, что в таких структурах расстояние между противоположными электродами одинаковой полярности периодически изменяется вдоль оси, за счёт чего появляется продольная компонента электрического поля. При модуляции формы электродов и заданном их минимальном расстоянии от оси сила фокусировки снижается примерно на $40 - 50\%$, что обусловлено влиянием высокочастотного дефокусирующего эффекта [25].

В реальных ускорителях, фокусирующая структура расположенная вдоль продольной оси ускорителя квазипериодическая – параметры ячеек структуры монотонно изменяются вдоль её оси, а не постоянны, как в случае модели приближения. Т.е. длина пространственного периода L_i монотонно изменяется вдоль оси структуры ускорителя. Это свойство необходимо учитывать в модели, для того, чтобы обеспечивать синхронизацию между ускоряющим пучком, в котором

скорость синхронной частицы v_S монотонно возрастает при движении вдоль оси структуры за счёт ускоряющей бегущей волны. Кроме того, в реальных ускорителях иногда изменяют параметры ячейки (апертуру) и величину приложенного между электродами напряжения в зависимости от практических потребностей.

1.3 Уравнения, описывающие динамику пучка частиц

Для описания пучка заряженных частиц, будем использовать модель, в рамках которой система большого числа взаимодействующих частиц рассматривается как непрерывная среда, распределённная в фазовом пространстве $\Omega, (\dim \Omega = n)$ с плотностью распределения частиц $\rho\,(x)$ (далее по тексту – фазовая плотность).

Фазовая плотность представляет собой дифференциальную форму, заданную на $m-$мерной поверхности $S\,(m < n)$ в некоторой области D фазового пространства Ω или во всей области D. Степень формы равна размерности поверхности пространства m, если она задана на поверхности, или n, если плотность задана в области D фазового пространства Ω. Согласно определению фазовой плотности при ее интегрировании получаем число частиц N_D, находящихся внутри области:

$$\int\limits_{D \cap S} \rho\,(x) = N_D. \tag{1.2}$$

При этом интегрирование выполняется по пересечению области фазового пространства D с поверхностью на которой задана форма, если она задана на поверхности, или по области D, если она задана в этой области.

В случае, если степень формы плотности пространства равна размерности фазового пространства, то пучок частиц можно описать с помощью функции распределения $F(t, q, \vec{v})$, которая характеризует распределение частиц в фазовом пространстве, где q – точка в конфигурационном пространстве, $\vec{v}$ – вектор скоростей, а t – время.

Уравнение, описывающее эволюцию функции распределения пучка

частиц можно записать в виде [14, 53]:

$$\frac{d\rho}{dt} = \frac{\partial \rho}{\partial t} + \sum_{i=1}^{d} \left(\frac{\partial \rho}{\partial q_i} q_i' + \frac{\partial \rho}{\partial p_i} p_i' \right), \tag{1.3}$$

где q_i и p_i – координата и декартова компонента импульса соответственно, q_i' и p_i' – соответствующие координате и компоненте импульса производные по времени. Функция распределения (ρ) постоянна вдоль любой траектории в рассматриваемом фазовом пространстве.
Если $\vec{p}'$ – определяется только внешним воздействием, то такое уравнение называют уравнением Лиувилля, а если внутренним и внешним воздействием – уравнением Власова.

Уравнения Власова – система уравнений, описывающих динамику плазмы заряженных частиц с учётом дальнодействующих сил посредством самосогласованного поля [54, 55]. Система уравнений Власова включает в себя также уравнения Максвелла и уравнения для заряда и тока, выраженные через функции распределения f_s. Для описания взаимодействия частиц Власов предложил использовать самосогласованное поле, создаваемое заряженными частицами плазмы [55]. В результате им были составлены следующие уравнения для описания характеристик поля:

$$\frac{\partial f_e}{\partial t} + \vec{v}\frac{\partial f_e}{\partial \vec{x}} - e\left(\vec{E} + \frac{1}{c}\left[\vec{v}, \vec{B}\right]\right)\frac{\partial f_e}{\partial \vec{p}} = 0, \tag{1.4}$$

$$\frac{\partial f_i}{\partial t} + \vec{v}\frac{\partial f_i}{\partial \vec{x}} - e\left(\vec{E} + \frac{1}{c}\left[\vec{v}, \vec{B}\right]\right)\frac{\partial f_i}{\partial \vec{p}} = 0, \tag{1.5}$$

$$rot\vec{B} = \frac{4\pi\vec{j}}{c} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}, \quad rot\vec{E} = -\frac{1}{c}\frac{\partial \vec{B}}{\partial t}, \tag{1.6}$$

$$div\vec{B} = 4\pi\rho, \quad div\vec{B} = 0, \tag{1.7}$$

$$\rho = e\int (f_i - f_e)\, d^3\vec{p}, \quad \vec{j} = \int (f_i - f_e)\, \vec{v} d^3\vec{p}. \tag{1.8}$$

В уравнениях: e – заряд электрона, c – скорость света, $\vec{E}\,(\vec{r}, t)$ и $\vec{B}\,(\vec{r}, t)$ – самосогласованные электрические и магнитные поля в точке $\vec{r}$ в момент времени t заряженными частицами.

В случае движения частиц в среде фазовая плотность $\rho(x)$ также зависит от времени и определяется с помощью уравнения Лиувилля в случае пучка малой плотности (1.3), а в случае интенсивного пучка – с помощью уравнения Власова (1.4). Характеристические линии уравнения фазового распределения являются траекториями движения частиц в электромагнитном поле, создаваемом внешними по отношению к рассматриваемой среде источниками полей.

1.4 Метод крупных частиц

Рассмотрим среду с малой плотностью частиц. При моделировании такой среды применяется метод крупных частиц [15], в соответствии с которым рассматриваемая среда описывается с помощью набора конечного числа частиц (дискретное распределение), движущихся по аналогичным траекториям как и в случае непрерывного распределения (т.е. вдоль характеристический линий уравнения Лиувилля). В результате задачу управления пучком частиц с малой плотностью можно рассматривать как задачу управления ансамблем траекторий невзаимодействующих частиц.

Отдельная траектория (характеристическая линия уравнения Лиувилля) описывается дифференциальным уравнением вида:

$$\frac{d\vec{x}}{dt} = f\left(t, \vec{x}, u\right), \tag{1.9}$$

где $t \in [t_0, T]$ – параметр траектории, в качестве которого можно взять время;

$x \in \Omega$, $u \in U \subset R^r$ – управления, задаваемые в виде $r-$мерной векторной функции.

Примечание: предполагается, что вектор решений уравнения f определён в области $[t_0, T] \times \Omega \times U$, а решение дифференциального уравнения (1.9), имеющее вид $x(t_0) = x_0$ существует и единственно для любых рассматриваемых значениях x_0.

1.5 Управление пучком заряженных частиц малой плотности

Зададим распределение плотности частиц в начальный момент времени как:

$$\varrho\left(t_0, x\right) = \varrho_0\left(x\right) = \left(\varrho_0\right)_{1\ldots m}\left(x\right) dx^i \wedge \ldots \wedge dx^m, \tag{1.10}$$

где $x^i \quad \left(i = \overline{1, m}\right)$ – координаты фазового пространства на поверхности S.

Если фазовая плотность удовлетворяет уравнению Лиувилля:

$$\varrho\left(t + \delta t, F_{f,\delta t x}\right) = F_{f,\delta t \varrho(t,x)}, \tag{1.11}$$

где $F_{f,\delta t x}$ и $F_{f,\delta t \varrho(t,x)}$ – переносы Ли [43] точки x и тензора $\varrho\left(t, x\right)$ вдоль векторного поля f [43], тогда можно ввести функционал, характеризующий качество динамики частиц (качество управления) задавая неотрицательные кусочно-дифференцируемые функции вида [43]:

$$\varphi\left(t, x\right) \in C^1\left(\left[t_0, T\right] \times \Omega\right) \quad g\left(x\right) \in \overline{C^1}\left(\Omega\right), \tag{1.12}$$

в таком случае функционал будет иметь вид:

$$\Phi\left(u\right) = \int\limits_{t_0}^{T} \int\limits_{\Omega} \varphi\left(t, x\right) \varrho\left(t, x\right) + \int\limits_{\Omega} g\left(x_T\right) \varrho\left(T, x_T\right). \tag{1.13}$$

Введённый функционал имеет большое теоретическое значение, с которым связаны следующие важные определения [24]:

— задача минимизации функционала (1.13) по управлениям u из класса допустимых управлений U называют задачей программного управления пучком малой плотности с учётом плотности распределения частиц.

— управление $u_0 = u_0\left(t\right)$ характеризующее минимум функционала (1.13) называют оптимальным управлением функционала.

— в случае задания функционала только на выходном сечении пучка (при $\varphi \equiv 0$) задачу управления называют задачей

терминального управления пучком малой плотности с учётом плотности распределения частиц.

– вариацию $\Delta u\,(t)$ называют допустимой, если управление $u + \Delta u\,(t)$ удовлетворяет условию: $u + \Delta u\,(t) \in U$.

1.6 Вычисление градиента функционала

Градиент – это характеристика, показывающая направление и величину скорости изменения функции в отдельной точке. Соответственно вычисление градиента функционала позволяет определить характер изменения функционала и его скорость, на основании чего могут быть выбраны такие управляющие параметры системы, при которых значение функционала (и его решение) будут оптимальными.

Рассмотрим уравнение удовлетворяющее дифференциальному уравнению (1.9) для вариации x. При вариации управления δu имеем:

$$\frac{d\delta x^i}{dt} = \frac{\partial f^i}{dx^j}\delta x^j + \delta_u f^i, \tag{1.14}$$

где $\delta x^i(t_0) = 0, \quad i = \overline{1, m}$;
$\delta_u f^j = \frac{\delta f^j}{\delta u^k}\delta u^k$.

Суммирование по повторяющимся индексам производится в соответствии с правилом Эйнштейна [43]: если один и тот же индекс в обозначении встречается и сверху и снизу, то такой член полагается просуммированное по всем значениям, которые может принимать этот индекс. Пример: $v_k = a_i b_k^i = \sum\limits_{i=1}^{n} a_i b_k^i$, где n – размерность пространства, в котором определены a и b. В случае использования для выражений с частными производными верхние индексы записываемые в знаменателе, считаются для применение нижними индексами и наоборот [43], например $\frac{\partial f}{\partial x^i}dx^i$ обозначает $\sum\limits_{i=1}^{n} \frac{\partial f}{\partial x^i}dx^i$.

Решение уравнения (1.14) имеет вид:

$$\delta x^i(t) = \int\limits_{t_0}^{t} G_j^i\left(t, t'\right) \delta_u f^j\left(t'\right) dt', \tag{1.15}$$

где $G_j^i\left(t, t'\right)$ – фундаментальная матрица (функция Грина) уравнения (1.14).

Фундаментальная матрица уравнения (1.14) удовлетворяет условиям:

$$\frac{dG_j^i\left(t, t'\right)}{dt} = \frac{\partial f^i}{\partial x^k} G_j^k\left(t, t'\right), \quad G\left(t, t\right) = E, \tag{1.16}$$

где E – единичная матрица.

С использованием полученного решения вариацию функционала (1.13), при вариации управления δu, можно записать в виде:

$$\delta_u \Phi = \int\limits_{t_0}^{T} \int\limits_{\Omega} \frac{\partial g}{\partial x} G\left(T, t'\right) \delta_u f\left(t, x\right) \varrho\left(t, x\right) dt, \tag{1.17}$$

где $\delta_u f = \frac{\partial f(t,x,u)}{\partial u} \delta u$.

Вводя дифференциальную форму удовлетворяющую уравнению:

$$\frac{d\psi}{dt} = -\psi \frac{\partial f}{\partial x}, \tag{1.18}$$

и конечным условия вида:

$$\psi\left(t, x\right) = -\frac{\partial g}{\partial x}|_{x=x_T} G\left(T, t\right), \tag{1.19}$$

вариацию функционала (1.13) можно записать как:

$$\delta_u \Phi = -\int\limits_{t_0}^{T} \int\limits_{\Omega} \psi\left(t, x\right) \delta_u f\left(t, x\right) \varrho\left(t, x\right) dt. \tag{1.20}$$

Далее полагаем что u – кусочно-постоянная вектор-функция:

$$u = u_i, \quad t \in [t_{i-1}, t_i], \quad i = \overline{1, M}, \tag{1.21}$$

где $t_M = T$.

Тогда функционал (1.13) можно рассматривать как функцию конечного числа параметров rM, а производная от функционала по этим параметрам будет иметь вид:

$$\frac{\partial \Phi}{\partial u_i^k} = -\int_{t_0}^{T} \int_{\Omega} \psi(t,x) \frac{\partial \delta_u f(t,x)}{\partial u_i^k} \varrho(t,x)\, dt, \quad i = \overline{1, M}, \quad k = \overline{1, r}. \tag{1.22}$$

Переходя от интегрирования по фазовому пространству к суммированию по отдельным частицам в соответствии с методом крупным частиц усредним выражение для градиента функционала (1.22) суммой по крупным частицам, тогда получим следующее значение для градиента функционала [43]:

$$\frac{\partial \Phi}{\partial u_i^k} = -\int_{t_0}^{T} \sum_{j=1}^{N} \psi(t,x_j) \frac{\partial \delta_u f(t,x_j)}{\partial u_i^k} dt, \quad i = \overline{1, M}, \quad k = \overline{1, r}, \tag{1.23}$$

где x_j – положение $j-$ой частицы в рассматриваемом фазовом пространстве.

