МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Системное программирование и компьютерные технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Моделирование систем эстафетной схемы с использованием распараллеленного алгоритма»

Студент А.О. Ефимова _____
(И.О. Фамилия) (личная подпись)

Руководитель доктор ф.-м. наук, профессор, А.И. Сафронов
(Ученая степень, звание, И.О. Фамилия)

Консультант К.А. Селиверстова
(Ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Бакалаврская работа включает: 50 страниц, 36 рисунков, 2 таблицы, 3 раздела, 22 использованных источников.

Цель работы – увеличить скорость обработки входных параметров и получение выходных данных при математическом расчете в эстафетной схеме метания элемента.

Объект исследования – математические расчеты, необходимые для получения выходных данных при различных входных параметрах.

Предмет исследования – параллельные алгоритмы, повышающие скорость вычислений.

Полученные результаты – разработан и реализован распараллеленный алгоритм математической модели эстафетной схемы.

Рекомендация внедрения результатов работы – полученный алгоритм с применением технологии распараллеливания может быть в дальнейшем использован для работы с математической моделью эстафетной схемы, при экономии времени и ресурсов, затрачиваемых на выполнение расчетов.

Область применения – разработанный алгоритм возможно применять при проведении различных математических расчетов в компьютерном моделировании.

Значимость работы – позволяет наиболее эффективно использовать ресурсы центрального процессора и ускоряет процесс вычислений.

Abstract

The title of the graduation work is « Relay circuit simulation modeling using parallelization algorithm».

The graduation work consists of an explanatory note on 50 pages, introduction, three parts including 36 illustrations, 2 tables, the list of 22 references including 3 foreign sources.

The aim of this work is to increase the input parametres processing and to receive output data speed with the use of mathematical calculations in the relay throwing element.

The object of the study is mathematical calculations, which are necessary for receiving output data in terms of various input data.

The subject of the study is increasing calculation speed parallel algorithms.

The graduation work is divided into several logically connected parts which include theoretical information, design of algorithms and their implementation, comparative analysis of algorithms.

The result of the graduation work is the designed and implemented parallel algorithm of mathematical relay model. It is highly recommended to use this parallel algorithm in mathematical relay model, which will save time and resources, spent on calculating

The results of this work can be applied in various math calculations in computer modelling using this algorithm. The relevance of this work lies in using the resource of CPU more effectively that increases calculation time.

Содержание

Введение

В современном мире одним из важных разделов физики является внутренняя баллистика, которая стала инструментом в разработке и реализации различных систем вооружения. Данный раздел способен ответить на ряд решающих вопросов при создании нового продукта, а также при усовершенствовании старого. Решение каждой задачи при рассмотрении модернизаций и создании продуктов требует мощных математических расчетов, занимающих большое количество времени. Для повышения точности при проведении подсчетов и ускорения данного процесса, проводятся различные исследования по созданию программных продуктов для математических расчетов.

Во внутренней баллистике существуют различные схемы, требующие подробного изучения. Для рассмотрения, в данной выпускной квалификационной работе была выбрана одна из них – эстафетная схема метания. Данная схема интересна своим решением в вопросе повышения скорости метаемого элемента, по которому заряд патрона делится на две или три части. При разделении заряда на две части патрон называют бинаром, на три части – тринаром.

В простом варианте рассмотрения данной схемы с бинаром, используют пластмассовую гильзу с воспламенителем, в которую засыпают основную часть пороха, а сверху устанавливают картонную диафрагму, разделяющую на две части патрон, где во вторую часть засыпают дополнительную часть пороха из того же или другого вещества.

При данном подходе необходимо проводить множество различных расчетов, связанных с варьированием величиной частей пороха, расположением диафрагмы, материалами метаемых элементов и многих других параметров.

Для улучшения работы над данными расчетами проводится множество исследований в разработке приложений, решающих ряд вопросов с точностью и затраченным временем. На данный момент, существуют разработанные на

различных языках программирования приложения, использующие разные виды технологий.

Большую часть существующих разработанных алгоритмов занимают последовательные, однако благодаря тенденциям использования всех возможных ресурсов персональных компьютеров, можно заметить, что актуальным также становится и алгоритм с многопоточным программированием, с использованием распараллеливания.

Существующая на данный момент модель эстафетной схемы в последовательной реализации хорошо выполняет поставленные перед ней задачи, а также затрачивает относительно малое время для расчетов. Для уменьшения затраченного времени также были проведены исследования с использованием технологии OpenCL на языке C, которая предназначена для распараллеливания алгоритма по средствам графического и центрального процессоров. Большие плюсы в данной технологии, как и в технологии Cuda для C/C++, что все математические расчеты, загружающие центральный процессор, как элементарные задачи уходят на графический, благодаря чему CPU может выполнять весь остальной алгоритм. Благодаря такому распределению, мы получаем как минимум 2 параллели с мощными ресурсами для вычислений.

В данной выпускной квалификационной работе будут рассматриваться последовательный алгоритм на языке C++ и алгоритм с использованием технологии OpenMP, что позволит решить существующую проблему вычислений при использовании эстафетной схемы в баллистике, не затрагивая графический процессор.

Объект исследования – математические расчеты, необходимые для получения выходных данных при различных входных параметрах.

Предметом исследования являются – параллельные алгоритмы, повышающее скорость вычислений.

Цель исследования – получение более высокой скорости обработки входных параметров и получение выходных данных при математическом расчете.

Задачи исследования:

- рассмотреть теоретические сведения по исследуемому объекту;
- рассмотреть существующие решения проблемы и провести их анализ;
- спроектировать последовательный алгоритм для расчетов;
- спроектировать алгоритм с технологией распараллеливания для расчетов;
- провести анализ эффективности двух алгоритмов;

В первом разделе данной выпускной работы будут рассмотрены все теоретические аспекты, связанные с математическими расчетами и моделью эстафетной схемы. Во втором разделе будет приведено описание работы над алгоритмами. Заключительным звеном будет являться сравнение двух алгоритмов (последовательного и с использованием OpenMP) и выявление их положительных и отрицательных сторон.

1 Теоретические сведения

1.1 Характеристика объекта исследования

Наука, описывающая движение метаемого элемента в канале ствола, а также рассматривающая все процессы, его сопровождающие, называется внутренней баллистикой. Все расчеты, включая действие пороховых газов и установление других закономерностей при выстреле, непосредственно связывают в себе несколько областей научных знаний, таких как химия, механика, термодинамика, газодинамика, термохимия и другие.

В настоящее время, к инструментам для расчетов всех процессов во внутренней баллистике присоединились высшая математика и информационные технологии, позволяющие создать модель и рассчитать все выходные результаты исходя из начальных параметров, не привлекая физических экспериментов.

Благодаря применению математического и компьютерного моделирования, все необходимые опыты возможно просчитывать без материальных затрат и другого оборудования, что является одним из стимулов к созданию наиболее точных алгоритмов для подсчета выходных значений по известным данным.

Для рассмотрения модели выстрела метаемым элементом из ствола необходимо знать основы сложного процесса превращения химической энергии пороха при горении в тепловую энергию газов, а затем в механическую энергию движения снаряда и других подвижных частей ствола, что можно также назвать механической работой.

Таким образом, расчеты для внутренней баллистической модели можно разделить на временные периоды, исходя из самого процесса выстрела [6]:

1. Предварительный период. В этот промежуток времени рассматривается момент воспламенения пороха, проводятся химические расчеты, связанные с горением.

2. Первый (основной) период. Его начало совпадает с окончанием врезания ведущих поясков, а конец наступает с прекращением горения пороха.

3. Второй период. Следующий сразу за основным и продолжающийся до момента вылета метаемого элемента из ствола.

4. Период последействия газов. Промежуток времени, соответствующий моменту вылета метаемого элемента из ствола до окончания действия пороховых газов на него после выстрела.

Исходя из вышеперечисленных периодов внутрибаллистического процесса, можно отметить такие источники, как [5]-[9]:

- пиростатика – раздел внутренней баллистики, изучающий горение пороха и вытекающие аспекты газообразования при условии изохорного процесса (процесса, сопровождающегося неизменным объемом); в данном разделе устанавливаются особенности горения порохов различных составов;
- пиродинамика – раздел внутренней баллистики, изучающий процессы и явления в канале ствола при выстреле, устанавливающий связи между особенностями характеристик ствола, начальными условиями метаемого элемента и физико-химическими процессами в механике;
- баллистическое проектирование – совокупность этапа моделирования и сборки орудия; раздел, отвечающий за физические расчеты по разработке, реализации и сборке частей орудия.

На рисунке 1 изображены зависимости давления и скорости от времени, а также разделены периоды, начиная от предварительного, связанного с воспламенением пороха, заканчивая вылетом из ствола метаемого элемента, сопровождающегося действием на него пороховых газов (период последействия). На данной картинке показан общий случай выстрела во внутренней баллистике без разделения метаемого элемента.

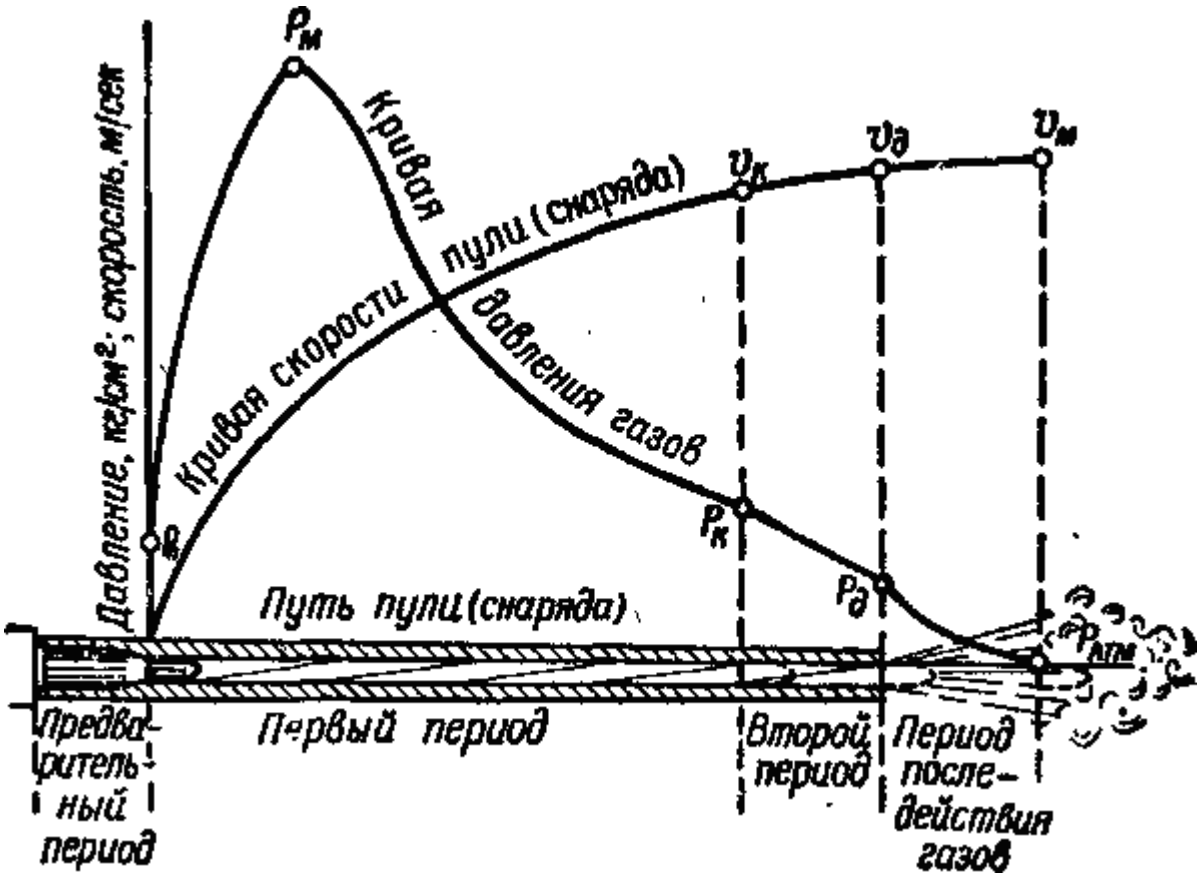

Рисунок 1 – Визуальное представление выстрела

Как показано на рисунке 1, максимальное давление пороховых газов внутри ствола достигается в основном периоде, а максимальной скорость метаемого элемента становится в тот момент, когда давление снижается до минимального.

