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)

Компьютерные технологии и математическое моделирование

(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Исследование параллельных алгоритмов для обработки данных в сети»

Студент Н.С.Гуртовой _____

(И.О. Фамилия) (личная подпись)

Руководитель доктор ф.-м. наук, профессор, А.И. Сафронов

(Ученая степень, звание, И.О. Фамилия)

Консультант Е.В.Косс

(Ученая степень, звание, И.О. Фамилия)

Тольятти 2022

Аннотация

Данная выпускная квалификационная работа включает в себя: 2 таблицы, 36 рисунков, 47 страниц, 3 раздела и 22 использованных источников.

Цель работы – сокращение времени, которое затрачивается на обработку исходных данных и математических вычислениях при схеме «Эстафетная».

Объект исследования – решаемые математические задачи, решив которые,можно получить выходные параметры при разных входных данных.

Предмет исследования – алгоритмы и методы распараллеливания, повышающие скорость решения.

Полученные результаты – спроектирован многопоточный алгоритм для эстафетной схемы метания элемента.

Рекомендовано внедрять результаты работы, так как реализованный алгоритм с использованием технологии параллельного выполнения, в дальнейшем будет применяться для работы с математической моделью эстафетной схемы, при этом, сокращая затраты времени и других ресурсов, которые уходят во время процесса рассчитывания решения.

Область применения – Разработанную программу можно применить во время математических подсчётов в компьютерном моделировании.

Значимость работы – Разработанный алгортм позволяет наиболее эффективным способом пользоваться ресурсам центрального процессора, и не менее важно, повышает скорость решения задачи.

Abstract

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

The graduation work consists of an explanatory note on 47 pages, introduction, three parts including 36 illustrations, 2 tables, the list of 22 references including 3 foreign sources.

The object of this bachelor's work is to rise the input parameters processing and to take output data speed with the use of mathematical calculations in the relay throwing element.

The object of study is the mathematical problems to be solved, by solving which, you can get the output parameters for different input data.

The subject of research is algorithms and parallelization methods that increase the speed of the solution.

Results - a multi-threaded algorithm for the relay-race scheme of throwing an element was designed.

It is recommended to implement the results of the work, since the implemented algorithm using the parallel execution technology will be used in the future to work with the mathematical model of the relay scheme, while reducing the time and other resources that are spent during the process of calculating the solution.

Scope - The developed program can be used during mathematical calculations in computer simulation.

Application area - The developed algorithm allows the most efficient way to use the resources of the central processor, and no less important, increases the speed of solving the problem.

Содержание

Введение

В настоящее время, одним из наиболее важных научных разделов обширного предмета физики, как не странно является внутренняя баллистика, ведь она вносит определённый вклад в разработку и реализацию различных систем вооружения. Помимо прочего, данный раздел физики отвечает на важный ряд вопросов, возникающих в то время, когда разрабатывается новая система вооружения, и даже при усовершенствовании старого. «Решение очередных задач, при выполнении модернизаций и создании новых продуктов, требует ресурсоёмких математических вычислений, занимающих огромное количество времени. Чтобы повысить точность во время проведения расчетов и ускорения процесса, выполняются всевозможные исследования по созданию программ, для математических расчетов.» [3]

«В разделе внутренней баллистики существуют разные схемы, которые требуют глубокого погружения в детали. В качестве ознакомления, в данной бакалаврской работе была приведена одна из них – эстафетная схема метания. Она вызывает интерес своим подходом в решении вопроса увеличения скорости метаемого элемента, при котором заряд патрона делится на две или три части. Если заряд делится на две составляющие, то такой патрон считается бинаром, а если на три – тринаром.» [3]

«В стандартном варианте рассмотрения эстафетной схемы с двумя составляющими, используется пластмассовая гильза с воспламенителем, которую заполняют основной частью пороха, а сверху устанавливают картонную диафрагму, разделяющую патрон на бинар, где во второй части присутствует дополнительная часть порохового заряда из того же или другого вещества.»[13]

«При подобном решении обязательно нужно проводить большое количество различных вычислений, связанных с варьированием величины частей пороха, положением диафрагмы, материалами метаемых элементов и многих других параметров.

Чтобы улучшить работу расчета данных, проводятся различные исследования в разработке приложений, которые решают проблемы с точностью и израсходованным временем. В настоящее время, на различных языках программирования, уже разработаны приложения, которые используют разные виды технологий.»[8]

«Существенную долю уже созданных алгоритмов занимают последовательные, но благодаря тенденциям применения всех возможных ресурсов персональных компьютеров, трудно не заметить, что актуальным также становится и алгоритм с многопоточным программированием, с использованием распараллеливания.» [2]-[4]

«Имеющаяся в настоящее время модель эстафетной схемы в виде последовательного алгоритма хорошо решает поставленные перед ней задачи, потратив при этом относительно немного времени для расчетов.»[6]

«Чтобы уменьшить затраченное время, по мимо прочего, были произведены исследования с участием технологии OpenCL на языке программирования C, предназначенная для создания многопоточного алгоритма, учитывая ресурс центрального и графического процессоров. Немалую выгоду в данной технологии, как и в технологии «Cuda» для C++, несёт в себе то, что все математические исчисления, которые нагружают центральный процессор, как элементарные задачи возлагаются на графический, благодаря чему он может без нагрузок исполнять весь оставшийся алгоритм. Такое распределение позволяет получить как минимум две параллели с огромными ресурсами для расчётов.»[1]

В представленной бакалаврской работе будет рассмотрен последовательный алгоритм, написанный на языке C++, а также, распараллеленный алгоритм с использованием многопоточной технологии «OpenMP», что даст нам возможность исправить существующие недочёты вычислений при использовании эстафетной схемы в баллистике, не пренебрегая при этом ресурсами графического процессора.

Объект исследования – математические вычисления, которые нужны для вывода данных при меняющихся входных переменных.

Предмет исследования – многопоточные алгоритмы, ускоряющие процесс расчётов.

Цель исследования – получить повышение в скорости обработки входящих и выходящих параметров при математическом подсчёте.

Задачи исследования:

- изучить теоретическую составляющую по объекту исследования;
- ознакомиться с насущными решениями проблемы и проанализировать их;
- разработать последовательный алгоритм для вычисления;
- реализовать алгоритм для расчетов с применением многопоточных технологий;
- выполнить сравнительный анализ производительности разработанных алгоритмов.

В первой части настоящей работы, мы рассмотрим все теоретические сведения, которые так или иначе влияют на математические вычисления и модель броска элемента “эстафетной схемы”. Во втором пункте будет освещен процесс работы над алгоритмами, а Итогом данной бакалаврской работы будет считаться сравнительный анализ производительности двух алгоритмов, разработанных для эстафетной схемы метания: многопоточного и последовательного, а так же выявление плюсов и минусов этих алгоритмов.

1 Теоретические сведения

1.1 Характеристика объекта исследования

«Наука, которая посвящена перемещению метаемого компонента в канале ствола, освещающая весь спектр проходящих в этот момент процессов, носит название " Внутренняя баллистика ". Все исчисления, начиная с воздействия пороховых газов и установление прочих закономерностей при явлении выстрела, представляет собой не малое количество областей академических знаний, например, химию, газодинамику, механику, термодинамику, термохимию и др.

На данный момент, к механизмам, выполняющим расчет всех процессов внутренней баллистики добавилась высшая математика и информатика, которые позволяют разработать модель и провести вычисления всех выходных параметров, имея на входе начальные условия, не затрагивая при этом физические эксперименты.