1.7 Электрическое поле в квазистационарном приближении

В случае когда размеры области, в которой движутся частицы, существенно меньше такого характерного параметра поля как длина волны λ, которой характеризуется частота колебаний электромагнитного поля ω, $\lambda = 2\pi c/\omega$, говорят о том, что рассматривается **квазистационарное приближение**. В квазистационарном приближении вместо волнового уравнения рассматривается уравнение Пуассона или Лапласа:

$$\triangle u = 0. \tag{1.24}$$

Для дальнейшего анализа будем учитывать следующие допущения:

— потенциал на поверхности электродов определяется в соответствии с выражением:

$$u_p = \pm u_0 \cos(\omega t); \tag{1.25}$$

— электроды представляют собой гиперболическую поверхность:

$$r^2 \cos(2\varphi) = \mu \left(\pm 1 - \frac{4T}{\pi} I_0 (kr) \sin \eta \right), \tag{1.26}$$

где $\eta(z) = \int\limits_{Z_0}^{Z} k'(z) dz' k(z)$ – функция пространственной модуляции поверхности электродов;

I_0 – обозначает функцию Бесселя (модифицированную) нулевого порядка.

Рассмотрим i-ую ячейку структуры, в которой фаза пространственной модуляции η изменяется от $(i-1)\pi$ до $i\pi$, $\left(i = \overline{1, N}\right)$, N – число ячеек. Считая что $k(z)$ и T медленно изменяются при изменении z, длину i-ой ячейки можно определить как $L_i = \pi / k(z_i)$ (z_i – любое z внутри ячейки). Тогда медленное изменение параметра k означает, что

$$\frac{dk}{dz} << \frac{k\left(z_i\right)}{L_i} = \frac{k\left(z_i\right)^2}{\pi}, \tag{1.27}$$

с учётом которого точное решение уравнения (1.24) при условии выполнения квазистатического приближения (1.24) будет:

$$u\left(r, \varphi, z\right) = -u_0 \left(\frac{r^2 \cos\left(2\varphi\right)}{\mu} + \frac{4T}{\pi} I_0(kr) \sin(\eta) \right) \cos\left(\omega t\right). \tag{1.28}$$

При этом на границах электродов потенциал $u_{gr.} = \pm u_0$. Подставляя это

решение в уравнение (1.24) получаем:

$$\frac{1}{r}\frac{\partial}{\partial r}r\frac{\partial u}{\partial r}+\frac{1}{r^2}\frac{\partial^2 u}{\partial \varphi^2}+\frac{\partial^2 u}{\partial z^2}=\frac{2}{\mu}\cos\left(2\varphi\right)+\frac{4T}{\pi}\int\limits_{z_0}^{z}k\left(z'\right)dz'\cdot$$

$$\cdot\left(\frac{1}{r}\frac{\partial}{\partial r}r\frac{\partial I_0\left(kr\right)}{\partial r}-\frac{2}{\mu}\cos\left(2\varphi\right)+k^2\frac{4T}{\pi}I_0\left(kr\right)\left(-\int\limits_{z_0}^{z}k\left(z'\right)dz'\right)\right)=0. \tag{1.29}$$

Определим понятие апертуры канала, для чего обозначим минимальное расстояние от оси до ближайшего к оси электрода (по всем сечениям) параметром a (т.е. $a = r_{min}$), тогда апертуру канала можно определить из уравнения(1.26) как:

$$a^2=\mu\left(1-\frac{4T}{\pi}I_0\left(ka\right)\right), \tag{1.30}$$

с учётом чего решение уравнения (1.24) можно переписать виде:

$$u\left(r,\varphi,z\right)=-u_0\left(\chi\frac{r^2}{a^2}\cos\left(2\varphi\right)+\frac{4T}{\pi}I_0(kr)\sin(\eta)\right)\cos\left(\omega t\right), \tag{1.31}$$

где $\mu=\frac{a^2}{\chi}$; $\quad\chi=1-\frac{4T}{\pi}I_0\left(kr\right)$.

Расстояние от другого электрода до оси меняется противоположным образом: там, где расстояние от первого электрода до оси достигает максимума, расстояние от второго электрода до оси достигает минимума, и наоборот. При этом величины наибольших и наименьших значений совпадают. Из выражения (1.30), по аналогии с определением апертуры r_{min}, очевидно, что наибольшее расстояние от электрода до оси r_{max} будет:

$$r_{max}^2=\mu\left(1+\frac{4T}{\pi}I_0\left(kr_{max}\right)\right). \tag{1.32}$$

Используя введённые соотношения для r_{min} и r_{max} ((1.30) и (1.32) соответственно) можно ввести коэффициент модуляции, которым удобно характеризовать параметры ячейки ускоряющей структуры:

$$m=\frac{r_{min}}{r_{max}}\rightarrow m^2=\frac{1-\frac{4T}{\pi}I_0\left(kr_{max}\right)}{1+\frac{4T}{\pi}I_0\left(kr_{max}\right)}. \tag{1.33}$$

С учётом введённых коэффициентов период ячейки можно определить как:

$$T = \frac{\pi}{4}\frac{m^2 - 1}{m^2 I_0(ka) + I_0(mka)}. \tag{1.34}$$

Допуская, что в приосевой области поле характеризуется потенциалом (1.31), а также, то, что косинус двойного угла можно выразить как $\cos(2\varphi) = \cos^2(\varphi) - \sin^2(\varphi)$, выражение (1.31) можно переписать в виде:

$$u(r, \varphi, z) = -u_0 \left(\chi \frac{r^2}{a^2} + \frac{4T}{\pi} I_0(kr) \sin(\eta) \right) \cos(\omega t). \tag{1.35}$$

Амплитуду разности потенциалов в полупериод ускорения $\beta\lambda/2$ в первом приближении можно записать как:

$$U = 2\frac{\left(n^2 - 1\right)}{\left[n^2 \lambda_0(ka) + I_0 n(ka)\right]} u_0 = 2Au_0, \tag{1.36}$$

где $A = \frac{\left(n^2-1\right)}{\left[n^2\lambda_0(ka)+I_0 n(ka)\right]}$;
m – отношение максимального расстояния от оси симметрии структуры до ближайшей точки электрода к минимальному расстоянию от оси;
$k = \frac{2\pi}{\beta\lambda}$;

Учитывая, что аргумент модифицированной функции Бесселя нулевого порядка меньше 1 (т.к. $k = \frac{2\pi}{\beta\lambda}$), можно ограничиться первыми двумя членами разложения функции Бесселя по степеням аргумента, т.е. $I_0(kr) \approx 1 + \frac{kr^2}{4}$. Продифференцировав выражение (1.35) и учитывая что $r^2 = x^2 - y^2$ можно получить выражения составляющих электрического поля в декартовых координатах:

$$E_x = u_0 \left(\frac{2\chi}{a^2} x + \frac{2k^2 T}{\pi} x \sin(\eta) \right) \cos(\omega t). \tag{1.37}$$

$$E_y = u_0 \left(-\frac{2\chi}{a^2} y + \frac{2k^2 T}{\pi} y \sin(\eta) \right) \cos(\omega t). \tag{1.38}$$

$$E_z = u_0 \frac{4kT}{\pi} I_0(kr) \cos(\eta) \cos(\omega t). \tag{1.39}$$

Уравнения (1.37) – (1.39) характеризуют электрическое поле в

квазистационарном приближении.

1.8 Уравнения динамики частиц

Одно из уравнений продольной динамики можно получить, используя уравнение энергетического баланса:

$$m_e \frac{d\gamma}{ds} = \frac{e}{c^2}\left(E_x \frac{dx}{ds} + E_y \frac{dy}{ds} E_z \frac{dz}{ds}\right), \tag{1.40}$$

где $\gamma = (1-\beta)^{-1/2}$ – фактор Лоренца (приведённая энергия);
β – приведённая продольная скорость;
s – интервал вдоль траектории;
m и e – соответственно масса и электрический заряд частицы.

С учётом того, что поперечные компоненты скорости частиц (x и y компоненты) много меньше продольной компоненты (z), то энергию частицы можно рассматривать как энергию только её продольного движения. Таким образом, пренебрегая первым и вторым членами уравнения (1.40), а также вводя безразмерную относительную продольную координату $\zeta = z/\lambda$ и учитывая что:

$$\frac{d}{ds} = \beta\gamma \frac{d}{dz} = \frac{\beta\gamma}{\lambda}\frac{d}{d\zeta}, \tag{1.41}$$

тогда уравнение продольного движения можно записать как:

$$\frac{m}{\lambda}\frac{d\gamma}{d\zeta} = \frac{eE_z}{c^2}. \tag{1.42}$$

Учитывая что частицы движутся вблизи оси пучка, $(kr << 1)$, вместо функции Бесселя нулевого порядка можно рассматривать только первый член разложения её по степеням r, т.е. $I_0(kr) \approx 1$. Определим амплитуду напряжения между электродами как $U_0 = 2u_0$ (согласно формуле (1.36) $A = 1$). Введём вместо параметра k безразмерный параметр $\overline{k} = k\lambda$.
Учитывая выражение для продольной компоненты электрического поля (подставляя уравнение (1.38) в уравнение (1.42)), можно записать ***первое***

уравнение продольной динамики в виде:

$$\frac{d\gamma}{d\zeta} = \frac{2eU_0}{\pi m_e c^2}\overline{k}T\cos(\eta)\,cos(\varphi)\,, \tag{1.43}$$

где $\varphi = \omega t$ обозначает фазу движущейся частицы.

Фазу пространственной модуляции (частицы) η можно представить в виде:

$$\eta = \pi\left((i-1) + \frac{z - c_{i-1}}{L_j}\right), \quad z \in [c_{i-1}, c_i]\,, \tag{1.44}$$

то есть как кусочно-линейную функцию координаты z. Здесь $c_i = \sum_{j=1}^{i} L_j, \quad (i = \overline{0, N})$ – координаты границ ячеек (определяются из длины ячейки структуры).

Другое уравнение динамики продольного движения, характеризующее фазу частицы имеет вид:

$$\frac{d\varphi}{d\zeta} = 2\pi\gamma(\gamma^2 - 1)^{-1/2}. \tag{1.45}$$

В декартовых координатах уравнения поперечного движения выглядят следующим образом:

$$m_e\gamma\frac{dv_x}{dt} = eE_x, \quad v_x = \frac{dx}{dt}, \tag{1.46}$$

$$m_e\gamma\frac{dv_y}{dt} = eE_y, \quad v_y = \frac{dy}{dt}. \tag{1.47}$$

Подставляя выражения для напряжённости электрического поля E_x и E_y (уравнения (1.37) и (1.38) соответственно) имеем:

$$m_e\gamma\frac{dv_x}{dt} = eu_0\left(\frac{2\chi}{a^2}x + \frac{2k^2T}{\pi}x\sin(\eta)\right)\cos(\varphi)\,, \tag{1.48}$$

$$m_e\gamma\frac{dv_y}{dt} = eu_0\left(-\frac{2\chi}{a^2}y + \frac{2k^2T}{\pi}y\sin(\eta)\right)\cos(\varphi)\,. \tag{1.49}$$

Введя безразмерную относительную апертуру $\overline{a} = a/\lambda$, а также

учитывая, что

$$\frac{d}{dt} = \frac{\beta\omega}{2\pi}\frac{d}{d\zeta}, \quad \beta\gamma = \sqrt{\gamma^2 - 1}, \tag{1.50}$$

придем к следующему виду уравнений поперечного движения:

$$\frac{\beta\omega}{2\pi}\frac{dx}{d\zeta} = v_x \to \frac{m_e\omega\sqrt{\gamma^2-1}}{2\pi}\frac{d_v x}{d\zeta} = \frac{eu_0}{\lambda^2}\left(\frac{2\chi}{\overline{a}^2}x + \frac{2\overline{k}^2 T}{\pi}x\sin(\eta)\right)\cos(\varphi), \tag{1.51}$$

$$\frac{\beta\omega}{2\pi}\frac{dy}{d\zeta} = v_y \to \frac{m_e\omega\sqrt{\gamma^2-1}}{2\pi}\frac{d_v x}{d\zeta} = \frac{eu_0}{\lambda^2}\left(\frac{2\chi}{\overline{a}^2}x + \frac{2\overline{k}^2 T}{\pi}x\sin(\eta)\right)\cos(\varphi). \tag{1.52}$$

Упростим полученные уравнения. Для этого введём безразмерные переменные

$$\overline{x} = \frac{x}{R_x}, \quad \overline{y} = \frac{y}{R_y}, \tag{1.53}$$

$$\overline{v_x} = \frac{v_x}{V_x}, \quad \overline{v_y} = \frac{v_y}{V_y}, \tag{1.54}$$

где R_0, V_0 – параметры, определяемые характеристиками пучка на входе в канал ускорения.