Для моделирования эстафетной схемы, суть которой заключается в использовании разделения основного метательного заряда на две или более частей, при применении модели гетерогенных сред, можно рассматривать расчетную схему, показанную на рисунке 2.

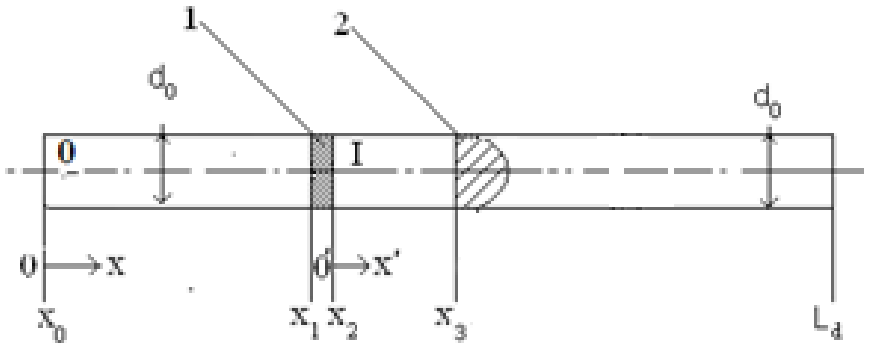

Рисунок 2 – Расчетная схема эстафетного выстрела

На данном рисунке изображена схема бинар, при которой метающий пороховой заряд разделен на две части диафрагмой. Первая область – 0, имеющая длину x_0x_1; вторая область - I, имеющая длину x_2x_3; 1 – диафрагма, имеющая толщину x_1x_2; 2 – пуля (метаемый элемент); d0 – диаметр цилиндрического канала системы; Ld – длина ствола с учётом длины патронника.

Для гетерогенных сред, классическим случаем является движение горящих и, в общем случае, еще не воспламенившихся пороховых элементов по каналу ствола во время выстрела. Данным вопросом занимались многие авторы в своих трудах, такие как М. А. Гольдштик, С. Соу, А. А. Шрайбер, В. Н. Николаевский, Р. И. Нигматулин, Л. Е. Стернин, А. Н. Крайко и многие другие. На практике, наиболее распространенными оказались концепции представления различных фаз гетерогенных потоков в виде отдельных взаимопроникающих сплошных сред, сформулированные в 1956 – 1957 годах Х. А. Рахматулиным и С. Трусделлом и обобщенные в работах А. Н. Крайко, Л. Е. Стернина, Р. И. Нигматулина.

В описанных трудах различных авторов, были приняты следующие допущения:

– параметры течения изменений принимаются только тех расстояниях, которые намного больше чем размер частиц или расстояние между ними;

– всякая фаза будет являться частью обмена смеси, а во всех точках пространства одновременно существуют различные фазы;
– вычислить движение фазы самостоятельно от смеси можно только с помощью учета взаимодействия других фаз;
– теплопроводность, как и вязкость, существенная только при взаимодействии фаз;
– если частицы одинаковых размеров сталкиваются, то такими столкновениями можно пренебречь;

Также стоит отметить, что многие ученые занимались теориями движений гетерогенных нереагирующих сред. Примерами авторов работ, где, по-видимому, впервые были учтены фазовые переходы являются В.Н. Вилюнов [5], опубликовавший свои труды в 1964 году и Р. И. Нигматулин, с работой 1967 года. Я. А. Каневский, К. П. Станюкович и Дж. Корнер еще в начале 50-х годов 20-го века принимали попытки учета двухфазного характера течения.

В 1969 году были опубликованы труды В. Н. Вилюнова, которые в своем содержании доказывают повышение скорости метания при переходе от классической схемы выстрела (КСВ) к схеме типа «порох – поршень – порох» (ППП), которая стала одним из первых прототипов создания эстафетной схемы метания.

По историческим фактам можно сказать, что развитие работ в модернизации внутренней баллистики с 50-х годов 20-го века привело к эстафетной схеме, описывающейся в данной работе.

Для построения модели эстафетной схемы на основе подхода гетерогенных сред были приняты следующие допущения:

– момент времени зажигания метаемого элемента считается общим для всех его частей;
– движение диафрагмы, метательного и дополнительного элементов начинается в момент достижения давления форсированной сборки;
– пренебрегаем силой трения в стволе и сопротивлением воздуха;

– геометрический закон горения справедлив для всех пороховых частиц частей метаемого заряда;
– дополнительная часть метаемого заряда до момента разделения неподвижна, относительно движущейся сборки;
– пренебрегается перетоком газов между областями 0 и I.

В области 0 располагается часть разделенного метательного заряда. Записав уравнения модели гетерогенных сред в неинерциальной системе координат можно удобно описывать процессы, протекающие в области 1, движущейся с ускорением.

1.2 Математическая модель описания процессов

Модель эстафетной схемы во внутренней баллистике основывается на вычислениях, произведенных с использованием высшей математики, включающей в себя курс дифференциальных уравнений, математического анализа, численных методов и других разделов.

При рассмотрении начальных и граничных условий, можно использовать систему уравнений 1:

$$\begin{cases} \frac{\partial}{\partial t}(\rho s\varphi) + \frac{\partial}{\partial x}(\rho u s\varphi) = M; \\ \frac{\partial}{\partial t}(\rho s\varphi u) + \frac{\partial}{\partial x}(\rho s\varphi u^2 + \rho s\varphi) = M\omega - \tau_{\text{ТР}} + p\frac{\partial s\varphi}{\partial x} - N\rho S\varphi\frac{du_D}{dt}; \\ \frac{\partial}{\partial t}(\rho S\varphi E) + \frac{\partial}{\partial x}\left(S\varphi u(\rho E + p)\right) = \\ = -p\frac{\partial(1-\varphi)S\omega}{\partial x} - \tau_{\text{ТР}}\omega + M\left(Q + \frac{\omega^2}{2}\right) - N\rho S\varphi u\frac{du_D}{dt}; \\ \frac{\partial}{\partial t}(\rho_2(1-\varphi)S) + \frac{\partial}{\partial x}(\rho_2(1-\varphi)S\omega) = -M; \\ \frac{\partial}{\partial t}(\rho_2(1-\varphi)S\omega) + \frac{\partial}{\partial x}(\rho_2(1-\varphi)S\omega^2) + (1-\varphi)S\frac{\partial p}{\partial x} = \\ = \tau_{\text{ТР}} - M\omega - N\rho_2(1-\varphi)S\frac{du_{\text{П}}}{dt}; \\ \frac{\partial z}{\partial t} + \omega\frac{\partial z}{\partial x} = \frac{a_1 p}{e_b}; \\ P\left(\frac{1}{\rho} - \alpha\right) = RT. \end{cases} \quad (1)$$

Как можно увидеть из системы дифференциальных уравнений, при значении N = 0, уравнение описывает процессы в области 0, при N = 1, расчеты производятся для области 1. Решение данной системы уравнений после воспламенения присоединенного метательного заряда будет последовательно проводиться для областей 0 и 1.

Также, для проведения расчетов используются соотношения 2.

$$\begin{cases} E = \varepsilon + \frac{u^2}{2}; \\ \varphi = 1 - n\Lambda_0\left(1 - \varphi(z)\right); \\ \psi(z) = k_1 z(1 + \lambda_1 z); \\ M = SnS_{02}\rho_2\sigma(z)a_1 p; \\ \sigma(z) = 1 + 2\lambda_1 z; \\ \tau_{\text{ТР}} = \frac{1}{2}C_x\rho(u-\omega)|u-\omega|S_n\frac{\pi d_{op}^2}{4}\left(1-\psi(z)\right)^{2/3}; \\ Cx = \begin{cases} \frac{24}{Re} + 0.48, \ 0 < Re < 3\cdot 10^5; \\ 0.1, Re \geq 3\cdot 10^5; \end{cases} \\ Re = \frac{\rho|u-\omega|\varphi\sqrt{S_{02}}}{\mu}; \end{cases} \quad (2)$$

где t – время;

x – координата;

P – давление;

ρ – плотность;

ρ_2 – плотность вещества топлива;

T – температура;

T_0 – температура газов горения;

u – скорость газов;

ω – скорость частиц;

φ – пористость;

E – полная энергия единицы объема газа соответственно;

ε – внутренняя энергия единицы объемов газа;

S – площадь поперечного сечения канала;

Z – относительная толщина сгоревшего свода;

M – скорость массоприхода от горения топлива;

$\tau_{\text{ТР}}$ – сила взаимодействия между фазами;

N – признак системы координат;

du_p/dt – ускорение диафрагмы;

Q – тепловой эффект горения топлива;

R – универсальная газовая постоянная;

α – коволюм (объём молекул пороховых газов);

a_1 – коэффициент в законе скорости горения;

e_b – толщина горящего свода зерна топлива;

n – концентрация;

Λ_0 – начальный объем частицы топлива;

$\psi(z)$ – относительный сгоревший объем частиц топлива;

k_1 и λ_1 – коэффициенты формы частиц топлива;

S_{02} – начальная площадь частиц топлива;

$\sigma(z)$ – относительная горящая поверхность частицы топлива;

C_x – коэффициент сопротивления;

d_{0p} – диаметр шара квивалентного по объему частице топлива;

Re – число Рейнольдса;

μ – вязкость газа.