Вследствие использования математического и компьютерного моделирования, мы имеем возможность просчитывать без лишних затрат и использования различного необходимого оборудования все нужные для анализа эксперименты, что стало одним из стимулов к созданию особо корректных алгоритмов, подсчитывающих выходные значения по подготовленным входным данным.»[6]

«Чтобы рассмотреть математическую модель выстрела метаемым элементом из ствола мы должны понимать базовые понятия трудоемкого процесса преображения химической энергии пороха во время горения в тепловую энергию газов, а впоследствии и в механическую энергию движения снаряда и иных подвижных частей ствола, что также определяется как механическая работа.»[19]

Так, вычисления для внутренней баллистической модели разделяются на периоды времени, основываясь на самом процессе выстрела [6]:

- Подготовительный период. В данный временной промежуток можно наблюдать начало воспламенения пороха, здесь же выполняются химические вычисления, которые связаны с горением.
- Первый (основной) период. Начало этого периода совмещается с завершением врезания основных поясков, а окончание данного периода наступает с завершением горения пороха.
- Второй период. Период, который следует сразу после окончания основного и длится до самого момента вылета метаемого элемента из ствола.
- «Период последействия газов. Момент, соответствующий времени вылета метаемого элемента из канала ствола до того, как пороховые газы прекратят оказывать на него действие после выстрела.»[16]

Подводя итог из периодов внутрибаллистического процесса, которые описаны выше, можно отметить такие источники информации, как [5]-[9]:

- «Пиростатика – область внутренней баллистики, исследующая процесс горения пороха и иные вытекающие нюансы газообразования, учитывая изохорический процесс (процесс, сопровождающийся постоянным объемом); в данном разделе уточняются специфики горения порохов разных составов;»[16]
- «Пиродинамика – глава внутренней баллистики, исследующая процессы в канале ствола при выстреле, определяющий зависимости среди специфических технических характеристик внутри ствола, исходными данными метаемого элемента и физико-химическими явлениями в механике;»[17]
- «Баллистическое проектирование – комплекс периода моделирования и определения облика орудия; раздел, решающий вопросы физических вычислений разработки, реализации и сборки компонентов орудия.»[17]

«На изображении 1 представлена зависимость давления и скорости от времени, помимо этого, поделены периоды, начиная от предварительного, сопряженного с воспламенением пороха, заканчивая вылетом из ствола метаемого элемента, который сопровождается воздействием на него пороховых газов (период последействия). На данной иллюстрации представлен общий случай выстрела во внутренней баллистике при классической схеме метания элемента.»[7]

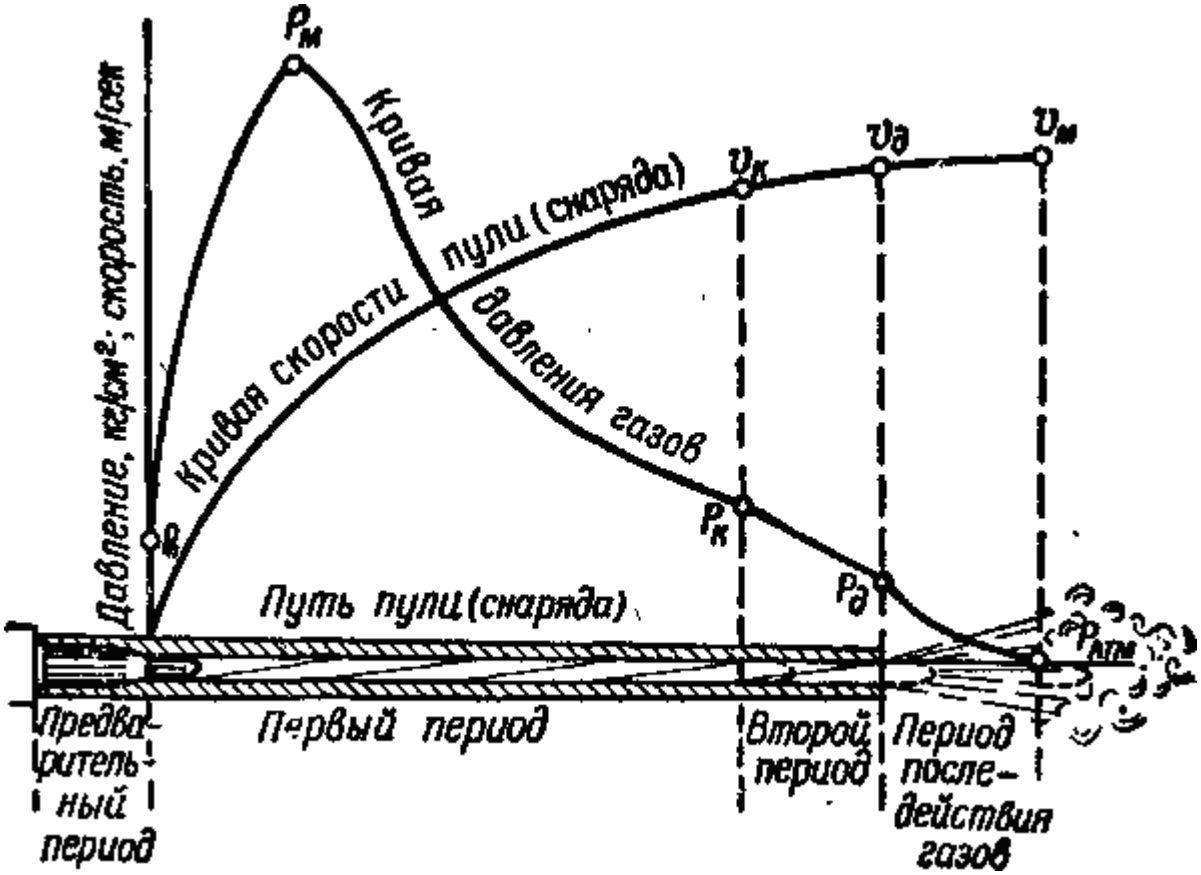


Рисунок 1 – Визуальное представление выстрела

«Как продемонстрировано на рисунке 1, наибольшее давление пороховых газов внутри ствола наступает в первом периоде, а предельная скорость метаемого элемента достигается в тот момент, когда давление опускается до минимальных значений.

Для того, чтобы смоделировать эстафетную схему, сущность которой содержится в том, чтобы разделить метательный заряд на две или более

составляющих, при использовании модели гетерогенных сред, мы можем применить расчетную схему, изображённую на рисунке 2.»[15]

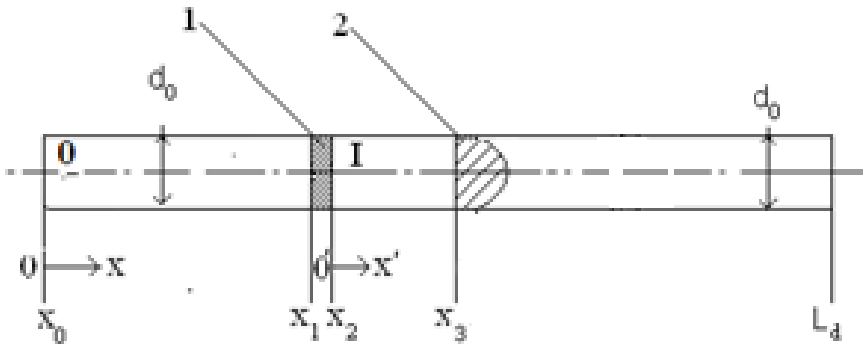


Рисунок 2 – Расчетная схема эстафетного выстрела

Здесь можно увидеть иллюстрацию схемы бинар. При её использовании, метающий пороховой заряд разбит диафрагмой две области. Первая часть – 0, обладающая длинной x1-x0; вторая часть - I, соответственно, обладающая некой длинной x3-x2; 1 – это диафрагма, которая имеет собственную толщину x2 – x1; 2 – сам снаряд (или метаемый элемент); Здесь же, d0 – некий диаметр самого трубчато-образного канала исследуемой системы; А так же, Ld – обозначение длинны имеющегося ствола с учётом имеющейся длины, соответственно, патронника.

«Таким образом, для так называемых гетерогенных сред, стандартным и обобщённым случаем называется некое перемещение, например, горящих и, собственно, в общем (имеющимся) случае, еще не загоревшихся пороховых компонентов по каналу ствола в момент выстрела. В этом вопросе разбиралось множество авторов в своих трудах, например, В. Николаевский, Р. Нигматулин, А. Шрайбер,М. Гольдштик , С.Соу, Л. Стернин, А. 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 алгоритмов 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;»[15]

«В том числе, необходимо отметить, что многие ученые исследовали теорию движений гетерогенных нереагирующих сред. Примерами авторов работ, где, по-видимому, впервые были учтены фазовые переходы являются В.Н. Вилюнов, показавший миру свои труды в 1964 году и Р. И. Нигматулин, с работой 1967 года. К. П. Станюкович, Я. А. Каневский и Дж. Корнер уже в начале 50-х годов 20-го века принимали попытки учета двухфазного характера течения.

В 1969 году была опубликована работа В. Н. Вилюнова, содержание которой доказывает увеличение скорости метания при использовании вместо классической схемы выстрела (КСВ), схему типа «а порох – поршень – порох» (ППП), которая является одним из первых прототипов создания эстафетной е схемы метания.» [5]

«Если верить историческим фактам, можно сказать, что развитие исследований в модернизации внутренней баллистики с 1950-х годов привело к эстафетной схеме, использующейся в данной работе.