Если заданы максимальные значения $\frac{dx}{dz}$ и $\frac{dy}{dz}$, которые можно обозначить как d_x и d_y соответственно, то V_x и V_y можно записать в виде:

$$V_x = d_x\beta c, \quad V_y = d_y\beta c. \tag{1.55}$$

Используя введённые параметры уравнения поперечного движения (***3–6 уравнения динамики***) преобразуем к виду:

$$\frac{d\overline{x}}{d\zeta} = \frac{V_x}{R_x}\frac{2\pi}{\beta\omega}\overline{v_x}, \tag{1.56}$$

$$\frac{d\overline{v_x}}{d\zeta} = \frac{\pi e U_0 R_x}{m_e\lambda^2 V_x\omega}\frac{1}{\sqrt{\gamma^2-1}}\left(\frac{2\chi}{\overline{a^2}} + \frac{2\overline{k}^2 T}{\pi}\sin\eta\right)\overline{x}\cos\varphi, \tag{1.57}$$

$$\frac{d\overline{y}}{d\zeta} = \frac{V_y}{R_y}\frac{2\pi}{\beta\omega}\overline{v_y}, \tag{1.58}$$

$$\frac{d\overline{v_y}}{d\zeta} = \frac{\pi e U_0 R_y}{m_e\lambda^2 V_y\omega}\frac{1}{\sqrt{\gamma^2-1}}\left(-\frac{2\chi}{\overline{a^2}} + \frac{2\overline{k}^2 T}{\pi}\sin\eta\right)\overline{y}\cos\varphi. \tag{1.59}$$

1.9 Задание начального распределения

Начальное распределение в фазовом пространстве продольного движения зададим в виде:

$$\varrho_\varphi = (2\pi)^{-1}, \quad \varphi - \varphi_0 \in [-2\pi, 0], \quad \gamma = \gamma_0, \tag{1.60}$$

где ϱ_φ – ϕ-ая компонента плотности распределения;
φ_0 и γ_0 начальная фаза и энергия соответственно.

Зададим распределение частиц по поперечным координатам и скоростям в начальном сечении пучка $\zeta = 0$ в виде:

$$\begin{aligned} x &= R_x \sigma \cos\varphi_x \cos\theta, \quad v_x = V_x \sigma \sin\varphi_x \cos\theta, \\ y &= R_y \sigma \cos\varphi_y \cos\theta, \quad v_y = V_y \sigma \sin\varphi_y \cos\theta. \end{aligned} \tag{1.61}$$

Такой вид распределения означает, что частицы находятся на поверхности и внутри эллипсоида, который описывается уравнением:

$$\frac{x^2}{R_x^2} + \frac{v_x^2}{V_x^2} + \frac{y^2}{R_y^2} + \frac{v_y^2}{V_y^2} = 1. \tag{1.62}$$

Если частица, находится на поверхности этого эллипсоида, то $\sigma = 1$.

Предположим, что компонента плотности распределения частиц, соответствующая координатам $\sigma, \varphi_x, \varphi_y, \theta$, может быть записана в виде:

$$\varrho_{\sigma\varphi_x\varphi_y\theta} = \varrho_\sigma(\sigma)\, \varrho_{\varphi_x}(\varphi_x)\, \varrho_{\varphi_y}(\varphi_x)\, \varrho_\theta(\theta). \tag{1.63}$$

Будем предполагать, что распределение частиц однородно по координатам φ_x, φ_y. Определим величину плотностей распределения по σ, θ таким образом, чтобы частицы равномерно заполняли эллипсоид

(1.62), т.е. чтобы выполнялось условие $\varrho_{x,v_x,y,v_y} = const$:

$$\varrho_{x,v_x,y,v_y} = \begin{pmatrix} \frac{\partial x}{\partial \sigma} & \frac{\partial x}{\partial \varphi_x} & \frac{\partial x}{\partial \varphi_y} & \frac{\partial x}{\partial \theta} \\ \frac{\partial v_x}{\partial \sigma} & \frac{\partial v_x}{\partial \varphi_x} & \frac{\partial v_x}{\partial \varphi_y} & \frac{\partial v_x}{\partial \theta} \\ \frac{\partial y}{\partial \sigma} & \frac{\partial y}{\partial \varphi_x} & \frac{\partial y}{\partial \varphi_y} & \frac{\partial y}{\partial \theta} \\ \frac{\partial v_y}{\partial \sigma} & \frac{\partial v_y}{\partial \varphi_x} & \frac{\partial v_y}{\partial \varphi_y} & \frac{\partial v_y}{\partial \theta} \end{pmatrix} = -\sigma^3 \sin\theta \cos\theta. \tag{1.64}$$

Из выражения (1.64) очевидно, что:

$$\varrho_\sigma = \sigma^3, \quad \varrho_\theta = \sin^3\theta \cos\theta. \tag{1.65}$$

Определим координаты $\Sigma(\sigma)$ и $\Psi(\theta)$ в которых распределение частиц равномерно, с учётом чего выражение (1.65) можно записать в виде:

$$\varrho_\sigma = \frac{d\Sigma}{d\sigma} = \sigma^3, \quad \varrho_\theta = \frac{d\Psi}{d\theta} = -\sin\theta \cos\theta. \tag{1.66}$$

Откуда очевидно, что:

$$\Sigma = \frac{\sigma^4}{4}, \quad \Psi = -\frac{1}{2}\sin^2\theta, \tag{1.67}$$

$$\sigma = (4\Sigma)^{1/4} = \left(\overline{\Sigma}\right)^{1/4}, \quad \overline{\Sigma} = 4\Sigma \in [0,1] \tag{1.68}$$

$$\theta = \arcsin\sqrt{-2\Psi} = \arcsin\sqrt{\overline{\Psi}}, \quad \overline{\Psi} = -2\Psi \in [0,1]. \tag{1.69}$$

Считая что $R_x = R_y = R, \quad V_x = V_y = V$ переменные можно записать в безразмерном виде:

$$\overline{x} = \frac{x}{R}, \quad \overline{v_x} = \frac{v_x}{V}, \tag{1.70}$$

$$\overline{y} = \frac{y}{R}, \quad \overline{v_y} = \frac{v_y}{V}. \tag{1.71}$$

При этом в начальный момент $\zeta = 0$ имеем:

$$\overline{x}(0) = \sigma \cos\varphi_x \cos\theta, \quad \overline{v_x}(0) = \sigma \sin\varphi_x \cos\theta, \tag{1.72}$$

$$\overline{y}(0) = \sigma \cos\varphi_y \sin\theta, \quad \overline{v_y}(0) = \sigma \sin\varphi_y \sin\theta. \tag{1.73}$$

Уравнения (1.72), (1.73) имеют 4 линейно независимых решения,

которые выбираются исходя из начальных условий:

$$\begin{array}{llll} \overline{x_1}(0)=1 & \overline{v_{x_1}}(0)=0 & \overline{y_1}(0)=0 & \overline{v_{y_1}}(0)=0, \\ \overline{x_2}(0)=0 & \overline{v_{x_2}}(0)=1 & \overline{y_2}(0)=0 & \overline{v_{y_2}}(0)=0, \\ \overline{x_3}(0)=0 & \overline{v_{x_3}}(0)=0 & \overline{y_3}(0)=1 & \overline{v_{y_3}}(0)=0, \\ \overline{x_4}(0)=0 & \overline{v_{x_4}}(0)=0 & \overline{y_4}(0)=0 & \overline{v_{y_4}}(0)=1. \end{array} \tag{1.74}$$

Примечание: индексы в уравнении обозначают первое, второе, третье и четвёртое решения соответственно.

Очевидно, что с учётом линейно независимых решений начальные значения (1.72), (1.73) можно представить в виде:

$$\begin{array}{c} \overline{x}(0)=\overline{x_1}(0)\,\sigma\cos\varphi_x\cos\theta+\overline{x_2}(0)\,\sigma\sin\varphi_x\cos\theta, \\ \overline{v_x}(0)=\overline{v_{x_1}}(0)\,\sigma\cos\varphi_x\cos\theta+\overline{v_{x_2}}(0)\,\sigma\sin\varphi_x\cos\theta, \\ \overline{y}(0)=\overline{y_3}(0)\,\sigma\cos\varphi_y\sin\theta+\overline{y_4}(0)\,\sigma\sin\varphi_y\sin\theta, \\ \overline{v_y}(0)=\overline{v_{y_3}}(0)\,\sigma\cos\varphi_y\sin\theta+\overline{v_{y_4}}(0)\,\sigma\sin\varphi_y\sin\theta. \end{array} \tag{1.75}$$

Тогда решения системы уравнений (1.56) – (1.59) поперечной динамики будут иметь вид:

$$\begin{array}{c} \overline{x}(t)=\overline{x_1}(t)\,\sigma\cos\varphi_x\cos\theta+\overline{x_2}(t)\,\sigma\sin\varphi_x\cos\theta, \\ \overline{v_x}(t)=\overline{v_{x_1}}(0)\,\sigma\cos\varphi_x\cos\theta+\overline{v_{x_2}}(t)\,\sigma\sin\varphi_x\cos\theta, \\ \overline{y}(t)=\overline{y_3}(t)\,\sigma\cos\varphi_y\sin\theta+\overline{y_4}(0)\,\sigma\sin\varphi_y\sin\theta, \\ \overline{v_y}(t)=\overline{v_{y_3}}(t)\,\sigma\cos\varphi_y\sin\theta+\overline{v_{y_4}}(t)\,\sigma\sin\varphi_y\sin\theta. \end{array} \tag{1.76}$$

1.10 Оптимизация структуры с ПОКФ

Качество ускоряющей структуры характеризуется функционалом вида:

$$\Phi=\alpha_\phi\Phi_\phi+\alpha_\gamma\Phi_\gamma+\alpha_{tr}\Phi_{tr}, \tag{1.77}$$

$$\Phi_\phi=\int g_\phi(x)\,\varrho(x)\,dx, \tag{1.78}$$

$$\Phi_\gamma=\int g_\gamma(x)\,\varrho(x)\,dx, \tag{1.79}$$

$$\Phi_{tr} = \int g_{tr}(x)\,\varrho(x)\,dx, \tag{1.80}$$

где g_ϕ, g_γ, g_{tr} – штрафные функции;

$\alpha_\phi, \alpha_\gamma, \alpha_{tr}$ – поправочные коэффициенты;

$\varrho(x)$ – плотность частиц в фазовом пространстве.

Штрафные функции имеют следующие значения:

$$g_\phi = \begin{cases} (\varphi - \varphi_{max})^2, & \varphi > \varphi_{max}, \\ 0, & \varphi \in [\varphi_{min}, \varphi_{max}], \\ (\varphi - \varphi_{min})^2, & \varphi < \varphi_{min}, \end{cases} \tag{1.81}$$

$$g_\gamma = \begin{cases} (\gamma - \gamma_{max})^2, & \gamma > \gamma_{max}, \\ 0, & \gamma \in [\gamma_{min}, \gamma_{max}], \\ (\gamma - \gamma_{min})^2, & \gamma < \gamma_{min}, \end{cases} \tag{1.82}$$

$$g_{tr} = \alpha_{x_1} \begin{cases} (\overline{x_1} - \overline{x_{max}})^2, & \overline{x_1} > \overline{x_{max}}, \\ 0, & \overline{x_1} \le \overline{x_{max}}, \end{cases} + \alpha_{x_2} \begin{cases} (\overline{x_2} - \overline{x_{max}})^2, & \overline{x_2} > \overline{x_{max}}, \\ 0, & \overline{x_2} \le \overline{x_{max}}, \end{cases} +$$
$$+\alpha_{y_1} \begin{cases} (\overline{y_1} - \overline{y_{max}})^2, & \overline{y_1} > \overline{y_{max}}, \\ 0, & \overline{y_1} \le \overline{y_{max}}, \end{cases} + \alpha_{y_2} \begin{cases} (\overline{y_2} - \overline{y_{max}})^2, & \overline{y_2} > \overline{y_{max}}, \\ 0, & \overline{y_2} \le \overline{y_{max}}, \end{cases} \tag{1.83}$$

2 Разработка программного комплекса моделирующего динамику пучка заряженных частиц

В главе приведено описание этапов разработки программного компонента для моделирования динамики пучка частиц; проведён анализ возможных методов оптимизации и оценочных критериев в соответствии с которыми проведена оценка выбранных способов оптимизации.

2.1 Оцениваемые критерии и характеристики

При оценке эффективности программы используются различные критерии и характеристики, имеющие как практический, так и теоретический характер. Т.к. для отдельной вычислительной системы алгоритм может быть разработан с учётом полного использования преимуществ системы и поэтому достигает на ней высокой степени эффективности, чтобы однозначно оценить эффективность алгоритма используются следующие критерии системной и пространственной эффективности [33–35].