Таким образом, начальные условия для нулевой области описываются следующими закономерностями, как показано в системе 3:

$$\begin{cases} T(x,0) = T_G; \\ P(x,0) = P_\Phi; \\ u(x,0) = w(x,0) = 0; \\ \varphi(x,0) = \varphi_\text{H}; \\ \psi_\text{H} = \frac{\frac{1}{\Delta} - \frac{1}{\rho_2}}{\frac{f}{P_\Phi} + \alpha - \frac{1}{\rho_2}}; \\ Z_\text{H} = \frac{2\psi_\text{H}}{k_1(1+\sigma_\text{H})}; \\ \sigma_\text{H} = \sqrt{1 + 4\frac{\lambda_1}{k_1}\psi_\text{H}}. \end{cases} \quad (3)$$

Для области 1 применима система 4:

$$\begin{cases} T'(x', t_R) = T'(x'); \\ p'(x', t_R) = p'(x'); \\ u'(x', t_R) = u'(x'); \\ w'(x', t_R) = 0; \\ \varphi'(x, t_R) = \varphi'(x'); \\ z'(x', t_R) = z'(x'). \end{cases} \quad (4)$$

Для данной задачи можно выбрать граничные условия системы 5:

$$\begin{cases} u(0,t) = w(0,t) = 0; \\ u(x_D,t) = w(x_D,t) = u_D; \\ u'(0,t) = w'(0,t) = 0; \\ u'(x'_S,t) = w'(x'_S,t) = u'_S. \end{cases} \quad (5)$$

где $x_D, x'_S -$ координаты положения диафрагмы и метаемого элемента (пули) соответственно;

$u_D, u'_S -$ скорости диафрагмы и метаемого элемента соответственно.

Для того, чтобы определить значения данных переменных будем интегрировать уравнения движения сборки, а когда произойдет разделение, интегрируем уравнения движения поршня и метаемого элемента соответственно системам уравнений 6 и 7:

$$\begin{cases} m_{AS}\frac{du_D}{dt} = p_1 s \\ q_s\frac{du'_s}{dt} = 0 \end{cases}, \text{для } t \le t_R \qquad (6)$$

$$\begin{cases} m_D\frac{du_D}{dt} = (p_1 - p_2)s \\ q_s\frac{du'_s}{dt} = p_3 s - q_s\frac{du_D}{dt} \end{cases}, \text{для } t > t_R \qquad (7)$$

Таким образом, можно сделать вывод, что $p_3 > \frac{q_s}{q_D}(p_1 - p_2)$.

Для решение поставленных математических задач с граничными и начальными условиями необходимо выбрать метод численного решения. Если выбрать определяющим фактором, что потоки импульса, энергии и массы должны определяться из решения задачи распада произвольного разрыва для среды, где нет «собственного» давления, а также из решения распада произвольного разрыва параметров газа на скачке площади сечения, то наилучшим образом подойдет численный метод С. К. Годунова [11] – [12]. Исходя из данного метода, разностными сетками с шагами h и h' покрываются расчетные области 0 и 1.

При последовательном решение системы 1 полагается, что воспламеняется часть метаемого элемента в 0-ой области, значит сборка движется как единое целое. Следующим этапом будет воспламенения дополнительной части метаемого элемента в области 1. Для всех расчетов принимается масса равная 5×10^{-4} кг, или 0,5 г. При данной массе понижается давление пороховых газов в патроннике, а суммарный метаемый элемент может быть увеличен без превышения давления, допустимого для данной системы.

На рисунке 3 видно, что диафрагма и дополнительная часть с метаемым элементов будут разделяться примерно через 0.45 мс после начала процесса. В данный момент легкая диафрагма притормаживается со снижением скорости от 270 м/с до 125 м/с (в области 1), а после возрастания давления на диафрагме со

стороны области 0, скорость возрастет до 320 м/с, однако потом замедляется до 230 м/с. Так как давление за метаемым элементом упадет, то скорость диафрагмы снова повысится, а после возвращения волны уплотнения от дна канала, диафрагма ускорится, практически догоняя метаемый элемент. Таким образом, от 0.74 мс до 0.76 мс метаемый элемент приобретет скорость до 580 м/с.

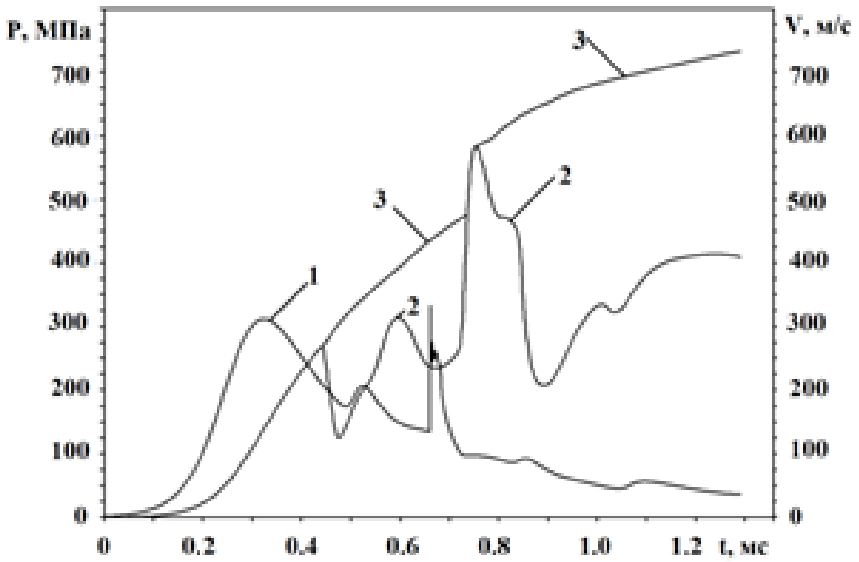

Рисунок 3 – Расчет для выстрела при эстафетной схеме

Как видно на рисунке, в последние моменты времени скорость диафрагмы снижается, однако скорость метаемого элемента увеличивается постепенно примерно до 732 м/с на дульном срезе карабина. Когда метаемый элемент покидает канал ствола, диафрагма движется с примерной скоростью 400 м/с.

Таким образом, благодаря данной математической модели для эстафетной схемы, можно путем математических расчетов увидеть, что скорость метаемого элемента будет выше при эстафетной схеме с разделением метаемого элемента на две и более части, чем при классической.

2 Проектирование алгоритмов и их реализация

2.1 Разработка последовательного алгоритма

Благодаря проведенному исследованию по теоретическим аспектам, можно построить блок-схему последовательного алгоритма, позволяющего определить время движения метаемого элемента (МЭ) по каналу ствола, его скорость, давление в областях 0 и 1, а также его положение в координатной сетке 0x.

Разработанная блок-схема последовательного алгоритма расчетов приведена на рисунке 4.

Рисунок 4 – Алгоритм расчетов по эстафетной схеме

2.2 Реализация разработанного алгоритма

Для реализации последовательного алгоритма был выбран высокий язык программирования C++, по средствам которого была построена работа в среде программирования NetBeans.

На рисунках 5 – 6 показано объявление констант и начальных условий из разработанной программы.

```
#include <stdio.h>
#include <math.h>
#include <errno.h>
// CONST
const int N_max = 192 + 1;
const int NO_max = 180 + 1;
const double PI = 3.14159256;

double BG = 0.518E-05;
double FIPR = 0.4;
double RKPR;
double KINDI = 0;
double KINDII = 0;
// INTEGER
int I, N, NO, KK, NDS, NN;
// FILE
FILE *file1, *file2;
// double with initial values
double  SIGSB = 5.68E-08,
                TIME = 0.0,
                TIM = 0.0,
                US = 0.0,
                USO = 0.0,
                PIND = 0.0,
                APCH1 = 0.0,
                APCH1O = 0.0,
                DVDT = 0.0;
double R1N,FI1N,P1N,U1N,SV1N,S11,FI11,SV11;
// double load from file
double  KZAP,KIOBL,BPCH,
                X0,X1,X2,X3,
                XPL,XPP,XS,ALKM,ALD,
                D0,D1,D2,D3,D,
                EB,B2,C2,AKP,ALP,AMP,
                NPOR,DPOR,
                AFL,FP,CV,TPG,AL,EK,
                W1,W2,QPPP,QS,
                PFF, RP,
                EBO,B2O,C2O,AKPO,ALPO,AMPO,
                NPORO,DPORO,
                AFLO,FPO,CVO,TPGO,ALO,EKO,
                PFFO,XIGN;
```

Рисунок 5 – Введение констант и переменных

```
// double
double  RGC, RGCO, QP, QPO,
                PM, CM, RM, SM, BPM, DELTA, WKM,
                SEND, QPP, QPPO, DXP, TH,
                G1, G2, G3, G4, G5, G6,
                XX1, XX2, DI1, DI2;

double  AMSB, W0, W0O, S02, S02O, KPER, KMOV, IIGN, H, HO, NLD,
                AES0, PFNA, PMAX, W2O, DELTAO,
                PSI0, SIG0, Z0, TG,
                POMAX, PONMAX, XSO, WKMSCH,
                UMAX, UMAX1, THPCH, UMAXO, UMAXO1, THO, THOPCH,
                UBB, PBB, RBB, UMM, PMM, RMM, EKB, ALB,
                DRR, USN, USON;

double  UPOUT, USOUT, XPLOUT, XSOUT, DXN, DXON, AMS, AES,
                EKG1, EBG1, EKK1, EXK1, EKG2, EBG2, EKK2, EXK2,
                ESQ1, ESB, EKP, ESQ1SB, ESQ1P, ESQ2, ESN, ESQ2SN, AESD;

double  RKP, RRK1, WW1, ARK1, AWRK1, AR1, AUR1, AER1, ZN, PST, WST, FIST,
                ARK2, AWRK2, AR2, AUR2, AER2, AR3, AUR3, AER3,
                DWSDX, PSIZ, ANI, SIGZ, GM, RE,
                RKL, RRK2, WW2, SK, UR, PR, RR, UL, PL, RL,
                CX, DP, ATR, TTR, Q, FMASG, FIMPS, FENS, FMASK,
                APCH, RSFI, RUSFI, RESFI, RKSFI, FIN, RN, UN, EN, PN, DPDX,
                KIN, RKWSFI, WN, ZN1,
                USI, PMSB, XSN, DX,
                WNEND, SNEND, pglobal;

double R[N_max], P[N_max], U[N_max], W[N_max], FI[N_max],
           Z[N_max], RO[NO_max], PO[NO_max], UO[NO_max], WO[NO_max],
           FIO[NO_max], ZO[NO_max], S[N_max],SV[N_max], SO[NO_max];

//COMMON/R/Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,Y10,P1,P2,P3,P4,P5,
//        ,P6,P7,P8,P9,P10,PSH,G7,G8,G9,GR,
//        ,U1,U2,U3,U4,U5,U6,R1,R2,R3,R4,R5,R6,A7,A8
double Y1 = R[1], Y2 = R[2], Y3 = R[3], Y4 = R[4], Y5 = R[5], Y6 = R[6],
                Y7 = R[7], Y8 = R[8], Y9 = R[9], Y10 = R[10],
                P1 = R[11], P2 = R[12], P3 = R[13], P4 = R[14], P5 = R[15], P6 = R[16],
                P7 = R[17], P8 = R[18], P9 = R[19], P10 = R[20],
                PSH = R[21],
                G7 = R[22], G8 = R[23], G9 = R[24], GR = R[25],
                U1 = R[26], U2 = R[27], U3 = R[28], U4 = R[29], U5 = R[30], U6 = R[31],
                R1 = R[32], R2 = R[33], R3 = R[34], R4 = R[35], R5 = R[36], R6 = R[37],
                A7 = R[38], A8 = R[39];
```

Рисунок 6 – Введение констант переменных

Как показано на данных рисунках, входные параметры для данной математической модели представлены типом double, означающим, что все перечисленные переменные будут хранить данные длиной 8 байт с плавающей точкой [4].