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

- Момент времени зажигания основной части метательного заряда считается начальным параметром для всех динамических процессов, сопровождающих выстрел эстафетной схемы;
- Движение диафрагмы, метательного элемента и частиц дополнительного заряда начинается в момент достижения давления форсированной сборки;
- Игнорируем силу трения в стволе и сопротивление воздуха;
- Геометрический закон горения справедлив для всех пороховых частиц частей метаемого заряда;
- Дополнительная часть метательного заряда до момента разделения неподвижна, относительно движущейся сборки;
- Пренебрегаем 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 0, движущейся с начальным ускорением. В области I, движущейся с начальный ускорением, уравнения модели гетерогенных сред необходимо записывать в неинерциальной системе координат .»[8]

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, уравнений численных схемы методов и других исследования разделов.»[6]

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

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

«Как можно увидеть из системы дифференциальных уравнений, при значении N = 0, иные уравнение описывает процессы в несколько области 0, при N = 1,остояние расчеты образом производятся для области I. Решение необходимо входные уравнений после воспламенения уровня присоединенного течении метательного заряда гетерогенных будет последовательно возможных проводиться уравнений для пороха областей 0 и I.»[6]

Также, для проведения расчетов используются соотношения 2.

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

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;

τ_{TP} – 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;

$\mu -$ вязкость газа.

Таким образом, все начальные условия, существующие для нулевой области, описываются соответственно, следующим математическим закономерностям, видно, как это показано в приведённой ниже 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 существующей области 1 применима математическая 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 также мы можем выбрать граничные математические условия, приведённые в системе 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 $x_D, x'_S -$ соответственно координаты для положения так называемой диафрагмы и метаемого элемента (пули или снаряда) соответственно;

$u_D, u'_S -$ В данной системе, скорость диафрагмы и имеющегося метаемого элемента соответственно.

«Для того, чтобы мы могли определить значения этих переменных, мы будем применять метод интегрирования уравнения движения сборки, а в тот момент, когда в нашей системе произойдет момент разделения, мы просто интегрируем полученные уравнения, описывающие движение толкающего поршня, и, конечно же, метаемого пороховым зарядом элемента, пользуясь при этом, соответственно физическим законам, системам математических уравнений 6 и 7:» [18]

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

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

Таким непростым образом, совсем легко можно будет сделать логичный вывод, говорящий о том, что разделение форсированной сборки будет происходить только при выполнении следующего условия:

$$p_3 > \frac{q_s}{q_D}(p_1 - p_2).$$

«Поэтому, для последующего решения поставленных перед нами математических и физических задач с имеющимися граничными, а также, начальными общими условиями, перед проведением расчётов, необходимо в первую очередь выбрать один из существующих математических методов численного решения. При этом если допустить выбор определяющего решение фактора, говорящий о том что внутренние потоки физического импульса, а так же энергии и соответственно массы имеющейся системы должны будут определяться исследователем из физического решения сложной задачи распада внутри произвольного сильного разрыва присущий для такой среды, о которой

можно сказать, что в ней нет «собственного» внутреннего давления, а также принимая во внимание из имеющегося решения цельного распада произвольного разрыва всех параметров порохового газа определённого на замеченном скачке в секторе площади сечения, получается что для нас наилучшим в общем смысле образом будет подходить известный для математиков численный метод учёного С. К.Годунова. Таким образом принимая во внимание все аспекты данного метода, всевозможными имеющимися сетками с имеющимися шагами h и h' - определённо покрываются наши расчетные исследуемые области 0 и I.»[6]

«Стоит отметить что при таком последовательном способе решения математической системы 1 также предполагается, что будет возгораться в том числе и часть лежащего внутри метаемого элемента в нулевой области (Обозначенной цифрой 0), это говорит нам о том что сборка которую мы исследуем движется во время моделирования как скреплённое воедино целое. Теперь предстоит следующий этап – воспламенение внутренней дополнительно добавленной части метаемого элемента (пули или снаряда) в обозначенной цифрой I области. Мы будем считать что для всех математических расчетов будет взято за константу масса которая будет иметь значение $Q = 0{,}5$, другими словами 0,5г. Очевидно что при таком количестве заряда, существенно понизится давление, которое создают пороховой заряд в патроннике, а общее количество метательного заряда может быть повышено без выхода за пределы максимального давления учтённого для данной системы (Орудия).

Обратим внимание на изображение 3, на котором можно заметить доп. часть с метаемым элементом, и диафрагма отделятся друг от друга примерно через 0.5 мс после начала эксперимента.

Исследования показали, что в приведённое выше время маленькая диафрагма будет замедлена с потерей скорости от 270 метров в секунду до 125 метров в секунду (внутри области 1), после чего давление возрастёт со стороны другой области, что приведёт к резкому скачку скорости до 320 метров в секунду, но со временем и она упадёт до 230 метров в секунду. Поэтому из-за

того, что давление, идущее вслед метаемому элементу, упадёт, то диафрагмовая скорость движения снова возрастёт, после чего возвратится волна уплотнения, отскочившая от дна канала, она ускорится до момента, в котором почти догонит метаемый элемент. И тогда, в момент времени 0.74 мс – 0.76мс снаряд приобретет скорость до 580 м.с

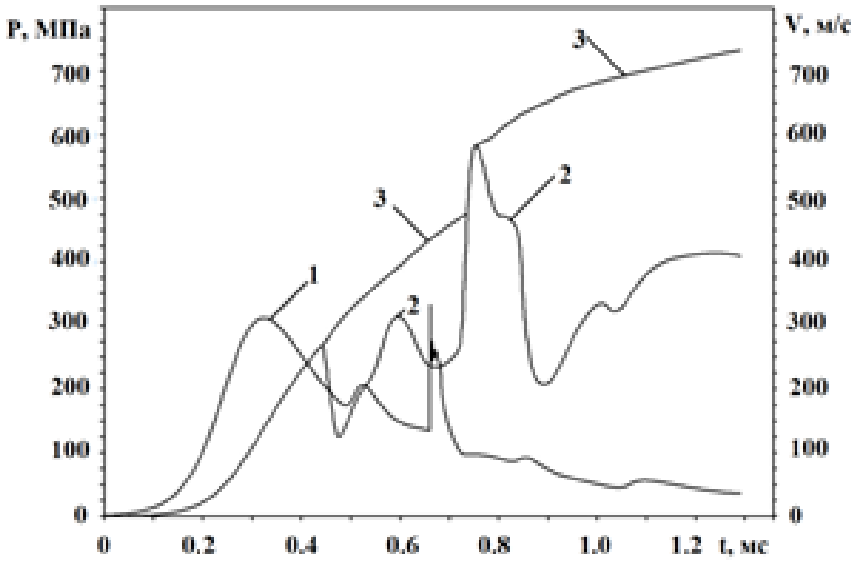


Рисунок 3 –Расчет выстрела при эстафетной схеме

Можно заметить, что на представленном изображении, ближе к концу скорость падает, при этом скорость снаряда растёт постепенно до 740 м.с на дульном срезе карабина. Когда же элемент метания вылетает из орудия, диафрагма к этому моменту передвигается примерно 400 метров в секунду.»[6]

Вывод по разделу 1

В данном разделе были изучены физические основы процесса метания. Была описана основная задача внутренней баллистики и рассмотрен метод произвольного разрыва. Разобраны используемые основные законы и уравнения, были описана принципы работы классической и эстафетной схемы метания

Изученная информации будет взята за основу для реализации последовательного и параллельного алгоритмов.

2 Разработка алгоритмов

2.1 Разработка последовательного алгоритма

Теперь, когда мы провели необходимые изучения теории, мы можем приступить к реализации блок-схемы алгоритма, работающего последовательно. Он поможет нам узнать время, которое тратится на то, чтобы снаряд покинул ствол орудия, скорость, которую он развивает, давление, оказываемое в областях 0 и I, и что немало важно, графическое положение в сетке координат 0x