Системная эффективность оценивает скорость выполнения задачи алгоритмом. Выполняя различные алгоритмы на одной системе с одними и теми же наборами данных можно определить относительное время выполнения алгоритма на системе, в таком случае оценка времени становится мерой системной эффективности для каждого из алгоритмов. В рамках работы для оценки по настоящему критерию используется отношения времени выполнения контрольного алгоритма ко времени выполнения оптимизированного алгоритма.

Примечание: временные замеры производятся с помощью вызова системной функции вывода времени (GetSystemTime).

Пространственная эффективность – мера относительного количества внутренней памяти, используемой алгоритмом. Она может указать, какого типа компьютер способен выполнять этот алгоритм и полную системную эффективность алгоритма. Из-за увеличения объёма памяти в новых системах, анализ пространственной эффективности менее важен, но может быть показательным, например для оценки возможности выполнения алгоритма на старых системах с малыми объёмами оперативной памяти. В рамках настоящей работы не имеет большого смысла производить оценку по этому критерию, т.к. объёмы обрабатываемых данных и методы их обработки не налагают больших ограничений в области оперативной памяти.

Сверх этого необходима и качественная оценка разработанных алгоритмов и способов их оптимизации которую можно провести по таким критериям как: простота использования, возможности масштабирования, независимости от среды исполнения и т.п.

2.2 Этапы разработки и архитектура

С точки зрения архитектуры, разрабатываемый программный комплекс имеет 2 уровня: интерфейсный и прикладной [33], в результате чего программный комплекс разбит на 2 функциональных компонента:

- интерфейсный компонент – обеспечивающий методы пользовательского интерфейса: ввод/вывод данных, обработка поступивших данных, взаимодействие с прикладной частью (доступность/недоступность метода оптимизации в случае аппаратных ограничений среды функционирования);
- прикладной компонент – выполняющий решение дифференциальных уравнений, описывающих математическую модель в соответствии с поступившими входными данными и передачу выходных данных в интерфейсный компонент.

Интерфейсный компонент сказывается незначительно на потребление системных ресурсов, т.к. выполнен с использованием интерфейса программирования приложений Windows API (на низком уровне), поэтому

анализу будет подвергнут только прикладной компонент и входящие в его состав модули.

Для удобства приведём уравнениями динамики частиц повторно:

- первое уравнение продольной динамики (уравнение (1.43)):

$$\frac{d\gamma}{d\zeta} = \frac{2eU_0}{\pi m_e c^2}\overline{k}T\cos(\eta)\,cos(\varphi), \tag{2.1}$$

где $\omega t = \varphi$ – фаза частицы.

Фаза пространственной модуляции частицы η имеет вид:

$$\eta = \pi\left((i-1) + \frac{z - c_{i-1}}{L_j}\right), \quad z \in [c_{i-1}, c_i], \tag{2.2}$$

где $c_i = \sum_{j=1}^{i} L_j, \quad (i = \overline{0, N})$ – продольные координаты границ ячеек (определяются из длины ячейки структуры);

- второе уравнение продольной динамики (уравнение (1.45)):

$$\frac{d\varphi}{d\zeta} = 2\pi\gamma(\gamma^2 - 1)^{-1/2}; \tag{2.3}$$

- первое уравнение поперечной динамики (уравнение (1.56)):

$$\frac{d\overline{x}}{d\zeta} = \frac{V_x}{R_x}\frac{2\pi}{\beta\omega}\overline{v_x}; \tag{2.4}$$

- второе уравнение поперечной динамики (уравнение (1.57)):

$$\frac{d\overline{v_x}}{d\zeta} = \frac{\pi e U_0 R_x}{m_e\lambda^2 V_x\omega}\frac{1}{\sqrt{\gamma^2-1}}\left(\frac{2\chi}{\overline{a^2}} + \frac{2\overline{k}^2 T}{\pi}\sin\eta\right)\overline{x}\cos\varphi, \tag{2.5}$$

- третье уравнение поперечной динамики (уравнение (1.58)):

$$\frac{d\overline{y}}{d\zeta} = \frac{V_y}{R_y}\frac{2\pi}{\beta\omega}\overline{v_y}; \tag{2.6}$$

- четвёртое уравнение поперечной динамики (уравнение (1.59)):

$$\frac{d\overline{v_y}}{d\zeta} = \frac{\pi e U_0 R_y}{m_e \lambda^2 V_y \omega} \frac{1}{\sqrt{\gamma^2 - 1}} \left(-\frac{2\chi}{\overline{a^2}} + \frac{2\overline{k}^2 T}{\pi} \sin\eta \right) \overline{y} \cos\varphi. \qquad (2.7)$$

Разработка программного компонента отвечающего за математические вычисления была разбита на 3 этапа:

- разработка прототипа программного компонента;
- анализ способов оптимизации и возможности их применения;
- применение выбранной оптимизации и последующая отладка результата.

В процессе разработки прототипа заложены основы математической модели, которая включает в себя описание уравнений динамики пучка частиц (см. выше), а также способы их решения, в качестве которых были использованы: метод Рунге-Кутта 4-го порядка (далее по тексту – метод РК) [46] и метод leapfrog [47]. Использование различных способов для вычисления позволяет произвести независимую оценку результата вычисления каждого из способов. Анализ способов вычисления показал, что оба способа соответствуют предъявляемым требованиям по точности (полученная точность решений составила порядка $10^{-}15$ у обоих способов). Метод РК является методом численного интегрирования 4-го порядка, а метод leapfrog – 2-го, на основании чего метод РК принят в качестве основного метода интегрирования правых частей дифференциальных уравнений.

На втором этапе разработки были составлены алгоритмы с учётом применения наиболее популярных средств оптимизации [35], проведён анализ средств оптимизации с точки зрения ускорения процесса вычислений и как следствие – снижения использования системных ресурсов. Основные средства оптимизации вычислений можно разделить на следующие классы: специализированные математические библиотеки, средства оптимизации CPU и средства оптимизации GPU. Приведём ниже краткую характеристику каждого из средств:

- математические библиотеки – это набор специальным образом оптимизированных программ, предназначенных для решения различных математических задач. Наибольшее распространение получили такие библиотеки как: BLAS, Intel Math Kernel Library, LAPACK, atlas. Существуют также и оптимизированные для решения дифференциальных уравнений библиотеки, такие как: FETK, odeint, alglib. Анализ эффективности оптимизации использования этих библиотек не производится, т.к. основное внимание уделяется средствам распределения вычислений (с использованием результатов которых впоследствии возможно произвести анализ эффективности использования данных библиотек для конкретных задач);

- CPU-средства – платформы включающие в себя специальный компилятор и набор средств осуществляющих распределение вычислений с помощью CPU, например: OpenMP, MPI;

- GPU-средства – аналогичные CPU-платформам для распределения вычислений с существенным отличием в том, что процесс вычислений выполняется на GPU, например: amp, CUDA Toolkit, OpenACC.

Третий этап разработки программного компонента включает в себя оптимизацию разработанных вычислительных алгоритмов и их последующую отладку. В ходе отладки алгоритмов были устранены недочёты связанные со спецификой применения средств оптимизации, с учётом чего были произведены соответствующие доработки вычислительных модулей.

2.3 Характеристики испытательного стенда

Испытательный стенд, на котором были произведены вычисления имеет следующие характеристики:

- операционная система: Arch Linux;

- ядро: 4.2.1-1-ARCH x86_64;

- компилятор: 64 bit gcc: 5.2.0;

- характеристики CPU: AMD Athlon 64 x2 (dual core): 2.6 ГГц;
- размер процессорного кэша: L1: 128 Кб, L2: 1024 Кб;
- процессорные инструкции: (mmx, sse, sse2, sse3, sse3m 3DNow!);
- объём оперативной памяти 4096 Мб;
- GPU: NVidia GeForce GT610, Nvidia GeForce GTX970.

Подробные характеристики используемых GPU приведены в таблицах № 2.1, 2.2.

Таблица 2.1 – Характеристики GPU NVidia GeFroce GT 610

Технические характеристики		
№	Характеристика	Значение
1	Частота графического процессора, МГц	810
2	Частота шейдерных блоков, МГц	1200
3	Объём видеопамяти, Мб	1024
4	Тип видеопамяти	GDDR3
5	Разрядность шины видеопамяти, бит	64
Математические характеристики		
6	Число универсальных процессоров	48
7	Число текстурных блоков	8
8	Число блоков растеризации	4
9	Поддержка CUDA	Версии 2.1
10	Поддержка OpenCL	Версии 1.2
11	Рассеиваемая тепловая мощность, Вт	29
12	Дополнительно	DX 11, OpenGL 4.2

Таблица 2.2 – Характеристики GPU MSI GeForce GTX 970

Технические характеристики		
№	Характеристика	Значение
1	Частота графического процессора, МГц	1140
2	Частота шейдерных блоков, МГц	7010
3	Объём видеопамяти, Мб	4096
4	Тип видеопамяти	GDDR5
5	Разрядность шины видеопамяти, бит	256
Математические характеристики		
6	Число универсальных процессоров	1664
7	Число текстурных блоков	104
8	Число блоков растеризации	56
9	Поддержка CUDA	Версии 5.2
10	Поддержка OpenCL	Версии 1.2
11	Рассеиваемая тепловая мощность, Вт	29
12	Дополнительно	DX 12, OpenGL 4.5, Vulkan API

2.3.1 Входные и выходные данные

Входными данными для вычислительного модуля является массив значений, содержащий параметры структуры с ПОКФ, производящей ускорение пучка заряженных частиц:

- Задающиеся в виде констант: частота, начальная энергия, значение амплитуды, начальный радиус пучка, фундаментальные константы (масса и заряд частицы, π, скорость света, квадрат скорости света), значение шага приращения функции ($h = 0.0002$);
- Изменяющиеся параметры (для каждой из секций свой набор параметров): длина ячейки, эффективность и апертура. Данные наборы параметров определяют объём входных данных (для отладки было использовано 300 наборов параметров).

Выходные данные формируются в виде массива решений дифференциальных уравнений, описывающих заданную математическую модель.

2.4 Разработка прототипа

Разработаны и отлажены вычислительные модули программного компонента производящие вычисление дифференциальных уравнений двумя способами: методом РК и методом Leapfrog. Исходный код разработанных вычислительных модулей на языке C приведён в Приложении № 1. Для отладки вычислительных модулей были использованы уравнения продольной динамики частиц, т.к. эти уравнения не отличаются по характеру от уравнений поперечной динамики и в полной мере позволяют оценить ход работы вычислительного модуля и выполнить его отладку. Результаты работы вычислительных модулей (характеристики используемых системных ресурсов) приведены в таблицах 2.3, 2.4. Полученные данные используются как контрольные для оценки результата оптимизации вычисления.

Таблица 2.3 – Результат вычисления по методу РК

№ п/п	Критерий	Значение
1	Время вычисления, с	8,55487
2	Объём потребляемой памяти,	7
3	Объём дискового пространства, %	–
4	Порядок точности результата	10^{-15}

Таблица 2.4 – Результат вычисления по методу leapFrog

№ п/п	Критерий	Значение
1	Время вычисления, с	9,53127
2	Объём потребляемой памяти, %	10
3	Объём дискового пространства, %	–
4	Порядок точности результата	10^{-15}

В результате моделирования динамики пучка частиц были получены следующие характеристики пучка частиц в канале с ПОКФ (Рисунки 2.1, 2.2, 2.3). Полученные характеристики пучка частиц были использованы для наглядной отладки правильности результата вычислений.

- ;
- распределение крупных частиц в фазовом пространстве продольного движения на выходе из канала с ПОКФ;

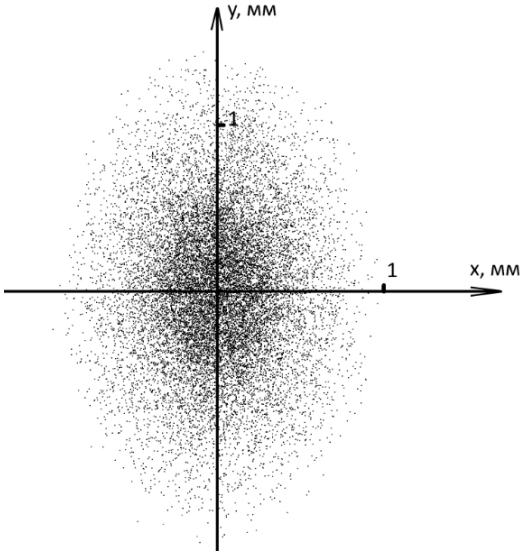

Рисунок 2.1 – Распределение крупных частиц по поперечным координатам x и y на выходе из канала с ПОКФ.