В разработанной программе будет использоваться ряд идентификатора, который необходимо определить заранее, как показано на рисунке 7.

```
void D31(double U, double P, double R, int IZN, double SK, double H2, double A2)
{
    Y1 = R*U/SK;
        if (fabs(U) >= 0.000101) goto g30;
        U3 = U;
    P3 = P;
    R3 = R;
    GR = 0.0;
        goto g31;
g30:
    PSH = P;
    Y2 = (R*U*U + P)/SK - PSH*(1.0/SK - 1.0);
    Y3 = H2*P/R - P*A2 + G4*U*U/2.0;
    Y4 = H2*Y2/Y1;
    Y5 = 2.0*G3*(A2*Y2 + Y3);
        Y6 = Y4 + A2*Y1;
        Y4 = Y6*Y6 - Y5;
        if(Y4>0) GR=+1.0;
        if(Y4<0) GR=-1.0;
        U3 = (Y6 - IZN*sqrt(fabs(Y4)))/G3;
        R3 = Y1/U3;
    P3 = Y2 - Y1*U3;
g31:
    return;
}
//---------------------------------------------------------------------------
void D4(double U, double P, double R, int IZN, double SK, double H1, double A1)
{
    if (fabs(U) >= 0.000101) goto g40;
    U4 = U;
    P4 = P;
    R4 = R;
    goto g41;
g40:
    Y1 = U*R*SK;
    Y2 = (R*U*U + P)*SK + 1.0*((1 - IZN)*(1.0 - SK)/2.0);
    Y3 = H1*P/R - P*A1 + G2*U*U/2.0;
    Y4 = H1*Y2/Y1 + Y1*A1;
    Y5 = 1.0 - (1.0 + IZN)*(1.0 - SK)/2.0;
    Y6 = (Y3*Y5 + Y2*A1)*(4*H1 - 2.0*Y5*G2);
    U4 = (Y4 + IZN*sqrt(fabs(Y4*Y4 - Y6)))/(2.0*H1 - Y5*G2);
    R4 = Y1/U4;
    P4 = (Y2 - Y1*U4)/Y5;
g41:
    return;
```

Рисунок 7 – Введение идентификаторов g**

На рисунке 7 видно, что вводимые идентификаторы являются частью функции со спецификатором void. Подобные спецификаторы используются для того, чтобы вызываемая функция не имела возвращаемых значений [1], [2].

После определения всех входных параметров, включая чтения данных из файла, согласно блок-схеме, начинаю производиться расчеты из математически построенных частей программного кода: подпрограмма расчета распада произвольного разрыва; расчет распада произвольного разрыва для среды, не имеющей собственного давления; подпрограмма счета шага течения смеси; расчет потоков; расчет параметров во внутренних ячейках. Последним блоком является подпрограмма вывода информации. Все описанные основные части программного кода реализованы с помощью функций со спецификатором отсутствия возвращаемых данных void.

После описания всех вызываемых функций и входных параметров, в программе составляется основной алгоритм работы модели - тело программы. В него входят открытие файлов со входными и выходными данными, по отношению к которым необходимо назначение особых прав.

Для работы с описанной и реализованной моделью важно сохранять входные данные неизменными, в связи с чем единственной важной функцией для файла с исходными параметрами является «rb+» (read - чтение).

При работе с выходными значениями необходимо реализовать функцию записи в файл, при этом не изменяя при следующих итерациях записи. Для этого особыми правами были назначены «wb+» (write - запись).

Для перехода к расчетам необходимо вывести все входные параметры из файла, организуя их вывод так, как будет удобно для дальнейшей работы. Как показано на рисунке 8, вывод из исходного файла будет производиться построчно.

```
int main ()
{
        unsigned int start_time =  clock();

        filel=fopen("DAN","rb+");
        file2=fopen("exe3","wb+");

        fscanf(filel, "%lg%lg%lg", &KZAP, &KIOBL, &BPCH);
        fscanf(filel, "%d%d", &N, &NO);
```

Рисунок 8 – Открытие и вывод из входного файла данных

После того, как все необходимые начальные значения введены в программу, начинается работа вышеописанных функций для проведения математических расчетов.

По окончанию их работы, начинается запись всех полученных значений в выходной файл, как показано на рисунке 9.

```
fprintf(file2, "%lg %lg %lg \n", KZAP, KIOBL, BPCH);
fprintf(file2, "%d %d \n", N, NO);
fprintf(file2, "%lg %lg %lg %lg \n", X0, X1, X2, X3);
```

Рисунок 9 – Запись полученных значений в файл

Для удобства при реализации алгоритма был добавлен вывод полученных значений на экран, как показано на рисунке 10.

```
printf("%lg %lg %lg \n", KZAP, KIOBL, BPCH);
printf("%d %d \n", N, NO);
printf("%lg %lg %lg %lg \n", X0, X1, X2, X3);
```

Рисунок 10 – Вывод полученных значений на экран

По окончанию всех основных математических расчетов, было проведено масштабирование, как показано на рисунках 11 – 13.

```
//РАССЧЕТ МАСШТАБОВ ЗАДАЧИ

WKM = PI*((pow(D0,2) + D1*D0 + pow(D1,2))*(X1 - X0)
SM = PI*pow(D,2)/4.0;
DELTA = W1/WKM;
RM = DELTA;
PM = FP*RM;
CM = sqrt(PM/RM);
BPM = ALKM/CM;
```

Рисунок 11 – Расчет масштабов задачи

```
//ПЕЧАТЬ ЗНАЧЕНИЙ МАСШТАБОВ
    printf("Значения масштаба: \n");
    printf("%lg \n", DELTA);
    printf("%1.16e \n", PM);
    printf("%1.16e \n", RM);
    printf("%1.16e \n", CM);
    printf("%lg \n", SM);
    printf("%lg \n", ALKM);
    printf("%1.16e \n", BPM);
    printf("%lg \n", WKM);
```

Рисунок 12 – Вывод значений на экран

```
fprintf(file2, "%1.16e \n", DELTA);
fprintf(file2, "%1.16e \n", PM);
fprintf(file2, "%1.16e \n", RM);
fprintf(file2, "%1.16e \n", CM);
fprintf(file2, "%1.16e \n", SM);
fprintf(file2, "%1.16e \n", ALKM);
fprintf(file2, "%1.16e \n", BPM);
fprintf(file2, "%lg \n", WKM);
```

Рисунок 13 – Запись значений масштаба в файл

Для определения времени работы программы, была использована функция clock (), которая вызывается из библиотеки <ctime>, доступной для

высокого языка программирования C++ [10]. Данная функция помогает определить время работы программы путем задания начала и конца отсчета. Как показано на рисунках 14 и 15, начало и конец отсчета соответственно совпадают с границами тела программы.

```
int main ()
{
    unsigned int start_time =  clock();
```

Рисунок 14 – Начало использования функции clock ()

```
    //-----------------------------------------------------
    fclose(file2);

    unsigned int end_time = clock(); // конечное время
    unsigned int search_time = end_time - start_time;
    printf("Время выполнения %x секунд \n", search_time);
    return 0;
}
```

Рисунок 15 – Конец использования функции clock ()

2.3 Разработка алгоритма с методом распараллеливания OpenMP

Описанный выше последовательный алгоритм является сложным для математических подсчетов, а также достаточно тяжелым для выполнения на центральном процессоре.

Для его усовершенствования была выбрана технология, позволяющая распределить нагрузку на CPU по нескольким потокам – OpenMP (Open Multi-Processing). Данная технология является открытым стандартом для распараллеливания уже созданных программ с последовательными алгоритмами на языках высокого уровня C, C++ и Fortran. В ней описаны совокупности директив компилятора с библиотечными процедурами и

переменными окружения, предназначенными для программирования приложений с несколькими потоками на многопроцессорных системах с общей памятью. Основным преимуществом данной технологии является простота в реализации многопоточных задач.

Первая версия OpenMP была создана в 1997 году для хорошо развитого и распространенного языка в то время Fortran [3]. Доработать данную технологию для C и C++ получилось уже в 1998 году, а в дальнейшем уже начали появляться все более новые версии (в 2008 году появилась версия 3.0, в 2014 году – 4.0).

В данной выпускной квалификационной работе будет предоставлена реализация алгоритма по последней версии OpenMP 4.0.

Основной задумкой данной технологии является создание основным потоком (master) подчиненных потоков (slave) и распределение между ними нагрузки, что позволяет произвести разгрузку очереди на общем потоке и уменьшить время выполнения программы.

Ключевым моментом изначально было ограничение в создании дочерних потоков – их количество должно быть меньше или равно числу процессоров, однако, благодаря доработкам в OpenMP, данное ограничение стало не обязательным. На данный момент, количество создаваемых потоков может быть ограничено как программистом, по средствам вызова процедур из библиотеки, так и переменными окружения. Для ограничения создаваемых потоков путем переменных окружения, можно воспользоваться OMP_NUM_THREADS.

Программист, работающий над созданием распараллеленного алгоритма, может быстро реализовать данную технологию в своем приложении. Для этого необходимо решить, какие части кода будут выполняться путем нескольких потоков, выполнит ли распараллеливание той, или иной части кода, поставленные перед ним задачи.

Для порождения потоков обычно выбирают те фрагменты программного кода, в которых заключены трудоемкие математические расчеты, чтобы

наиболее эффективно использовать ресурсы центрального процессора. Таким образом, для распараллеливания в вышеописанном алгоритме были выбраны такие части программы как расчет параметров во внутренних ячейках, подпрограмма вывода информации, задание значений переменных и др.

2.4 Реализация распараллеливания последовательного алгоритма

Для реализации распараллеливания алгоритма был создан новый проект, имеющий прежние входные значения и структуру кода.

Чтобы использовать технологию OpenMP, необходимо добавить дополнительный параметр в свойствах проекта. Для этого можно перейти в свойства проекта, затем выбрать компилятор соответствующего языка, на котором реализован алгоритм, и в строке «дополнительные параметры» указать –fopenmp, как показано на рисунке 16.