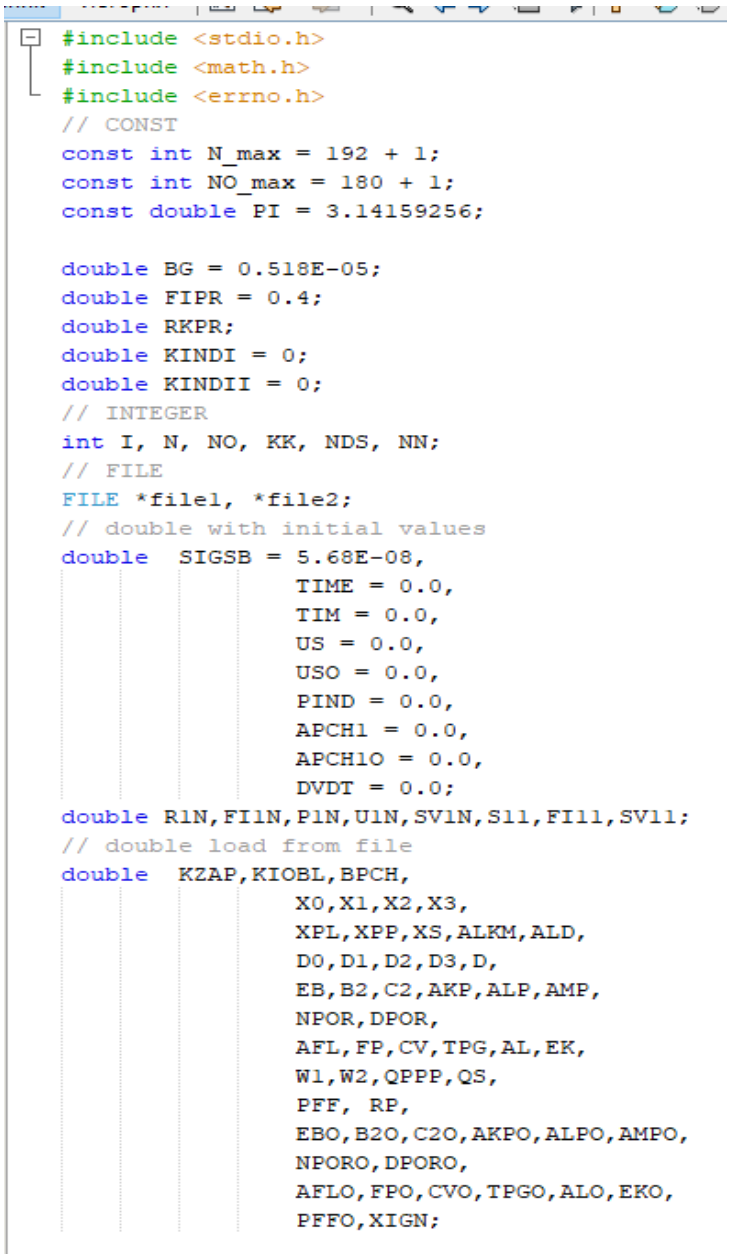
Реализованная в рамках бакалаврской работы схема представлена на изображении 4.

Рисунок 4 – Алгоритм эстафетной схемы

2.2 Написание разработанного алгоритма

Теперь перейдём к построению алгоритма на языке C++.

На скане экрана 5 и 6 мы объявили константы и начальные условия



```
#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 – Объявление констант и других 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 – Объявление констант и других переменных

Мы будем использовать идентификаторы, которые нужно объявить заранее, что и было выполнено на изображении 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**

По изображению выше легко заметить: объявленные переменные — это неотъемлемая часть функции void. Void – это спецификатор, который нужен чтобы функция, которую мы вызываем, не возвращала значения. [2].

Далее выполняются математические расчёты, такие как: произвольный разрыв, счёт шага смеси, вывод информации и тд.

Теперь, когда мы определили функции и параметры, нужно разработать главный двигатель – тело программы. Оно будет открывать файлы с необходимыми исходными данными, а также записывать выходные.

Чтобы сохранить исходные данные без изменений, важно применить функцию «rb+» (read – чтение)[4].

Также, немаловажна работа со значениями, которые мы получаем на выходе. Для них назначим функцию «wb+»(write - запись).

Чтобы перейти к математическим исчислениям, нам нужно организовать вывод всех исходных данных из файла так, чтобы было удобно с ними работать в дальнейшем. Пример на рисунке 8.

```
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 – Открываем и выводим из входного файла данные

Теперь, программа начинает свою основную работу, а после, начинается записывает полученные значения в файл, как на изображении 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

```
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

После того, как были закончены все основные математические расчеты, нужно провести масштабирование задачи, что и было сделано на рисунке 11 и 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 того, чтобы определить сколько длилось выполнение программы, необходимо использовать функцию clock(), как показано на изображении 14

```
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 Технологии и методы распараллеливания

Процесс выполнения программы, описанной выше приходится очень трудоёмким для математических вычислений, и не менее сложным для его работы внутри центрального процессора.

Поэтому, было принято решение улучшить его работу с помощью технологий распараллеливания – OpenMP (Open Multi – Processing) и MPI (Message passing Interface).

«Если рассматривать систему с разделённой памятью, то реализацию параллельного алгоритма требует физического распределения фрагментов сети между вычислителями с последующей организацией обменов между узлами в соответствии с так называемым расписанием. Программную реализацию такого алгоритма удобнее всего будет выполнять с использованием технологии MPI.

Главной проблемой при реализации предложенного алгоритма является необходимость передачи списков смежности между узлами вычислительной техники. Функции MPI предназначены для передачи данных, расположенных в памяти последовательно. Эффективная реализация списков смежности подразумевает независимый динамический массив для каждой вершины отдельно. По этой причине для передачи списков смежности разумно использовать упаковку и распаковку динамических данных при помощи специальных функций.

Технология MPI позволяет эффективно работать как с разделенной, так и с общей памятью. В последнем случае скорость обменов данными кардинально возрастёт, что что необходимо учесть.»[14]

По этой причине, в рамках данной бакалаврской работы было принято решение ограничиться системой с общей памятью, которая не требует разделения данных между вычислителями, и вся работа проделывается одним из них.

Чтобы разработать данную программу, стало необходимо применить технологию OpenMP, которая является фундаментом распараллеливания разработанных последовательных алгоритмов на языке C++ и других. Она предназначена как раз для систем с общей памятью, и она является достаточно простой в использовании.

В данной бакалаврской работе использовалась версия OpenMP 4.0.

Технология предполагает собой создание потока master как основной, и потоков slave как подчинённых, и деление нагрузки между ними, что вследствие и приведёт к увеличению скорости работы программы.

«- многозадачный режим (режим разделения времени), при котором для выполнения нескольких процессов используется единственный процессор. Этот режим является псевдопараллельным: активным (исполняемым) может быть один единственный процесс, а все остальные процессы находятся в состоянии ожидания своей очереди. Применение режима разделения времени может повысить эффективность организации вычислений (например, если один из

процессов не может выполняться из-за ожидания вводимых данных, то процессор будет задействован для выполнения другого, готового к исполнению процесса). Кроме того, в данном режиме проявляются многие эффекты параллельных вычислений (необходимость взаимоисключения и синхронизации процессов и др.), и как результат этот режим может быть использован при начальной подготовке параллельных программ;

- параллельные вычисления, когда в один и тот же момент может выполняться несколько команд обработки данных. Такой режим вычислений может быть обеспечен не только при использовании нескольких процессоров, но и при помощи конвейерных и векторных обрабатывающих устройств;

- распределенные вычисления — термин, который обычно применяют для указания параллельной обработки данных, 9 когда используется несколько обрабатывающих устройств, достаточно удаленных друг от друга, в которых передача данных по линиям связи приводит к существенным временным задержкам. Как результат эффективная обработка данных при таком способе организации вычислений возможна только для параллельных алгоритмов с низкой интенсивностью потоков межпроцессорных передач данных. Перечисленные условия являются характерными, например, при организации вычислений в многомашинных вычислительных комплексах, образуемых объединением нескольких отдельных ЭВМ с помощью каналов связи локальных или глобальных информационных сетей.»[10]

При распараллеливании алгоритма обычно выбирают такие места программы, где проходят труднейшие математические вычисления, для того чтобы максимально качественно пользоваться ресурсом центрального процессора. Поэтому, в качестве объектов распараллеливания были выбраны такие фрагменты кода как исчисление параметров во внутренних ячейках, вывод информации и др.

2.4 Разработка параллельного алгоритма после последовательного

Чтобы приступить к разработке параллельного алгоритма, создадим новый проект, который будет иметь те же самые исходные данные и всё ту же конструкцию программного кода.

Для того, чтобы применить методы OpenMP, придётся сначала внести дополнительную настройку нового проекта (-fopenmp), как проиллюстрировано на изображении 16.