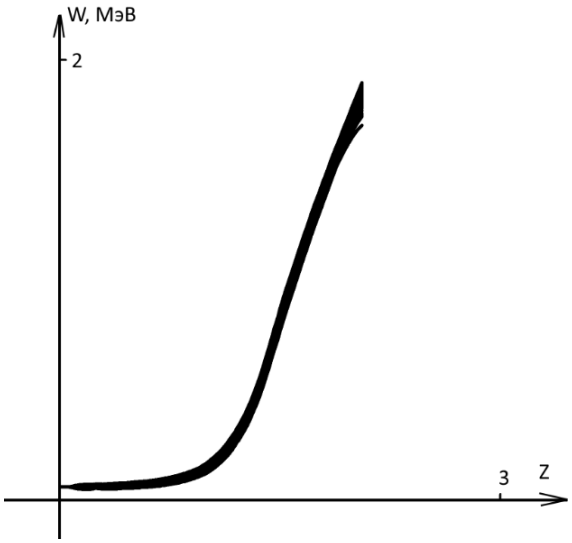

Рисунок 2.2 – Зависимость энергии крупных частиц W от продольной координаты z

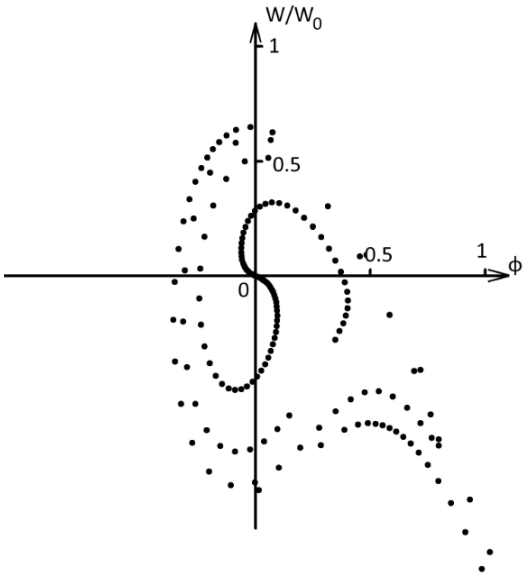

Рисунок 2.3 – Распределение крупных частиц в фазовом пространстве продольного движения на выходе из канала с ПОКФ

2.4.1 Алгоритм работы

Алгоритм работы вычислительного модуля представлен на рисунке 2.4. Входные данные берутся из txt-файла исходных данных в кодировке utf-8 расположенного в каталоге программы.

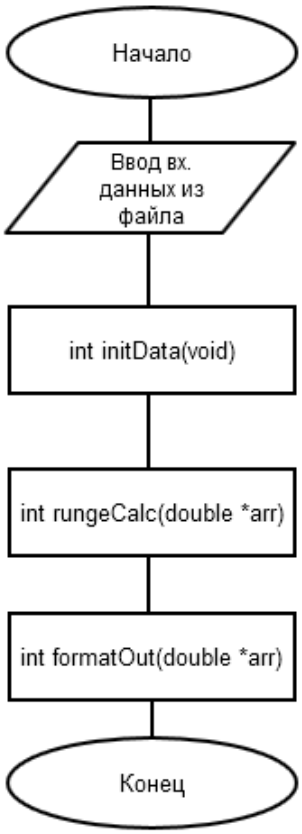

Рисунок 2.4 – Схема последовательного алгоритма.

Последовательность операций разбита на следующие функции:

- `int initData(void)`:

  функция выполняющая выделение памяти, чтение данных из файла и соответствующее присвоение исходных данных переменным. Под исходными данными подразумеваются начальные значения, задаваемые в математической модели, параметры ускоряющей структуры. Кроме инициализации исходных данных, функция задаёт значения используемых констант: заряд электрона, число пи, скорость света, квадрат скорости света;

- `int rungeCalc(double *arr)`:

  функция выполняющая интегрирования методом РК;

- `int formatOut(double *arr)`:

  функция производящая форматирование выходных данных в зависимости от потребностей.

Преимущество разбиения процесса вычисления на элементарные функциональные объекты заключается в простоте понимания алгоритма, а также в простоте последующей оптимизации – в случае оптимизации достаточно произвести вызов модифицированной функции, вместо исходной. Алгоритм интегрирования по методу РК приведён на рисунке 2.5. Алгоритм является стандартным и приведён для наглядного сравнения с оптимизированными версиями алгоритмов.

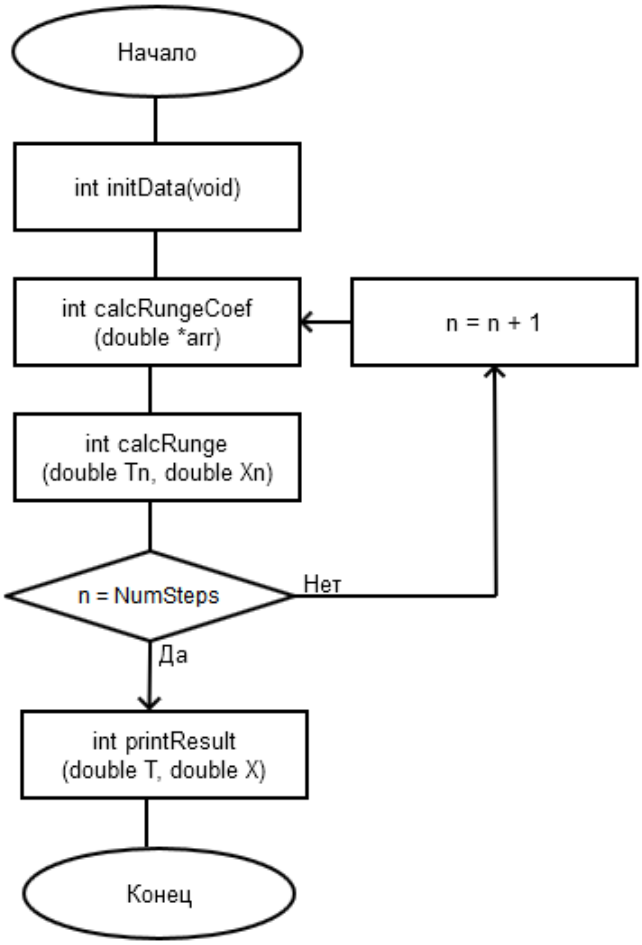

Рисунок 2.5 – Схема алгоритма метода РК.

В алгоритме:

- `int initData(void)`:

  функция выполняющая установку начальных значений: параметры ускоряющей структуры, количество шагов интегрирования, требуемая точность;

- `int calcRungeCoef(double *arr)`:

  функция вычисляющая коэффициенты k_1, k_2, k_3, k_4 по методу РК;

- `int calcRunge(double Tn, double Xn)`:

  функция вычисляющая значения функции и аргумента (T_n, X_n);

- `int printResult(double T, double X)`:

  функция выполняющая вывод результата вычислений.

2.4.2 Оптимизация с помощью математических библиотек

Оптимизация с помощью библиотеки BLAS

BLAS (Basic Linear Algebra Subprograms) – стандарт интерфейса программирования приложений для библиотек, выполняющих основные операции линейной алгебры, такие как умножение векторов и матриц. Впервые библиотека опубликована в 1979 году. Библиотека используется в составе больших вычислительных библиотек, таких как Intel Math Kernel Library (Intel MKL), ATLAS, CUDA (в составе библиотеки cuBLAS), LAPACK а также других популярных математических пакетах. Подробный перечень функций, а также документация по применению доступна на официальном сайте [48].

Из-за того, что BLAS оперирует матрицами были реализованы следующие функции вычислительного модуля, использующие вычислительные возможности библиотеки:

- функция производящая предварительный разбор входных данных (разбиение на массивы, кратные размеру кластера), в том числе инициализацию выровненных массивов для последующего их заполнения:

  ```
  blArrPtr* makeBlasMatrix(double *arrName, uint size);
  ```

- функция производящая операции умножения между двумя элементами средствами библиотеки BLAS:

  ```
  blArrPtr* multiplyBlasArr(blArrPtr* left, blArrPtr* right);
  ```

- функция производящая сложение между двумя элементами средствами библиотеки BLAS:

  ```
  blArrPtr* appendBlasArr(blArrPtr *left, blArrPtr *right).
  ```

Примечание: blArrPtr* – ссылочный тип, представляющий собой структуру содержащую переданный массив (указатель на элемент)

числовых значений (в случае нехватки элементов до нужного размера соответствующего кластеру – массив дополняется нулями) и размерность массива. Использование динамического (ссылочного) типа даёт очевидные преимущества в виде сокращения потребляемой памяти в процессе обработки данных.

Разработанные функции элементарных операций заменяют (перегружают в терминах объектно-ориентированных языков) элементарные операции, выполняемые в ходе интегрирования по методу РК. Преимущество такого способа заключается в независимости производимых операций от входных данных, т.к. разработанные функции производят предварительную обработку и основные операции в ходе вычисления алгоритма. По сравнению с последовательным алгоритмом вычисления (рисунок 2.4), оптимизированный с помощью библиотеки BLAS отличается только функцией, выполняющей интегрирование по методу РК (3-ий блок на схеме), последовательность действий которого приведена на рисунке 2.6. В алгоритме:

- `int initData(void)`:

  функция выполняющая установку начальных значений: параметры ускоряющей структуры, количество шагов интегрирования, требуемая точность;

- `blArrPtr* makeBlasMatrix(double *arrName, uint size)`:

  функция формирующая массивы по заданным исходным данным (дополняется нулями в случае отсутствия данных);

- `blArrPtr* calcRungeBLAS(blArrPtr* arr)`:

  функция вычисляющая коэффициенты k_1, k_2, k_3, k_4 по методу РК с помощью библиотеки BLAS;

- `int calcRunge(double Tn, double Xn)`:

  функция вычисляющая значения функции и аргумента (T_n, X_n);

- `int printResult(double T, double X)`:

  функция выполняющая вывод результата вычислений.

Для оценки оптимизации используются две версии библиотеки BLAS – оригинальная, написанная на языке $Fortran$ (используется как внешняя функция языка C), а также перенесённая на язык C библиотека cblas, для того, чтобы оценить разницу оптимизации обоими типами библиотек. В силу разработанных вспомогательных функций – разницы между вызовом той или иной версии библиотеки нет (выбор версии библиотеки осуществляется с помощью макроса до процесса компиляции). Полученные результаты оптимизации приведены в таблицах 3.1, 3.2 для библиотеки языка и $Fortran$ соответственно.

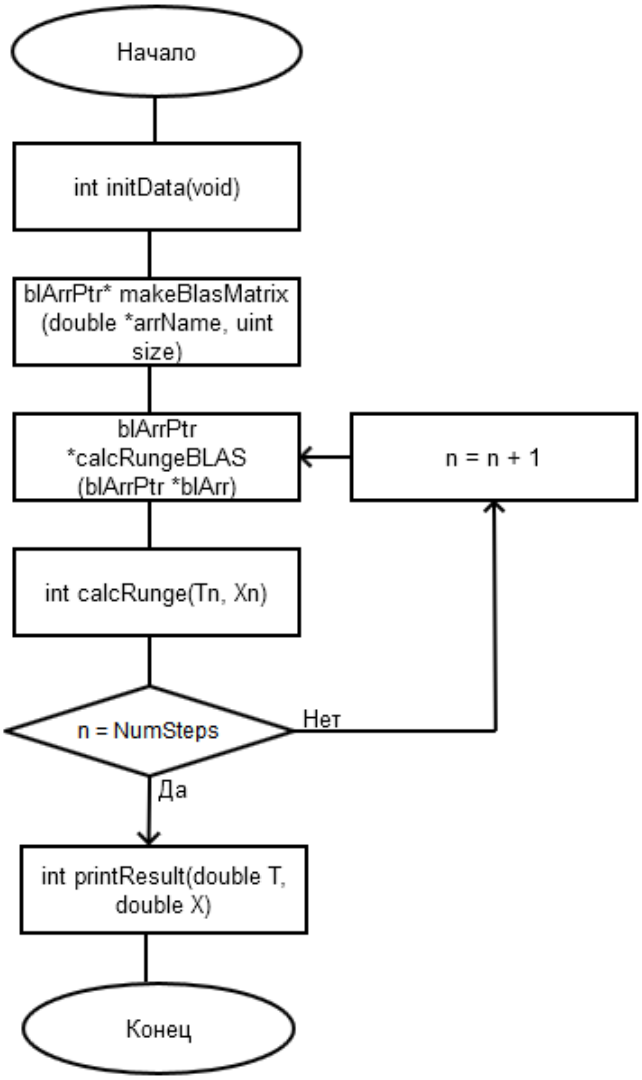

Рисунок 2.6 – Схема алгоритма метода РК оптимизированного с помощью библиотеки BLAS.

2.4.3 Оптимизация с помощью CPU

С помощью только средств CPU оптимизацию процесса вычисления можно выполнить следующими способами:

- многопроцессорная оптимизация;
- многопоточная оптимизация;
- комбинированный способ оптимизации.