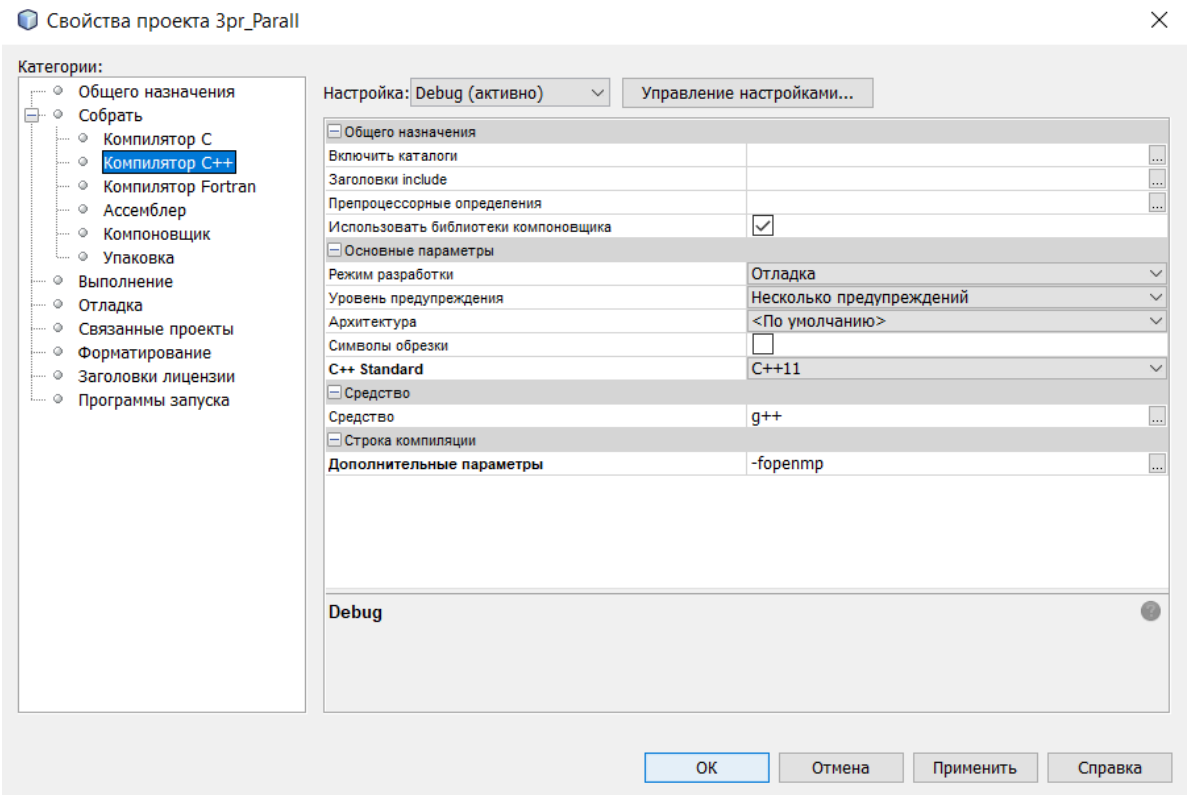

Рисунок 16 – Настройки проекта

После применения новых свойств проекта, необходимо дописать новую подключаемую библиотеку «omp.h», как показано на рисунке 17.

```
#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <ctime>
#include <omp.h>
```

Рисунок 17 – Подключение библиотеки OpenMP

Для дальнейшей работы над распараллеливанием, рассмотрим участки кода выбранных ранее частей алгоритма для создания дочерних потоков.

В подпрограмме счета шага течения смеси для распараллеливания был выбран расчет параметров во внутренних ячейках. Для реализации разделения расчетов на несколько потоков можно рассмотреть несколько вариантов применения технологии. В выбранном фрагменте кода наилучшим образом подходит распределение цикла for на несколько потоков, так как это позволит производить расчеты над каждой итерацией на отдельной нити. Как показано на рисунке 18, можно выбирать самостоятельно количество порождаемых потоков (#pragma omp parallel for num_threads(4)), но, в данном случае, для лучшей демонстрации возможности повышения производительности путем использования технологии OpenMP, было вобрано не ограничивать число потоком самостоятельно, без участия системы.

```
    // РАСЧЕТ ПАРАМЕТРОВ ВО ВНУТРЕННИХ ЯЧЕЙКАХ
#pragma omp parallel for
    //#pragma omp parallel for num_threads(4)
        for(I = 2; I<=N; I++)
        {
                RKL = RP/RM*(1.0 - FI[I - 1]);
        RKP = RP/RM*(1.0 - FI[I]);
                RPRK(RKL, W[I - 1], RKP, W[I], RRK2, WW2);
                if ((RKL > RKPR) || (RKP > RKPR)) WW2 = 0.0;
                if (S[I]*FI[I] <= S[I - 1]*FI[I - 1]) goto g204;
        SK = S[I - 1]*FI[I - 1]/S[I]/FI[I];
```

Рисунок 18 – Пример распараллеливания участков программы

Для применения технологии распараллеливания можно рассматривать как целые функции, как это показано на примере расчета параметров, так и несколько ее частей. Для подпрограммы вывода информации, наиболее эффективным оказалось применение технологии распараллеливания для каждого цикла, как показано на рисунках 19 – 21.

```
#pragma omp parallel for
    for(I = 1; I <= N; I++)
    {...7 строк }
```

Рисунок 19 – Распараллеливание цикла внутри функции

```
#pragma omp parallel for
for(I = 1; I <= KK; I++)
{...4 строк }
```

Рисунок 20 – Распараллеливания второго цикла функции

```
#pragma omp parallel for
        for(I = 1; I <= KK; I++)
        {...6 строк }
```

Рисунок 21 – Распараллеливание N-ого цикла функции

Существуют различные способы реализации распараллеливания последовательных алгоритмов по технологии OpenMP, для выбора наилучшей стратегии и повышения эффективности работы программы, были проведены сравнения между возможными реализациями для данного последовательного алгоритма с выбранной математической моделью.

3 Сравнительный анализ алгоритмов

3.1 Тестирование реализованных алгоритмов

Реализация и тестирование алгоритмов, полученных в ходе выполнения данной выпускной квалификационном работы, было проведено на персональном компьютере с процессором Intel (R) Core (TM) i5-8300H, имеющим частоту 2304 МГц, 4 ядра и 8 логических процессоров. Оперативная память данного ПК составляет 8 Гб. На рисунке 22 показаны сведения об операционной системе, на которой были проведены все описанные далее тесты. Данная информация предоставляется в связи с тем, что тестирование разработанных программ может давать различные результаты по параметрам времени и нагрузки системы на разных операционных системах и физическом оборудовании.

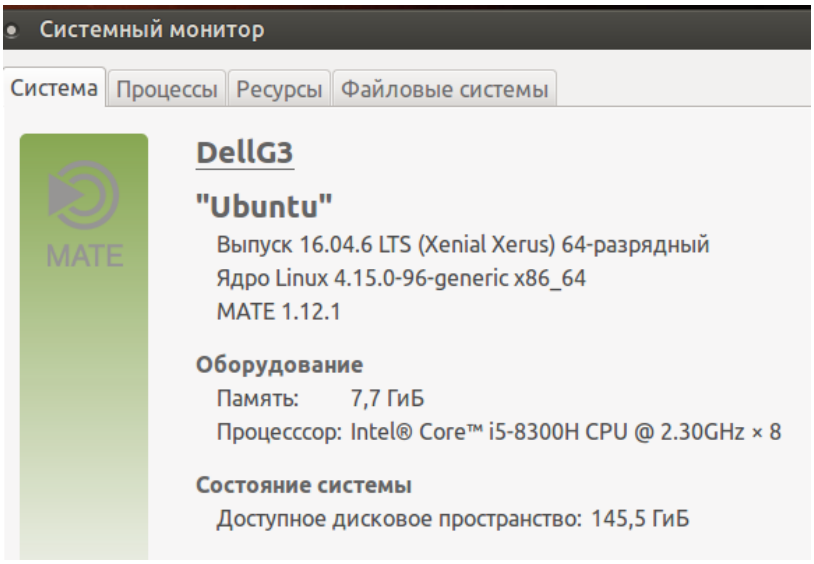

Рисунок 22 – Операционная система ПК

Полученные в ходе описанного исследования программные реализации, по средствам тестирования данных алгоритмов, дали одинаковый результат в расчетах, как показано на рисунках 23 - 26. При выполнении каждой

программы были использованы одни входные данные, записанные в общем входном файле «DAN», а также были получены одинаковые файлы «exe3» с конечными значениями, имеющие идентичные размер и данные. Внутренние структуры проектов предоставлены на рисунках 27 и 28.