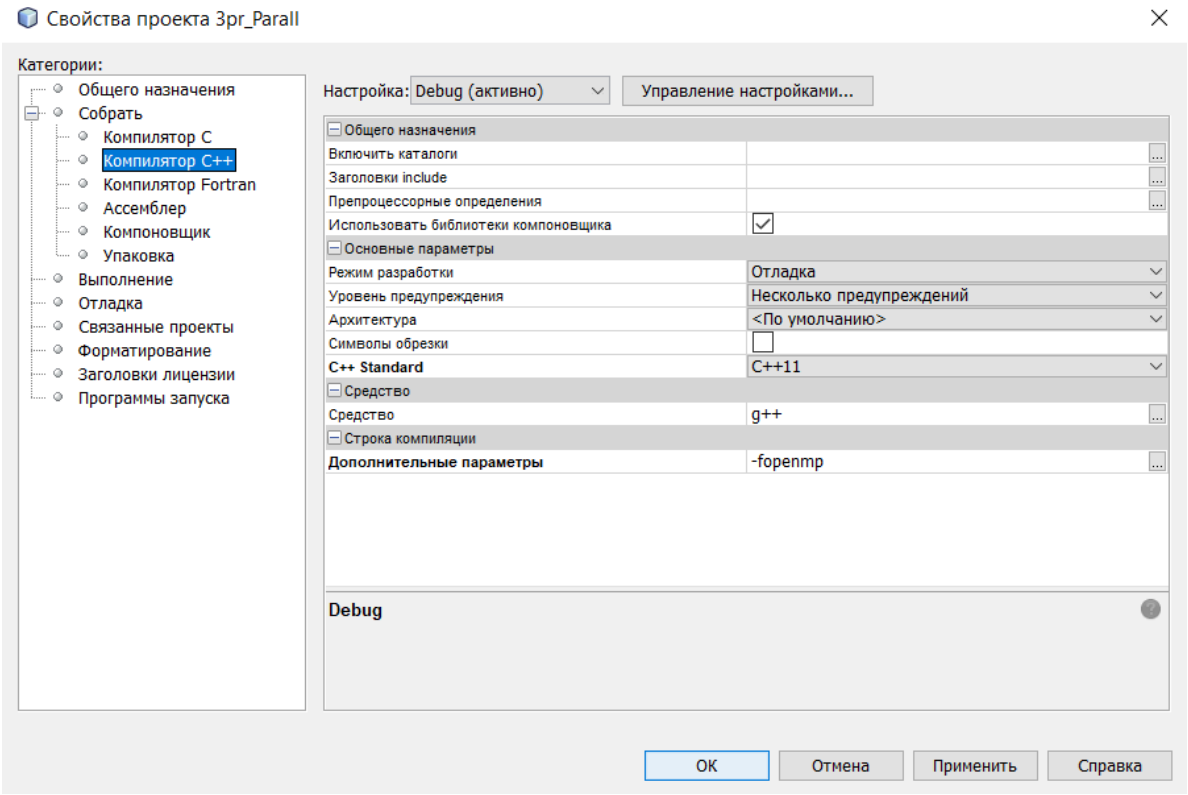


Рисунок 16 –Настройка будущего проекта

Применив новые свойства, активируем необходимые библиотеки.

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

Рисунок 17 –Подключение библиотеки OpenMP

При выполнении данной бакалаврской работы было принято решение применять распараллеливание на уровне задач, т.к. часто происходит так, что последовательные алгоритмы, которые модернизировали в этом направлении на уровне задач, являлись самым простым и при этом самым эффективным методом. Такое решение возможно лишь в том случае, когда программа состоит из подпрограмм, независящих друг от друга.

Чтобы продолжить исследование параллельных алгоритмов, необходимо проанализировать фрагменты кода и определить, какие места лучше всего подойдут.

Первичный осмотр может показать, что программный код невозможно сделать многопоточным, из-за множества операторов “Goto” (Переход к заданной точке в коде), а каждый такой переход было бы необходимо контролировать, что только замедлило работу программы.

Поэтому, логично предположить, что решением окажется распараллеливание сложных и длинных циклов внутри каждой подпрограммы.

В приведённом ниже фрагменте кода лучше всего подходит разделение на потоки цикла for, ведь это даст процессору возможность рассчитывать каждую итерацию в отдельном потоке.

```
//-------------------------------------------------------------------------------
// ПОДПРОГРАММА СЧЕТА ШАГА ТЕЧЕНИЯ СМЕСИ
//-------------------------------------------------------------------------------
void STEP(double R[], double P[], double U[], double W[],
    double FI[], double Z[], double S[], double SV[],
    double AL, double EK, double H, double AKP, double ALP, double AMP,
    double EB, double AFL, double W0, double S02, double QP, double& XS,
    double US, double& USN, double DVDT, double APCH1, int& N, double KMOV,
    double KIOBL, double P2, double AMSB)
{
    //omp_set_num_threads(8);
    G1 = EK + 1.0;
    G2 = EK - 1.0;
    G3 = EK + 1.0;
    G4 = EK - 1.0;
    G5 = G2 / G1;
    G6 = G4 / G3;

    RKP = RP / RM * (1.0 - FI[1]);
    RPRK(RKP, -W[1], RKP, W[1], RRK1, WW1);
    RPRS(1.0, EK, U[1], P[1], R[1], AL, EK, -U[1], P[1], R[1], AL, UBB, PBB, RBB, UMM, PMM, RMM, EKB, ALB);

    // РАСЧЕТ ПОТОКОВ

    ARK1 = SV[1] * RRK1 * WW1;
    AWRK1 = SV[1] * RRK1 * WW1 * WW1;
    AR1 = S[1] * UMM * RMM * FI[1]; S11 = S[1];FI11 = FI[1];SV11 = SV[1];
    AUR1 = S[1] * FI[1] * (RMM * UMM * UMM + PMM);
    AER1 = S[1] * FI[1] * UMM * (RMM * PMM * (1.0 / RMM - AL) / (EK - 1.0) + RMM * UMM * UMM / 2.0 + PMM);
    ZN = Z[1] - TH / 2.0 / H * (W[1] - fabs(W[1])) * (Z[2] - Z[1]) + AFL * P[1] * PM / EB * TH * BPM;

    if (ZN > 0.985) ZN = 1.0;
    PST = P[1];
    WST = W[1];
    FIST = FI[1];
    #pragma omp parallel
    {
        #pragma omp parallel for schedule(static)
        // РАСЧЕТ ПАРАМЕТРОВ ВО ВНУТРЕННИХ ЯЧЕЙКАХ

        for (I = 2; I <= N; I++)
```

Рисунок 18 –Пример распараллеливания участков программы

Применять функции OpenMP можно как для целых функций, как это реализовано в данной работе на примере расчёта параметров, так и для каких-либо её частей.

```
#pragma omp parallel
{
    #pragma omp parallel for schedule(static)
    for (I = 1; I <= KK; I++)
    {
        EKG2 = EKG2 + pow(UO[I] * 10, 2) / 2.0 * RO[I] * SO[I] * SM * FIO[I] * HO * ALKM;
        EBG2 = EBG2 + 100000 * PO[I] * (1.0 / RO[I] - ALO / RM) / (EKO - 1.0) * RO[I] * SO[I] * SM * FIO[I] * HO * ALKM;
        EKK2 = EKK2 + (1.0 - FIO[I]) * RP * SO[I] * pow(WO[I] * 10.0, 2) / 2.0 * SM * HO * ALKM;
        EXK2 = EXK2 + (1.0 - FIO[I]) * RP * SO[I] * SM * HO * ALKM * QPPO;
    }
}
```

Рисунок 19 –Распараллеливание цикла внутри функции

```
#pragma omp parallel for reduction(+:AMS,AES)
for (I = 1; I <= KK; I++)
{
    AMS = AMS + R[I] * S[I] * SM * FI[I] * H * ALKM + (1
    AES = AES + (pow(U[I] * 10, 2) / 2.0 + 100000 * P[I]
}
```

Рисунок 20 –Распараллеливания второго цикла функции

```
#pragma omp parallel
{
    #pragma omp parallel for schedule(static)
    for (I = 1; I <= KK; I++)
    {
        EKG1 = EKG1 + pow(U[I] * 10, 2) / 2.0 * R[I] * S[I] * SM * FI[I] * H * ALKM;
        EBG1 = EBG1 + 100000 * P[I] * (1.0 / R[I] - AL / RM) / (EK - 1.0) * R[I] * S[I] * SM * FI[I] * H * ALKM
        EKK1 = EKK1 + (1.0 - FI[I]) * RP * S[I] * pow(W[I] * 10.0, 2) / 2.0 * SM * H * ALKM;
        EXK1 = EXK1 + (1.0 - FI[I]) * RP * S[I] * SM * H * ALKM * QPP;
    }
}
```

Рисунок 21 –Распараллеливание N-ого цикла функции

Вывод по разделу 2

В данном разделе был реализован последовательный алгоритм, а также исследованы принципы распараллеливания, используя простую технологию OpenMP, и разработана распараллеленная версия.

3 Сравнительный анализ алгоритмов

3.1 Тестирование реализованных алгоритмов

Заключительная часть данной бакалаврской работы заключается в том, что теперь нам предстоит определить, насколько параллельный алгоритм работает лучше последовательного. Тесты проведены на компьютере с процессором Intel®Core™ i5-8300H, с частотой 2304 МГц, и четырьмя ядрами. Оперативная память в объёме 8 гигабайт. На изображении 22 также предоставлены все сведения о системе. Приведённая выше информация нужна для того, чтобы при разнице в производительности, учитывалась физическая составляющая компьютеров.