Многопроцессорная оптимизация – широко применима на гибридных системах, состоящих из нескольких вычислительных процессоров, объединённых в единый вычислительный кластер. В этом случае данные между элементами кластера передаются с помощью сообщений; такие системы также называют – системами с разделённой памятью [36]. Слабым местом такой оптимизации является её сложность – необходимо предусмотреть все возможные взаимодействия между вычислительными процессорами посредством передачи сообщений, в том числе необходимо организовать корректную передачу данных между узлами кластера, что сказывается на используемых ресурсах в процессе вычисления. При разработке алгоритмов с учётом многопроцессорной оптимизации отладка зачастую затруднена. Подробная информация про с помощью Open MPI приведена в [56].

В случае многопоточной оптимизации – система как правило представляет единое целое и имеет как минимум 1 вычислительный процессор, который обрабатывает данные в едином адресном пространстве [36]. Основные сложности оптимизации заключаются в синхронизации потоков, выполняющих вычисления, между собой, для исключения одновременного обращения (чтение+запись) к одному и тому же участку памяти, что может привести к непредвиденным результатам работы. Более простой и быстрый метод с точки зрения реализации; нашёл отражение во множестве программных интерфейсов (Windows API, boost, QT, OpenMP и т.п.). Отладка алгоритмов с учётом многопоточной оптимизации менее сложная, чем в случае многопроцессорной оптимизации.

Комбинированный способ оптимизации, называемый также “гибридным”, сочетает в себе свойства двух предыдущих способов

оптимизации и более трудоёмкий, т.к. необходимо учитывать особенности обоих способов оптимизации одновременно, что зачастую неоправданно исходя из трудо-временных затрат. В рамках настоящей работы этот способ оптимизации не рассматривается. Кроме того, получили распространение способы оптимизации вычислений по глобальной сети Интернет (облачные вычисления), но в рамках фундаментальных определений эти способы можно отнести к многопроцессорной оптимизации (взаимодействие между “узлами кластера” в данном случае осуществляет через например стек TCP/IP).

Оптимизация с помощью Open MPI

MPI (Message Passing Interface) – стандарт передачи данных между процессорами [40]. Наибольшее применение стандарт MPI нашёл при разработке программ для вычислительных кластеров и суперкомпьютеров (системы с распределённой памятью). Основным средством коммуникации между процессами в MPI являются сообщения, реализация которых описывается в POSIX IPV. Существуют большое многообразие реализаций многопроцессорной оптимизации (описываемой стандартом MPI) под различные платформы и различные языки программирования, в настоящей работе используется открытая реализация – Open MPI [40] выполненная в виде интерфейса программирования приложений (API), с помощью средств которого осуществляется межпроцессорное взаимодействие.

Главная особенность многопроцессорной оптимизации заключается в том, что для её применения необходима модификация существующего алгоритма до параллельного вида (параллельный алгоритм как правило существенно отличается от последовательного алгоритма выполнения). Составление параллельного алгоритма выполнения – трудоёмкая задача, что налагает определённые ограничения по применению многопроцессорной оптимизации.

Оптимизацию алгоритма с помощью MPI можно описать последовательностью, изображённой на рисунке 2.7

- запуск программы -> создание копий полностью идентичных процессов запущенной программы (дочерних процессов) на узлах системы;

- параллельное выполнение кода в каждой из запущенном процессе и взаимодействие между ними с помощью сообщений;
- передача результирующих данных в родительский процесс;
- завершение программы, очистка выделенной памяти и вывод результата.

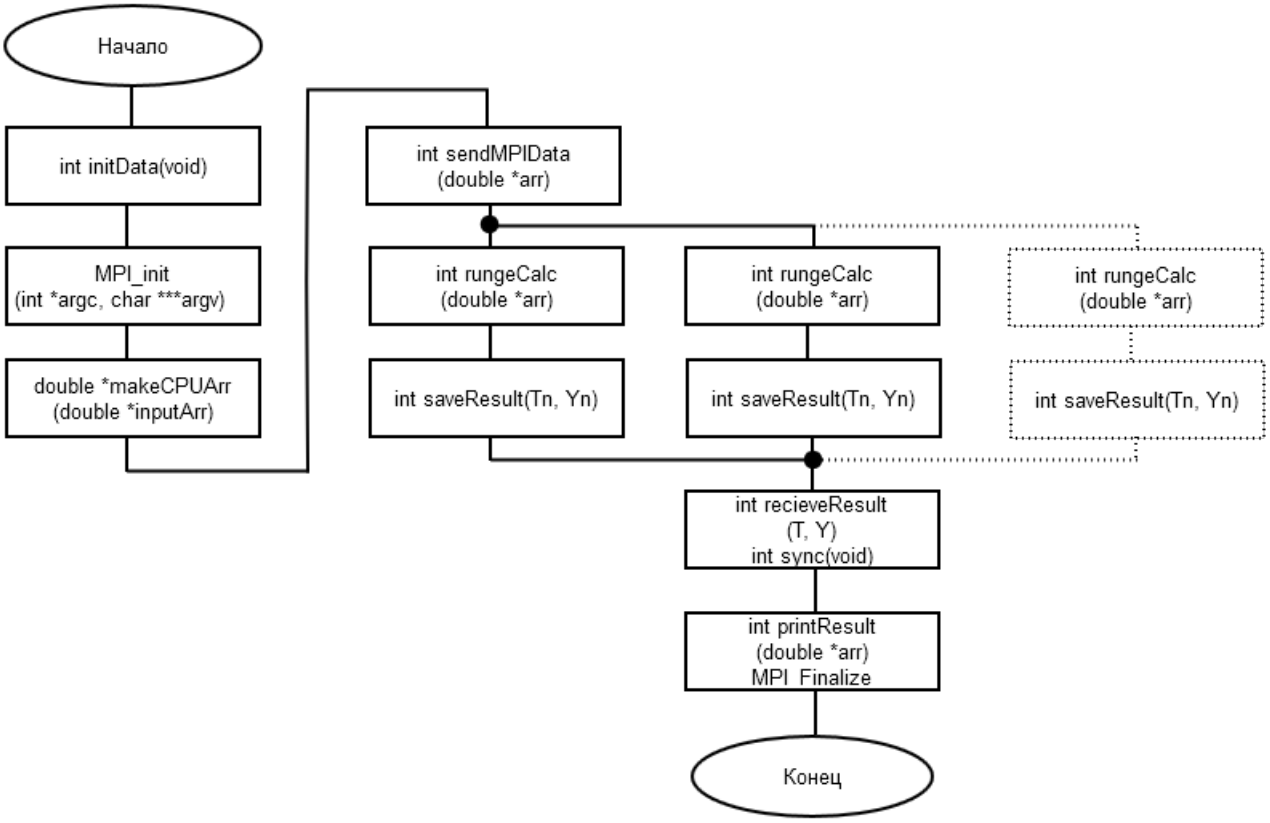

Рисунок 2.7 – Схема алгоритма вычислений оптимизированного с помощью Open MPI.

В алгоритме:

- `int initData(void)`:

  функция выполняющая установку начальных значений: параметры ускоряющей структуры, количество шагов интегрирования, требуемая точность;

- `double *makeCPUArr(double *inputArr)`:

  функция выполняющая разделение массива данных для обработки на массивы, кратные количеству CPU;

- `int sendMPIData(double *arr)`:

  функция осуществляющая отправку данных в отдельный процесс с помощью сообщений в соответствии с Open MPI API [40];

- `int rungeCalc(double *arr)`:

  функция вычисляющая значения функции и аргумента (T_n, X_n) по методу РК (алгоритм см. выше);

- `int saveResult(double Tn, double Xn)`:

  функция сохраняющая результат в переменную для последующей отправки в главный процесс;

- `int recieveResult(double Tn, double Xn)`:

  функция сохраняющая результат вычисления для последующей отправки в главный процесс;

- `int sync(void)`:

  функция выполняющая синхронизацию процесса выполнения программы для исключения блокировки программы;

- `int printResult(double *arr)`:

  функция выполняющая вывод результата вычислений.

В силу того, что испытательный стенд не является многопроцессорной высокопроизводительной системой (имеет всего 2 вычислительных ядра) полученные характеристики оптимизации с помощью Open MPI, приведённые в таблице 3.3, существенно уступают другим результатам оптимизации, (но могут их превзойти при выполнении в оптимальной для данного метода оптимизации среде).

Оптимизация с помощью OpenMP

OpenMP (Open Multi-Processing) – открытая реализация стандарта многопоточной оптимизации [38], которая представляет из себя интерфейс программирования приложений (специальные директивы компилятора, библиотечные функции и переменные сред) поставляемый совместно с компилятором, обрабатывающим эти директивы. Первая реализация появилась в 1997 году для языка $Fortran$; для языка C/C++ реализация выпущена в 1998 году. В конце ноября 2015 года вышла в свет спецификация стандарта 4.5 учитывающая многие современные особенности аппаратных платформ. В научной среде получила распространение проприетарная реализация стандарта (входящая в состав Intel MKL) [57].

В отличии от многопроцессорной оптимизации, многопоточная оптимизация не требует существенных модификаций алгоритма выполнения – требуется лишь минимальная модификация исходных текстов в виде добавления специальных директив компилятора. За счёт применения такого подхода к оптимизации повышается скорость разработки и отладки.

Алгоритм вычислений, оптимизированный с помощью OpenMP не имеет существенных различный с контрольным (исходным) алгоритмом: в оптимизированном варианте присутствуют дополнительные директивы компилятора, осуществляющие многопоточную оптимизацию вычислений [38]. Результаты оптимизации с помощью OpenMP приведены в таблице 3.4.

2.4.4 Оптимизация с помощью GPU

Оптимизация вычислений с помощью GPU заключается в исполнении вычислений на GPU, архитектурные особенности которых более оптимальны для выполнения параллельной обработки инструкций (разница архитектуры CPU и GPU приведена в [37]. Существенный недостаток такой оптимизации также следует из архитектурных особенностей GPU – малый объём адресного пространства (в современных и специализированных GPU проблема практически отсутствует). Исходя из этого целесообразно использовать GPU для оптимизации обработки больших объёмов данных,

т.к. при передаче и приёме данных с/на GPU происходит задержка во времени, затрачиваемая на передачу данных из одного адресного пространства в другое и взаимодействие с GPU.

Оптимизация с помощью CUDA

CUDA (Compute Unified Device Architecture) – архитектура параллельных вычислений от компании Nvidia (де-факто – стандарт GPU оптимизации) [39]. Реализации CUDA API существуют для большинства современных языков программирования, таких как: common lisp, C#, haskell, java, Perl, python, ruby, fortran (PGI compiler for cuda). CUDA включает два интерфейса API (возможно использовать один из них):

- высокоуровневое API (CUDA Runtime API) – работает над низкоуровневым API, все вызовы runtime транслируются в простые инструкции, обрабатываемые низкоуровневым Driver API. В состав высокоуровневого API входят математические библиотеки (CUBLAS, CUFFT);

- низкоуровневый API (CUDA Driver API) – работает непосредственно с устройством и предполагает знания об их работе.

Подробнее про архитектуру GPU, а также особенности CUDA API можно прочесть на официальном сайте [39].

В CUDA существуют 3 основных определения, вокруг которых построен весь процесс вычислений: устройство (device), хост (host) и ядро (kernel):

- хост – процессор, управляет процессом вычислений: запускает задачи, выделяет память на устройстве, перемещает память в/на GPU (в рамках CUDA внутренняя память устройства и хоста различаются). При описании функций процессора в соответствии с соглашением API, к имени функции добавляется суффикс h_;

- устройство – GPU, выполняет операции, передаваемые от процессора. При описании функций процессора в соответствии с соглашением API, к имени функции добавляется суффикс d_;

- ядро – вычислительная задача (последовательный код), параллельно выполняемая хостом на устройстве (в несколько потоков) . Код ядра отличается от последовательного кода:

  – в ядре можно получить идентификатор текущего (выполняемого потока) на устройстве (используется для управления процессом вычислений на GPU; аналогичный принцип используется для управления процессом вычислений в многопоточных приложениях). Исторически GPU обрабатывали преимущественно 3-х мерные изображения, отсюда и пошла аналогия задания количества активных вычислительных потоков с помощью 3-х мерной координатной сетки.
  Размерности $grid_x \times grid_y \times grid_z$ блоков, в соответствии с которой задаются размеры блока в определённых 3-х мерных координатах. Иначе говоря 1 блок состоит из $block_x \times block_y \times block_z$ потоков, а всего доступно соответственно $grid_x \times grid_y \times grid_z \times block_x \times block_y \times block_z$ потоков.
  Количество потоков в 1 блоке ограничено аппаратными особенностями GPU (в современных GPU количество потоков 1024 и более, в более старых GPU – менее 1024).

  – в ядре имеется возможность синхронизации выполняемых потоков, а также доступны специальные переменные, характеризующие “текущее положение ядра” (threadIdx, blockIdx – содержащие координаты потока в текущем блоке и координаты блока в общей сетке блоков соответственно);

  – в ядре отсутствует рекурсия, нет статических переменных внутри функций и переменного числа аргументов;

  – поддерживается два вида доступа к памяти: линейный с доступом по 32-битным указателям, и т.н. CUDA-массивы с доступом только через функции текстурной выборки.