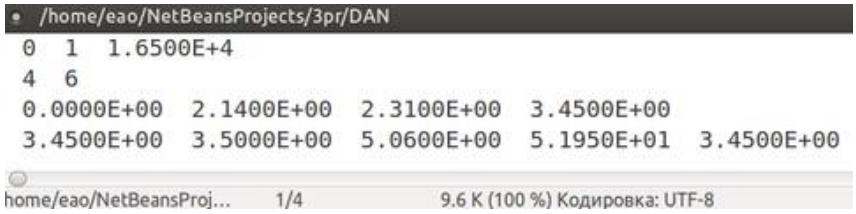

Рисунок 23 – Входные данные для последовательного алгоритма

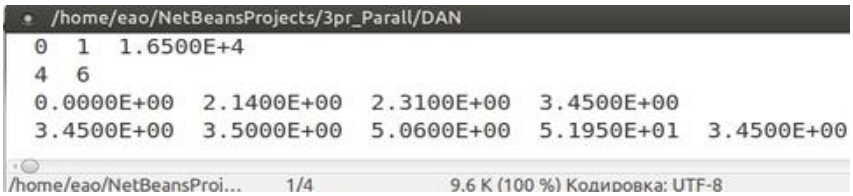

Рисунок 24 – Входные данные для распараллеленного алгоритма

Рисунок 25 – Полученные данные при последовательном алгоритме

Рисунок 26 – Полученные данные при распараллеленном алгоритме

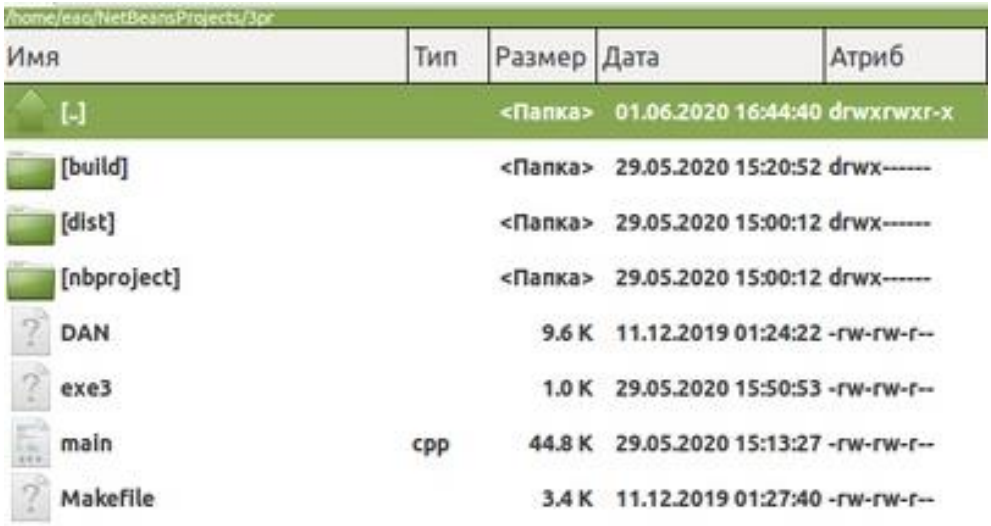

Рисунок 27 – Проект для тестирования последовательного алгоритма

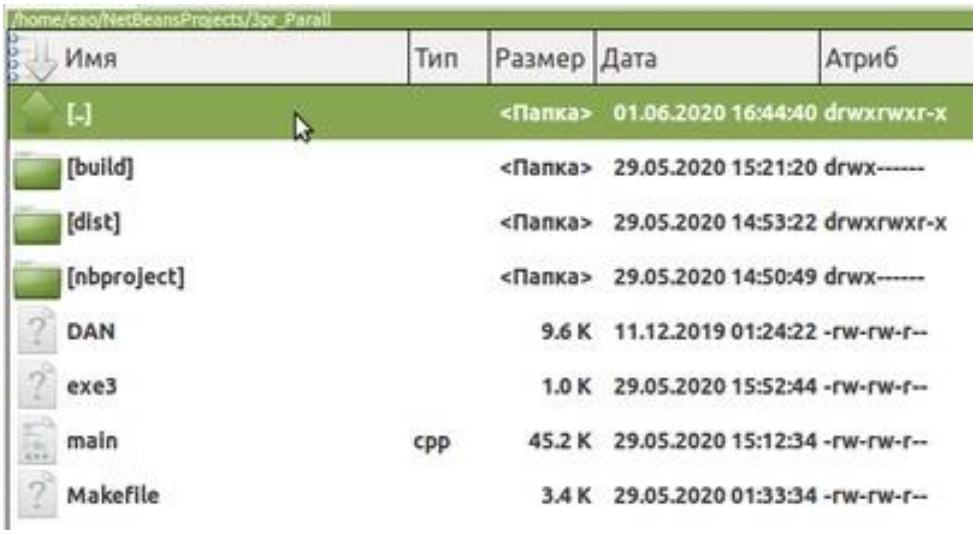

Рисунок 28 – Проект для тестирования распараллеленного алгоритма

3.2 Анализ скорости выполнения программ

После проведенных тестов, доказавших верное выполнение поставленных задач перед каждой программой, стало необходимо провести сравнительный анализ разработанных и реализованных алгоритмов.

В первую очередь, одним из показателей эффективности алгоритмов, учитывая их точность по созданной математической модели, стало время, которое затрачивается на выполнение каждой программы, производящей необходимые расчеты по эстафетной схеме. Как было описано во втором разделе, для фиксации затраченного времени и чистоты исследования были использованы одинаковые функции, не затрагивающие особенности каждого алгоритма.

Так как скорость работы программ зависит напрямую от загруженности операционной системы различными внешними процессами, запущенными в момент отладки и выполнения проекта, было проведено несколько тестов с фиксацией времени, затраченного на каждый алгоритм. Для проведения анализа, все данные о времени выполнения программ, полученные по их завершению, предоставлены в таблице 1.

Таблица 1 – Данные по времени выполнения программ

Тестируемый алгоритм	Время выполнения при первом запуске программы (мс)	Время выполнения при втором запуске программы (мс)	Время выполнения при третьем запуске программы (мс)
Последовательный	494	415	449
Распараллеленный	247	201	220

По данной таблице, можно построить диаграмму, демонстрирующую проведенный опыт, как показано на рисунке 29.

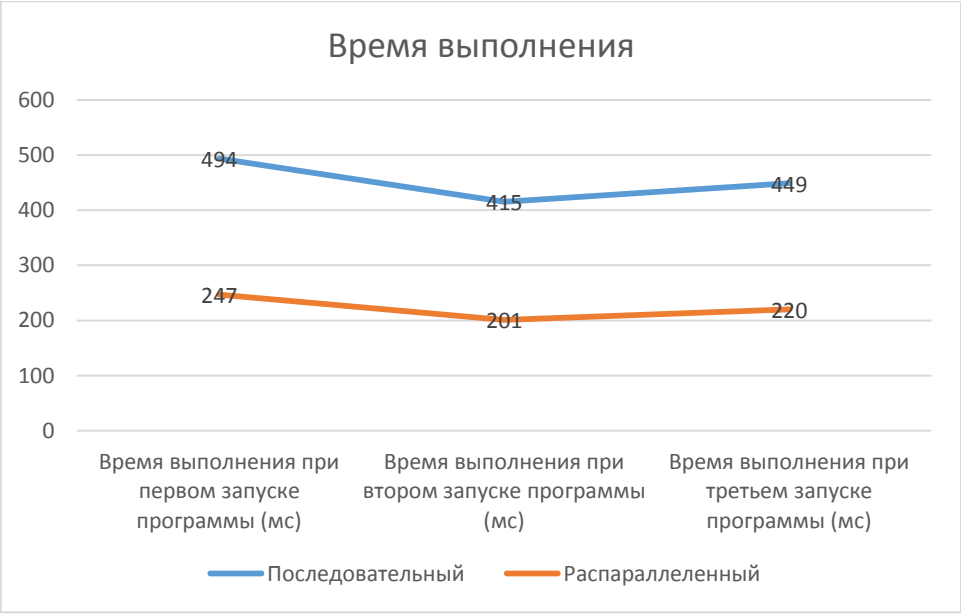

Рисунок 29 – Диаграмма значений затраченного времени

Как видно на данной диаграмме и по таблице 1, время выполнения каждого алгоритма одинаково зависит от запущенных внешних процессов. Однако, если рассматривать кривые на рисунке 29, то заметна линейная зависимость между двумя выполняемыми алгоритмами. Чтобы установить зависимость по скорости выполнения двух проектов, воспользуемся формулой 8.

$$\frac{v_1}{v_2} = \frac{t_1}{t_2} = k_n \qquad (8)$$

где v_1 – скорость выполнения последовательного алгоритма;

v_2 – скорость выполнения распараллеленного алгоритма;

t_1 – время выполнения последовательного алгоритма;

t_2 – время выполнения распараллеленного алгоритма;

k_n – коэффициент ускорения при распараллеливании программы в n-ом запуске.

Таким образом, получим расчеты, где $k_1 = \frac{494}{247} = 2$, $k_2 = \frac{415}{201} = 2{,}06$, $k_3 = \frac{449}{220} = 2{,}04$. Для того, чтобы получить среднее значение коэффициента ускорения при распараллеливании программы на n-ом запуске, воспользуемся определением среднего арифметического значения, как показано по формуле 9.

$$k = \frac{k_1 + k_2 + k_3}{3} \qquad (9)$$

При расчете по данной формуле 9, получим среднее значение коэффициента ускорения программы, равное 2,03.

Если анализировать эффективность проделанной работы над реализованными алгоритмами по скорости их выполнения, то полученное значение k = 2,03 будет являться коэффициентов эффективности.

3.3 Анализ нагрузки системы при выполнении программ

Для полного рассмотрения эффективности при выполнении данной выпускной квалификационной работы, также необходимо провести тесты, рассматривающие нагрузку на систему, которая возникает при выполнении двух описанных и реализованных алгоритмов.

Первый тест был проведен с использованием системного монитора, который является встроенным инструментом в операционной системе Ubuntu MATE. С помощью него можно рассмотреть какую нагрузку от 0 до 100% задают запущенные процессы на каждый из 8 процессоров, а также как в этот момент используется память и сеть. Чтобы точно знать, что те или иные изменения произошли именно из-за запуска программы, все изменения отслеживаются в течении одной минуты (60 секунд), отсчет времени расположен на горизонтальной оси.

На рисунке 30 показано состояние системы по системному монитору до начала эксперимента, когда не были запущены проекты.

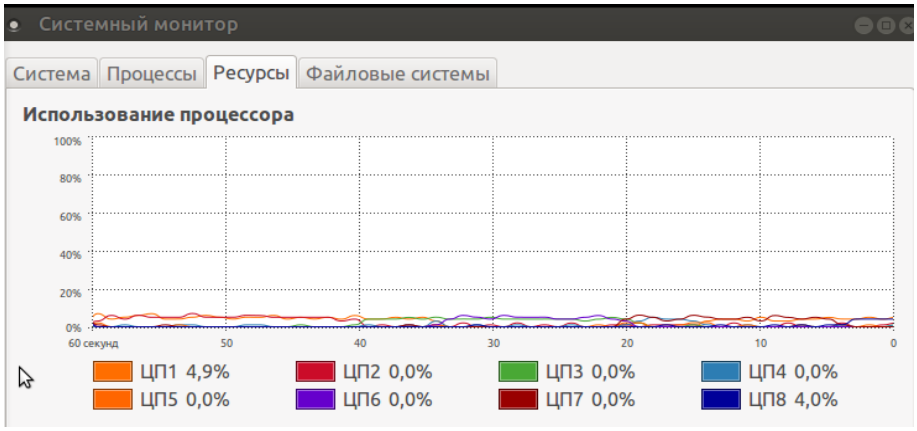

Рисунок 30 – Системный монитор без запуска программ

Затем, чтобы протестировать подаваемую на систему нагрузку выполнением последовательного алгоритма, необходимо завершить работу всех запущенных приложений на ПК и запустить только один проект с реализацией последовательного алгоритма. На рисунке 31 предоставлен отчет системного монитора, по которому мы можем зафиксировать некоторые изменения.

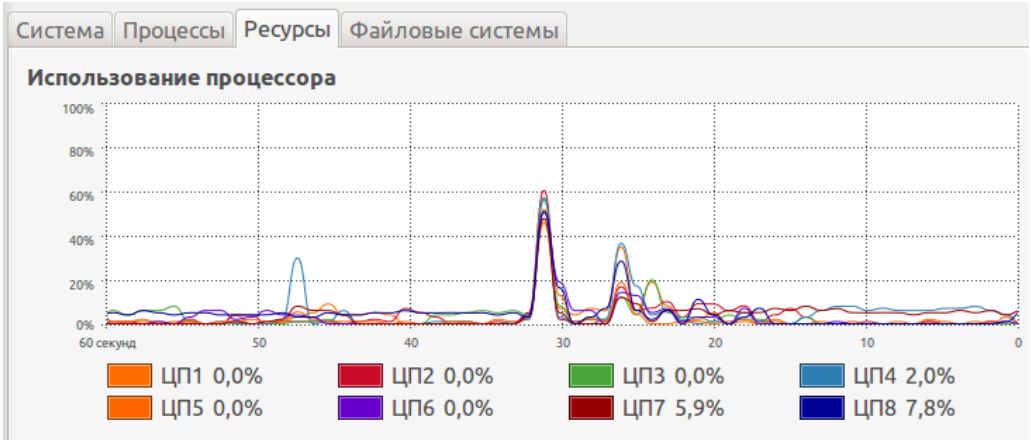

Рисунок 31 – Системный монитор при выполнении последовательного алгоритма

Как видно, по нагрузке на процессоры происходит резкое и продолжительное изменение. При первом скачке изменений практически все

восемь ЦП достигают 60%. При следующей волне нагрузки, во время выполнения последовательной программы, ЦП уже не достигают 40%, большинство из них загружаются лишь на 20% и менее.