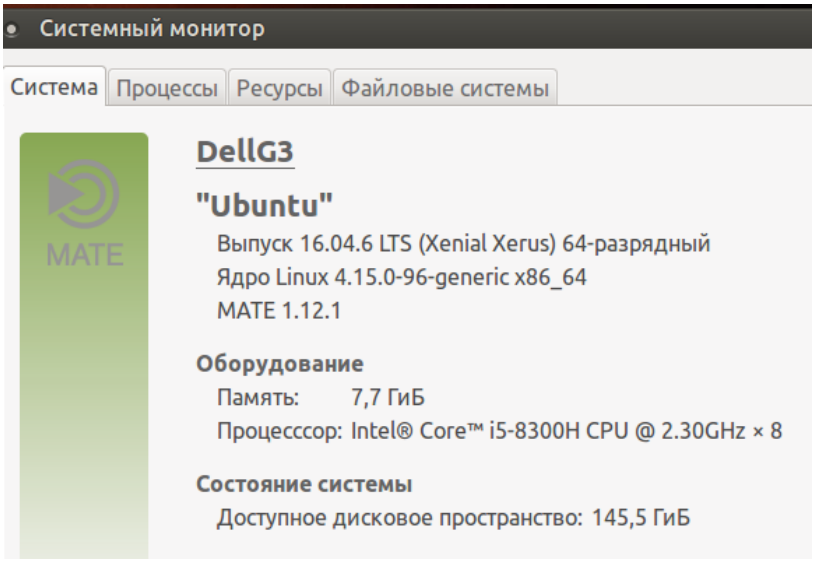


Рисунок 22 –Операционная система ПК

Что касается правильности подсчётов, оба алгоритма выдают одинаковые значения при одинаковых входных параметрах, что доказывают нам изображения 23 – 26.

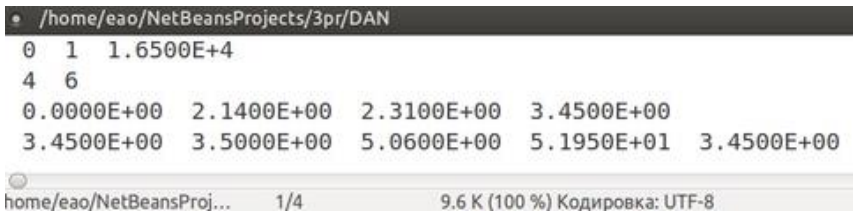


Рисунок 23 –Входные данные для последовательного алгоритма

Рисунок 24 –Входные данные для распараллеленного алгоритма

Рисунок 25 – Полученные данные при последовательном алгоритме

Рисунок 26 – Полученные данные при распараллеленном алгоритме

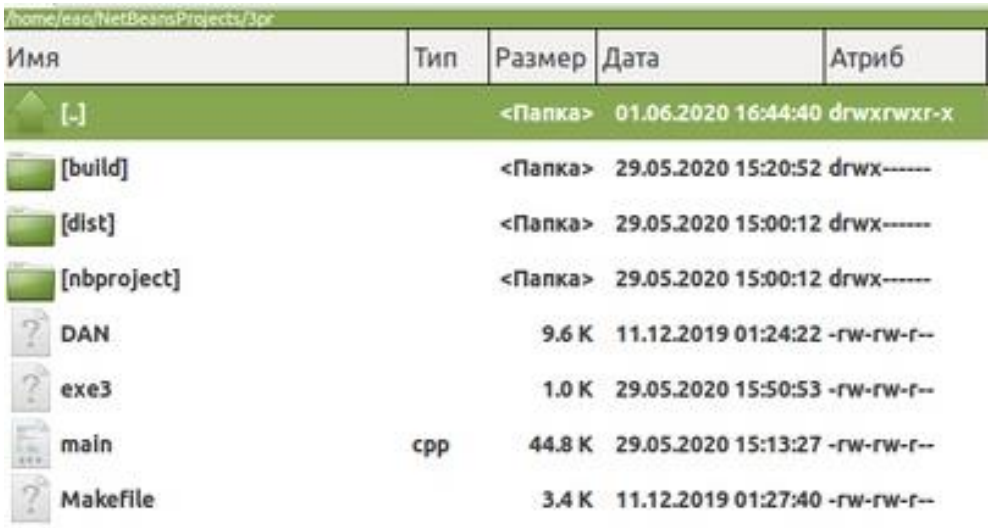


Рисунок 27 – Проект для тестирования последовательного алгоритма

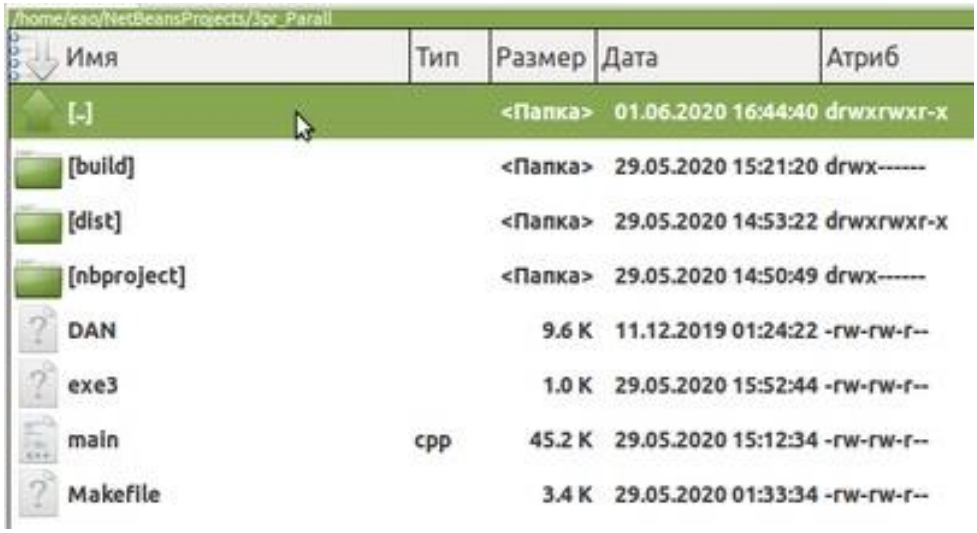


Рисунок 28 – Проект для тестирования распараллеленного алгоритма

3.2 Анализ скорости выполнения программ

Теперь предстоит разобраться, какой алгоритм окажется эффективнее. Учитывая изначальные задачи, важным показателем эффективности очевидно стало время, которое программа затрачивает на проведение расчётов по эстафетной схеме. Также, ввиду того, что скорость выполнения алгоритмов

зависит от того как загружена операционная система внешними процессами, стало необходимым проведение нескольких тестов с сохранением времени, затраченным на каждый алгоритм. Все полученные данные предоставлены в таблице 1.

Таблица 1 – Данные по времени выполнения программ

Алгоритм подвергающийся тестированию	Время выполнения при 1 запуске алгоритма (мс)	Время выполнения при 2 запуске алгоритма (мс)	Время выполнения при 3 запуске алгоритма (мс)
Последовательный	495	414	450
Многопоточный	248	200	200

Исходя из данной таблицы получаем диаграмму, которая демонстрирует нам проведённый опыт.

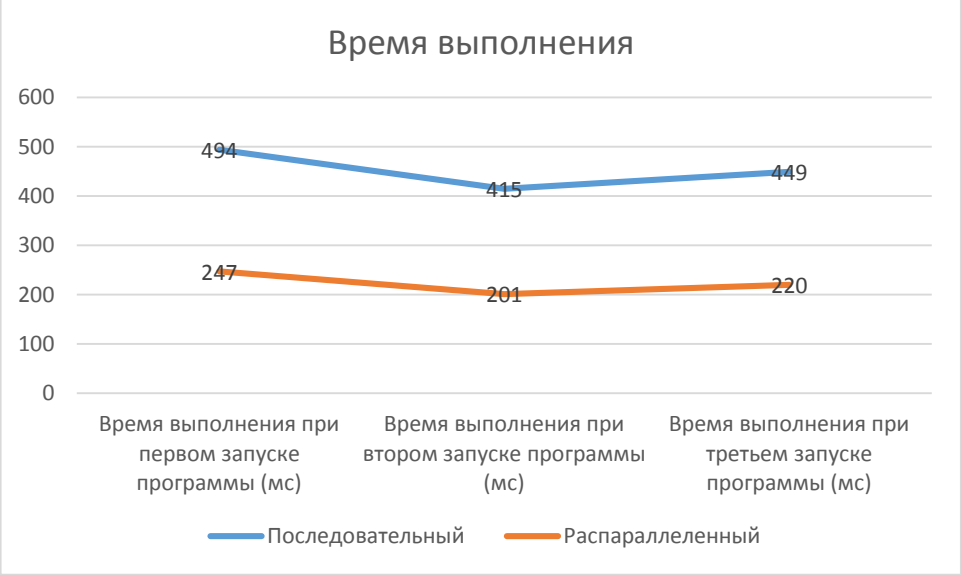


Рисунок 29 –Диаграмма значений затраченного времени

Можно заметить, что работа обеих программ равноценно зависит от остальных внешних процессов. Также видна линейная зависимость, поэтому, для установления зависимости в скорости работы двух алгоритмов, применим формулу 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.