Типовой алгоритм CUDA-программы выглядит следующим образом:

- исходная задача разбивается на меньшие задачи по данным;

- производится инициализация устройства, его параметров и т.п.;

- входные данные делятся на кратные блоки, которые размещаются в разделяемой памяти системы;
- после задания характеристик устройства процессор инициирует запуск ядер, выполняемых на GPU, каждое ядро обрабатывается блоком потоков;
- полученное ядро загружается в разделяемую память из глобальной, в которой производятся необходимые вычисления;
- полученные в результате вычислений данные копируются из разделяемой памяти GPU обратно в глобальную память CPU.

Учитывая типовой алгоритм CUDA-программы, оптимизированный последовательный алгоритм вычисления динамики пучка частиц можно представить в виде, представленном на рисунке 2.8.

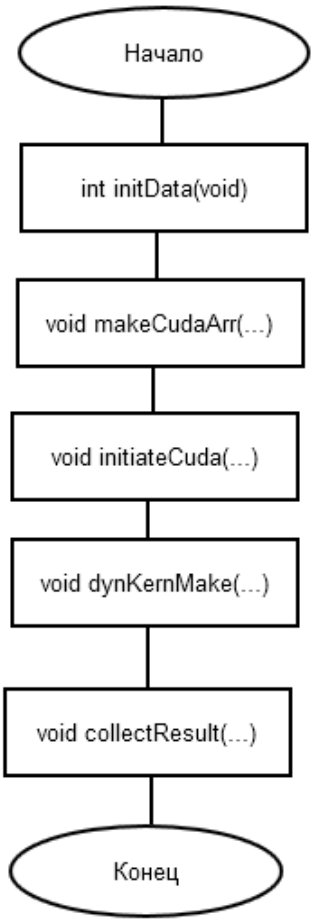

Рисунок 2.8 – Схема алгоритма вычислений оптимизированного с помощью CUDA.

Для оптимизации вычислений с помощью CUDA были разработаны следующие дополнительные функции вычислительного модуля:

- `int initData(void)`:

  функция выполняющая установку начальных значений: параметры ускоряющей структуры, количество шагов интегрирования, требуемая точность;

- `void makeCudaArr(double *arr)`:

  функция, выполняющая разбивку массива исходных данных на выровненные массивы для отправки на вычислительное ядро;

- `void initiateCuda(const uchar4 * const h_input, unsigned char* const h_output, size_t numRows, size_t numCols)`:

  функция, производящая выделение памяти для входных/выходных данных и перемещение обрабатываемых данных из CPU на GPU;

- `void getTimeCuda(void);`

  функция производящая замеры времени, производящая вызов функции API cudaEventElapsedTime;

- `void dynKernMake(const uchar4* const d_input, unsigned char* const d_output, int numRows, int numCols)`:

  функция инициализирующая вычислительное ядро, которое содержит вызов функции rungeCalc, алгоритм которой приведён на рисунке 2.5.

При оптимизации CUDA-программы необходимо правильно подобрать размер и количество блоков памяти, обрабатываемых GPU. Большее количество потоков в блоке уменьшит задержки при обращении к памяти, но при этом снизит доступное для работы число регистров. Кроме того, блок из 512 потоков неэффективен (разработчики Nvidia рекомендуют использовать блоки по 128 или 256 потоков, как компромиссное значение для достижения оптимальных задержек и количества доступны регистров). В процессе оптимизации программ CUDA можно выделить следующие важные моменты:

- активно использовать разделяемую памяти, так как она быстрее глобальной видеопамяти GPU;

- операции чтения/записи из глобальной памяти GPU необходимо по возможности объединять (coalesced). Для этого используют специальные типы данных для чтения/записи сразу по 32/64/128 бита за одну операцию.

Результаты оптимизации вычислений приведены в таблицах 3.5 и 3.6.

Оптимизация с помощью OpenACC

OpenACC (Open Accelerators) – стандарт для параллельной оптимизации, разрабатываемый с участием таких компаний как Cray, Nvidia, PGI и другие [49]. Стандарт описывает набор директив компилятора, предназначенных для упрощения создания параллельных программ с учётом оптимизации как CPU, так и GPU (главная его особенность). Директивы поддерживаются набором свободных компиляторов языка C – gcc [58]. В рамках работы используется компилятор от фирмы PGI (доступный бесплатно для научной среды) [59].

Основные компании, разрабатывающие стандарт OpenMP активно участвуют и в разработке настоящего стандарта он во многом схож с OpenMP, в чём и заключается его преимущество, по сравнению с низкоуровневой CUDA, которая является более сложным средством оптимизации вычислений.

Для оптимизации вычислений с помощью OpenACC нет необходимости коренным образом изменять алгоритм вычислений, достаточно лишь добавить директивы компилятора к участкам кода (как и в случае OpenMP), в которых необходима оптимизация, после чего необходимо произвести компиляцию. Полный перечень директив, а также их описание приведено в [60]. Результаты оптимизации приведены в таблицах 3.7 и 3.8.

3 Анализ результатов оптимизации

3.1 Результат оптимизации с помощью математических библиотек

Результаты оптимизации алгоритмов с помощью математических библиотек приведены в таблицах 3.1, 3.2

Таблица 3.1 – Результат оптимизации с помощью библиотеки BLAS(C)

№ п/п	Критерий	Значение
1	Время вычисления, с	4,22308
2	Объём потребляемой памяти,	13
3	Объём дискового пространства, %	–
4	Прирост производительности	в 2,03 раза

Таблица 3.2 – Результат оптимизации с помощью библиотеки BLAS(Fortran)

№ п/п	Критерий	Значение
1	Время вычисления, с	4,12157
2	Объём потребляемой памяти,	13
3	Объём дискового пространства, %	–
4	Прирост производительности	в 2,08 раза

Использование математической библиотеки BLAS для оптимизации даёт прирост в производительности в 2 раза. В реальной практике такой способ оптимизации вычислений используется в комбинации с каким-либо другим, с учётом чего можно сделать вывод, что если использовать разработанный алгоритм совместно с каким-либо другим методом, то можно получить больший прирост производительности. В работе данный способ не используется, т.к. такой способ не позволяет оценить результат оптимизации конкретного средства.

3.2 Результат оптимизации с помощью CPU

Результат оптимизации с помощью Open MPI приведён в таблице 3.3. Результаты оптимизации алгоритмов с помощью OpenMP приведён в таблице 3.4.

Таблица 3.3 – Результат оптимизации с помощью Open MPI

№ п/п	Критерий	Значение
1	Время вычисления, с	10,76923
2	Объём потребляемой памяти,	18
3	Объём дискового пространства, %	–
4	Ухудшение производительности	в 1,26 раза

Таблица 3.4 – Результат оптимизации с помощью OpenMP

№ п/п	Критерий	Значение
1	Время вычисления, с	3,53846
2	Объём потребляемой памяти,	18
3	Объём дискового пространства, %	–
4	Прирост производительности	в 2,42 раза

Как и было предположено, результат многопроцессорной оптимизации хуже, чем результат многопоточной оптимизации. В случае многопроцессорной оптимизации наблюдалось снижение производительности относительно контрольного алгоритма, что вызвано малым количеством вычислительных процессоров и как следствие замедлением процесса вычисления, чего не наблюдается в случае многопоточной оптимизации.

3.3 Результат оптимизации с помощью GPU

Результаты оптимизации с помощью CUDA для NVidia GeForce GT610 и NVidia GeForce GTX970 приведены в таблицах 3.5 и 3.6 соответственно. Результаты оптимизации с помощью OpenACC для NVidia GeForce GT610 и NVidia GeForce GTX970 приведены в таблицах 3.7 и 3.8 соответственно.

Таблица 3.5 – Результат оптимизации с помощью CUDA (NVidia GeForce GT610)

№ п/п	Критерий	Значение
1	Время вычисления, с	2,76154
2	Объём потребляемой памяти,	12
3	Объём дискового пространства, %	–
4	Прирост производительности	в 3,10 раза

Таблица 3.6 – Результат оптимизации с помощью CUDA (NVidia GeForce GTX970)

№ п/п	Критерий	Значение
1	Время вычисления, с	1,51537
2	Объём потребляемой памяти,	20
3	Объём дискового пространства, %	–
4	Прирост производительности	в 5,65 раза

Таблица 3.7 – Результат оптимизации с помощью OpenACC (NVidia GeForce GT610)

№ п/п	Критерий	Значение
1	Время вычисления, с	2,74324
2	Объём потребляемой памяти,	12
3	Объём дискового пространства, %	–
4	Прирост производительности	в 3,12 раза

Таблица 3.8 – Результат оптимизации с помощью OpenACC (NVidia GeForce GTX970)

№ п/п	Критерий	Значение
1	Время вычисления, с	1,55385
2	Объём потребляемой памяти,	20
3	Объём дискового пространства, %	–
4	Прирост производительности	в 5,51 раза

В процессе оценки производительности были использованы 2 разные GPU. В результате использования устаревшей GPU удалось достичь прироста производительности в 3 раза, что можно объяснить малой вычислительной мощностью GPU. В случае применения актуальной GPU прирост производительности составил более чем в 5 раз, с учётом того, что остальная система не была подвергнута изменениям это является хорошим показателем. С другой стороны – эффективность оптимизации устаревшей GPU также является хорошим результатом, если сравнить его с результатом оптимизации с помощью GPU (CPU по техническим характеристикам, если уместно производить сравнение, превосходит GPU в несколько раз) и показывает достойный уровень снижения потребления системных вычислительных ресурсов.

3.4 Выводы

Используемый в работе тестовый стенд не обладает хорошими характеристиками как с точки зрения CPU, так и с точки зрения GPU, но даже с таким оборудованием можно достичь хороших результатов снижения потребления вычислительных ресурсов.

С точки зрения сложности, оптимизация с помощью CPU более простая по реализации, но менее эффективная, т.к. существующие возможности CPU по скорости вычислений практически исчерпаны, в то время как вычислительные возможности GPU ещё не исчерпали себя.

Оптимизация с помощью GPU при прочих равных является более эффективной как с экономической, так и с практической точки зрения (соотношение сложности оптимизации к затрачиваемому времени на оптимизацию).

OpenACC при всей его простоте позволяет достичь оптимизации вычислений не хуже CUDA и при использовании подобающего оборудования можно достичь большего снижения использования системных вычислительных ресурсов.

В работе использована для совместимости старый стандарт API CUDA (версии 2.1). В настоящее время существует версия 5.2, обладающая большими возможностями [39], использование особенностей которого позволит улучшить прирост производительности в несколько раз. Также стоит упомянуть про численный тип используемый в процессе вычислений – double, число с двойной точностью, в то время как CUDA работает в основном с типом float (число с одинарной точностью). Адаптация разработанных алгоритмов под соответствующий тип позволит получить также прирост производительности вычислений.

Использование специализированных математических библиотек при минимальных трудозатратах также позволяет получить достойные результаты оптимизации вычислений. Главным достоинством такого способа является возможность встраивания такого рода оптимизации в способы, использующие специальные средства CPU или GPU.

Таким образом, основным выводом является то, что не даже не обладая большими вычислительными ресурсами физически можно достичь хороших результатов снижения потребляемых ресурсов при решении задач численной оптимизации путём комбинирования рассмотренных выше способов.

Заключение

Рассмотрена математическая модель описывающая динамику пучка частиц в канале с ПОКФ. Произведено моделирование динамики пучка частиц в канале с ПОКФ. Для рассматриваемой модели разработано программное обеспечение с целью исследования следующих способов оптимизации вычислительных алгоритмов:

- специализированных математических библиотек;
- технологий оптимизации CPU: OpenMP и OpenMPI;
- технологий оптимизации GPU: CUDA и OpenACC.

Произведена оптимизация разработанного программного обеспечения с помощью озвученных выше способов, в результате удалось улучшить производительность в 1 – 5 раза. На основании анализа полученных характеристик эффективности оптимизации сделаны выводы относительно каждого из рассматриваемых способов минимизации использования системных вычислительных ресурсов.

И в завершении стоит отметить, что результаты проделанной работы по минимизации использования системных вычислительных ресурсов могут быть легко адаптированы к другим задачам оптимизации, требующим повышения производительности или снижения потребления системных вычислительных ресурсов.

СПИСОК ЛИТЕРАТУРЫ

[1] Расчет параметров радиально сходящегося электронно-ионного пучка в цилиндрическом импульсном источнике / Алцыбеев В.В., Пономарев В.А., Овсянников Д.А. и др. // материалы Международной конференции “Устойчивость и процессы управления”. 2015.

[2] Dynamics of Subcritical Reactor Driven by Proton Linac / A. Golovkina, I. Kudinovich, D. Ovsyannikov et al. // Proceedings of 2015 International Congress on Advances in Nuclear Power Plants. 2015. P. P. 1912–1917.