Из данного теста видно, что разницей между нагрузкой системы без выполнения программ и с выполнением последовательного алгоритма можно считать 60% для всех восьми ЦП.

Далее необходимо протестировать данным методом алгоритм, в котором была использована технология OpenMP. На рисунке 32 зафиксированы показания системного монитора по данному тесту.

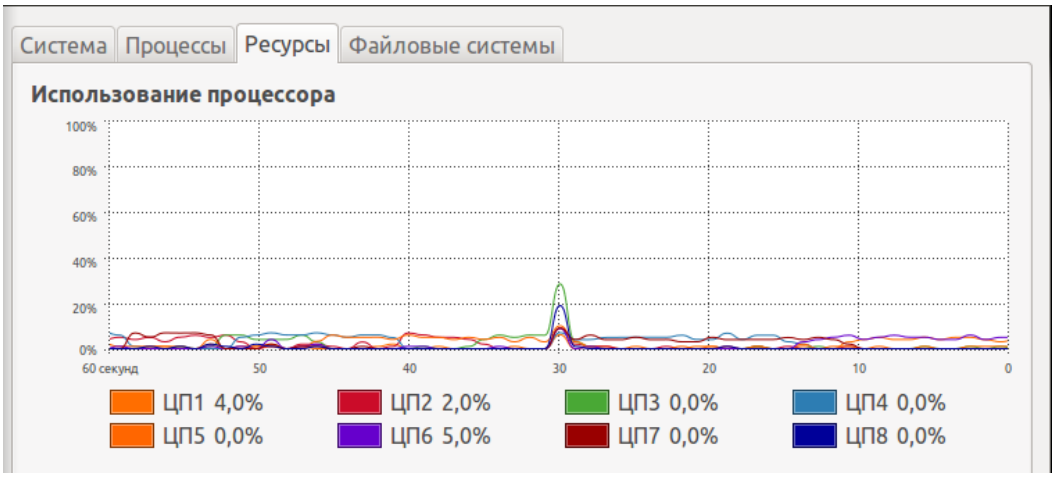

Рисунок 32 - Системный монитор при выполнении распараллеленного алгоритма

Если проанализировать полученные данные по последнему тесту, то видно, что явная нагрузка на систему осуществляется только в один момент времени. В отличии от последовательного алгоритма, мы не видим несколько «волн» нагрузок, вызывающие сильную нагрузку на ЦП. Во время единственного момента, в котором потребовались ресурсы ЦП более чем на 10%, видно, что из восьми ЦП были нагружены только два: ЦП3 был нагружен приблизительно на 30%, ЦП6 на 20%.

Если рассматривать данные тесты по максимальным значениям нагрузки хотя бы одного ЦП, тогда для последовательного алгоритма будет приниматься значение 60%, а для распараллеленного 30%.

Таким образом можно сказать, что при данном методе тестирования нагрузки на систему алгоритм с использованием технологии OpenMP эффективнее последовательного на 30%, или в 2 раза.

Для проведения последнего сравнительного теста был выбран дополнительный инструмент для мониторинга производительности ПК, созданный только для операционных систем AIX и Linux – Nmon. Функции этого инструмента предоставляют информацию о параметрах CPU, объема памяти, информации о сетях и дисках, состояние файловой системы, NFS и многое другое. Nmon является приятным интерактивным инструментом, позволяющим быстро узнавать информацию о системе пользователям и администраторам, а также его легко и быстро можно установить из репозитория по умолчанию в Ubuntu и Debian.

В первую очередь была протестирована система без запущенных программ. На данный момент, интересно было получить информацию о состоянии CPU, поэтому программа Nmon была запущена в одном из своих режимов «c». На рисунке 33 показано состояние системы и используемые ресурсы на CPU в момент, когда не были запущены программы.

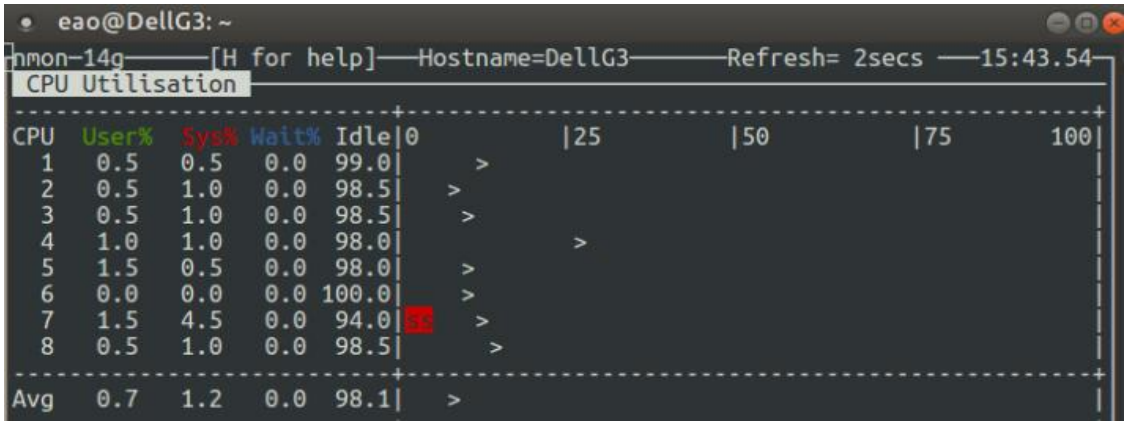

Рисунок 33 – Состояние CPU без запущенных программ

Как показано на рисунке 33, когда не запущены программы, система нагружена только на одном из процессоров, который занимается системными задачами, а пользователь в этот момент не загружает CPU персонального компьютера.

Затем была запущена программа с реализацией последовательного алгоритма. На рисунке 34 показано состояние системы в момент выполнения проекта.

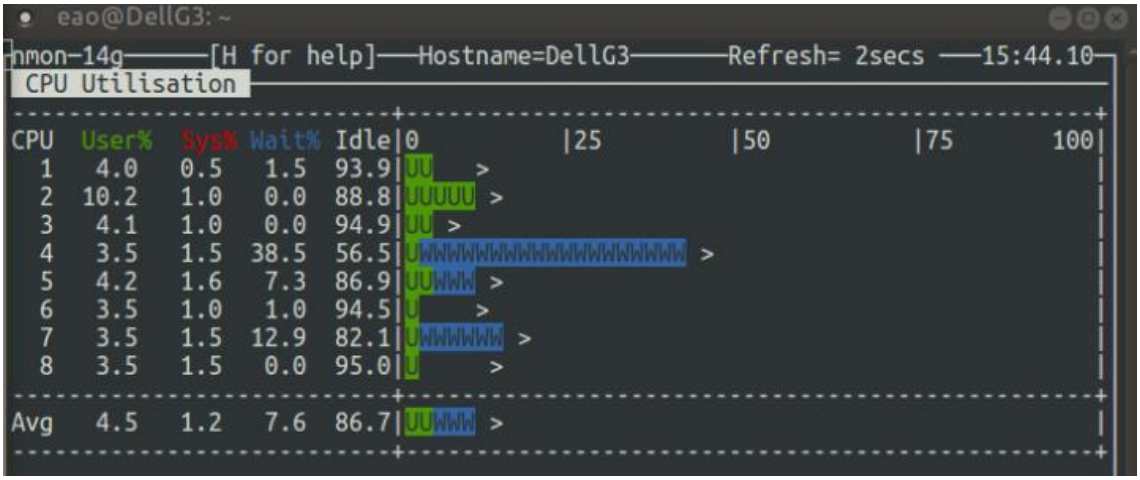

Рисунок 34 – Состояние CPU с запущенным последовательным алгоритмом

При сравнении отчетов программы Nmon по рисункам 33 и 34 видна разница в задействованных ресурсах и нагружаемых процессах. Так, при выполнении последовательного алгоритма, были задействованы все процессоры, но нагрузка не была распределена равномерно. Пользователь занимает ресурсы каждого процессора, наибольшая нагрузка производится им на второй, а именно на 10,2%. Если рассматривать общую нагрузку на CPU, то четвертый процессор нагружен больше остальных, а именно на 43,5%. Таким образом, из-за неравномерного распределения нагрузки на CPU, его общая загруженность составляет примерно 65%.

Последним тестовым запуском проверяется состояние CPU в момент выполнения программы распараллеленным алгоритмом. На рисунке 35 предоставлен отчет программы Nmon.

Рисунок 35 - Состояние CPU с запущенным распараллеленным алгоритмом

Как видно из отчета программы Nmon, нагрузка, создаваемая пользователем на систему, распределяется равномерно, при этом нет ни одного процессора, загруженного выполнением проекта с распараллеленным алгоритмом больше остальных более чем на 1%. Заметим, что на четвертом процессоре, так же, как и при последовательном алгоритме, происходит большая часть процессов, а его загрузка составляет максимальное значение, а именно 18,5%. Общая нагрузка программой на CPU составляет приблизительно 43%.

Если сравнивать общую нагрузку программ на CPU, тогда отметим показатели при последовательном алгоритме – 65%, и при распараллеленном – 43%. Таким образом, при первой реализации с последовательным алгоритмом, мы получаем больше нагрузки на 22%, чем с распараллеленным алгоритмом, или в 1,5 раза.

3.4 Анализ ускорения выполнения программ по Закону Амдала

Согласно Закону Джина Амдала, который был сформулирован им в 1967 году, ускорение выполнения любой программы на вычислительной машине, имеющей несколько процессоров, ограничено временем, которое необходимо затратить на совершение последовательных инструкций в программном коде. Данный закон описывает зависимость ограничения роста производительности при распараллеливании вычислений от параметров системы и структуры алгоритма.

Для проведения анализа возможных результатов при распараллеливании алгоритма данный закон требует знаний о количестве процессоров вычислительной машины и доли распараллеленного кода. Знание этих данных поможет вычислить предполагаемое максимально возможное ускорение работы над вычислительными процессами в программе. Вычисление коэффициента ускорения происходит по формуле 10.

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}} \quad (10)$$

где S_p – коэффициент ускорения работы программы;

p – количество процессоров;

α - доля последовательных вычислений.

Анализируя данную формулу, можно сделать вывод, что при меньшей доли последовательного кода и большем количестве процессоров значение коэффициента ускорение будет увеличиваться.

Ранее была описана операционная система и персональный компьютер, на котором были проведены все тестирования реализованных алгоритмов. По вышеописанным характеристикам, значение параметра p будем принимать равное 8. Проанализировав распараллеленный алгоритм, получим долю

последовательного кода равную приблизительно 40%. Таким образом, по формуле 11 получим максимальный коэффициент ускорения.

$$S_p = \frac{1}{0{,}4+\frac{1-0{,}4}{8}} = 2{,}1 \ (11)$$

Полученный результат, согласно Закону Амдала, по максимальному значению ускорения работы программы на 8 процессорах и при доле последовательного кода равной 40%, полностью удовлетворяет значениям, полученным при тестировании последовательного и распараллеленного алгоритмов.