Так, мы получим расчёты, где $k_1 = \frac{494}{247} = 2$, $k_2 = \frac{415}{201} = 2{,}06$, $k_3 = \frac{449}{220} = 2{,}04$. Определим среднее арифметическое по формуле 9.

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

Среднее значение по данной формуле равно 2,03 – коэффициент эффективности.

3.3 Анализ нагрузки системы при выполнении программ

Чтобы определить полный спектр эффективности программ, во время выполнения текущей бакалаврской работы, было принято решение провести нагрузочное тестирование.

Один из тестов проводился, используя системный монитор, являющийся инструментом в ОС Ubuntu MATE. Так можно посмотреть нагрузку от 0 до 100%. Задаётся активный процесс на все 8 процессоров, помимо этого единовременно используется сеть и память. Все изменения отслеживаются в течении 60 секунд.

Изображение 30 иллюстрирует состояние системы до начала опытов.

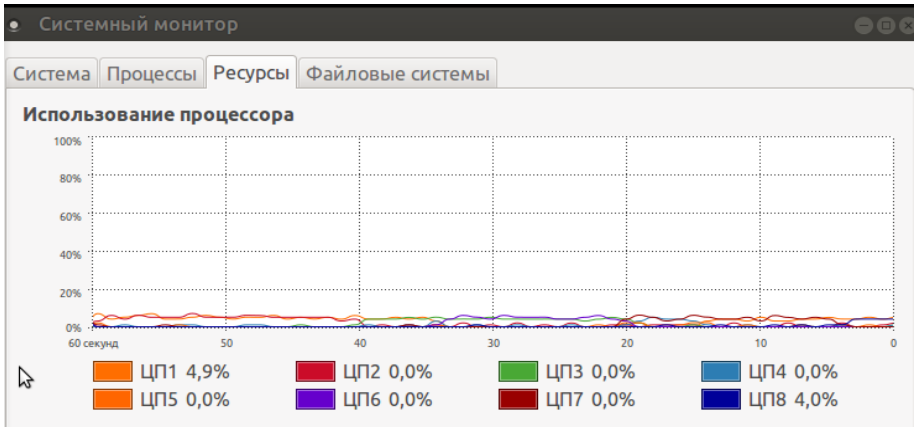


Рисунок 30 – Системный монитор без запуска программ

Теперь попробуем завершить все запущенные приложения на ПК и запускаем только реализованный алгоритм.

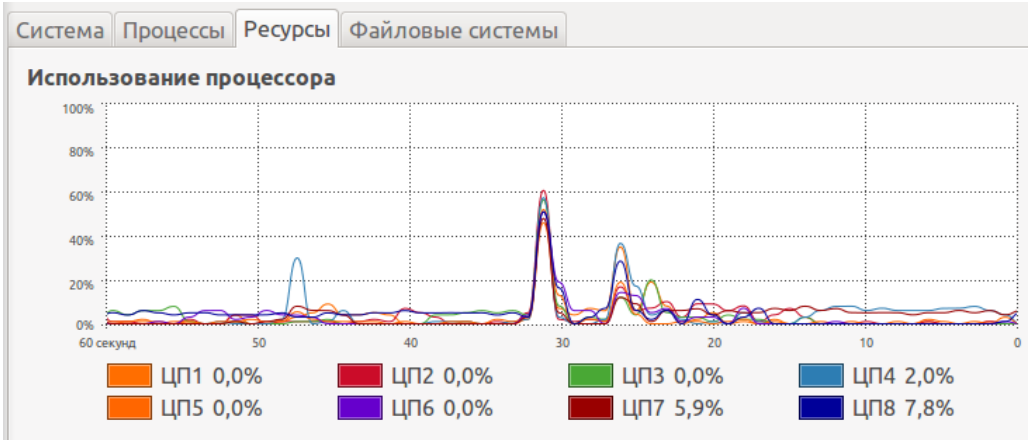


Рисунок 31 – Системный монитор при выполнении последовательного алгоритма

При загруженном процессоре становится заметна разница. В момент первого скачка изменений все 8 ЦП достигают загруженности в 60%. При последующих скачках показатели ниже

Судя по этому тесту можно сказать, что разница между системой в покое и в нагрузке – 60%

Теперь протестируем распараллеленный алгоритм.

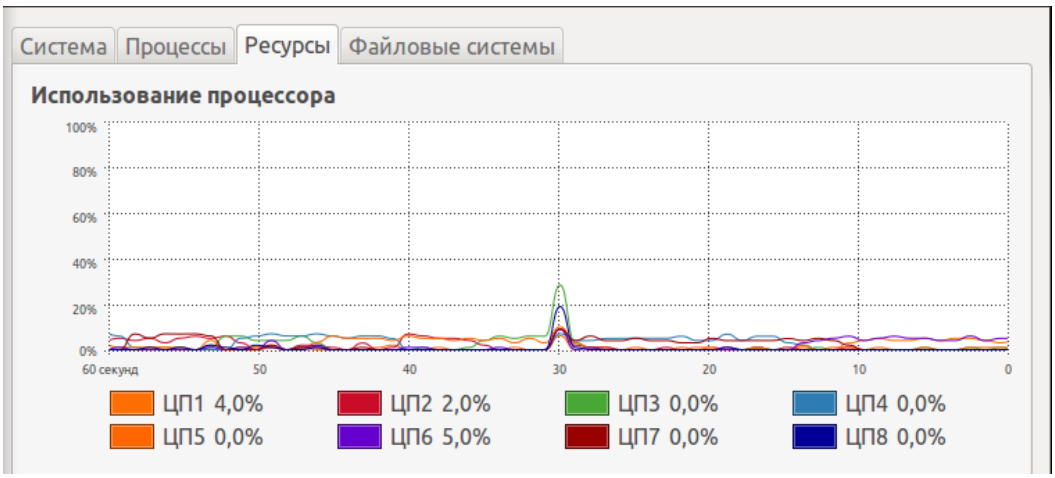


Рисунок 32 - Системный монитор при выполнении распараллеленного алгоритма

Анализируя данные последнего теста, нагрузка на систему возникает только в один момент времени, чего не скажешь про последовательный алгоритм. Средняя нагрузка составляла 30%, это говорит о том, что алгоритм, с использованием OpenMP нагружает систему в 2 раза меньше.

Чтобы провести последний сравнительный тест, в качестве помощника в этом хорошо подошёл инструмент под названием Nmon – разработанный специально под системы AIX и Linux. Эта утилита отображает всю информацию о CPU и не только. Здесь, как показано на рисунке 33, опять же, система тестировалась без запущенных программ.

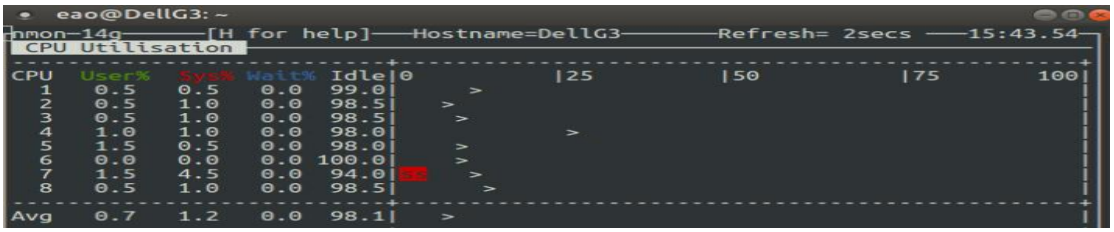


Рисунок 33 –Состояние CPU без запущенных программ

На следующей иллюстрации, как и в прошлых тестах, показывается тестирование системы выполнением программы.