[3] Vladislav V. Altsybeyev, Dmitri A. Ovsyannikov Optimization of Beam Parameters in APF Channel // 27rd International Linear Accelerator Conference, LINAC 2014 - Proceedings, 2014. 2014.

[4] Dynamics of Processes in Subcritical Reactor Driven by Linear Accelerator / A. Golovkina, I. Kudinovich, D. Ovsyannikov et al. // Proceedings of 24th Russian Particle Accelerator Conference (RuPAC). Obninsk, Russia: 2014. P. 467–469.

[5] Ovsyannikov A. Mathematical control models in problems of beam dynamics optimization // Abstracts of International Conference dedicated to the 90th Anniversary of Academician N.N. Krasovskii. Ekaterinburg: 2014. P. P. 253–254.

[6] Numerical simulations of the radial electron flow formation for the triode type source / V. Altsybeyev, A. Ovsyannikov, D. Ovsyannikov et al. // IEEE 10th International Vacuum Electron Sources Conference / IVESC. Ekaterinburg: 2014. p. P. 13.

[7] Алцыбеев В. В. Овсянников Д. А. Управление пучком заряженных частиц с учётом их взаимодействия // XII Всеросийское совещание по проблемам управления ВСПУ. 2014. С. 2141–2149.

[8] Data processing in nuclear medicine / E. Kotina, D. Ovsyannikov, V. Ploskikh et al. // Cybernetics and Physics. Ekaterinburg, 2014. Vol. 3, no. 2. p. 55–61.

[9] Power Plant Based on Subcritical Reactor and Proton LINAC / A. Golovkina, I. Kudinovich, D. Ovsyannikov et al. // Abstracts of 5th International Particle Accelerator Conference. Dresden: 2014. p. 201.

[10] Н.С. Едаменко, Д.А. Овсянников. Моделирование динамики пучков заряженных частиц // Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика, информатика, процессы управления. №. 2. Санкт-Петербург, Россия: 2013. С. 61–65.

[11] В.В. Алцыбеев, Д.А. Овсянников. Об оптимизации структуры с фокусировкой ускоряющим полем // Процессы управления и устойчивость: Труды 44-й международной научной конференции аспирантов и студентов. Санкт-Петербург, Россия: 2013. С. 130–135.

[12] Д.А. Овсянников. О проблеме оптимизации динамики пучков взаимодействующих частиц // Материалы конференции “Управление в технических, эргатических, организационных и сетевых системах”. Санкт-Петербург, Россия: 2012. С. 1941–1946.

[13] Ovsyannikov A., Ovsyannikov D. New approach to optimization of RFQ radial matching section // Proceedings of IPAC’10, Kyoto, Japan. Kyoto, Japan: 2010. P. P. 1351–1353.

[14] Drivotin O. I. Covariant formulation of the Liouville and the Vlasov equations // ArXiv e-prints. 2016. Jan. 20 p.

[15] Drivotin O. I., Vlasova K. Numerical optimization of RFQ channel // Conference: 2014 20th International Workshop on Beam Dynamics and Optimization (BDO). 2014. June. DOI: 10.1109/BDO.2014.6890012.

[16] Drivotin O. I. Degenerate Solutions of the Vlasov Equation // Conference: 23 Russian Particle Acceleration Conference RUPAC 2012. 2012. Sept. 24-28. St.-Petersburg, Russia, http://accelconf.web.cern.ch/accelconf/rupac2012/papers/tuppb028.pdf.

[17] Drivotin O. I. Covariant Formulation of the Vlasov Equation // Conference: International Particle Accelerators Conference

IPAC2011. 2011. Sept. 4-9. San-Sebastian, Spain, ,2011, At http://accelconf.web.cern.ch/AccelConf/IPAC2011/papers/wepc114.pdf.

[18] Drivotin O. I., A. O. D. Self-Consistent Distributions for Charged Particle Beam in Magnetic Field // International Journal of Modern Physics A (Impact Factor: 1.7). 2009. Feb. Vol. 24, no. 05. P. 816–842. DOI: 10.1142/S0217751X09044322.

[19] Drivotin O. I., A. O. D. Particle distributions for beam in electric field // Particle Accelerator Conference, 1999. Proceedings of the 1999, Volume: 3. 1999. DOI: 10.1109/PAC.1999.794283.

[20] О.И. Дривотин, Д.А. Овсянников. Методы анализа самосогласованных распределений для пучков заряженных частиц. СПб: ВВМ, 2013. 115 с.

[21] Д.А. Овсянников, О.И. Дривотин. Моделирование интенсивных пучков. СПб: СПбГУ, 2003. 173 с.

[22] Д.А. Овсянников, Ю.А. Свистунов. Моделирование и оптимизация динамики заряженных частиц в ускорителях. СПб: СПбГУ, 2003. 102 с.

[23] Д.А. Овсянников, Н.В. Егоров. Математическое моделирование систем формирования электронных и ионных пучков. СПб: СПбГУ, 1998. 198 с.

[24] О.И. Дривотин, Д.А. Овсянников. Самосогласованные распределения для пучков заряженных частиц. СПб: НИИХ СПбГУ, 2001. 108 с.

[25] И.М. Капчинский. Теория линейных резонансных ускорителей: динамика частиц. Л.: Энергоиздат, 1982. 228 с.

[26] Wiedemann H. Particle Accelerator Physics. 4 edition. Springer-Verlag Berlin Heidelberg, 2015. 1021 p. ISBN3319183176, 9783319183176.

[27] Д.А. Овсянников, И.Д Рубцова, В.А Козынченко. Некоторые проблемы моделирования интенсивных пучков заряженных частиц в линейных ускорителях. СПб: ВВМ, 2013. 144 с.

[28] А.С Рошаль. Математические методы управления пучками. М.: Атомиздат, 1979. 224 с.

[29] Р. Хокни, Дж. Иствуд. Численное моделирование методом частиц. М.: Мир, 1987. 640 с.

[30] М.Ю. Балабанов, В.А. Козынченко, Н.И. Петров. Моделирование пучков заряженных частиц на высокопроизводительном комплексе // Тезисы 16 международного семинара: динамика пучков и оптимизация. СПб: ВВМ, 2010. с. 18.

[31] М.Ю. Балабанов. О выборе начального управления в задачах оптимизации динамики пучков заряженных частиц // Вестн. С. -Петерб. ун-та Сер. 10: Прикладная математика, информатика, процессы управления. 2010. № 3. С. 82–93.

[32] М.Ю. Балабанов. Принципы организации асимметричных распрс-делебнных вычислительных систем для решения задач сверхбольшой вычислительной сложности // Вестник КГТУ. 2007. С. 105–108.

[33] Bass L., Clements P., Kazman R. Software Architecture in Practice (3rd Edition) (SEI Series in Software Engineering). Addison-Wesley Professional; 3 edition (October 5, 2012), 2012. 640 p.

[34] Эванс Эрик. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. Вильямс, 2010. 448 с.

[35] Моделирование и оптимизация динамики заряженных частиц в ускорителях / Кормен Т.А., Лейзерсон Ч.И., Ривест Р.Л. и др.; под ред. 2-е издание Алгоритмы: построение и анализ. М.: «Вильямс», 2005. ISBN 5-8459-0857-4.

[36] В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. СПб: БХВ-Петербург, 2002. 608 с. ISBN 5-94157-160-7.

[37] Дж Сандерс, Э. Кэндрот. Технология CUDA в примерах: введение в программирование графических процессоров. М.: ДМК Пресс, 2015. 232 с. ISBN 978-5-97060-297-3.

[38] Официальный сайт стандарта OpenMP. — URL: http://openmp.org.

[39] Cuda-zone - официальный ресурс посвящённый разработке с использованием Cuda-toolkit. — URL: https://developer.nvidia.com/cuda-zone.

[40] Официальный сайт стандарта Open MPI. — URL: https://www.open-mpi.org/.

[41] Ю. Магда. Использование ассемблера для оптимизации программ на C++ (+ CD). СПб: БХВ-Петербург, 2003. 496 с. ISBN: 5-94157-414-2.

[42] С.В. Зубков. Assembler. Для DOS, Windows и Unix. М.: ДМК Пресс, 2013. 640 с. ISBN 978-5-94074-900-4.

[43] О.И. Дривотин. Математические основы теории поля. СПб: Изд-во СПбГУ, 2010. 200 с. ISBN 978-5-288-05124-1.

[44] Л.Д. Ландау, Е.М. Лившиц. Теория поля. М.: Наука, 1967. 460 с.

[45] А. Лихтенберг. Динамика частиц в фазовом пространстве. М.: Атомиздат, 1972. 304 с.

[46] В.А. Ильина, П.К. Силаев. Численные методы для физиков-теоретиков. Москва-Ижевск: Институт компьютерных исследований, 2004. №. 2.

[47] R.L. Nelson and H. Kwan, "Leapfrog"methods for numerical solution of differential equations, sinewave generation, and magnitude approximation,"Signals, Systems and Computers, Conference Record of the Twenty-Ninth Asilomar Conference on, Pacific Grove. №. 2, CA, USA: 1995. doi: 10.1109/ACSSC.1995.540811.

[48] BLAS. Официальный сайт — URL: http://www.netlib.org/blas/.

[49] Официальный сайт стандарта OpenACC. — URL: http://www.openacc.org/.

[50] Matlab - многофункциональный математический пакет. — URL: http://matlab.ru/.

[51] CST: PS - Система автоматизированного проектирования ускорителей частиц и расчёта их характеристик. — URL: https://www.cst.com/Products.

[52] Klein H. Development of the different RFQ accelerating structures and operation experience // Nuclear Science, IEEE Transactions on. 1983. Vol. 30, no. 4. P. 3313–3322.

[53] Р. Либов. Введение в теорию кинетических уравнений, пер. с англ. М.: Мир, 1974. 372 с.

[54] А.А. Власов. О вибрационных свойствах электронного газа // Журнал экспериментальной и теоретической физики. 1938. №. 8, № 3. с. 291.

[55] А.А. Власов. Теория вибрационных свойств электронного газа и её приложения // Уч. зап. МГУ. 1945. №. 75, № 2.

[56] Open MPI публикации. — URL: https://www.open-mpi.org/papers/.

[57] Библиотека intel math kernel library (intel mkl). — URL: http://software.intel.com/ru-ru/intel-mkl/.

[58] Справка компилятора gcc по возможности имплементации OpenACC. — URL: https://gcc.gnu.org/wiki/OpenACC.

[59] Официальный сайт OpenPGI. — URL: http://www.pgroup.com/index.htm.

[60] API OpenACC. — URL: http://www.openacc.org/sites/default/files/213462

Приложение № 1

Исходный код реализации метода РК

```
#include <stdio.h>

double funcX(double, double);
double funcZ(double, double);
const double one_sixth = 1/6;

double funcX(double t, double x)
{
    return x;
}

double funcZ(double t, double z)
{
    double omeg = 1.0;
    return -omeg*omeg*z;
}

double Runge4p(double x, double y,double h,double f(double,double))
{
    double k1,k2,k3,k4,rez;

    k1=h*f(x,y);
    k2=h*f(x+h/2.0,y+k1/2.0);
    k3=h*f(x+h/2.0,y+k2/2.0);
    k4=h*f(x+h,y+k3);
    return(y+(k1+2*k2+2*k3+k4)/6);
}


 void Runge_Kutta_2nd_Order( double f(double, double, double),
  double x0, double y[],
  double c, double h, int number_of_steps )
```

```
 {
    double k1, k2, k3, k4;
    double h2 = 0.5 * h;
    double ych2;
    uint i;

    for (i = 0; i < number_of_steps; x0 += h, i++)
    {
        ych2 = y[i] + c * h2;
        k1 = h * (*f)(x0, y[i], c);
        k2 = h * (*f)(x0+h2, ych2, c + 0.5 * k1);
        k3 = h * (*f)(x0+h2, ych2 + 0.25 * k1 * h, c + 0.5 * k2);
        k4 = h * (*f)(x0+h, y[i] + c * h + h2 * k2 , c + k3);
        y[i+1] = y[i] + ( c + one_sixth * (k1 + k2 + k3) ) * h;
        c += one_sixth * ( k1 + k2 + k2 + k3 + k3 + k4 );
    }
}
```

Исходный код реализации метода leapFrog

```
#include <stdio.h>
#include <math.h>

#define K 4
//a = dv/dt, given x
double double(real x)
{
    return -K*x;
}
// offset the velocity
void advVel(double x, double *v, double dt2)
{
    *v += dt2*deriv(x);
}
// calc energy
double energy(double x, double v)
{
    return 0.5 * (K*x*x + v*v);
}

static double e0;

// print
void output(double x, double v, double t, double dt)
{
    if (t > 0)
    {
        advVel(x, &v, -0.5*dt);
    }
    printf("%f %f %f %f\n", t, x, v, energy(x, v) - e0);
}

void leapfrog(double *x, double *v, double *t, double dt)
```

```
{
    *x += dt * (*v);
    *v += dt * deriv(*x);
    *t += dt;
}
```