По полученным данным, при сравнении скорости выполнения расчетов, распараллеленный алгоритм работает быстрее последовательного приблизительно в 2,03 раз, что меньше максимального значения ускорения по Закону Амдала. При сравнении нагрузки на систему во время выполнении программ, были получены данные, что распараллеленный алгоритм работает в 1,5 раза эффективнее при исследовании с помощью инструмента Nmon и в 2 раза при использовании системного монитора, как средства фиксации производительности. Каждое из полученных значений эффективности удовлетворяет выполнению Закона Амдала и являются меньшими значениями, чем максимальное, найденное по формуле 10.

Чтобы увеличить скорость работы распараллеленного алгоритма, можно уменьшить долю последовательного кода в программе, а также рассмотреть варианты использования данного алгоритма на системе с большим количеством процессоров.

Для рассмотрения теоретически возможных максимальных коэффициентов ускорения по Закону Амдала, возьмем доли последовательного кода равные 50%, 40% (протестированные выше), 30%, 20%, 10%, 0%, а также количество возможных процессоров 4, 8, 10, 100, 1000. Полученные по формуле 10 значения отображены в таблице 2.

Таблица 2 – Коэффициент по Закону Амдала

$p\backslash\alpha$	50%	40%	30%	20%	10%	0%
4	1,6	1,8	2,1	2,5	3,07	4
8	1,78	2,1	2,58	3,33	4,7	8
10	1,18	2,17	2,7	3,57	5,26	10
100	1,98	2,46	3,26	4,81	9,17	100
1000	1,99	2,5	3,33	4,98	9,91	1000

По полученным данным видно, что получение линейного прироста производительности при повышении количества процессоров системы возможно только при использовании алгоритма, не содержащего последовательных вычислений, где доля α равна нулю. На рисунке 36 предоставлена диаграмма полученных данных.

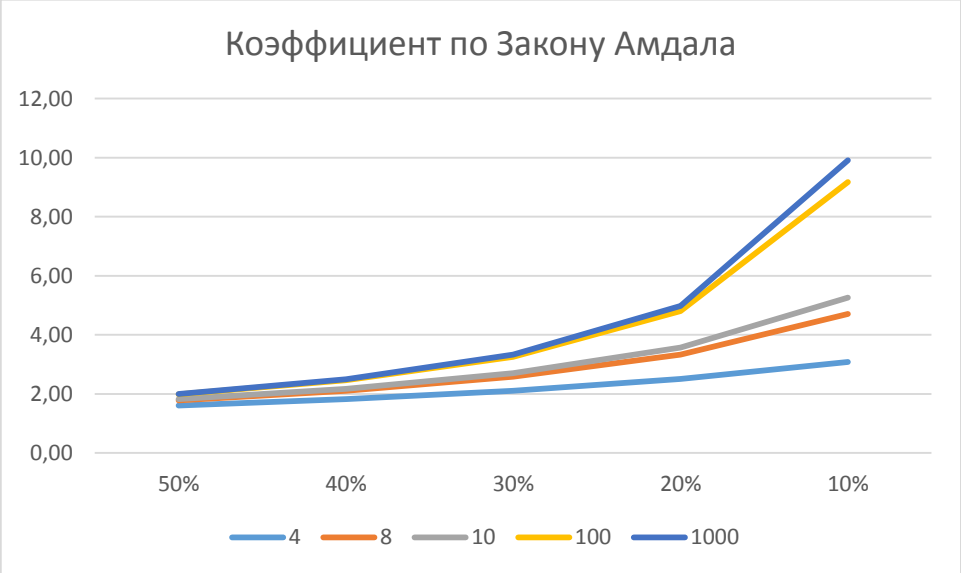

Рисунок 36 – Диаграмма полученных данных

Как показано на рисунке 36, увеличение числа процессоров будет более эффективным при большей доле распараллеленного кода программы, меньшей доле последовательного.

Таким образом, данный закон доказывает, что не для каждой задачи возможен подход увеличения числа процессоров для повышения производительности алгоритма, более эффективным может стать метод распараллеливания и уменьшения доли последовательного кода. По данным из таблицы 2 можно сделать вывод, что при работе алгоритма с 10% последовательного кода в системе с 4 процессорами можно добиться большей эффективности, чем при работе в системе с 1000 процессоров, если доля последовательного кода будет составлять не менее 40%.

Заключение

В ходе выполнения выпускной квалификационной работы на тему «Моделирование систем эстафетной схемы с использованием распараллеленного алгоритма» были рассмотрены теоретические основы существующей математической модели эстафетной схемы, описывающие процессы и взаимосвязи физических явлений во время движения метаемого элемента в стволе и его разделение на основную и дополнительные части.

Во время проведенных исследований, были изучены материалы, связанные с математическими расчетами, позволяющими получить значения выходных данных по исходным параметрам при использовании эстафетной схемы.

Цель исследования, связанная с ускорением вычислительной работы алгоритма над входными и выходными значениями математической модели, была достигнута благодаря выполнению всех поставленных в данной работе задач.

Таким образом, были рассмотрены существующие решения проблемы вычислительного характера, такие как время, затрачиваемое на выполнение алгоритма, и необходимые для него используемые ресурсы.

По существующей математической модели, были разработаны и реализованы алгоритмы с последовательным и параллельным выполнением расчетов. Спроектированные программные реализации были написаны на языке программирования высокого уровня C++ в кроссплатформенной свободной интегрированной среде разработки приложений NetBeans IDE. Для работы с полученными проектами и анализа их эффективности была выбрана операционная система Ubuntu MATE, а также использованы вспомогательные инструменты ОС - Nmon.

Для разработки программы с распараллеливанием последовательного алгоритма была выбрана технология OpenMP, которая не задействует графический процессор, как это было в других научных исследованиях с целью

ускорения работы над расчетами. Этот выбор технологии предоставляет возможность повысить скорость вычислительных процессов на ЭВМ, не имеющих мощных ресурсов GPU.

Как показал анализ эффективности двух разработанных программ, даже без возможности подключения технологий с использованием графических процессоров, можно увеличить скорость выполнения расчетов приблизительно в 2,1 раза, согласно Закону Амдала, и уменьшить нагрузку на CPU более чем в 1,5 раза, используя достаточно простую технологию распараллеливания OpenMP.

По полученным расчетам, было доказано, что повышения производительности алгоритма будет зависеть от доли последовательного программного кода и количества используемых процессоров.

Таким образом, в данной выпускной квалификационной работе были отображены все основные моменты математической модели метания элемента по эстафетной схеме, а также разработаны и программно реализованы алгоритмы с последовательным и параллельным выполнением вычислений.

Благодаря проведенным тестам и расчетам по Закону Амдала, была доказана эффективность работы распараллеленного алгоритма для моделируемой системы эстафетной схемы.

Список используемых источников

1. Scott Meyers Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14.
2. Scott Meyers. Effective Modern C++. O'Reilly, 2015.MPI: A Message Passing Interface Standard Version 2.2. [Электронный ресурс]: Режим доступа: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf.
3. Wilkinson B., Allen M. Parallel programming techniques and applications using networked workstations and parallel computers. - Pearson Education, 2005. - p. 468.
4. Антонов А. С. Параллельное программирование с использованием технологии OpenMP: учебное пособие / А. С. Антонов. - М.: Изд-во МГУ, 2012. - 77 с.
5. Березкин Б. И., Березкин С. Б. Начальный курс С и С++. Москва «Диалог-Мифи», 2005 г.
6. Вилюнов В. Н. Газовая динамика двухфазного течения в соплах. Издательство Томского университета. 1986 г. URL : http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000095140
7. Воеводин В. В. Вычислительная математика и структура алгоритмов. - Москва: Издательство МГУ, 2006. - 112 с.
8. Гергель В.П. Теория и практика параллельных вычислений - Национальный Открытый Университет "ИНТУИТ" - 2016 - ISBN: 978-5-94774-645-7 - Текст электронный // ЭБС Лань - URL: https://e.lanbook.com/book/100527
9. Гринько Г. В., Сафронов А.И. Внутренняя баллистика ствольной системы эстафетной схемы // Материалы III научно-практической всероссийской конференции (школы-семинара) молодых ученых, 2017. pp. 132-134.
10. Дягтерев М. Е. Высокоскоростной патрон с разделенным пороховым зарядом «Искра-М». Российский оружейный журнал

«Калашников» Оружие, боеприпасы, снаряжение, №3, Санкт- Петербург: ООО «Азимут» 2011, с.11

11. Дягтерев М. Е. Высокоскоростные пулевые патроны «Искра-М» для гладкого ствола». Российский оружейный журнал «Калашников» Оружие, боеприпасы, снаряжение, №4, Санкт- Петербург: ООО «Азимут» 2014, с.62-67.

12. Малявко А. А. Параллельное программирование на основе технологий OpenMP, MPI, CUDA. - Новосибирск: Изд-во НГТУ, 2015. - 116 с. ISBN 978-5-7782-2614-2.

13. Малявко А. А. Параллельное программирование на основе технологий OpenMP, MPI, CUDA. 2-е изд., испр. и доп. Учебное пособие для академического бакалавриата М. : Издательство Юрайт,2019-129-Высшее образование-978-5-534-11827-8: - Текст электронный // ЭБС Юрайт - https://biblio-online.ru/book/parallelnoe-programmirovanie-na-osnove-tehnologiy-openmp-mpi-cuda-446247

14. Миллер Р., Боксер Л. Последовательные и параллельные алгоритмы. - М.: Бином. Лаборатория знаний, 2006. - с. 406

15. Немнюгин С., Стесик О. Параллельное программирование для многопроцессорных вычислительных систем - СПб.: БХВ-Петербург, 2002.

16. Норейка Р. М. Стрелковое тестирование нового серийного высокоскоростного патрона «Искра-М» Новосибирского механического завода. Российский оружейный журнал «Калашников». Оружие, боеприпасы, снаряжение, №10, Санкт- Петербург: ООО «Азимут» 2014. С.54- 57.

17. Официальный сайт OpenMP - http://openmp.org/wp/

18. Петров В. Ю. Информатика. Алгоритмизация и программирование. [Текст]: учебное пособие / В.Ю. Петров. - Часть 1 - Санкт-Петербург: Университет ИТМО, 2015. - 91 с.

19. Сафронов А. И. Внутренняя баллистика ствольной системы с присоединенной камерой подгона [Текст] / А. И. Сафронов, А. Ю. Крайнов // Вестник ТГПУ. - 2004. Вып.6(43). С. 67-70.

20. Сафронов А. И., Потапенко В.В. Анализ и баллистическое проектирование системы с присоединенной камерой подгона. Вестник Самарского государственного аэрокосмического университета. №3(19), 2009 г, с. 212-216.

21. Семёнов Р. И. Оптимизация параметров системы эстафетного выстрела с использованием генетического алгоритма // Труды международной конференции "Система обеспечения пожарной безопасности. Состояние, тенденции, пути развития", 2017. pp. 215-218.

22. Энтони Уильямс Параллельное программирование на C++ в действии. Практика разработки многопоточных программ - Издательство "ДМК Пресс" - 2012 - ISBN: 978-5-94074-448-1 - Текст электронный // ЭБС Лань - URL: https://e.lanbook.com/book/4813