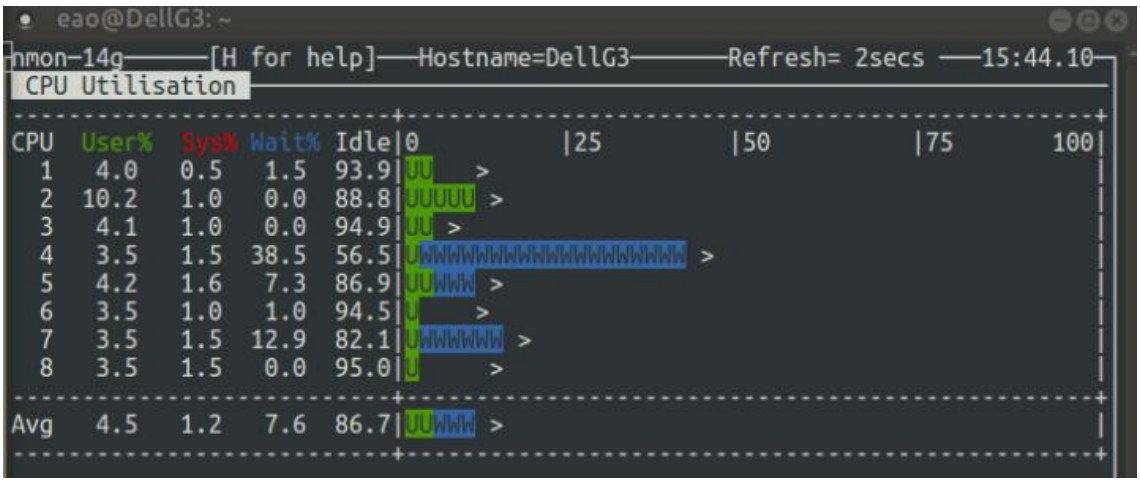


Рисунок 34 –Состояние CPU с запущенным последовательным алгоритмом

Сразу отчётливо видна разница в загруженности центрального процессора, судя по рисункам 33 и 34, а именно, компьютерные процессоры хоть и были полностью заняты, но нагрузка на них была распределена не равнозначно. Теперь очередь проверить работу параллельного алгоритма.

Рисунок 35 - Состояние CPU с запущенным распараллеленным алгоритмом

Здесь сразу можно отметить, что загруженность равномерно распределилась по всем процессорам.

3.4 Анализ ускорения выполнения программ по Закону Амдала

Благодаря работам Джина Амдала, можно использовать его закон, который он сформулировал ещё в 1967 году, описывающий зависимость границ роста производительности во время построения многопоточного алгоритма.

Зная количество процессоров компьютера и количество кода, поддающегося распараллеливанию, можно вычислить ускорительный коэффициент согласно 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.

Проводя анализ данной формулы, легко предположить, что для маленького количества кода и большем количестве процессоров этот параметр будет значительно расти.

Приведённая в пример ОС, а также использованный для данной работы ПК, на котором проводились все тесты также используются в формуле 11

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

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.

«Закон Амдала показывает, что прирост эффективности вычислений зависит от алгоритма задачи и ограничен сверху для любой задачи. Не для

всякой задачи имеет смысл наращивание числа процессоров в вычислительной системе.

Более того, если учесть время, необходимое для передачи данных между узлами вычислительной системы, то зависимость времени вычислений от числа узлов будет иметь минимум. Это накладывает ограничение на масштабируемость вычислительной системы, то есть означает, что с определенного момента добавление новых узлов в систему будет увеличивать время расчёта задачи.»[20]-[24]

Получив эти данные, и сравнив скорость выполнения алгоритмов, можно сделать вывод, что параллельный алгоритм выполняет свою работу быстрее в 2 раза. Во время анализа нагрузки на центральный процессор, выяснилось, что многопоточный алгоритм работает в 1,5 раза лучше. Все полученные значения эффективности удовлетворяют требованиям закона Амдала.

Сокращать время выполнения параллельного алгоритма можно за счёт уменьшения доли последовательного кода, или же путём использования более мощных систем, с большим количеством процессоров.

Разберём теоретически возможные коэффициенты ускорения, для этого стоит взять части последовательного алгоритма равные 50%,40%...0% и количество возможных процессоров: четыре, восемь, десять, сто и тысячу. Результаты по формуле 10 отобразим в таблицу 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.»[11]-[12]

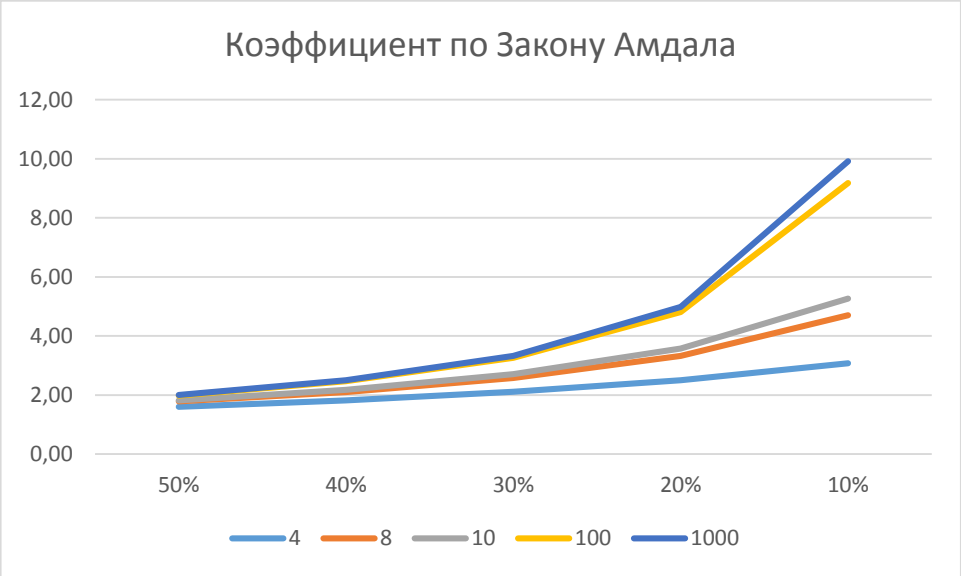


Рисунок 36 – Диаграмма полученных данных

Рисунок 36 говорит нам о том, что увеличивать количество процессоров будет иметь смысл при увеличении количества параллельного кода и соответственно уменьшении последовательного.

Это говорит о том, что не каждая задача решается путём увеличения количества потоков. Чтобы повысить работоспособность алгоритма, может быть достаточно сократить процент последовательного алгоритма. Исходя из таблицы 2 заметим, что если в алгоритме останется 10% последовательного кода, то даже 4х процессорная система будет работать лучше, чем 1000 процессорная.

Вывод по разделу 3

В данном разделе был проведён сравнительный анализ работы двух алгоритмов: последовательного и параллельного. А также, представлены итоги проведенной исследовательской работы.

Заключение

В ходе выполнения бакалаврской работы на тему «Исследование параллельных алгоритмов для обработки данных в сети» довелось исследовать теорию основ существующей модели «Эстафетная схема», которая освещает процесс и взаимосвязь физических явлений во время метания пули из винтовки.

Так же, в первой главе данной работы, довелось ознакомиться с материалами, которые тесно связаны с высшей математикой, а именно, математическими расчётами, использованными в исследуемой схеме.

Была достигнута основная цель бакалаврской работы, которая связана с сокращением времени, которое затрачивается на вычислительную работу алгоритма

Для достижения таких результатов, необходимо было рассмотреть решения проблемы, которые уже существуют, например, время, которое затрагивается на то, чтобы программа завершила свою работу.

Также для разработки распараллеленного алгоритма необходимой стала задаче разработки последовательного, что и было выполнено во второй главе, рамках данной выпускной квалификационной работы.

Чтобы спроектировать распараллеленный алгоритм, было принято решение пользоваться известной технологией OpenMP, которая, в свою очередь, не затрагивает графический процессор для своих нужд, как это происходило в других научных исследованиях, похожих на данное.

Сравнительный анализ, проведённый в третьем главе доказал перспективу использования параллельных алгоритмов для ускорения программ, т.к. приведённый в данной бакалаврской работе алгоритм был ускорен примерно в 2 раза.

Подводя итог, заметим, что в данной бакалаврской работе осветились все главные моменты математической модели метания снаряда по «Эстафетной схеме», а часть материала, была предоставлена на XLVII Самарской областной студенческой научной конференции в виде доклада.

Список используемых источников

1. Антонов А. С. Параллельное программирование с использованием технологии OpenMP: учебное пособие / А. С. Антонов. - М.: Изд-во МГУ, 2012. - 77 с.
2. Березкин Б. И., Березкин С. Б. Начальный курс С и С++. Москва «Диалог-Мифи», 2005 г.
3. 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
4. Воеводин В. В. Вычислительная математика и структура алгоритмов. - Москва: Издательство МГУ, 2006. - 112 с.
5. 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
6. 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.
7. 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
8. 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.
9. Малявко А. А. Параллельное программирование на основе технологий OpenMP, MPI, CUDA. - Новосибирск: Изд-во НГТУ, 2015. - 116 с. ISBN 978-5-7782-2614-2.

10. 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

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

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

13. 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.

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

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

16. 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.

17. 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.

18. 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.

19. 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/

20.Joe Albahari. Threading in C# [Электронный ресурс]: Режим доступа: http://www.albahari.info/threading/threading.pdf

21. Scott Meyers Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14.

22. 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.

23. Stephen S. Skiena. The Algorithm Design Manual 2nd ed. 2008 Edition

24. Wilkinson B., Allen M. Parallel programming techniques and applications using networked workstations and parallel computers. - Pearson Education, 2005. - p. 468.